SAMHAIN: Host Based Intrusion Detection via File Integrity Monitoring

Martinus Nel

SAMHAIN: Host Based Intrusion Detection via File Integrity Monitoring

GIAC (GSEC) Gold Certification

Author: Martinus Nel, martinus.nel@gmail.com
Advisor: Hamed Khiabani, Ph. D.

Accepted: April 23rd, 2014

Abstract

Monitoring changes to critical files is not only crucial, but also a requirement for some security standards, compliance regulations and laws. Organisations have a need for scalable and highly configurable solutions, so that the product can be specifically tailored for the environment and exact specifications. Implementing tools like Samhain can provide organisations with the confidence that data is not modified in any unauthorised manner and help cover any regulatory compliances. This paper will cover in detail how to set up the open source Samhain project in a client server configuration as a complete solution. Attention was given to the various capabilities of the software and what affects they have on the system.

1 Introduction

This paper will focus on the installation and configuration of Samhain in a client / server architecture with some specific compile and runtime options explored. In depth examples and explanations of each step will be given to make it clear how the various parts interact with each other and how it all fits together. Examples for the configuration of applications that is required by Samhain will only receive sufficient coverage to make things work, but additional sources of information on said software will be provided.

If one is not familiar with Host Based Intrusion Detections Systems (HIDS) such as Samhain, then SANS have a very concise introductory article on what HIDS[1] is. Once an attacker has gained access to your system, one of the objectives is to modify the system in some way to suite him. This may be installing new software or modifying existing files. System administrators should to be alerted as soon as possible of such unauthorised changes so that they can be acted upon.

Part of the overall HIDS picture is that of File Integrity Monitoring (FIM). In short, to monitor a predefined set of files for specific changes. For example, a log file should only grow in size, not shrink. The ownership and permissions of "**/etc/shadow**" should never change or the checksum of a binary should never change unless via authorised upgrade. As a starting point, an initial baseline has to be established against the known uncorrupted files. Once the baseline is set, there are configurable rules as to the definition of a change to a file. The tool then scan all the files periodically, depending on configuration, for changes and report them as required. In the case of Samhain, there are various reporting options such as log file, direct e-mail or custom script.

There are a wide range of FIM products available today, both open source[2] and proprietary[3], that provides various feature sets. Besides being a fantastic festive season, Samhain is also a fantastic FIM. It is chosen here as it has an excellent feature set whilst keeping focus on its goals and not trying to be an overblown HIDS or Security Information Event Management (SIEM) system. Both HIDS and SIEM systems are still required as part of the overall security profile of the environment, but the focus here is just that of FIM. It further helps that the initial cost of

1 http://www.sans.org/security-resources/idfaq/what_is_hips.php
2 For example http://aide.sourceforge.net/ and http://www.ossec.net/
3 For example http://www.tripwire.com/ and http://www.logrhythm.com/

procurement is zero due to the liberal license used. This is not to say that it will be a free ride implementing and managing Samhain, as there will be significant time requirements like with any other FIM or HIDS.

Besides all the standard features that are expected of a FIM (such as checksum of files, time and date stamp change, file owner ship and permission change, &c.), Samhain has a variety of features that makes it stand out from the crowd. Samhain can be used where PCI DSS (or other compliance policies) is required with regards to the monitoring of growing log files. To minimise the impact on disk IO and get immediate notifications of specific file changes, Samhain can leverage the "**inode**" notify (inotify[4]) Linux kernel subsystem. There are also various stealth options available that will be investigated to obfuscate the binary and configuration data. Further obfuscation will be done by hand to minimise the chance of the FIM being discovered by an attacker on the local system. Like most modern tools, Samhain can be also be centrally managed via a web console. The full range of features that Samhain provide is well documented and is left for an exercise to the reader. If so required, Samhain can also be further extended by coding custom modules against it. Samhain also goes a step beyond just file integrity monitoring, as it can monitor open ports or kernel integrity.

Although trust in downloading the authentic source of Samhain will be covered, trust in the OS should also be established. Reputable Linux distributions offer checksums of the images provided, and those should be verified before using the images. Specifically to CentOS, they provide MD5, SHA1 and SHA256 sums for the ISO's[5]. As Samhain is open source, the entire code base can be verified so that the highest level of trust can be instilled in the product. This will allow the user to establish a complete chain of trust from the OS image to the final build of Samhain. Trust here is essential, as unauthorised changes to files can lead to the disclosure of sensitive information or a disruption in services. Trust in the hardware is a subject for a different paper, such as Bryan Parno's research on Bootstrapping Trust in a "Trusted" Platform (Parno, 2008).

4 http://man7.org/linux/man-pages/man7/inotify.7.html
5 http://mirror.synergyworks.co.uk/centos/6.5/isos/x86_64/

2 Prerequisites

- This paper will be using CentOS 6.5 x86_64 for all examples. Any other distribution can be used, but the various commands should be modified to suit the distribution.
- A host that can run two small virtual machines (VM) and a temporary web server for the kickstart file found in Appendix A.
  - A quick solution for a web server can be running "**python -m SimpleHTTPServer**" from the folder that has both the kickstart file and OS media.
- For installation instructions via kickstart see the RedHat documentation (redhat.com, 2014).
- Both virtual servers host names must be fully qualified (FQDN) and correctly set in the host file. Both virtual servers should have each others entries also in the host file. The correct format for host entries are defined in RFC 952 (Harrenstien, Stahl & Feinler, 1985).
- MySQL and Apache is required on the server. This guide will assume a vanilla MySQL and Apache configuration (**yum -y install httpd mysql-server**). There are a multitude of resources available to further configure MySQL[6] and Apache[7]. The services should also be set to run by default (**chkconfig mysqld on; chkconfig httpd on**).
- MySQL development package is required on the server (**yum -y install mysql-devel**).
- MySQL must have a root password set (**mysqladmin -u root password NEWPASSWORD**). While your at MySQL, also look at "**/usr/bin/mysql_secure_installation**".
- Port "**50888 TCP**", or whatever port was set when building, should be open if there are any external firewalls between the server and client. Local firewall rules will be covered.
- ImageMagick is required on the client (**yum -y install ImageMagick**).
- Zlib development packages on both server and client (**yum -y install zlib-devel zlib-static**).
- The GNU multiple precision library, as it is faster than the built in MP that comes with Samhain (**yum -y install gmp-devel gmp-static**).
- Some additional C libraries required for static linking (**yum -y install glibc-static**).
- The development group must be installed on both server and client (**yum -y groupinstall "Development tools"**).

6 https://www.google.com/search?q=mysql+installation+guide
7 https://www.google.com/search?q=apache+installation+guide

None of the software prerequisites are configured in the kickstart file in Appendix A, as they are explicitly listed so that the reader can see exactly what is required on top of a vanilla build. Additional libraries or tools may be required for some of the modules / compiler options that are available in Samhain but not covered here.

Virtual machine requirements (when using a VM and the supplied kickstart file).

- Number of CPU's 1
- RAM 1 GiB
- HDD 8 GiB
- Network Bridged

3 Implementation

During implementation of Samhain in a server / client configuration, there is some jumping between work done on the server and work done on the client, or both. In cases where there is just a dollar ($) or a hash (#) at the start of the command without any host name, the command applies to both server and client. In other cases the command will be prepended with the server name in square brackets ([]) akin to the Bourne-gain shell (Bash) prompt. Also take note that the hash symbol means being "**root**" and dollar symbol means being a normal user. Some of the example commands may run over multiple lines, so take care when copying such commands. It has also been found that copying the quotation marks ("") to the terminal does not work correctly, this will likely depend the local system settings.

3.1 Download and Signature Checking

For downloading and unpacking instructions of Samhain, see Appendix B. After the package has been downloaded and unpacked, the GNU Privacy Guard (GPG) key fingerprint should to be checked to confirm that the source files are authoritative. Without checking the fingerprint, there is no way to know for sure that the tarball was downloaded from the authoritative source. This is not to guarantee that these files are uncompromised, but it is better than just relying on hashes (Ullrich, 2013). Start by requesting the public key from a key server, and then look for the fingerprint in this key. After verifying the key fingerprint with the one on this paper and with the one displayed on the Samhain download page, the second stage tarball can be extracted.

```
$ gpg --keyserver pgp.mit.edu --recv-key 0F571F6C
...
$ gpg --fingerprint 0F571F6C | grep "Key fingerprint"
      Key fingerprint = EF6C EF54 701A 0AFD B86A  F4C3 1AAD 26C8 0F57 1F6C
$ gpg --verify samhain-3.1.0.tar.gz.asc samhain-3.1.0.tar.gz
...
gpg: Good signature from "Rainer Wichmann <rwichmann@la-samhna.de>"
...
$ tar -xvzf samhain-3.1.0.tar.gz
...
```

If the signature or the key fingerprint did not match, make sure to download the tarball again. It could be something as simple as a corrupt download.

3.2 Server Setup

The server side component of Samhain is called Yule (also a festival like Samhain). After the package have been checked and extracted, make sure to log in as the "**root**" user and that the current working directory is the top level directory of the unpacked source files. Start by creating a user for the service and generating a GPG key as that user. Also, as some work will be done with GPG and the "**root**" is used, temporary access to root's tty is required. Be aware that this will very likely open up some specific attack vector, so do not leave it open.

```
[root@samserver] # cd samhain-3.1.0
[root@samserver] # adduser yule
[root@samserver] # chmod o+rw `env | grep -i tty | awk -F "=" '{print $2}'`
```

An alternative to giving temporary access to root's tty is to set a password for the user "**yule**" and then log into the system with that account. Since this exercise is done on a VM, there might be some difficulty in getting random data for the GPG key generation. Install and run HArdware Volatile Entropy Gathering and Expansion[8] (HAVAGE) to help alleviate this issue. After the HAVAGE service is started, switch over to the "**yule**" user and generate the GPG key.

```
[root@samserver] # yum -y install
http://www6.atomicorp.com/channels/atomic/centos/6/x86_64/RPMS/haveged-1.3-
2.el6.art.x86_64.rpm
[root@samserver] # service haveged start
[root@samserver] # su - yule
[yule@samserver] $ gpg --gen-key
```

8 https://www.irisa.fr/caps/projects/hipsor/

With regards to the GPG options, the defaults will suffice for this exercise. Further details can be found in the online documentation[9]. Hit "**CTRL+D**" or type "**exit**" to return to the "**root**" shell which should still be in the source directory of Samhain. The individual configure switches will not be covered here. These options are briefly described when executing "**./configure --help**" and are fully covered by the documentation (both online and packaged with the tarball).

```
[root@samserver] # ./configure \
--with-gpg=/usr/bin/gpg \
--enable-network=server \
--with-database=mysql \
--enable-xml-log \
--with-port=50888 \
--enable-identity=yule
…
[root@samserver] # make && make install
…
  checking whether paths are trustworthy
  configuration file /etc/yulerc ... OK
  state directory /var/run ... OK
  state directory /var/log/yule ... OK
  data directory /var/lib/yule ... OK
…
[root@samserver] # make install-user
…
```

As a consequence of embedding the checksum of the GPG binary, no updates to the GPG binary can be made without recompiling. This serves as an extra layer of protection against modifying the crucial components that Samhain relies on. At the point of installation the user will be asked to provide the GPG pass-phrase that was previously used. Install the Linux specific initialisation file, set the service to start at boot time, start the service and open the firewall.

9 http://www.gnupg.org/documentation/manuals/gnupg-devel/GPG-Options.html

```
[root@samserver] # cp init/samhain.startLinux /etc/init.d/yule
[root@samserver] # chmod 755 /etc/init.d/yule
[root@samserver] # chkconfig --add yule
[root@samserver] # chkconfig yule on
[root@samserver] # service yule start
[root@samserver] # sed -i '/--dport 22 -j ACCEPT/a \
-A INPUT -m state --state NEW -m tcp -p tcp --dport 50888 -j ACCEPT -m comment --comment
"Yule"' /etc/sysconfig/iptables
[root@samserver] # service iptables reload
```

Have a quick look in the log file for any obvious errors, and then move on to the Apache configuration if everything is clean.

3.3 Apache Configuration

The Apache configuration is rather simple for our purpose and is described in Appendix C. Once Apache is up and running, visit "**http://samserver.acme.com/yule.html**" to make sure the status page works. This page is auto refreshed every couple of seconds by default and is useful to see all the current connections to the server at a this point in time.

3.4 Client Setup

As "**root**" on the client system create a GPG key. As before, if there are entropy issues run "**havged**". Capture the fingerprint for this new key so that it can be use when building the Samhain binary.

```
[root@samclient] # gpg --gen-key
…
[root@samclient] # MY_FP=`gpg --fingerprint samhain | \
grep fingerpr | sed 's/ //g' | awk -F "=" '{print $2}'`
```

The above assume that the "**Real name**" field in the GPG key process was set to "**samhain**". Now lets put in a little bit of security by obscurity[10] by changing the name of the

10 http://www.danielmiessler.com/study/security_and_obscurity/

binary and process. The name "**john**", which is the name of the binary in John the Ripper[11], is used here to try and hide what this binary does. Pick a name that will not stand out in a process listing as obviously being part of a HIDS. Also understand how much (or little) a name change actually hides what this binary does before relying on it to obscure a HIDS from an attacker.

It is further specified that the configuration and data files should be pulled from the server, so that even if the local system is compromised, the attacker can not determine what files are monitored for what changes. Make sure to change the below IP to your log server's IP.

```
[root@samclient] # ./configure --with-gpg=/usr/bin/gpg \
--enable-network=client \
--with-config-file=REQ_FROM_SERVER \
--with-data-file=REQ_FROM_SERVER/var/lib/john/john \
--enable-stealth=129 \
--enable-install-name=john \
--with-fp=${MY_FP} \
--with-port=50888 \
--with-logserver=samserver.acme.com \
--with-sender=john \
--enable-static \
--with-timeserver=bigben \
--enable-ptrace
[root@samclient] # make
[root@samclient] # mkdir /var/lib/john/ /usr/local/sbin
[root@samclient] # cp init/samhain.startLinux /etc/init.d/john
[root@samclient] # chmod 744 /etc/init.d/john
[root@samclient] # cp samhain /usr/local/sbin/john
[root@samclient] # cp samhain_setpwd /usr/local/sbin/john_setpwd
[root@samclient] # cp samhain_stealth /usr/local/sbin/john_stealth
[root@samclient] # pushd /usr/local/sbin
```

Set the password as some random HEX[12] value and overwrite the binary. This allows the

11 http://www.openwall.com/john/
12 http://en.wikipedia.org/wiki/Hexadecimal

server to reject connections by unknown clients. By statically compiling the binary, subversion via "**libc**" is no longer possible. Make sure to replace the time server "**bigben**" with an in-house time server. Consider purchasing a GPS time server if there is not one already, as they can be cheap[13] to acquire and then live within your control. If that is not an option, create a local time server that source from multiple[14] geographically separate locations. This will minimise potential interference with the time source.

```
[root@samclient] # /usr/local/sbin/john_setpwd john jingle 161718abcd212324
INFO   old password found
INFO   replaced:  f7c312aaaa12c3f7  by:  161718abcd212324
INFO   finished
[root@samclient] # /bin/mv john.jingle john && popd
```

Change the description in the initialisation script, further improving the obfuscation, and make sure that the new "**john**" starts at boot.

```
[root@samclient] # chkconfig --add john
[root@samclient] # chkconfig john on
[root@samclient] # sed -i 's/File Integrity Checking Daemon/Customised john for password
cracking/' /etc/init.d/john
```

As a side note, at this stage it is possible to further obfuscate the intention of the binary by using a packer such as the Ultimate Packer for eXecutables[15] (UPX). Usage of such packagers (including the one that ships with Samhain) is out of scope, and will not be covered.

3.5 More Server side Configuration

The HEX key from the previous example is now required to tell the server about the client. This way the server authenticates the incoming connection. Any connection that can not be authenticated is rejected. As always, make sure to use the FQDN of the client.

13 https://www.google.com/search?q=gps+time+server#q=gps+time+server&tbm=shop
14 http://www.pool.ntp.org/en/use.html
15 http://upx.sourceforge.net/

```
[root@samserver] # /usr/local/sbin/yule -P 161718abcd212324 | \
sed 's/HOSTNAME/samclient.acme.com/' >> /etc/yulerc
```

Now edit "**/etc/yulerc**" and move the client key above the GPG signature. The last couple of lines from "**/etc/yulerc**" might look like Appendix D; example A. This needs to be changed so that the GPG signature is at the end of the file with nothing below it and the client line is just above the GPG signature as per Appendix D; example B.

Initially the configuration file only has 2 lines at the top that concerns the key (plus an empty line). After the configuration file has been re-signed, there will be 3 lines at the top with regards to the key. See Appendix D; example C and D. Thus there is a safe two step process to delete the key; the first 3 lines and the last 11 lines. There is a script (amongst many others) called "**samhainadmin.pl**" in the scripts directory that will do some of these steps for you. The intention here is to understand the procedure before blindly running scripts.

```
[root@samserver] # sed -i '1,3d' /etc/yulerc
[root@samserver] # sed -i -e :a -e '$d;N;2,11ba' -e 'P;D' /etc/yulerc
```

Whenever changes are made to the configuration file, it has to be re-signed. Fortunately this process is rather simple. Switch user to "**yule**", sign the configuration, move the new signed configuration in place and reload the service.

```
[root@samserver] # su - yule
[yule@samserver] $ gpg -o yulerc.asc -a --clearsign --not-dash-escaped /etc/yulerc
[yule@samserver] $ exit
[root@samserver] # /bin/mv /home/yule/yulerc.asc /etc/yulerc
[root@samserver] # chmod 600 /etc/yulerc && restorecon /etc/yulerc
[root@samserver] # service yule reload
```

At this point the client may show up in the "**http://samserver.acme.com/yule.html**" page. If not, keep in mind that the page is auto-refreshed and it is easy to miss. Have a look at the log file if in doubt.

Martinus Nel, martinus.nel@gmail.com

3.6 Final Client side Configuration

Now create the configuration file and embed it in a postscript file. In short, hiding one file in another is a form of steganography[16]. Go and download a good looking picture such as http://th01.deviantart.net/fs37/PRE/i/2008/277/2/8/Samhain_by_midnightstouch.jpg (Rae, 2008) which is a Samhain (the festival) inspired wall paper. Make sure to use "**samhain_stealth -i**" to test how much data can be hidden so that the configuration file is not too large for the target. Great care should be taken with the transmission and storage of the clear text configuration file. Intercept of the clear text file by an attacker will allow him to know exactly what files are monitored and how said files are monitored. At a minimum, store the clear text configuration on a separate, hardened system with full disk encryption. In the example below, the "**+compress**" option to "**convert**" oddly means to store the image in uncompressed format.

```
[root@samclient] # wget
http://th01.deviantart.net/fs37/PRE/i/2008/277/2/8/Samhain_by_midnightstouch.jpg
[root@samclient] # convert +compress Samhain_by_midnightstouch.jpg \
Samhain_by_midnightstouch.ps
[root@samclient] # cp samhainrc.linux rc.`hostname`
[root@samclient] # gpg -a --clearsign --not-dash-escaped rc.`hostname`
[root@samclient] # /bin/mv rc.`hostname`.asc rc.`hostname`
[root@samclient] # /usr/local/sbin/john_stealth -s Samhain_by_midnightstouch.ps \
rc.`hostname`
IMA START AT: 5755  MAX. CAPACITY: 299586 Bytes
 .. hide rc.samclient.acme.com in Samhain_by_midnightstouch.ps ..
 .. finished
```

Now copy the post script file over to the server and set the correct ownership.

```
[root@samclient] # scp Samhain_by_midnightstouch.ps \
samserver:/var/lib/yule/rc.${HOSTNAME}
[root@samclient] # ssh samserver chown yule:yule /var/lib/yule/rc.${HOSTNAME}
```

16 http://www.symantec.com/connect/articles/steganography-revealed

Every time a change is made to the configuration file it has to be re-signed, stenographically hidden and uploaded to the server. Although the "**root**" user is used to copy the file to the server in this example, using direct "**root**" access in a production environment is not recommended as it is an obvious target[17]. At this stage the database can be built with the following commands.

```
[root@samclient] # /usr/local/sbin/john -t init -p info
INFO   : [2014-02-28T14:08:57+0000] msg=<Downloading configuration file>
INFO   : [2014-02-28T14:08:57+0000] msg=<Session key negotiated>
INFO   : [2014-02-28T14:09:01+0000] msg=<File download completed>
…
MARK   : [2014-02-28T14:09:12+0000] msg=<File check completed.>, time=<11>,
kBps=<111437.240000>
ALERT  : [2014-02-28T14:09:12+0000] msg=<EXIT>, program=<Samhain>, status=<None>
```

This will display a lot of information which can be ignored for now, what is important here is to know that the binary is working as expected. Take note that the "**-t init**" option appends to the database and does not overwrite it. Thus this option should only be used when creating a new database and not for updates. Now that the database is built it needs to be signed and copied to the server.

```
[root@samclient] # gpg -a --clearsign --not-dash-escaped /var/lib/john/john
[root@samclient] # scp /var/lib/john/john.asc samserver:/var/lib/yule/file.${HOSTNAME}
[root@samclient] # ssh samserver chown yule:yule /var/lib/yule/file.${HOSTNAME}
```

Per the way that Samhain looks for files, it is important that all configuration files start with "**rc**", all database files start with "**file**" and that the host names are fully qualified. Once the database is copied over, delete the local copy and test that a check against the server can be made.

```
[root@samclient] # /bin/rm /var/lib/john/*
[root@samclient] # /usr/local/sbin/john -p info -t check --foreground
INFO   : [2014-03-03T08:31:13+0000] msg=<Downloading configuration file>
```

17 http://bsdly.blogspot.de/2013/10/the-hail-mary-cloud-and-lessons-learned.html

```
INFO   :  [2014-03-03T08:31:14+0000] msg=<Session key negotiated>
INFO   :  [2014-03-03T08:31:18+0000] msg=<File download completed>
…
INFO   :  [2014-03-03T08:31:18+0000] msg=<Downloading database file>
INFO   :  [2014-03-03T08:31:39+0000] msg=<File download completed>
```

If all defaults were used there will be some error messages that looks like: "**No matches found**", but what is important right now is that the configuration file and database downloads successfully. As the above command is executed, run "**tail -f /var/log/yule/yule_log**" on the server for further information.

3.7 Cleanup

There should be no breadcrumbs left behind for an attacker to follow up on, thus some clean up is required if this work was done on a production server. If a custom package is built for later deployment, take care what files to include.

- Delete all the source files and any tarballs that were downloaded.
- Delete all entries from the "**root**" and "**yule**" shell history. For the slightly paranoid inclined, run "**shred -uz**" instead of "**rm**". Although, make sure to read the shred man page with regards to modern file system implementations.
- Remove all the development packages that were installed.
- Remove "**/usr/local/sbin/john_stealth**" and "**/usr/local/sbin/john_setpwd**".
- Any other logs that were created, such as STDOUT redirects.

Basically, get rid of any evidence that this program was installed. Keep in mind that development tools on production servers is not considered good practice[18] since these packages may further assist the adversary in staging attacks. Rather build the required packages on a build server, test them, create a rpm / deb / tarball package and then deploy said packages on your production environment.

18 http://security.stackexchange.com/questions/12113/installing-development-tools-on-production-server-security-risk

3.8 Troubleshooting

From time to time things do not work as expected. Here are some pointers for figuring out what may be wrong and where.

- Start with tailing the log file on the server: "**tail --f /var/log/yule/yule_log".**
- If there are no logs generated on the server, run yule with a priority of info and in the foreground: "**yule -p info --foreground".**
- Change the log level in "**/etc/yulerc**" to info or above (always remember to re-sign the configuration file as described).
- Recompile without some of the options to test.
- Have a look at the official documentation (la-samhna.de, 2014).
- Post on the forum / mailing[19] list for further advice.

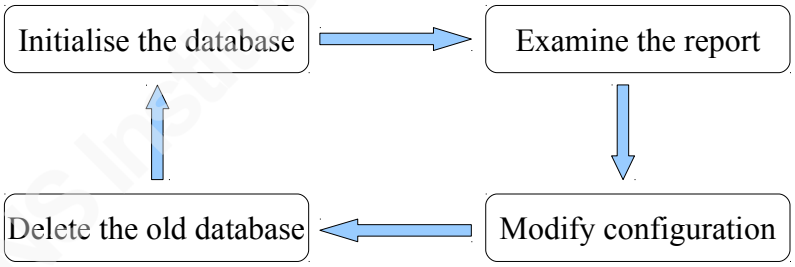
19 http://www.la-samhna.de/elist.html

4 Profiling

Samhain does not understand what is right and wrong for the particular environment it is running in. As such the environment needs to be profiled. The simplest way to do this is to build Samhain in a standalone mode.

```
[root@samclient] # ./configure && make
[root@samclient] # mkdir /var/lib/samhain/
[root@samclient] # cp samhainrc.linux /etc/samhainrc
[root@samclient] # ./samhain -t init -p info > /root/my_output 2>&1
```

Now examine "**/root/my_output**". This is just using the default configuration file which clearly is not going to capture the requirements of this environment. After examining the output file, make the appropriate changes to the Samhain configuration in "**/etc/samhainrc**". By default the database will be created in "**/var/lib/samhain/samhain_file**". As previously stated, do not run "**samhain -t init**" more than once without deleting the database. The refinement procedure will look like this:



A clean configuration should generate no errors when the database is initialised or when checks are made against the database.

```
[root@samclient] # grep -v INFO /root/my_output
ALERT : [2014-03-03T11:25:40+0000] msg=<START>, program=<Samhain>, userid=<0>,
path=</etc/samhainrc>,
hash=<AAF3791DDD69961EE35D71AC8EF4A387A8E581A674FCD7E0>
NOTICE : [2014-03-03T11:25:41+0000] msg=<Module not initialized>, module=<INOTIFY>,
return_code=<9>
```

Martinus Nel, martinus.nel@gmail.com

```
MARK  :  [2014-03-03T11:25:49+0000] msg=<File check completed.>, time=<8>,
kBps=<151518.944000>
ALERT  :  [2014-03-03T11:25:49+0000] msg=<EXIT>, program=<Samhain>, status=<None>
```

After a configuration file has been put together that generates no errors, the next step is to decide how and what files should actually be monitored. By utilising the various predefined policies and building our own policies, a detailed inventory of the system can be made. The default configuration file that is shipped is well commented, so there is no need to regurgitate all the options here. From a layout perspective, use the default configuration file layout where each directory has its own set of policies or alternatively each policy can be defined and then all the files that apply to that policy. This will entirely depend on the individual managing the configuration, but both are valid syntactically.

It is worth mentioning that besides all the policies, there are configuration directives that will directly affect the performance of the system. If this system runs an IO heavy application, such as a write heavy database, then look at changing "**SetIOLimit**" to create less of an impact. For CPU intensive environments, adjust "**SetNiceLevel**" accordingly. Depending on the sensitivity of the environment that is being monitored, "**SetFileCheckTime**" can also be changed to increase or decrease the check frequency. There are no absolute right or wrong numbers for some of these settings. Only a detailed analysis of the application and environment will provide the acceptable overhead of the file integrity monitoring on the application.

Here it will help if the system was built deterministically, for example with a kickstart file, and with the least amount of packages required for the intended function. Any other packages will just add more work in profiling. This is where standardised OS builds and application groups come in handy. Each group, say FTP servers, only has to be profiled once and then the same configuration can be applied to all conforming systems.

Once a working configuration has been created, build Samhain in server / client mode as previously described. It is important that the environment is fully profiled and that Samhain is tuned before it is put into production. Once the system enters production state, the users need to know with high confidence that alerts are genuine and not simple configuration errors. No one

wants to wake up 3am in the morning to an alert that "**/var/log/messages**" changed in size. Missing changes to "**/etc/shadow**" on a system that should authenticate against a 3rd party instead of local files is even worse than being woken up in the early mornings. Thus the requirement for profiling is critical. From here the alerts should be integrated into whatever monitoring platform or Security Information and Event Management (SIEM[20]) is in place.

20 http://searchsecurity.techtarget.com/definition/security-information-and-event-management-SIEM

5 Honey Pot

It would be nice if intruders announce themselves once they are on our system. As such, lets create a couple of files with catchy names and tell Samhain to monitor those files for any changes (that includes access times). For a quick example, copy some files into "**/home/**". This is perhaps not the best of examples, as the attacker will likely realise the trap right after he looked at the files. Yes, it is already too late for him not to be detected, but the attacker should preferably stay clueless that he set off alarms. For more information on honey files, see Yuill, Zappe, Denning, and Feer (2004).

```
[root@samclient] # cp /etc/passwd /home/cracked_passwords
[root@samclient] # cp /etc/hosts /home/credit_cards_2014.xls
```

Now, in Samhain's configuration file there is a section called "**[IgnoreNone]**", add these files in that section. Test this by simply concatenating those files and then run the check. The output should be something like:

```
CRIT : [2014-03-03T12:10:53+0000] msg=<POLICY [IgnoreNone] --------T->,
path=</home/cracked_passwords>, atime_old=<[2014-03-03T10:37:43]>, atime_new=<[2014-03-
03T12:10:43]>,
```

From here make sure that whatever monitoring system / SIEM has been implemented, alerts very loudly to events on the honey files. Further information with regards to file changes can be gathered from "**auditd**"[21]. It is outside the scope to set up and configure "**auditd**", but there are some[22] guides[23] available online. Take note that "**auditd**" puts additional overhead on the system, so take care only to monitor critical files with it.

21 http://man7.org/linux/man-pages/man8/auditd.8.html
22 http://doc.opensuse.org/products/draft/SLES/SLES-security_sd_draft/cha.audit.comp.html
23 http://www.cyberciti.biz/tips/linux-audit-files-to-see-who-made-changes-to-a-file.html

6 Conclusion

At the end of the day, the clear text configuration of each machine being monitored is neither kept on the client nor on the server. The clear text configuration files should be handled as described in 3.6 Final Client side Configuration.

The SIEM / monitoring system makes sure alerts are sent out of any critical policy breach (via e-mail, SMS or voice). With honey files in place, an attacker may reveal himself to administrator much quicker than it might have taken to detect him via other means. Further more, when compiled with "**--enable-nocl=""**" there is no access to any help files via "**./samhain --help**". Nor are any manual pages installed to indicate that there is some file monitoring tools on the client. Even a process output will not show any obvious file monitoring activity as the binary was renamed. Of course, if an attacker get access to the server, he can see all the clients who log in as the logs are in plain text. There are further compiler options so that the logs can also be encrypted.

In the voice of Shrek (Adamson & Jenson, 2001): "Security is like an onion, it has many layers." Remember that host based intrusion detection is just one more layer in this onion and that file integrity monitoring is just one part of HIDS. Besides this, a good firewall, network intrusion detection, monitoring, centralised logging, log analysis, TCP wrappers, SELinux (or some other mandatory access control mechanism), brute force blockers like fail2ban and much more is required as part of a complete security posture. As an example of this, the entire host based intrusion detection is rendered moot if the hacker kills the process and there is no monitoring (or a stateful configuration management engine) to make sure that the service is running.

Although Samhain will fulfil compliance objectives, with regards to FIM, from various laws such as SOX[24] or standards such as PCI DSS[25], each individual organisations requirements will differ. The user has to carefully consider the available products to find the best match, taking into consideration not only the technical merits of each product, but also trust in the code and total cost of ownership.

24 http://www.soxlaw.com/
25 https://www.pcisecuritystandards.org/security_standards/

7 References

Adamson, A., Jenson., V. (2001). Shrek. Retrieved from http://www.imdb.com/title/tt0126029/?ref_=ttqt_qt_tt

Feinler, E., Harrenstien, K. & Stahl, M (1985). Network Working Group Request for Comments 952. Retrieved February 27, 2014, from http://tools.ietf.org/html/rfc952

la-samhna.de. (2014). Download – la-samhna.de. Retrieved from http://www.la-samhna.de/samhain/s_download.html

la-samhna.de. (2014). Documentation – la-samhna.de. Retrieved from http://www.la-samhna.de/samhain/s_documentation.html

Parno, B. (2008). Bootstrapping Trust in a "Trusted" Platform. 3rd USENIX Workshop on Hot Topics in Security

Rae, L. (2008). Retrieved from http://midnightstouch.deviantart.com/art/Samhain-99733274

redhat.com. (2014). Chapter 32. Kickstart Installations – redhat.com. Retrieved from https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Installation_Guide/ch-kickstart2.html

Ullrich, J. B (2013). InfoSec Handlers Diary Blog. Retrieved from https://isc.sans.edu/diary/PHPnet+compromise+aftermath+Why+Code+Signing+Beats+Hashes/16901

Yuill, J., Zappe, M., Denning, D., & Feer, F. (2004). Honeyfiles: Deceptive Files for Intrusion Detection. Proceedings of the 2004 IEEE

Martinus Nel, martinus.nel@gmail.com

8 Appendix A – Kickstart configuration

Please note that this is a minimal kickstart configuration for the specific purpose of this paper and is not meant for production use. If the reader wishes to use this kickstart file as is, see the requirements page.

```
keyboard uk
lang en_GB
install
timezone --utc Europe/London
rootpw password
cdrom
network --onboot yes --device eth0 --bootproto dhcp --noipv6
firewall --service=ssh
selinux --enforcing
bootloader --location=mbr
clearpart --linux --drives=sda
part /boot --fstype=ext4 --size=512 --fsoptions=nosuid,nodev,noexec
part pv.01 --grow --size=1
volgroup rootvg --pesize=4096 pv.01
logvol swap --fstype swap --vgname=rootvg --size=512 --name=swaplv
logvol / --fstype ext4 --vgname=rootvg --size=6144 --name=rootlv
skipx
text
reboot
%packages
@base
vim-enhanced
```

9 Appendix B – Download and unpack Samhain

Download Samhain with:

```
$ wget http://www.la-samhna.de/samhain/samhain_signed-3.1.0.tar.gz
```

Unpack and verify as described on the download page (la-samhna.de, 2014).

```
$ tar -xvzf samhain_signed-3.1.0.tar.gz
samhain-3.1.0.tar.gz
samhain-3.1.0.tar.gz.asc
```

10 Appendix C – Apache sample configuration

This sample configuration is sufficient for our purposes of bringing up the Samhain status page. Further Apache, IPtables and SELinux configuration is left to the reader.

```
[root@samserver] # cat << EOF > /etc/httpd/conf.d/samhain.conf
<Directory "/var/log/yule/">
  Options ExecCGI
  AllowOverride None
  Order allow,deny
  Allow from all
</Directory>

Alias /yule.html "/var/log/yule/yule.html"
EOF
```

Run a test against the Apache configuration, and if that passes start the service.

```
[root@samserver] # apachectl configtest
Syntax OK
[root@samserver] # service httpd start
Starting httpd:                                [  OK  ]
```

Now open up the local firewall for HTTP traffic.

```
[root@samserver] # sed -i '/--dport 22 -j ACCEPT/a \
-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT -m comment --comment
"HTTP"' /etc/sysconfig/iptables
[root@samserver] # service iptables reload
```

Finally give the "**apache**" user read access to the html file and create a simple SELinux module for the same purpose.

Martinus Nel, martinus.nel@gmail.com

```
[root@samserver] # setfacl -m u:apache:r /var/log/yule/yule.html
[root@samserver] # yum -y install policycoreutils-python
[root@samserver] # cat << EOF > http_yule_ro.te

module http_yule_ro 1.0;

require {
        type var_log_t;
        type httpd_t;
        class file read;
}

#============= httpd_t ==============
allow httpd_t var_log_t:file read;
EOF
[root@samserver] # checkmodule -M -m -o http_yule_ro.mod http_yule_ro.te
[root@samserver] # semodule_package -o http_yule_ro.pp -m http_yule_ro.mod
[root@samserver] # semodule -i http_yule_ro.pp
```

Note that this SELinux module allows read access to "**var_log_t**" labelled files and not just "**yule.html**".

11 Appendix D – Sample GPG headers and footers

Example A, client entry at end of file, below GPG signature.

```
# Client=HOSTNAME@00000000@C39F0EEFBC64E4A8BBF72349637CC07577F714B482...
# Client=HOSTNAME@8F81BA58956F8F42@8932D08C49CA76BD843C51EDD1D664051...
-----BEGIN PGP SIGNATURE-----
…
-----END PGP SIGNATURE-----
Client=samclient.acme.com@0AA9A1E20DF8D678@AE5D3DE54E3BF79FCC2B9BED9770...
```

Example B, client entry moved above GPG signature.

```
# Client=HOSTNAME@00000000@C39F0EEFBC64E4A8BBF72349637CC07577F714B628...
# Client=HOSTNAME@8F81BA58956F8F42@8932D08C49CA76BD843C51EDD1D664053...
Client=samclient.acme.com@0AA9A1E20DF8D678@AE5D3DE54E3BF79FCC2B9BE7DD0...
-----BEGIN PGP SIGNATURE-----
…
-----END PGP SIGNATURE-----
```

Example C, initial GPG header.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
```

Example D, GPG header after signing.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
NotDashEscaped: You need GnuPG to verify this message
```

Martinus Nel, martinus.nel@gmail.com

Last Updated: May 24th, 2019

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS San Antonio 2019	San Antonio, TXUS	May 28, 2019 - Jun 02, 2019	Live Event
SANS Atlanta 2019	Atlanta, GAUS	May 28, 2019 - Jun 02, 2019	Live Event
Security Writing NYC: SEC402 Beta 2	New York, NYUS	Jun 01, 2019 - Jun 02, 2019	Live Event
Enterprise Defense Summit & Training 2019	Redondo Beach, CAUS	Jun 03, 2019 - Jun 10, 2019	Live Event
SANS Zurich June 2019	Zurich, CH	Jun 03, 2019 - Jun 08, 2019	Live Event
SANS London June 2019	London, GB	Jun 03, 2019 - Jun 08, 2019	Live Event
SANS Kansas City 2019	Kansas City, MOUS	Jun 10, 2019 - Jun 15, 2019	Live Event
SANS SEC440 Oslo June 2019	Oslo, NO	Jun 11, 2019 - Jun 12, 2019	Live Event
SANSFIRE 2019	Washington, DCUS	Jun 15, 2019 - Jun 22, 2019	Live Event
Security Operations Summit & Training 2019	New Orleans, LAUS	Jun 24, 2019 - Jul 01, 2019	Live Event
SANS Cyber Defence Canberra 2019	Canberra, AU	Jun 24, 2019 - Jul 13, 2019	Live Event
SANS ICS Europe 2019	Munich, DE	Jun 24, 2019 - Jun 29, 2019	Live Event
SANS Cyber Defence Japan 2019	Tokyo, JP	Jul 01, 2019 - Jul 13, 2019	Live Event
SANS Munich July 2019	Munich, DE	Jul 01, 2019 - Jul 06, 2019	Live Event
SANS Paris July 2019	Paris, FR	Jul 01, 2019 - Jul 06, 2019	Live Event
SANS London July 2019	London, GB	Jul 08, 2019 - Jul 13, 2019	Live Event
SANS Cyber Defence Singapore 2019	Singapore, SG	Jul 08, 2019 - Jul 20, 2019	Live Event
SEC450 Security Ops-Analysis Beta 1	Crystal City, VAUS	Jul 08, 2019 - Jul 13, 2019	Live Event
SANS Pittsburgh 2019	Pittsburgh, PAUS	Jul 08, 2019 - Jul 13, 2019	Live Event
SANS Charlotte 2019	Charlotte, NCUS	Jul 08, 2019 - Jul 13, 2019	Live Event
SANS Rocky Mountain 2019	Denver, COUS	Jul 15, 2019 - Jul 20, 2019	Live Event
SANS Columbia 2019	Columbia, MDUS	Jul 15, 2019 - Jul 20, 2019	Live Event
SANS San Francisco Summer 2019	San Francisco, CAUS	Jul 22, 2019 - Jul 27, 2019	Live Event
SANS Pen Test Hackfest Europe 2019	Berlin, DE	Jul 22, 2019 - Jul 28, 2019	Live Event
DFIR Summit & Training 2019	Austin, TXUS	Jul 25, 2019 - Aug 01, 2019	Live Event
SANS Riyadh July 2019	Riyadh, SA	Jul 28, 2019 - Aug 01, 2019	Live Event
SANS Boston Summer 2019	Boston, MAUS	Jul 29, 2019 - Aug 03, 2019	Live Event
SANS July Malaysia 2019	Kuala Lumpur, MY	Jul 29, 2019 - Aug 03, 2019	Live Event
Security Awareness Summit & Training 2019	San Diego, CAUS	Aug 05, 2019 - Aug 14, 2019	Live Event
SANS London August 2019	London, GB	Aug 05, 2019 - Aug 10, 2019	Live Event
SANS Crystal City 2019	Arlington, VAUS	Aug 05, 2019 - Aug 10, 2019	Live Event
SANS Melbourne 2019	Melbourne, AU	Aug 05, 2019 - Aug 10, 2019	Live Event
SANS Krakow May 2019	OnlinePL	May 27, 2019 - Jun 01, 2019	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced