A Serial Combination of Anomaly and Misuse IDSes Applied to HTTP Traffic

Elvis Tombini, Hervé Debar
{elvis.tombini|herve.debar}@francetelecom.com
France Télécom
Caen - France

Ludovic Mé
lme@rennes.supelec.fr
Supélec
Rennes - France

Mireille Ducassé
mireille.ducasse@irisa.fr
IRISA/INSA
Rennes - France

Abstract

Combining an "anomaly" and a "misuse" IDSes offers the advantage of separating the monitored events between normal, intrusive or unqualified classes (ie not known as an attack, but not recognize as safe either). In this article, we provide a framework to systematically reason about the combination of anomaly and misuse components. This framework applied to web servers lead us to propose a serial architecture, using a drastic anomaly component with a sensitive misuse component. This architecture provides the operator with better qualification of the detection results, raises lower amount of false alarms and unqualified events.

1. Introduction

Intrusion Detection Systems (IDS) aim at discovering if intruders try to break into a system or if users try to abuse their privileges. There are two main classes of intrusion detection techniques, misuse detection and anomaly detection. Misuse detection components use a set of attack signatures to detect traces of attacks in a given set of events. This method may work well with known attacks, but it is unable to detect unknown attacks. Furthermore, a compromise has to be made between the sensitivity of the signatures and the risk of false alarms. Very sensitive signatures result in high false alarm rate. Very selective signature increase the risk of missing attack variants. Anomaly detection components are based on a model of acceptable behaviors. This model is obtained either through a learning process [3, 5], or given a priori [10, 12]. The incoming events are then compared with the model. If they deviate from the model, they are considered anomalous. The structure of the model is critical to this method. If the detector is too sensitive it raises a high number of false alarms and if its configuration is too lax it misses attacks.

Each method reveals only partial information about the events it monitors. Misuse detection is only able to determine if a set of events corresponds to known attacks. If no attack is discovered, it does not mean that the corresponding events are safe. They may contain attacks for which no signature exist, thus these events are unqualified. Likewise, anomaly detection can only recognize a behavior as safe. If not, it does not reveal an attack, but an unknown behavior.

Combining anomaly and misuse detection enables a better qualification of the analyzed events. However, as no component so far can be considered 100% correct and complete, qualification conflicts arise. EMERALD [9] explicitly handles these conflicts through a correlation component. This, however, means that the events are analyzed twice, even when it can be known beforehand that some events are intrusive or safe.

The contribution of this article is twofold. It provides a *framework* to systematically reason about the combination of anomaly and misuse components. This should help designers of IDS in other contexts than ours to decide which architecture is the more accurate. We also propose an IDS sensor architecture for analyzing HTTP traffic: the events not qualified by a simple anomaly detection component are sent to a very sensitive misuse component. The experimental prototype shows that the approach drastically reduces the amount of unqualified events and false alarms without missing critical attacks.

In the following, Section 2 describes WIDS, the misuse detector we used to perform our experimentations and the targeted web servers. Section 3 proposes a formalism to describe anomaly and misuse intrusion detection results. Section 4 justifies an architecture where an anomaly component and a very sensitive misuse component are connected in sequence. Section 5 describes the experimental validation of the proposed ar-

Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)
1063-9527/04 $ 20.00 IEEE

chitecture. Section 6 discusses related approaches.

2. Misuse analysis of web server log files

Analyzing web server log files is an important issue, as web servers and proxies provide an universal gateway to the information system. We have been using a web intrusion detection system (WIDS) inspired by the work of Almgren et al. [1] to analyze in batch mode web server log files to detect compromise attempts and worm infections. In a few instances, the tool has also been run in real-time mode to analyze on the fly logs as they are produced by the target server or proxy. CLF- or ECLF-formatted files are understood by the tool, although the extended information provided by ECLF is considered unsafe and is ignored, as it is provided by the web browser.

2.1. Focus of the tool

Our WIDS tool distinguishes itself from [1] in several areas. We have chosen to implement three cascaded modules, a normalization module that cleans and segments the input log lines, a pattern matching engine and a prolog engine. As such, we do not use the false alarm filtering and host analysis modules described in [1]. The false alarm filtering module is partially replaced by prolog signatures. The host analysis module requires state tracking. Assuming that each monitored server interact with more than 100k clients per day (a conservative estimate for production servers), the additional memory required for tracking cannot be taken for granted on production web servers and proxies.

We also distinguish our work from [11] as we need to be able to run both off-line and on-line, on multiple web servers and proxies on multiple hardware architectures and operating systems. As such, the only reliable and archivable data source is the web server log trail and we cannot capture system level or network level information. Moreover, HTTP is a stateless protocol; partial state information may be rebuilt from the server logs, using a combination of client IP address and CGI script parameters. Our current focus excludes this capability, although we plan to work on HTTP session reconstruction and modeling in a future version of WIDS.

We understand that information is lost in this process, and as such not all attacks will be detected by WIDS. We consider that the ease of installation and use gained at the same time provide more benefit to the diffusion of the tool than the number of attacks missed. A rough estimate of this is given in table 1, which summarizes the number of bugtraq-referenced vulnerabilities relevant to Apache and IIS web servers during the last 18 months, ordered by decreasing severity.

Criticality	Detectable		Invisible	
	Apache	IIS	Apache	IIS
Easy R2U, Remote2R	2	1	0	0
Difficult R2U, Remote DoS	6	2	1	0
Easy U2R	8	1	8	2
Local DoS, Difficult U2R	2	8	11	4
Total	18	12	20	6
Full total	30		26	

Table 1. Distribution of web-related vulnerabilities in 2003/S1-2004 (18 months)

Table 1 shows that more than half of the vulnerabilities relevant to our intrusion-detection system and published by CVE or bugtraq during the last 18 months can be detected by our tool. Moreover, all critical vulnerabilities (i.e. resulting in remote attack of any effect) by all kinds of attackers are detectable. This justifies our focus as the tool will only miss local attacks by insiders. As these persons already have full access to the servers, breach of security becomes more a human resources issue than a technical issue.

2.2. Description of the WIDS tool

WIDS is implemented entirely in C++, with the adjunction of the pcre library for regular expression matching and SWI prolog for rule support. It is supported by a simple yet powerful signature language that configures the 3 modules using the same syntax. Signatures are grouped in sections, the section type indicating the module that should understand the syntax of the active component of the signature. The active component is either a regular expression or a prolog-like rule; it returns a $TRUE$ boolean value to indicate a match of the signature, $FALSE$ otherwise.

Sections also have a notion of flow control, that will make the analysis follow different paths according to the previous findings. As such, an essential originality of WIDS is that signatures are cumulative, and that it will try (depending on the control statements) to match as many signatures as possible. This makes it possible to write specific signatures that take into account different aspects of an attack. It is then the role of the prolog engine to evaluate the set of matched signatures, and modulate the diagnostic according to a set of rules. The main goal of this set of rules is to ensure that any request that could result in a possible

compromise of the server is rated higher than failed or irrelevant attempts.

The current ruleset has 652 signatures split in 30 sections (including one normalization and one rule section). 5 signatures cover error cases related to the operation of the analysis engine. 6 signatures cover the operation of the prolog engine; these signatures expressed as Prolog rules are here mostly as examples, since they tend to need site-specific tuning to effectively reduce false positives. 62 signatures aim at providing contextual information, such as the type of request, the status code or the various encodings used. 28 signatures implement the detection of generic attack methods, such as frequent patterns observed in cross-site scripting. 348 signatures implement the detection of CGI script attacks. Finally, 203 signatures implement the detection of interesting targets; this is particularly interesting because while the number of vulnerable CGI scripts is constantly increasing, the targets open to an attacker remain few. As such, we successfully detect attacks against newly-announced CGI vulnerabilities, because of the usage of UNIX commands, Windows or Unix system files, or other simple characteristics.

2.3. Log Analysis using WIDS alone

We have measured the HTTP traffic of two web servers: an academic one (Supélec) during 92 consecutive days in 2003, and an industrial one (FT) during 30 consecutive days, in 2001. The Supélec web server hosts resources in two manners. A part of these resources is freely available from everywhere, while another part is only reachable from the Supélec intranet. The resources hosted by the Supélec web server are composed of 99% of static resources and 1% of dynamic resources. The dynamic resources are PHP scripts, such as a local search engine, and some CGI scripts, such as counters. The logged events are composed 91.5% of requests toward static resources and 8.5% of requests toward dynamic resources. The publishing process is not totally centralized since a part of the web site is managed by a webmaster and academics manage their own pages themselves. The FT web server hosts resources for professional purposes. These resources are available from everywhere. The resources hosted by the FT web server are composed of 98.5% of static resources and 1.5% of dynamic resources. The dynamic resources are java servlets. The logged events are composed 98% of requests toward static resources and 2% of requests toward dynamic resources The publishing process is under the responsibility of a few persons.

Table 2 presents the results obtained when applying WIDS to log files of the two previously described web servers. For our purpose, we only used the step of analysis with the signature sets previously described without any re-evaluation done by the prolog engine.

	Supélec	**FT**
days	92	30
logged events	2,402,749	3,008,842
resources	27,801	2,670
Severity	Percentage	Percentage
0	79.1372	89.1331
1	11.9839	1.9027
2	7.5018	8.9494
3	0.7427	0.0134
4	0.5948	0.0014
5	0.0200	0
6	0.0134	0
7	0.0010	0
8	0.0005	0
9	0.0002	0
> 9	0.0145	0

Table 2. Web server log file analysis by a WIDS misuse component. The percentages are rounded off to the fourth figure after the dot.

The Supélec log file contains around 2,5 millions events targeting around 28 thousand different resources. More than 79% of the events are of severity 0, namely for them no signature at all match. The FT log file contains around 3 millions events targeting around 26 thousand different resources. More than 89% of the events were of severity 0.

Hence, our context has two important characteristics: (a) According to the high sensitivity of our signature set, we consider that events with a severity 0 are truly safe. (b) As a consequence, a very high rate of HTTP traffic is safe. As a result, WIDS spends an enormous amount of processing power on traffic that cannot lead to a server compromise.

3. Combining Anomaly and Misuse

Intrusion-detection systems need to be updated periodically to ensure that they have the latest knowledge about vulnerabilities. This has the unfortunate side effect that a web site using an application on which a script is suddenly tagged vulnerable will incur a large number of alerts that are triggered by the normal usage of the application. Hence, it would be interesting to filter out all the known normal usage of web applications before looking for specific attacks.

However, an anomaly-detection-only intrusion-detection system makes the automation of IDS alert

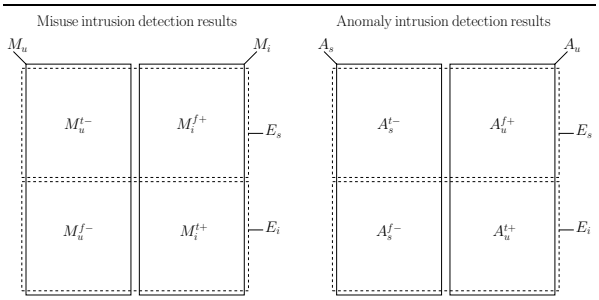

Figure 1. Anomaly and misuse detection results relative to the truly intrusive and safe events

processing difficult. The good thing about misuse detection is that it precisely identify the exploited vulnerability. Its relevance to the monitored information system and the impact of the attack can automatically be assessed, and hence the attack stopped with little to no human intervention. Therefore, we believe that combining both anomaly and misuse detection can effectively provide a technique for mitigating attacks with as little human intervention as possible.

In this section we propose a set model to formally describe anomaly and misuse intrusion detection results. We first define our notations for each intrusion detection component, and then we comment the results of these two components combination. We use here the classical terminology:

- false positive: false alarm, i.e. alarm raised by an IDS while the monitored system is **not** under attack;
- false negative: attack against the monitored system that is **not** detected by the IDS;
- true positive: attack against the monitored system detected by the IDS;
- true negative: no alert emitted by the IDS when no attack against the monitored system.

3.1. A Set Model for Intrusion Detection Results

Intrusion detection component processes events. No existing component can be considered 100% correct. A misuse component can for example declare an event intrusive, while it is safe. Likewise, an anomaly component can declare an event safe while it is intrusive. In this section, we set definitions in order to formalize intrusion detection results according to the true nature of processed events. The definitions are illustrated in Figure 1.

Definition 1 ***Safe and intrusive events***
Let E be the set of events an IDS processes, E_i be the subset of E that represents the set of truly intrusive events and E_s be the subset of E that represents the set of truly safe events.

An event cannot be both intrusive and safe, and it cannot be anything else, we therefore have the following property:

Property 1 *$E_i \cup E_s = E$ and $E_i \cap E_s = \emptyset$.*

Definition 2 ***Anomaly detection diagnosis***
Let A_s (resp. A_u) be the subset of E that represents the set of events **declared** *safe (resp. unknown) by an anomaly detection.*

Definition 3 ***Misuse detection diagnosis***
Let M_i (resp. M_u) be the subset of E that represents the set of events **declared** *intrusive (resp. unknown) by a misuse detection.*

Anomaly (resp. misuse) detection is designed to qualify events as safe or unknown (resp. intrusive or unknown). So an event cannot be qualified both safe and unknown by anomaly (resp. intrusive or unknown by misuse). Moreover an event cannot be qualified in any other way. We therefore have the following property:

Property 2 *$A_s \cup A_u = E$, $A_s \cap A_u = \emptyset$, $M_i \cup M_u = E$ and $M_i \cap M_u = \emptyset$.*

Since the results of both anomaly and misuse detections are not perfect, we have to define their correct results and their mistakes.

Definition 4 ***Anomaly detection results***

- *Let A_u^{f+} be the intersection of A_u and E_s that represents the set of false positives in the anomaly detection, $A_u^{f+} = A_u \cap E_s$;*
- *Let A_u^{t+} be the intersection of A_u and E_i that represents the set of true positives in the anomaly detection, $A_u^{t+} = A_u \cap E_i$;*
- *Let A_s^{f-} be the intersection of A_s and E_i that represents the set of false negatives in the anomaly detection, $A_s^{f-} = A_s \cap E_i$;*
- *Let A_s^{t-} be the intersection of A_s and E_s that represents the set of true negatives in the anomaly detection, $A_s^{t-} = A_s \cap E_s$.*

Definition 5 ***Misuse detection results***

- *Let M_i^{f+} be the intersection of M_i and E_s that represents the set of false positives in the misuse detection $M_i^{f+} = M_i \cap E_s$;*

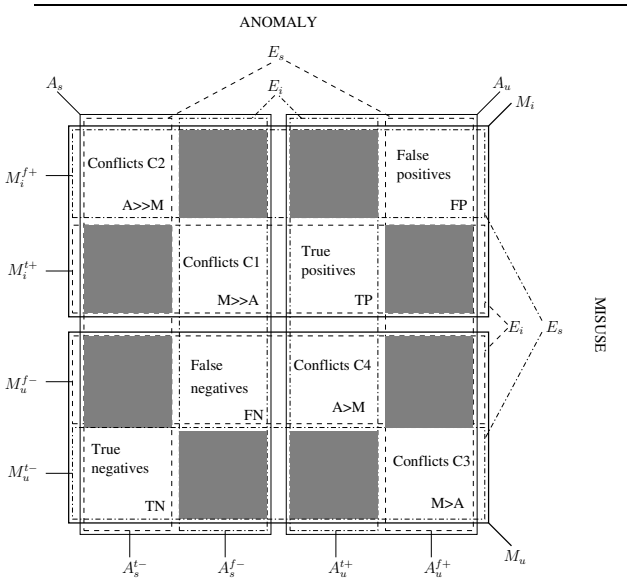

Figure 2. Combination of anomaly and misuse detection – sets in grey are empty because they are subsets of $E_s \cap E_i = \emptyset$; $X >> Y$ means that X is correct while Y is not ; $X > Y$ means that X is more accurate than Y

- *Let M_i^{t+} be the intersection of M_i and E_i that represents the set of true positives in the misuse detection $M_i^{t+} = M_i \cap E_i$;*
- *Let M_u^{f-} be the intersection of M_u and E_i that represents the set of false negatives in the misuse detection $M_u^{f-} = M_u \cap E_i$;*
- *Let M_u^{t-} be the intersection of M_u and E_s that represents the set of true negatives in the misuse detection $M_u^{t-} = M_u \cap E_s$.*

Since an event declared safe (resp. intrusive or unknown) is actually either a safe (E_s) or an intrusive event (E_i), the diagnosis is either correct (A_u^{t+}, A_s^{t-}, M_i^{t+} or M_u^{t-}) or a mistake (A_s^{f-}, A_u^{f+}, M_i^{f+} or M_u^{f-}). According to Definitions 4 and 5, we have the following property:

Property 3

$$\begin{array}{ll} A_u^{f+} \cup A_u^{t+} = A_u & A_u^{f+} \cap A_u^{t+} = \emptyset \\ A_s^{f-} \cup A_s^{t-} = A_s & A_s^{f-} \cap A_s^{t-} = \emptyset \\ M_i^{f+} \cup M_i^{t+} = M_i & M_i^{f+} \cap M_i^{t+} = \emptyset \\ M_u^{t-} \cup M_u^{f-} = M_u & M_u^{t-} \cap M_u^{f-} = \emptyset \end{array}$$

3.2. Resolving Conflicts

Using both anomaly and misuse intrusion detection combines the advantages of each approach, intuitively leading to a better qualification of the results.

A combination, however, implies possible conflicts between the two detectors. We list all the possible cases in the following using definitions and notations introduced in Section 3.1. We start with the cases where anomaly and misuse agree. Then, we list the actual conflicts and discuss when components are correct. The different cases are illustrated on Figure 2. When a component is correct while the other is not we denote it with $>>$. For example, $M >> A$ means that for the given conflict misuse is correct while anomaly is not. When the events are qualified by none of the components, the conflicts are somewhat softer than when at least one component qualifies the events. In that case, we only say that one component is more accurate than the other, and we denote it with $>$. For example, $M > A$ means that for the given conflict misuse is more accurate than anomaly.

- $A_u^{t+} \cap M_i^{t+}$: an intrusive event is declared unknown (i.e. potentially intrusive) by the anomaly detector and intrusive by the misuse detector. There is no conflict here, both components are correct, it is a set of true positives, TP.
- $A_s^{t-} \cap M_u^{t-}$: a safe event is declared safe by the anomaly detector and unknown (i.e. potentially safe) by the misuse detector. There is no conflict here, both components are correct, it is a set of true negatives, TN;
- $A_u^{f+} \cap M_i^{f+}$: a safe event is declared unknown (i.e. potentially intrusive) by the anomaly detector and intrusive by the misuse detector. There is no conflict here, both components are incorrect, it is a set of false positives, FP;
- $A_s^{f-} \cap M_u^{f-}$: an intrusive event is declared safe by the anomaly detector and unknown (i.e. potentially safe) by the misuse detector. There is no conflict here, both components are incorrect, it is a set of false negatives, FN;
- $A_s^{f-} \cap M_i^{t+}$: an intrusive event is declared safe by the anomaly detection while it is declared intrusive by the misuse detection. There is a conflict, anomaly is incorrect and misuse is correct, it is the conflict set, C1;
- $A_s^{t-} \cap M_i^{f+}$: a safe event is declared safe by the anomaly detection while it is declared intrusive by the misuse detection. There is a conflict, anomaly is correct and misuse is incorrect, it is the conflict set, C2;
- $A_u^{f+} \cap M_u^{t-}$: a safe event is declared unknown (i.e. potentially intrusive) by the anomaly detector and

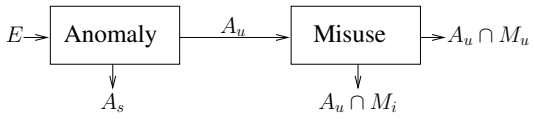

Figure 3. Serial combination of anomaly and misuse detection

also unknown (i.e. potentially safe) by the misuse detector. There is a soft conflict. Misuse is more accurate than anomaly, it is the (soft) conflict set, C3;

- $A_u^{t+} \cap M_u^{f-}$: an intrusive event is declared unknown (i.e. potentially intrusive) by the anomaly detector and also unknown (i.e. potentially safe) by the misuse detector. There is a soft conflict. Anomaly is more accurate than misuse, it is the (soft) conflict set, C4.

Note that all the other intersections are empty because they are subsets of $E_s \cap E_i$ which is empty after property 1. For example, $A_s^{t-} \cap M_i^{t+} = A_s \cap E_s \cap M_i \cap E_i$.

4. Anomaly Detection First Serial Combination (ADFSC)

In Section 2.3 we underlined that the HTTP traffic that we monitor is mostly safe. This has consequences on the relative size of the sets described in the previous section. This section briefly presents the serial combination of anomaly and misuse detection, with anomaly detection first (ADFSC). It then discusses on the formal model and why this is justified for HTTP traffic.

Figure 3 illustrates the serial combination of anomaly and misuse detection, with anomaly detection first. In this combination, the anomaly component filters out the events declared safe (A_s). Events declared unknown by the anomaly detection (A_u) are then analyzed by the misuse detection component. The operator is provided with three sets of events:

- A_s: the events in this set are, by definition, considered as safe by the operator. For an effective result of the combination, the anomaly model must be constructed such that the false positives that may be present in this set are low in number.
- $A_u \cap M_i$: this is the set of events declared unknown (i.e. potentially intrusive) by the anomaly detector, and intrusive by the misuse detector which actually confirms the results of the anomaly detector. The misuse component brings its ability to diagnose the alerts emitted by the anomaly component.

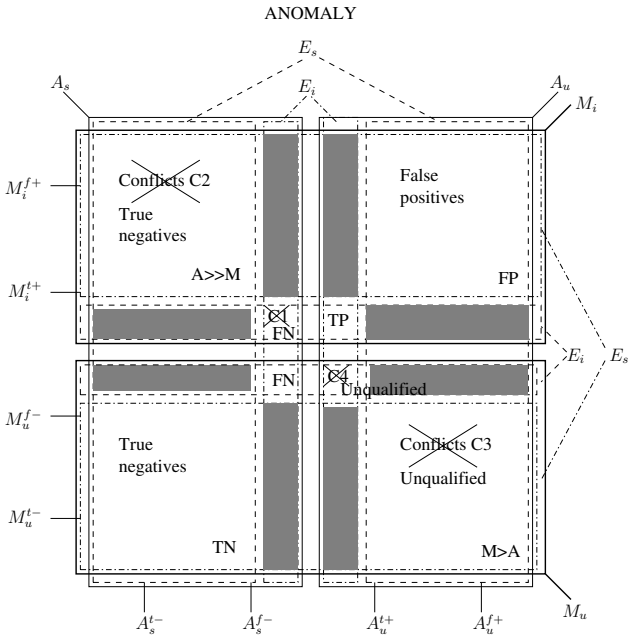

Figure 4. Serial combination of anomaly and misuse detection for HTTP traffic – sets in grey represents empty sets ; E_s is much larger than E_s ; conflict sets are re-interpreted

- $A_u \cap M_u$: this is the set of events declared unknown (i.e. potentially intrusive) by the anomaly detector and unknown as well (i.e. potentially safe) by the misuse detector. These events should be considered as false positives (respectively negatives) issued by the anomaly (respectively misuse) detector and the anomaly model (respectively the signatures database) must be updated.

Figure 4 updates Figure 2 taking into account the characteristics of the monitored events and the serial combination of anomaly and misuse detection. HTTP traffic being mostly safe means that E_s is much larger than E_i. Then the serial combination of anomaly and misuse detection allows to re-interpret the conflict sets.

For the conflict of C1 ($A_s^{f-} \cap M_i^{t+}$), the anomaly component is wrong. Not sending these events to the misuse component means that the conflict is ignored by ADFSC, these events become false negatives. The challenge of the approach we propose here is thus to make sure that C1 set is negligible, namely that the anomaly component does not declare safe too many intrusive events (i.e. false negative issued by the anomaly component low in number).

For the conflict of C2 ($A_s^{t-} \cap M_i^{f+}$), the anomaly component is right. Not sending these events to the misuse component means that the conflict is properly solved by ADFSC, these events become true negatives.

The conflicts of C3 ($A_u^{f+} \cap M_u^{t-}$) and C4 ($A_u^{t+} \cap M_u^{f-}$) are still present. However, as both components actually do not qualify these events, they can be simply interpreted as unqualified, containing safe events as well as new attacks.

These events require further investigation in order to be qualify as safe or intrusive, i.e. false positives (respectively negatives) issued by the anomaly (respectively misuse) detector. The anomaly model (respectively the signatures database) must then be updated. The way this update is realized is out of the scope of the paper. Nevertheless, one of our operational objectives is to minimize the cardinality of $A_u \cap M_u$ so that a human processing by the security administrator is possible. As a consequence, the anomaly detector must filter out a high amount of safe events

The serial combination presented here, feeding the misuse component with unknown events from the anomaly component, principally addresses the problem which arises when using a single misuse component, the false positive rate. Indeed the anomaly component task is to filter out safe events among which some would have lead to alarms if processed by the misuse detection component. However, the subsets $M_i^{t+} \cap A_s^{f-}$ and $M_u^{f-} \cap A_s^{f-}$, which contain attacks, are also declared safe by the anomaly detection. This implies a constraint on the anomaly model constitution in order to avoid anomaly detection false positives.

5. Experimental Results

We use for our experiments an ADFSC as the one illustrated in Figure 3. In fact, the anomaly detection component is integrated just behind the normalization module in the WIDS in order to take advantage of the requests normalization (such as decoding) and thus avoid masquerading. The HTTP traffic used for our experimentations comes from the Supélec web server log files. These log files are less than one year old and we have a full access to the web server. We did not use log files from FT web server because they were too old (2001) to be replayed against the current FT web site and thus we could not do real verification.

5.1. The Anomaly Detection Model

The anomaly model is based on the couple made by the resources accessed and the combination of parameters used, if any. For instance, the following requests:

- `/backgroung.jpg`
- `/index.php?subject=network&news_id=85`
- `/index.php?subject=security&news_id=23`
- `/index.php`

lead us to add the 3 following couples in our model:

- (`/backgroung.jpg`, { })
- (`/index.php`, { `subject`, `news_id` })
- (`/index.php`, { })

The anomaly model is constructed using a list of such couples from a training set: distinct couples are extracted from this training set and classified in the ascending order according to their number of occurrence. Thus, we focus on the most requested resources. Each couple is selected by an operator, *i.e.* manually qualified as a safe request or not. According to the measures presented in Section 2.3 and illustrated in Table 2, it is humanly possible to handle such a task with a few days training set. In the experimentation presented below, we have to verify a thousand resources and this took us a few hours.

The resource accessed is not the only useful criterion to qualify a request as safe or unknown. Each field of the log entry and also the combinations of some fields, may be useful criteria as well. For instance, some resources may only be reachable from a specific intranet; in this case, the client IP address is an important criterion. For these experimentations we only use the resource name and the combination of its parameters, if any. If the requested resource belongs to the anomaly model, the anomaly detector checks if the combination of parameters used is allowed or not, such as what is done by DIDAFIT [7, 8].

Since web sites are organized as trees, this leads us to a tree structure. So far, we have used a simple tree structure for our experimentations: each node contains a character of the resource name, each leaf contains the constraints relative to the resource. A resource is thus a branch from the root node to a leaf. Whereas this structure is very simple, it is efficient enough for our experimentations.

The tree anomaly model is used as follows: when a resource is requested, the anomaly detection first checks if this resource belongs to the model. If not, this request is declared unknown, else, associated constraints are applied to the incoming request. If the constraints are satisfied, the request is declared safe, if not, the request is declared unknown by the anomaly detection.

5.2. Qualitative Results

The first seven days are used as a training set while the remaining 85 days are used as test set. The training set is composed of 191,976 log entries which are requests toward 17,465 distinct resources. Our anomaly model is composed of 17,313 selected resources from the training set. All static resources are included in the anomaly model. Dynamic resources such as some PHP scripts and counter managed by the webmaster are included in the

	Single Misuse Detection	Serial Combination	Filtering Rate
Severity	log entries	log entries	Percent.
0	1,747,179	100,395	94.25
1	266,401	10,704	95.98
2	166,470	8,711	94.76
3	16,410	485	97.04
4	13,159	382	97.09
5	409	73	82.15
6	353	154	56.37
7	24	9	62.50
8	12	12	0
9	4	4	0
>9	352	352	0
All	2,210,773	121,281	94.52

Table 3. Filtering rate toward Supélec log files. The percentages are rounded at the second figure after the dot.

model because these resources are are well-known by the operator, while dynamic resources managed by the academics were excluded from the model.

In order to evaluate the filtering rate of the ADFSC and its ability to address the problem of false positives, we perform a single misuse detection using our WIDS component, and compare its results to the ADFSC results (see Table 3).

The first column shows the severity levels that can be associated to a logged event. The second column shows the results of the single misuse detection using the WIDS presented in Section 2.3 on the Supélec web server test set. The third column shows the results of the serial combination presented in Section 4 using the anomaly detection component described in Section 5.1 and the WIDS presented in Section 2.3. The fourth column is the filtering rate observed between single misuse detection and the serial combination for each severity level:

The anomaly detection component in the serial combination filters out 94.25% of null severity logged events (A_s). The set of unknown events from misuse detection (M_u) decreases from about 1,7 million events to about 100 thousands events that are unknown for both anomaly detection and misuse detection ($M_u \cap A_u$). These logged events are requests toward static resources such as pictures (GIF) and HTML pages. The 4.5412% remaining are requests toward resources that have not been learnt by the anomaly model. They may be new resources or resources that were not requested during the training set.

The serial combination is also efficient in filtering low severity levels (between 1 and 4). More than 94% of the events with a severity up to level 4 are filtered out by the anomaly detection component (A_s). The logged events filtered out by the anomaly detection component are requests toward dynamic resources managed by the webmaster (*e.g.* the search engine and some CGI evoked in Section 2.3). The logged events not filtered out by the anomaly detection component are unknown uses of the search engine (*ie* unknown combination of parameters) and resources (both static and dynamic) managed by academic authors that contain specific encoding or ambiguous file name that match some signature of the misuse component. The anomaly detector of the ADFSC filters out a large part of the events that would have generated false positives if analyzed by the WIDS tool. Thus, the FP set is negligible regarding the C2 set.

The filtering rate for severity level between 5 and 7 is between 56% and 82%. Whereas severity levels between 1 and 4 are most likely to be false positives due to the high sensitivity of the WIDS tool, severity from 5 to higher mainly address critical events. Thus for the severity levels 5 to 7, the anomaly detection component addresses the problem of false positives since events that would have been declared critical by the misuse detection component are filtered out by the anomaly detection component when the ADFSC is used.

Some false positives remain. As an example, the Supélec web server is using the Count.cgi program. Even if the actual version has no known weakness, it is important to monitor this kind of activity since a former version can be used without the webmaster knowing. In the case of the Supélec web server, it appears that this program is only available in a specific directory. So we added the Count.cgi program, its path and usual parameters to the anomaly model as safe event. Thus any other instance of this program on the web site is analyzed by the misuse detection component. With the single misuse detection, this program causes 227 alarms of level 4. It only causes 27 alarms of same level with the serial combination. After investigation, it appears that the Count.cgi program was just not properly used by an academic author. Using the single misuse detection, the flood of alarm overwhelms the operator and this error may remain unseen, while using the serial combination, this error is easy to detect thus to correct.

Other alerts are mainly generated by attempts toward Microsoft FrontPage extensions. The low level of severity associated to this behavior is due to the small number of signature matched and the fact that these attacks were unsuccessful. Knowing that the web server is not vulnerable to this attack, the operator can add these requests to the anomaly model without cancelling any signature in the misuse component.

From severity level 8 to higher, investigations revealed

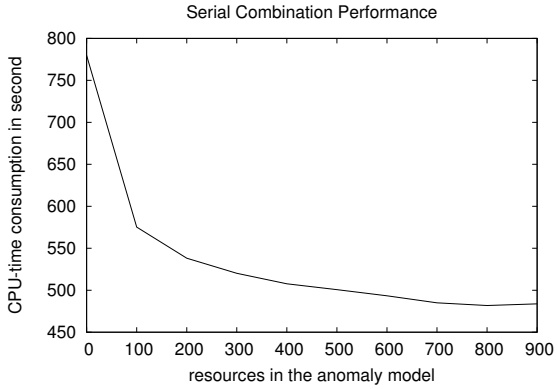

Figure 5. ADFSC CPU-time consumption

that all the alerts are generated by nimda attempts. With such events, it is a good thing that the anomaly component does not filter out anything.

The last line of Table 3 sums up the overall results for the entire test set. The overall filtering rate shows that the anomaly detection component filters out 94.52% of the log events. This means that the set of events declared safe by the anomaly detection (A_s) is composed of 2,089,547 events. Since the model is hand made, we consider that the false negative sets from anomaly detection (A_s^{f-} which is composed of FN and C1) are negligible regarding to the true negative sets from anomaly detection (A_s^{t-} which is composed of TN and C2). Comparing to the results presented in Section 2.3, we can see that the amount of safe events (E_s) in HTTP traffic is higher than the 79% exposed with our WIDS.

5.3. Performance

To show that our serial combination do not overload the intrusion detection process, we compare the CPU-time consumption of the WIDS used alone and of the ADFSC.

This experiment is realized using one day of log as a training data set and the six next days as test sets. The training set is composed of 23499 log entries and 13506 distinct resources and the test set is composed of 168477 log entries and 16552 distinct resources.

In order to show the influence of the anomaly model toward the CPU-time consumption, we experiment ADFSC with a more and more complete anomaly model. A simple statistical analysis is initially done on the training set in order to classify the requested resources in the ascending order. Then, the anomaly model is filled with one hundred of the safe resources, and the CPU-time consumption is measured. This operation is repeated up to nine hundred safe resources in the anomaly model.

Measures of CPU-time consumption are illustrated in Figure 5. The first measure is realized with the misuse detection component alone. ADFSC is less CPU-time consuming than the single misuse detection component. The high decrease we can observe with only one hundred resources in the anomaly model is a consequence of the measures presented in Section 2.3, namely a high rate of requests is safe and these requests target a small subset of the resources available on the web server.

6. Related Work

In Section 2.1 we have evoked the work of Vigna et al. in [11] concerning IDS designed to monitor web server activity. In this section we discuss work around the combination of intrusion detection components and policy-based intrusion detection.

Intrusion Detection Component Combination

Another combination of intrusion detection component is used by EMERALD [9] and its predecessors (IDES [4] and NIDES [6, 2]). This combination consists in analyzing incoming events in parallel. One analysis is done with a misuse detection component and another analysis is done with an anomaly detection component. While standalone misuse detection and standalone anomaly detection provides two sets of results (intrusive or unknown and safe or unknown), this parallel combination provides four sets of results, such as those presented in Figure 2: $A_s \cap M_i$, $A_s \cap M_u$, $A_u \cap M_i$ and $A_u \cap M_u$. Two of these sets, $A_s \cap M_i$ and $A_u \cap M_u$ are linked to conflicts between misuse and anomaly detectors. These conflictual sets need further investigations in order to fully qualify events. This task is handled by a correlation component (so called a resolver). Clearly, this approach in more generic than ours, but all the intelligence is in the resolver, which must be adapted to the characteristics of the input data set. According to our best knowledge, there is actually no publication which describes this resolver in full detail, with presentation of the results obtained on real data.

This parallel architecture implies a double analysis of each event, even when it can be known beforehand that some events are intrusive or safe.

The parallel combination used by EMERALD aims at improving intrusion detection results for every kind of event (network, system calls, applications). At the opposite, we chose to focus on a specific set of events (HTTP traffic) and thus deduce the most effective combination to improve the intrusion detection diagnosis given the characteristics of such an input data set.

Policy-Based Anomaly Detection

Many anomaly based detectors exist. It is not our purpose here to compare our simple anomaly detector to

existing stand alone anomaly detection components. We just give here a few elements about DIDAFIT [7, 8], a policy-based anomaly component, which offers an anomaly model close to ours. DIDAFIT is designed to monitor SQL transactions between databases and applications such as web servers. Its model of behavior is based on allowed queries, as our is based on allowed requests. These allowed queries are listed and grouped using some pattern matching features. We use a tree structure for the same purpose. In our work, there is no control over the HTTP request parameters values when DIDAFIT controls the SQL requests parameters values.

7. Conclusion

In this paper, we have provided a framework to systematically reason about the combination of anomaly and misuse components. Depending on the input data set and on the designer objective, each combination presents some advantages. Characteristics measured in web servers context lead us to ADFSC, an architecture in which an anomaly detector comes first, feeding a misuse detector with potentially intrusive events, is accurate.

The anomaly detector brings its ability to recognize safe events. The unknown events are then analyzed by the misuse detector which brings its capacity of diagnosis. While known events are directly declared safe by the anomaly detector, the misuse detector provides the operator with qualified events. In such a context, the operator task is less time consuming and its ability to take countermeasures is improved

The monitored events are separated between normal, intrusive or unqualified classes (i.e. not known as an attack, but not recognize as safe either).The events neither recognized by the anomaly detector, nor by the misuse detector may correspond to new attack, previously unknown, but nevertheless detected, or to unknown safe resources newly available on the web server.

The major constraint we had to face was to design an anomaly detection component which generates a low rate of false negatives. To do so, we have used a simple anomaly component with a hand-made model.

The experimental results presented here show that the approach drastically reduces the amount of false positives and unqualified events. The anomaly detection component filtering out a high amount of events that would have been declared intrusive by the standalone misuse detection component, the ADFSC indeed clearly addresses the problem of false positives. The unqualified events require further investigation in order to be qualified as safe or intrusive. The anomaly model or the signatures database can be updated as a consequence. The drastically reduced amount of unqualified events allows a manual updating process by an operator.

Finally, the serial combination of the anomaly and misuse detection component presents a reasonable CPU-time consumption compared to a standalone misuse detection component, as we use a simple anomaly detection model.

References

[1] Magnus Almgren, Hervé Debar, and Marc Dacier. A Lightweight Tool for Detecting Web Server Attacks. In *Proceedings of NDSS 2000*, pages 157–170, 2000.

[2] Debra Anderson, Thane Frivold, and Alfonso Valdes. Next-Generation Intrusion Detection Expert System (nides) - a Summary. Technical Report SRI-CSL-95-07, SRI, May 1995.

[3] D. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

[4] Dorothy E. Denning, D. L. Edwards, R. Jagannathan, T. F. Lunt, and P. G. Neumann. A prototype IDES — A Real-Time Intrusion Detection Expert System. Technical report, Computer Science Laboratory, SRI International, 1987.

[5] S. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion Detection Using Sequences of System Calls. In *Journal of Computer Security*, volume 6, pages 151–180, 1998.

[6] Harold S. Javitz, Alfonso Valdez, Teresa F. Lunt, Ann Tamaru, Mabry Tyson, and John Lowrance. Next generation intrusion detection expert system (NIDES) Technical Report A016–Rationales, SRI International, March 1993.

[7] W. L. Low, J. Lee, and P. Teoh. DIDAFIT : Detecting Intrusions in DAtabases Through FIngerprinting Transactions. In *Proceedings of ICEIS 2002*, pages 121–128, 2002.

[8] W. L. Low, S. Y. Lee, and P. Y. Wong. Learning fingerprints for a database intrusion detection system. In *Proceedings of ESORICS 2002*, pages 264–280, 2002.

[9] Phillip A. Porras and Peter G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of 20th National Information Systems Security Conference*, 1997.

[10] Prem Uppuluri and R. Sekar. Experiences with specification-based intrusion detection. In *Proceedings of RAID'2001*, LNCS 2212, pages 172–189, October 2001.

[11] G. Vigna, W. Robertson, V. Kher, and R.A. Kemmerer. A Stateful Intrusion Detection System for World-Wide Web Servers. In *Proceedings of ACSAC 2003*, pages 34–43, December 2003.

[12] J. Zimmermann, L. Mé, and C. Bidan. An Improved Reference Flow Control Model for Policy-Based Intrusion Detection. In *Proceedings of ESORICS 2003*, October 2003.