LinuxFocus article number 294
http://linuxfocus.org

IDS - Intrusion Detection System, Part II

by Klaus Müller
<Socma(at)gmx.net>

About the author:
At present "Klaus Müller a.k.a. 'Socma' is still a student, he keeps busy with Linux programming and with security related topics.

Translated to English by:
Jürgen Pohl
<sept.sapins(Q)verizon.net>

Abstract:

In Part I we focused on typical attacks on Intrusion Detection Systems. Part II introduces methods for their discovery and our responses - the application of signatures and filters among them. Finally, we are introducing Snort and LIDS.

Analysis Possibilities

Previously we elaborated on attacks to protect IDSs and various existing systems against. Next, we cover the methods of analysis and how an IDS determines if there was an attack or not, respectively if the attack was successful or not.

Basically, we are differentiating between Misuse Detection and Anomaly Detection. Misuse Detection is utilizing specific defined patterns to unmask an attack. These pattern are called "signatures", they will be discussed in a dedicated section. For now we need to know that we can define signatures which search the network traffic for certain strings (e.g. /etc/passwd), deny access requests to specific files and raise an alert. The advantage of Misuse Detection is the low probability of false alarms since the search criteria of signatures can be tightly defined. The disadvantages are also obvious, new attacks are frequently missed because they were not defined (see section....on signatures).

The other method is Anomaly Detection. It simply means a profile of the user's normal activities was generated. If the user's behavior deviates too much from the profile an alarm is being triggered. The first step of this analysis is the creation of profiles (data base) of "normal" user activities. A variety of steps may be recorded: How often does the user execute specific commands? When does he execute specific commands? How often does he open specific files? ....One small example: - User "Example" executes */bin/su* three times a day (this value would be in the profile). Suddenly - one day - user "Example"

executes *su* seven times a day, more than twice as often than normal. Anomaly Detection would detect this "abnormal" behavior and warn the administrator about user "Beispiel's" seven *su* executions while three are the "normal" average. The disadvantages of this procedure became clear to me when I began with its implementation (see example at the end - COLOID). The method to install a database for user activities is quite computation intensive. We monitor, for example, how often the user has opened ten specific files. With each *open()* command has to be checked if it is one of the ten specific files and if the result is positive, the corresponding counter is going up. Nevertheless, this is a great opportunity to uncover new techniques of attack since they will most likely show up as "abnormal". Furthermore, the administrator himself can define which deviation shall be determined to be "abnormal", e.g. a deviation of 10% or even 75%....By utilizing this method we have to watch out to generate the user profiles in a "safe" network, otherwise the behavior of the attacker will be regarded as normal and the conduct of the legitimate user as abnormal.

In general Anomaly Detection includes the following procedures:

- Threshold Detection = in this area counters are being utilized, they are counting how often what is being executed, opened, started...This static analysis can be augmented by the so called Heuristic Threshold Detection.
- Rule-Based Detection = based on defined rules, should the usage deviate from the rules an alarm is being triggered.
- Static Measure = the behavior of user/systems conforms to a signature which was either pre-defined or established by other means. A program to log normal user activities for the definition of the signatures is often included.

Heuristic Threshold Detection in this case means the counter (how often what may be executed) is initially not set to a static but a dynamic value. Does a user normally execute */bin/login* 4 times the counter would possibly be set to 5 ...
The Protocol Anomaly Detection represents a sub-group of the Anomaly Detection. This is a relatively new technique, it works basically like the Anomaly Detection. Each protocol has a "pre-defined" signature (see corresponding RFC's). It is the goal of the Protocol Anomaly Detection to find out if the behavior of the protocol is like pre- defined or not. More attacks are based on protocol misuse than one might believe, this sub-group is therefore quite important for IDSs. Looking back at the section on scanning we may find some
indicators to Protocol Anomaly Detection.

- Check if no flag is set (NULL scan)
- Check if all flags are set(XMas scan)
- Check if "nonsense" combinations of flags are set, like SF
- .....

In related RFCs we will find the correct specifics, also what kind of behavior should not happen, resp. which kind of behavior should happen in response to a specific event. In addition there is Application Anomaly Detection (it works nearly like Application Based IDSs). In some literature I found indications to that extend, so I looked into it. Of course a program has "normal" behavior, e.g. how does it react to Event X... Y or if a user's input is wrong. Often existing binaries (e.g. *ps, netstat*, etc.) are being replaced by user input, in case of *ps* to hide specific processes. Through Application Anomaly Detection it would be possible to detect "abnormal" behavior of a program. Some application based IDSs work in this fashion, but I have little experience with them.

Finally another new method of ID-Systems: Intrusion Prevention. It is applied in some new ID-Systems, it differs from the described methods of intrusion detection. Instead of analysing *logfiles/* of the traffic, meaning: uncover attacks after they happened, they attempt to prevent attacks.

Contrary to the classical IDSs no signatures are being used to detect the attack. The following is a short explanation on how these IPSs work, their functionality should become clear with the following examples:

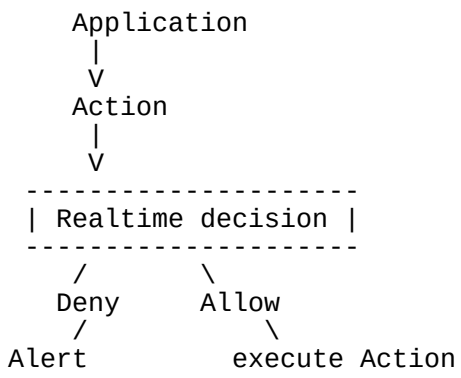
- Monitor application behavior
- Create application rules
- Alert on violations
- Correlate with other events
- System Call Interception
- .....

’Monitor application behavior’ comes close to Application-Based-IDSs, i.e. the behavior of an application is being analyzed and logged, e.g. what data it is normally requesting, what programs it is interacting with, which resources it requires. Like Anomaly Detection it tries to find out how a program normally operates, resp. what it is allowed to do.

The third issue (’Alert on violations’) shouldn’t need any explanations, it means only in case of a deviation (meaning when an attack is detected) an alert is being triggered. This may result simply in a log entry or blocked resources.

With the second step (’Create application rules’) a so called application rule set is being established based on information from the analysis in Part 1 (’Monitor application’). This rule set provides information on what an application is allowed to do (what resources it may request) and what an application is not allowed to do.

’Correlate with other events’ means information sharing of cooperating sensors, this provides better attack protection.



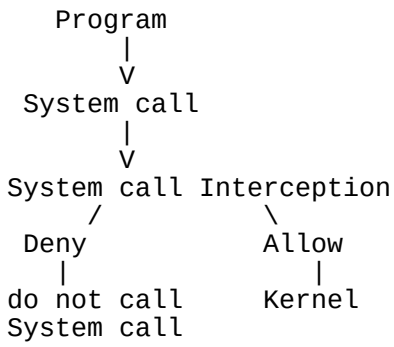
This simplified diagram shall clarify the process one more time. Prior to an activity a ’realtime decision’

is being executed (i.e. the activity is being compared to the rule set) If the activity is illegal (e.g. the program requests data or wants to change them even though it is not permitted to access system data) an alarm is set off. In most cases the other sensors (or a central console) will also be informed. This shall prevent other computer in the network from opening /executing specific files. If the activity conforms to the rule set, permission will be granted and the processing will finally proceed.

Finally to the last issue on our list: ’System call interception’. Manipulated system calls (e.g. so called rootkits) are being detected frequently. The approach to the interception of system calls is quite simple: Bevor a system call is being "accepted" it is checked out thoroughly. Checking means, for example, asking the following questions (also see [5]):

- who issued the system call (which program) ?
- under which user authorization runs the process (root...) ?
- what is the system call trying to access ?

This allows the monitoring for attempted changes of important config/system files, we "simply" have to check if the system call satisfies the pre-defined ruleset or not. .



Since Intrusion Prevention is in comparison with other methods relatively new, more information on this topic will be available.

In conclusion a hint to OKENA, a very potent IPS. In the white paper section of www.okena.com you may find additional information on Storm Watch. To find out about the shortcomings of Storm Watch read [6].

Signatures

Now we will address the use of signatures on IDSs, the second part will cover their weaknesses.

Concept

With the help of signatures known attacks can be recognized, a signature looks for a certain pattern in the data traffic. This pattern may be a variety of things like strings, conspicuous header (with unusual flag combinations), ports which are known to be used by trojans. Most of the attacks have certain characteristics, e.g. specific flags are set or particular strings in the payload. Through signatures it is attempted to discover an attack from these characteristics.

I would like to begin with the so called Payload Signatures. Here is part of a packet payload:

```
00 00 00 00 E9 FE FE FF FF E8 E9 FF FF FF E8 4F    ................
0 FE FF FF 2F 62 69 6E 2F 73  68 00 2D 63 00 FF FF .../bin/sh.-c...
```

As we will see later in the description of the SNORT rules there are some options on what to do. Frequently the content of the payload is being searched for specific strings (in Snort with ’content’ or ’content-list’). Assuming someone wants to access for example a password file (e.g. */etc/passwd*) the payload can be searched (for */etc/passwd*) if the packet contains this string counter-measures can be initiated, like:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80        \
(msg:"WEB-MISC/etc/passwd";flags: A+; content:"/etc/passwd"; \
nocase;classtype:attempted-recon; sid:1122; rev:1;)
```

Another possibility is the detection of packets which do not contain a specific string.

Another approach to protect from a possible buffer overflow is the control of of the packet size on specific ports. It is in general possible to define the source port and the destination port, requests from specific ports or to specific ports can be prohibited. String signatures in general are payload signatures. Payload signatures are checking the payload of a packet, e.g. the string signature of the payload is being searched for a specific string. .
What else could be detected from signatures? To check the payload for specific strings is not always the best. To let the signature search for the flag combination of the TCP header is another possibility. If in a package the SYN as well as the FIN bit are set this represents an anomaly which an attacker could use to find specific features of the operating system (or the operating system itself). As we mentioned in the beginning there are specific ports which are known to be preferred by trojans. Examples for such ports are 31337 or 27374.

Maybe it will be clearer if I explain the process with an example. Let us have a look at the typical telltales of an synscan attack:

- different source IPs
- TCP Src and Dest Port is 21
- Type of service 0

- IP id 39426
- SF set (SYN and FIN)
- different sequence numbers set
- different acknowledgment numbers set
- TCP Window size of 1028

In cases like this it should be the function of a signature to differentiate between "normal" and "abnormal" features of a connection. Some IDSs hold special databases with information, as shown above, they will be searched for matches.
In principal, abnormalities can be detected in the synscan example through signature check:

- Source and Destination Port are 21 (File Transfer Protocol - ftp) .
- Same source and destination port number do not inevitably indicate the imminence of an attack, only the likelyhood.
- SF set, as mentioned above, this should not happen since one should not request a connection and terminate it at the same time.
- Acknowledgment number is unequal 0, even though only SF and not ACK are set. If ACK is not set the acknowledgment number should be 0.
- IP ID is always 39426, even though this number should not remain constant (according to RFC), but this does not inevitably indicate an synscan attack - the same applies to the constant window size...

The development of the signature for the detection of a synscan attack needs to take in account more than the criteria mentioned above. Purpose of the signature should be the detection of existing as well as new versions of attacks. For this reason general and special characteristics should be combined to increase the probability of the attack detection. Even though it would be possible to write a new signature for every version of an attack, this could occupy us for ever preventing us from doing more important things. Therefore we need to pay attention to the signature detecting as many attacks (and versions of it) as possible, without the need of signature editing.

Signatures should be written to detect specific attacks, but there need to be general signatures to find abnormalities. An example of a signature for the detection of a specific attack is shown above (the attack of synscan). A more general signature, for example, could be checking for the following criteria:

- acknowledgment number unequal 0 even though ACK not set
- abnormal flag combinations in the TCP header (SYN and FIN) or others (see description of the scans)
- ....

Signatures which in general search for abnormalities in protocols are called Protocol analysis based signatures, while another group is known as "Packet Grepping".

Weaknesses

Even though the method of payload signatures (including string signatures) seams to be quite reliable there are ways to circumvent them. I will possibly write a paper on how to work around Snort rules, I will limit myself here to the essential. A signature, as shown above, searches for "*/etc/passwd*", with *nocase* capitalization is being ignored. However, if we approach */etc/passwd* not directly but indirectly, for example, if the attacker uses *' /etc/blablabla/.../.../\passwd '* would the signature still ring alarm ? No, because it searches for */etc/passwd* (and other capitilized/non-capitalized versions). Another limit of these signatures is the detection of mostly known attacks and their search for known vulnerabilities. Newer versions of specific attacks are often not discovered....Other signatures - which are specialized on specific attacks - or generalized signatures have the advantage of discovering new attacks. However, one has to be careful with the creation of the signature rules. A signature which is specialized to detect a specific attack will fail to find a slightly modified variation (instead of a constant IP ID value of 39426 the new attack has a variable value...). When mapping general signatures (protocol analysis based signatures) we have to make sure the rules are really defined globally, that means they have to be enabled to point to abnormalities which couldn't or shouldn't occur.

Another failing shows up in closer examination of the Unicode attack (see [4]). Here is a typical description of a security hole in MS IIS which was enabled by the Unicode attack:

"Synopsis:
A flaw exists in Microsoft Internet Information Server (IIS) that may allow remote users to list directory contents, view files, delete files, and execute arbitrary commands. Attackers may use the Unicode character set to craft URLs to access resources via IIS that would normally be inaccessible. All recent versions of IIS are affected by this vulnerability. Exploitation of this vulnerability is trivial. ISS X-Force is aware of widespread exploitation of this vulnerability. "

A problem for IDSs exists in the fact that characters in UTF-8 have various codes with the same result, e.g. "A" : U+0041, U+0100,

U+0102, U+0104, U+01CD, U+01DE, U+8721. Since MS IIS is case insensitive there are again many possibilities to display several characters (for example there are 83.060.640 different possibilities to display AEIOU).

If an attacker, for example, calls "http://victim/../../winnt/system32/cmd.exe" IIS would generate one error. However, by replacing "../.." with a UTF-8 equivalent no error is being generated: "http://victim/..%C1%9C../winnt/system32/cmd.exe". Furthermore, so called UN-escape UTF-8 codes may make it possible to open pages to which we shouldn't have access to. A NIDS has difficulties to detect these attacks since the event takes place on the application layer - in such a case the application of a HIDS would be more suitable. Encryption in general represents a problem for sensors - that fact is in the meantime frequently exploited. The proceedings of some "IDS-producers" clearly shows the limits of signatures, the available signatures detected some known Unicode-attacks, after small modifications of the attack the signatures became useless again. Only NetworkICE has successfully developed signatures which discover these types of attacks (Snort and ISS Real Secure developed signatures which, however, detected only the known Unicode attacks).

Responses

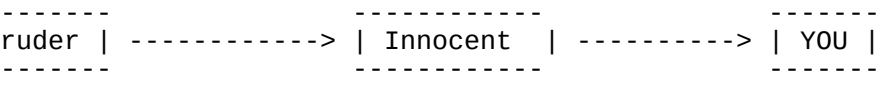
As previously explained, IDSs analyze the activities on the PC and in the net - but how useful would be an IDS if it wouldn't react and alert us? An IDS would be worthless, resp. just a waste of machine performance. 'Response' can basically differentiated in Active Response and Passive Response. We will explain the differences in the specific section.
Interested reader may be referred to OPSEC

```
Secured by Check Point' appliances are security solutions that integrate
Check Point VPN-1/FireWall-1 technology onto our partners' hardware
platforms.
```

This system permits the integration of existing systems into Fire Wall-1. An additional advantage is its worldwide recognition (it has about 300 partners). If you discover an attack you could lock out the IP address of the attacker (only as a possible response option) ...If you are interested in OPSEC read the "Deployment Platforms", there you will also find the conditions for "joining" the system (to become partner).

Active Response

Active Response means automatic reaction if IDS detects an attack (or an attempt of such). Depending on the severity of the attack most of the IDSs offers several options for the response.
1) Take action against the potential intruder. 2) "Simply" collect additional data (on the intruder and his attack, resp. its implications). 3) Change configuration
The first feasible reaction would be the initiation of steps against the intruder. This may include a variety of steps, like locking the person's access or setting off attacks against the intruder. As we explained in connection with honeypots it is often not only difficult to direct attacks against the intruder but also illegal. In this context the so-called "Third Party Effect" often shows up. What actually is this effect? Graphically explained a Third Party Effect looks like this:



The Third Pary Effect simply means an innocent person (or innocent network) was successfully attacked by the intruder, subsequently the intruder is using that network to attack our network (and possibly other networks). So, what is the problem ? The problem is our network would unmask network 'innocent' as the attacker and consequently counter-attack 'innocent' in disregard of 'intruder' invading that network to launch an attack against us. As a result of our attack (under the wrong assumption we would only attack the intruder) indeterminable damage could be caused at 'innocent'. If 'intruder' was smart enough to hide his track we will be held responsible for the damage, not the 'intruder'.

The second option (to collect additional information) is less problematic. Should a potential

attack/intrusion be detected "only" additional information about the user and his attack are being collected. If an IDS determines that a specific user has successfully expanded his privileges (or another kind of attack occurred) the observation of that user could be expanded, e.g. logging commands (if not activated), where does the user log in, how long does he stay, when did he log in, when and how often does he log in the next couple of days, does he try to transfer (FTP) specific binaries... this way a profile of the attacker is being created. With this we have the advantage of being able to analyze detailed logs and close potential loopholes, it will also enable us to take possible legal steps. As a third option I see the modification of the system, the firewall, etc. If the attacker is using specific IP addresses we can lock the user from connecting to the network through this IP. Of course we can block and log other access requests from suspicious locations. In specific cases we could simply block any access to the own network (or any requests tp specific ports, from specific network interfaces...). Another possibility of Active Response is to discontinue a TCP connection (also known as TCP kill). In order to disconnect another computer we are sending a RST (Reset Flag), this "kills" the session. Normally RST is sent when a mistake occurred in the connection..., in this case it can be utilized by an IDS (like ISS RealSecure) to end the session with another computer (for Win-NT exists a tool with the same name).

```
" tcpkill - kill TCP connections on a LAN
  .......
  tcpkill  kills specified in-progress TCP connections (use-
  ful for libnids-based applications which  require  a  full
  TCP 3-whs for TCB creation). "
```

This is an excerpt from ’man tcpkill’ ....
As you can see, there are a real comprehensive possibilities to react to attacks. To launch counterattack seems to be attractive but should not be carried out.

Passive Response

Compared to Active Response most commonly only warnings are logged, they have to be monitored by the admin/user. Here are the options to react:

1) warnings, hints...loging
2) generation of so called reports which monitor the systems for a specific time and produce an account.

Almost every IDS is capable to generate warnings or to send indications to the user/browser. If it is attempted - for example - to delete an important system file, to start specific services (whose use should be prohibited) ... a warning , announcing the incident, could be generated, also who took part in it and at what time. In the meantime more and more IDSs have the option of generating so called reports. The status of the system can be monitored over an extended time, activities can be logged and a status report can be generated. Almost all IDSs provide the option of passive response.

Filter

Filters are utilized to identify attacks by their signature. This signature is indirectly related to the previously mentioned signatures, here we are looking for typical characteristics of an attack (like

dest/src ports, *dest/src IPs*...). In another part of this section I will by means of N-code introduce and explain some examples of filters for known attacks. At the end of this article you will find a page (advanced users guide - nfr) which offers an outline on N-code - if you are not familiar with it.
land:

```
# This is an example on how to detect

# in N-code a land attack
 filter pptp ip () {
        if(ip.src == ip.dest)
        {
                record system.time,
                        eth.src, ip.src, eth.dst, ip.dest
                to land_recdr;
        }
 }
```

Since unknown variables have been used, here a short explanation :

- ip.src = the source-IP adress
- ip.dest = the destination-IP adress
- eth.src = MAC adress of the "target-machine"
- eth.dst = MAC adress of the "source-PCs"
- record system.time = logs the point in time when the condition ip.src == ip.dest was met

As you can see N-code also knows the operator == , if you read the Advanced User's Guide you will find other existing similarities (with other high level languages), e.g. N-code knows the operators + , - , *... or assembled operators like >=, != .... or as shown above ==. Xmas Scan: As you may recall from "Types of Attacks" in an Xmas Scan all flags are set. Therefore it must be plausible to verify if they are all set or not. For this we need the values of the individually set bits:

```
 Bit                    Value
 ---------------------------------------
 F-FIN                  1
 S-SYN                  2
 R-RST                  4
 P-PSH                  8
 A-ACK                  16
 U-URG                  32
---------------------------------------
```

filter xmas ip() { if(tcp.hdr) { $dabyte = byte(ip.blob,13); if(!($dabyte ^ 63)) { record system.time, ip.src,tcp.sport,ip.dest, \ tcp.dport, "UAPRSF" to xmas_recorder; return; } } }
Here, again, are some unknown variable, let me explain:

- tcp.hdr = if tcp.hdr == 0, the packet does not contain any valid TCP Header, if tcp.hdr == 1 it does
- tcp.dport = TCP destination port
- tcp.sport = TCP source port
- ip.blob =payload content of a packet (without header)
- "UAPRSF" means URG,ACK,PSH,RST,SYN and FIN are set

$dabyte is a local variable assigned to byte (ip.blob,13). To explain the "byte expression" here is a small demonstration of the TCP code bits:

```
| Src Port | Dest Port | Seq Number | ACK Number | HDR Length | Flags |\
URG | ACK | PSH | RST | SYN | FIN | Win Size | Chksum | Urg Off | Opt |
```

We realize why 13 bytes in byte() have been specified, because 13 bytes are sufficient to sustain the flags. Before we are able to understand on how byte works first some remarks to blob. If you refer to Chapter 3 of the Advanced User Guide blob is an "an arbitrarily sized sequence of bytes". One ’byte’ returns one byte from the specified offset of a blob, the common syntax looks like this: byte (str blob_to_search, int offset). The first argument specifies the blob to be searched (above: ip.blob), the second argument defines the offset (in ’blob_to_search’) of the sought-after byte. With ’if(!($dabyte ^ 63))’ it is being checked if all flags are set, this should result in 63 if the values of all flags are added (32+16+8+4+2+1), if someone likes to know: with ^ a bit by bit XOR is being executed.

Besides the options mentioned N-code offers many comprehensive possibilities. It is for example possible to find:

- if the packet is an IP packet (ip.is)
- the length of the IP packet (ip.len)
- the applied protocol of the IP packet (ICMP,TCP or UDP) (ip.protocol)
- the check sum of an ICMP packet (icmp.cksum)
- the content of the payload of the ICMP packets (in blob) (with icmp.blob)
- if the packet contains a valid ICMP header (icmp.hdr)
- if the packet is a ICMP packet (icmp.is)
- the type of ICMP packet, such as Echo Reply, Destination unreachable....
- ..... etc .....

Additional information on N-code you may find in Advanced User’s Guide Guide at:http://www.cs.columbia.edu/ids/HAUNT/doc/nfr-4.0/advanced/ advanced-htmlTOC.html

In future versions of these papers quite a few more filter will be described, check for new releases in regular intervals ;)

Standards

In this section I am going to introduce you to various "standards" as well as lists/agreements which are being shared by many tool "experts".

CVE

CVE stands for Common Vulnerabilities and Exposures, which is nothing more than a list of

vulnerabilities/exposures. At first glance that may sound funny, it may come in quite handy later on. Various tools are using different terms for detected vulnerabilities, by utilizing CVE a uniform description of various vulnerabilities/exposures, which everyone understands, can be used. Therefore it is no longer mandatory to use the same tool as other users.

CVE provides a name for a specific vulnerability/exposure and a uniform (and standardized) description, preventing misunderstandings between users of different systems. CVE is defining vulnerability as "problems that are normally regarded as vulnerabilities within the context of all reasonable security policies" and exposures as "problems that are only violations of some reasonable security policies". In CVE the differentiation between vulnerability and exposures is fundamental. Samples of a vulnerability are, for example, phf, world-writeable password files...Examples of exposures are the use of programs which can be attacked by Bruteforce or the use of services which are being attacked in general. By definition and with these samples it should be feasible to differentiate between vulnerabilities and exposures (in CVE). The fundamental difference may be the ability of the attacker to apply commands as a different user or to read/write to files even though this should not be possible (due to the file permissions). Exposures in contrast permit a user to extract additional information about the system (and its status), these activities are running in the background...Exposures arise from faulty security settings which can be "fixed". Vulnerabilities may be understood as security gaps in the "normal" security system (which should include the possibility to minimize the threat of potential attackers through permission check. However this "list" should remain current but not every vulnerability or exposure is immediately "accepted". After a vulnerability/exposure has been detected it receives a "candidate number" at first (this happens through the CNA - Candidate Numbering Authority). In addition it will be posted on the Board (by the CVE Editor) and discussed whether the vulnerability/exposure should be accepted. If the Board concludes not to accept the candidate (for the time being) the reason for this will be posted on the website. If the candidate is being accepted he will be added to the list (and become official part of CVE). By now it should be clearer that every (potential) vulnerability initialy receives a "candidate number" since it needs to be discussed if the candidate should be accepted or not. The vulnerability receives the ’candidate number’ to differentiate it from the official entries in the list. Every candidate owns 3 basic fields (they "identify" him):

- Number
- Description
- References

In this context the number is the actual name of the candidate, composed of "year of appearance" an additional number, which reveals which sequential candidate in a year it is:

```
CAN-Year - sequential candidate of the year
```

As indicated earlier an accepted candidate is added to the list. As a result the ’CAN-YEAR- Candidate number’ becomes ’CVE-Year-Candidate number’. An example: ’CAN-2001-0078’ becomes ’CVE-2001-0078’ in the list.

That’s it, for more information visit the official webpage of CVE.

Examples

In this final part some IDSs will be introduced.

Snort

Because Snort is very well known and offers many options I will describe it in more detail than the other IDS samples. Basically, Snort can be in one of three modes: Sniffer, Packet Logger and Network Intrusion Detection System. In Sniffer mode Snort generates packets on the console, in Packet Logger mode it logs them on the hard drive and the Network Intrusion Detection mode allows to analyze packets. I will concentrate mainly on the last mode, but here is a short introduction to Sniffer and Packer Logger mode:
In Sniffer mode a variety of packet information can be read out, like TCP/IP packet header:

```
[Socma]$ ./snort -v
```

As output we will get only the IP/TCP/ICMP/UDP header. There is a large number of options, only a few will be mentioned here.

```
-d = will deliver the packet data
-e = shows the Data Link Layer
```

Packet Logger Mode:
In difference to the Sniffer mode the Packet Logger mode can log the packets on the hard drive. We need only to assign a directory to which Snort should log to and it will automatically switch to Packet Logger mode:

```
#loggingdirectory must exist:
[Socma]$ ./snort -dev -l ./loggingdirectory
```

When entering "-l" Snort sometimes fetches the address of the remote computer as the directory (to which is logs), sometimes it takes the local host address. In order to log to the home network we need to specify the home network in the command line:

```
[Socma]$ ./snort -dev -l ./loggingdirectory -h 192.168.1.0./24
```

Another possibility is to log in TCP-DUMP format:

```
[Socma]$ ./snort -l ./loggingdirectory -b
```

Now the entire packet will be logged, not only specific sections, this eliminates the need to specify additional options. It is possible to use programs like tcpdump to translate the files to ASCII text, but Snort can do that too:

```
[Socma]$ ./snort -dv -r packettocheck.log
```

Network Intrusion Detection Mode: To switch to NIDS mode we may use a command like this:

```
[Socma]$ ./snort -dev -l ./log -h 192.168.1.0/24 -c snort.conf
```

In this case is snort.conf the configuration file. It is used to let Snort know where to find its "rules" to determine if there is an attack or not, if the request should be permitted...The rules, as defined in snort.conf. will then be applied to the packet to analyze it. If no specific output directory was established the default /var/log/snort is being used. The output of snort depends on the alert modus - based on this the information is somewhat sooner or later available.

Modus	How/what is displayed
-A fast	Time, Source and Destination IPs/ports, the Alert Message
-A full	Default Setting
-A unsock	Sends the Warnings to a UNIX socket
-A none	Stops the Alerts

As we have seen, with -b we can log in binary mode, with -N packet logging is completely discontinued. But this is not the limit, for example, Snort is able to send messages to syslog. Default setting for this is LOG_AUTHPRIV and LOG_ALERT. To send messages to syslog we need only to enter "-s", example follows. Furthermore, we have the possibility to send messages to the smbclient or Win-pop-up warnings to a Windows computer. To utilize this "feature" we have to enter "-enable-smbalerts" at the configuration of Snort.

```
[Socma]$ ./snort -c snort.conf -b -M MYWINWORKSTATION
```

Here a example of the application of the alert modes:

```
[Socma]$ ./snort -b -A fast -c snort.conf
```

Besides the options described there are others like the following:

```
-D = starts Snort in daemon mode
-u usersnort= starts Snort with UID 'usersnort'
-g groupsnort = starts Snort with GID 'groupsnort'
-d = also log the data of the applications layer
r
```

Snort offers many options, if you run into a problem just enter "snort -h" or look in the mailing lists if your problem has appeared somewhere else. The following section covers Snort rules, if you don't care to understand the existing rules or even to write your own you may bypass this section. As I indicated at the end of this part (about Snort) you can download the Snort Users Manual from www.snort.org, it is our real source for this.

Snort Rules:
For a better understanding of Snort it is mandatory to know the Snort Rules. Snort is sometimes using specific variables which can be defined with use of ’var’:

```
var: <name> <wert>      var

MY_NET [192.168.1.0/24,10.1.1.0/24]
alert tcp any any -> $MY_NET any (flags:S;msg: "SYN packet";)
```

There are other ways to enter the variable name:

```
$variable = defines the Meta variable
$(variable) = here the value of the variable ’variable’ is entered
$(variable:-default) = if ’variable’ is defined, its  value is entered here,
is ’variable’ not defined the value ’default’ is entered.
$(variable:?msg) = enters the value of the variable ’variable’ or
if not defined puts the message ’msg’ out.
```

If you have dealt with shell programmimg before, the following should not be alien to you:

```
[Socma]$ shelltest=we
[Socma]$ echo hello $shelltestlt
hello
[Socma]$ echo hello ${shelltest}lt
hello world
```

The application of $(variable) in Snort and ${variable} in shell is identical. There are other equivalents (or similar terms) in shell programming:

```
[Socma]$ shelltest = bash
[Socma]$ echo ${shelltest:-nobash}
bash
[Socma]$ echo ${notdefined:-nobash}
nobash
```

The application of the term ’$(variable:-default)’ differs only in the fact that the shell is using { and } instead of (and). The last term exist also in shell:

```
[Socma]$ shelltest = bash
[Socma]$ echo ${shelltest:?"then csh"}
bash
[Socma]$ echo ${notdefinedvariable:?"not defined or nil"}
not defined or nil
```

This short excursion’s purpose was to "associate knowledge", I was able to memorize the syntax of Snort faster by referring in my mind to the terms of the shell which I remembered.
Many command line options can be set in the configurations file. For this ’config’ is being used:

```
        config <directive> [: <value> ]
```

The most important ’directives’ are:
alertfile = changes the file in which alerts are stored

daemon = start process as daemon (-D)
reference_net = sts the home network (-h)
logdir = sets the logging directory (-l)
nolog = Logging gets switched of
set_gid = changes GID (-g)
chroot = chroot'ed in th especified directory (-t)
set_uid = sets UID (-U)

If, for example, you want to change alertfile in e.g. "mylogs" you proceed like this:

```
        config alertfile : mylogs
```

Back to the actual rules (here an example of a ftp.rules excerpt):

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP EXPLOIT overflow";\
flags: A+; content:"|5057 440A 2F69|";\
classtype:attempted-admin; sid:340;rev:1;)
```

Basically, Snort rules consist of two parts: the rule header and the rule options. The rule header informs about two things:

- source and destination IP addresses
- protocol
- the actions which should be initiated by the rule

In the ftp rule above the header is the following part:

```
Action       source ip         destination ip
   |             |                  |
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
        |               |                |
        Protokoll       From any port   Port
```

As you can see the rule header ends at the first (and after this the rule options begin. There are several possible actions (in this case 'alert') which can be launched should the Snort rules discover something suspicious:

- alert = depending on which alert method is being used (default is 'alert full'), an alert is triggered and the packet involved is being logged
- log = only the packet is being logged
- pass = results in the packet being ignored
- activate = generates an alert and returns to another dynamic Snort rule (more to this soon)
- dynamic = the rule remains inactive until it gets activated by another rule, after that it works like 'log' (see above)

The second field (protocol - tcp here) specifies which protocol needs to be analyzed. Possible are: tcp, udp, icmp and ip (in the future there may be others ...like ARP, GRE).
In connection with the next field (source ip) we are frequently finding the ! - operator (negations operator).

```
alert tcp !$EXTERNAL_NET any -> $HOME_NET 21
```

As a result of the negations operator each packet which does not arrive from $EXTERNAL_NET is being logged. There are additional possibilities to enter a number of IP addresses, meaning a list of IP addresses. The addresses have to be separated by a comma and enclosed by [ ].

```
alert tcp ![ip address,ip address] any -> ....
```

Another alternative is the use of "any", that includes every IP address.

```
log tcp any any -> ...
```

The last part of the rule header is the specification of the ports, in our example ftp. It is not only possible to monitor a specific port but a specific range (several ports). Here are the options:

```
:portnumber                    -> all ports smaller equal
                               portnumber
portnumber:                    -> all ports higher equal
                               portnumber
fromportnumber:toportnumber    -> all ports between fromportnumber
                               and toportnumber (and those included)
```

Of course it is possible to use the negations operator, with the consequence that all ports would be monitored except those entered, e.g. .

```
!:21            -> all ports which are not smaller equal 21
```

Something that has not been explained but has been used all along is the direction operator "->".

```
"source" -> "destination"
```

However, there is another variant <> :

```
"source" <> "destination"
```

This means Snort will search the source as well as the destination for the address.
As I mentioned, there is the step ’activate’, it generates an alert and returns to another dynamic Snort rule.
If a specific rule has completed its actions it may activate another rule. Basically the difference between normal rules and "activated rules" is the fact that only a specific field must be specified: "activates". Dynamic rules, on the other hand, work like logs (see above), only difference: "activate_by" must be entered. One more field must be entered: "count". After the "activate rule" has done its job the dynamic rule is invoked, but only for "count" packets (which means for 40 packets when count = 40).

```
activate tcp !$HOME_NET any -> $HOME_NET 143 (flags : PA;  \
  content : "|E8C0FFFFFF|\bin|;activates : 1; msg : "IMAP buf!";)
dynamic tcp !$HOME_NET any -> $HOME_NET 143 (activated_by : 1; \
  count : 50;)
```

Some of the options, like the rule options, have not been introduced yet, I will explain them now, but they will make more sense to you later. Please note the fields *activates* and *activated_by* (dynamic rule) in our example above. The first rule invokes the dynamic rule after the first rule has completed its job, this is also indicated by the statement activated_by = 1 of the dynamic rule.

Now to the second part of the Snort rules: the rule options. Let's use again the first ftp.rule:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP EXPLOIT
overflow"; flags: A+;\
content:"|5057 440A 2F69|";classtype:attempted-admin;\
sid:340; rev:1;)
```

The rule option in this case (the rule header stops at the first ")):

```
 (msg:"FTP EXPLOIT overflow";\
flags: A+; content:"|5057 440A 2F69|";\
classtype:attempted-admin; sid:340; rev:1;)
```

There are 34 keywords, I will limit my explanations to the most important and/or most used. For those who like to get an overview of all possible keywords please have a look in the Snort Users Manual.

msg - displays the alert messages and logs them in the Packet Logger Mode
logto - logs the packets in a specific file
dsize - compares the packet size with a different value v
flags - checks the TCP flags for specific values
content - searches for a specific pattern/string in a packet
content-list - searches for a specific pattern/string in a packet
nocase - upper - and lower case in searched strings are neglected
react - active response (blocks websites)
sid - Snort rule id
classtype - sorts the potential attacks in groups
priority - sets the sensitivity
So far, so good, but how do the individual rules work ? msg:
We are finding 'msg' quite often when we browse the rules, this option is responsible for the generation of alerts and their logging.

```
   msg:"<text>";
```

t "<text>" is the message which is written/displayed to alertfile
logto:
Each packet, for which the rule is applicable is being logged in a specific file.

```
logto: "<filename>";
```

In this case "<filename>" is the file to which the applicable files will be logged.
dsize:
This is used to specify the size of a packet. If we know the size of the buffer of a specific service this option may be used to defend against a possible buffer overflow. Compared to ’content’ it is a bit faster, therefore it is being utilized more frequently to test buffer overflows.

```
dsize: [>|<] <size>;
```

The two optional operators > and < indicate that the packet size should be larger resp. smaller than the specified value flags:
This checks which flags are set. At present 9 flags are available in Snort:

```
F      FIN
S      SYN
R      RST
P      PSH
A      ACK
U      URG
2      Bit 2 assigned
1      Bit 1 assigned
0      no TCP flags set
```

There are additional logical operators to specify criteria for the testing of flags

```
+ ALL flag     = Hit at all specified flags (others as well).
* ANY flag     = Hit at all specified flags.
! NOT flag     = if the specified flags are not set.
```

In general the keyword ’flags’ is used this way:

```
flags: <Flag valuet>;
```

The reserved bits may be utilized to detect unusual behaviour, e.g. attempts of IP stack fingerprinting.
content:
One of the most used keywords (besides ’msg’) is ’content’. It can be used to search the payload of packets for particular content. If the specified content is detected predefined steps are launched against the user. After the detection of the content in the payload of a packet the remainder of the Snort rules will be executed. Without applying ’nocase’ (see below) capitalization will be considered. The content of the payload shall be searched for binaries as well as text. Binary data are enclosed in | | and shown as byte code. The byte code shows binary information in form of hexadecimal numbers. In context with this keyword the negations operator (!) may be applied, for example we can issue an alert if a packet contains a specific text.

```
content: [!] "<content>";
```

The statement of ! is not mandatory

```
 alert tcp any any -> 192.168.1.0/24 143 \
(content: "|90C8 C0FF FFFF|/bin/sh";\
msg: "IMAP buffer overflow";)
```

As demonstrated in this rule the binary data are enclosed in | |, following this, normal text can be applied. Have a look at the description of ’offset’ and ’depth’ in the Snort User Manual., they are frequently used in context with ’context’.

content-list:

This keyword works similar to ’content’ with the difference that a number of strings, to be used for searching packets, can be entered. We enter the suitable hexa numbers, strings, etc. in a file. This file, containing the words to be searched for, will be specified in the application of ’content list’. We have to keep in mind to write the strings vertically spaced (each string in one line), e.g.

```
"kinderporno"
"warez"
.....
```

Following this, we can - by applying ’content-list: [!] "<filename>" ’ search this file. Of course ! is optional, it has the same effect as in ’content’. .

nocase:
This rule plays an important role in context with ’content’ keywords. Normally capitalization is adhered to, by using ’nocase’ case sensitivity is being ignored:

```
 alert tcp any any -> 192.168.1.0/24 21 (content: "USER root";\
  nocase; msg: "FTP root user access attempt";)
```

Without the use of ’nocase’ only for ’USER root’ would be searched, since ’nocase’ was specified the case sensitivity does not apply.
If searching a packet for a specific content (applying ’content’ or ’content_list’) results in a hit, ’react’ can be used to respond to it. As a general response specific pages requested by the user will be blocked (porno pages...). By applying Flex Resp connections may be discontinued or warnings may be sent to the browser. The following options are possible/legal:

block - disconnects and sends a notification.
warn - sends a visible warning (soon available)
These ’basic arguments’ may be complemented by additional arguments (so called ’additional modifiers):

msg - the text to be sent by invoking the keyword ’msg’ is included in the notification to be sent to the user
proxy : <portnummer> - the proxy is being used to send the notification (soon available)

```
react : <basic argument [, additional modifier ]>;
```

The ’react’ keyword is added at the end of the rule options, it can be used like this:

```
 alert tcp any any <> 192.168.1.0/24 80 (content-list: "adults"; \
   msg: "This page is not for children !"; react: block, msg;)
```

sid:
’sid’ or Snort rules IDentification is used to identify "special" Snort rules. This makes it legal/possible for "output plugins" to identify every rule. There are a number of sid groups:

```
 < 100 = reserved for future use
 100-1 000 000 = rules which come with Snort
 > 1 000 000 = being used for local rules

 sid-msg.map contains a mapping of msg tags for sid’s.
It is being used foe post processing to assign a warning to an id.
 sid:  <snort rules id>;

 alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80  \
  (msg: "WEB-IIS file permission canonicalization"; uricontent:\
  "/scripts"..%c1%9c../"; flags: A+;nocase;sid: 983;rev:1;)
```

classtype:
By using ’classtype’ we can sort attacks into various groups. In the rules we can determin the priority of a potential attack resp. in which group it belongs. Rules which are categorized in the configuration file will be assigned a standard priority automatically.

```
 classtype: <class name>;
```

Categories of rules are to be defined in classification.config. The following syntax is being used:

```
 config classification: <class name>,<class description>,\
       <standard priority>
```

In the following paragraph - the description of ’priority’ - we will learn what kind of groups of attacks there are.
priority:
This keyword is being used to assign "security priorities" to our rules, meaning how damaging a potential attack could be. The higher the priority the more damaging the potential security risk. In context with the earlier explained ’class types’ the priorities are easier to understand:

```
Class type                    Description                Priority
-------------------------------------------------------------------
 not-suspicious                Any "unsuspicious" traffic    0
 unknown                       Unknown traffic                  1
 bad-unknown                  Potential "bad" traffic      2
 attempted-recon              Attempted Information Leak           3
 successful-recon-limited     Information Leak                     4
 successful-recon-largescale  Large Scale Information Leak          5
 attempted-dos                Attempted DoS-attack                6
 successful-dos               Successful DoS-attack              7
 attempted-user              Attempted to get user privileges 8
 unsuccessful-user            Unsuccessful attempt to get
                                        user privileges
```

```
successful-user             Successful attempt to get
                                    user privileges
attempted-admin              Attempt to get admin privileges     10
successful-admin            Successful attempt to get admin privileges
```

As mentioned earlier, higher priorities represent bigger security risks. A user receiving admin privileges would be the most serious attack.

```
alert tcp any any -> any 80 (msg: "WEB-MISC phf Versuch";\
       flags: A+;content: "/cgi-bin/bash";priority:10;)
```

The topic of ’rules’ is admittedly quite complex but not that difficult. Study a few rules, research in the manual what they are for and after a while the "rule monster" will make sense ;) Resources for Snort and documentation can be found at http://www.snort.org. You will find there some valuable .pdf’s, e.g. the Snort User Manual, which is the main source for this description.

LIDS

Since Stealth’s paper (and sources) and the LIDS-Hacking-HOWTO we know that LIDS does not really improve the security of the computer, but it is in some cases more of a root kit ;)

Let’s look at the concept and then at the (imaginary) strength and the weaknesses of LIDS. It was originally developed to protect - for example - important system files and to hide specific processes from the user. In addition, it should not be allowed to simply bind modules, the necessary modules will be bound with the start of the system. Earlier I mentioned the LIDS-Hacking-HOWTO and Stealth’s paper, both explaining the workings and the weaknesses of LIDS, I will limit myself to the most important features. You will find a link to both texts at the end of this section.

The main job of LIDS is the protection of the file system. To be able to protect important files/directories they are being sortet into groups:

- Read Only = This file/directory has only read access, changes are not permitted
- Append Only = It is only permitted to "attach" content to the file
- Exception = These files are not protected
- Protect (un)mounting = permits or denies someone to (un)mount a file system

For "real" protection some system calls are being manipulated to make sure the protections are maintained (e.g. sys_open(), sys_mknod(), ...)
Furthermore, LIDS prevents specific processes from being killed (or become visible). Purpose of the procedure is to prevent the attacker from seeing specific processes which could be monitoring him. A ’ps-ax’ call should also not reveal our processes. To truly hide the process, it is being marked as ’PF_HIDDEN’. In case ps is working to generate information about the processes it will be prevented from doing so on processes which are marked ’PF_HIDDEN’. This in itself is not sufficient to safely hide a process because temporarily it has still an entry in the proc file system (/proc), consequently LIDS is also manipulating that function in order to prevent the process to show up in the /proc directory. Besides this, there is the possibility to restrict the privileges of a process with capatibilities. If, for

example, CAP_CROOT is set to 0 the process is being prevented from using chroot (see /usr/src/linux/include/linux/capatibilites.h).
In addition LIDS has the possibility to run in one of two security options: ’security’ or ’none_security’. To define the difference between ’security’ and ’none_security’ the global variable ’lids_load’ is being utilized. Default value is ’1’, this means LIDS runs in ’security’ mode - meaning the restrictions are enacted. If ’security=0’ is set at the start (LILO prompt) ’none_security’ is being activated. As a result all security checks, restrictions... are deactivated. With ’lids_load=0’ the computer operates like LIDS would not be installed. An additional possibility to change the security option is the application of ’lidsadm-S’ online, for this a password needs to be specified.

LIDS does also offer the possibility to protect firewall rules by activating CONFIG_LIDS_ALLOW_CHANGE_ROUTES and switching off CAP_NET_ADMIN. If someone wants to modify the firewll rules CAP_NET_ADMIN needs to be activated to prevent that anybody can change the rules. In addition to this Sniffer can be deactivated and a port scanner can be integrated into the kernel. .

Lids also offers several "response options" (see section on Response) for example to send send notifications with the pager, per SMS, to the administrator.

In general, there are many options, Stealth explains in his paper how to abuse LIDS: http://www.securitybugware.org/Linux/4997.html. The LIDS Howto can be found at: http://www.lids.org/lids-howto/lids-hacking-howto.html.

COLOID

COLOID is the acronym for Collection of LKMs for Intrusion Detection, it was founded by myself a while ago.
Since it came to be known that parts of the project were not working well and up to date the project was temporarily stopped. However, I would like to elaborate a bit on the originally planned features: With the first module (prev_exec) we wanted - amongst others - to prevent temporarily the execution of specific binaries (in my source I used the GNU complier GCC) at specific times. It is defined in the source when an execution should be prohibited - the time can be specified to the minute. If a user wants to execute GCC at that time it will be blocked, GCC is not going to executed. Should a user execute GCC at a "permitted" time, the arguments will be checked and a search for .c files takes place. Purpose of this procedure is the search of the source (someone wants to compile) for "dangerous functions". Defaults were ’scanf’ and ’strcpy’ ....etc...it is possible to add more functions. As I noted earlier LKM searches the source, should it detect one of the functions, the execution of GCC is prohibited. In addition a log file is written and a ’beep’ generated.

So far the module worked but the concept was not enough general and could easily be circumvented.

The second module was ’anom_detection’ which used the Anomly Detection mentioned earlier. Actually two LKMs are belonging to this section:

1) Anomaly_Detection.c which generates the database of normal user activities
and
2) Misuse_Detect.c which checks if the behaviour of the user (by using the database as a benchmark) deviates from the normal (logged in the database) behavior.

Plan was to let LKM check the following criteria:

How often did the user execute the following commands:

- su
- login
- chmod
- chown
- insmod
- ps
- lsmod
- rm
- last
- lastlog
- ftp

When has the user executed the following commands:

- login
- su

Or when has the user normally started the PC and when did he initiate a shutdown
Other:
How often did he (try to) open the following files:

- /etc/passwd
- /etc/group
- /etc/shadow
- /etc/ftpusers
- /etc/ftpgroups
- /etc/ftpaccess
- /etc/hosts.allow
- /etc/hosts.deny
- /etc/inetd.conf
- .....

How often did he execute programs with the SUID bit set ?
We have to pay attention to what files to specify for monitoring (how often did he open them). If we

choose too many files the PC performance will take a hit, reasonable decent work will not be possible.

There existed other smaller LKMs without a complete source, the source of the two module mention above is available from my web page, possibly someone may do something with it.... .

Closing Words

Should anyone have more ideas to the content of this paper, please drop me a note at : Socma(Q)gmx.net . For additional stimulation, praise, critique ....please sent me an email
References (besides those mentioned in the text):

1. http://online.securityfocus.com/infocus/1524
2. http://online.securityfocus.com/infocus/1534
3. http://online.securityfocus.com/infocus/1544
4. http://online.securityfocus.com/infocus/1232
5. http://www.entercept.com/products/entercept/whitepapers/downloads/systemcall.pdf
6. http://www.computec.ch/dokumente/intrusion_detection/
   angriffsmoeglichkeiten_auf_okenas_stormwatch/
   angriffsmoeglichkeiten_auf_okenas_stormwatch.doc

2005-01-14, generated by lfparser_pdf version 2.51