



Log Analysis using OSSEC

Daniel B. Cid
dcid@ossec.net



Agenda

- Defining LIDS (Log-Based IDS)
- OSSEC Overview
- Installation demo
- Log decoding and analysis with OSSEC
- Writing decoders
- Writing rules
- Examples of rules and alerts in the real world

Concepts

- OSSEC does “security log analysis”
 - *It is not a log management tool*
 - *Only stores alerts, not every single log*
 - *I still recommend log management and long term storage of ALL logs*
- Security Log Analysis can be called LID(S)
 - **Log-based Intrusion Detection System**
 - *We could even call it OSSEC LIDS, since some users only use the log analysis side of OSSEC*

Defining LIDS

- Log-Based Intrusion Detection

Log Analysis for intrusion detection is the process or techniques used to detect attacks on a specific environment using logs as the primary source of information.

LIDS is also used to detect computer misuse, policy violations and other forms of inappropriate activities.



LIDS benefits

- Cheap to implement
 - OSSEC is free, for example
 - Does not require expensive hardware
- High visibility of encrypted protocols
 - SSHD and SSL traffic are good examples
- Visibility of system activity (kernel, internal daemons,..)
- Every application/system can be a part of it
 - They all have some kind of log!
 - Including firewalls, routers, web servers, applications, etc



What is OSSEC?

- Open Source Host-based IDS (HIDS)
- <http://www.ossec.net>
- Main tasks:
 - *Log analysis*
 - *File Integrity checking (Unix and Windows)*
 - *Registry Integrity checking (Windows)*
 - *Host-based anomaly detection (for Unix – rootkit detection)*
 - *Active response*

OSSEC is an Open Source Host-based Intrusion Detection System. It performs log analysis, integrity checking, Windows registry monitoring, Unix-based rootkit detection, real-time alerting and active response.



Why OSSEC?

- Solves a real problem and does it well (log analysis)
- Free (as in cookies and speech)
- Easy to install
- Easy to customize (rules and config in xml format)
- Scalable (client/server architecture)
- Multi-platform (Windows, Solaris, Linux, *BSD, etc)
- Secure by default
- Comes with hundreds of decoders/rules out of the box:
 - *Unix Pam, sshd (OpenSSH), Solaris telnetd, Samba, Su, Sudo, Proftpd, Pure-ftpd, vsftpd, Microsoft FTP server, Solaris ftpd, Imapd, Postfix, Sendmail, vpopmail, Microsoft Exchange, Apache, IIS5, IIS6, Horde IMP, Iptables, IPF. PF, Netscreen, Cisco PIX/ASA/FWSM, Snort, Cisco IOS, Nmap, Symantec AV, Arpwatch, Named, Squid, Windows event logs, etc ,etc,*



Why OSSEC (2)?

- External references:
 - OSSEC #1 open source security tool in the enterprise
<http://www.linuxworld.com/news/2007/031207-top-5-security.html>
 - OSSEC #2 IDS tool in the security tools survey.
<http://sectools.org/ids.html>
- Additional references:
<http://www.ossec.net/wiki/index.php/InTheNews>



Installing OSSEC

- Simple and easy
 - *Two models:*
 - Local** (when you have just one system to monitor)
 - Client/Server** for centralized analysis (recommended!)
 - *Select installation type and answer a few questions*
 - *It will setup the appropriate permissions, create users, etc*
- Installation Demo (of latest version 1.2)
 - `# tar -zxvf ossec*.tar.gz`
 - `# cd ossec*`
 - `# ./install.sh`
 - ... (answer all questions)*
 - `# /var/ossec/bin/ossec-control start (after completed)`



Understanding OSSEC

- OSSEC two working models
 - *Local (useful when you have only one system to monitor)*
 - *Agent/Server (recommended!)*
- By default installed at **/var/ossec**
- Main configuration file at **/var/ossec/etc/ossec.conf**
- Decoders stored at **/var/ossec/etc/decoders.xml**
- Binaries at **/var/ossec/bin/**
- All rules at **/var/ossec/rules/*.xml**
- Alerts are stored at **/var/ossec/logs/alerts.log**
- Composed of multiple processes (all controlled by **ossec-control**)



Internal processes

- Remember the Secure by default?
 - Installation script does the chroot, user creation, permissions, etc
 - User has no choice to run it “less secure”
- Each process with limited privileges and tasks
 - Most of them running on chroot
 - Most of them with separated unprivileged user
- Processes:
 - Analysisd – on chroot as user ossec
 - Remoted – on chroot as user ossecr
 - Maild – on chroot as user ossecm
 - Logcollector – as root, but only reads the logs, no analysis
 - Agentd – on chroot as user ossec (agent only)

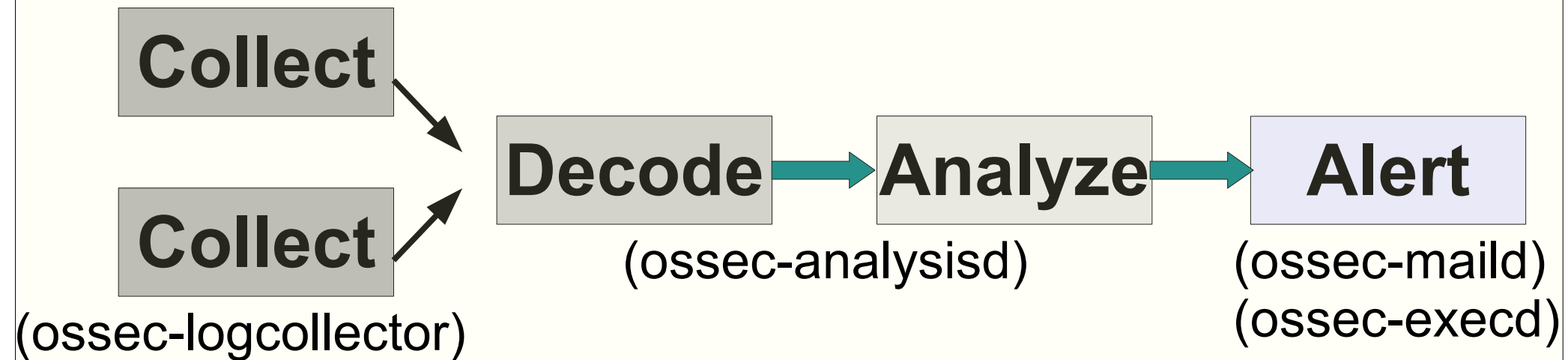


Internal processes (2)

- Each daemon has a very limited task:
 - **Analysisd** – Does all the analysis (**main process**)
 - **Remoted** – Receives remote logs from agents
 - **Logcollector** – Reads log files (syslog, Flat files, Windows event log, IIS, etc)
 - **Agentd** – Forwards logs to the server
 - Maild – Sends e-mail alerts
 - Execd – Executes the active responses
 - Monitord – Monitors agent status, compresses and signs log files, etc
- **ossec-control** manages the start and stop of all of them

Log flow (local)

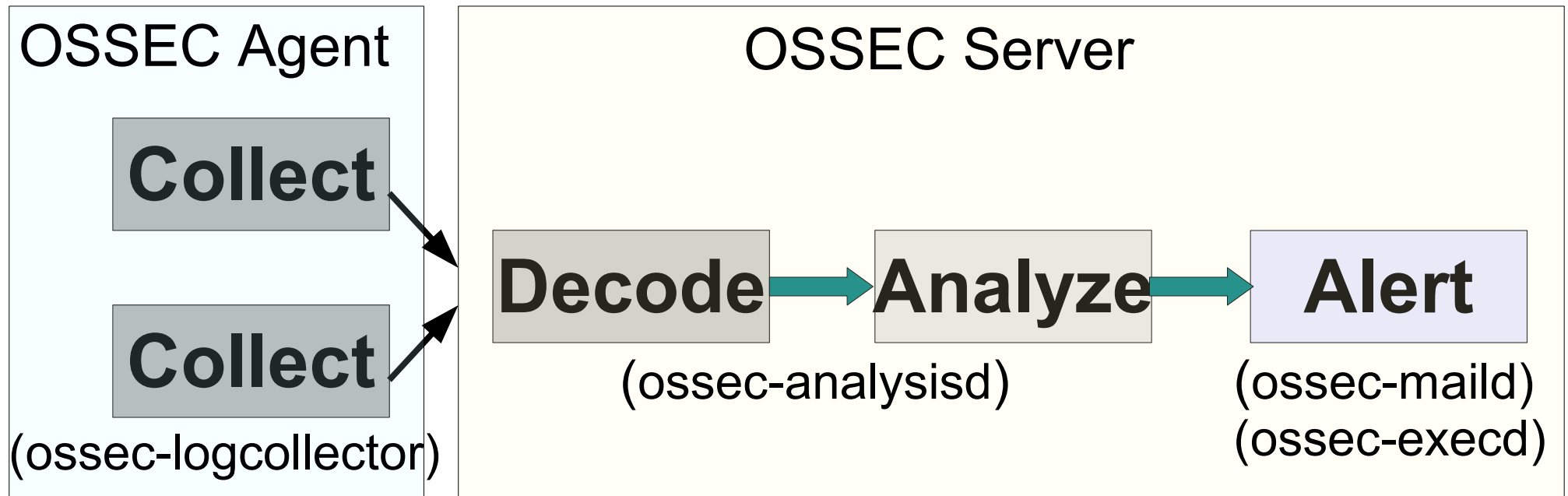
OSSEC Local



- Generic log analysis flow breakdown (for ossec local)
 - Log collecting is done by **ossec-logcollector**
 - Analysis and decoding are done by **ossec-analysisd**
 - Alerting is done by **ossec-maild**
 - Active responses are done by **ossec-execd**



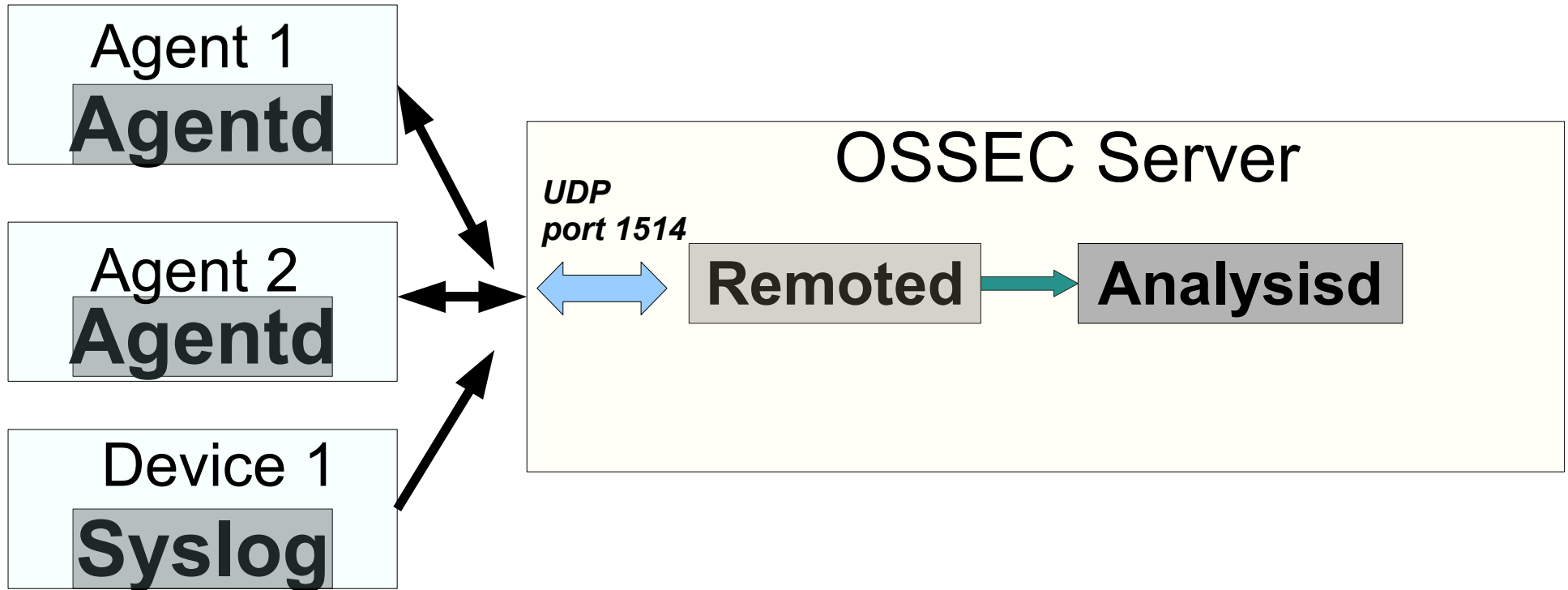
Log flow (agent/server)



- Generic log analysis flow for client/server architecture
 - Log collecting is done by **ossec-logcollector**
 - Analysis and decoding are done by **ossec-analysisd**
 - Alerting is done by **ossec-maild**
 - Active responses are done by **ossec-execd**



Network communication



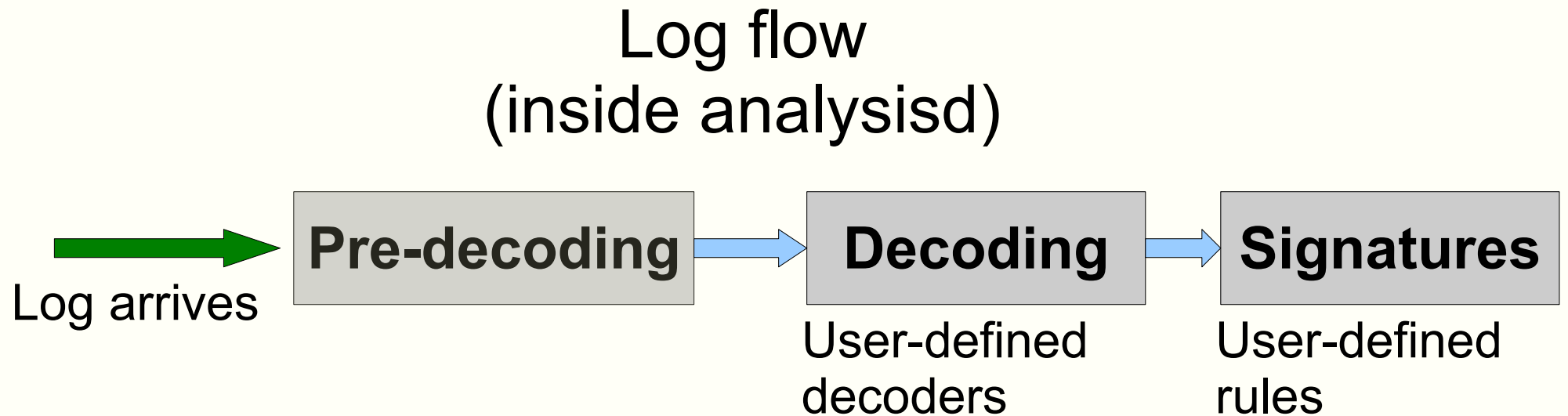
- Agent/Server network communication
 - Compressed (zlib)
 - Encrypted using pre-shared keys with blowfish
 - By default uses UDP port 1514
 - Multi-platform (Windows, Solaris, Linux, etc)



Deep into Log Analysis

- Focus now on the main process (**ossec-analysisd**)
 - *It does the log decoding and analysis*
 - *Hard worker!*
- Log **pre-decoding**
- Log **decoding**
- Log **Analysis**
- Example of alerts

Internal log flow



- Log flow inside analysisd
- Three main parts:
 - **Pre-decoding** (extracts known fields, like time, etc)
 - **Decoding** (using user-defined expressions)
 - **Signatures** (using user-defined rules)



Log pre-Decoding (1)

- Extracts generic information from logs
 - Hostname, program name and time from syslog header
 - Logs must be well formatted
- How OSSEC does it?
 - Log comes in as:
*Apr 13 13:00:01 **enigma** **syslogd**: restart*
 - How will it look like inside OSSEC?
 - time/date** -> Apr 13 13:00:01*
 - hostname** -> **enigma***
 - program_name** -> **syslogd***
 - log** -> restart*



Log pre-Decoding (2)

- Decoding of a SSHD message:

- Log comes in as:

- Apr 14 17:32:06 enigma **sshd**[1025]: Accepted password for root from 192.168.2.190 port 1618 ssh2*

- How will it look like inside OSSEC after pre-Decoding?

- time/date** -> Apr 14 17:32:06

- hostname** -> enigma

- program_name** -> **sshd**

- log** -> Accepted password for root from 192.168.2.190 port ...

Log pre-Decoding (3)

- Decoding of an ASL message (Mac users):
 - Log comes in as:
*[Time 2006.12.28 15:53:55 UTC] [Facility auth] [Sender sshd] [PID 483]
[Message error: PAM: Authentication failure for username from
192.168.0.2] [Level 3] [UID -2] [GID -2] [Host mymac]*
 - How will it look like inside OSSEC after pre-Decoding?
*time/date -> Dec 28, 2006 15:53:55
hostname -> mymac
program_name -> sshd
log -> error: PAM: Authentication failure for username from 192.168.0.2*



Log Decoding (1)

- Process to identify key information from logs
 - Most of the time you don't need to worry about it
 - OSSEC comes with hundreds of decoders by default
 - Generally we want to extract source ip, user name, id ,etc
 - User-defined list (XML) at **decoders.xml**
 - Tree structure inside OSSEC
- How a log will look like after being decoded:
 - Apr 14 17:32:06 enigma sshd[1025]: Accepted password for **root** from **192.168.2.190** port 1618 ssh2*
 - time/date** -> Apr 14 17:32:06
 - hostname** -> enigma
 - program_name** -> sshd
 - log** -> Accepted password for root from 192.168.2.190 port ...
 - srcip** -> **192.168.2.190**
 - user** -> **root**



Writing decoders 101

- Writing a decoder. What it requires?
 - Decoders are all stored at etc/decoders.xml
 - Choose a meaningful name so they can be referenced in the rules
 - Extract any relevant information that you may use in the rules
- sshd example:
 - We want to extract the user name and source ip
 - If **program name** was pre-decoded as sshd (remember pre-decoding?), try this regular expression

```
<decoder name="sshd-success">  
  <program_name>sshd</program_name>  
  <regex>^Accepted \S+ for (\S+) from (\S+) port </regex>  
  <order>user, srcip</order>  
</decoder>
```

Writing decoders 102

- Decoders guidelines
 - Decoders must have either **prematch** or **program_name**
 - **regex** is used to extract the fields
 - **order** is used to specify what each field means
 - Order can be: id, srcip, dstip, srcport, dstport, url, action, status, user, location, etc
 - Offset can be: "after_prematch" or "after_parent"

- Vsftpd example:

*Sun Jun 4 22:08:39 2006 [pid 21611] [dcid] OK LOGIN: Client
"192.168.2.10"*

```
<decoder name="vsftpd">  
  <prematch>^\w\w\w \w\w\w\s+\d+ \S+ \d+ [pid \d+] </prematch>  
  <regex offset="after_prematch">Client "(\d+.\d+.\d+.\d+)"$</regex>  
  <order>srcip</order>  
</decoder>
```

Writing decoders 103

- Grouping multiple decoders under one parent
 - Use **parent** tag to specify the parent of the decoder
 - Will create a tree structure, where the sub-decoders are only evaluated if their parent matched.
- sshd example 2:

```
<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>  
</decoder>
```




Writing decoders 103 (2)

- sshd example 3:

```
<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>  
</decoder>
```

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

Writing decoders 103 (3)

- Apache access log example:
 - We extract the srcip, id and url

192.168.2.190 - - [18/Jan/2006:13:10:06 -0500] "GET /xxx.html HTTP/1.1"
200 1732

```
<decoder name="web-accesslog">
  <type>web-log</type>
  <prematch>^\d+.\d+.\d+.\d+ </prematch>
  <regex>^(\d+.\d+.\d+.\d+) \S+ \S+ [\S+ \S\d+] </regex>
  <regex>"w+ (\S+) HTTP\S+ (\d+) </regex>
  <order>srcip, url, id</order>
</decoder>
```

Log Rules (1)

- Next step after decoding is to check the rules
 - Internally stored in a tree structure
 - User-defined XML
 - Very easy to write!
 - Allows to match based on decoded information
 - **Independent of initial log format**, because of decoders
 - OSSEC comes with more than **400 rules by default!**
- Two types of rules:
 - **Atomic** (based on a single event)
 - **Composite** (based on patterns across multiple logs)



Writing your own rules 101

- Writing your first rule. What it requires?
 - A Rule id (any integer)
 - A Level - from 0 (lowest) to 15 (highest)
 - Level 0 is ignored, not alerted at all
 - Pattern - anything from “regex”, to “srcip”, “id”, “user”, etc
- First example (simple sshd rule)
 - If log was decoded as *sshd*, generate rule “111”

```
<rule id = "111" level = "5">  
  <decoded_as>sshd</decoded_as>  
  <description>Logging every decoded sshd message</description>  
</rule>
```

Writing your own rules 102

- Second rule, for failed sshd messages
 - We will create a second rule, dependent on the first
 - Higher severity (level 7)
 - Will only be executed if the first one matches (if_sid)
 - Match is a simple pattern matching (looking for Failed pass)

```
<rule id = "111" level = "5">  
  <decoded_as>sshd</decoded_as>  
  <description>Logging every decoded sshd message</description>  
</rule>
```

```
<rule id="122" level="7">  
  <if_sid>111</if_sid>  
  <match>^Failed password</match>  
  <description>Failed password attempt</description>  
</rule>
```



Writing your own rules 103

- Using additional rule options
 - We will create a third rule, dependent on the second
 - Will only be called if the second one matches!
 - Looks if the hostname was decoded as **mainserver**
 - Looks if the **decoded IP address** is outside the network

```
<rule id="122" level="7">  
  <if_sid>111</if_sid>  
  <match>^Failed password</match>  
  <description>Failed password attempt</description>  
</rule>
```

```
<rule id="133" level="13">  
  <if_sid>122</if_sid>  
  <hostname>^mainserver</hostname>  
  <srcip>!192.168.2.0/24</srcip>  
  <description>Higher severity! Failure on the main server</description>  
</rule>
```



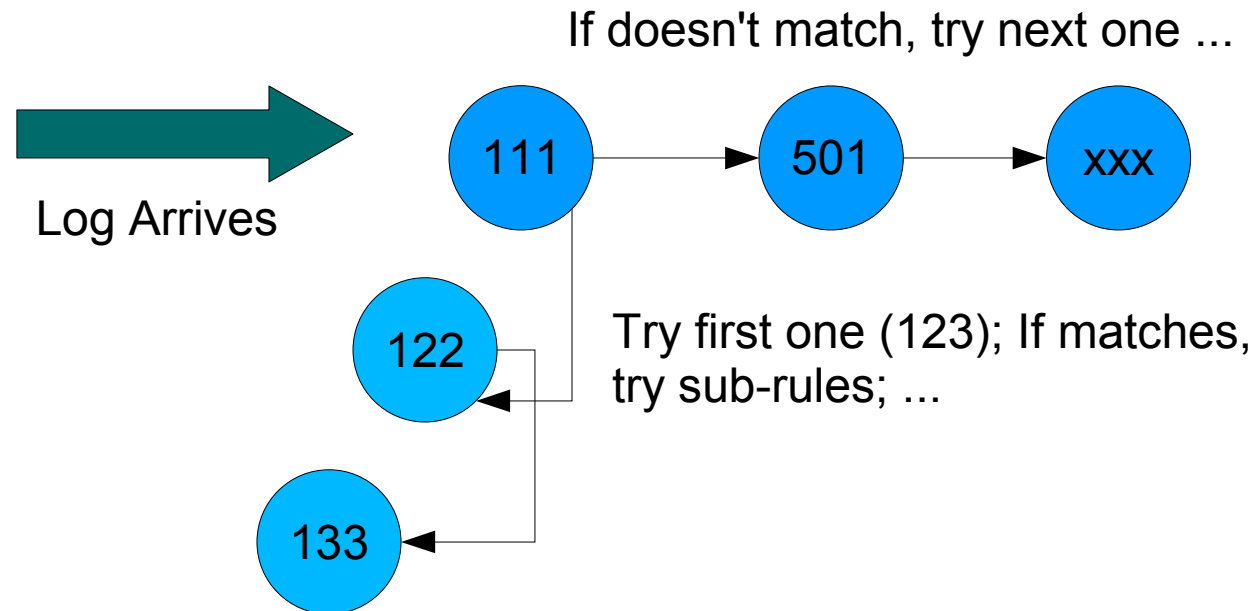
Writing your own rules 103(2)

- Rule for Apache web logs
 - We will create one generic rule for all web logs (501)
 - One sub-rule to alert on ids **4xx** or **5xx** (HTTP errors)
 - We use here the “id” tag, which is also set in the decoder

```
<rule id="501" level="3">  
  <decoded_as>web_log</decoded_as>  
  <description>Generic rule for apache logs</description>  
</rule>
```

```
<rule id="502" level="6">  
  <if_sid>501</if_sid>  
  <id>^4|^5</id>  
  <description>Log with id 4xx or 5xx</description>  
</rule>
```

Rule structure after ...



- Internal structure after first five rules.
 - Not a flat format (like most log analysis tools)!
 - **Very fast! Non-sshd messages are only checked against the first rule (111), not the sub ones**
 - **Average of only 7/8 rules per log, instead of 400 (what we have enabled by default)**



Writing your own rules 103(3)

- A few more advanced rule options
 - Rule for successful sshd logins
 - Policy-based options, based on time, day of the week, etc
 - You can use groups to classify your rules better

```
<rule id = "153" level = "5">  
  <if_sid>111</if_sid>  
  <match>Accepted password </match>  
  <description>Successful login</description>  
  <group>login_ok</group>  
</rule>
```

```
<rule id="154" level="10">  
  <if_sid>153</if_sid>  
  <time>6 pm - 8:30 am</time>  
  <description>Alert! Logins outside business hours!</description>  
  <group>login_ok,policy_violation</group>  
</rule>
```



Writing your own rules 200

- Composite rules
 - Rule for multiple failed password attempts
 - We set frequency and timeframe
 - **if_matched_sid: If we see this rule more than X times within Y seconds.**
 - **same_source_ip: If they were decoded from same IP.**

```
<rule id="133" level="7">  
  <if_sid>111</if_sid>  
  <match>^Failed password</match>  
  <description>Failed password attempt</description>  
</rule>
```

```
<rule id="1050" level="11" frequency="5" timeframe="120">  
  <if_matched_sid>133</if_matched_sid>  
  <same_source_ip />  
  <description>Multiple failed attempts from same IP!</description>  
</rule>
```

Rules in real world

- Do not modify default rules
 - They are overwritten on every upgrade
 - Use **local_rules.xml** instead (not modified during upgrade)
 - Use and abuse of if_sid, if_group (remember, classify your rules under groups), etc
 - Use an ID within the range 100000-109999 (user assigned)
- If adding support for new rules or new log formats
 - Send them to us, so we can include in ossec
 - We will assign a range ID for your rules

Rules in real world (2)

- Alerting on every authentication success outside business hours
 - Every authentication message is classified as “authentication success” (why we use if_group)
 - Add to **local_rules.xml**:

```
<rule id="100005" level="10">  
  <if_group>authentication_success</if_group>  
  <time>6 pm - 7:30 am</time>  
  <description>Login during non-business hours.</description>  
</rule>
```

Rules in real world (3)

- Changing frequency or severity of a specific rule
 - Rule 5712 alerts on SSHD brute forces after 6 failed attempts
 - To increase the frequency, just overwrite this rule with a higher value. Same applies to severity (level).
 - You can change any value from the original rule by overwriting it
 - Add to **local_rules.xml**:

```
<rule id="5712" level="10" frequency="20" overwrite="yes">  
  <if_matched_sid>5710</if_matched_sid>  
  <description>SSHD brute force trying to get access to </description>  
  <description>the system.</description>  
  <group>authentication_failures,</group>  
</rule>
```



LID Examples - Squid logs

- Rule to detect internal hosts scanning the outside
 - Useful to detect worms, malicious users, etc
 - Will fire if same internal system generates multiple 500/600 error codes on different URLs

```
<rule id="35009" level="5">  
  <id>^5|^6</id>  
  <description>Squid 500/600 error code (server error).</description>  
</rule>  
<rule id="35058" level="10" frequency="6" timeframe="240">  
  <if_matched_sid>35009</if_matched_sid>  
  <same_source_ip />  
  <different_url />  
  <description>Multiple 500/600 error codes (server error).</description>  
</rule>
```



LID Examples - Squid logs 2

- Indication of an internal compromised system:

Received From: (proxy) 10.1.2.3->/var/log/squid/access.log

Rule: 35058 fired (level 10) -> **"Multiple 500/600 error codes (server error)."**

Portion of the log(s):

```
179993 1.2.3.4 TCP_MISS/504 1430 GET http://xx.com/cgi/stats/awstats.pl
- NONE/- text/html
179504 1.2.3.4 TCP_MISS/504 1410 GET http://xx.com/awstats.pl - NONE/-
text/html
179493 1.2.3.4 TCP_MISS/504 1422 GET http://xx2.com/stats/awstats.pl -
NONE/- text/html
179494 1.2.3.4 TCP_MISS/504 1438 GET http://xx2.com/cgi-
bin/stats/awstats.pl - NONE/- text/html
179507 1.2.3.4 TCP_MISS/504 1426 GET
http://xx3.com/awstats/awstats.pl - NONE/- text/html
```

LID Examples - Web logs

- Rule to detect large URLs
 - Any URL longer than 2900 characters is very suspicious

```
<rule id="31115" level="13" maxsize="2900">  
  <if_sid>31100</if_sid>  
  <description>URL too long. Higher than allowed on most </description>  
  <description>browsers. Possible attack.</description>  
  <group>invalid_access,</group>  
</rule>
```




LID Examples - Web logs 2

- Indication of an attack detected
 - Now, what if you see that from an internal box?

OSSEC HIDS Notification.
2007 Feb 18 20:52:27

Received From: (jul) 192.168.2.0->/var/log/apache/access_log

Rule: 31115 fired (level 13) -> "URL too long. Higher than allowed on most browsers."

Portion of the log(s):

```
142.167.9.242      -      -      [18/Feb/2007:21:43:49      -0400]      "SEARCH
  \x90\xc9\xc9\xc9\xc9\xc9
\x90\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9
9\xc99\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9
\x90\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9
\x90\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9...
```



LID Examples – Snort logs

- Multiple IDS events from same source IP address

2007 May 08 14:10:58 (jul) 192.168.2.0->/var/log/snort/alert

Rule: 20152 (level 10) -> 'Multiple IDS alerts from same IP Address.'

[**] [1:648:7] SHELLCODE x86 NOOP [**][Classification: Executable code was detected] [Priority: 1] 142.167.24.154:1238 -> 192.168.2.32:80

[**] [1:648:7] SHELLCODE x86 NOOP [**][Classification: Executable code was detected] [Priority: 1] 142.167.24.154:1238 -> 192.168.2.32:80

[**] [1:648:7] SHELLCODE x86 NOOP [**][Classification: Executable code was detected] [Priority: 1] 142.167.24.154:1238 -> 192.168.2.32:80

[**] [119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [Classification: Preprocessor] 142.167.24.154:1238 -> 192.168.2.32:80

[**] [119:15:1] (http_inspect) OVERSIZE REQUEST-URI DIRECTORY [**][Classification: access to a potentially vulnerable web application] [Priority: 2] 142.167.24.154:1238 -> 192.168.2.32:80

[**] [1:1070:9] WEB-MISC WebDAV search access Classification: access to a potentially vulnerable application] 142.167.24.154:1238 -> 192.168.2.32:80



LID Examples - Auth logs

- Brute force attempts
- Not only for SSHD, but also ftpd, imapd, webmails, etc

OSSEC HIDS Notification.
2007 Feb 21 05:37:59

Received From: enigma->/var/log/authlog

Rule: 5712 fired (level 10) -> "SSHD brute force trying to get access to the system."

Feb 21 05:37:58 enigma sshd[7235]: Failed password for invalid user admin from 125.152.17.236 port 42198 ssh2

Feb 21 05:37:58 enigma sshd[14507]: Invalid user admin from 125.152.17.236

Feb 21 05:37:56 enigma sshd[10566]: Failed password for invalid user admin from 125.152.17.236 port 42132 ssh2

Feb 21 05:37:56 enigma sshd[11502]: Invalid user admin from 125.152.17.236



LID Examples - Auth logs 2

- Brute force attempts followed by a success

Rule: 5720 (level 10) -> 'Multiple SSHD authentication failures.'

Src IP: 125.192.xx.xx

Feb 11 09:31:58 wpor sshd[4565]: Failed password for root from 125.192.xx.xx port 42976 ssh2

Feb 11 09:31:58 wpor sshd[4565]: Failed password for admin from 125.192.xx.xx port 42976 ssh2

Feb 11 09:31:58 wpor sshd[4565]: Failed password for admin from 125.192.xx.xx port 42976 ssh2

Rule: 40112 (level 12) -> 'Multiple authentication failures followed by a success.'

Src IP: 125.192.xx.xx

User: admin

Feb 11 09:31:58 wpor sshd[7235]: Accepted password for admin from 125.192.xx.xx port 42198 ssh2



Conclusion

- OSSEC is very extensible and provides out of the box functionality
- Try it out and check for yourself! :)
- Lots of new features planned for the future
- Web Interface also available
- Look at our manual and FAQ for more information:
<http://www.ossec.net>
- For questions and support, subscribe to our mailing list or visit us at **#ossec** on freenode

QUESTIONS ?