# Global Information Assurance Certification Paper

# An Approach to Detect Malware Call-Home Activities

*GIAC (GCIH) Gold Certification*

Author: Tyler (Tianqiang) Cui, tianqiang.cui@gmail.com
Advisor: Joel Esler

Accepted: December 16th, 2013

Abstract

It is very common for active malware to call home, either to fetch updates and instructions or to send back stolen information. In an internal network where web access to the Internet must go through a proxy, the traffic that doesn't pass through the proxy and by default is dropped by the gateway firewall could be valuable to detect malware call-home activities. This paper describes an approach to detect such malware call-home activities by redirecting the otherwise dropped traffic to a sinkhole server in a proxy environment.

[August 2013]

# 1. Introduction

In the internal network of a large organization, there may be a number of security measures or products in place, such as anti-virus, security patch management, Intrusion Prevention Systems (IPS), Firewalls, etc., and there is still some malware that goes undetected.

One of the activities that malware will conduct is "call-home", to either fetch updates and instructions from the remote Command and Control (C&C) servers, or send back stolen information. It is challenging but also may be fruitful to proactively detect these malware call-home activities.

This paper describes an approach to detect certain malware call-home activities by redirecting their traffic otherwise dropped by a gateway firewall to a sinkhole server for analysis in a proxy environment.

## 1.1. Background

In internal networks that have relatively strict access controls, desktop computers (workstations, laptops, etc.) always have to go through web proxies to access the Internet, and traffic going to the Internet directly from desktops usually is not allowed.

At the same time, though increasingly more malware are becoming proxy-aware, there are still a big percentage of malware are not proxy-aware.

What will happen is, many non proxy-aware pieces of malware will attempt to communicate to C&C servers on the Internet directly and will then be blocked by the gateway firewall.

Tyler Cui, tianqiang.cui@gmail.com

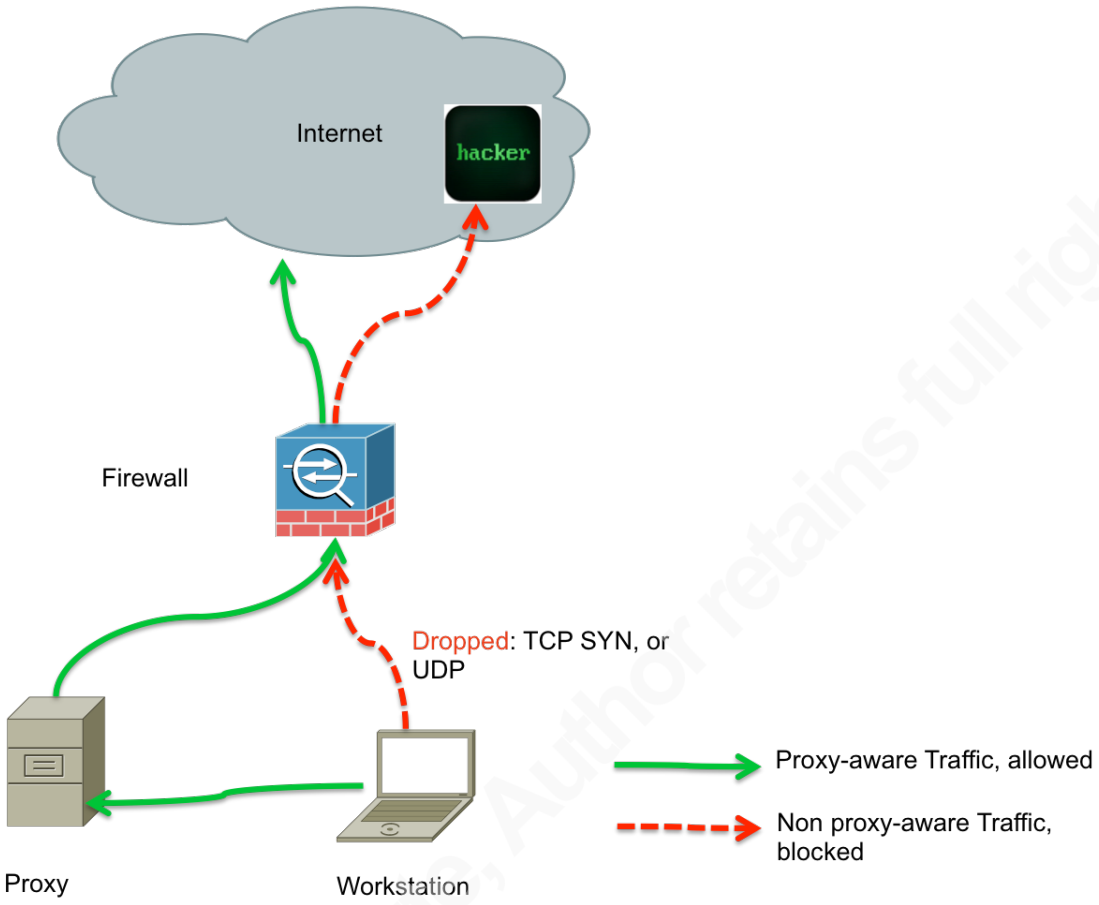


Figure 1-1 Desktops outgoing traffic

As illustrated in Figure 1-1, due to non proxy-aware packets (TCP SYN, or UDP) are blocked by the gateway firewall, the attempted conversations between the desktops and the peers on the Internet will never be established.

## 1.2. Opportunity for malware detection

The traffic being blocked by the firewall could be a valuable data source for malware call-home detection. The ratio of malware call-home activities among the traffic being blocked by the firewall could be much higher than the one among the traffic passing through proxy.

### 1.1.1. Firewall log review

By reviewing the firewall logs, it should be very helpful for detecting malware call-home activities on the internal network. For example, by checking:

- The IP addresses attempted to talk to malicious IP addresses on the Internet

- The common ports used by malware for call-home activities, e.g. HTTP, FTP, SMTP, and IRC.

### 1.1.2. Requirement for deeper insight

It may still be hard to tell whether some traffic is malicious just by reviewing firewall logs. As illustrated in Figure 1-1, due to the traffic being blocked by the firewall (e.g. for TCP protocol, the connections are never established), we could not know what requests the desktops attempt to send to the Internet.

If we could go one step further (i.e. to forward the traffic otherwise dropped by the firewall to a sinkhole server, listen on the corresponding ports, interact with the incoming requests, and record the detail requests), it might give us deeper insight into the traffic being blocked by firewall, and potentially it could be very helpful for security incident monitoring and response:

- The data can be analyzed to proactively detect malware.
- The data can be used for investigation when there is a malware incident.

For this purpose, some research and test have been done for this approach.

## 1.3. Scope

The purpose of the test was just to verify that the approach will work as expected, and to explore how much valuable information the approach could provide for malware monitoring and investigation, e.g. picking up the abnormal behaviors, and giving enough detail information of the abnormal behaviors for further analysis. Thus this paper will not go further into the malware analysis area for the malware being tested.

# 2. Test Environment

## 1.1. List of systems

In this test, a network was set up in VMware environment, and the following systems were used:

| System | OS | IP | Application | Segment |
|---|---|---|---|---|
| Firewall | Checkpoint R70 (SecurePlatform) | 172.16.17.101(to Internet)<br>192.168.0.101(to internal)<br>10.0.0.101(to server) | | |
| Workstation | Windows XP SP3 | 192.168.0.142 | | Internal |
| Proxy | Ubuntu 12 | 192.168.0.143 | Squid | Internal |
| Log-Server | Ubuntu 12 | 10.0.0.100 | Apache 2.2 (with ModSecurity) | Server |

## 1.2. Topology diagram

The topology of the test environment is shown as Figure 2-1:

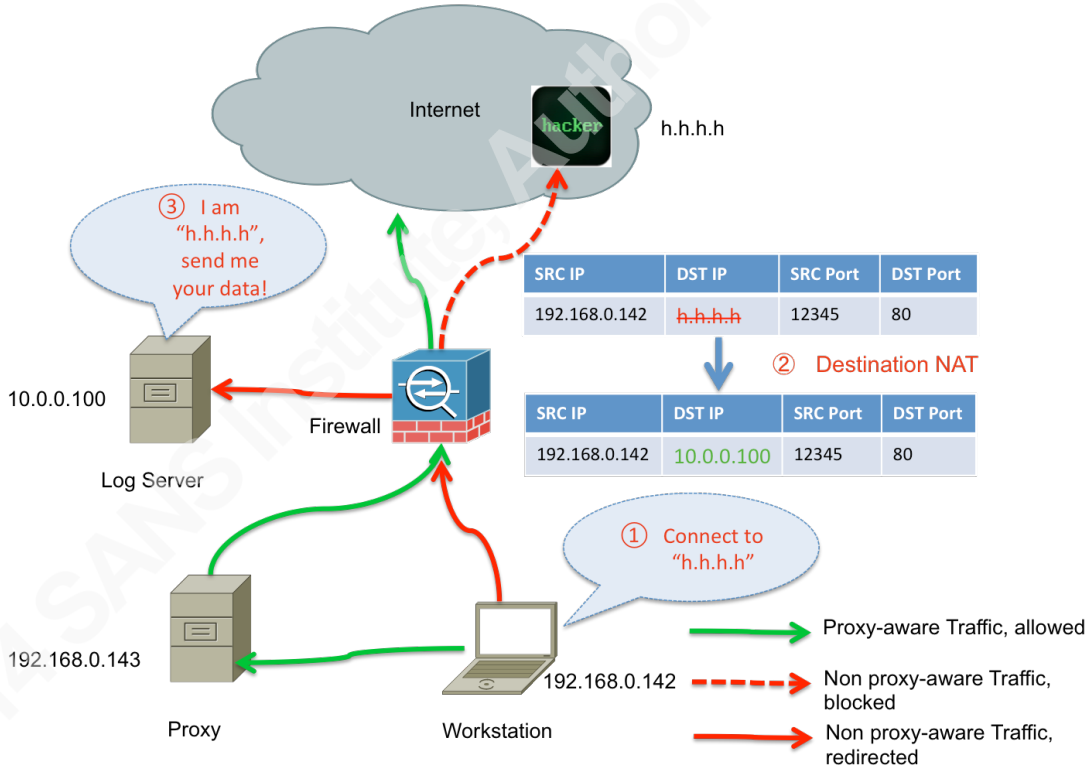


Figure 2-1 Desktops outgoing traffic being redirected

In the scenario the malware on the desktop is not proxy-aware, when it calls home to the malicious IP address, h.h.h.h on the Internet, the traffic will reach the firewall directly. When the firewall receives the traffic, since it is not coming from the proxy, the firewall will do a Destination NAT, and change the destination IP address to the log

server, 10.0.0.100. The desktop is not aware of the translation, so the conversation will be established successfully with the log server instead of the malicious IP address on the Internet.

## 1.3. Configuration

### 1.1.1. Firewall configuration

There needs to be a firewall rule to forward the traffic of interest to a log server. This can be achieved with the features like so-called "Destination NAT", or "Port Forwarding" on a Checkpoint firewall.

Ideally, the Port Forwarding rule should be the second to last rule, just above the last default rule, "Deny all".

The figure 2-2 demonstrates the rule sets that worked as expected in this test:

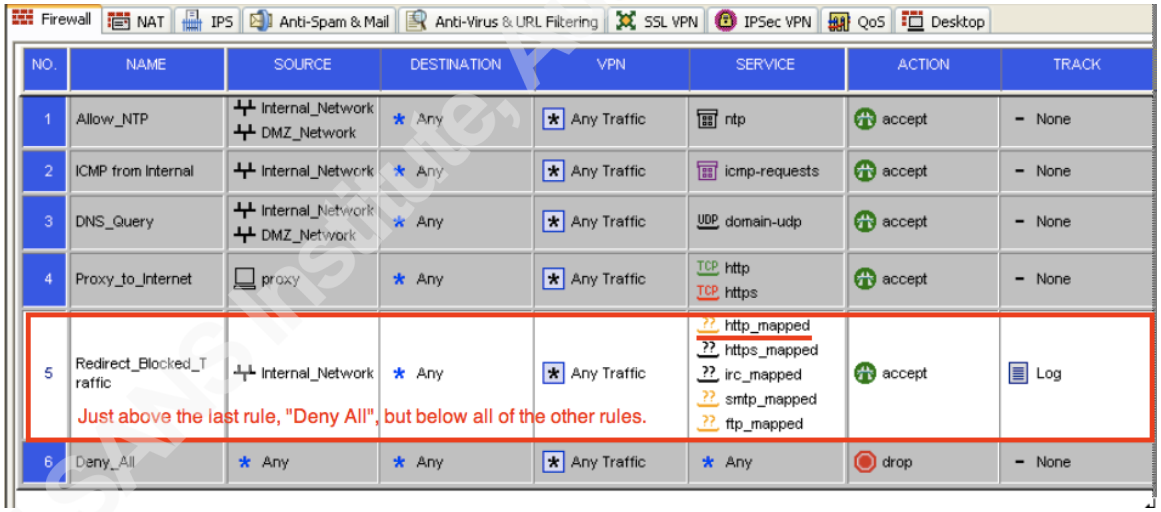


Figure 2-2 Firewall rules added

The rule No. 5 was added just above the last rule, "Deny All", to forward the otherwise dropped traffic of FTP, SMTP, HTTP, HTTPS and IRC (TCP 6667) to the corresponding ports on the log server. If you use the default implicit "Deny All" rule and don't have an explicit "Deny All" rule at the end of the rule sets, then this rule should be the last rule on the rule sets.

On a Checkpoint firewall, the feature, Port Forwarding was used to do the traffic forwarding (Destination NAT). For instance, the settings of the service, "http_mapped" is shown as below, it forwards all the traffic with destination port 80 to port 80 of the log server, 10.0.0.100:

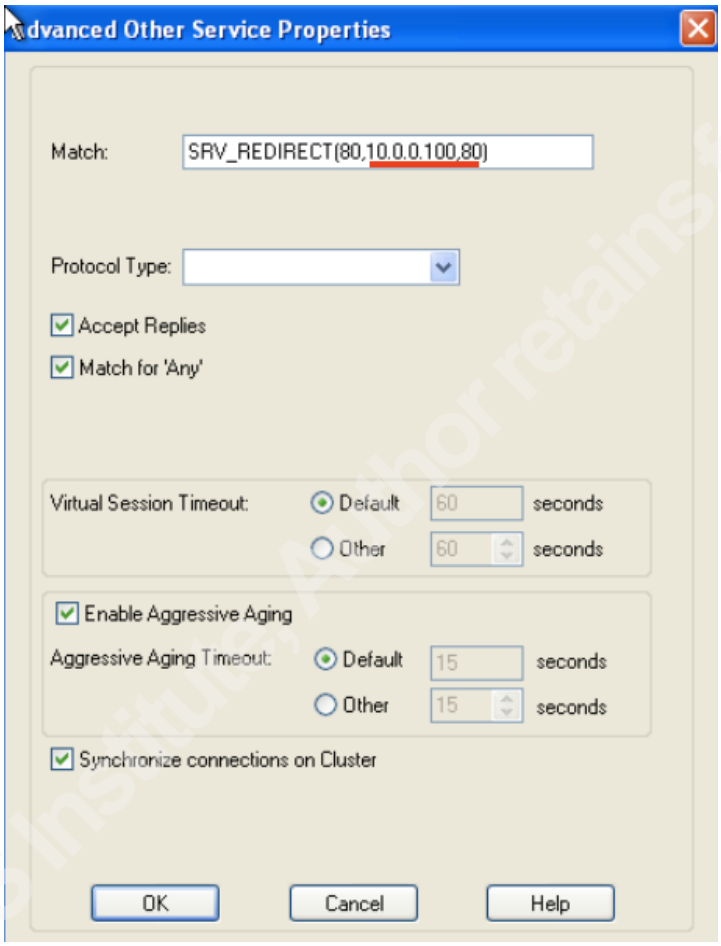


Figure 2-3 Port Forwarding on Checkpoint

Similar features should also be supported by other mainstream firewalls.

### 1.1.2. Log server configuration

There will need to be daemons listening on the log server on each of the corresponding ports, log the requests, and even to interact with the clients for some protocols.

The logging functions need to log as much information as possible, at least the ones of interest as below:

- Time stamp
- Source IP Address
- Source Port
- Original Destination IP/Hostname (this is only available for HTTP/HTTPS protocol)
- Original Destination Port (remain unchanged)
- The payload

The original destination IP address is changed by the firewall when being forwarded to the log server. For HTTP/HTTPS traffic, luckily the "Original Destination IP/Hostname" is available on their "Host" header. But for other protocols like FTP, SMTP, IRC, etc., there is no such information available. To find out the original destination IP address for those protocols, we might have to correlate with the firewall NAT logs by the timestamp, source IP and source Port.

In this test Apache with the ModSecurity module was used to collect logs for port 80 and port 443, and socat (dest-unreach.org) and Bash scripts were used to collect logs for port 21, 25 and 6777. For detailed information, refer to Section 3.

### 1.1.3. Scripts to parse logs collected

For a network with thousands or even tens of thousands of desktops, the volume of the logs for just the HTTP protocol may be substantial because some legitimate applications may not be proxy-aware. And usually it is necessary to use scripts to automate the analysis process. In this test Bash and Perl scripts were used to check the suspicious requests on port 80 and 443, which potentially initiated by malware. The scripts support whitelist feature, for example, it could handle usually trusted networks like "microsoft.com" and "google.com".

If there is strict access control in place, i.e. only allowed to access the Internet through a proxy, there shouldn't be too much traffic logged for protocols like FTP, SMTP and IRC, and it may be practical to review those requests by just reading them one by one. So there were no scripts used for these protocols in the test.

# 3. Test results

## 1.1. HTTP/HTTPS

### 1.1.1. Service listening

As mentioned previously, there was an Apache server running on the log server and listening on ports 80 and 443. The open source Web Application Firewall, "ModSecurity" was used to collect the logs, including request header and request body (for POST request).

When visiting the Internet from the desktops through the proxy, everything worked well.

If the web traffic was not sent through a proxy, all the requests would be redirected to a page, index.php on the log server by URL rewriting as below:

```
root@log-server:/var/www# pwd
/var/www
root@log-server:/var/www# cat .htaccess
RewriteEngine on
RewriteRule ^.*$ index.php [L]
```

The page index.php (refer to Appendix 1.) was used to display alert information. For instance, if a new employee was not aware of the proxy and attempted to connect to any websites on the Internet directly, message below would be displayed on the browser being used:

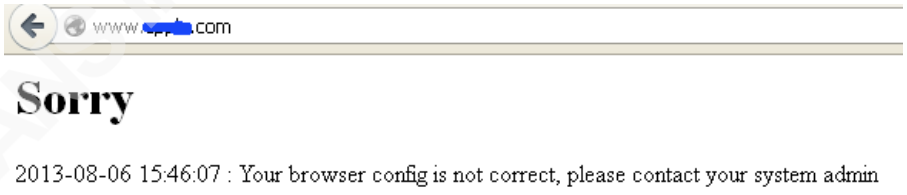


Figure 3-1 HTTP/HTTPS alert to users

Then a human user would know his browser configuration was wrong, but a malicious piece of software would not be aware.

### 1.1.2. Samples tested

For testing purpose, a few malware samples were downloaded from the Internet (Inside Your Botnet & Malwr) and run on the Windows XP workstation.

### 1.1.3. Test result

#### 1.1.1.1. Log entry example

A typical entry of the logs captured is below, it includes useful information such as timestamp, source IP, source Port, requested URL, Host, User-Agent, Cookie, and POST data body.

```
--fef9507d-A--
[31/Jul/2013:18:36:33 --0700] Ufm7oX8AAQEAAARIBR0AAAAB 192.168.0.142 49263 10.0.0.100 80
--fef9507d-B--
POST /safebrowsing/downloads?client=navclient-auto-ffox&appver=22.0&pver=2.2&wrkey=AKEgNiv22
ai_5il5YmqHF_DRfd3QeZjlLXN-Myt4-J2qhKXtLLgBc_49PIBT-w== HTTP/1.1
Host: safebrowsing.clients.google.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Length: 110
Content-Type: text/plain
Cookie: PREF=ID=7d9df8b64a44caa6:U=1895a97dbdc88f2e:FF=0:TM=1373673272:LM=1374325048:S=dt_Db
32zXhUGXMh_Z3DkkbL_3s0DiqNwaiSL3DMtj7XK7HwIgQH7aO4BS_ZP5mF31SVc9bU63Gu00dC_wS9fJd2mmXb6pFsuc
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

--fef9507d-C--
goog-malware-shavar;a:112550-121620:s:110721-118260:mac
goog-phish-shavar;a:286603-293703:s:137825-142303:mac
```

Figure 3-2 Detail HTTP log entry

In the example above, the client attempted to talk to google.com, which is trusted and could be whitelisted.

#### 1.1.1.2. Host header analysis

The Host request-header field specifies the Internet host and port number of the resource being requested, as obtained from the original URI given by the user or referring resource (rfc2616). After extracting the Host header from the logs and sorting by number of hits, we could see some of the hosts requested were very suspicious:

- Hosts with high number of hits, which could be malware behavior
- Hosts use IP addresses instead of a domain names
- Hosts with very long and meaningless domain names

In the example below, the hosts highlighted could be identified as suspicious quickly, and identified as traffic generated by malware.

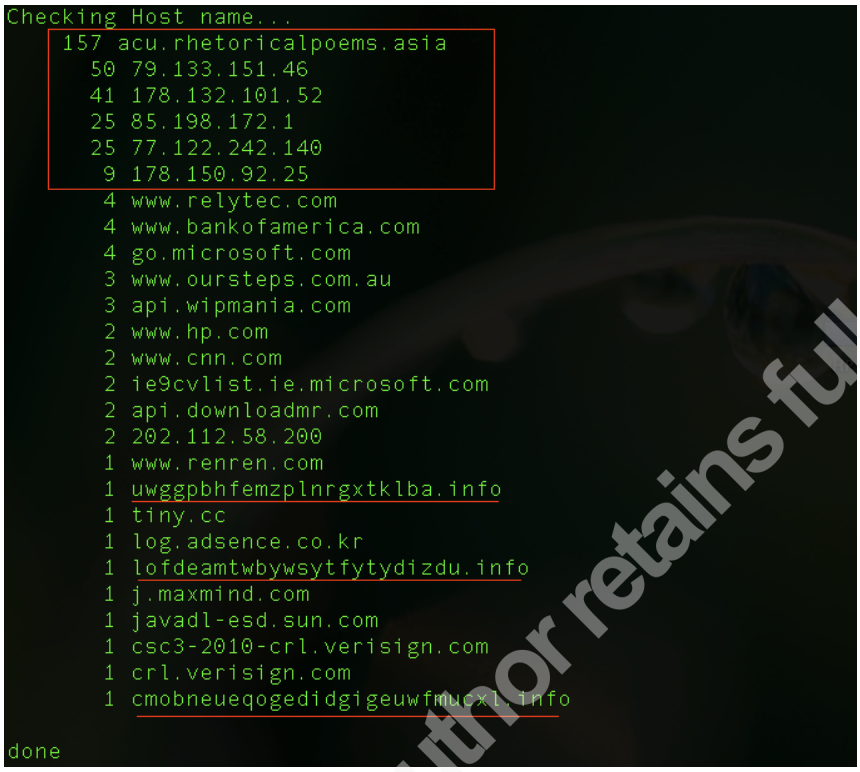


Figure 3-3 Host names sorted by number of hits

#### 1.1.1.3. User-Agent header analysis

The User-Agent request-header field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations (rfc2616). User-Agent header could also be very helpful to detect malware. In the example below, after extracting the "User-Agent" header from the logs and sorting by number of hits, we could see some of the User-Agents as highlighted were apparently suspicious:

- There were User-Agents for 64-bit Linux, or Opera, which didn't exist in the test environment.
- There was User-Agent "Mozilla/4.0".

By examining the detail logs further, it's confirmed they were all generated by malware.

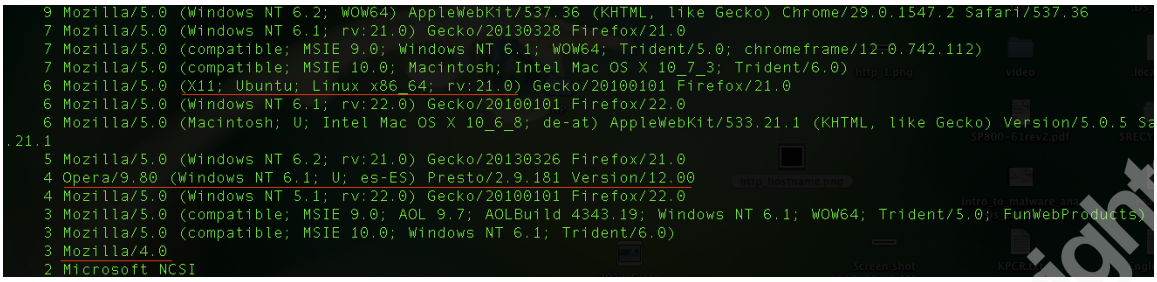


Figure 3-4 User agents sorted by number of hits

Further work could be done to detect suspicious User-Agents that may indicate malware behavior, for example, the User-Agent shown on the logs could be further compared with:

- All known User-Agents. (In this example, "Mozilla/4.0" could be detected)
- All User-Agents showed up on the proxy logs of an organization. (Then in this example, the malware could be detected if the corporate didn't have 64-bit Linux as end user desktops.)

#### 1.1.1.4. Requested URL analysis

Reviewing the URL requested could also be very helpful, especially the ones that:

- With high number of hits
- With sensitive names, such as gate.php, cfg.bin (always used by Zeus)

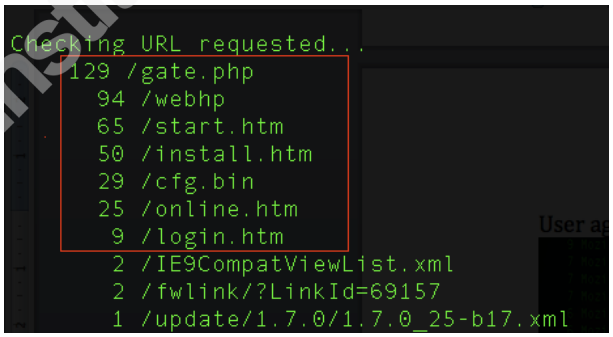


Figure 3-5 URLs sorted by number of hits

In the example above, all the URLs highlighted were suspicious. If we take the URL "gate.php" as an example and look into the detail HTTP logs, we can see the request was suspicious because it attempted to talk to a suspicious domain, acu.rhetoricalpoems.asia.

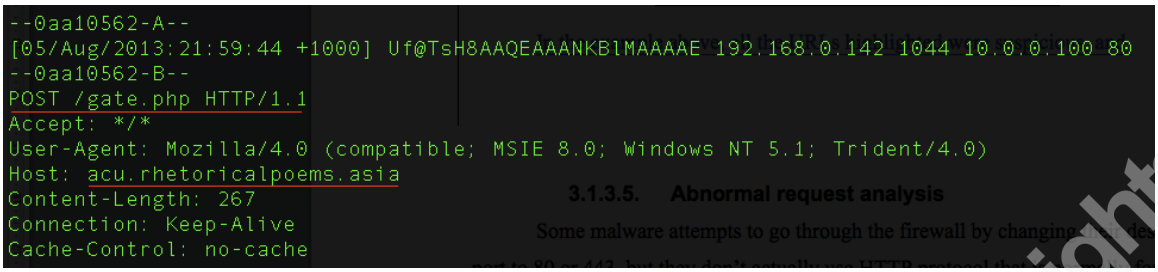


Figure 3-6 Detail log of suspicious URL gate.php

### 1.1.1.5. Abnormal request analysis

Some malware attempts to go through the firewall by changing their destination port to 80 or 443, but they don't actually use HTTP protocol that is normally found on those ports. This can be detected by checking if the traffic follows the convention of the normal HTTP protocol. A script abnormal_scan.pl (refer to Appendix 9.) has been written for this purpose.

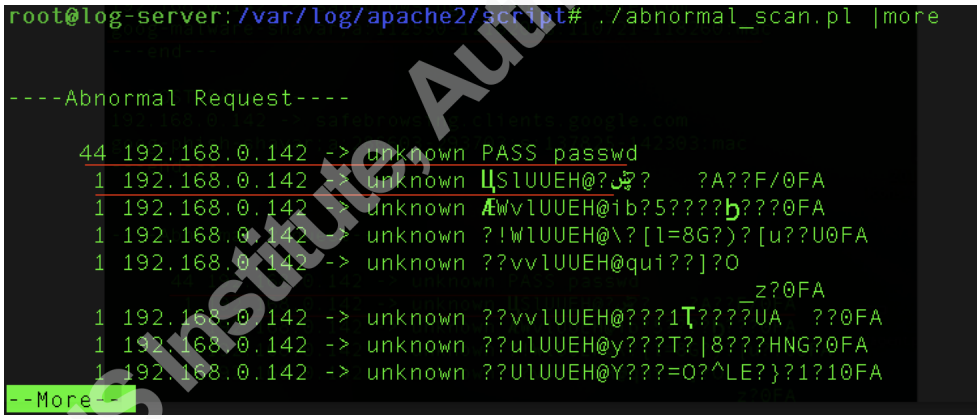


Figure 3-7 Abnormal requests sorted by number of hits

In the example above, the source IP, 192.168.0.142 attempted to send some traffic through port 80 and 443, but the traffic wasn't HTTP. The following information could be found in the detail logs, which was highly suspicious:

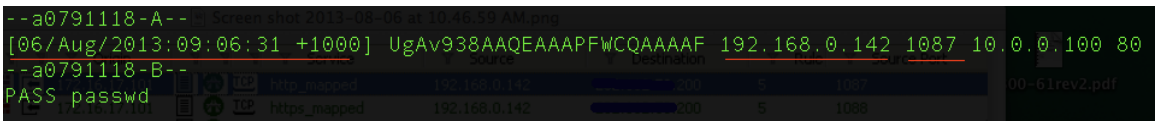


Figure 3-8 Detail log of an abnormal request

In this case the original destination IP address was unknown in the log. However, it could be find out by correlating the firewall logs with fields including timestamp, source IP, and source Port.

| No. | Date | Time | Origin | Service | Source | Destination | Rule | Source Port |
|---|---|---|---|---|---|---|---|---|
| 420 | 6Aug2013 | 9:06:30 | 172.16.17.101 | http_mapped | 192.168.0.142 | [redacted]200 | 5 | 1087 |
| 421 | 6Aug2013 | 9:06:46 | 172.16.17.101 | https_mapped | 192.168.0.142 | [redacted]200 | 5 | 1088 |

Figure 3-9 Correlation with firewall NAT logs

Then we could investigate the original destination IP address further by other measures (such as Google search, Whois lookup, passive DNS, etc.) to identify if it was malicious.

## 1.2. FTP

In an internal network, machines with wrong FTP servers configured or users unaware of the firewall policies may attempt to talk to FTP servers on the Internet directly. Some malware or key loggers may upload stolen information to the Internet through FTP protocol. So in this test FTP traffic to TCP port 21 was captured.

### 1.1.1. Services listening

Socat, a more complex variant of netcat was running on the log server and listening on port 21, and a Shell script, ftp.sh (customized based on Maik Ellinger's FTP script for Honeyd project, refer to Appendix 2) was used to simulate FTP services and to log the incoming traffic.

```
socat TCP-L:21,reuseaddr,pktinfo,fork EXEC:"ftp.sh"
```

### 1.1.2. Samples tested

To gain deeper insight into the outgoing FTP traffic otherwise dropped by firewall, in this test, the Relytec Key Logger software was tested, which can send stolen information at specified intervals.

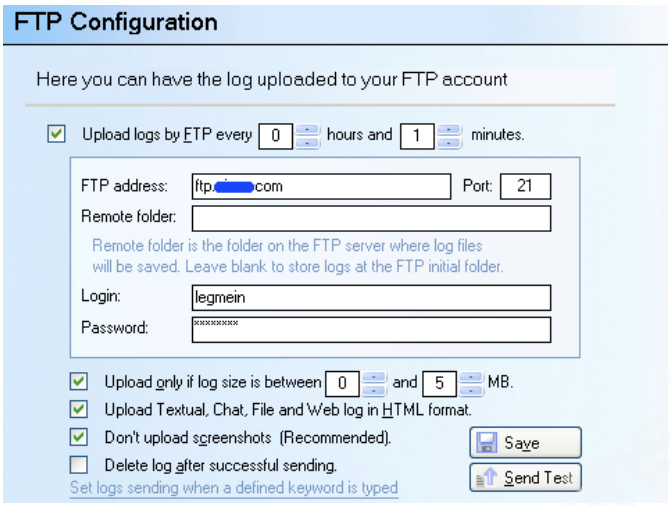


Figure 3-10 Relytec Key Logger FTP Configuration

### 1.1.3. Test result

The script ftp.sh working with socat successfully intercepted the FTP communication initiated from the key logger, and collected useful information as below:

- Time stamp
- Username to login
- Password to login
- Source IP (the compromised machine)
- Source Port
- Name of the file the key logger attempted to upload

```
2013-07-31 20:56:35 192.168.0.142 1094 USER legmein
2013-07-31 20:56:35 192.168.0.142 1094 PASS password_123
2013-07-31 20:56:35 192.168.0.142 1094 TYPE A
2013-07-31 20:56:35 192.168.0.142 1094 MKD ftp_reports
2013-07-31 20:56:35 192.168.0.142 1094 SITE CHMOD 700 ftp_reports
2013-07-31 20:56:35 192.168.0.142 1094 CHMOD 700 ftp_reports
2013-07-31 20:56:35 192.168.0.142 1094 QUOTE SITE CHMOD 700 ftp_reports
2013-07-31 20:56:35 192.168.0.142 1094 CWD ftp_reports
2013-07-31 20:56:36 192.168.0.142 1094 PASV
2013-07-31 20:56:36 192.168.0.142 1094 STOR ftp_instructions.txt
```

Figure 3-11 FTP logs

The original destination IP address of the FTP server was not available in the log, because it was changed by the NAT. However, this could be found by correlating with firewall logs by time stamp, source IP, and source Port.

## 1.3. SMTP

In an internal network, machines with wrong SMTP servers configured may attempt to talk to SMTP servers on the Internet directly. While some malware and key logger may also use SMTP to either spread SPAM or send stolen information to specified mailbox. So in this test SMTP traffic to TCP port 25 was captured.

### 1.1.1. Services listening

Socat was also running on the log server and listening on port 25, and a Shell script, smtp.sh (customized based on Maik Ellinger's SMTP script for Honeyd project, refer to Appendix 3) was used to simulate the SMTP service and to log the incoming traffic.

```
socat TCP-L:25,reuseaddr,pktinfo,fork EXEC:"smtp.sh"
```

### 1.1.2. Samples tested

In this test, again, the Relytec Key Logger software was tested, which would send stolen information at specified intervals.

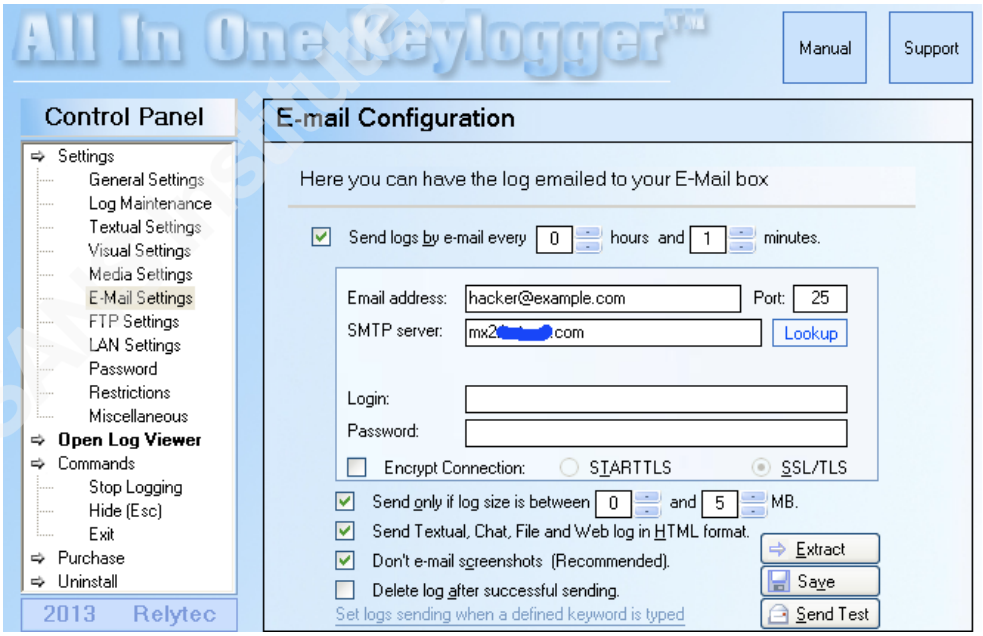


Figure 3-12 Relytec Key Logger SMTP Configuration

### 1.1.3. Test result

The script smtp.sh working with socat successfully intercepted the SMTP communication initiated from the key logger, and collected useful information as below:

- Time stamp
- Source IP (the compromised machine)
- Source Port
- Sender of the email (Could be fake)
- Recipient of the email
- CC, BCC list of the email
- Subject of the email

```
2013-08-02 20:39:31 192.168.0.142 1048 EHLO 192.168.0.142
2013-08-02 20:39:32 192.168.0.142 1048 MAIL FROM: <hacker@example.com>
2013-08-02 20:39:32 192.168.0.142 1048 RCPT TO: <hacker@example.com>
2013-08-02 20:39:32 192.168.0.142 1048 DATA
2013-08-02 20:39:33 192.168.0.142 1048 Subject: Report from ROOTEVER. Id=[q68v2yyq7ms]
2013-08-02 20:39:33 192.168.0.142 1048 From: hacker@example.com
2013-08-02 20:39:33 192.168.0.142 1048 To: hacker@example.com
```

Figure 3-13 SMTP logs

The information above should be helpful to identify if the email was suspicious or not. The original destination IP address of the SMTP server was not available in the log, because it was changed by the NAT. However, if necessary this could be found out by correlating with firewall logs by time stamp, source IP, and source Port.

## 1.4. IRC

In an internal network that doesn't allow outgoing IRC traffic, machines with IRC clients installed may still attempt to talk to IRC servers on the Internet directly. At the same time, it is still common for some malware to contact C&C servers on the Internet through IRC protocol. So in this test IRC traffic to TCP port 6667 was captured.

### 1.1.1. Listening service

Socat was also set up on the log server to listen on port 6667, and a Shell script, irc.sh (refer to Appendix 4) was used to log the incoming traffic.

```
socat TCP-L:6667,reuseaddr,pktinfo,fork EXEC:"irc.sh"
```

### 1.1.2. Sample tested

For testing purposes, a few malware samples were downloaded from the Internet and ran on the Windows XP workstation.

### 1.1.3. Test result

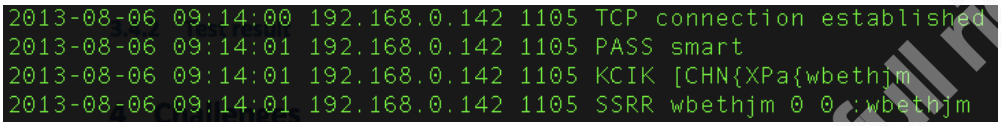


Figure 3-14 IRC logs

The information above was collected from the logs for common IRC ports, and was helpful to try and identify if the traffic was suspicious or not.

The original destination IP address of the IRC server was not available in the log, because it was changed by the NAT. However, if necessary, this could be found by correlating with firewall logs by time stamp, source IP, and source Port.

# 4. Challenges

The test worked as expected and demonstrated potentially a new approach to detect malware call-home activities. However, there are some challenges to be either solved or considered for this approach.

## 1.1. Security consideration

The approach itself might introduce new security risk to the network, and they should be considered and addressed before implementation.

### 1.1.1. Log server security

The scripts such as ftp.sh and smtp.sh that interact with the clients might have potential vulnerabilities, e.g., shell injection vulnerabilities. So they should be carefully reviewed to make sure they are secure.

### 1.1.2. Sensitive information disclosure

The log server will log some sensitive information if legitimate users connect to the listening services due to misconfiguration, such as:

- Sensitive information in POST data body
- FTP, SMTP, IRC credentials

The countermeasures below could be used to address the concern:

- Hardening the log server to restrict the access to the logs
- Replacing passwords captured with ******** on the logs

#### 1.1.3. Reminder to users

This approach is intended to detect bot activities, especially bot activities that are malicious. However, sometimes a human may also connect to the ports that the log servers are listening on for some reason. It is necessary to pop up an alert stating that his/her machine's configuration may be wrong and it is advised to contact your organization's helpdesk.

## 1.2. Malware evolvement

Malware is evolving, and in the past a few years, some malware authors are making it harder to tell if a request is legitimate or malicious by using similar technologies like sinkhole. Malware authors may use a mechanism to bypass such an approach as described above, for example:

- Make the malware be proxy-aware
- Don't send useful information until the Command & Control server send some identification information that the bot trusts

## 1.3. Limitations

#### 1.1.1. Strict access control environment

There are also limitations for this approach. It is only useful in a network environment with a strict access control via proxy. In a loose access control environment, there is no proxy, and the end users can access any services on the Internet, this approach is then useless.

Ideally, the approach is best used in a network environment with a proxy, as all or most of the traffic from end users to the Internet must go through the proxy. And the proxy cannot be a transparent proxy, otherwise the network devices responsible for port

forwarding will forward all the outgoing web traffic to proxy, including the ones initiated by malware, no matter it is proxy-aware or not.

### 1.1.2. Support for other protocols

The test didn't consider other protocols such as UDP, or TCP ports other than 21, 25, 80, 443 and 6667. Those techniques could also be used by malware to call home and were not included in the test. Further research should be done to detect malware call-home activities using those techniques.

## 5. Conclusion

This approach forwards otherwise dropped traffic to a sinkhole server to detect malware call-home activity in a proxy environment. It takes advantage of the fact that the ratio of malicious traffic among the traffic dropped by the gateway firewall is usually much higher than the one through proxy. Just like in a complex building an intruder has much higher possibility to hit some prohibited area than an internal employee who are familiar with the environment.

The test confirmed this approach worked as expected, and it could provide many pieces of useful information to identify the traffic as suspicious or not. The test gives more information and insight into the attempted outgoing requests for security incident detection and response purpose. However there are still some challenges and limitations, e.g. sometimes it is still hard to tell if a request is legitimate even if we could see it.

Though "modern malware is slowly becoming proxy aware" (Tom, 2011), there are still a big percentage of malware that is not. If implemented appropriately, this approach could be very helpful for an organization to catch malware call-home activities on the network.

## 6. References

Ellinger, Maik. *FTP (WU-FTPD) Honeypot-Script simulates a few functions of a old WU-FTPD 2.6.0* Retrieved June 12, 2013, from Honeyd Web site: http://www.honeyd.org/contrib.php

Tyler Cui, tianqiang.cui@gmail.com

Ellinger, Maik. *SMTP (Sendmail) Honeypot-Script simulates a few functions of a Sendmail 8.12.2 Server* Retrieved June 12, 2013, from Honeyd Web site: http://www.honeyd.org/contrib.php

ModSecurity. *Logging.* Retrieved from http://www.modsecurity.org/documentation/modsecurity-apache/1.9.3/html-multipage/07-logging.html

Dest-unreach.org. *Socat: Multipurpose relay.* Retrieved from http://www.dest-unreach.org/socat/

Inside Your Botnet. *Malware samples.* Retrieved from http://www.exposedbotnets.com/

Malwr. *Malware samples.* Retrieved from https://malwr.com/

Tom, B (2011). *Proxy Authentication*. Retrieved from http://www.digitalboundary.net/wp/?p=347

Greg, L. Bassett & Jeff, Boerio (2009). *Getting Ahead of Malware*. Retrieved from http://www.intel.com.au/content/dam/doc/white-paper/intel-it-enterprise-security-malware-paper.pdf

RFC2616. *Hypertext Transfer Protocol -- HTTP/1.1*. Retrieved from http://www.w3.org/Protocols/rfc2616/rfc2616.html

## Appendix: relevant scripts and files

*1. index.php* - The page that all requests will be redirected to and displays alert information:

```
<?php
echo "<h1>Sorry</h1>";
echo date("Y-m-d H:i:s") . " : ";
echo "Your browser config is not correct, please contact your system admin";
?>
```

Tyler Cui, tianqiang.cui@gmail.com

*2. ftp.sh* - The script that simulates FTP service to interact with clients and logs request details:

```
#!/bin/bash

DATE=`date`
log=./ftp-$1.log
AUTH="no"
PASS="no"

echo "220 FTP server ready."
while read incmd parm1 parm2 parm3 parm4 parm5
do
        # remove control-characters
        incmd=`echo $incmd | sed s/[[:cntrl:]]//g`
        parm1=`echo $parm1 | sed s/[[:cntrl:]]//g`
        parm2=`echo $parm2 | sed s/[[:cntrl:]]//g`
        parm3=`echo $parm3 | sed s/[[:cntrl:]]//g`
        parm4=`echo $parm4 | sed s/[[:cntrl:]]//g`
        parm5=`echo $parm5 | sed s/[[:cntrl:]]//g`

        # convert to upper-case
        incmd_nocase=`echo $incmd | gawk '{print toupper($0);}'`

        if [ $AUTH = "no" ]
        then
                if [ "$incmd_nocase" != "USER" ]
                then
                        if [ "$incmd_nocase" != "QUIT" ]
                        then
                                echo "530 Please login with USER and PASS."
                                continue
                        fi
                fi
        fi

        case $incmd_nocase in
          QUIT* )
                echo "221 Goodbye."
                exit 0;;
           SYST* )
                echo "215 UNIX Type: L8"
                ;;
           HELP* )
                echo "214-The following commands are recognized (* =>'s unimplemented)."
                echo "   USER   PORT   STOR   MSAM*  RNTO   NLST   MKD    CDUP"
                echo "   PASS   PASV   APPE   MRSQ*  ABOR   SITE   XMKD   XCUP"
                echo "   ACCT*  TYPE   MLFL*  MRCP*  DELE   SYST   RMD    STOU"
                echo "   SMNT*  STRU   MAIL*  ALLO   CWD    STAT   XRMD   SIZE"
                echo "   REIN*  MODE   MSND*  REST   XCWD   HELP   PWD    MDTM"
                echo "   QUIT   RETR   MSOM*  RNFR   LIST   NOOP   XPWD"
                echo "214 Direct comments to ftp@$domain."
                ;;
           USER* )
                parm1_nocase=`echo $parm1 | gawk '{print toupper($0);}'`
                if [ "$parm1_nocase" = "ANONYMOUS" ]
                then
```

```
                echo "331 Guest login ok, send your complete e-mail address as a password."
        AUTH="ANONYMOUS"
                else
                  echo "331 Password required for $parm1"
        AUTH=$parm1
                fi
                ;;
         PASS* )
                PASS=$parm1
                if [ "$AUTH" = "ANONYMOUS" ]
                then
                   echo "230-Hello User at $1"
                   echo "230 Guest login ok, access restrictions apply."
                else
                        if [ ! -z "$PASS" ]
                        then
                                echo "230 Login successful."
                        else
                                echo "530 Login incorrect."
                        fi
                fi
                ;;
         MKD* )
                # choose :
                echo "257 \"$parm1\" new directory created."
                #echo "550 $parm1: Permission denied."
                ;;
         CWD* )
                # choose :
                echo "250 CWD command successful."
                # echo "550 $parm1: No such file or directory."
                ;;
         NOOP* )
                echo "200 NOOP command successful."
                ;;
         PORT* )
                echo "200 PORT command successful."
                ;;
         TYPE* )
                echo "200 Type set to I/A."
                ;;
         PASV* )
                echo "227 Entering Passive Mode (10,0,0,100,100,53)"
                ;;
         ACCT* )
                echo "502 $incmd command not implemented."
                ;;
         SITE* )
                echo "200 command successful."
                ;;
          * )
                echo "500 '$incmd': command not understood."
                ;;
        esac
        thedate=`date +"%Y-%m-%d %H:%M:%S"`
        echo "$thedate $SOCAT_PEERADDR $SOCAT_PEERPORT $incmd $parm1 $parm2 $parm3 $parm4
        $parm5" >> $log
done
```

*3. smtp.sh* - The script that simulates SMTP service to interact with clients and logs request details:

```
#!/bin/bash

DATE=`date`
host=`hostname`
domain=`dnsdomainname`
log=./smtp-$1.log
MAILFROM="err"
EHELO="no"
RCPTTO="err"
echo "220 ESMTP Sendmail"
while read incmd parm1 parm2 parm3 parm4 parm5
do

        # default to log commands
        log_cmd='yes'

        # remove control-characters
        incmd=`echo $incmd | sed s/[[:cntrl:]]//g`
        parm1=`echo $parm1 | sed s/[[:cntrl:]]//g`
        parm2=`echo $parm2 | sed s/[[:cntrl:]]//g`
        parm3=`echo $parm3 | sed s/[[:cntrl:]]//g`
        parm4=`echo $parm4 | sed s/[[:cntrl:]]//g`
        parm5=`echo $parm5 | sed s/[[:cntrl:]]//g`

        # convert to upper-case
        incmd_nocase=`echo $incmd | gawk '{print toupper($0);}'`
        #echo $incmd_nocase
        case $incmd_nocase in
          QUIT* )
                echo "220 2.0.0 closing connection"
        exit 0;;
          RSET* )
                echo "250 2.0.0 Reset state"
                ;;
          HELP* )
                echo "214-2.0.0 This is sendmail"
                echo "214-2.0.0 Topics:"
                echo "214-2.0.0       HELO    EHLO    MAIL    RCPT    DATA"
                echo "214-2.0.0       RSET    NOOP    QUIT    HELP    VRFY"
                echo "214-2.0.0       EXPN    VERB    ETRN    DSN     AUTH"
                echo "214-2.0.0       STARTTLS"
                echo "214-2.0.0 For more info use \"HELP <topic>\"."
                echo "214-2.0.0 To report bugs in the implementation send email to"
                echo "214-2.0.0       sendmail-bugs@sendmail.org."
                echo "214-2.0.0 For local information send email to Postmaster at your site."
                echo "214 2.0.0 End of HELP info"
                ;;
          HELO* )
                if [ -n "$parm1" ]
                then
                  EHELO="ok"
                  echo "250 Hello, pleased to meet you"
                else
                  echo "501 5.0.0 HELO requires domain address"
                fi
                ;;
```

```
EHLO* )
        if [ -n "$parm1" ]
        then
          EHELO="ok"
          echo "250-excellent"
          echo "250-AUTH PLAIN LOGIN"
          echo "250 HELP"
        else
          echo "501 5.0.0 EHLO requires domain address"
        fi
        ;;
MAIL* )
        haveFROM=`echo $parm1 | gawk '{print toupper($0);}'`
        if [ "$haveFROM" == "FROM:" ]
        then
                        if [ -n "$parm2" ]
                        then
                           MAILFROM="ok"
                           echo "250 2.1.0 $parm2 $parm3 $parm4... Sender ok"
                        else
                           echo "501 5.5.2 Syntax error in parameters scanning \"$parm2\""
                           MAILFROM="err"
                        fi
        else
          echo "501 5.5.2 Syntax error in parameters scanning \"\""
        fi
        ;;
RCPT* )
        #echo $MAILFROM
        if [ "$MAILFROM" == "ok"  ]
        then
                haveTO=`echo $parm1 | gawk '{print toupper($0);}'`
                if [ "$haveTO" == "TO:"  ]
                then
                        if [ -n "$parm2" ]
                        then
                           RCPTTO="ok"
                           # echo "553 sorry, that domain isn't in my list of allowed rcpthosts (#6.7.1)"
                           echo "250 2.1.0 $parm2 $parm3 $parm4... Recipient ok"
                        else
                           echo "501 5.5.2 Syntax error in parameters scanning \"\""
                           RCPTTO="err"
                        fi
                fi
        else
          echo "503 5.0.0 Need MAIL before RCPT"
        fi
        ;;
STARTTLS* )
        echo "454 4.3.3 TLS not available after start"
        ;;
NOOP* )
        echo "250 2.0.0 OK"
        ;;
STARTTLS* )
        echo "454 4.3.3 TLS not available after start"
        ;;
NOOP* )
        echo "250 2.0.0 OK"
        ;;
DATA* )
        echo "354 OK"
```

```
                ;;
        FROM* )
                ;;
        TO* )
                ;;
        SUBJECT* )
                ;;
        CC* )
                ;;
        BCC* )
                ;;
        AUTH* )
                echo "503 AUTH mechanism not available"
                ;;
        * )
                echo "500 5.5.1 Command unrecognized: \"$incmd\""
                log_cmd='no'
                ;;
    esac
    if [ $log_cmd == 'yes' ]
    then
                thedate=`date +"%Y-%m-%d %H:%M:%S"`
                echo "$thedate $SOCAT_PEERADDR $SOCAT_PEERPORT $incmd $parm1 $parm2 $parm3
    $parm4 $parm5" >> $log
    fi
done
```

*4. irc.sh* - The script that listens for IRC traffic and log request details:

```
#!/bin/bash

firstdate=`date +"%Y-%m-%d %H:%M:%S"`
printf "%s %s %s %s\n" "$firstdate" "$SOCAT_PEERADDR" "$SOCAT_PEERPORT" "TCP connection established"
        >> irc.txt

while read line; do
        thedate=`date +"%Y-%m-%d %H:%M:%S"`
        printf "%s %s %s %s\n" "$thedate" "$SOCAT_PEERADDR" "$SOCAT_PEERPORT" "$line" >> irc.txt
done
```

*5. audit.sh* - The script to count and sort the fields of interest (Host, User-Agent, URL) of HTTP/S logs:

```
#!/bin/bash
DIR='/var/log/apache2'

echo "Checking Host name..."
gawk '$1 ~ /^Host:/ {print $2}' $DIR/modsec_audit.log|sort|uniq -c|sort -rn > hostname.txt
./f.pl hostname.txt
echo "done"
echo ""

echo "Checking User Agent..."
gawk -F ': ' '$1 ~ /^User-Agent/ {print $2}' $DIR/modsec_audit.log|sort|uniq -c|sort -rn > useragent.txt
./f.pl useragent.txt
echo "done"
echo ""
```

```
echo "Checking URL requested..."
gawk '$1 ~ /^GET|^POST/ {print $2}' $DIR/modsec_audit.log|sort|uniq -c|sort -rn > url.txt
./f.pl url.txt
echo "done"
```

*6. f.pl* - The script to filter out the items listed in the whitelist files:

```
#!/usr/bin/perl

use strict;

my $file = $ARGV[0];

open R, "< $file" or die $!;
my @result;
while (my $e = <R>) {
        if($e !~ /^#/ && length $e > 1) {
                push @result, $e;
        }
}
close R;

open(FIL, "< fil_${file}") or die $!;
while ( my $f = <FIL>) {
        chomp $f;
        @result = grep (!/^\s+\d+\s+$f/, @result);
}
print "@result\n";
close(FIL);
```

*7. fil_hostname.txt* - The filter file that lists hostnames which could be whitelisted (Host header):

```
# syntax for hostname filter:
# it's strongly recommended to use "$" at the end of each entry..
*\.windowsupdate\.com$
.*\.ibm\.com$
.*\.google\.com$
.*\.msftncsi\.com$
.*\.hotmail\.com$
.*\.dell\.com$
```

*8. fil_url.txt* - The filter file that lists URLs which could be whitelisted (URL requested):

```
# syntax for url filter:
# the lines can not start with "^"; but if it is the end of line, "$" should be used.
\/msdownload\/update\/v3\/static\/trustedr\/en\/
\/favicon\.ico$
\/$
\/ncsi\.txt$
```

*9. abnormal_scan.pl* - The script to check if the request doesn't follow HTTP protocol

```
#!/usr/bin/perl
```

```
use strict;
use Switch;

my $dir = '/var/log/apache2';
my $sn;
my $flag_a;
my $flag_b;
my $flag_c;
my $client_ip;
my $remote_host;

open(AB, "> request_abnormal.txt") or die($!);
open(FILE, "< $dir/modsec_audit.log") or die($!);
foreach my $line (<FILE>) {
        chomp $line;
        if($line =~ /--(\w{8})-(\w)--/) {
                $sn = $1;
                switch($2) {
                        case "A" { $flag_a = 'y'; $flag_b = ''; $flag_c = '';  }
                        case "B" { $flag_a = ''; $flag_b = 'y'; $flag_c = ''; }
                        case "C" { $flag_a = ''; $flag_b = ''; $flag_c = 'y'; }
                        case "Z" { $flag_a = ''; $flag_b = ''; $flag_c = ''; $sn = ''; $client_ip = ''; $remote_host = '';
        }
                        else { print "error!\n"; exit; }
                }
        } else {
                if(length $flag_b && length $sn) {
                        if(length $line && $line =~ /^Host:\s+(.*)$/) {
                                $remote_host = $1;
                        }
                        if(length $line && $line !~ /GET|POST|\w+:\s/) {
                                if( length $remote_host == 0 ) {
                                        $remote_host = 'unknown';
                                }
                                print AB "$client_ip -> $remote_host $line\n";
                        }
                } elsif(length $flag_a && length $sn) {
                        if( $line =~ /\].*?(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s+/ ) {
                                $client_ip = $1;
                        }
                }
        }
}
close(FILE);
close(AB);
print "\n\n----Abnormal Request----\n\n";
system("cat request_abnormal.txt|sort|uniq -c|sort -rn");
```

Tyler Cui, tianqiang.cui@gmail.com