GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

Practical OSSEC

GIAC (GCIH) Gold Certification

Author: Chad Robertson, chadrober@gmail.com
Advisor: Egan Hadsell

Accepted: July 5, 2011

Abstract

OSSEC is a simple to install host-based intrusion detection system. The difficulty is in tuning the installation so that the resulting alerts are pertinent to the environment. Agents can be installed on a variety of systems; Web servers, mail servers, VMWare servers, WAFs. All of these server types likely produce logs with very different syntax. Each log type requires custom decoders and rules to be created for OSSEC to alert appropriately if none are included by default or found within the community. Resulting alerts must be ranked by level of criticality based on not only one single log event but possibly the presence of other events occurring during small windows of time. All of this must result in notification being sent to the appropriate party and at the appropriate level to allow them to respond to the incident. This paper will briefly discuss installing OSSEC agents on both Windows and Linux systems. It will then explore how to configure rulesets and decoders for various commonly found enterprise servers. Finally, it will describe the process of tuning rulesets so that the resulting alerts are both valuable and pertinent.

© 2011 SANS Institute, Author retains full rights.

© 2011 The SANS Institute Author retains full rights.

1. Introduction

"OSSEC is an Open Source Host-based Intrusion Detection System. It performs log analysis, file integrity checking, policy monitoring, rootkit detection, real-time alerting and active response" (Trend Micro, 2010). Large organizations with equally large budgets might utilize a SIEM ("Security Information and Event Management") or STRM ("Security Threat Response Management") to accomplish these tasks (Swift, 2006). OSSEC provides similar functionality and because it is open source allows access even if budget is an issue. Also, if a SIEM / STRM is present OSSEC will integrate into it to provide additional benefit. As such, OSSEC is a solid alternative solution or a valuable complement to protect resources and still satisfy compliance requirements such as PCI-DSS (Third Brigade Inc., 2009)

Leveraging OSSEC to produce meaningful results across diverse systems can be difficult. Disparate enterprise applications oftentimes do not provide a similar mechanism for logging events (Xavier, 2008). To utilize OSSEC, applications must first be identified or configured to provide a common logging supported format. OSSEC only supports text-based logs with the exception of the Windows EventLog (Cid & Ozturk, 2006).

After identifying supported applications there remains the challenge of tuning decoders within OSSEC to parse those logs for useful data. OSSEC provides a number of decoders to users by default, but if a decoder for a particular application does not already exist one must be created using delimiters that exist within the logs (Klein, 2009). Those applications that do not, by default, create logs with clear delineation will result in convolution and must be reconfigured or require the logs be reformatted by external means such as sed or awk (Bhatia, 2010).

Once the logs have been parsed and the relevant data identified then the rules within OSSEC must be tuned to prevent deluging administrators with irrelevant alerts. Tuning OSSEC includes customizing rulesets and creating local rule overrides to existing signatures (Cid D. B., 2010).

Chad Robertson, chadrober@gmail.com

© 2011 The SANS Institute Author retains full rights.

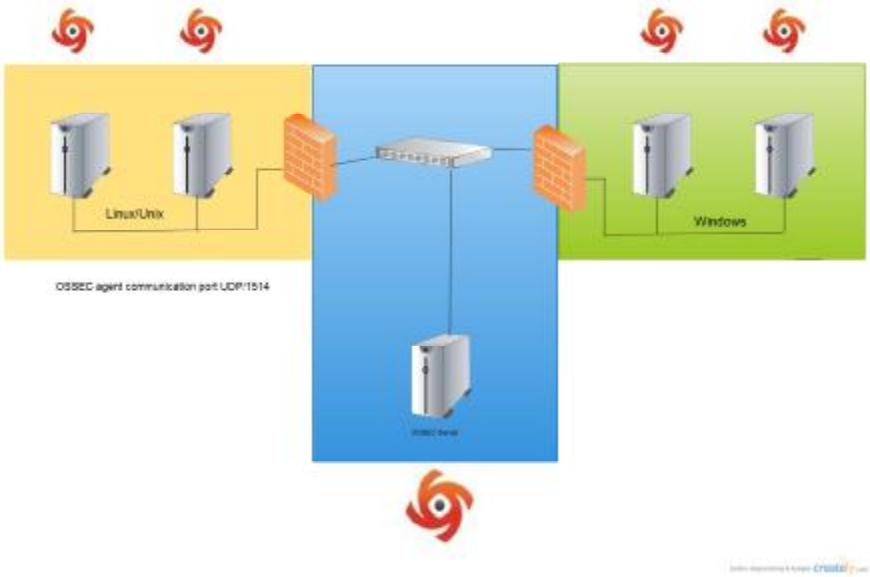
This paper will briefly cover how OSSEC is architected (agents and servers) within a Windows and Linux environment. Then it will explain how to configure OSSEC to monitor a variety of both default and custom log sources. It will then take a look at the results of the default OSSEC configuration on reviewing those logs. Next it will demonstrate how to tune OSSEC to provide more useful data to administrators. Finally, it will show how to create signatures based on dynamic input.

2. Architecting a OSSEC environment

Before installing OSSEC it's important to carefully consider each system that will become an agent to assess what OSSEC functions are necessary. It's also important to review where the server will be located to access any need for additional firewall access between differing security level environments.

2.1. System requirements

OSSEC utilizes a client / server architecture. Communication occurs on UDP port 1514 and is encrypted using the symmetric key Blowfish algorithm (Brenton, 2010). Monitoring agents may be installed on a variety of operating systems, but the management server to which the agents communicate with must be installed on a BSD/Linux/Unix system (Hay, Cid, & Bray, 2008).



Chad Robertson, chadrober@gmail.com

© 2011 The SANS Institute Author retains full rights.

2.2. Server Roles

It's important to consider the server's role within the organization when designing an OSSEC environment. If your organization must abide by Payment Card Industry Digital Security Standards (henceforth PCI-DSS) then you are mandated to one primary function per server (PCI Security Standards Council, 2010). Each server may generate unique log files that may require custom configuration to be monitored by OSSEC.

There may be situations where the server's role is such that it is unnecessary to monitor some aspects of its functionality. For example, if the agent is running on a web server you may not require non-web-related logs to be monitored. OSSEC provides the ability to ignore, both recursively and directly, any monitored file.

2.3. Testing

When customizing OSSEC it is useful to test the customizations to verify the fields are being identified correctly. OSSEC comes with a tool that provides that functionality. The testing binary "ossec-logtest" exists within the /bin directory and can be ran without any parameters being passed to it. It starts in interactive mode and allows pasting in strings and then responds with how that string would be interpreted by OSSEC.

The following log is generated when a network interface card enters promiscuous mode.

```
Feb  7 12:27:11 TEST-SERVER kernel: [14544.590716] device eth1 entered promiscuous mode
```


© 2011 The SANS Institute Author retains full rights.

When that log is tested against OSSEC, the following is shown:

```
user@TEST-SERVER:~# /var/ossec/bin/ossec-logtest
2011/06/22 15:31:18 ossec-testrule: INFO: Reading local decoder file.
2011/06/22 15:31:18 ossec-testrule: INFO: Started (pid: 21550).
ossec-testrule: Type one log per line.

Feb  7 12:27:11 TEST-SERVER kernel: [14544.590716] device eth1 entered
promiscuous mode


**Phase 1: Completed pre-decoding.
     full event: 'Feb  7 12:27:11 TEST-SERVER kernel: [14544.590716] device
eth1 entered promiscuous mode'
     hostname: 'TEST-SERVER'
     program_name: 'kernel'
     log: '[14544.590716] device eth1 entered promiscuous mode'

**Phase 2: Completed decoding.
     decoder: 'iptables'

**Phase 3: Completed filtering (rules).
     Rule id: '5104'
     Level: '8'
     Description: 'Interface entered in promiscuous(sniffing) mode.'
**Alert to be generated.
```

From the example above it can be seen that the log is decoded as "iptables" and a rule triggered. This method of testing can help track down errors in customized rules without affecting the running processes.

2.4. Unknown log errors

If OSSEC receives a log that it doesn't know how to decode it will generate an event 1002 – "Unknown problem somewhere on the system." When OSSEC receives new logs (or a few hundred of them) that it does not understand it will send the administrator with these alerts. The solution is to configure a minimal decoder to identify a unique field within the log so that OSSEC no longer considers the log unknown. (see the first decoder shown in the section 5.2 below)

© 2011 The SANS Institute Author retains full rights.

3. Decoders

OSSEC must first understand what is in a log before it can determine if an alert is required. It accomplishes this by parsing the log and normalizing the data contained therein using default and custom decoders. Decoders contain parameters that match syntax within logfiles to be forwarded to the rules for processing.

3.1. Default Decoders

OSSEC comes with an assortment of decoders by default. These decoders can parse a wide array of logging sources such as Apache, SSH, and Windows event viewer. They provide basic HIDS functionality to an assortment of applications.

Within this sections the SSH decoder will be reviewed and the XML tags contained within it will be defined. This section isn't designed to provide the reader with comprehensive knowledge OSSEC decoder formats. Instead, it will describe how the default tags within the SSH decoder find and parameterize information for later processing by the rules.

© 2011 SANS Institute, Author retains full rights.

© 2011 The SANS Institute Author retains full rights.

3.1.1. SSH decoder

Let's take a closer look at how a OSSEC decoder works. Below is the major portions of the SSH decoder that ships with OSSEC (as of version 2.5.1).

```
<decoder name="sshd">
 <program_name>^sshd</program_name>
</decoder>

<decoder name="sshd-success">
 <parent>sshd</parent>
 <prematch>^Accepted</prematch>
 <regex offset="after_prematch">^ \S+ for (\S+) from (\S+) port </regex>
 <order>user, srcip</order>
 <fts>name, user, location</fts>
</decoder>

<decoder name="ssh-denied">
 <parent>sshd</parent>
 <prematch>^User \S+ from </prematch>
 <regex offset="after_parent">^User (\S+) from (\S+) </regex>
 <order>user, srcip</order>
</decoder>

<decoder name="ssh-failed">
 <parent>sshd</parent>
 <prematch>^Failed \S+ </prematch>
 <regex offset="after_prematch">^for (\S+) from (\S+) port \d+ \w+$</regex>
 <order>user, srcip</order>
</decoder>

<decoder name="ssh-error">
 <parent>sshd</parent>
 <prematch>^error: PAM: Authentication \w+ </prematch>
 <regex offset="after_prematch">^for (\S+) from (\S+)$</regex>
 <order>user, srcip</order>
</decoder>

<decoder name="ssh-invalid-user">
 <parent>sshd</parent>
 <prematch>^Invalid user|^Illegal user</prematch>
 <regex offset="after_prematch"> from (\S+)$</regex>
 <order>srcip</order>
</decoder>
```

First, basic terminology needs to be explained. The decoders within OSSEC are written in XML format. They are organized into related blocks making them easier to understand. As is common to XML formats, each section is be opened, then defined, and then closed. For more information on the XML format please see Appendix 1.

© 2011 The SANS Institute Author retains full rights.

<parent> tag

The code shown begins by defining the name of the decoder and the program in which it is associated. This definition is then used as a parent to subsequent sections within the decoder block. The <parent> tag is used to link a subordinate codeblock to its parent. As you can see within Figure X, "ssh-success" is linked to the "sshd" decoder by the parent tag.

<prematch> tag

The prematch tag attempts to find a match within the log for the string defined. In the case of sshd-success, the regular expression (for more information about regular expressions see appendix 2) the word "Accepted" is looked for at the beginning of the string.

<offset> tag

This tag is used to speed up processing by requiring a match to have already taken place before trying to match additional text. In the case of this decoder, the regex type of the offset tag is utilized to match text after the prematch tag has been matched. So, in this example, if and only if the string begins "Accepted" is the string matched for the offset regex.

Parenthesis ()

Parenthesis are used within OSSEC decoders to define something that OSSEC needs to retain and pass to the rules portion of the process. Within the sshd-success decoder, the first \S+ in the line is not retained. The thing that the decoder expects to see is the word "for" which is also not retained. If the string has matched what is expected up until now then the next \S+ should be the user and is necessary for rules processing later. It must therefore be retained and is thus surrounded by parenthesis. The next word, "from" is expected but unnecessary. The next \S+ is expected to be the source IP address and is required for later rules processing. It is therefore surrounded by parenthesis and retained by the engine.

© 2011 The SANS Institute Author retains full rights.

<order> tag

The order tag defines what the previous parenthesis groups contain and the order in which they were received. In our example, two pieces of the alert were retained by the engine; the username used when the login occurred and the IP address from which the login took place. The order tag labels those strings as "user" and srcip so that OSSEC understands what the strings being retained represent.

<fts> tag

The fts tag is used to designate a decoder as one in which the first time it matches the administrator would like to be alerted. In the sshd-success, the administrator will be alerted the first time the decoder matches either the user or location.

4. Rules

After an OSSEC decoder has identified the important details within the log string, the next step is for it to use the configured rules to determine if an alert should be generated.

4.1. Default Rules

OSSEC comes with an extensive ruleset by default. Because of that OSSEC can detect a great many potential incidents out of the box. However, the large number of rules also cause OSSEC to produce an equally large number of false positives.

Below, this paper will take a closer look at a sampling of rules associated with the previously examined decoder.

4.1.1. SSH Rules

Following the above SSHd decoder, below are three rules. Each rule shown is related to the other and that relationship will be defined. As before, I will note the rules and then define the XML tags and how they interrelate.

© 2011 The SANS Institute Author retains full rights.

© 2011 SANS Institute, Author retains full rights.

```
<group name="syslog,sshd,">
 <rule id="5700" level="0" noalert="1">
  <decoded_as>sshd</decoded_as>
  <description>SSHD messages grouped.</description>
 </rule>

 <rule id="5710" level="5">
  <if_sid>5700</if_sid>
  <match>illegal user|invalid user</match>
  <description>Attempt to login using a non-existent
user</description>
  <group>invalid_login,authentication_failed,</group>
 </rule>

  <rule id="5712" level="10" frequency="6" timeframe="120"
ignore="60">
  <if_matched_sid>5710</if_matched_sid>
  <description>SSHD brute force trying to get access to
</description>
  <description>the system.</description>
  <same_source_ip />
  <group>authentication_failures,</group>
 </rule>
```

<group> tag

The group tag, as you might suspect, creates a logical grouping of rules. This is for use within the rules configuration file only.

<rule id> tag

This tag associates a number to the rule. The numeric range of included rules is from 00000 to 99,999. Custom rules should range from 100,000 to 119,999. If you chose another ID for a custom rule your rule might conflict with the default rules (Cid & Ozturk, ossec.net, 2006). See **appendix 3** for a list of rule ranges and their category.

© 2011 The SANS Institute Author retains full rights.

The rule ID tag is also where the level of the rule is defined. Levels note the criticality of a rule and act as a threshold for various reporting and alerting functions. For example, one might define that only level 14 and up alerts should generate an email, while anything below that threshold is silently ignored.

The rule id tag is also where the noalert option can be included. This prevents alerts from being generated for that particular rule. This is particularly useful for testing custom rules or temporarily disabling rules that need additional tuning.

The frequency option is used to count the number of occurrences of a particular rule matching. It is always seen with the timeframe option since the number of occurrences must be contained within a set timeframe.

The timeframe option provides a mechanism to limit the time in which a certain number of log entried can match a particular rule. In the example above the rule will only match the 6th occurrence of the trigger within the 120 second timeframe.

<decoded as> tag

The decoded as tag allows OSSEC to know which decoder to associate the rule. In the example above the rules are being associated with the sshd decoder. This means that all logs decoded as sshd will fall within the scope of these rules.

<description> tag

The description tag provides a space for the rule author to note the purpose of the rule. The text contained therein will be included in the event alert if one is generated.

<if_sid> tag

The if_sid tag creates a dependency on a previously defined rule. In the example above, the if_sid tag makes rule id 5710 dependent upon rule 5700. If rule 5700 was not triggered then rule 5710 will be ignored. If rule id 5700 was triggered then any rule marked as dependent will be within scope.

© 2011 The SANS Institute Author retains full rights.

<match> tag

The match tag acts as a simple text match . If the log string contains the text noted within the match tag then the rule will be triggered. In this example, if the log contains either "illegal user" OR "invalid user" (because of the pipe character (|), See appendix 2) the rule will be triggered.

<group> tag

The group tag is used to define a subgroup within the rule. See **appendix 4** for a complete list of available groups. There are other places within OSSEC that one might wish to reference a specific group of rules. For example, one might wish to create active response rules for a particular group of rules (Dave, 2010).

<if_matched_sid> tag

This associates the current rule with a previously defined one. In the case of the rules shown above, the rule 5712 is associated with rule 5710. This nesting of rules is necessary so that dependent conditions can trigger an event. In the example above, if the illegal user rule is triggered once the resulting response level is 5, however, if that rule is triggered 6 times in 2 minutes, then rule 5712 is triggered with a level of 10. While a level 5 event may be considered benign, the level 10 event is much more concerning.

5. Custom rules and decoders

The previous section explored the XML components of default rules and decoders. This section will look at a log generated by an application currently not supported within OSSEC and then detail how to write a decoder for the application and rules to alert based on decoded information.

5.1. Application – WS_FTP

WS_FTP is a Windows based file transfer server solution. It provides the ability to transfer files over FTP, SSL, SSH, and HTTP/S transfer protocols (Ipswitch, 2010). WS_FTP is used by more than 40 million customers transferring billions of files every week (Ipswitch, 2010). The widespread use of WS_FTP OSSEC's lack of native support makes it an ideal candidate for custom rules and decoders.

© 2011 The SANS Institute Author retains full rights.

Any external facing FTP server should be secured from intrusion and have mechanisms in place to review the log files produced. The logs files produced by WS_FTP are text based which make it easy to integrate into OSSEC. Below is a sample log generated by WS_FTP. The log file shows a remote user attempting to login as an anonymous user.

Jun 20 15:42:17 FTP-SERVER SSH: Sent server version: SSH-2.0-WS_FTP-SSH <SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251>

Jun 20 15:42:17 FTP-SERVER SSH: Connection established <SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251>

Jun 20 15:42:17 FTP-SERVER SSH: Received client version: SSH-2.0-PuTTY_Local:_Nov_21_2010_15:53:55 <SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251>

Jun 20 15:42:17 FTP-SERVER SSH: Began Key Exchange <SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251>

Jun 20 15:42:19 FTP-SERVER SSH: Completed Key Exchange. New keys in place <SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251>

Jun 20 15:42:19 FTP-SERVER SSH: No User. Possible reasons: Invalid username, invalid license, error while accessing user database <SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251, User=anonymous>

Jun 20 15:42:20 FTP-SERVER SFTP: Invalid User <Host= FTP-SERVER, User=anonymous><Command=NOTIFICATION, Parameters=default>

Jun 20 15:42:20 FTP-SERVER SSH: No User. Possible reasons: Invalid username, invalid license, error while accessing user database <SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251, User=anonymous>

Jun 20 15:42:20 FTP-SERVER SFTP: Invalid User <Host=FTP-SERVER, User=anonymous><Command=NOTIFICATION, Parameters=default>

Jun 20 15:42:20 FTP-SERVER SSH: Connection closed <SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251>

Jun 20 15:42:20 FTP-SERVER SSH: Close Transport <SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251>

© 2011 The SANS Institute Author retains full rights.

Anonymous and brute-force FTP login attempts are a common occurrence on externally facing FTP servers. An administrator should be concerned with and alerted to suspect authentication attempts that meet certain criteria. The method used to write OSSEC rules to alert both to anonymous login attempts and to login attempts that pass a certain threshold will be demonstrated below.

5.2. WS_FTP Decoder

First, some unique portion of the log must be identified. This unique token must be used to identify the log by an OSSEC decoder. It is often easiest to identify a field to define within a decoder by dumping the log into ossec-test and see how it interprets it by default. At times the log syntax will conflict with existing customizations and those conflicts must be considered.

To identify this log format the decoder shown below was created:

```
<decoder name="WS-FTP">
 <program_name>SSH|SFTP</program_name>
</decoder>
```

Testing this decoder results in the following:

```
user@TEST-SERVER:~# /var/ossec/bin/ossec-logtest
2011/06/22 16:43:24 ossec-testrule: INFO: Reading local decoder file.
2011/06/22 16:43:24 ossec-testrule: INFO: Started (pid: 21685).
ossec-testrule: Type one log per line.

Jun 20 15:42:17 FTP-SERVER SSH: Sent server version: SSH-2.0-WS_FTP-SSH
<SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251>

**Phase 1: Completed pre-decoding.
    full event: 'Jun 20 15:42:17 FTP-SERVER SSH: Connection established
<SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251>'
    hostname: 'FTP-SERVER'
    program_name: 'SSH'
    log: 'Connection established <SessionID=8416817, Listener=192.168.0.2:22,
Client=217.25.214.12:58251>'

**Phase 2: Completed decoding.
    decoder: 'WS-FTP'
```


© 2011 The SANS Institute Author retains full rights.

Now that the logging messages are being identified (grouped), OSSEC need to know what parts of the log are relevant. To accomplish that we write the two additional decoders shown below:

```
<decoder name="WS-FTP-Client-Connected">
  <parent>WS-FTP</parent>
  <prematch>Connection established</prematch>
  <regex offset="after_prematch">Client=(\d+.\d+.\d+.\d+)</regex>
  <order>srcip</order>
</decoder>


<decoder name="WS-FTP-User">
  <parent>WS-FTP</parent>
  <prematch>Invalid User</prematch>
  <regex offset="after_prematch">User=(\S+)></regex>
  <order>user</order>
</decoder>
```

The test below demonstrates the "WS-FTP-Client-Connected" decoder capturing the IP addess from the log.

```
user@TEST-SERVER:~# /var/ossec/bin/ossec-logtest
2011/06/22 19:18:50 ossec-testrule: INFO: Reading local decoder file.
2011/06/22 19:18:50 ossec-testrule: INFO: Started (pid: 21982).
ossec-testrule: Type one log per line.

Jun 20 15:42:17 FTP-SERVER SSH: Connection established <SessionID=8416817,
Listener=192.168.0.2:22, Client=217.25.214.12:58251>


**Phase 1: Completed pre-decoding.
    full event: 'Jun 20 15:42:17 FTP-SERVER SSH: Connection established
<SessionID=8416817, Listener=192.168.0.2:22, Client=217.25.214.12:58251>'
    hostname: 'FTP-SERVER'
    program_name: 'SSH'
    log: 'Connection established <SessionID=8416817, Listener=192.168.0.2:22,
Client=217.25.214.12:58251>'

**Phase 2: Completed decoding.
    decoder: 'WS-FTP'
    srcip: '217.25.214.12'
```

Chad Robertson, chadrober@gmail.com

© 2011 The SANS Institute Author retains full rights.

The test has shown that the source IP (srcip) has been identified correctly. Next, the "WS-FTP-User" decoder must be tested:

```
user@ TEST-SERVER:~# /var/ossec/bin/ossec-logtest
2011/06/22 19:20:44 ossec-testrule: INFO: Reading local decoder file.
2011/06/22 19:20:44 ossec-testrule: INFO: Started (pid: 22487).
ossec-testrule: Type one log per line.

Jun 20 15:42:20 FTP-SERVER SFTP: Invalid User <Host=FTP-SERVER,
User=anonymous><Command=NOTIFICATION, Parameters=default>


**Phase 1: Completed pre-decoding.
    full event: 'Jun 20 15:42:20 FTP-SERVER SFTP: Invalid User <Host= FTP-SERVER,
User=anonymous><Command=NOTIFICATION, Parameters=default>'
    hostname: 'FTP-SERVER'
    program_name: 'SFTP'
    log: 'Invalid User <Host=FTP-SERVER
User=anonymous><Command=NOTIFICATION, Parameters=default>'

**Phase 2: Completed decoding.
    decoder: 'WS-FTP'
    dstuser: 'anonymous'
```

Now that both decoders have been verified to work as expected, we must write rules to utilize the newly decoded fields.

5.3. WS_FTP Rules

The relevant fields within the log have been identified by the decoder. Now rules must be created that alert when those captured values match specific patterns. The three rules shown here will alert to two very common events that occur on an FTP server. The first rule will group the subsequent rules similar to the method used within the custom decoder aboe. The second rule will trigger when someone attempts to log into the server using invalid credentials. The final rule will depend upon the previous and only trigger after the previous rule has triggered six times.

© 2011 The SANS Institute Author retains full rights.

The first rule is shown below.

```
<group name="local,syslog,">

 <rule id="100066" level="0" noalert="1">
  <decoded_as>WS-FTP</decoded_as>
  <description>WS-FTP messages grouped.</description>
 </rule>
```

This rule is intended to group the WS_FTP rules together. It also functions to identify all logs belonging to WS_FTP to prevent a frustrating and very common alert from being generated.

The next rule will trigger if two conditions are met. The first condition is that 100066 must have already triggered. The second condition is that the text string "Invalid User" is found within the log.

```
 <rule id="100067" level="14">
  <if_sid>100066</if_sid>
  <match>Invalid User</match>
  <description>Attempt to login using a non-existent
user</description>
  <group>invalid_login,authentication_failed,</group>
 </rule>
```

The last rule will trigger only if 100067 has triggered six times within the past 120 seconds.

```
<rule id="100068" level="14" frequency="6"
timeframe="120" ignore="60">
  <if_matched_sid>100067</if_matched_sid>
  <description>WS-FTP brute force
attempt!</description>
  <same_source_ip />
  <group>authentication_failures,</group>
 </rule>
```


© 2011 The SANS Institute Author retains full rights.

The newly created rule should be tested before the service is restarted to verify they work as expected. Rule 100068 cannot be tested using ossec-test since it relies on another rule.

The results of the rule test are shown below:

```
root@TEST-SERVER:~# /var/ossec/bin/ossec-logtest
2011/06/22 20:23:45 ossec-testrule: INFO: Reading local decoder file.
2011/06/22 20:23:45 ossec-testrule: INFO: Started (pid: 22995).
ossec-testrule: Type one log per line.


Jun 20 15:42:20 FTP-SERVER SFTP: Invalid User <Host=FTP-SERVER,
User=anonymous><Command=NOTIFICATION, Parameters=default,
Client=217.25.214.12:58251>


**Phase 1: Completed pre-decoding.
    full event: 'Jun 20 15:42:20 FTP-SERVER SFTP: Invalid User <Host= FTP-SERVER,
User=anonymous><Command=NOTIFICATION, Parameters=default,
Client=217.25.214.12:58251>'
    hostname: 'FTP-SERVER'
    program_name: 'SFTP'
    log: 'Invalid User <Host=FTP-SERVER, User=anonymous><Command=NOTIFICATION,
Parameters=default, Client=217.25.214.12:58251>'

**Phase 2: Completed decoding.
    decoder: 'WS-FTP'
    dstuser: 'anonymous'

**Phase 3: Completed filtering (rules).
    Rule id: '100067'
    Level: '14'
    Description: 'Attempt to login using a non-existent user'
**Alert to be generated.
```


© 2011 The SANS Institute Author retains full rights.

The result of the test proves that the custom decoder and rules work as expected.

6. Advanced Customizations

There are many sites that track sources of malicious traffic. Some of these sites provide mechanisms to download routinely updated lists of blacklisted IPs. These blacklists can be utilized by security personnel to monitor for malicious connectivity to protected resources by writing scripts to import them into OSSEC rules automatically. A method to automatically create rules based on web available blacklists is shown below.

6.1. Scripts

The following script will download a list of malicious IPs from Emerging Threats.

```
#!/bin/bash
# Clean-up and remaining items from the last run
rm /var/ossec/logs/blacklists/cleaned.lst

# Create file
touch /var/ossec/logs/blacklists/cleaned.lst

# Grab the list of IPs from Emerging Threats and output into a working directory
wget http://rules.emergingthreats.net/fwrules/emerging-Block-IPs.txt -O
/var/ossec/logs/blacklists/shunlist.lst

# The emerging threats file contains text other than IPs. The following section cleans that
up. It also verifies that the file downloaded is in fact a text file and not something possibly
malicious.
file=$(file -ib /var/ossec/logs/blacklists/shunlist.lst)
if [ "$file" == "text/plain; charset=us-ascii" ]
 then
 cat /var/ossec/logs/blacklists/shunlist.lst | grep -v '#'
/var/ossec/logs/blacklists/shunlist.lst | grep -v '^$' >
/var/ossec/logs/blacklists/cleaned.lst
 sort -u /var/ossec/logs/blacklists/cleaned.lst > /var/ossec/logs/blacklists/sorted.lst
else
exit
fi
```


© 2011 The SANS Institute Author retains full rights.

The script creates a nicely formatting list of malicious IPs. This list can then be easily iterated through to create rules within OSSEC.

The script below creates rules based on the IP list.

```
#!/bin/bash

# Set the field separator to a newline

IFS="
"
# Begin creating rules at #100499
calc="100499"

# Create a variable to the sorted IP list
file="/var/ossec/logs/blacklists/sorted.lst"

# The following section loops through the IPs contained in the list
# above creating rules for each, beginning at 100499

cat ${file} | \
while read IP
do

calc=$(($calc+1))

echo "<group name=\"web\">"
echo "<rule id=\"$calc\" level=\"14\">"
echo "<if_sid>31100</if_sid>"
echo "<srcip>$IP</srcip>"
echo "<description>Shunnnnnnn!</description>"
echo "</rule>"
echo "</group>"
echo
done
```


© 2011 The SANS Institute Author retains full rights.

The output of that file is then redirected into a file to be included in the OSSEC rules library. A sample of the finished xml-formatted file is shown below:

```
<group name="web">
<rule id="100671" level="14">
<if_sid>31100</if_sid>
<srcip>115.68.17.0</srcip>
<description>Shunnnnnnn!</description>
</rule>
</group>

<group name="web">
<rule id="100672" level="14">
<if_sid>31100</if_sid>
<srcip>115.68.21.172</srcip>
<description>Shunnnnnnn!</description>
</rule>
</group>

<group name="web">
<rule id="100673" level="14">
<if_sid>31100</if_sid>
<srcip>115.68.4.20</srcip>
<description>Shunnnnnnn!</description>
</rule>
</group>

<group name="web">
<rule id="100674" level="14">
<if_sid>31100</if_sid>
<srcip>115.86.180.47</srcip>
<description>Shunnnnnnn!</description>
</rule>
</group>
```

This is just one creative way to utilize OSSEC. In this example once a source is known to be malicious and is noted within the Emerging Threats site any matching log entries will generate an alert. Administrators can then review other related information and react accordingly.

© 2011 The SANS Institute Author retains full rights.

7. Conclusion

OSSEC is a powerful tool that those concerned with security can utilize to protect critical infrastructure. Using the knowledge contained within this document should allow readers to get started writing custom rules and decoders for a custom environment. It also hopefully inspires the reader to think of creative ways to extend OSSEC functionality beyond its standard uses.

This document has covered only a small portion of OSSEC rich featureset. Readers are encouraged to explore OSSEC's additional functionality to see the other ways it can help protect users and data.

© 2011 SANS Institute, Author retains full rights.

Chad Robertson, chadrober@gmail.com

© 2011 The SANS Institute Author retains full rights.

8. References

. (2011). Retrieved from www.ossec.net: http://www.ossec.net/wiki/Know_How:RuleIDGrouping

B., J. (2010, 11 3). . Retrieved from INetU Managed Hosting: http://blog.inetu.net/2010/11/payment-card-industry-new-pci-dss-v-2-0-updates/

Bhatia, S. K. (2010). Retrieved from www.cs.umsl.edu: www.cs.umsl.edu/~sanjiv/classes/cs2750/lectures/re.pdf

Brenton, C. (2010, 02 17). Retrieved from chrisbrenton.org: http://www.chrisbrenton.org/2010/02/combining-logwatch-and-ossec-%E2%80%93-part-3/

Cid, D. (2007). . Retrieved from www.ossec.net: http://www.ossec.net/wiki/Know_How:Regex_Readme

Cid, D. B. (2010). Retrieved from ossec.net: http://www.ossec.net/wiki/Know_How:Ignore_Rules

Cid, D. B., & Ozturk, A. (2006, 05 12). Retrieved from ossec.net: http://www.ossec.net/ossec-docs/ossec-hids_oahmet_eng.pdf

Dave. (2010, 04). . Retrieved from osdir.com: http://osdir.com/ml/ossec-list/2010-04/msg00029.html

Hay, A., Cid, D., & Bray, R. (2008). Syngress.

Ipswitch. (2010). . Retrieved from http://www.ipswitchft.com: http://www.ipswitchft.com/Products/Ws_Ftp_Pro/

Itlibitum Corp. (2006). . Retrieved from www.xml.su: http://www.xml.su/

Klein, J. C. (2009, 10 13). Retrieved from madirish.net: http://www.madirish.net/?article=434

PCI Security Standards Council. (2010, 10 28). Retrieved from www.pcisecuritystandards.org: https://www.pcisecuritystandards.org/documents/pci_dss_v2.pdf

Swift, D. (2006, 12 23). . Retrieved from sans.org: http://www.sans.org/reading_room/whitepapers/logging/practical-application-sim-sem-siem-automating-threat-identification_1781

Third Brigade Inc. (2009, 02 25). . Retrieved from ossec.net: http://www.ossec.net/ossec-docs/ossec-PCI-Solution.pdf

Trend Micro. (2010). . Retrieved from http://www.ossec.net/

Xavier. (2008, 03 27). . Retrieved from rootshell.be: http://blog.rootshell.be/2008/03/27/log-correlation-for-free/

Chad Robertson, chadrober@gmail.com

© 2011 The SANS Institute Author retains full rights.

Appendix 1 (Itlibitum Corp, 2006)

Structure of XML Document

:?xml version="*1.0*" encoding="*UTF-8*" dalone="*no*" ?>	XML Declaration
	White space characters (space, carriage return, line feed, tab, etc)
:!-- *Comments* -->	Comments
:root_element>	Open tag of "*root_element*"
<*subElement*>	Open tag of "*subElement*"
...text...	Data
<*subSubElement attr_name="attr_value*">	Open tag of "*subSubElement*" with attribute "*attr_name*" equal "*attr_value*"
<![CDATA[*...any characters (including ·kup)..*]]>	CDATA Section
</*subSubElement*>	Close tag of "*subSubElement*"
</*subElement*>	Close tag of "*subElement*"
iptyElement/>	Tag of empty element
:/*root_element*>	Close tag of "*root_element*"

© 2011 SANS Institute

© 2011 The SANS Institute Author retains full rights.

Appendix 2 (Cid D. , 2007)

The following expressions are supported:

```
\w  ->  A-Z, a-z, 0-9 characters
\d  ->  0-9 characters
\s  ->  For spaces " "
\t  ->  For tabs.
\p  ->  ()*+,-.:;<=>?[] (punctuation characters)
\W  ->  For anything not \w
\D  ->  For anything not \d
\S  ->  For anything not \s
\.  ->  For anything
```

Each regular expression can be followed by:

```
+  ->  To match one or more times (eg \w+ or \d+)
*  ->  To match zero or more times (eg \w* or \p*)
```

The following special characters are also supported:

```
^ -> To specify the beginning of the text.
$ -> To specify the end of the text.
| -> To create an "OR" between multiple patterns.
```

Any of the following characters must be escaped with a "\" before use:

```
$ -> \$
( -> \(
) -> \)
\ -> \\
```

Chad Robertson, chadrober@gmail.com

© 2011 The SANS Institute Author retains full rights.

(Know_How:RuleIDGrouping, 2011)

Rule ID Range	General Category
00000 - 00999	Internally reserved for ossec
01000 - 01999	General syslog
02100 - 02299	NFS
02300 - 02499	Xinetd
02500 - 02699	Access control
02700 - 02729	Mail/procmail
02800 - 02829	Smartd
02830 - 02859	Crond
02860 - 02899	Mount/Automount
02900 - 02929	Dpkg logs
02930 - 02959	Yum logs
03100 - 03299	Sendmail
03300 - 03499	Postfix
03500 - 03599	Spamd
03600 - 03699	Imapd
03700 - 03799	MailScanner
03800 - 03899	Ms Exchange (IIS SMTP)
03900 - 03999	Courier (imapd/pop3d/pop3-ssl)
09900 - 09999	vpopmail
09800 - 09899	vm-pop3d
09700 - 09799	Dovecot

Chad Robertson, chadrober@gmail.com

© 2011 The SANS Institute Author retains full rights.

04100 - 04299	Generic Firewall
04300 - 04499	Cisco PIX/FWSM/ASA Firewall
04500 - 04699	Netscreen Firewall
04700 - 04799	Cisco IOS
04800 - 04899	SonicWall Firewall
05100 - 05299	Kernels (Linux, Unix, etc)
05300 - 05399	Su
05400 - 05499	sudo
05500 - 05599	Pam unix
05600 - 05699	Telnetd
05700 - 05899	sshd
05900 - 05999	Adduser or user deletion.
06100 - 06199	Solaris BSM Auditing
06200 - 06299	Asterisk
06300 - 06399	MS DHCP logs
07100 - 07199	Tripwire
07200 - 07299	Arpwatch
07300 - 07399	Symantec Anti Virus
07400 - 07499	Symantec Web Security
07500 - 07599	McAfee VirusScan Enterprise
07600 - 07699	Trend Micro OSCE (Office Scan)
07700 - 07799	Microsoft Security Essentials

© 2011 The SANS Institute Author retains full rights.

09100 - 09199	PPTP
09200 - 09299	Squid syslog
09300 - 09399	Horde IMP
09400 - 09499	Roundcube
09500 - 09599	Wordpress WPsyslog2
09600 - 09699	cimserver
10100 - 10199	FTS
11100 - 11199	FTPd
11200 - 11299	ProFTPD
11300 - 11399	Pure-FTPD
11400 - 11499	vs-FTPD
11500 - 11599	MS-FTP
12100 - 12299	Named (bind DNS)
13100 - 13299	Samba (smbd)
14100 - 14199	Racoon SSL
14200 - 14299	Cisco VPN Concentrator
17100 - 17399	Policy

© 2011 The SANS Institute Author retains full rights.

© 2011 SANS Institute, Author retains full rights.

18100 - 18499	Windows system
19100 - 19499	Vmware ESX
20100 - 20299	IDS
20300 - 20499	IDS (Snort specific)
30100 - 30999	Apache error log.
31100 - 31199	Web access log
31200 - 31299	Zeus web server
31300 - 31399	Nginx error log.
35000 - 35999	Squid
40100 - 40499	Attack patterns.
40500 - 40599	Privilege scalation.
40600 - 40999	Scan patterns.
50100 - 50299	MySQL.
50500 - 50799	PostgreSQL
60000 - 60299	Atomic Secured Linux.

© 2011 The SANS Institute Author retains full rights.

© 2011 SANS Institute, Author retains full rights.

100000 - 109999	User defined rules

© 2011 SANS Institute, Author retains full rights.

© 2011 The SANS Institute Author retains full rights.

Appendix 4 (Cid & Ozturk, ossec.net, 2006)

Group Type	Group Name	Description
Reconnaissance	connection_attempt	Connection attempt
	web_scan	Web scan
	recon	Generic scan
Authentication Control	authentication_success	Success
	authentication_failed	Failure
	invalid_login	Invalid
	login_denied	Login denied
	authentication_failures	Multiple failures
	adduser	User account added
	account_changed	User account changed or removed
Attack/Misuse	automatic_attack	Worm (nontargeted attack)
	exploit_attempt	Exploit pattern
	invalid_access	Invalid access
	spam	Spam
	multiple_spam	Multiple spam messages
	sql_injection	SQL injection
	attack	Generic attack
	rootcheck	Rootkit detection
	virus	Virus detected
Access Control	access_denied	Access denied
	access_allowed	Access allowed
	unknown_resource	Access to nonexistent
	firewall_drop	Firewall drop
	multiple_drops	Multiple firewall drop
	client_misconfi	Client misconfiguration
	client_error	Client error
Network Control	new_host	New host detected
	ip_spoof	Possible ARP spoofing
System Monitor	service_start	Service start
	service_availability	Service availability at risk
	system_error	System error
	system_shutdown	Shutdown
	logs_cleared	Logs cleared
	invalid_request	Invalid request

© 2011 The SANS Institute Author retains full rights.

© 2011 SANS Institute, Author retains full rights.

Group Type	Group Name	Description
	promisc	Interface switched to promiscuous mode
	policy_changed	Policy changed
	config_changed	Configuration changed
	syscheck	Integrity checking
	low_diskspace	Low disk space
	time_changed	Time change
Policy Violation	login_time	Login time
	login_day	Login day

© 2011 SANS Institute, Author retains full rights.

© 2011 The SANS Institute Author retains full rights.