

Intrusion Detection Systems Correlation: a Weapon of Mass Investigation

Pierre Chifflier and Sébastien Tricaud

INL
101/103 Bvd MacDonald
75019 Paris, France
`pierre.chifflier@inl.fr / sebastien.tricaud@inl.fr`

Abstract. This paper describes how correlation can be used to reduce false positives, discover new attacks and fight the evasion of intrusion detection systems. Events from different sources, network-based, host-based and others, are used to increase the accuracy of alerts and attacks understanding. A complete solution based on the Prelude IDS framework and the Intrusion Detection Message Exchange Format (IDMEF) standard is proposed, using Voice over IP (VoIP) as an example. Taking advantage of existing products in a hierarchical manner renders more efficient the extraction of relevant security issues. We also propose an algorithm to use correlation results to lower the amount of work needed on sensors, to concentrate on higher-level attack detection.

Key words: IDS, Hybrid IDS, IDMEF, Prelude, Correlation, Management, Assessment, Visualization

1 Introduction

The increasing number of networked machines and services has led to an increased number of activities. Separating an unauthorized activity from a regular one has become more and more complex. Attacks vary from external to internal sources, such as malicious employees or compromised machines. As a result, companies are deploying intrusion detection systems (IDS) and/or intrusion prevention systems (IPS) to detect and prevent security problems.

However, classical network-IDSs have quickly shown some limits and problems. One of the most important is false positives, that is, alerts that are triggered by normal activity. False positives require further analysis by the security administrator to be correctly classified. These are the most important problems, because they create a significant traffic of events, diverting the administrator from genuine alerts and distracting him or her from the real security effort.

False positives can be caused by different problems:

- Actions that are malicious in some environments can be considered as normal traffic in others. For example, scanning the network may be authorized for a host conducting a network discovery.
- Tweaking rulesets and signatures for IDS is a very difficult task, which may be almost impossible in some cases (for example detecting polymorphic code). It cannot even prevent from zero day attacks. Rulesets are always written after a known vulnerability.
- Encrypted data can be difficult to decode and is usually ignored.
- Some poorly-designed or complex applications can deal with unknown traffic, which will be considered as a threat by an IDS. This is often the case for load-balancers, backup servers, or monitoring tools.

- Most security devices are designed to work as standalone. They cannot effectively detect the severity of an alert, due to a lack of knowledge of its context. While a failed login attempt is a benign event in a network, simultaneous failed login attempts on several targets should be considered seriously.

An other limitation is the IDS evasion, which can be eased by several factors:

- Non-existent signature for pattern recognition
- Non-recognized protocol
- IDS focused bypassing: fragmentation, encoding, etc..

On the other hand, host-based intrusion detection systems (HIDS) are almost immune to false positives and network IDS evasion techniques, but can detect only a very limited range of alerts, directed at the monitored host.

However many other services or devices are able to provide information on security events: most firewalls have a security module, as do some network devices like routers or switches, etc. These elements provide useful information. Nevertheless they often use proprietary formats, making analysis and correlation difficult.

In this paper, we will propose a solution which combines the advantages of both systems: using HIDS to obtain precise information, and NIDS to achieve global supervision of the network. We will also propose a method of combining information from heterogeneous sources, and classical IDSs. We will use this method to see how we can detect attacks a single host could not before, explain how we can reduce the risk of IDS evasion and see attacks we could not have seen without using correlation [1]. Finally, we will see how to increase the level of accuracy associated with an alert, thus reducing the number of false positives.

2 Architecture

Our architecture is separated into nine layers.

This architecture is based on the following considerations:

1. Never modify existing sensors: we believe modifying running software is a mistake. This is difficult, must be re-done after updates, may break related software, introduces bugs, etc.,
2. Any collector should get sensor alerts using a unified data format. This is an abstraction for the analyzer to work on homogeneous information sources. While this may be difficult to combine with the previous point, we will see later that there is a way to resolve this problem,
3. The clock of each sensor must be synchronized on the same reference. This will avoid false positives or a bad correlation comprehension.

We present the following architecture from the lowest layer, which is the **source** of any **event**, to the highest, which is the **feedback** the analyst will get.

2.1 Source

A source is a device or application producing events that are of interest. For example, firewall, router or switch logs, SNMP traps fall in this category.

feedback
representation
attack
analyzer
collector
alert
sensor
event
source

Fig. 1. Layered architecture of an Intrusion Detection System

2.2 Event

An event is emitted by the source, and detected by the sensor.

2.3 Sensor

Many products have been developed in past years to detect intrusions, including host-based IDS and network-based IDS, but other components, such as firewalls, network devices (routers, switches, etc.) are security-aware. Any element capable of collecting information should be integrated in the security solution.

Sensors can be Snort, Nufw, Cisco PIX Firewall, Prelude LML, and most applications capable of logging events: Apache, OpenSSH, FreeRadius, etc.

Sensors are responsible for acquiring events from sources, usually logs or captured network traffic. Sensors will pre-process data to extract relevant information, format them, and possibly generate an alert. In our architecture, sensor alerts are sent to the collector.

Sensors work on a continuous flow of information in real-time. This means the level of analysis and formatting must be kept at the strict minimum. Otherwise the sensor will be subject to problems if the traffic increases, which may be voluntary in case of an attack.

2.4 Alert

An alert can be emitted after the sensor has performed an analysis of the incoming event, and has matched some criteria.

2.5 Collector

This element is in charge of collecting raw data from sources, and converting it to the unified format if needed. The conversion must be even faster than the sensors, because collectors will generally be in charge of several sensors at a time.

At this point, the input format can vary a lot between sources. Rules must be written for transformation. Hopefully, once a rule is written, there are only a few changes to be done when software implementations used by sensors are upgraded to new versions.

If the sensor knows how to produce alerts directly in the unified format, then it just needs to be written onto the storage layer.

Unlike sensors, collectors must be aware of the underlying storage method. Given that the amount of data can be very large, the storage used is likely to be a database.

2.6 Analyzer

The analyzer digests information to the operator. Sensor events are retrieved in the standard format. Those events can be grouped by similarity, to suppress duplicates or aggregate related events. Alerts are processed by the analyzer to alter their impact/severity, or to temporarily ignore them.

It will eventually trigger alerts, warn the operator, or store data for the representation layer. This part relies on the accuracy of the previous layers.

For example, in the Prelude architecture, the manager is the storage component and deals with analysis. To reduce the load of managers and improve scalability, managers can be distributed hierarchically: each lower level manager is in charge of a set of sensors, the upper layer is in charge of the underlying managers, etc. Each level of manager can correlate events, to present only representative events to the upper manager. However, all events from the lower layer must be kept to be able to correlate events, and to be able to investigate further if a problem is suspected.

2.7 Attack

The event is called attack after the analyzer has confirmed its harmful intent.

2.8 Representation

Representation displays alerts graphically to the operator. It is a major part in the solution, because it helps for post-event analysis. Because too much information at this point will make the solution barely usable to the operator, false positives must be eliminated first.



Fig. 2. Desktop representation of a correlated event

2.9 Feedback

From all events that have been filtered previously, feedback is used to reintroduce data for either taking counter measures or adding information for correlation.

3 IDMEF

3.1 Introduction

IDMEF stands for Intrusion Detection Message Exchange Format. The goal of IDMEF is to have a generic language for event description used by IDS components to interact with each other. It helps building a full description of each event. Being an object-oriented model, IDMEF is extensible using objects aggregation and subclassing.

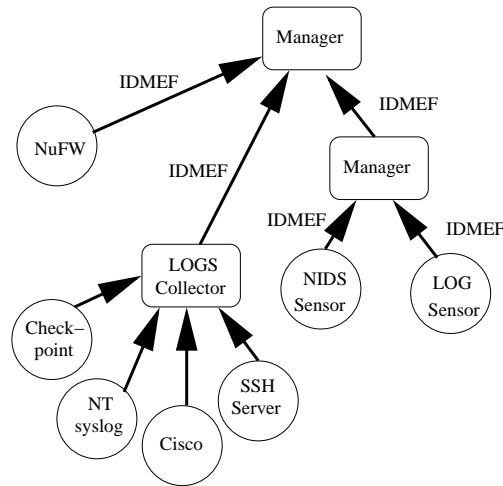


Fig. 3. IDMEF components architecture

The most obvious place of IDMEF in a network, is in a tree model, where sensors report events to a higher node, such as in figure below:

In this model, IDMEF agnostic components report to a log collector that is IDMEF aware and connected to the manager.

On the very top, the leading manager stores data for further analysis, or warns the security manager if the alert is critical enough.

IDMEF is useful to help programmers represent an attack event in an IDS engine, and is a good source to define information that will be stored.

Security components use IDMEF as a standard format to share information.

IDMEF is an XML structured document that has been standardized by the IETF as RFC 4765[2]. Such a format permits easy searching, storage and exchange of information.

One of the common problems is the multiple ways of doing the same thing. Syslog is a good example of this: there is no standard way for logging. A failed SSH login attempt will produce something like:

```
Apr  5 23:42:01 localhost sshd[9286]: Failed password for nemard_jean \
      from 192.168.0.100 port 57318 ssh2
```

Linux PAM logs the same event as

```
Apr  5 23:43:49 localhost su[9333]: FAILED su for nemard_jean by s.clave
```

IDMEF unifies the representation of this two events.

From these lines, one can extract the following information:

- Failed login attempt
- Target user
- Target machine
- Source: IP address for the ssh login attempt, original user for local su process.
- Date and Time

In IDMEF vocabulary, the failed login attempt is the **classification text**. **Target** and **Source** fill their respective object name. Date and Time are collected in the **DetectTime** object.

Figure 4 represents how a mix of those two lines could be modeled.

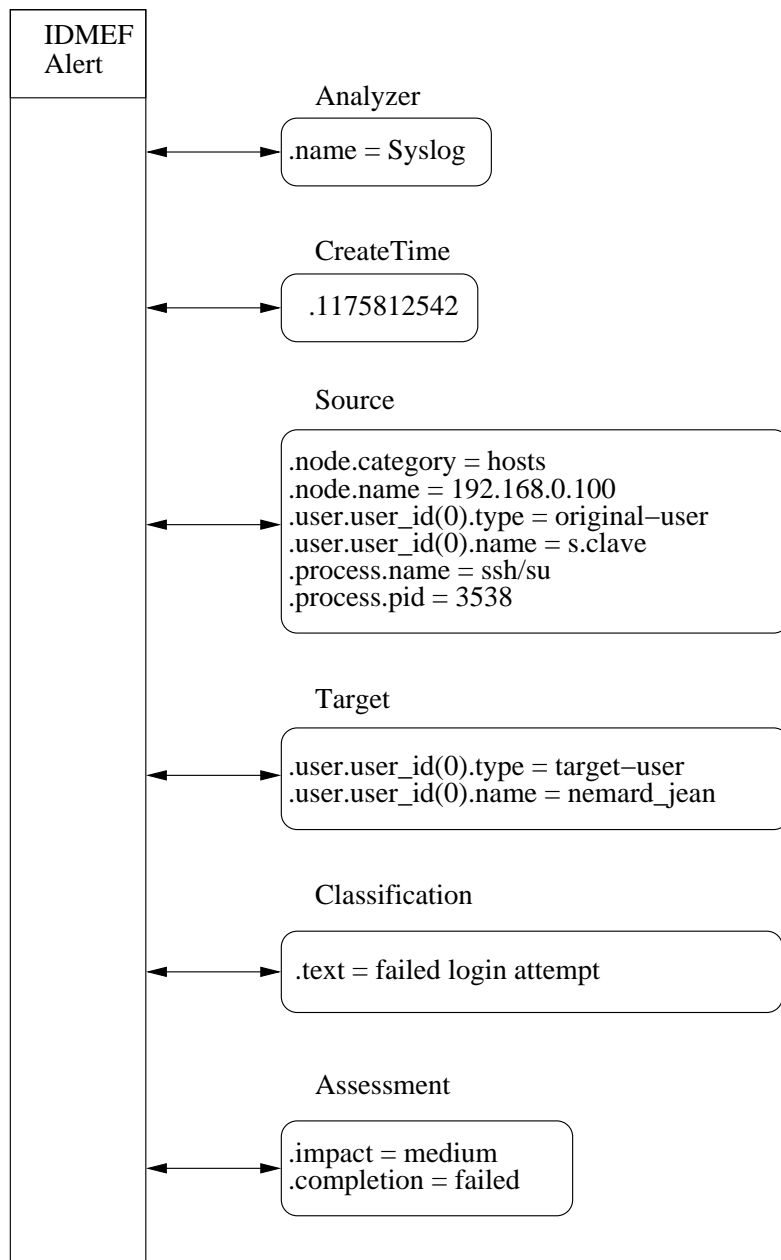


Fig. 4. IDMEF alert example

3.2 Problems Addressed by the Data Model

IDMEF has been designed to address five problems when representing an event:

- **Disparity in alert representation:** alert information is inherently heterogeneous. IDMEF offers an object-oriented model to fit the different needs for any sensor alert. For example, Snort adds the data payload to the IDMEF message.
- **Intrusion detection environments:** the meaning of the alert classes varies from one sensor to another. IDMEF model is flexible enough to integrate contrasting information using the support classes.
- **Analyzer capabilities:** the data model defines extensions that allow carrying both simple and complex alerts.
- **Operating environments:** the Operating System or Network alter how alerts should be considered. Information can be added to appropriate classes if relevant for the analyzer.
- **Commercial vendor objectives:** non-technical information can be added, for commercial purposes.

IDMEF is the key element to exchange data uniformly within a security architecture. Using tools to transform any kind of specific security component event into IDMEF is a step towards unification. Sending IDMEF data to an IDMEF collector such as Prelude Manager, or any product able to gather IDMEF data securely is a prerequisite for correlation.

4 Prelude IDS

We are going to introduce Prelude IDS, which is an IDMEF powered Intrusion Detection System, that can integrate third party components to read and/or deliver data to managers.

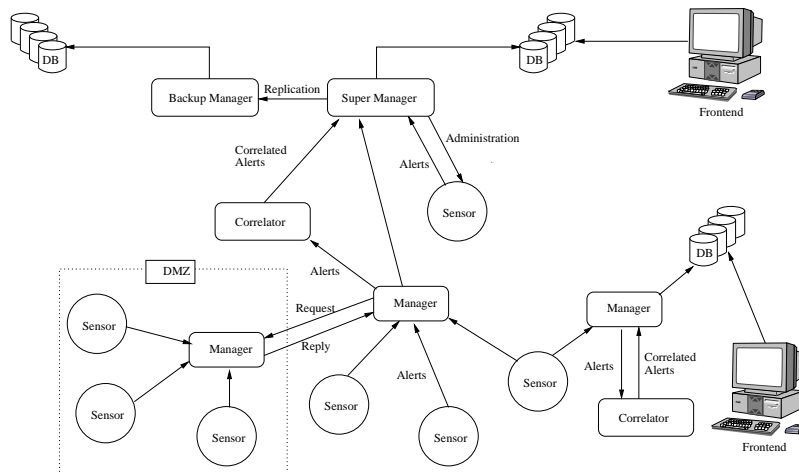


Fig. 5. Deployed architecture of the Prelude framework

Figure 5 shows how to deploy a typical prelude installation for large networks. A set of sensors is deployed on important points: host-based sensors on servers, network-based sensors on network strategic points, like connections between each subnet.

Each sensor is connected to a manager, using a trusted connection. This is particularly important to avoid attacks against the detection system itself, by reporting fake events to the manager to overload it, or to hide real attacks in a flood of false positives. The manager is in charge of:

- collecting alerts from sensors
- converting alerts in IDMEF format if needed
- storing alerts in a database

Managers are disposed hierarchically: standard managers are connected to a set of sensors. These managers can be inter-connected, and are connected to the super-manager, which will be the central point of the analysis network, acting as the authority.

It is very important point to note, is that this architecture allows the deployment of sensors in an isolated subnet, for example a DMZ, without compromising the security policy: in this case, the manager will collect information and will relay them to another manager, without having to allow connections from the DMZ to the internal network. This configuration is similar to a proxy, and is called a reverse relay.

Because the super-manager is the critical point of the architecture, it can be connected to backup manager to replicate alerts.

Finally, front-ends can be connected on databases, so the operator can extract information, reports, and analysis from stored alerts without interfering with the sensors and the managers.

From earlier sections, two major problems are going to appear:

- Scalability
- False positives

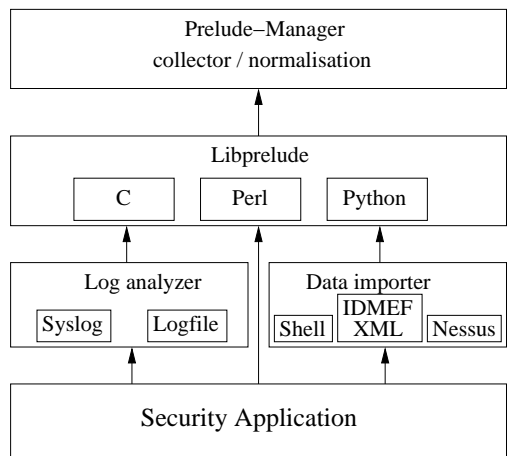


Fig. 6. Implementation architecture of the Prelude framework

Figure 6 shows the Prelude architecture. It is divided in four layers with a bottom up approach. The first layer is the **security application**. This security application can use libprelude directly to report in IDMEF format, or generate a log/event that is then imported into the Prelude framework. Using libprelude directly can be very convenient since it reduces the cost of developing the alert infrastructure for the software.

The second layer is the **log analyzer**, the application reading and extracting useful data from logs gathered using software like syslog-ng. Prelude offers this application

natively under the name of **Prelude LML**, LML for Log Monitory Lackey. There is also an IDMEF data importer, which differs from syslog aggregation. It simply parses output from sensors to make it an IDMEF message. As an example, it can extract Nessus reports.

The third layer, which is the main component in the Prelude architecture, is **libprelude**. Libprelude provides the unified framework for sensors to talk to managers, managers to administrate sensors, poll events from managers etc..

The fourth layer is the **Manager and Correlation** layer. It collects data from all those information sources and events and stores them in a database.

Going back to two problems identified, we are going to see how Prelude can be used to solve them.

Scalability is needed to digest the information collected. If there are too many sensors, the alert rate is going to be so high, that the database will be busy writing the precious information.

The proposed solution has a very good scalability factor, due to the fact that components are organized in a hierarchical way. To extend the current architecture, one only needs to add a manager and a set of sensors: the manager will handle events, discharging the upper layer from the first-part analysis.

False positives are usually generated by NIDS. Since host based IDS do not generate a lot, we can greatly reduce their number by correlating both sensors.

5 Alert Correlation

Alert correlation is the method combining various security related events to get a picture of what is really happening. Correlation creates a new alert by extracting information from sensor alerts.

For example, consider the following event:

```
User login failed with an invalid user
```

It triggers a medium security impact. This event is about a failed login attempt by a user that failed to remember his or her password. Now if ten failures happen in a short period of time, this is most likely a brute force attack.

5.1 Preliminary Steps to Correlation

To correlate efficiently, alerts should first pass two steps: **eliminating duplicates** and **factorizing alerts**.

Eliminating duplicates will suppress identical or similar alerts. A typical example is the detection of the same event by two similar sensors.

A probabilistic approach to detect similar values is explained by A. Valdes and K. Skinner [3]. Their approach provides a unified mathematical framework for matching similar but not identical alerts, where the minimum degree of match required to fuse alerts is controlled by a single configurable parameter. The overall similarity is weighted by a specifiable expectation of similarity.

Each alert is mapped in one of the following super-event: invalid, privilege violation, user subversion, denial of service, probe, access violation, integrity violation, system environment corruption, user environment corruption, asset distress, suspicious usage, connection violation, binary subversion and action logged.

Factorizing alerts will reduce the number of alerts to one correlation alert. This correlation alert summarizes information contained in alerts and adjusts its properties.

In [1], H. Debar and A. Wespi propose an algorithm to relate data acquired by the unification of IDS events in order to have a more condensed view of the security issues raised by intrusion-detection systems. Their algorithm helps the Tivoli Enterprise Console (TEC) to address a better attack understanding in the following areas: alert flooding, context, false positives and IDS deployment scalability.

5.2 Correlation

Correlation detects an alert sequence reported by one or more sensors.

From the analysis of this sequence, a new alert can be produced to represent its global meaning. This alert is called a correlated alert.

The combination of several heterogeneous sensors increases the accuracy of the attack comprehension.

The following points are relevant when analysing an alert:

- **IP source and destination:** linking the incoming traffic with the outgoing is important and usually not considered by NIDS. IP address is a useful source to understand relationships between different alerts and to link them in the same sequence. For example, a known bad source or destination IP address can influence the current alert severity.
- **Application used:** a known weak service, such as FTP, could have been sniffed to retrieve the password and then relate to successful logins on other services, then use this account to get higher privileges. That can be used to build up an attack scheme and fix the problem.
- **Normal behavior:** sudden changes in regular activity is suspicious.
- **Activity time:** if the office work hours are from 7:00am to 7:00pm, then a login outside this time range triggers a so-called login-time correlation alert.
- **Company context (Service):** the security policy of the company may restrict access to a particular set of servers/data/applications.
- **Third party logs:** upon reception of a critical alert, a specific connection becomes more interesting. Other connections from the same source become part of the analyzed data. This helps understanding not only the attack, but also the path taken.
- **Vulnerabilities:** particular sequence of alerts may be interpreted as a successful attack. This can be used to discover zero day attacks.
- **Other:** any event that can be interesting in the analysis that is not written above.

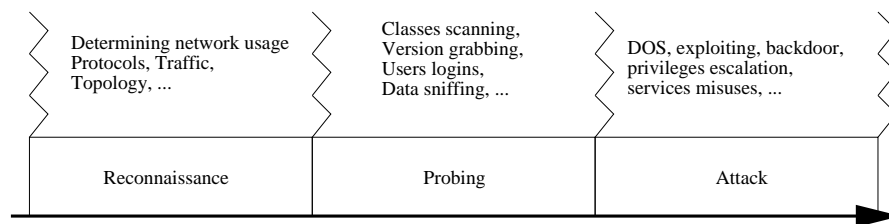


Fig. 7. Typical attack progression

We separate correlation in two distinct analysis levels: simple and global correlation. This is similar to the description of micro and macro correlation defined in [4].

5.3 Simple Correlation

A sequence of alerts targeted on the same service or host can be grouped to produce a correlation alert. The analysis of this sequence does not need additional information on attack context.

For example, a sequence of failed login attempts on a SSH server will be correlated into a brute force attack.

5.4 Global Correlation

Simple correlation is not efficient against constructed attacks, because the attacker will target different services and hosts, and will distribute attacks on a wide time range.

Global correlation considers the context of all events that has lead to the current attack and tries to find similarities, to build up an attack sequence.

The idea of chaining events is to know if a followed attack path is successful or not. For example, consider the three following events:

- Event A: a vulnerability scan or a probe from a specific IP address.
- Event B: a successful connection from the same source IP address to an internal host afterwards.
- Event C: a successful connection from the same internal host to the same external IP address afterwards.

Usually, when only one of these events is reported, there is a high probability of a false positive. Global correlation presents the three events as a logical sequence that could not be produced by a false positive.

Chaining those events is a fairly strong indicator that an attack against an internal host has been successful.

5.5 Correlation with Prelude IDS

Prelude improves the global security by analyzing events from different sources, and logging them in a standard format for security events, using IDMEF [2].

Prelude handles correlation with the program called Prelude Correlator (PC). It is driven by rulesets that match a particular IDMEF Classification using Perl Compatible Regular Expressions (PCRE). The Prelude Correlator status is currently in development.

In the Prelude architecture, sensors must be registered to the manager and communicate securely. The PC is seen as a regular sensor. A sensor is registered with read-write access to IDMEF and/or the administrative console.

Because IDMEF unifies any event that is given to the manager engine, IDMEF read access to poll alerts is required for collecting alerts.

Simple Correlation Example : the first thing to consider is the event that triggered the alert. In our case, we want to correlate a brute-force attack against our ssh servers. Below is the Prelude LML ruleset that generated a **invalid user login** event:

```
regex=(Illegal|Invalid) user (\S+) from ([\d\.]+); \  
classification.text=User login failed with an invalid user; \  
id=1904; \  
revision=1; \  
analyzer(0).name=sshd; \  
analyzer(0).manufacturer=OpenSSH; \  

```

```

analyzer(0).class=Remote Login; \
assessment.impact.severity=medium; \
assessment.impact.completion=failed; \
assessment.impact.type=user; \
assessment.impact.description=Someone tried to login with
                                the invalid user '$2' from $3; \
source(0).node.address(0).category=ipv4-addr; \
source(0).node.address(0).address=$3; \
source(0).service.iana_protocol_name=tcp; \
source(0).service.iana_protocol_number=6; \
target(0).service.port=22; \
target(0).service.name=ssh; \
target(0).service.iana_protocol_name=tcp; \
target(0).service.iana_protocol_number=6; \
target(0).user.category=os-device; \
target(0).user.user_id(0).type=target-user; \
target(0).user.user_id(0).name=$2; \
last

```

The ruleset format is very close to the IDMEF structure. Variables such as \$1 will retrieve data and set the value according to the field position in the regular expression value.

This ruleset triggers a MEDIUM alert impact. Those alerts are likely to appear quite often. The operator will then configure the sensors to ignore this information. This will hide real attacks.

Correlation adds a level of information where those alerts are grouped in one correlation alert. This not only makes the job easier for the operator, but also improves the comprehension of the attack occurring.

This correlation alert is produced by the following PC ruleset:

```

# Detect brute force attempt by user
# This rule looks for all classifications that match login or authentication
# attempts, and detects when they exceed a certain threshold.

pattern = classification.text: [Ll]ogin[Aa]uthentication; \
        assessment.impact.completion: failed; \
        target(*).user.user_id(*).name: (.+); \
        messageid: (.+); \
        analyzer(*).analyzerid: (.*

new_context = BRUTE_USER_$1[*]; expire: 120; threshold: 30;
reset_timer = $BRUTE_USER_$1[*];

$BRUTE_USER_$1[*] += \
    source(>>) = source; \
    target(>>) = target; \
    correlation_alert.alertid(>>).alertid = $2; \
    correlation_alert.alertid(-1).analyzerid = $3[-1]

for $user in $1 {
    check_correlation = $BRUTE_USER_$(user)

    $BRUTE_USER_$(user) += \

```

```

        classification.text = Brute force attack;          \
        correlation_alert.name = Multiple failed login;   \
        assessment.impact.severity = high;              \
        assessment.impact.description = Multiple failed attempts
            have been made to login to a user account; \
        additional_data(>>).data = ignore_atomic_event

    alert = $BRUTE_USER_$(user);
    destroy_context = $BRUTE_USER_$(user)
}

```

When a company has configured all its important applications or devices to log information, this can create a huge quantity of data. A majority of logs events are not security related while some can be relatively significant.

This model filters relevant alerts to avoid most of the log file analysis, which is a difficult and time-consuming task. It can even be impossible when the number of objects to monitor becomes important.

Most tools and products have not been created to cooperate, and many of them use vendor-specific extensions. A wrong, yet often-used approach, is to try to patch these applications to modify their log format. This is wrong, because it requires each different used application to be changed, involves a large amount of work, and needs to be repeated after each update.

Finally, monitoring the security of a network always tends to create false positives, which can be very difficult to filter. This creates significant number of alerts, dissimulating the real problems in a big flow of useless events.

The next advance in security monitoring is to use human knowledge to automatically analyze all events, and correlate them to distinguish normal activity from real threats.

5.6 Adaptive Intrusion Detection Algorithm (AIDA)

Once an incident has been detected and reported, the information is stored for later use. Yet this information could also be used for several objectives:

- Monitor a specific host after a signature of a successful attack has been detected,
- Gather more information on the source or destination of the attack. After an alert has been reported, the manager can send commands to sensors (not specifically the sensor reporting the alert) to increase the level of logging concerning a host. For example, this information will be stored to allow forensic analysis,
- Possibly, prepare a response to block traffic or isolate a specific host.

Preparing an incident response is a difficult step, because it requires a high level of trust in the alert report. If the system issues alerts too easily, a malicious attacker will use it to run a Denial of Service on the response component.

The algorithm presented here is still in development, and is based on the latest evolutions of Prelude IDS and the IDMEF format.

One major objective of this algorithm is to **increase the alert accuracy factor**. It is based on the previous architecture to correlate events from different sources, and modify the score according to the correlation results. This can help the reduction of false positives by setting a threshold high enough to send alerts only when the alert is confirmed or when the associated risk is high.

It is an important point to note that we do not rely on the correlator only: if an alert is issued by a sensor with a sufficient level or accuracy, the algorithm takes the

appropriate measures directly. Those measures are either commands to pilot the sensor or intensify the alert trust score.

Unlike Adaptive Learner for Alert Classification (ALAC) [5], our algorithm does not rely only on the analysis part, but also on the configuration of the sensors.

AIDA algorithm relies on the following hypothesis:

1. Input data must use a common format. The IDMEF format is recommended.
2. Each alert, in addition to the standard severity and impact, must have a Trust Score (TS):

$$TS = \text{severity of the alert} \times \text{accuracy of the alert}$$

3. The correlator and the manager must be configured to have thresholds on each alert category.
4. TS for each alert can be configured differently on each sensor
5. Managers must be able to send commands to sensors.

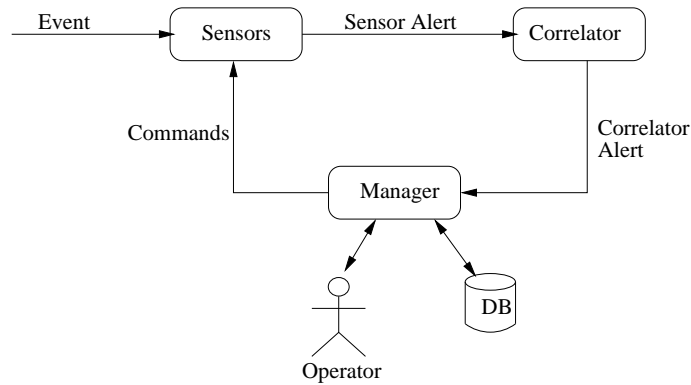


Fig. 8. Adaptive Intrusion Detection Algorithm (AIDA)

The alert is received and treated as usual, by the correlator and the manager. However, the manager can decide to make different actions:

- if the score is important (higher than threshold), then an alert is immediately raised for the operator
- if the score is not important, but the alert belongs to the appropriate category, the manager issues a command to the sensors, ordering them to increase the monitoring level for the host, for a limited period of time. The time limitation is essential to ensure the general stability of the system.
- if the alert is treated as usual, it is archived for later use by the correlator or this algorithm.

The role of the correlator appears to be even more important here: after receiving alerts, its role is to use the correlator to adjust the Trust Score accordingly: if the alert is a duplicate, then the score is not updated. If the alert has been detected on different targets, then the score will be adjusted to reflect an increased accuracy.

Before applying correlation to Session Initiation Protocol (SIP) attacks, we introduce the most classical attacks on Voice over IP (VoIP). SIP is a signalling protocol

which is used to establish the VoIP communication between two or more users. It comes along with the Real-Time Protocol (RTP), which includes the media stream.

6 Classical Voice over IP attacks

Today's enterprise telecommunication is switching to IP based technologies. VoIP has chosen UDP as a core transport protocol to benefit from fast transmission of data and is as such vulnerable to IP security issues. And in the end, the Session Initiation Protocol was created as a H323 replacement, with the goal of having a simple human readable protocol just like HTTP or SMTP.

VoIP security is extensively described in [6], which provides a good overview of the general attack scenarios on a VoIP architecture, and presents many tools (including Prelude IDS) for securing VoIP services.

Because¹ it² is³ pretty⁴ easy⁵ to⁶ find⁷ references⁸ to⁹ SIP¹⁰ vulnerabilities¹¹, this paper will not expand on it. SIP attacks will be explained and categorized.

Any SIP attack fits in one of the following six categories:

- **Denial of Service:** When a service or person is unreachable or interrupted.
- **SPIT:** SPIT is the equivalent of SPAM transposed into the Voice over IP world. While it is almost nonexistent now, this is a threat that can be fought in a way equivalent to SPAM. With subtle differences due to SIP nature different from SMTP.
- **Fraud:** Fraud is the art of using a service that requires paying where a malicious user bypasses this restriction.
- **Eavesdropping:** When a third party is intercepting a communication, then it is doing eavesdropping. This is a RTP issue, SIP knows only some basics that can let you know whether the communication is encrypted or not.
- **Spoofing:** When you are not talking to the person you want to.
- **Fuzzing:** Injected crafted SIP payload.
- **Replay:** When an attacker replays a previously sent request to alter data (ie. media proxy, ..) and eavesdrops on it.

6.1 Session Initiation Protocol

SIP is a standard (RFC 3261 [7]) for media signalization. It is responsible for creating, modifying and terminating sessions with one or more participants. It was designed as a replacement for H.323 to be simple, text-based and transport independent.

The following is a SIP payload that is sent when one wants to call `grosnaindesbois@example.com`:

¹ <http://www.cert.org/advisories/CA-2003-06.html>
² <http://www.sans.org/newsletters/cva/vol2.8.php>
³ <http://ipcommunications.tmcnet.com/hot-topics/security/articles/2552-lucent-plugs-sip-vulnerability.htm>
⁴ http://www.cisco.com/en/US/products/products_security_advisory09186a00806e0b9f.shtml
⁵ <http://labs.musecurity.com/advisories/MU-200703-01.txt>
⁶ <http://www.securiteam.com/securitynews/5GP020KLFU.html>
⁷ <https://www.kb.cert.org/vuls/id/438176>
⁸ http://www.blackberry.com/btsc/articles/225/KB12700.f.SAL_Public.html
⁹ <http://secunia.com/advisories/9674/>
¹⁰ <http://bugs.digium.com/view.php?id=9313>
¹¹ <http://nvd.nist.gov/nvd.cfm?cvename=CVE-2007-3438>

```

INVITE sip:grosnaindesbois@example.com SIP/2.0
From: BAR <sip:nonalaeroport@example.com>;tag=deadbeaf31337
To: <sip:grosnaindesbois@example.com>
Call-ID: a8673ca6f981417810f668aa72d14243@example.com
Cseq: 20 PUBLISH
Contact: <sip:nonalaeroport@example.com>
Max_forwards: 70
User Agent: yaua, Yet Another UA v.0.42
Content-Type: application/sdp
Subject: STR Test
Expires: 7200
Content-Length: 0

```

Fig. 9. Example of SIP message.

With a simple email address, when registered, you can locate a user and communicate. Official IANA ports are 5059 (SIP Directory Service), 5060 (SIP) and 5061 (SIP-TLS).

SIP provides a set of features, such as instant messaging or voice mail. It has actually a lot of extensions and is currently (July 2007) subject to twenty internet drafts and forty-nine requests for comments. Because of this jungle, a Draft “A Hitchhiker’s Guide to the Session Initiation Protocol” is being written.

Being transport agnostic requires SIP to perform the routing as well. Which is convenient to keep the same signaling path without relying on Internet Protocol (IP).

When it comes to security there is a lot to worry about. SIP has several attack vectors: it is mostly used on top of the UDP transport protocol, stack implementations are having hard time following submitted corrections, drafts and extensions¹², implementations are also subject to attacks and SIP is a complex protocol.

This section shows a quick overview of SIP attacks, in order to understand later the methodology used to prevent those attacks.

Method misuses SIP methods describe the kind of message you are dealing with. To enumerate a few:

- **REGISTER**: Register users location
- **INVITE**: Initiate/Change a session
- **ACK**: Acknowledgement
- **CANCEL**: Cancel a running request
- **BYE**: Finish a session
- **OPTIONS**: Send any stuff and kitchen sink
- **MESSAGE**: Instant messaging

Each method is either a request or a reply function. SIP proxy, caller or callee answer differently according to implementation and method type.

Replies work almost like HTTP. They are codes that are categorized:

- **1xx**: Temporary / Informative
- **2xx**: Success
- **3xx**: Redirection
- **4xx**: Client error
- **5xx**: Server error
- **6xx**: Global error

¹² For example, SIP Presence is still a draft (<http://www.ietf.org/internet-drafts/draft-ietf-simple-partial-notify-09.txt>) and can be subject to contact list removal etc.

User enumeration Depending on one's proxy, one can enumerate users of a directory if you send a request like:

```
OPTIONS sip:schmilblickman@voip.wengo.fr SIP/2.0
Via: SIP/2.0/UDP 192.168.10.225:5060;rport;branch=z9hG4515ter42261
Route: <sip:gateway-wengo:5060;lr>
To: tricaud <sip:schmilblickman@voip.wengo.fr>;tag=4245715392
Max-Forwards: 70
User-Agent: phapi/eXosip/0.2.0
Content-Length: 0
```

If the OPTIONS method is activated on the remote proxy, a success will reply a "200 OK", meaning the user is present in the directory. If not, you are likely to have:

```
SIP/2.0 404 Not Found
```

There can be many financial implications using this, such as industrial spying (sending an OPTIONS request each month to know if somebody is still working in the company).

Denial of Service attacks Denial of Service attacks (DoS) are quite common. This can be fairly easy to run:

- Most physical IP phones are weak, can be easily fuzzed, don't validate data, have a poor SIP stack, ...
- Same can apply to softphones.
- A VoIP platform involves DNS, databases, several servers and software, which creates a bigger dependency than the regular PSTN phone network.
- The SIP protocol is not mature enough: for example, the CANCEL method cancels any waiting request if you use the same Call-ID, From, To and CSeq fields.
- Any machine on the signalling or media path running out of service will stop the communication.

Eavesdropping Eavesdropping is easier in the IP world: most communications are crystal clear to any machine on the signalling or media path. Few are encrypted: among RTP solutions, SRTP is a good choice, ZRTP improves SRTP but is still a draft. Some SIP requests do not need authentication, some authentication methods are weak and lots of voice mail only require the Caller ID to redirect to the right box, which eases mailbox access spoofing.

Also, since SIP is easily extensible and sometimes lacks features or is not mature enough, implementors write extensions. This can be overridden to access a specific user settings, authentication (Remote-Party-ID and P-Asserted-Identity, ...)

SQL injections In VoIP architectures, the contact information is usually stored in a database to get account parameters. This can be account password, credits left, etc.

Let's see a very trivial example.

If the following query is performed in the database:

```
SELECT password FROM voip_users WHERE sipfrom = 'frog@example.com';
```

The `sipfrom` field is fed from the SIP message doing the REGISTER.

It is extracted by the SIP parser, which should set the variable between 'sip:' and ' '. Now take the following REGISTER:

```
REGISTER sip:frog@example.com SIP/2.0
Via: SIP/2.0/UDP 192.168.1.1:5060;rport;branch=123456789
From: Croa Croa <sip:frog@example.com>;tag=123456789
To: <sip:frog@example.com>
Call-ID: 000001@192.168.1.1
CSeq: 20 REGISTER
Accept: application/sdp
Content-Length: 0
```

Suppose we replace the request line with something more convenient:

```
REGISTER sip:<frog@example.com AND 1=(UPDATE voip_users SET password = 'cracked' \
WHERE sipfrom = 'frog@example.com');> SIP/2.0
```

When the `sipfrom` will be processed, if there is no real protection, then the password of the user 'frog@example.com' will be updated, allowing the cracker to use his or her account.

Fuzzing Fuzzing is a functional test of a protocol or file in order to produce an unexpected behavior of a program. It is similar to the boundary value analysis technique¹³. This is the best method used today for vulnerability discovery. Though well known, this technique is often neglected by vendors. The first publication about fuzzing has been written back to 1990 ([8]), and has given good results against many known applications ([9], [10]). Fuzzing is so efficient, that most tested IP phones shut down, leading to a denial of service and sometimes reveals remote holes. It has also been subject of multiple vulnerabilities in SIP implementations [11].

Fuzzing methods vary: it can be as simple as locating the length value in a packet header and altering the payload not to reflect this. One of the most efficient way of doing it is also to randomize payload alteration.

The leading project in SIP fuzzing vulnerabilities discovery is PROTOS. As stated on their web site: *“Many of the implementations available for evaluation failed to perform in a robust manner under the test. Some failures had information security implications, and should be considered as vulnerabilities”*.

SPIT SPAM for VoIP is almost nonexistent at the time but can ruin a provider reputation. Tools must be ready to face this.

7 Visualizing Correlation

When it comes to IDS correlation, one important factor is what data we are correlating: this importance factor varies according to the attack timeline, the source and/or destination address or the attack level.

The visualization brings the need for a human analysis. Even if the automatic aspect of data correlation is preferred, it is good to help the system administrator with graphics so that we can use a correlator of choice: the analysis engine which is between the chair and keyboard.

It is easier for the human brain to recognize patterns than digging within hundreds of lines of data. A good graph will instantaneously reveal the issues. This is why it is often preferred to use a graphical representation of attacks. Several techniques can help to improve the reading time: the parallel coordinate plot, graphviz data with

¹³ http://en.wikipedia.org/wiki/Boundary_value_analysis

nodes organized by colors in two or three dimensions, treemap for a view of the most important data, starplot with several variables etc.

Choosing the right data and the right scale must be done meticulously, being the key for the analysis.

Once this is done, we can define a confidence level to each input data and recognize attack patterns.

We do not graph random data. Our methodology is to extract the most interesting IDMEF fields in order to help humans to quickly understand attacks that would have taken a longer time to see otherwise.

7.1 Methodology

An alert, as described in the IDMEF format, contains many interesting parameters:

- Source
- Destination
- Impact
- Completion
- Attack vector
- Sensor/Analyzer type
- ...

Traditional representation methods cannot use many parameters, on the result would be unreadable. However, these parameters are interesting depending on the situation. For example, if we consider only three parameters (source, destination, attack vector):

- A ssh scan will link a unique source to many destinations, using the same attack vector,
- Some brute-force attacks (for ex. Nessus) will link a source to a destination using many attack vectors,
- A Distributed Denial of Service (DDoS) will link many sources to a unique destination.

If every parameter we want to evaluate is associated to a value, then we have n -dimensional vectors to plot.

In order to represent such data, we choose to use **parallel coordinates** plot [12], to transform the problem of representing a multi-dimensional dataset into a 2-D representation. Parallel coordinates plot technique is very powerful and well suited for visualizing network data, which is also extensively exploited in [13] and [14]. The idea is to display values on separate columns (corresponding to parameters), and draw lines between values to show their relation. A single alert is thus represented as a polyline (a series of segments) whose endpoints are the values for each parameter.

The first parameter will often be the time the alert was detected by the analyzer, so that the order of the events can also be extracted from the graph. It also allows to differentiate alerts with the same parameters, because they are very unlikely to have the same detection time.

We obtain a two-dimensional graph representing all alerts during a time range, such as figure 10. See Appendix B for the code you can use along with the Prelude Manager to produce this graph. The number of common segments joining two points can be interpreted as a value characterizing the importance of the relation.

Yet, we can use some knowledge on the alerts to interpret the graph:

- The time line needs to be represented

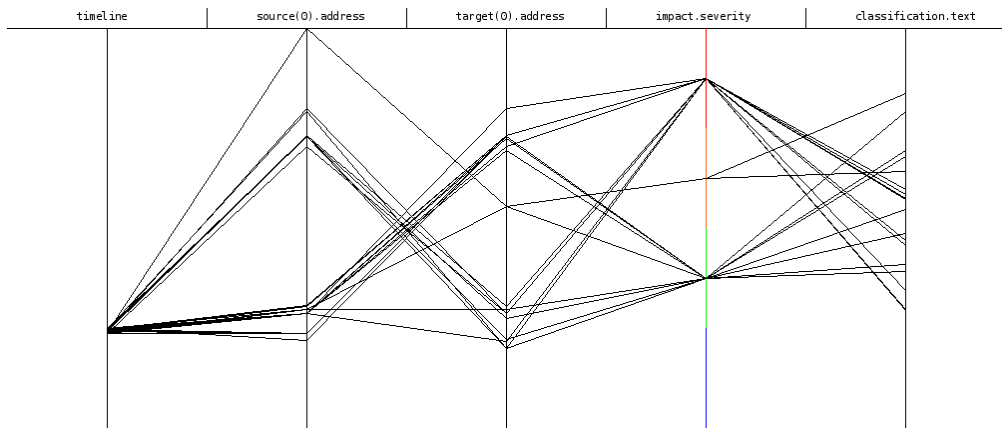


Fig. 10. Typical result of IDMEF graph

- Several identical segments joining two points will represent the fact that alerts contain identical parameters (for ex, in case of the repetition of the same attack)
- All precedent examples (ssh scan, DDoS, brute-force attack) will fit nicely in this representation

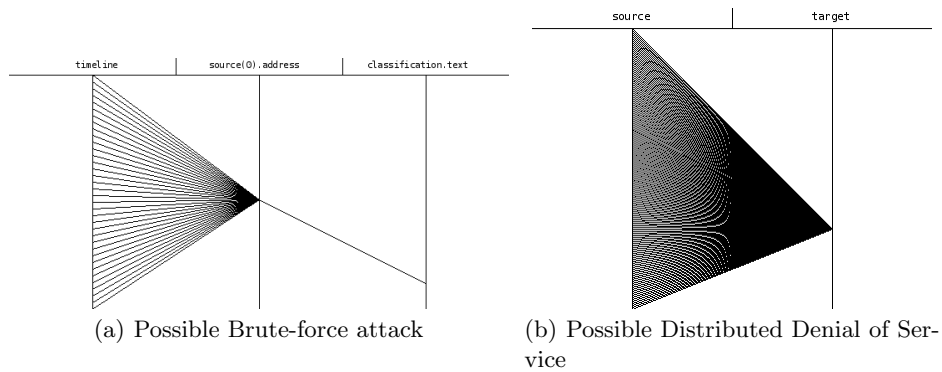


Fig. 11. Attack Graphs

For example, figure 11 shows the representation of typical alerts: a repeated attack from the same source to the same target, and a multiple-source to unique target attack. The former can be, for example, a brute-force attack on a SSH server, while the latter is more likely a Distributed Denial of Service attack, or for a scan which source is faked.

The interpretation of the graph is left to the operator, since it requires some knowledge of the environment of the alerts. However, it can be a great help to extract important information on the distribution of alerts, if some hosts are regularly appearing as sources or targets, etc.

7.2 Evasion

Depending on the parameters chosen for the graph, the representation may not be very useful. This will typically be the case when the attacker is trying to evade the detection, like in figure 12.

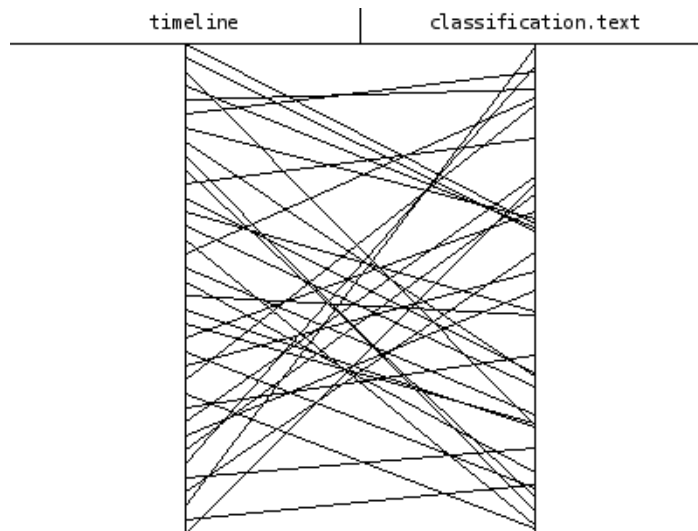


Fig. 12. Evasion graph

In this case, attacks are detected on different targets, so that every parameter of the attack is changing, on a long period. This is a typical low and slow evasion. To detect it, one must be able to correlate alerts on a long time period, as we will see in the next chapter, for voice over IP.

8 Correlation illustrated by Voice over IP

This chapter explains, from the previously defined Voice over IP attacks, how IDS correlation can help to discover hardly detectable attacks from a single IDS.

Two examples are taken:

- **Low and slow** user password guess attack detection function
- **Fraud** detection

8.1 Low and slow user password guess attack detection function

A real-time analyzer can hardly see those, because it would require a huge amount of memory to load every attack pattern and take CPU time to see if each alert is coming from a previously seen source. This is one of the easiest evasion technique : no need to alter packets, just delay the attack.

This example shows the case of a user with several authentication failure requests.

Name	Type	OS	Node Name
prelude-lml	Prelude LML 0.9.10.1	Linux 2.6.18-028stab045	Asterisk server
Classification		Source	Target
9 x Wrong password (failed)		192.168.33.180	127.0.0.

Fig. 13. Asterisk alerts view in Prewikka

To detect this attack, Asterisk rulesets have been written and are now part of the Prelude IDS project, see Appendix A for details. Figure 13 shows how the Prewikka interface display those alert in the default view with a default source-target only correlation.

When isolating an authentication failure, while doing the analysis, even though there is a high chance the security analyst would not be interested by this event, that could easily be called a false positive: there is a really high factor this would not trap the analyst attention.

Figure 14 shows how the analyst would see those events when not time correlated.

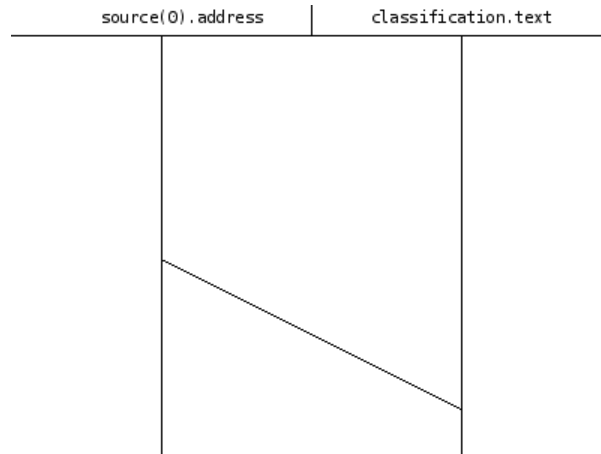


Fig. 14. IDMEF graph without timeline

The source remains the same and the conducted attack is also the same, there is no visible variation on this graph.

Now in figure 15, if the time factor is introduced, the graph is more exploitable.

Another possibility would be to use another information (color, third-dimension, or adding a number) to indicate the number of attacks represented on the same segment.

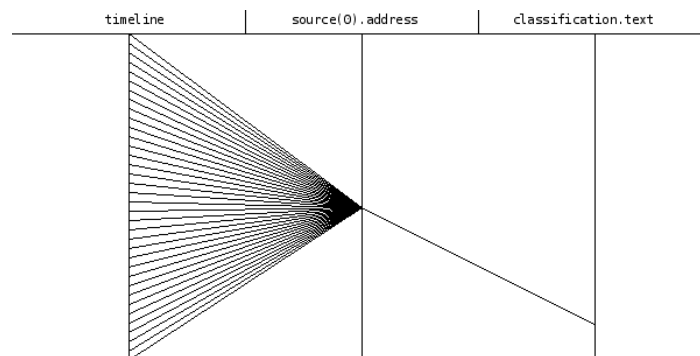


Fig. 15. IDMEF graph with timeline

This shows the same attack from the same source has been conducted, but spread on a twenty-four hours timeline. It would not be a surprise to guess the account might be under a brute-force password detection attack.

This is exactly this difference from the old fashion way to visualize alerts that the IDS correlator must trigger. Because there is a very low chance a user will do an authentication failure on a repeated interval, the alert becomes relevant to the security analyst.

To perform the correlation work, this algorithm is used internally to the correlator:

Prerequisites

- On one side we have a DATABASE the correlator can dig
- On the other side, we have the CORRELATOR connected to the manager

```
var \$max_failure_trigger = 10

# For each event, the correlator writes the 'idmef' object
for each CORRELATOR(&idmef):
  \$sourceip = idmef.Get('alert.source(0).node.address(0).address')
  \$alertid = idmef.alert_id

  count = DATABASE('SELECT COUNT(*) FROM Alerts WHERE Alerts.CreateTime > LAST_DAY(CURDATE())')

  if count > \$max_failure_trigger:
    idmef_alert = idmef.New()
    idmef_alert = idmef.Set('alert.classification.text', 'Low and slow user password guess')
    idmef_alert.send()
```

8.2 Fraud detection

Attack Fraud is hot topic in VoIP. VoIP services are quite complex, and are mostly software based, coming with their flaws. Figure 16 shows a very simple set of classical VoIP services that handle the registration, SIP proxy, billing and a gateway for another SIP network.

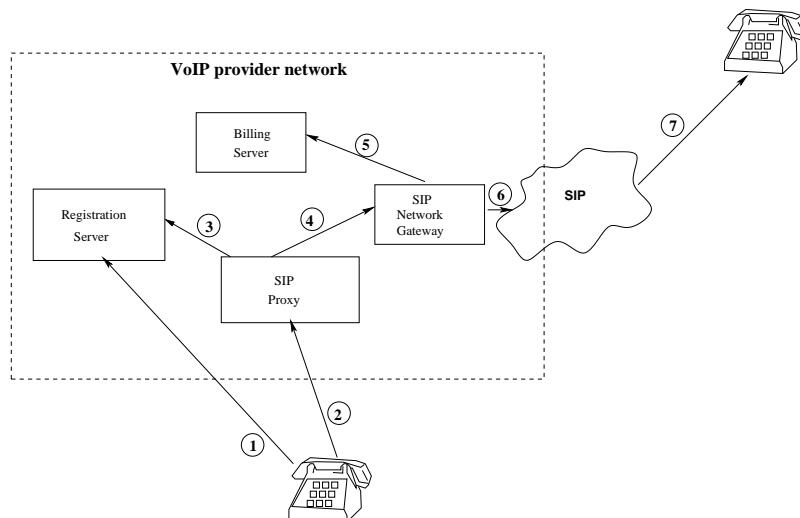


Fig. 16. VoIP registration and call to another network

- **Step 1:** The caller registers to the **registration server**
- **Step 2:** The caller calls a SIP user from an other network
- **Step 3:** The SIP proxy checks if the user is registered
- **Step 4:** The call is routed to the **SIP gateway**
- **Step 5:** The **SIP gateway** checks if the caller has enough credit to call the other network user
- **Step 6:** The **SIP gateway** calls the **SIP network**
- **Step 7:** The **SIP network** locates the callee phone and the call can start

In order to detect fraud attacks, a slightly different approach is proposed here. Each machine running the VoIP service provide a set of applications to manage the call. From the user registration, the billing and the call routing either within the VoIP network or on the other SIP network. Markers are put on each used service and compared in the end, to see if they all match. If they do not, it becomes a problem.

In this example, there is a malicious user that uses an attack which is not part of any IDS signature. For signature based IDS, this is a false negative.

The attack targets the billing server. Four kind of attacks have been described in the *Billing Attacks on SIP-Based VoIP Systems* paper [15]. InviteReplay, FakeBusy, ByeDelay and ByeDrop billing attacks. **InviteReplay** will replay intercepted INVITE messages exploiting the lack of anti-replay protections in SIP stacks. **FakeBusy** requires to hijack the target VoIP subscriber and then allow the control of the call duration, billing the source VoIP subscriber a wanted time for the attacker. **ByeDelay** extends the calling time by intercepting the BYE request, sending to the called and to the callee the 200 OK request to keep the RTP stream between the two Man in The Middle (MiTM) hosts. The original BYE will be sent at a chosen time. **ByeDrop** is similar to ByeDelay. Instead of delaying the BYE request, it will drop it and keep the RTP flow between the two MiTM hosts.

In this example, the correlation engine will focus on ByeDrop billing attack discovery.

Figure 17 shows how the message flows are exchanged for the attack to work.

For the ByeDrop billing attack, BYE SIP requests are intercepted by the two MiTM hosts and a 200 OK reply to the SIP agents to tear down the call. For both users the communication ended properly. However, the two MiTM do not forward the BYE request to the VoIP platform and they keep sending random RTP traffic to each other.

Detection To detect this attack, we record a set of network level data that are unique from one machine to an other.

A passive connection profiler such as Sancp¹⁴ is deployed on the VoIP network to monitor changes in the RTP stream.

Since VoIP and in particular SIP uses several techniques to bypass NAT, the IP Identification and the source port fields are used to monitor RTP communication changes.

The IP Identification field is a 16 bits value by the sender to aid in assembling the fragments of a datagram. This is set on each IP packet sent for the time the datagram or fragment could be alive in the internet.

This way we will know if the RTP stream is emitted from a different source.

When Sancp sees an IP Identification field gap, it will send an alert to the Prelude Manager. Because there is a risk of false positive, the correlation is based on this change at a given interval. When the IP Identifier gap is too large, the RTP connection is teared down. To avoid a DOS attack exploiting the correlation engine and have an attack to cut this RTP connection by simply spoofing the IP Identifier, we send a SIP OPTIONS packet to the source user to force an authentication within five minutes.

¹⁴ <http://metre.net/sanpc.html>

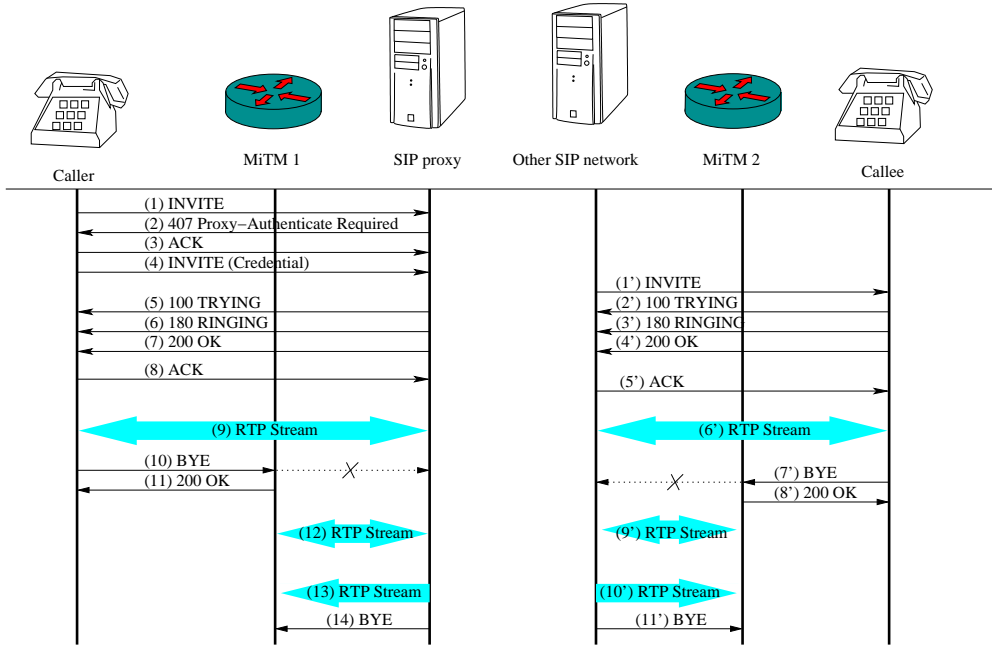


Fig. 17. ByeDrop attack

9 Conclusion

In this paper, we have shown how correlation can be used to reduce false positive rates which are inherent to classical network-based IDS. We have proposed a comprehensive and scalable architecture based on the Prelude IDS framework which is able to integrate events from heterogeneous sources.

Using a standard format for handling security alert is a required step to be able to develop higher level algorithms, such as correlation. IDMEF is an extensible and well-defined proposed standard, which is particularly adapted for these needs.

It is also important to note that the quality of rules used in event correlation makes a huge difference in intrusion detection proficiency.

We have proposed an algorithm using post-analysis data to control the sensors, in an adaptive way, relative to the threats. This algorithm is particularly useful to maintain a standard level of network monitoring, which will be increased relatively to the alerts reported by the sensors. The manager component can then archive data, which will be used for forensics analysis.

This algorithm also tries to increase reliability of alerts, so they can be used for automatic incident response. This is still work in progress, and represents a practical use of the recent advances in intrusion detection systems.

We have also shown that using a good representation for alerts is critical: by combining the strength of human interpretation with the power of machines and correlation algorithms, we end up with a result which is greater than just the sum of parts.

Visualization can also help lower the expertise level required for novices, speed up the work for experts, and communicate results to customers and managers.

Future works include:

- A greater emphasis on correlated events
- More sophisticated alert fusion

– High speed data mining

We believe IDMEF standard, as its implementation in Prelude IDS solves a great issue in the ability to make existing components communicate to correlate alerts for reducing false positives, see IDS evasions and discover new attacks. The visualization part is also an area of emerging interest and further research should conduct on better IDS events visualization.

A Asterisk ruleset for Prelude LML

```
#Nov 29 09:44:15 NOTICE[23701] chan_sip.c: Registration from '<sip:dmc@asterisk-server>'
failed for '192.168.33.180' - Wrong password
regex=Registration from '<sip:(\S*)>' failed for '(\S+)' - (.*)$; \
classification.text=$3; \
id=6000; \
revision=1; \
analyzer(0).name=Asterisk; \
analyzer(0).manufacturer=Digium; \
analyzer(0).class=Private Branch Exchange; \
assessment.impact.severity=medium; \
assessment.impact.completion=failed; \
assessment.impact.type=user; \
assessment.impact.description=SIP user could not be registered by the SIP server; \
source(0).node.address(0).address=$2; \
target(0).service.name=sip; \
target(0).user.user_id(0).type=original-user; \
target(0).user.user_id(0).name=$1; \
last;
```

Out commits regarding Prelude LML VoIP modifications are available at the following URL: <https://trac.prelude-ids.org/changeset/10077>

B Prelude Manager pool IDMEF grapher

```
#!/usr/bin/python
#
# Graph IDMEF Messages
#

import time
import sys

import gd
import PreludeEasy

#
# GD Constants
#

timeline_x = 100
severity_x = 300
classification_x = 500

header_size_y = 20
image_width = 800
image_height = 400+header_size_y

severity_high_y = 50 + header_size_y
severity_medium_y = 150 + header_size_y
severity_low_y = 250 + header_size_y
severity_info_y = 350 + header_size_y

im = gd.image((image_width, image_height))

client = PreludeEasy.Client("PoolingTest")
client.Init()

client.PoolInit("192.168.33.215", 1)
```

```

gd_init()

def plot_timeline():
    t = time.localtime()
    hour = t[3]
    minute = t[4]
    second = t[5]

    hour_factor = 400.0 / 24.0
    mn_factor = hour_factor / 60.0

    hour_y = hour_factor * hour
    mn_y = mn_factor * minute

    plot_y = hour_y + mn_y

    return int(plot_y)

def unique_alert_number(ClassificationText):
    number = 0

    for c in ClassificationText:
        number += ord(c)

    return number

def classification_text_pos(text):
    classification_factor = 400.0 / 10000.0
    nb = unique_alert_number(text)
    print "Unique number = " + str(nb)

    c_y = classification_factor * nb
    print "Position C-Y = " + str(c_y)

    return int(c_y + header_size_y)

def handle_alert(idmef):

    classificationtext = idmef.Get("alert.classification.text")

    severity = idmef.Get("alert.assessment.impact.severity")
    if severity:
        time_y = plot_timeline() + header_size_y
        print "Time Y = " + str(time_y)
        if severity == "high":
            im.line((timeline_x, time_y),(severity_x, severity_high_y), black)
            if classificationtext:
                c_y = classification_text_pos(classificationtext)
                im.line((severity_x, severity_high_y),(classification_x, c_y), black)
        if severity == "medium":
            im.line((timeline_x, time_y),(severity_x, severity_medium_y), black)
            if classificationtext:
                c_y = classification_text_pos(classificationtext)
                im.line((severity_x, severity_medium_y),(classification_x, c_y), black)
        if severity == "low":
            im.line((timeline_x, time_y),(severity_x, severity_low_y), black)
            if classificationtext:
                c_y = classification_text_pos(classificationtext)
                im.line((severity_x, severity_low_y),(classification_x, c_y), black)
        if severity == "info":
            im.line((timeline_x, time_y),(severity_x, severity_info_y), black)
            if classificationtext:
                c_y = classification_text_pos(classificationtext)
                im.line((severity_x, severity_info_y),(classification_x, c_y), black)

    im.writePng("idmef-graph.png")

while 1:
    idmef = client.ReadIDMEF(1)
    if idmef:
        handle_alert(idmef)

```

time.sleep(1)

References

1. Debar, H., Wespi, A.: Aggregation and correlation of intrusion-detection alerts. In Lee, W., Mé, L., Wespi, A., eds.: *Recent Advances in Intrusion Detection*. Volume 2212 of *Lecture Notes in Computer Science.*, Springer (2001) 85–103
2. Debar, H., Curry, D., Feinstein, B.: RFC4765: The intrusion detection message exchange format (idmef) (March 2007) Status: EXPERIMENTAL.
3. Valdes, A., Skinner, K.: Probabilistic alert correlation. *Lecture Notes in Computer Science* **2212** (2001) 54–??
4. Schultz, E.: Intrusion detection event correlation: Approaches, benefits and pitfalls. (March 2007) CERIAS Security Seminars.
5. Pietraszek, T.: Using adaptive alert classification to reduce false positives in intrusion detection
6. Sisalem, D., Ehlert, S., Geneiatakis, D., Kambourakis, G., Dagiuklas, T., Markl, J., Rokos, M., Botron, O., Rodriguez, J., Liu, J.: Towards a secure and reliable voip infrastructure. (May 2005) <http://www.snocer.org/Paper/COOP-005892-SNOCER-D2-1.pdf>.
7. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: RFC3261: Sip: Session initiation protocol (2002)
8. Miller, B.P., Fredriksen, L., So, B.: An empirical study of the reliability of UNIX utilities. *Communications of the Association for Computing Machinery* **33**(12) (1990) 32–44
9. Forrester, J.E., Miller, B.P.: An empirical study of the robustness of windows NT applications using random testing. 59–68
10. Barton P. Miller, G.C., Moore, F.: An empirical study of the robustness of macos applications using random testing. (July 2006) *First International Workshop on Random Testing*.
11. CERT: Advisory ca-2003-06 multiple vulnerabilities in implementations of the session initiation protocol (sip). (2003)
12. Inselberg, A., Avidan, T.: The automated multidimensional detective. In: *INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization*, Washington, DC, USA, IEEE Computer Society (1999) 112
13. Marchette, D.J.: *Computer Intrusion Detection and Network Monitoring: A Statistical Viewpoint*. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2001)
14. Conti, G.: *Security Data Visualization*. No Starch Press, San Francisco, CA, USA (2007)
15. Zhang, R., Wang, X., Yang, X., Jiang, X.: Billing attacks on sip-based voip systems. (2007) http://www.usenix.org/events/woot07/tech/full_papers/zhang/zhang.pdf.