OSSIM

Open Source Security Information Management

General System Description

Wednesday, 26 November 2003 **Version: 0.18**

Team

The current development team for the project is:

Dominique Karg, <dk@ossim.net>	Technical Director
Jesús D. Muñoz, <jesusd@ossim.net>	Documentation
David Gil, <dgil@ossim.net>	Development
Fabio Ospitia <fot@ossim.net>	Development
Santiago González, <sgonzalez@ossim.net>	Software Integration
Julio Casal, <**jcasal@ossim.net**>	Coordinator

Index

Foreword

The goal of OSSIM is to fill a gap that we see in our daily needs as security professionals.

Considering the important technological advances of recent years that have made tools with capacities such as those of IDS available to us, we are surprised that it is so complex from a security standpoint to obtain a snapshot of a network as well as information with a level of abstraction that allows practical and manageable monitoring.

In developing this project, our primary intent is to rectify this situation with a functionality that can be summarized in a single word:

CORRELATION

Correlation means the ability to view all events in all systems in one place and in the same format, and from this priveleged vantage point compare and process the information, thereby allowing us to improve detection capabilities, prioritize events according to the context in which they occurred, and monitor the security situation of our network.

The idea of correlation is also implicit in the vision of our project in the sense of bundling and integrating products. Within the general framework of OSSIM, we want to include a number of magnificent products developed in recent years that create new possibilities when their functionalities are interrelated.

In developing OSSIM, we encountered new needs that helped us improve the precision of our system and develop the functionality that is now a core element of OSSIM:

RISK ASSESSMENT

In each case, in order to decide whether or not to perform an action we evaluate the threat represented by an event in relation to certain assets, keeping in mind the reliability of our data and the probability the event will occur.

This is where our system becomes more complex, and we must therefore be able to implement a *security policy*, a *network inventory*, a real-time *risk monitor*—all configured and managed within a single *framework*... In any case, we cannot let complexity keep us from achieving our objective: product integration.

Consequently, this project is quite ambitious; it inherits all the functionalities and impressive development progress made by an entire community of experts, while our role is that of mere integrators and organizers.

In this sense, our project aspires to exemplify the capacity of the open source movement to grow on its own and contribute innovative solutions in specific sectors like network security, in which free-software solutions contribute another valuable functionality: auditability of the systems we install on our network.

1. Introduction

1.1. Introduction

We developed this project in response to a scenario that has repeated itself over and over in recent years:

In the event an intrusion succeeds, once the perimeter defenses are cleared dozens of machines are compromised. There is a flow of permanent and durable connections for several hours, anomalous connections that establish paths completely contrary to what is acceptable. External processes create a bridge to enter the internal network, where more machines are compromised, one after the other, following more and more anomalous, and dangerous, paths.

The users and administrators of the victim organization who are working on those machines never notice anything strange at the time, and they rarely identify the attack after the fact.

Although somewhat exaggerated and comical, a real world analogy would be the thief that breaks into an office in broad daylight kicking in any one of its doors, walks right past people working at their desks, strolls into the filing room, photocopies the documentation of interest, finds a safe, and starts pounding away with a hammer without a second thought. Meanwhile, all the employees just sit there, engrossed in their work...

Something is wrong with how attacks on company networks are detected. We apparently have the right technology; we have the ability to detect extremely specific events using intrusion detection systems. Yet we are not able to monitor all the alerts they send due to two reasons:

- **volume**
- **unreliability**

In other words, we get too many alerts, and they are not reliable. We get too many **false positives**.

We receive information that is very detailed but partial, and that does not lend itself to abstraction. We are not able to detect attacks defined by more complex behavior; our second problem is **false negatives**.

1.2. What is OSSIM?

OSSIM is a distribution of open source products that are integrated to provide an infrastructure for security monitoring.

Its objective is to provide a framework for centralizing, organizing, and improving detection and display for monitoring security events within the organization.

Our system will include the following **monitoring tools**:

a. *Control panel for high-level display*
b. *Risk and activity monitors for mid-level monitoring*
c. *Forensic console and network monitors at the low level*

These tools utilize **new capabilities** developed in SIM post-processing, whose objective is to improve detection reliability and sensitivity:

a. *Correlation*
b. *Prioritization*
c. *Risk assessment*

Post-processing in turn makes use of the preprocessors, a number of detectors and monitors already known to most of the administrators that will be included in our distribution:

a. *IDS (pattern detectors)*
b. *Anomaly detectors*
c. *Firewalls*
d. *Various monitors*

Finally, we need an administrative tool that configures and organizes the various modules, both external and native, that comprise OSSIM. That tool is the framework, which allows us to inventory assets; to define the topology, a security policy, and correlation rules; and to link up the various integrated tools.

1.3. Open Source Infrastructure for Security Monitoring

Solution vs. Product

OSSIM is not a product; it is a solution, a system personalized for the needs of each organization and created by the interconnection and integration of various specialized modules.

In our solution, the following elements and definitions are just as important as the code:

a. The *architecture*
b. Correlation *models and algorithms*
c. The definitions of *environment* and *framework*
d. The definition of the model for *perimeter security management*
e. The map and procedures for auditing *detection capacity*

We are developing this project in the open source tradition both to make the code available for improvement and to generate discussion about and awareness of these models and algorithms.

Open Architecture

OSSIM has an open monitoring architecture and therefore integrates many open source products, always with the intention of abiding by the standards and practices of the open source community (which we believe will become the standards for monitoring solutions in all environments).

Integrated Solution

As an integrated solution, OSSIM offers tools and functionality for monitoring at any level, from the lowest (detailed IDS signatures, intended for the security technician) to the highest (a control panel designed for strategic management), and everything in between (forensic consoles, correlation levels, asset and threat inventories, and risk monitors).

Open Source Software

OSSIM was conceived as an integration project, and our intent is not to develop new capabilities but to take advantage of the wealth of free software "gems," programs developed and inspired by the best programmers in the world (including snort, rrd, nmap, nessus, and ntop, among others), by integrating them within an open architecture that preserves all their value and capabilities. Our solution will be responsible for integrating and interrelating the information provided by these products.

By virtue of being open source projects, these tools have been tested and improved by dozens or hundreds of thousands of installations all over the world, and consequently they have evolved into robust, highly tested, and therefore *reliable* elements.

The fact that they are open source means that anyone who wants to can *audit* them; these open source tools are above suspicion of harboring back doors.

2. The Detection Process

If we had to summarize what we are trying to accomplish or what our project is about in a single phrase, it would be this: **“Improve Detection Capability”**

In this section we will introduce the concepts related to network detection that will be developed throughout the document.

Detectors

We define a detector as any program capable of processing information in real time, usually low level information like traffic or system events, and capable of sending alerts when previously defined situations arise.

These situations can be defined in either of two ways:

1. By patterns, or rules defined by the user
2. By anomaly levels

Detection Capability

Detection capabilities have improved enormously in recent years, the best example being IDSs, which are capable of detecting patterns at the most detailed level.

In order to discuss the capability of a detector, we will define it using two variables:

- *Sensitivity* or the capability our detector has for extensive and complex analysis in identifying possible attacks.

- *Reliability*, which, as its name suggests, is the level of certainty provided by our detector when we receive warning of a possible event.

Inadequate Detection

Throughout this document we will see that despite advances in the scope of detection systems, their capabilities are still far from acceptable.

Because of detector inadequacy in these two areas, we are confronted with the two main problems encountered today:

- *False positives.* Detector unreliability or alerts that actually do not correspond to real attacks.

- *False negatives.* Inadequate detection means that attacks go unnoticed.

The following table summarizes these ideas:

I. Detector capability

	Property	Result in its absence
Reliability	The level of certainty provided by our detector when we receive warning of a possible event	False Positives
Sensitivity	The capability our detector has for extensive and complex analysis in locating possible attacks	False negatives

The Detection Process

We will refer to the global process developed by the SIM as the *detection process*, including the organization's various detectors and monitors as well as those executed by the system to process this information.

The detection process normally involves three well-defined phases:

- **Preprocessing:** Detection itself, in which detectors generate alerts and information is consolidated before being sent.
- **Collection:** All the information from these detectors is sent and received at a central location.
- **Post-processing:** What we do with the information after we have it all centralized.

Post-processing
Preprocessing and collection are traditional capabilities and do not contribute anything new to our solution. But in post-processing, once we have all the information in one place, we can implement mechanisms that will improve detection sensitivity and reliability. We increase the complexity of the analysis by including methods to discard false positives or on the other hand prioritize or discover more complex patterns that our detectors have overlooked.

In OSSIM we use three post-processing methods:

1. Prioritization: We prioritize alerts received using a contextualization process developed by defining a topological security policy in combination with the inventory of our systems.

2. Risk assessment: Each event is evaluated in relation to its associated risk, in other words, in proportion to the assets at risk, the threat represented by the event, and the probability it is real.

3. Correlation: We analyze a collection of events to obtain more valuable information.

The following figure shows how the properties mentioned above are affected by these processes:

II. Post-processing

	Processing	Effect
Prioritization	Assess the threat by contextualizing an event	Improves reliability
Risk assessment	Assess the risk in relation to the value of assets	Improves reliability
Correlation	Compare various events to obtain more valuable information	Improves reliability, sensitivity, and abstraction

So after processing them, our system will use the *alerts* provided by detectors to produce what will be referred to in this document as *alarms*.

An alarm will usually be the result of various alerts and have a higher level of abstraction that allows us to identify more complex patterns—and it will provide better reliability.

The Detection Map

With the goal of defining detection capability, OSSIM will develop a *detection map*, which will involve categorizing the possibilities in the detection of attacks and security events.

Auditing Detection Capability

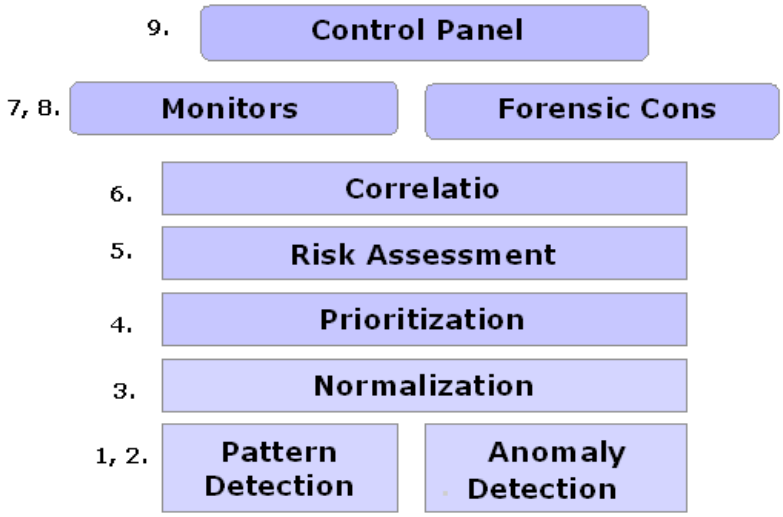
Using this *detection map* we will be able to define a new auditing method that will allow us to measure the situation and needs of an organization in relation to the effectiveness of its detection systems when an attack is in place.

This perspective is very different from the traditional audit or intrusion test, since we are interested not in locating security failures but in the capability to detect the eventual exploitation of these failures.

In this project we will therefore develop a Detection Capability Auditing Procedure with which we will provide a mechanism for assessing an organization’s situation in relation to its capacity for detecting attacks.

3. Functionality

In order to understand what OSSIM offers we can define the functionality of the system using a simplified, graphical form with the following nine levels:



We discuss each of these levels below to help give a practical description of our system:

3.1 Pattern Detectors

Most traditional detectors operate using patterns, the best example of which is the IDS or intrusion detection system, which is capable of detecting patterns defined using *signatures* or rules.

There is another class of pattern detectors that are included in most devices like routers and firewalls, and they are capable of detecting, for example, port scans, spoofing attempts, and possible fragmentation attacks.

We also have detectors for security events within an operating system. They are capable of sending alerts for possible security problems, and they almost all include their own logger, like syslog for UNIX.

Any element in the network, such as a router, a work station, a firewall, etc., has some capacity for detection. In our system, we are interested in collecting events from all critical systems in order to achieve one of our principle objectives: a comprehensive view of the network.

3.2 Anomaly Detectors

The ability to detect anomalies is more recent than that for patterns. In this case we do not have to tell the detection system what is good and what is bad; it can learn on its own and alert us when behavior deviates enough from what it has learned is normal.

Since the two processes work in opposite ways, this new functionality provides a point of view that is both different and complementary to pattern detection.

Anomaly detection can be especially useful for preventing, for example, perimeter attacks, which are one continuous anomaly: in the direction of the communications and the path they define, in the flow of data, in their size, duration, time, content, etc.

This technique provides a solution—until now beyond reach—for controlling access of privileged users, as in internal attacks by, for example, disloyal employees, in which no policies are violated and no exploits carried out. Yet they represent an anomaly in the use and manner of use of a service.

Now let's look at some other examples in which these detectors would be useful:

- A new attack for which there still are no signatures could produce an obvious anomaly yet circumvent pattern detection systems.

- A worm that has been introduced into the organization, a spamming attack, and even the use of P2P programs would generate a number of anomalous connections that are easy to detect.

- We could likewise detect:
    - Use of services that is abnormal in origin and destination
    - Use at abnormal times
    - Excess use of traffic or connections
    - Abnormal copying of files on the internal network
    - Changes in a machine's operating system
    - Etc.

We might think that, as an undesirable byproduct, these detectors will generate a number of new alerts, amplifying our signal and making our problem worse (our objective is to limit the number of alerts). However, if we take them as additional information that complements the traditional pattern alerts, we will be able to evaluate and therefore differentiate those that could result in a higher risk situation.

3.3 Centralization and Normalization

Normalization and centralization (or aggregation) are aimed at unifying security events from all critical systems throughout the organization in a single format on just one console.

All security products normally have a capacity for centralized management using standard protocols; hence, aggregation is simple using these protocols. In OSSIM we attempt to avoid regular use of agents and instead use forms of communication that are native to systems.

Normalization requires a parser or translator familiar with the types and formats of alerts coming from different detectors. We will need to organize the database and adapt the forensic console in order to homogenize processing and display of all these events.

That way we will be able to observe all security events for a particular moment in time—whether they come from a router, a firewall, an IDS, or a UNIX server—on the same screen and in the same format.

When we have all network events centralized in the same database, we achieve a considerably comprehensive view of what's going on throughout the network, which, as we will see shortly, allows us to develop processes that enable us to detect more complex and widely dispersed patterns.

3.4 Prioritization

The priority of an alert should depend on the topology and inventory of the organization's systems. The reasons are quite clear, as demonstrated by the following examples:

a. If a machine running the UNIX operating system and Apache web server receives an alert about an attack on Microsoft IIS, the alert should be deprioritized.

b. If a user makes a suspicious connection to a server, the system should:

    - Give it maximum priority if the user is external to the network and attacking the client database.
    - Give it low priority if the user is internal to the network and attacking a network printer.
    - Discard it if the user is someone who normally tests development servers.

By prioritization we mean the process of contextualization, in other words, the evaluation of an alert's importance in relation to the organization's environment, which is described in a knowledge base for the network comprised of:

- An inventory of machines and networks (identifiers, operating systems, services, etc.)
- An access policy (whether access is permitted or prohibited, and from where to where)

To perform these tasks (as in risk assessment, as explained in the following section), we have a framework in which we can configure the following:

1. Security policy, or assessment of asset-threat pairs according to topology and data flow
2. Inventory
3. Asset assessment
4. Risk assessment (prioritization of alerts)
5. Assessment of the reliability of each alert
6. Alarm definition

Prioritization is one of the most important steps in filtering alerts received by detectors and should be executed using a continuous process of fine-tuning and feedback from the organization.

3.5 Risk Assessment

The importance given to an event depends on these three factors:

a. The value of the assets associated with the event

b. The threat represented by the event

c. The probability that the event will occur

Intrinsic Risk

These three factors are the building blocks for the traditional definition of risk: a measure of the potential impact of a threat on assets given the probability that it will occur.

Traditionally risk assessment is concerned with intrinsic risks, or latent risks, in other words, risks that an organization assumes by virtue of both the assets it possesses for the purpose of developing its business and circumstantial threats to those assets.

Immediate Risk

In our case, due to real-time capabilities we can measure the risk associated with the current situation in immediate terms.

In this case the measurement of risk is weighted by the damage it would produce and the probability that the threat is occurring in the present.

That probability, which is a derivative of the imperfection of our sensors, turns out to be nothing more than the degree of reliability of our sensors in detecting the potential intrusion-in-progress.

By immediate risk we mean the state of risk produced when an alert is received and assessed instantaneously as a measure of the damage an attack would produce, weighted by the reliability of the detector that made the report.

OSSIM calculates the immediate risk of each event received, and this will be the objective measure we use to assess the event's importance in terms of security. We assess the need to act using this measure only.

Our system likewise includes a risk monitor (described below) that assesses the risk accumulated over time of networks and groups of machines related to an event.

3.6 Correlation

We define the correlation function as an algorithm that executes an operation on input data and returns output data.

We must consider the information collected by our detectors and monitors to be specific yet partial; it illuminates only small areas along the spectrum defined by all the information we would really like to have.

We can think of correlation as the ability to take advantage of these systems and, using a new layer of processing, fill in more areas along that infinite spectrum of all possible information about a network.

If we got carried away, this idea could lead us to try to install one single system with a detector that can gather all possible information about the network, but it would require a display that shows absolutely everything in one place and almost limitless memory and storage capacity.

Therefore, correlation systems merely supplement the inadequate sensitivity, reliability, and limited range of our detectors.

Input and Output

In simple terms, our architecture has two clearly defined elements that provide information to its correlation functions:

- Monitors, which normally provide indicators
- Detectors, which normally provide alerts

The output will also be one of these two elements: alerts or indicators. Our correlation functions become new detectors and monitors.

3.6.1 Correlation Model

The OSSIM correlation model has ambitious objectives:

1. Develop *specific patterns* for detecting the known and detectable
2. Develop *nonspecific patterns* for detecting the unknown and the undetectable
3. Provide an *inference machine* that can be configured using interrelated rules and that has the ability to describe more complex patterns
4. The ability to link detectors and monitors recursively to create more abstract and useful objects
5. Develop algorithms for displaying a general view of the security situation

3.6.2 Correlation Methods

To achieve these objectives we employ two very different correlation methods, based on the following two principles:

- **Correlation using sequences of events**, focused on known and detectable attacks, relates the known patterns and behaviors that define an attack using rules implemented by a state machine.

- **Correlation using heuristic algorithms.** Using an opposite approach, we implement algorithms that attempt to detect risky situations using heuristic analysis. In an effort to compensate for the shortcomings of other methods, this one detects situations without knowing or displaying the details. It is useful for detecting unknown attacks and displaying a general view of the security state for a large number of systems.

3.6.2.1 Method 1: Correlation Using Heuristic Algorithms

For correlation OSSIM will implement a simple heuristic algorithm using event accumulation in order to obtain an indicator or snapshot of the general security state of the network.

In this process our first objective is to obtain what we defined earlier as immediate risk and subsequently a value we could call *accumulated risk*.

This provides high level monitoring that we can use as a “thermometer” for risky situations without ever knowing any details about the characteristics of the problem.

Continuing with that analogy, we will construct a sensitive thermometer that will display the total accumulated risk in a certain time frame. The thermometer will go up proportionately to the amount of events received recently and to how “hot" they are, and it will cool off as time goes on if no new events are received.

This correlation method supplements correlation using sequences of events with an opposite approach—in the latter we attempt to characterize possible attacks to the highest level of detail.

Consequently, the value of correlation using heuristic algorithms is twofold:

- It provides a quick global view of the situation.

- It detects possible patterns that other correlation systems might overlook, either because the attacks are unknown or due to some shortcoming.

CALM

CALM (Compromise and Attack Level Monitor) is an assessment algorithm that uses event accumulation with recovery over time. Its input is a high volume of events, and its output is a single indicator of the general state of security.

This accumulation is executed for any subject on the network, including any machine, group of machines, network segment, path, etc. that we are interested in monitoring.

Event Accumulation

Accumulation is calculated simply by the sum of two state variables that represent the immediate risk of each event:

- **“C” or level of compromise**, which measures the probability that a machine is compromised

- **“A” or level of attack** to which a system is subjected, which measures the potential risk due to attacks launched

Why separate these two variables for monitoring? First, because they characterize different situations: the level of attack indicates the probability that an attack has been launched, an attack that may or may not be successful; while the level of compromise provides direct evidence that there has been an attack and that it has been successful.

Second, the importance of both variables depends on the situation of the machine. Mainly due to the exposed state of perimeter networks, which are exposed to an enormous number of attacks, most of them automated, unfortunately a high level-of-attack value is a “normal” situation. However, any sign of compromise or movement that might lead us to think there is an attacker residing in these networks should be immediately pointed out and reviewed.

On the other hand, there are cases in which a machine generates anomalies within the network due to the nature of its function—such as a security scanner, a service with random passive ports, development, etc.—and these will normally have a high C and a low A.

A value is assigned to the C or A variable for a machine on the network according to three rules:

1. Any possible attack launched from machine 1 on machine 2 will increase the A (level of attacks experienced) of machine 2 and the C (level of compromise, or suspicious actions normally perpetrated by a hacker) of machine 1.

2. When there is an attack response (a response that could indicate the attack’s success), the value of C will increase for both machines 1 and 2.

3. If the events are internal, the C value will go up only for the originating machine.

Accumulation Over Time

Since CALM is intended for real-time monitoring, we are interested in a short-term time frame. In other words, We are interested in assessing recent events, and the algorithm should have short-term memory that places importance on the most recent events and discards the oldest.

The current implementation uses a simple variable for recovery over time. The system will periodically lower the C and A levels for each machine by a constant value.

3.6.2.2 Method 2: Correlation Using Sequences of Events

The Sequences Panel

The basic idea for detecting a sequence of patterns is simple: just create a list of rules like “if you receive event A and then B and then C, perform action D.”

To that end we use the sequences panel, where we define lists of rules for each sequence of events that we want to define.

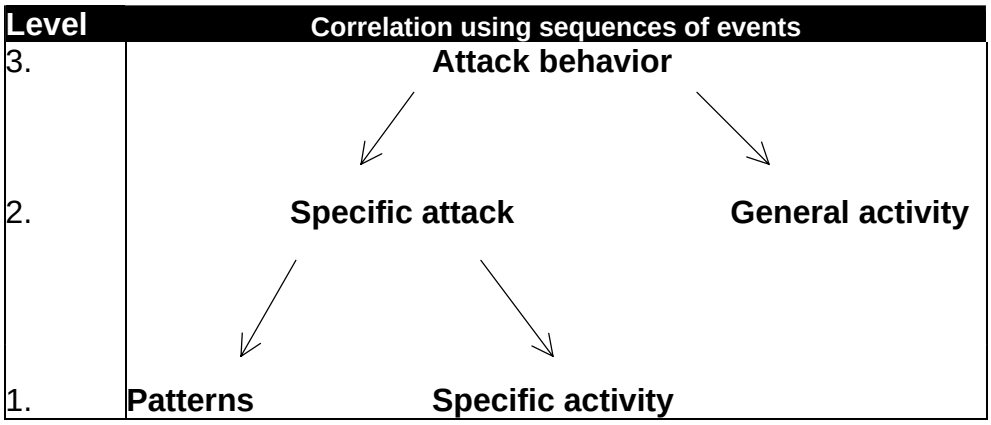
The complexity of the panel depends on the capacity of those rules for abstraction and OSSIM’s ability to analyze a variety of input.

Our panel can execute sequences with the following characteristics:

- Ability to define variable origins and destinations
- Accept both pattern input from detectors and indicator input from monitors
- Define the priority and reliability of new alerts
- Use “elastic” variables, or variables that can measure an event to define priority or reliability (e.g. total denial of service -> high priority, 50% denial -> medium priority, 15% denial -> low priority)
- Recursive architecture—we can create objects by correlating rules that function as detectors or monitors in new rules

3.6.3 Levels of Correlation

The recursive nature of our model allows the creation of an almost infinite hierarchy of levels, but for the purposes of this explanation we will define a hierarchy with three levels as shown in the following diagram:



We will discuss all three levels, but out of order to facilitate explanation:

Level 2.1 Specific Attack
This level deals directly with detectors and monitors. We will attempt to include both signatures and activity related to concrete attacks, i.e. those with a known first and last name, exactly as identified by the detector (for example, “compromised by ftp cwd overflow”).

The main objective of the specific attack level is to improve detection reliability. This means that instead of being satisfied with a possible attack signature, we look for more evidence demonstrating that the attack is either in progress or has failed.

This evaluation is what will make the difference in limiting false positives and prioritizing real attacks in a security detection system, since as we have seen the reliability of an event directly affects the calculation of risk.

Consider this simple example of correlation between a pattern detector and a monitor:

Using a signature the IDS detects a possible denial of service attack via synflood *and sends an alert triggering a query to the service monitor to see if it has experienced a drop in availability and to what degree. Consequently we can assign our alert “denial of service via synflood” a higher degree of reliability.*

We normally use more complex sequences in which we correlate alerts produced by signatures with the specific behavior characteristic of an attack. Now consider Trojan horse detection. We can detect various operations using IDS signatures:

> *Connect, active, get info, access, server2client_flow, client2server_flow, response, traffic_detect*

Detection of a connect operation is probably not very useful information in and of itself. There are dozens every day in perimeter environments, but if we detect any other operation—in particular an attack response—we should send a high priority alert.

Level 1.2 Detection Using Specific Activity

We will use the Trojan horse example to explain the concept of *specific activity*. Consider this simple case: following a connection attempt, if the Trojan horse is using port P, we just check this port for activity, i.e. transmission of data. If it is active, we have, as we did above, a confirmation that the connection attempt was probably successful. But this time we identified it by monitoring the activity of the Trojan itself instead of using an IDS signature.

For specific activity we use monitors to examine a concrete issue regarding activity associated with a possible specific attack by sending queries that start and stop the correlation engine for that issue.

Level 1.1 Detection Using Patterns

In this level, which we have already discussed, our system processes any alert sent by the pattern detectors.

Attack Responses

Based on the two previous points, we can conclude that *attack responses* are important because they verify the existence of an event, which is equivalent to an increase in the alert's reliability.

Our correlation engine is designed to continuously look for these attack responses, and following initial signs of a possible attack, we allocate our entire system to finding evidence that the attack is really under way. This allows us to differentiate failed attacks from successful ones.

For our Trojan horse example, we can represent this graphically in the following way:

Improved reliability for a Trojan horse

Succession of events	Alert type	Reliability
1. Trojan horse compromise or connection attempt	Signature: connection, client2server, access, getinfo	Low
	Specific activity: flow from attacker -> victim	
2. Detection of typical response to attack success or response to Trojan horse operation	Signatures: response, server2client	High
	Specific activity: flow from victim -> attacker	

Level 2.2 Detection Using General Activity

By detection using general activity we mean rules aimed at locating unknown or undetectable attacks, since signatures or patterns that characterize them are unavailable.

Attacks can be identified thanks to anomalous activity generated by the attacker, which we detect by monitoring indicators of general user activity such as ports or services, traffic, times, etc. for all users.

In some cases we can characterize attacks to a certain degree of detail using this technique. But generally we detect suspicious behavior with less precision than in specific attack detection, often in the grey area between attacks, network problems, and user misconduct.

Following are some examples of detection using general activity:

- Detection of an unknown worm that generates abnormal traffic and a number of atypical connections to formerly unused ports and destinations

- Detection of suspicious access, such as a user making a persistent connection to an administrative port for the first time

- Excessive traffic, with anomalous destinations and uses

Level 3. Attack Behavior

The third level is mainly the correlation of various specific attacks or general activities identified in the first level.

Remember that our system's correlation architecture is recursive, and that our rules can include new objects that function as detectors (and send alerts) or as monitors (and provide values) and that are comprised of a collection of rules from the previous level.

But we should not definitively characterize these new levels by the fact that objects from the previous level are used as input. This is not always be the case; we can mix and match methods as appropriate.

Each level corresponds to a certain degree of abstraction, so for this level we attempt to identify behavior patterns that help us identify the object, the path taken, and the attacker's behavior and method. To that end we define *attack behavior* as the sequence of attacks and activities carried out by a user on one or more compromised machines.

Following are some examples of attack behavior:

- Distributed attack, characterized by a relationship among various attackers and attacks experienced

- Access to a critical network from the Internet, following the flow of a perimeter attack that reaches a critical network from the Internet in a few hops

- Internal malicious user compromise, identified by various abnormal actions taken by an internal user

3.7 Monitors

Although they are simply monitors of previously executed processes, as such these functionalities are worth pointing out:

3.7.1. Risk Monitor

OSSIM possesses a risk monitor called RiskMeter that displays the values produced by the CALM algorithm. These values measure the compromise (C) and attack (A) risk levels derived from alerts that indicate the possibility that a machine has been compromised or is being attacked.

3.7.2. Use, Session, and Profile Monitors

As explained in the section on anomalies, OSSIM places a lot of importance on detailed monitoring of each machine and profile.

There are three types for this kind of monitoring:

- The *use monitor* provides general information about the machine, such as number of bytes transmitted per day.
- The *profile monitor* provides specific information about user activity, allowing us to establish a profile, for example “uses mail, pop, and http" is a normal user profile.
- The *session monitor* provides a real-time display of the sessions in which a user is engaged, as well as a snapshot of that machine on the network.

We believe that all three of these are essential for a security system, and in their absence the security administrator would be blind to past events, would not be able to distinguish normal from abnormal activity, and would not be able to see the network—like a traffic cop on a pitch black road.

This area of security coincides with network administration, but some overlap is inevitable since, for example, the saturation of a network or the anomalous behavior of a machine could indicate either a network problem or a security problem.

OSSIM provides these three monitoring capabilities using products with the ability to act as sniffers and see the network situation at the highest degree of detail.

3.7.3. Path Monitor

This monitor can trace in real time the paths used by various machines on the network for communications or links.

The diagram is refreshed at a certain time interval, creating a graphic whose branches appear and disappear over time.

This monitor receives data from two other monitors: the session monitor identifies every link present on the network, and the risk monitor sends the risk level for every machine so the path monitor can render them with colors according to risk and calculate the aggregate risk.

As for link monitoring, there are two methods:

3.7.3.1. Hard Link Analysis (TCP Link Analysis)

Using this method we graph only persistent TCP sessions. The purpose of this method is to identify network attacks involving the simultaneous intrusion of various machines, a typical situation for perimeter intrusions.

3.7.3.2. Soft Link Analysis

OSSIM uses soft link analysis to graph all links perceived on the network, including UDP as well as TCP and ICMP, often resulting in chaotic network maps.

3.8 Forensic Console

The forensic console provides access to all the information gathered and stored by the collector.

This console is a search engine that operates on the event database, allowing the administrator to analyze security events in relation to all critical elements of the network a posteriori and in a centralized manner.

Unlike the risk monitor discussed in the previous section, this console allows us to explore every event that occurs in the system to the maximum degree detail.

3.9 Control Panel

And finally, the control panel allows us to view the network security situation at a high level. It monitors a series of indicators that measure the state of the organization in relation to security.

The control panel allows us to define a series of thresholds or objectives that the organization should meet. These thresholds are defined as either absolute or relative values such as degree of anomaly.

We can assign alarms to be sent or any automatic process to be executed when these thresholds are surpassed.

The way information is displayed in the control panel is important, so it should be as concise and simple as possible. To that end we need a flexible configuration that shows only information that is relevant at the moment of interest.

The control panel is our general thermometer for everything happening in the network. And we use it to access all the monitoring tools to inspect any problem that has been identified.

For example, it can display the following:

- Constant monitoring of risk levels for the organization's main networks
- Monitoring of machines or subnetworks that surpass a security threshold
- Constant monitoring of general network, system, and service level parameters:
    - Throughput and traffic on main networks
    - Main database resources
    - Latency of critical services
    - Number of transactions of critical services
- Monitoring network or service level parameters that surpass an established threshold:
    - Number of emails, viruses, and external accesses
    - Latency of services, and their use of traffic
- Monitoring profiles that surpass thresholds for:
    - Use of traffic
    - Use of critical services
    - Use of anomalous services
    - Changes in configuration
    - Any other anomalous activity

In our opinion, the control panel should be totally customized. Unlike all the other functions, it will only include one sample to personalize.

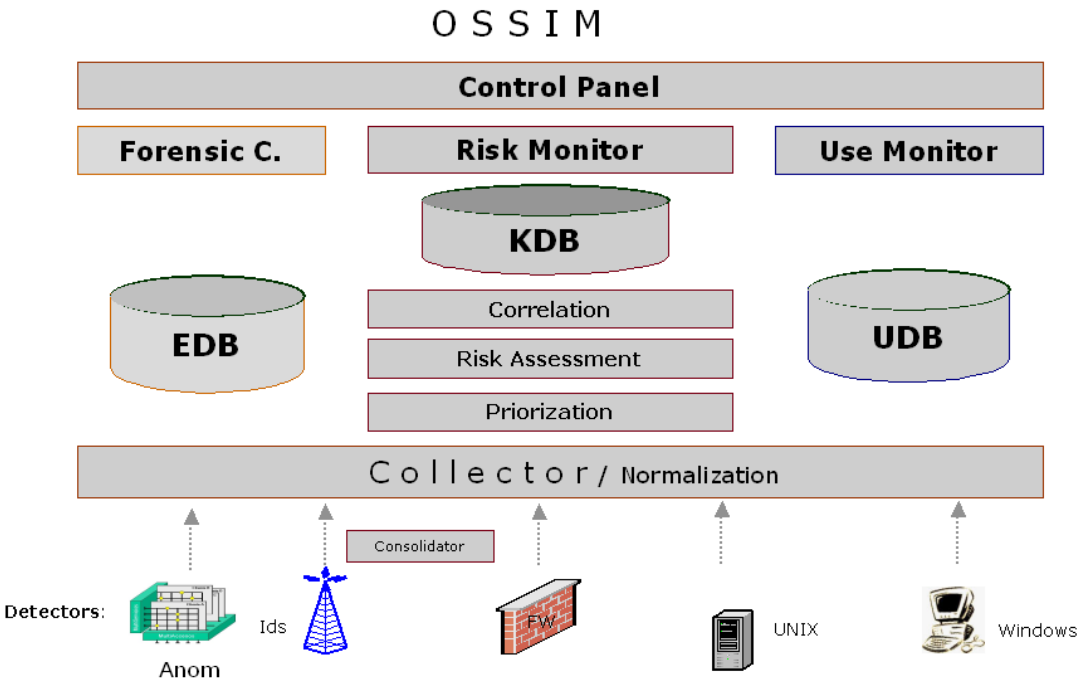
4. Architecture

4.1. General Architecture

The system's processes are divided into two basic stages, each corresponding to one of two different parts of the architecture:

- Preprocessing, carried out by the monitors and detectors
- Post-processing, executed in a centralized console

The following is a general diagram of the architecture displayed according to process:



In addition to all the functionalities described above, this diagram shows three databases:

- EDB, the event database, which is the largest because it stores every individual event perceived by our detectors
- KDB, the knowledge or framework database, in which we parameterize the system to be familiar with the network and define the security policy
- UDB, the profile database, which stores all information gathered by the profile monitor

4.2 Data Flow

To give a better understanding of how each product is integrated, we offer the following step-by-step description of data flow beginning with the generation of an event:

1. Events are processed by the detectors until an alert is produced in response to identification of a pattern or anomaly.

2. If necessary, alerts are processed by the consolidators before being sent. The consolidators send alerts in groups to occupy a minimum of bandwidth.

3. The collector receives alerts using various open communication protocols.

4. The parser normalizes and saves them in the event database.

5. The parser also prioritizes alerts according to the security policy defined in the framework and information in the systems inventory about the system under attack.

6. The parser assesses the immediate risk represented by the alert and sends an alarm to the control panel as necessary.

7. Prioritized alerts are sent to each correlation process, which updates their state variables and eventually sends new alerts with more complete or reliable information. These alerts are re-sent to the parser to be stored, prioritized, assessed for risk, etc.

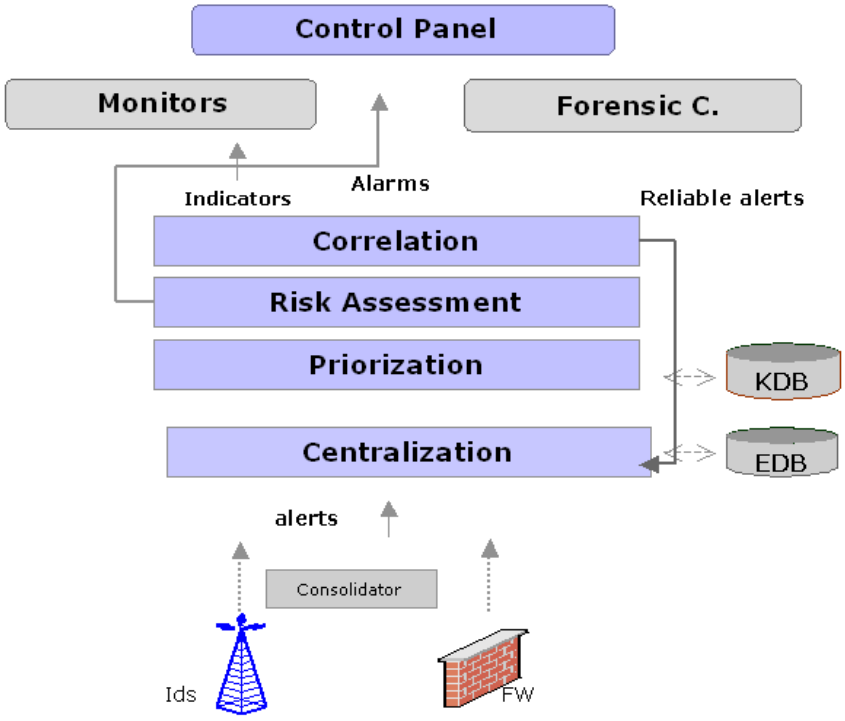
8. The risk monitor periodically displays the state of each risk index as calculated by CALM.

9. The control panel shows the most recent alarms, updates the state of all the indices which it compares to their thresholds, and sends out new alarms or performs the appropriate actions as necessary.

10. From the control panel, the administrator can link and view all events occurring at the time of the alert using the forensic console.

11. The administrator can also check the concurrent state of the machine involved using the use, profile, and session monitors.

The following graphic shows the data flow:



4.3 Distribution Architecture

OSSIM is a distribution rather than a product, meaning that integration takes precedence over development. The OSSIM project aims at intercommunication, making these products talk to each other.

Due to the increasing comlexity of development, so we have defined two levels:

4.3.1 The Kernel

The first level developed by the OSSIM GNU project is what a typical distribution would call the kernel, which encompasses the following tasks:

- Define the data structure
- Provide interfaces for talking to all the different products
- Work on what was formally called post-processing
- Provide a framework in the first layer of administration for linking to all other administrative systems
- Implement the control panel

4.3.2. Third-party Products

Here we discuss third-party products not developed by the project yet integrated in our solution. There are two product types:

- Open source products, which may be modified and/or patched depending on the product and will usually be included in the distribution
- For-pay products, which obviously will not be included in distributions nor will they be patched or modified

Consult the Road Map section to see the relationship between architectures.

Contact

OSSIM is an open project that hopes to benefit from collaboration with the open source community and security solutions developers and integrators. To that end, any contribution, comment, or discussion about the contents of this document is entirely welcome.

We are developing collaborative efforts in all areas of this project, including participation in software development, definition of the architecture, documentation and methodology development, portability to different platforms, translation, etc.

Current contact addresses are as follows:

Official homepage:	http://www.ossim.net
Project page:	http://sourceforge.net/projects/os-sim/
Contact email:	contact@ossim.net
Support email:	support@ossim.net
Releases:	http://freshmeat.net/projects/os-sim/
CVS:	http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/os-sim