

OSSIM

Correlation engine explained.

Sample two:

- Advanced features.
- Nessus & Snort correlation.

Test scenario:

- Network Worm

Dominique Karg dk@ossim.net
<http://www.ossim.net>

Index

1. Foreword	3
2. Directives	4
2.1 WHAT DO WE CALL "RISK"?	4
2.2 ALERT VS. ALARM.....	6
2.3 SNORT-NESSUS CORRELATION.....	6
2.4 DIRECTIVE NUMBERING.....	6
3. Possible Worm	12
4. Conclusion	14
Appendix A – Plugins.....	15

1. Foreword

Again we want to show some of the possibilities the ossim correlation engine offers to users concerned by false-positives and reliable attack detection. This paper extends the first correlation paper we published some months ago entitled “Correlation Engine Explained. RPC DCOM Example”.

The xml-directive syntax is being reviewed and extended within this paper but for a better overall understanding we encourage the reader to review the above mentioned correlation paper first.

New topics covered within this paper include:

- Risk assessment.
- Alerts vs. Alarms.
- Snort-Nessus correlation.
- Directive numbering.
- New directive syntax.
- New examples.

2. Directives

OSSIM's logical correlation directives are defined using xml 1.0. In order to show the possibilities we have as of today I'll try to explain the basic syntax (again and extended).

The correlation engine reads all the directives on startup in order to match individual rules or events. It's functionality resembles a logical tree consisting of „if“ and „or“ statements, joined together to provide reliable means of identifying attacks or network misbehaviour.

2.1 What do we call “Risk”?

Since we'll be covering terms such as “risk”, “asset”, “priority”, “reliability”, etc... and those terms are very important for the correct understanding of how everything fits together first of all we should make clear what we mean when we mention them.

The main formula for risk calculation would look like this:

$$\text{Risk} = (\text{Asset} * \text{Priority} * \text{Reliability}) / 10$$

Where:

- Asset (0-5).
- Priority (0-5).
- Reliability (0-10).

The resulting risk is must be between 0 and 10.

Asset

First of all you'll have to define how valuable your resources are, how important they are for you; what you're going to protect after all. Is the host storing your customer database valuable for you? Or maybe it's the cluster network that runs your 250 G5 Macs? Perhaps that 4TB disc-array containing your hardly earned porn collection?.

Whatever you choose you'll have to assign a relative value to the equipment you want to take care off.

This value is being measured between 0 and 5. We could have chosen a bigger range, let's say 0-10 or even 0-100 but seriously, 0-5 should be enough to define a “relative” value to all of your assets, let it be hosts or networks.

Priority

When we talk about priority we're talking about threat. It's the importance of the isolated attack. It has nothing to do with your equipment or environment, it only measures the relative importance of the attack.

This will become clear using a couple of examples.

1. Your unix server running samba gets attacked by the sasser worm (since this paper is about worms I'll try to use them as examples).

The attack *per se* is dangerous, it has compromised thousands of hosts and is very easy to accomplish. But... does it really matter to you? Surely not, but it's a big security hole so it'll have a high priority.

2. You're running a CVS server on an isolated network that is only accessible by your friends and has only access to the outside. Some new exploit tested by one of your friends hits it.

Again, the attack is dangerous, it could compromise your machine but surely your host is patched against that particular attack and you don't mind being a test-platform for one of your friends...

Default value: 1.

Reliability

When talking about classic risk-assessment this would be called "probability". Since it's quite difficult to determine how probable it is our network being attacked by some sort of vulnerability, we'll transform this term into something more IDS related: reliability.

Surely many of you have seen unreliable signatures on every available NIDS. A host pinging a non-live destination is able to rise hundreds of thousands spade events a day. Snort's recent http-inspect functionality for example, although good implemented needs some heavy tweaking in order to be reliable or you'll get thousands of false positives a day.

Coming back to our worm example. If a hosts connects to 5 different hosts on their own subnet using port 445, that could be a normal behaviour. Unreliable for IDS purposes. What happens if they connect to 15 hosts? We're starting to get suspicious. And what if they contact 500 different hosts in less than an hour? That's strange and the attack is getting more and more reliable.

Default value: 1.

Risk

Each of the three terms covered before are relevant for its own. But combined together, they're very useful. We are able to ignore dangerous attacks to non-

important hosts, are able to catch even the most insignificant attack against our high-profile hosts/networks, etc...

Playing with those three terms you're able to adapt your IDS (HIDS or NIDS) to your environment because only you know what's critical regarding your information security.

2.2 Alert vs. Alarm.

Reaching this point there's another thing that needs explanation. What's the difference between alerts and alarms within ossim.

I'll keep this short and easy. An alert is any event we are able to receive from any source (snort, nessus, iptables, IIS, p0f, whatever). And event usually has an assigned priority and that's it.

An alarm is an event (or x events) that have reached a risk level high enough that they deserve that name. By default it's 2 (rounded, 1.6 i.e. would mean 2).

So, all these possibilities would raise an alarm:

- An event with priority 2, reliability 2, asset 4.
- Priority 5, reliability 1, asset 4.
- ...

We try to provide some useful default priority/reliability values. We're not perfect and surely a lot of them are wrong. It's up to the user to fix that and hopefully report it back to us so we can fix it too.

2.3 Snort-Nessus cross correlation.

This one is really simple but it requires some user interaction.

First we run a Nessus scan against the hosts whose events we want to correlate. After restarting the server it knows that the X scanned hosts/networks have a number of possible vulnerabilities. As of this writing we've got 631 nessus-snort references that can be correlated.

If a snort event hits a host that has a confirmed (possible) vulnerability that matches our snort event, we simply add the priorities and reliabilities of both events getting a default value of 4 (1 +1 (default priority == 1) + 1 +1 (default reliability)). That means that for hosts or networks with assets 4 or 5 we'll get an instant alarm.

2.4 Directive numbering.

After thinking about possible directive numberings we want to establish the following directive numbering and category base, hoping to leave space for many years of directive writing:

- Generic ossim: 1-2999
- Attack correlation: 3000-5999
- Virus & Worms: 6000-8999
- Web attack correlation: 9000-11999
- DoS: 12000-14999
- Portscan/scan: 15000-17999
- Behaviour anomalies: 18000-20999
- Network abuse & error: 21000-23999
- Trojans: 24000-26999
- Miscelaneous: 27000-34999
- User contributed: 500000+

A directive starts containing three tags. A directive id (decimal, unique), a descriptive name that is shown when the directive is matched and it reaches an „alarm“ status (more on this later) and a priority level that is global to the directive.

```
<directive id="4" name="Possible Worm" priority="4">
```

Each directive has a starting rule that begins to match the directive and starts the event correlation, putting the event into backlog queue.

```
<rule type="detector" name="Rare dest connection" reliability="1"
  occurrence="1" from="ANY" to="ANY" port_from="ANY"
  port_to="25,80,135,137,139,445,1433,1434" plugin_id="1104"
  plugin_sid="ANY">
```

In this case we match anything going to the standard worm ports.

This is our starting rule, i'll explain the fields step-by-step.

- Type
-

What type of rule is this. There are two possible types as of today:

1. Detector

Detector rules are those received automatically from the agent as they are recorded. This includes snort, spade, apache, etc...

2. Monitor

Monitor rules must be queried by the server ntop data and ntop sessions.

- Name
-

The rule name shown within the event database when the level is matched.

Accepts: UTF-8 compliant string.

- Reliability

Each rule has its own reliability, determining how reliable this particular rule is within the whole attack chain.

Accepts: 0-10. Can be specified as absolute value (i.e. 7) or relative (i.e. +2 means two more than the previous level).

- Ocurrence

How many times we have to match a unique “from, to, port_from, port_to, plugin_id & plugin_sid” in order to advance one correlation level.

- Time_out

We wait a fixed amount of seconds until a rule expires and the directives lifetime is over.

- From

Source IP. There are various possible values for this field:

1. ANY

Just that, any ip address would match.

2. Dotted numerical Ipv4 (x.x.x.x)

Self explaining.

3. Comma separated Ipv4 addresses without netmask.

You can use any number of ip addresses separated by commas.

4. Using a network name.

You can use any network name defined via web.

5. Relative.

This is used to reference ip addresses from previous levels. This should be easier to understand using examples
1:SRC_IP means use the source ip referenced within the previous rule.
2:DST_IP means use the destination ip referenced two rules below as source address.

6. Negated.

You can also use negated elements. I.e.:

“!192.168.2.203,INTERNAL_NETWORK”.

If INTERNAL_NETWORK == 192.168.1.0/24 this would negate the whole class C and additionally 192.168.2.203.

- To

Source IP. There are various possible values for this field:

1. ANY

Just that, any ip address would match.

2. Dotted numerical Ipv4 (x.x.x.x)

Self explaining.

3. Comma separated Ipv4 addresses without netmask.

You can use any number of ip addresses separated by commas.

4. Using a network name.

You can use any network name defined via web.

5. Relative.

This is used to reference ip addresses from previous levels. This should be easier to understand using examples
1:SRC_IP means use the source ip referenced within the previous rule.
2:DST_IP means use the destination ip referenced two rules below as source address.

6. Negated.

You can also use negated elements. I.e.:

“!192.168.2.40,INTERNAL_NETWORK”.

If INTERNAL_NETWORK == 192.168.1.0/24 this would negate the whole class C and additionally 192.168.2.40.

The „To“ field is the field used when referencing monitor data that has no source.

Both „From“ and „To“ fields should accept input from the database in the near future. Host and Network objects are on the TODO list.

- Port_from

This can be a port number or a sequence of comma separated port numbers. ANY port can also be used.

Hint: 1:DST_PORT or 1:SRC_PORT would mean level 1 src and dest port respectively. They can be used too. (level 2 would be 2:DST_PORT for example).

- Port_to

This can be a port number or a sequence of comma separated port numbers. ANY port can also be used.

Hint: 1:DST_PORT or 1:SRC_PORT would mean level 1 src and dest port respectively. They can be used too. (level 2 would be 2:DST_PORT for example).

- Protocol
-

This can be TCP, UDP or ICMP. You can also use relative referencing like in 1:PROTOCOL, 2:PROTOCOL, etc...

- Plugin_id
-

The numerical id assigned to the referenced plugin. See appendix A.

- Plugin_sid
-

The numerical sub-id assigned to each plugins events, functions or the like. For example, plugin id 1001 (snort) references it's rules as normal plugin_sids. Plugin id 1501 (apache) uses the response codes as plugin_sid (200 OK, 404 NOT FOUND, ...)
ANY can be used too for plugin_sid.

- Condition
-

This parameter and the following three are only valid for „monitor“ and certain „detector“ type rules.

The logical condition that has to be met for the rule to match:

1. eq - Equal
2. ne - Not equal
3. lt - Lesser than
4. gt - Greater than
5. le - Lesser or equal
6. ge - Greater or equal

- Value
-

The value that has to be matched using the previous directives.

- Interval
-

This value is similar to time_out but used for „monitor“ type rules.

- Absolute
-

Determines if the provided value is absolute or relative. For example, providing 1000 as a value, gt as condition and 60 (seconds) as interval, querying ntop for HttpSentBytes would mean:

- Absolute true: Match if the host has more than 1000 http sent bytes within the next 60 seconds. Report back when (and only if) this absolute value is reached.
 - Absolute false: Match if the host shows an increase of 1000 http sent bytes within the next 60 seconds. Report back as soon as this difference is reached (if it was reached...)
- Sticky
-

A bit more difficult to explain. Take the worm rule. At the end we want to match 20000 connections involving the same source host and same destination port but we want to avoid 20000 directives from spawning so this is our little helper. Just set this to true or false depending on how you want the system to behave. If it's true, all the vars that aren't ANY or fixed (fixed means defined source or dest host, port or plugin id or sid.) are going to be made sticky so they won't spawn another directive.

In our example at level 2 there are two vars that are going to be fixed at correlation level 2: 1:SRC_IP and 1:DST_PORT. Of course plugin_id is already fixed (1104 == spade) and all the other ANY vars are still going to be ANY.

Within the worm directive we're going to see later this is:

- Sticky_different
-

Only suitable for rules with more than one occurrence. We want to make sure that the specified parameter happens X times (occurrence) and that all the occurrences are different.

Take one example. A straight-ahead port-scanning rule. Fix destination with the previous sticky and set sticky_different="1:DST_PORT". This will assure we're going to match "X occurrences" against the same hosts having X different destination ports.

In our worm rule the most important var is the DST_IP because as the number increases the reliable increases as well. Which host is going to do thousands of connections for the same port against different hosts??

- Groups
-

As sticky but involving more than one directive. If an alert matches against a directive defined within a group and the groups is set as "sticky" it won't match any other directive.

3. Possible Worm

This directive is a bit more refined than the one provided in the first correlation paper.

First, we have the start tag. This is directive number 4 so it has id=4 and it's name is „Possible Worm”. Did I tell you we want to catch worms with this one ?

```
<directive id="4" name="Possible Worm" priority="4">
```

This time our starting rule tries to match any anomalous spade connection against ports “25,80,135,137,139,445,1433,1434”, typical worm ports.

```
<rule type="detector" name="Rare dest connection" reliability="1"
  occurrence="1" from="ANY" to="ANY" port_from="ANY"
  port_to="25,80,135,137,139,445,1433,1434" plugin_id="1104"
  plugin_sid="ANY">
```

The next is a opening tag that means, after the previous rule match any of the next level rules.

```
<rules>
```

At second level our first rule tries to match 15 strange connection events from spade (plugin 1104, plugin_sid ANY) fixing source ip and destination port as stated earlier. Notice the sticky_different too, that means all of the 15 occurrences have to be different.

```
<rule type="detector" name="Too many rare connections (15) against same port"
  reliability="3" time_out="180" occurrence="15"
  from="1:SRC_IP" to="ANY" port_from="ANY" port_to="1:DST_PORT"
  plugin_id="1104" plugin_sid="ANY" sticky="true" sticky_different="DST_IP">
```

The same logic goes on until we've got 20000 occurrences or 43200 seconds (12 hours) have elapsed. For level 5 correlation we don't use sticky_different since we already have 2000 different hosts and don't want the server to keep track of that many hosts.

As before, as the correlation levels advances and we get more confident with the attack the reliability rises.

Ok, this is our final logical tree. Different levels use different colors.

Green: level 1

Yellow: level 2

Orange: level 3

Red: level 4

Bold: level 5

```
<directive id="4" name="Possible Worm" priority="4">
```

```

<rule type="detector" name="Rare dest connection" reliability="1"
occurrence="1" from="ANY" to="ANY" port_from="ANY"
port_to="25,80,135,137,139,445,1433,1434" plugin_id="1104"
plugin_sid="ANY">
  <rules>
    <rule type="detector" name="Too many rare connections (15) against same
port" reliability="3" time_out="
180" occurrence="15"
    from="1:SRC_IP" to="ANY" port_from="ANY" port_to="1:DST_PORT"
    plugin_id="1104" plugin_sid="ANY" sticky="true"
    sticky_different="DST_IP">
      <rules>
        <rule type="detector" name="Too many rare connections (300) against
same port" reliability="5" t
ime_out="1200"
        occurrence="300" from="1:SRC_IP" to="ANY" port_from="ANY"
        port_to="1:DST_PORT" plugin_id="1104" plugin_sid="ANY"
        sticky="true" sticky_different="DST_IP">
          <rules>
            <rule type="detector" name="Too many rare connections (2000) against
same port" reliability="10"
            time_out="1800" occurrence="2000" from="1:SRC_IP" to="ANY" port_from="ANY"
            port_to="1:DST_PORT" plugin_id="1104" pl
            ugin_sid="ANY" sticky="true" sticky_different="DST_IP">
              <rules>
                <rule type="detector" name="Too many rare connections
(20000) against same port" reliability="10"
                time_out="43200"
                occurrence="20000" from="1:SRC_IP" to="ANY"
                port_from="ANY"
                port_to="1:DST_PORT" plugin_id="1104" plugin_sid="ANY"
                sticky="true"/>
              </rules>
            </rule>
          </rules>
        </rule>
      </rules>
    </rule>
  </rules>
</rule>
</directive>

```

4. Conclusion

Again, this paper only covers a short part of what our engine is able to do but I think it provides a little more “in-depth” sight to everything.

The Snort-Nessus correlation example isn't being included because it only takes two rules to match as explained earlier. Please read README.Nessus within ossim's source tarball in order to get more info about that.

Appendix A – Plugins

Here is a list of plugins that can be used within directives as of today.

1001, 'snort'
1002, 'snort_tag'
1100, 'spp_portscan'
1101, 'spp_minfrag'
1102, 'http_decode'
1103, 'spp_defrag'
1104, 'spp_anomsensor'
1105, 'spp_bo'
1106, 'spp_rpc_decode'
1107, 'spp_stream2'
1108, 'spp_stream3'
1109, 'spp_telnet'
1110, 'spp_unidecode'
1111, 'spp_stream4'
1112, 'spp_arpspoof'
1113, 'spp_frag2'
1114, 'spp_fnord'
1115, 'spp_asn1'
1116, 'snort_decoder'
1117, 'spp_portscan2'
1118, 'spp_conversation'
1119, 'spp_tba'
1120, 'spp_tba2'
1121, 'spp_snmp'
1501, 'apache'
1502, 'iis'
1503, 'iptables'
1504, 'fw1'
1505, 'directive_alert'
1506, 'realsure'
1507, 'rrd_threshold'
1508, 'rrd_anomaly'
1509, 'threshold'
1510, 'cisco'
1511, 'p0f'
1512, 'arpwatch'
1513, 'prelude'
2001, 'ossim'
2002, 'arp_watch'
2003, 'p0f'
2004, 'open_nms'
2005, 'ntop'
3001, 'nessus'
3002, 'nmap'