OSSEC Active Response and Self Healing

Scott R. Shinn
Chief Technical Officer
About

- 20+ years in cybersecurity
- 7 years on ossec
- White House: 1st firewall
- Sr. scientist at Cisco/Wheelgroup: 1st Network IDS
- Founder of Plesk, web hosting automation
- Ran 120+ staff government SOC/NOC
Open Source Projects

- Atomic Yum repository, over 600 projects used on over 1 million servers
- OSSEC (Why you are all here)
- OpenVAS: Network Vulnerability Scanner
- grsecurity: Kernel Security
- Mod_Security: Web Application Firewall
OSSEC Projects

- Make things less difficult
- Packaging / Integration for RHEL/CentOS/Cloudlinux and Fedora
- Rules and Decoders
- Ossec-execd
  - Sqlite support, expanding action
- Ossec-dbd
  - DB Improvements/Scalability, Mariadb
The Value of OSSEC

- Immediate value with minimal configuration (keep things easy)
- Active Response
- Log Based Analysis
- Inotify (and Fanotify soon!) as a means for real-time detection
Challenges of OSSEC

- Retroactive: Actions occur after a Log event
- People don't scale (and Scott is lazy.)
- “Alert Fatigue”
- Multiple languages are not implemented well
- Not good for web attacks
Bypass Examples

- HTTP POSTS!
- Encoding attacks:
  - This is valid:
  - So is this: GET%20%2Findex.php%3Farg1%3Dhttp%3A%2F%2Fwww.attack.com%2Fexample.txt
- So is this in a POST: Robert); DROP /* xkcd reference */ TABLE Students
How We Use OSSEC

- Standalone, the client & server are the same system
- MySQL driven back end, with triggers to create real-time meta-events
- Process “complex” attacks, that spam multiple services
- Web Based Brute force attacks, with a WAF
- WAF + OSSEC = proactive
- grsecurity + OSSEC = proactive
Active Response: The Human Hybrid

- OSSEC is combined in a classic client / server configuration
- Zoneminder, a CCTV IDS is the source. Cameras have motion detection on a specific area
- Triggers a Level 13 alert to OSSEC, the active response to Instant messenger and IRC bots
- Human operator takes action
Configuration Example

- Event is bound to a single rule
- There is no timeout. This script writes to a log file, read by a bot
- Event only runs on the agent that triggered the alert

```xml
<command>
  <name>zm-notify-101099</name>
  <executable>zm-notify-101099</executable>
  <timeout_allowed>no</timeout_allowed>
  <expect />
</command>

<active-response>
  <command>zm-notify-101099</command>
  <location>local</location>
  <rules_id>101099</rules_id>
</active-response>
```
Active Response Output:

- (11:13:01 AM) <TurtleBot> Someone is in Mike’s spot!
- “Operator” takes further action.
Result
Active Reponse: Command Hybrid

- Not exactly Active Response, but you'll see.
- A “log” is created using the <command> syntax from the script: asl-port-check
- The command is executed every 60 seconds, the intent is to detect new ports
- OSSEC rules check the output of the command, and looks for changes with <check_diff />
Main Configuration Example

- Ossec.conf:

  `<localfile>
  <log_format>full_command</log_format>
  <command>/var/ossec/active-response/bin/asl-port-check</command>
  <frequency>60</frequency>
  </localfile>`
Command example:

- /var/ossec/active-response/bin/asl-port-check

```bash
#!/bin/bash
lsof -nP | grep LISTEN|grep TCP \n| egrep -v "127.0.0.1|:[1-9][0-9][0-9][0-9].*(ftp|rpc.statd|java|\-)|\[0-9]+\.[0-9]+:53 \n.*named|:30000.*sudo" | awk '{print $1,$3, $9}' | awk -F: '{print $1,$2}' |awk '{print $1,$2,$4}'| sort -u | awk -f /var/ossec/active-response/bin/ports.awk
```
Rule Example

- /var/ossec/etc/rules.d/example.xml

```xml
<rule id="100533" level="7" >
  <info>rev:2</info>
  <if_sid>530</if_sid>
  <match>ossec: output: '/var/asl/bin/asl-port-check|ossec: output: 'netstat -tan'</match>
  <check_diff />
  <options>no_email_alert</options>
  <description>Listening ports status has changed (new port opened or closed).</description>
</rule>
```
Self Healing

- Mix of Security and Non-security events.
- Originally started to handle a very uncommon semaphore exhaustion condition
- Later expanded to strip dangerous settings from libraries and applications
Self Healing Example 1

- Apache 2.0 Semaphore exhaustion
- Ossec.conf:

```xml
<command>
  <name>self-healing-30300</name>
  <executable>self-healing-30300</executable>
  <timeout_allowed>no</timeout_allowed>
  <expect />
</command>
<active-response>
  <command>self-healing-30300</command>
  <location>local</location>
  <rules_id>30300</rules_id>
  <rules_id>30301</rules_id>
</active-response>
```
Self Healing Example 1

- /var/ossec/etc/rules.d/51_asl_apache_sh_rules.xml
- Log snippet from apache looks like this:
  - No space left on device. (28)No space
  - There is no timestamp
- We tie two log events together to handle this condition

```xml
<rule id="1007" level="7">
  <match>file system full|No space left on device</match>
  <description>File system full.</description>
  <group>low_diskspace</group>
</rule>

<rule id="30300" level="8">
  <if_sid>1007</if_sid>
  <match>(28)No space</match>
  <description>Self Healing: Resolving Semaphore exhausting condition.</description>
</rule>
```
Self Healing Rule Example 1

#!/bin/sh
# Copyright 2013
# Atomicorp, Inc.

# Parent rule: 30300
# Level: 8
# Description: This event clears the apache semaphore set
# Version: 1.0.1

LOG=/var/ossec/logs/self-healing.log

date >> /var/ossec/logs/self-healing.log

# Clear semaphore table
/usr/bin/ipcs -s | /bin/grep apache | /bin/awk '{ print $2 }' | /usr/bin/xargs /usr/bin/ipcrm sem >> $LOG
Self Healing Example 2

- Context is in the environment of a hardened kernel using stack protection.
- Many Linux binaries (and libraries!) will request an executable stack when they do not require it.
- This bypasses subset stack protection options like execshield. In grsecurity however, this is not permitted.
- Normally we do this with RPM triggers, but this cannot handle source installations. OSSEC allows us to do that.
- Allows us to reduce support costs, and makes the system safer.
Self Healing Example 2

- Ossec.conf

```
<!-- 30302 -->
<command>
  <name>self-healing-30302</name>
  <executable>self-healing-30302</executable>
  <timeout_allowed>no</timeout_allowed>
  <expect />
</command>

<active-response>
  <command>self-healing-30302</command>
  <location>local</location>
  <rules_id>30302</rules_id>
</active-response>
```
Self Healing Example 2

- Log Event is generated by the kernel, and spans 3 rules, they collectively look for:
  grsec.... Segmentation fault.... bin/php
- In this example, the policy is unless allow; deny. So the scary application never actually runs. We catch the policy violation, and use this to trigger a Self-Healing condition
Self Healing Example 2

- Rule 30302 is triggered, and only executes on the agent where it was detected.
- The self healing rule parses the php.ini file for the path to extensions used, since we do not know exactly where the user has configured them.
- Once detected, stack-exec bits are stripped from the library:
  - Execstack -c /path/to/extension
Self Healing Example 2

SEARCH_PHP_INI=$(locate -r php.ini$)
PHP_INI="$SEARCH_PHP_INI /etc/php.ini /usr/local/lib/php.ini /usr/local/etc/php.ini /usr/local/cpanel/3rdparty/php/53/etc/php.ini"


for file in $PHP_INI; do
    if [ -f $file ]; then
        clear_php_exec_bit $file
    fi
done

for dir in $DIRS; do
    if [ -d $dir ]; then
        for i in `ls $dir`; do
            if [ -f $dir/$i ]; then
                clear_php_exec_bit $dir/$i
            fi
        done
    fi
done
Future Projects

- Grand Decoder restructuring (rules.d and decoders.d)
- Rule / Decoder updater
- MariaDB support (very close!)
- Fannotify for real-time integrity checking, and real-time rootkit / configuration analysis
- Compliance validation for DISA STIG's, DHS, NISPOM, etc. Work is already prototyped with the Aqueduct project
Rules/Decoder feed

- Work is available in my Bitbucket repo:
  - [https://bitbucket.org/sshinn/ossec-rules](https://bitbucket.org/sshinn/ossec-rules)
- Rules live in /var/ossec/etc/rules.d/ and are loaded much like apache configs from /etc/httpd/conf.d/
- Decoders live in /var/ossec/etc/decoders.d/
Rules.d example

- Format is <Number>-<Channel>-<name>
- This manages loading order, and shows dependencies between rules
- Rules are versioned using <info>rev:X</info>

- 00_asl_rules_config.xml
- 40_asl_syslog_rules.xml
- 50_asl_abrt_rules.xml
- 50_asl_apache_rules.xml
- 50_asl_arpwatch_rules.xml
- 50_asl_asterisk_rules.xml
- 50_asl_bro-ids_rules.xml
- 50_asl_cimserver_rules.xml
- 50_asl_cisco-ios_rules.xml
Decoders.d example

- Format: `<number>-<channel>-<name>`
- Overlapping / Missing decoders can cause a hard error, thus causing ossec to exit
- No decoder interdependencies yet, but this structure allows for it

- `50-asl-aix-ipsec-decoder.xml`
- `50-asl-apache-decoder.xml`
- `50-asl-arpwatch-decoder.xml`
- `50-asl-asterisk-decoder.xml`
- `50-asl-auditd-decoder.xml`
- `50-asl-bro-ids-decoder.xml`
- `50-asl-checkpoint-decoder.xml`
- `50-asl-cimserver-decoder.xml`
- `50-asl-cisco-ios-decoder.xml`
- `50-asl-cisco-vpn-decoder.xml`
Questions?

Scott Shinn
scott@atomicorp.com