Making the “ference” out of conference

Making_the_most_of_OSSEC

Michael Starks, CISSP, CISA
July 25, 2013
Blah, blah, blah.

Does it really matter?
What this presentation is:

- Chocked full of OSSEC tips
- Designed to be *useful and a reference*
- Info from years of experience using OSSEC in real-world environments
What this presentation is not:

- Full of interesting pictures
- A storyboard, like a good presentation should be
- Full of synergies, multidisciplinary approaches, swim lanes, key values and trend-setters.
root@osseccon#: ls -l /presentation

deployment
rules_and_decoders
taming_syscheck
Hacks && tips
demo_cdb_lists
./presentation/deployment
Before beginning the deployment

- Define what success looks like
- Assess your current situation (do you already have centralized logging/IDS?)
- Start with a small pilot
- Be prepared for ongoing tuning
- Dust off your incident response plan
For centralized management, use an almost empty ossec.conf. This is all you really need:

```xml
<ossec_config>
  <client>
    <server-ip>1.2.3.4</server-ip>
  </client>
</ossec_config>
```
For Windows agents, add this:

```xml
<active-response>
  <disabled>no</disabled>
</active-response>
```

The Windows agent executable is just an archive. Open it with 7zip to edit files at will!
Now put all of your configuration in `agent.conf` on the manager. It will be pushed to the agents periodically. When the agent restarts, it will read the `agent.conf` and apply the settings.
"But my agent doesn't restart automatically. How can I make it do that?"

Start monitoring the file in ossec.conf, then write a rule to look for the agent.conf file change (for example):

```xml
<rule id="100001" level="7">
  <if_group>syscheck</if_group>
  <match>:\program files\ossec-agent\shared\agent.conf$</match>
  <group>win_agent.conf_changed</group>
  <description>Windows agent.conf File Changed</description>
</rule>
```
Now create an active response to restart the agent when the file changes:

```xml
<active-response>
    <command>restart-win-agent</command>
    <location>local</location>
    <rules_group>win_agent.conf_changed</rules_group>
</active-response>
```
Not so fast! *Always* use `verify-agent-conf` to make sure everything is OK.

A blank result means it passed

```
./bin/verify-agent-conf
```

`verify-agent-conf`: Verifying
`[/var/ossec/etc/shared/agent.conf]`. 
You don't need a compiler on every system you deploy the agent to. Use a binary install:

```bash
# cd ossec-*/src
# make setagent
# make all
# make build
# cd ../../

# echo "USER_BINARYINSTALL="'y'"" >> ossec-hids*/etc/preloaded-vars.conf
```
ossec-authd and agent-auth make adding agents a breeze. Deploy agents and automatically add them to the manager.

```bash
#/var/ossec/bin/agent-auth -m 192.168.1.1 -p 1515
INFO: Connected to 192.168.1.1:1515
INFO: Using agent name as: melancia
INFO: Send request to manager. Waiting for reply.
INFO: Received response with agent key
INFO: Valid key created. Finished.
INFO: Connection closed.
```
Go big or go home

One OSSEC manager can easily handle thousands of agents

```
#cd src; make setmaxagents
#cd .. && ./install.sh
```
A bit of kernel tuning may help (adjust as needed)...

```bash
# ulimit -n 2048
# sysctl -w kern.maxfiles=2048
# sysctl -w net.core.rmem_default=5123840
# sysctl -w net.core.rmem_max = 5123840
```
I like to put OSSEC on its own LVM partition and optimize it a bit with the noatime value.

If something goes crazy and starts sending out millions of logs, it won’t fill up the root partition and take down the entire server.

```
/dev/VolGroup00/ossec /log/ossec ext3 defaults,noatime 1 2
```
Watch your disk space when using `<check_diff>` and `<report_changes>`.

Each monitored file/process output is uploaded to the manager.
Speaking of changes in files...

Beware: OSSEC could email you sensitive data in the clear should you monitor sensitive files.
./presentation/rules
No decoder? No problem! If you have an application that isn't supported by OSSEC, you can still write a rule to match on a string in the log.

```
<rule id="100000" level="15">
    <match>failed_login</match>
    <description>Testing Rule</description>
</rule>
```
You can combine a <match> and a <regex> in the same rule and that will act like an AND, even if the <regex> only contains a string.

```xml
<rule id="100000" level="15">
    <match>failed_login</match>
    <regex>bob</regex>
    <description>Bob is bad</description>
</rule>
```
ossec-logtest is your friend. *Always* test a log sample against a newly created rule or decoder. Not all logs are decoded properly.

It is perhaps the **single most useful utility** within **OSSEC**
Tune, tune, tune!

OSSEC can easily produce *thousands* of alerts a day, or even in an hour!

Your brain cannot process a steady stream of threats. You will eventually start to ignore alerts.

Make sure alerts are relatively *rare* and *relevant*
If you still want an active response to be fired, keep the rule at the same level and use this option:

<options>no_email_alert</options>
Make sure your alerts do not get marked as spam. Whitelist the address in `<email_from> in your ossec.conf`

Sometimes it seems as if the spam filter doesn't mind, then one day it kicks in...
Create custom groups on the fly:

```xml
<rule id="100000" level="3">
  <match>yada</match>
  <group>Seinfield</group>
  <description>One Yada</description>
</rule>
```
Then you can do stuff like this:

```xml
<rule id="100002" level="10" frequency="3" timeframe="5">
	<if_matched_group>Seinfeld</if_matched_group>
	<description>Yadayadayada</description>
	<group>multiple_yadas,</group>
</rule>
```
Frequency in a composite rule doesn't mean what you think it does

This rule matches on the fifth event.
“Huh? Come again?”

- OSSEC starts counting after the event is seen from the original, atomic rule
- Frequency means “greater than”
- Therefore, add two to the count to arrive at “equals”
Don't write rules dependent on composite rules. You'll get wonky results.

Instead, look at the atomic rule which the composite rule references and write your rule against that one.
By the way...

- Rules are evaluated from highest to lowest (except level 0, which is evaluated first)
- Make sure important rules have high levels
Use this to set different failed logon thresholds for different people

Failed login occurred

Level 15 rule looks at the user

Is it an admin? No.

Level 14 rule looks for a service account...
# cat /presentation/rules/tip10

- Make all changes in local_rules.xml
- If you edit the built-in rules, your changes will be lost when you upgrade!
XML is not always flexible enough for rules. Write your rules in C with compiled rules.

```
<rule id="100155" level="10">
  <if_sid>18111</if_sid>
  <compiled_rule>comp_mswin_targetuser_calleruser_diff</compiled_rule>
  <description>User changed someone else password.</description>
</rule>
```
./presentation/decoders
Make all decoder changes/additions in local_decoder.xml

If you edit the built-in decoders.xml, your changes will be lost when you upgrade!

If correcting a built-in decoder, you will need to first comment it out in decoders.xml
When writing decoders, prepare for some pain

It is an iterative and time consuming process to write a good decoder
It is *rare* to find a well-formatted, standards-based log

Just when you think you have the perfect decoder, along comes one log with an extra space
Writing good decoders is part art and part science.

It has to be strict enough to discourage log injection, while being loose enough to allow for subtle variations in logs.
Even when you get everything exactly right, it sometimes doesn't work for complex logs with many extracted fields.

That's why you can...
Write compiled decoders in C
Have a look at prelude.c for an example
Again, always use ossec-logtest to test your decoder against several log samples. It is perhaps the **single most useful utility within OSSEC**.
By default, syscheck ignores files after three changes.

Have you changed your index.htm more than three times? How would you know if it was compromised?

Set `<auto_ignore>no</auto_ignore>` in ossec.conf
syscheck does not alert on new files

Add <alert_new_files>yes</alert_new_files> to ossec.conf

Then overwrite rule 554 in local_rules.xml

<rule id="554" level="7" overwrite="yes">
  <category>ossec</category>
  <decoded_as>syscheck_new_entry</decoded_as>
  <description>New File Added</description>
  <group>syscheck</group>
</rule>
"But wait, you said I should only get alerted on what matters. Won't this create lots of alerts?"

This will create a situation where you could be bombarded with alerts.

The point is to think about which changes matter. Then tune out the non-important stuff.

This is very context-dependent. An installation with 1,000 agents will be very different than a single installation.
A quick and easy way not to get flooded with syscheck alerts just prior to patching your system is to clear the syscheck database.

You'll lose the ability to report on modified files using agent control, but previous alerts won't be affected.

```bash
./bin/syscheck_control -u <agent ID>
```
To speed up syscheck on faster systems and decrease the time until realtime integrity check starts, add something like this to local_internal_options.xml. Watch subsequent CPU usage.

```bash
syscheck.sleep=1
syscheck.sleep_after=100
```
If you want to retain the syscheck database but still not be alerted, create a granular rule using something like `<time>2 am - 3 am</time>` to ignore changes during a patching window.

This is not risk free. An attacker could still modify files during that time.
./presentation/hacks && ./tips
Multiple OSSEC managers can co-exist peacefully on one server
Use one for test and one for production
Use a separate instance when you need unique global options
To keep your production instance from getting mucked up

```
chattr +i /etc/ossec-init.conf
```

Run installer, then tell it “no” when it asks you to upgrade. Install to a different location.
Use this location for your test instance. Bind to a different address (real or virtual) using the `<local_ip>` option in ossec.conf.

Use a custom init script to manage both instances (chattr this, too)
Use OSSEC to monitor OSSEC. This is really useful to see if someone has messed with your configuration or to get back to a previous state by using `<report_changes>`

Ignore these directories:

```
<ignore>/var/ossec/queue</ignore>
<ignore>/var/ossec/logs</ignore>
<ignore>/var/ossec/stats</ignore>
<ignore>/var/ossec/var</ignore>
```
If also using OSSEC's syslog server, disable rids as there is no protection against log replay attacks anymore.

To disable the rids check, just create local_internal_options.conf and add:

```bash
# Verify msg id (set to 0 to disable it)
remoted.verify_msg_id=0
```
Know the difference between alerts and logs. Alerts are in alerts.log while the full logs are in archives.log

Log archival is *not enabled by default*. In the event of a compromise, you'll want this!
Alerts don't always contain the good stuff. They may be truncated.

Only the first ten lines are shown.

Be particularly aware of this when reviewing `<check_diff>` alerts.
Alert and full logs are archived daily with a chained checksum.

At the end of each day, OSSEC will generate an md5/sha1 sum of the current logs plus the md5/sha1 sum of the checksum from the logs of the previous day.
Posting checksums publicly will introduce a high level of non-repudiation

Possibilities:

- A page on your web site which Google will index
- Tweets
- Printed in a newspaper
- Go retro with a line-feed printer
How might you post them? Why, using OSSEC of course!

Create a script to do the heavy lifting, then have OSSEC execute it with full_command. Write a rule to watch for error messages in the output and set it to alert
./presentation/demo_cdb_lists