IPFC

Staying on top of your security infrastructure

"...Anything you don’t understand is dangerous until you do understand it", Larry Niven

Alexandre Dulaunoy <adulau@conostix.com> <adulau@foo.be>

Agenda overview

- IPFC - What & Why ?
- The life of an event in an IPFC framework

IPFC - What & Why ?

- IPFC project was started because ...
  - existing commercial solutions have one - often more :-[- of the followin problems
    - Expensive - serious entrance ticket prices
    - Proprietary software - not perennial
    - Security is an afterthought (syslog over UDP, SNMP-based systems)
    - non auditable agents / central processing
    - too complex to use effectively (featuritis, GUI-itis)
    - tries to do completely unrelated tasks (should trouble ticketing/alert resolution be in a monitorin app ?)
    - supports only "pre-cooked" log analysis methods, very difficult to add your own flavour
    - some products are not bad but only focus on part of the solution
  - existing non-commercial/open/free solutions
    - often focus only on part of the solution ...
    - (syslog analysis using regexps, syslog transfer, type analysis,...)
  - MSS (Managed Security Services) software
    - non-existing or moving to classical commercial solutions

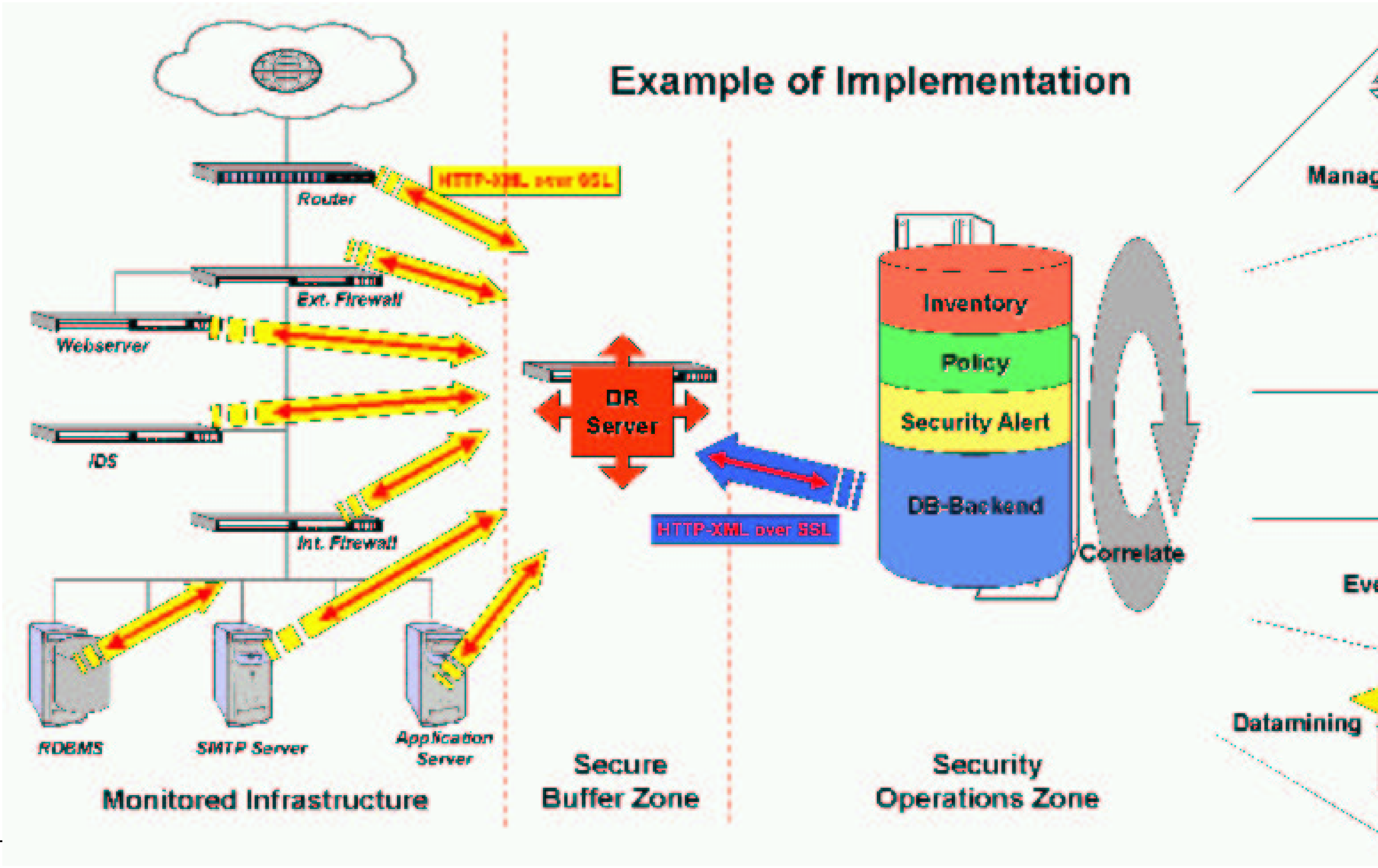
IPFC - Trigger for the project

- IPFC must be Free Software (released under the GNU General Public License)
- Limited trust between different zones in the IPFC architecture
- Security and only after features
- IPFC framework tries not to limit local policy choices, but aims to be flexible.

  (quite difficult)

  - People must be technical with a some: background

IPFC - Overview



IPFC - Overview of the framework

- Wrappers
  - Wrapper communicates with DR-Server using SSL/TLS
  - No open TCP or UDP ports on the monitored machine
  - Can be perl, shell script, C[++|#], python, ruby, or C...
  - Available on embedded system (serial back-to-back)

IPFC - Overview of the framework

DR-Server

- Temporary storage of data in transit between Wrappers and DB-Backend

- Apache based, but with
  - mod_ssl
  - mod_put
  - mod_eaccess
  - mod_gzip

- Central point in the infrastructure ... must invest in host security on this machine !
  - rsbac
  - lids
  - lsm
  - snare
  - grsecurity, PaX
  - TrustedBSD ...

IPFC - Overview of the framework

DB-Backend

- Stores events (and more) for the agents it manages
- postgresql 7.2
  - but ... easy to port to other DB (ACID is a must)

DB-Backend-daemon and DB-wrapnet

- Get events from DR-Server (SSL/TLS)
- Parse data
  - generic "transport-based" parsing - eg. syslog-lines
  - specific "event-type" parsing - eg. apache error-log
- Put parsed data in DB-Backend

IPFC - Overview of the framework

Reporter

- Gets a subset of the events out of the DB-Backend, recreates "original"/new logfile format
  - external tools to actually perform reporting (webalizer, lire, ...)

Correlator

The life of an event in IPFC

Event Generation

- I logged in on tournesol using OpenSSH.
- On tournesol, a syslog event was generated
- Feb 14 15:08:00 tournesol sshd(pam_unix)[2060]: sessior opened for user adulau by (uid=0)

Tournesol’s /etc/syslog.conf contains

- *.info;mail.none;authpriv.none;cron.none |/opt/ipfc/dev/syslog

Event Collection on the wrapper

Tournesol’s IPFC wrapper configuration file contains

```
<agent>
 <id>1</id>
 <name>tournesol</name>
 <events>
  <type>syslog</type>
  <location>pipe://opt/ipfc/dev/syslog</location>
 </events>
 <events>
  <type>null</type>
 </events>
 <status>
  <retrytime>60</retrytime>
  <parameters>osversion</parameters>
 </status>
</agent>
```

XML encapsulation

The wrapper encapsulates the event in an XML file :

```
<?xml version='1' standalone='yes'?>
<ipfc version="1" type="events">
  <data version="1" type="log"
     subtype="syslog-line" transport="syslog-line">
    <syslog-line format="base64">RmViID ... 0wKQ==</syslog-line>
    <syslog-line format="base64">RmViID ... 01MDAp</syslog-line>
  </data>
  <agent date="2002-2-14 15:08:04" id="1" wrapperid="0"
       generationid="2002-2-14 12:30:42" sequenceid="167"
       transacid="20020214150804126 54" />
</ipfc>
<ipfc-signature type="HMAC-SHA1" keyid="samplekey"
    format="base64">gBTW4X ... XgPtk=</ipfc-signature>
```

Message sending

The message gets <transacid>.events as name

The message is sent to a DR-Server using TLS/SSL PUT method in the directory /ipfc/tournesol-urls/events

- (HTTP/1.1 for persistent connections)

If message arrived OK -> also PUT <transacid>.events.ok

In DR-server logs :

```
158.64.4.14 - alex [14/Feb/2002:14:10:22 +0100] "PUT
/ipfc/tournesol/events/2002021415080412654.events HTTP/1.1" 200 89

158.64.4.14 - alex [14/Feb/2002:14:10:22 +0100] "PUT
/ipfc/tournesol/events/2002021415080412654.events.ok HTTP/1.1" 200 92
```

DB-wrapnet

DB-wrapnet gets logs from DR-server

- periodic scans to see if there are any XML files ready
- downloads only when corresponding ".ok" file exists, but no ".processed" file exists

```
<ipfc-signkeys>
  <key id="none" type="none"
       key="none" agent="1,2,3,15,17,1024,1025,1026"/>
  <key id="samplekey" type="HMAC-SHA1"
       key="a 5amPle and n0t so good key" agent="1"/>
  <key id="a1-2" type="HMAC-SHA1"
       key="tHis is A_tEs7Key!" agent="2"/>
</ipfc-signkeys>
```

and puts XML file in incoming queue for db-backend-daemon.

The life of an event in IPFC

Bird’s eye view :

- Look in incoming queues for new files
- Foreach file
  - parse the file
  - BEGIN DB transaction
  - store logfile identification information in DB
  - parse any "user" data in the file (setting specific attributes)
  - store specific data in the DB
  - if all OK => COMMIT transaction, move file to archiving location
  - if problem => ROLLBACK transaction, move file to "problems" location
- repeat until tired.

The life of an event in IPFC

```
{ ’agent’ =>
    [ {    ’generationid’ => ’2002-2-14 12:30:42’,
     ’sequenceid’ => ’167’,
     ’wrapperid’ => ’0’,
     ’id’ => ’1’,
     ’date’ => ’2002-2-14 15:08:04’,
     ’transacid’ => ’2002021415080412654’
      } ]
  ’data’ =>
    [ {    ’syslog-line’ =>
       [ { ’format’ => ’base64’,
           ’content’ => ’RmViIDE ... 0wKQ==’
         },{
           ’format’ => ’base64’,
           ’content’ => ’RmViIDE ... 01MDAp’
         } ],
     ’transport’ => ’syslog-line’,
     ’subtype’ => ’syslog-line’,
     ’version’ => ’1’,
     ’type’ => ’log’
      } ],
  ’version’ => ’1’,
  ’type’ => ’events’
};
```

Transport decoding "real-life" example

```
sub process_transport_log_line($$$$$) {
  my $dbh = shift;
  my $entry = shift;
  my $function_name = shift;
  my $table_name = shift;
  my $r_sql_hash = shift;

  my $current_entry;
  my $index = 0;
  my $rc = 1;

  while (($rc == 1) and (defined($entry->{’log-line’}->[$index]))) {
    my %sql_hash = %{$r_sql_hash};
    my $decoded_line = decode_entry($entry->{’log-line’}->[$index]);
    $sql_hash{’message’}=$decoded_line;
    $rc = &$function_name($dbh,$decoded_line,$table_name,\%sql_hash);
    $index++;
  }

  return $rc;
}
```

The life of an event in IPFC

Attributes in the database

We have unlimited attributes ... how to manage them ?

- A database table describes the valid attributes and their type
  - text,int,int8,date,boolean,numeric,null,hashtable

- A number of database tables contain the attributes themselves

Alerter

The data is in the DB-backend, now we must do something with it

Alerts can be configured in a policy-based way :

```
default
agent_type,event_type
agent_type,event_type,event_subtype
agent_type,event_type,event_subtype,event_version
agent_id,event_type
agent_id,event_type,event_subtype
agent_id,event_type,event_subtype,event_version
```

```
$ALERT{'default'}
        = "alert_default";
$ALERT{'agent_type=syslog,event_type=log'}
        = "alert_syslog_default";
$ALERT{'agent_id=1,event_type=log,event_subtype=syslog-line'}
        = "my_specific_handler_for_syslog_agent_1_logs";
```

Alerter handlers configuration example

```
my @dest_group1 = ('mail:alerts@company.com','file:/tmp/alerter');
my @dest_group2 = ('file:/tmp/alerter');

$default = { level => 1,
             alert_destinations => \@dest_group2,
             alert_summary => 'syslog alert $EVENT_ID',
             alert_message => 'Generic syslog message : $MESSAGE'};

$message{'(?i)session opened for user (.+)'} =
   {level => 3, alert_message => 'user $1 logged in'};

$message{'snmpdx: agent snmpd not responding'} =
   {level => 10,
    alert_message => 'forgot to patch, time to panic : $MESSAGE',
    alert_destinations => \@dest_group1};
```

Correlator

Correlation and alerting are based on the same principles

Think about it

- Events can generate alerts (through a policy definition)
- Alerts can be SMS, pager, email, log-lines
- ... or even new IPFC events with certain attributes dependent on the original message(s)
  - eg. a "user login succeeded/failed" event which groups NT, POP3, Unix login events,

LDAP authentication, RADIUS authentication

  - eg. a "repeated same event x times within y seconds" event
  - eg. a "repeated same user login failed event x times within y seconds"

Q&A

- Thank You.
- Q&A ?
- adulau@conostix.com, http://www.conostix.com/