Intrusion Detection Systems Elevated to the Next Level

Frank Becker, Matthias Petermann

December 4, 2005

1 Introduction

The name "Intrusion Detection System (IDS)" suggests one to get something that deployed in a network alarms you in case of an attack. Systems that also try to block those attacks are known as Intrusion Prevention Systems (IPS). The magic of deciding what is an attack and what is normal is left most of the time to single sensors matching traffic patterns or observing system activity. One who has operated such a system knows what he gets. Sometimes it works quite well, often it fails for several reason.

Those times, systems in a larger local network or on the Internet are permanently threatened by attacks of all kinds. So called "Internet-Worms" exploit vulnerabilities of applications or operating systems and use their capabilities to infect further systems in the network. Some of them include functionalities of Trojan horses or bots - they run hidden in the background and observe the user while he is working. They collect private or confidential data and post it to someone somewhere in the Net. In the meantime, the attacker has already got full control of the system. Just a normal day: Someone doing his business on your machine. Spreading Spam, DDOSing web servers, collecting passwords, or preventing you from playing the non-licenced MP3 via root-kits.

Not to mention that manual attacks are the great danger, actually.

Protecting systems and networks first has to be done on the system itself. Another layer are systems such as packet filters, application layer gateways, virus filters and so on. They do a good job - but are far from being perfect. Especially when it comes to zero day exploits the virus scanner cannot do anything. In fact, there is no effective protection if you are stuck to the requirement of the one operating system or application.

In sensitive environments there is at least a big need to recognise that an attack has happened at all. Only then it is possible to isolate infected systems, evaluate the damage and take measures against the origin.

Intrusion Detection Systems help to recognise evidences of attacks, give a hint of the origin and the destination of the attacker, and also help evaluate the incidents. They only can be efficient if several IDS techniques are combined together to give a picture of what has happened or to alarm of a certain event or combination of events. Further a decent number of distributed sensors are the only way to detect distributed attacks the spreading of worms and such patterns in large environments such as company networks.

2 IDS Technologies

There are two classes of IDS - network based IDS (NIDS) and host based IDS (HIDS). While NIDS aim to analyse the data crossing the network, HIDS reside on the hosts and keep track of every suspect occurrences. In the Open Source world there do exist many interesting projects that cover one of those technologies, each.

2.1 Snort

A well-known NIDS is Snort[1]. Snort is a so-called Packet-Sniffer. That means it analyses all IP packets that pass a specified network-interface. The analysis serves in two passes.

In the first part Snort uses pre-processors to detect anomalies on packet level, that includes the ability to detect port scans, manipulated packets

[1] http://www.snort.org

and denial of service attacks. There exists also a patch which makes use of the free ClamAV[2] virus scanner, so viruses can be detected just in time they got downloaded from the web.

The second pass depends on a special preprocessor - the stream-pre-processor, whose purpose is to reassemble packets to their original order in tcp-streams or known udp-protocols. They become directed to a pattern matching engine where they become investigated for known attack patterns and signatures of defective code, for example buffer overflows, exploits, worm signatures and so on. The signatures can be verbalised as a set of rules. Beside the chance to create own rules they can be got from third parties, for example the Bleedingsnort[3] project or from commercial suppliers like Sourcefire[4].

Newer versions of Snort are also able to interact with the Netfilter implementation of the Linux kernel in a way to let it work as an Intrusion Prevention System (IPS). This feature is called the inline-mode.

2.2 NetFlow

Beside the investigation of packet's content it is also interesting to know in which quantities packets flew between hosts at a particular time.

With a focus on getting statistic information of network utilisation the NetFlow protocol was invented by Cisco. The products which initially were built around this protocol provide traffic accounting services. The measure unit for NetFlow accounting is a flow, which is defined as a description of a packet flow from one host to another. The corresponding flow record contains information about source IP, target IP, source port, target port, flags, payload and the connection time.

A common implementation of NetFlow consists of two components. The one which is responsible for gathering the flow records out of the packet flow is called the flow-probe. Typical flow-probes export the gathered flow records from time to time though UDP packets to so-called NetFlow collectors, which are the other part of the implementation. These collectors receive the flow records and store them - for example in a database - for further proceedings.

While some active network components still have built-in flow-probe-functionality, on Linux systems one can use special software like fprobe[5] to extend any system with flow-probe capabilities. You will also find some free tools for the counterpart, for example the flow-tools[6].

IDSs can take benefit of NetFlow especially for detecting anomalies in network utilisation. Imagine a office department with working hours fixed to the daytime. It would be very suspicious if there would be large amounts of network traffic at night.

2.3 System log files

Thinking about host based IDS the first thought should be that there are a lot of log files, for instance on Unix systems. They run a centralised syslog service where almost every service (web server, mail server, ...) can connect to and dump its log data - difficult to manipulate afterwards in order to wipe out tracks of an attack.

By parsing the collected log files one can get information for example about:

- Failed login attempts
- Successful logins
- Access to web- and mail-services
- Firewall logs (blocked/accepted packets)
- Changes of the system configuration
- Creation of new user accounts

2.4 File fingerprinting

Another host based technology is file fingerprinting. When it comes to a successful intrusion, it is important to keep track of possible manipulations of system files. This can be achieved by building a cryptographic hash (fingerprint) over the content of each file and store them on a secure place. To check for modifications one has only to proceed a repeated hashing cycle and compare the hashes with the original ones.

With Samhain[7] you can get a quite powerful file integrity checker. Beside modifications of the file's content it is able to recognise:

- Changing file access rights

[2] http://www.clamav.net
[3] http://www.bleedingsnort.org
[4] http://www.sourcefire.com
[5] http://fprobe.sourceforge.net
[6] http://www.splintered.net/sw/flow-tools/
[7] http://la-samhna.de/samhain/

- Changing file owner / group
- Creation of new files
- Deletion of files

2.5 Syscall monitoring

On Unix based systems every library call - for example reading from a file - raises a couple of system calls to the kernel. The idea behind syscall monitoring is to keep track on syscalls and log anomalies. So it would be possible to detect:

- Opening, reading and writing files
- Opening of network sockets
- Forking new processes
- Execution of files

Systrace[8] is an implementation of a security layer for syscalls which was originally developed for OpenBSD[9]. Today it is also included in the NetBSD[10] base system and available as a kernel patch for Linux.

Systrace contains a kernel interface and user space tools to enforce free definable syscall policies on a per-user/per-process level. Processes that violate their policy during execution were logged.

For instance, it makes sense to enable Systrace for a web server and define a policy that denies that the server is able to open the system's password file. Even a completely misconfigured server would not be able to access the file, because it would be forbidden on kernel level. Also in case of a buffer overflow which would break the security constraints at application level, it could not read the file. Instead the policy manager makes a log entry for this event.

2.6 Virtual honeypots

A very valuable resource for IDS are honeypots. Honeypots are dedicated systems with potential vulnerable software installations. Their only purpose is to act as a trap for attackers.

Because functionality of this systems is not part of the daily business, every access to them is potentially suspicious. For example, mounting a file share on such a system, or trying to send mail through it. Running honeypots can help while studying the behaviour of attackers and serve as a supplement to emphase alerts from other IDSs.

As it is not really efficient to maintain separate machines just for this purpose it is useful to virtualise them. A project that provides a virtual honeypot software is honeyd[11]. Honeyd runs on most common Unix-like operating systems and is able to emulate a complete network environment, including routers and their latency. The virtual hosts are highly configurable by own scripts. Access to one of the hosts can be instantly logged.

3 Current problems

The introduced software systems are not designed to work together. Beginning with the circumstance that each of them uses a different format for log output, there is no tool to analyse the collected information from different IDSs at a common place.

For example, if a worm hits a network part it often does not affect just one IDS. When it uses a buffer overflow to spread itself, the buffer overflow would possibly be detected by a NIDS. If the worm changes a file on infected systems, this would be detected by a HIDS. Many different IDS could be triggered by the attack. While spreading around the network, such a worm produces a lot of events on different IDSs which all log to different places. The quantity of events quickly increases to a point where no human administrator would be ever able to separate the important from the less important, or even recognise a coherence in between them.

4 Guide to a solution

For an ideal solution we define three main goals:

1. A standardised data format
2. Centralised data storage
3. A common analysis tool

4.1 A standardised data format

Each of the IDSs stores its data in its own format. There is no generalisation. NIDSs provide quite different data than HIDSs, and as different are their data formats.

[8] http://www.citi.umich.edu/u/provos/systrace/
[9] http://www.openbsd.org
[10] http://www.netbsd.org
[11] http://www.citi.umich.edu/u/provos/honeyd

For centralised storage and analysis the first requirement is to normalise the different formats to a common one. The Internet Engineering Task Force (IETF) if working on a very promising approach of an universal data format for IDSs. It is called the Intrusion Detection Message Exchange Format (IDMEF[12]). Currently the draft undergoes an evaluation and has a good prospect to become declared as a RFC.

In technical sense IDMEF stands for an object oriented data format which consists of extensible classes. The current draft suggests the implementation in XML.

4.2 Centralised data storage

To enable centralised analysis it is essential to hold all the necessary data at one centralised place. This can be achieved by providing a framework - consisting of a centralised storage and an common interface for existing IDSs.

Prelude-IDS[13] is such a framework. It is called a "hybrid IDS" because it can utilise all the introduced IDSs as sensors for putting their events in a common database, using the IDMEF. The events were sent out to a special server process, the Prelude-Manager, through a SSL encrypted and authenticated connection. Some of the sensors - for example Snort and Samhain - already include direct library support for the Prelude-IDS. Others that only report to Syslog, can be imported by a special sensor called Prelude-LML. This sensor is able to read any kind of log file and parse it by Perl compatible regular expressions. The extracted values can be mapped to IDMEF. Prelude-LML is highly configurable and fits for many cases. For special needs which cannot be fulfilled by Prelude-LML, there are also excellent C-, Python- and Perl-Bindings to include Prelude's functionality into other sensors.

4.3 A common analysis tool

Prelude-IDS includes PreWikka which is a web-based IDS-console. With PreWikka one can browse all the events reported by the connected IDSs, put filters on them and perform basic inspection tasks.

[12] http://www.ietf.org/internet-draft/draft-ietf-idwg-idmef-xml-14.txt

[13] http://www.prelude-ids.org

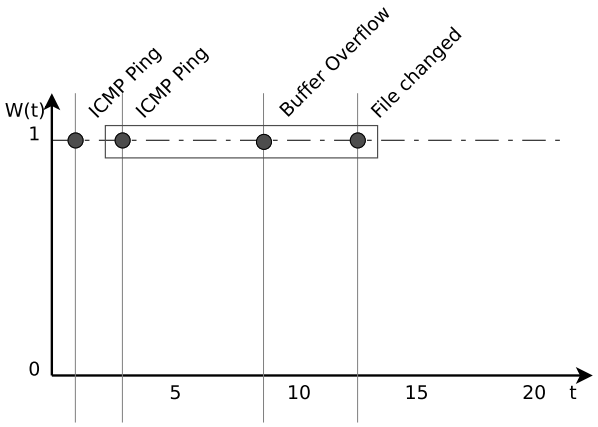

Figure 1: Simple correlation with logical conjunction - successful

5 An approach for correlation

With Prelude-IDS all events are available on a single place. Since PreWikka lacks correlation the main problem remains - when it comes to a large quantity of reported events no human can handle them at all. All the single events give no reliable information if there was really an attack or whether they are just an accumulation of random events with no relationship to each other.

So we decided to implement a rule-based correlation module. The purpose of such a module is to combine related events to an incident. This leads to a strong mitigation of the information the administrator has to evaluate.

5.1 Basics of correlation

Relationships between separate events can be expressed by a set of rules. A very simple correlation approach could be the inspection of a particular time window with the assumption that a specified set of events appear together. If that assumption would be fulfilled, the events become an incident.

A demonstrative way to show the flaws of such a system is to take a typical pattern how a simple Internet worm could act:

1. Worm sends out frequently broadcast pings from an infected host to the network
2. Worm attacks answering hosts with a Buffer Overflow
3. Worm changes a file on each newly infected host

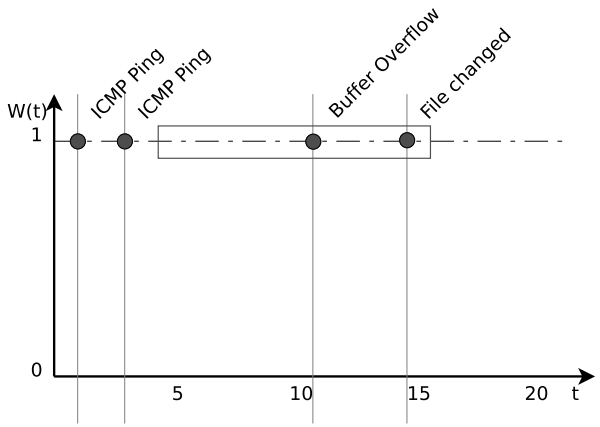

Figure 2: Simple correlation with logical conjunction - failed

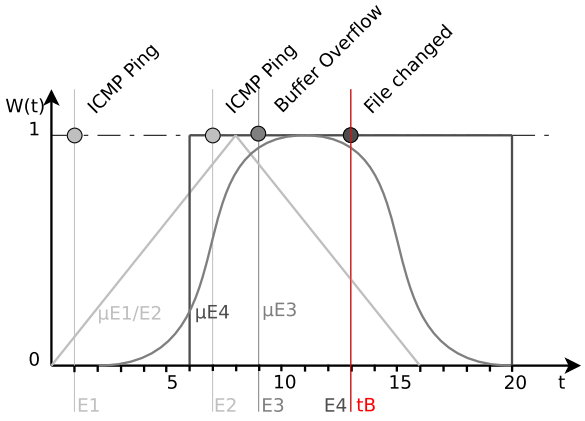

Figure 3: Events with assigned affiliation functions

Assuming that the whole process would take 10 seconds a matching rule can be constructed. It would consist of a logical AND-conjunction of the event "ICMP Ping", "Buffer Overflow" and "File Change".

As shown on Figure 1 that would work fine for the exact order of events within the expected time window. The big flaw would be exposed if one of the events runs a little bit out of the time window (Figure 2). Even when the behaviour of a worm is well-known and carefully described in the rule, divergences can happen as a result of load differences or network latency. As the rule uses logical conjunctions there are only the states "true" and "false". If one event is missed, the whole rule would lead to a sharp, wrong decision - in this case "false" which means the system would not detect the attack.

To avoid short wrong decisions a fine grained rating system must be introduced. A look at modern cybernetics opens a promising prospect to fuzzy logic which allows to utilise such a fine grained rating.

5.2 Fuzzy technologies

Fuzzy technologies were often associated with the term fuzzy logic[14] which is the most popular application for them. The basic principles of fuzzy logic are the set theory and the logic. The set theory deals with the conjunction of sets and the logic deals with the conjunction of conclusions. A special form of the logic is the Boolean algebra. Conclusions in Boolean algebra can only adopt two states - either true or false. The same applies on the resulting set which can be described by a two-valued affiliation function.

To achieve a smooth transition instead of using only two states the two-valued affiliation function has to be replaced by a continuous one. This function builds the foundation for the so called fuzzy set. A fuzzy set is defined by the ordered pair $(X, \mu M)$. X represents the basic set and μM is the continuous affiliation function. The affiliation function maps the set X to the interval $[0, 1]$ and shows how much the resulting value belongs to the fuzzy set.

5.3 Applying fuzzy technologies to IDSs

Goal of using fuzzy technologies in IDSs is to provide a facility to program smooth rule sets for unsharp detection of related events.

Therefore every type of event which is expected to be related to a certain incident will be assigned to an affiliation function. As the events are lined up on some kind of time bar the value of the function at the particular time stamp where it occurs expresses how much the event belongs to the supposed incident.

Employable affiliation functions have a definition range in between the real numbers and a value range in the interval $[0, 1]$. A few examples you can see in Figure 4.

The practical proceedings are quite different from the formerly introduced logical conjunction based approach. Since values of the affiliation

[14] http://en.wikipedia.org/wiki/Fuzzy_logic

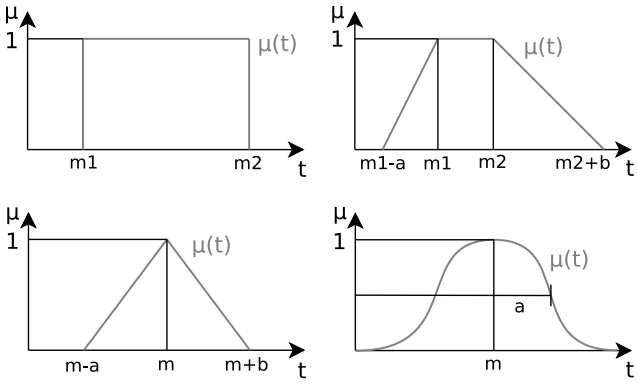

Figure 4: Examples of affiliation functions

functions depend on time there must be defined a special point in time which acts as relation point for the rule. A good choice for such a point is a critical event. In our example this would be the event "File Change". Further, the rule must describe types of events that could belong to that event and how likely it is affected by it.

In the example shown before there was a buffer overflow detected before the file has changed. Assuming, practical experiences had lead to the conclusion that this particular worm changes the file in-between 5 seconds after the buffer overflow, one can construct a related rule (see Figure 3). To avoid misinterpretation if the buffer overflow occurred slightly earlier or later, the buffer overflow gets assigned a affiliation function that slowly fades away - for example the Gauss function.

Nearly the same thing has to be done for the event "ICMP Ping". Since Pings are not as unusual as most of the critical events it is not practical to measure it on one single occurrence. So additionally one could deploy a count function which measures the frequency of the occurrence of the single "ICMP Ping" events and calculates a single value of them.

At the end we get a value in the interval $[0, 1]$ for each type of events addressed by the rule. It expresses the grade of affiliation of the occurred event to the incident.

To get a common result for this single values we can use the point-conclusion operator from the likelihood theory. So the single values only need to be multiplied with each other. The result is again a value in the interval $[0, 1]$ which can be used as an indicator for the likelihood the requested attack (incident) has happened.

5.4 Results and prospects

The fuzzy based correlation engine enables one to program large rule sets for known attack schemes. It also has the ability to show a little bit artificial intelligence when rules were programmed unsharp enough to cover generic attack patterns.

The output of the correlation engine can be used to feed analysers or trigger instant messaging systems. Also, the techniques of neural networks could help to tune the parameters of the affiliation functions or rating the final result.

6 Acronyms

HDNAS Hybrid Distributed Network Analysis System

HIDS Host based Intrusion Detection System

IDMEF Intrusion Detection Message Exchange Format

IDS Intrusion Detection System

IETF Internet Engineerung Task Force

IPS Intrusion Prevention System

NIDS Network based Intrusion Detection System

RFC Request For Comments