Malicious Traffic Detection in Local Networks with Snort

Loïc Etienne / EPFL - SSC

Abstract

Snort is an open source Network Intrusion Detection System combining the benefits of signature, protocol and anomaly based inspection and is considered to be the most widely deployed IDS/IPS technology worldwide. However, Snort's deployment in a large corporate network poses different problems in terms of performance or rule selection. This paper proposes different improvements to the Snort Security Platform: the use of another library is proposed to significantly improve the amount of traffic that can be analyzed, and Snort's multithreading possibilities are explored. A new rule classification has been devised, and rulesets suited to large corporate networks are proposed. The use of Oinkmaster has been tested and documented to seamlessly update Snort's rules.

1 Introduction

This paper will explore how Snort, an open source Network Intrusion Detection System, can be used to secure and monitor such a very large corporate network.

This thesis will start by a brief tour of horizon of network security in Chapters 3 and 4; Chapters 5 and 6 will present Snort's functionalities and rule system. It will show that Snort's default classification is inadequate for large network rule selection.

Chapter 7 will present findings concerning commonly available rules, starting with rules related to CERN policies. A solution to improve Emule detection when Skype is present is proposed. Rules that significantly improve the detection of infected and compromized devices at CERN are also proposed.

Chapter 8 will introduce a tool to handle rule updates, and Chapter 9 will suggest possible performance improvements to the Snort platform.

2 About CERN

2.1 CERN

CERN, the European Organization for Nuclear Research, is the largest particle physics laboratory in the world.

Commonly referred to as the birthplace of the worldwide web, it currently hosts scientists from some 580 institutes and counts 20 European member states.

CERN's missions are Research, Technology, Collaboration, and Education [1].

2.2 CERN Public Network

The CERN Public Network offers a great playground for any IDS system.

CERN hosts more than 10'000 visitors each year, most of whom bring their own, unmanaged and/or unpatched, personal computers. CERN hosts many conferences, and many students from universities all around the world.

With scientists representing more than 80 countries, network and web traffic span across the whole Internet.

The CERN Public Network is liberal, and there are few restrictions on the network usage. The network is heterogenous, and contains mainly Windows (XP, Vista, 2000, Seven), Linux (Scientific Linux CERN, Ubuntu, Debian, Redhat), and Mac OS (9, 10.4, 10.5) computers, but also hosts more exotic devices such as tablet PCs, mobile phones, and other various devices. As all the devices are registered, the CERN public network provides an easy way to interact and cross-check data with the corresponding users.

Table 1 shows a typical distribution of network traffic on a sample of 1.5 million packets. This table provides interesting results, because they significantly differ from what could be found in a typical enterprise, where most people have never heard the term of SSH for example.

#Protocol	% of traffic
TCP	93 %
UDP	6 %
Other	1 %
SSH	26 %
HTTP	12 %
SSL	2 %
X11	1 %
SMTP	<1 %
Other	59 %

Tab. 1: Protocol breakdown of traffic sample

The IDS typically sees 600Mbits/s on average, and more than 1Gbit/s during peak hours.

With such a broad variety of devices, origins, and amount of traffic, this network offers a perfect framework for IDS tuning. Clearly, naive approaches such as blocking all non-http traffic would be unacceptable in this context.

3 Network Traffic Analysis as Part of an IDS System

3.1 IDS

An Intrusion Detection System (or IDS) is composed of software and/or hardware designed to detect unwanted attempts of accessing, manipulating, and/or disabling of computer systems. An IDS is used to detect several types of malicious behaviors that can compromize the security and trust of a computers system. These threats are various, and include network attacks against vulnerable services, data driven attacks on applications, host based attacks such as privilege escalation, unauthorized accesses, or malware (viruses, worms) [2].

Terminology

False positive: A false positive is defined by an incorrect result of a test which erroneously detects something when in fact it is not present. In an IDS, it typically consists in detecting a network threat which is in fact non-existent.

Intrusion: Any set of actions that compromise the integrity, confidentiality or availability of a resource.

Attack: An attempt to bypass security controls on a computer. May precede an intrusion.

Signature: Network traffic is examined for preconfigured and predetermined patterns known as signatures. Many attacks or threats today have distinct signatures. Good security practice requires a database of known signatures to be constantly updated to mitigate emerging threats.

Alert: Event generated when a signature matches traffic activity.

Noise: Amount of unneeded, unwanted, or false-positive alerts; masking or reducing the number of real alerts.

Structure of an IDS An IDS is typically composed of three main parts: a sensor, an engine, and a console.

The sensor's main task is to analyze all the data according to some configuration data, and create events accordingly. The engine records events logged by the sensors in a database, and uses a system of rules to generate alerts from security events received. The console monitors events and alerts, and allows to interact with the latter.

In many cases, the three components are combined in a simple device or appliance. A more detailed explanation of the components can be found in [3].

Different Types of IDS There are two main types of IDS working at different points in the infrastructure:

Network IDS (NIDS): The NIDS scans all network traffic that is fed to it. The NIDS typically functions in the same way as an antivirus software: every single packet is scanned for patterns which may indicate a problem.

Host-based IDS (HIDS): HIDS are typically installed on every host. HIDS are more platform specific, and are focused on the target computer. HIDS can capture all the traffic generated by a host, which a NIDS typically cannot do in a switched network. HIDS are not necessarily traffic based, but also look at the system's state.

4 State of the Art - From Raw Packet Capture to Advanced Detection Mechanisms

Network Traffic Analysis can be performed in many different ways. Here is a list of the features that characterizes network traffic. Each of these features is part of the OSI model [4].

- Source & Destination IPs: Provide the source and destination addresses of every packet.
- Protocol: The transport protocol. Typically TCP or UDP.
- Source & Destination Ports: Complete the source and destination addresses.
- Size: The size of the packets.
- Flags: Whether the packet has some flag bits set. These could be: urgent, SYN, ACK, FIN, ...
- Payload: The data itself, that will be delivered to the application running on destination address and port.

Each of these features can provide valuable information for a NIDS. Today, many corporate switches can export raw data, NetFlow, sFlow or similar data. NetFlow data contains Source and Destination IP and port, and the amount of traffic transferred per flow.

On a higher level, it is also possible to analyze the payload of every packet. However this requires a full understanding of the protocols by the analyzer, as well as a full access to the traffic, which is not easily scalable.

NetFlow is an embedded instrumentation within Cisco IOS Software to characterize network operation. It gives the administrators the tools to understand who, what, when, where, and how network traffic is flowing.

Data is collected directly by switches, and can be exported to a reporting server. A flow is a n-tuple[1], usually identified by the source and destination IPs and ports, the protocol, and the unilateral number of bytes transferred.

This data can then be aggregated and analyzed to detect unwanted or malicious behavior. One could, for example, count the number of SMTP servers (destination port 25), the number of peers, or the number of SSH servers contacted by any host to detect misbehaving hosts (sending spam mail, using P2P, or doing ssh scans).

A worm detection system based on netflow data is proposed in [5].

High Level Analysis With this technique, the payload of every single packet is inspected. This requires much more resources than NetFlow based inspection techniques, as all packets have to be opened up to the 7th layer of the OSI model to be analyzed. However, this is obviously much more useful, as it can detect protocols running on any ports, as well as any byte pattern regardless of the underlying protocol.

Snort is an open source NIDS software [6]. Combining the benefits of signature, protocol and anomaly based inspection Snort is the most widely deployed IDS/IPS technology worldwide. It is able to perform "high level analysis" on all the traffic flowing through its sensor.

Snort is available in two different version: Snort itself, and the Snort Security Platform (Snort SP). Snort SP is an extension of Snort, using the same engine, but allowing much more options, such as multithreading, an

[1] n depending on the version

interactive shell, and performance improvements. Snort SP is still in early beta phase, and is very likely to improve its performance before the final release.

This paper will focus on the SnortSP-3.0.0b2, as there were stability problems with the latest beta (SnortSP-3.0.0b3).

Figure 1 presents the software architecture. SnortSP is designed to act as an "operating system" for packet-based network security applications, providing common functionality that all programs need.

From a developers point of view, SnortSP is what gathers data and handles any evasive techniques or other conditions that occur in suspicious and malicious traffic. SnortSP normalizes the data and then provides this cleaned up high level data to the engines for inspection.

Snort SP includes a new command line interface backed by the LUA embeddable programming language. This language allows to extend Snort functionalities with a new scripting language [7].

The engines are analysis modules that plug into Snort SP. Multiple engines can run simultaneously on the same traffic, in the same Snort SP instance.

The great advantage of this platform, is that it gives the opportunity to run multiple analyzers in parallel, thus increasing significantly the amount of traffic that can be analyzed. This will be presented in Chapter 9.1.

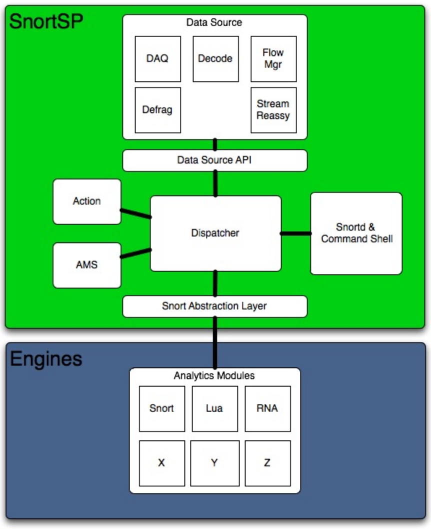

Fig. 1: The architecture of the Snort SP Platform [8]

Snort SP is shipped with the "Standard" Snort as engine. Snort engine is configured by giving it "rules". Each rule is a set of "what to look for" and "what to do when it is found". There can be hundreds of rules run in parallel in each Snort analyzer.

Snort is able to analyze traffic up to the seventh layer of the OSI model, by extracting and rebuilding application sessions of known protocols. It uses the libpcap library to locally access data on any network Interface.

Machine Learning could be seen as the future of IDS. It takes another approach towards traffic analysis. By trying to learn the expected traffic patterns, it generates an alert if some traffic is classified as unexpected. This is still an area of ongoing research, and real-world solution based on this technology are only starting to emerge [9].

One of the drawbacks of this solution is that every unexpected traffic or unexpected exchange of information will generate an alert. This solution could be efficient for small, well-defined networks, but are likely to do more harm than good in a large heterogenous network.

An excellent tour of horizon in this field can be found in [10]. Recent work attempts to bound the number of false alarms while optimizing the correct results [11].

5 Snort in a Large Corporate Network

5.1 Deployment

Sensor location is important. Typically, a good entry point is at the border between the LAN and the Internet. Placing Snort at this strategic point, allows the analysis of all traffic coming in and out of the local network. For this study, the Snort sensor was placed between the CERN Public Network and the Internet.

This is a compromize, because it does not allow to scan inside-to-inside traffic. Such an analysis could be performed for example using sflow (which is a statistical Netflow), or dumping all unauthorized traffic at the switch level.

It is also important to define what to detect with Snort. As Snort rules are able to detect anything in the traffic, it is important to clearly define the needs.

Is it enough to detect compromized hosts? Are there policies that need to be enforced? Is it useful to record all incoming attacks towards the network?

Those are all questions that need to be answered before deploying Snort rules.

5.2 Snort Rules

5.2.1 Sources

Snort being able to deploy any kind of rule, Snort rules are not included with the software. However, there are different sources for finding and deploying rules:

Vulnerability Research Team (VRT) These are the "official" Snort rules. They are provided by sourcefire and are updated on a weekly basis by the Sourcefire VRT.

Emerging Threats (ET) Emerging threats rules are an open source community based project. This set is the fastest moving and most diverse Snort set of rules. The rules are updated several times per day.

Community rules These rules are created by the Snort community. There are very few rules, and the last release is from 2007 for Snort 2.4. Most of the threats they detect are already implemented in ET or VRT.

Homemade rules and others These are the rules, created and maintained locally, according to the specific needs of the network. There also may be other rules out there. For specific and other "unique" threats, search engines may provide more specific rules, but it is needed to know what to look for. Recently, the Internet Storm Center (ISC) [12] started publishing rules when new 0-day exploits[2] emerged.

[2] exploitation of unpatched software vulnerabilities

5.3 Existing Classification Schemes

In March 2009, VRT and ET rules combined counted more than 22'500 unique signatures. Several attempts have been proposed by their editors to classify them. Snort currently proposes the following classification schemes:

5.3.1 Splitting in Files

Signatures are split in different files. File name range from a specific protocol (ie. *smtp.rules*) to whole meta-classes of rules (ie. *policy.rules*). Even if this classification is useful in some cases, most of the time it only gives a hint of what is detected by the contained rules. A good example of this classification are the *p2p.rules* files, which only contain rules detecting the use of P2P software on the network, and can pretty much be deployed untouched when the use of P2P software needs to be detected.

However, most of the time, this classification lacks details and formalization. There are for example 5'814 signatures in the *netbios.rules* file. These signatures are not classified, and range from alerting when a network share is accessed (which can be normal behavior), to successful Denial of Service attacks (which may indicate that a host has been compromized).

This classification method is not enough to successfully find a set of rules worth deploying among the 22'500 available rules.

5.3.2 Classtype

To help further with this classification, Snort developers also propose a "classtype" parameter for each of the rules. This is a good idea, but there are many rules that are missclassified.

Table 2 shows this classification for the *netbios.rules* example.

#	Classification
3157	protocol-command-decode
2631	attempted-admin
15	attempted-dos
7	attempted-recon
2	unsuccessful-user
1	bad-unknown
1	attempted-user

Tab. 2: Classtypes for the netbios.rules file

Most of the messages are cryptic (is a "string detect" a problem, or how bad is a "successful-recon-limited" for example), and Snort developers provided a very short description of each of the classtypes. This classification is presented as Appendix A.1.

It was impossible to generate an ideal ruleset using this classification. Even with the help of Appendix A.1, it is still difficult to make a match between the requirements (Section 5.1) and all the available rules.

Therefore another classification is needed.

5.4 Ideal Situation

In a perfect world, the network administrator should be able to choose what to enable according to his needs. To do that, the network administrator needs to perfectly know the environment.

Such knowledge include: What services are running on which computers, the operating system running on each computer, and the expected amount of traffic for each host and towards which destination.

This knowledge allows for a better tweaking of the NIDS, where signatures can be enabled only for the hosts where they are needed, and therefore significantly reduces the noise.

If such knowledge is attainable in a small company running only a few homogenous hosts, it is clearly not the case in large networks such as the CERN Public Network, where users come and go all year long with their own random hosts.

5.5 Performance Problem

Another fact that should be taken into account is that Snort will only be able to handle a limited amount of traffic, depending on the number and kind of rules deployed.

Therefore there needs to be a tradeoff between the number and kind of rules deployed, and the amount of traffic that is analyzed.

So the main question is "How to choose and optimize Snort's rules?". An attempted answer is provided in Chapters 6 and 7.

6 Snort Rules

6.1 Introduction

Chapter 5 quickly presented the problem of dealing with Snort rules. This chapter will try to present the different caracteristics of the rules that should be evaluated, and propose a new classification that corresponds to the CERN needs.

6.2 Definition

A Snort rule can be defined by many parameters. A rule is composed of two distinct parts: the rule header, and the rule options.

The rule header contains the rules action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information. The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken. Here is a sample rule:

```
alert tcp any any -> 10.0.0.0/24 80 \
(content:"|00 00 00 00|"; depth: 8; \
msg:"bad bytes"; sid:1234)
```

This rule will trigger an alert if four null bytes are found on the first eight bytes of all traffic sent to port 80 to the 10.0.0.0/24 network. The rule unique ID is 1234, and the alert message is "bad bytes". Rules are powerful, and there are many possibilities: It is possible to look for bytes at specific position, within range of other bytes, or to count the number of occurences of a match before alerting. It is also possible to use Perl Compatible Regular Expressions (PCREs) on the data, and to limit the search to specific bytes. All these options are presented in detail in [13].

For a rule to trigger an alert, all the elements contained in the rule options need to be true. These elements are checked sequentially. If the first one is false, then the others will not be checked. Therefore the order of the arguments is very important to optimize rules.

6.3 Metrics

Metrics that should be evaluated for each set of rules include the following.

6.3.1 Threat level

In this paper, the threats are split in three categories:

Category 1: "Compromized" These are the most important incidents. They include compromized hosts, hosts infected by viruses or malwares, or users performing illegal actions. Each incident should be detected and acted upon.

Category 2: "Policy Violations" When a user does not comply to the policies, an alert will be triggered by this set of rules. Typical examples are Peer-to-Peer (P2P) and Internet Relay Chat (IRC) rules.

Category 3: "Targeted Attacks, Scans, and others" Potential attacks fall into this category, even if unsuccessful. They do not mean that a host has been compromized. Incoming viruses and other incoming malwares will be classified here. They provide some information on the network activity, but do not necessarily require any action.

The direction of these alerts is important, because outgoing scans and attacks could indicate that a local host has been compromized, whereas incoming scans and attacks only indicate a current event for which not much can be done.

6.3.2 Resource Consumption

Each rule (or set of rules) should be analyzed for resource consumption. Performance is a critical factor with such a high network load. This factor will probably be difficult to evaluate given the type/amount of traffic. Example of solutions are:

- Compare CPU load;
- Compare percentage of traffic analyzed;
- Usage of Snort "Rule Profiling" (Chapter 9.3).

For such an evaluation to give conclusive results, rules should be evaluated with similar amount/quality of traffic.

6.3.3 Complexity

For each rule (or set of rules), the benefits should be evaluated. If the ratio of false positives for a rule is too high, then it may not be that useful. In-depth analysis of the rule and some interaction with the end-users are needed to properly evaluate this.

Rule complexity is mostly based on the number of bytes checked in the traffic; the more specific the better. Rules checking very few bytes are expected to generate a lot of false positives with a high amount of traffic. However it also depends on the bytes themselves: Checking for a long and common string will trigger more false positives than checking for a few, unusual bytes.

6.3.4 Dependencies

A clear understanding of the different protocols may help reduce the number of rules. For example it may be useless to catch the request and the reply, when a reply always comes from a request.

A tool is proposed in Appendix E.2 to help identify these dependencies by comparing the sets of IPs triggered by each rule.

There are often many rules related to a specific protocol / event, and it is often enough to detect the initial connection message instead of capturing all messages exchanges.

6.3.5 Policies & Recommandations

Each set of rules should be compared to the company policies and recommandations to evaluate their benefits. Why bother detecting normal and allowed traffic?

6.4 Classification

Using Snort's proposed classification (Chapter 5.3), and after an extended work on the rule sets (sampling them, analyzing them, deploying them and analyzing the results) the following classification scheme is proposed [14].

6.4.1 Compromized

This category contains all signatures that detect a successful exploit, or that indicate that a host has been compromized.

The following rule sets contain rules that fall into this category: attack-responses.rules, backdoor.rules, ddos.rules, emerging-attack_response.rules, emerging-virus.rules, virus.rules.

This proposed category only detects hosts compromized or running malware that could potentially lead a remote attacker to take control of it by opening a backdoor or stealing passwords. Adwares and other badwares are not included, and were put into the third category.

6.4.2 Policy

This category contains all signatures that help detecting P2P and IRC, which are disallowed at CERN.

The following rule sets contain rules that fall into this category: p2p.rules, emerging-p2p.rules, and local.rules.

The last one, *local.rules*, contains additional homemade rules to detect IRC usage.

The P2P sets contain signatures to detect all kind of traffic, and there are some rules that need to be disabled before this set gives usable results.

6.4.3 Attacks and others

Other source files fall into this large category. There are other policies such as Instant Messenging (IM), information about incoming attacks towards CERN, or sets to detect hosts running adware and other badware.

This last set was initially part of the "compromized" category, but due to the very large amount of devices running adware (during one day, the sensor detected 35 IPs running "Fun Web Products", and about the same number of IPs running a dozen of other adwares. Given that adware do not pose a threat per se, the related rules were disabled.

7 Snort Rules Evaluation

After the initial classification of the files, all sets of rules were thoroughly evaluated. Each of them was deployed and analyzed according to the metrics defined in Chapter 6.3. The CERN Public Network is ideal for such an analysis, because it provides means to interact and cross check with the users, and offers a great variety of devices and network traffic. With the large number of users bringing their own unmanaged laptops, there are a lot of infected devices helping to tweak the IDS.

7.1 CERN Policies

7.1.1 Peer-to-Peer

Introduction This paragraph will present the findings concerning P2P detection using Snort rules.

The first big surprise here was the amount of different protocols seen in the traffic. It seems that even if Bittorrent is the dominant P2P network, many users still rely on old and less wide-spread protocols. Many foreign users were detected using localized P2P software, with names unknown to most Europeans.

Snort's efficiency in detecting these protocols varies from case to case. Some P2P protocols are very easily recognizable, while others trigger too many false positives to provide useful data regarding P2P usage at CERN.

The rules ET and VRT both provide a file called *p2p.rules* containing all kind of rules detecting P2P traffic. Table 3 summarizes the content of these two files. All of these protocols were seen at CERN in a one-month time period. The "Others" rules detect 13 other file-sharing protocols, out of which six were seen at CERN over that same period. Over a year, this number would probably increase significantly.

Protocol	# rules	% of rules
Emule	26	29
Bittorrent	13	15
Napster	9	10
Gnutella	5	5
KaZaA	5	5
Skype	5	5
Others	28	31

Tab. 3: P2P Rules summary

Skype is without doubts the most popular VoIP application currently used on the Internet. It uses its own proprietary protocols, and all traffic is encrypted. Skype's understanding and detection has been the subject of many research papers in the last few years [15] [16] [17].

[18] concluded that Skype was made by clever people, that they made a good use of cryptography, and that Skype is incompatible with traffic monitoring and IDS systems. This fact has been confirmed in this paper.

At CERN, Skype is used daily by more than 1'000 hosts and users are required to run it on a specific port in order to avoid being affected by corresponding IDS alerts.

Being encrypted and having its first bytes serving as sequence number [18], Skype traffic is likely, after enough time, to generate alerts on all Snort rules based on only few bytes.

There are many rules triggering Skype alerts, and it seems that keeping only rule *5998* is sufficient to reduce the noise while keeping enough information to detect Skype usage. This result was attained by running the tool presented as Appendix E.2.

Rule *5998* detects Skype logins, which are mandatory. This login process is periodically repeated, so keeping only this rule also allows to record the timeframe of Skype's usage.

Emule is the file sharing protocol having the most rules in VRT and ET. However it also is the most difficult to detect file sharing protocol. With all rules based on two or four bytes, it has the weakest rules.

Emule rules pose problem on many levels; they are weak and computationally expensive: Most of them only check for patterns of two bytes in all UDP traffic. With random traffic, a two bytes pattern triggers an alert every 65'536 packet on average. With more than 100'000 IP packets per second going through the IDS during the day, this clearly poses a problem. Analyzing all traffic for small patterns, they also are quite computationally expensive. The two most time consuming P2P rules are *2003322* and *2003321.* According to Snort's performance profiling tool, each of them requires ten times more CPU time than other P2P rules.

With its default configuration and all Emule rules active, there were more than 230 devices detected as running Emule during a day. Almost all of those were also detected running Skype. An in-depth analysis of the alerts showed that the very large majority of Emule alerts were in fact generated by legit Skype traffic. Table 4 quickly presents the number of Emule alerts seen on the Skype port, and on other ports for 15 randomly selected hosts. More than 75% of the Emule alerts seem to be triggered by Skype. In almost all cases, a detailed analysis of the 25 remaining percents led to the conclusion that it was Skype running on an arbitrary port.

A Python tool was developed to try to find some patterns in the alerts. The idea was to find a subset of all the rules that successfully detected Emule traffic while keeping False Positives to a minimum.

While the perfect subset was not found, there was one rule that was almost always present in Emule traffic and did not seem to trigger too many false positives: *2001298* (presented below). This rule triggers on E2DK Server Status Request messages. The Emule protocol states

Host	Alerts on Skype port	On other port
1	17	0
2	16	0
3	0	11
4	19	0
5	41	0
6	0	10
7	18	0
8	15	0
9	10	0
10	12	0
11	15	0
12	0	27
13	0	6
14	22	0
15	9	0

Tab. 4: Number of Emule alerts on different ports. More than 75% of Emule alerts seem to be triggered by Skype.

that each Client should regularly send this message to stay in sync with the server [19].

```
alert udp $HOME_NET any -> $EXTERNAL_NET 4660:4799
(msg:"ET P2P eDonkey Server Status Request";
content:"|e3 96|"; offset: 0; depth: 2; classtype:
policy-violation;  sid: 2001298; rev:6;)
```

Rule *2001298* can even be improved by specifying the packet size to reduce the server load. The proposed modification is the following (Oinkmaster format, see chapter 8):

```
modifysid 2001298 "content:"|"dsize:6; content:"
```

There has been no case of false positive reported since Emule detection is based on this rule and this rule only, however a few true positives may have been missed. This rule only triggers on a port range, and if a user always connects to a server running outside of this port range or only uses the decentralized version of the protocol, it will remain undetected. However it is currently accepted at CERN to miss a few true positives, than to get a large number of false positives.

Bittorrent With 13 different rules, the Bittorrent protocol is well covered. There are all sorts of rules covering the entire possibilities of the network (DHT, tracker connection, transfers, User-Agents). This set was producing tens of thousands of alerts every day.

Running a home-made tool (Appendix E.2), a few dependencies between the rules have been found, and have allowed to considerably reduce the number of alerts. The most conclusive example is the following: At CERN, rule *2000334* trigger 20 times more alerts than rule *2181*, but *2000334* is never seen without *2181*, therefore it is enough to keep *2181* to cover the threat.

As a side note, it should be mentioned than most of todays Bittorrent clients support the use of protocol encryption. Usage of encryption renders inefficient all the rules based on peer to peer traffic, and therefore allow Bittorrent traffic to go through undetected.

A client connected to a https tracker, with protocol encryption enabled, and DHT disabled, cannot be detected by Snort. Such a configuration will be probably be common in a few months / years, rendering Snort inefficient to detect Bittorrent traffic. However this is not yet the case, and these rules detect users using this software every day.

A brief note on the future Most protocols are currently undergoing similar changes which will make them much more difficult to detect. Most of the current protocols now offer an “encrypted” mode, in which all packets are encrypted, and therefore no longer contain easily recognizable patterns.

When these changes become common, and the default configuration of P2P software enables them, it will be much more difficult to detect P2P using byte patterns in traffic. Other approaches such as machine learning or analysis of netflow data will probably give better results (see Chapter 4).

7.1.2 IRC

IRC is not allowed at CERN, due to its potential misuse in Botnets. Even if the use of IRC can be legit, IRC software is regularly used by attackers as part of underground networks for unauthorised access to computers.

Every instance of the IRC protocol should trigger a Snort alert. However, enforcing this policy is difficult, because many websites integrate chat applets based on the IRC protocol, triggering unneeded alerts.

Instead of using the provided IRC rules, CERN has written its own set of rules to detect IRC. This includes “pass” rules for several known & valid websites with embedded IRC.

CERN IRC rules are available as Appendix A.3.

The CERN IRC rules being complete and detecting every IRC protocol message, all other IRC rules have been disabled at CERN. There was no need to get more than one alert per message. Known malware using IRC to communicate should be detected by the CERN IRC signatures.

7.1.3 Other Policies

There are many other policies a company may try to enforce, and Snort has rules for most of them. However they were not part of this study. Most of them can be found in the rule files listed in Table 6 in Appendix A.2

7.2 Compromized

7.2.1 Definition

This set of rules was designed to detect compromized hosts or hosts infected by viruses or malware.

The set was initially containing the rule files listed in table 7 (Appendix A.2) [14].

7.2.2 Redefinition

Not compromized Threats detected by *spyware-put.rules* and *emerging-malware.rules* do not really fit into this category. These two files contain signature detecting adware and other badware, but this kind of software, even if very annoying for the end-user, do not indicate a “compromized” device. These files were quickly disabled due to the very large amount of hosts running such software. These sets may be reenabled in the future if usage of such software becomes a problem.

Shellcode Another file that was entirely disabled after some research is *shellcode.rules*. Most of the rules contained in this file were triggering regular alerts for many different hosts, and the amount of false positives it was generating was deemed excessive. Most of the rules in this file looked for specific binary pattern that may indicate a successful exploit. However, most of the byte patterns it was looking for were always regularly contained in legit files.

A quick example that can be easily checked is rule *1394*, which looks for a series of 31 consecutive 'A's (NOOP, byte value 0x41) in all traffic. Even if very specific, it seems that this string is used in many JPEG images to align the data and fill fields with placeholder data. A check was run against a web gallery, and out of the 1900 JPEG pictures it contained, 40 contained this specific pattern. So there are probably millions of legit images triggering this specific alert around the web.

Similar checks were conducted with the other *shellcode.rules* rules, and the results were always the same: there were normal files triggering alerts. Therefore this file was disabled.

7.2.3 Modifications done to the Set

Deploying the remaining set of files untouched triggers many alerts due to rules not complex enough and/or unneeded alerts.

This section will summarize the changes done to these files to optimize them for CERN environment. The modification details can be found as Appendix B.1.

Some rules were disabled because of their resource consumption, some because of their lack of complexity (they were triggering too many false positives), and others because they did not provide any useful results at CERN.

The process of selection was an iterative process. All of the rules were deployed, and all alerts were manually looked at and analyzed. All rules that were not meeting the requirements were disabled, and the new set was redeployed. This process was repeated until the fraction of false positives or unwanted alerts compared to real threats was acceptable.

All of these modifications are detailed as Appendix B.1, and are available in Oinkmaster format as Appendix C.2.

7.2.4 The Resulting Set

After these modifications, the resulting set is composed of 1'660 different rules suited for the CERN Public Network. Over 24 hours, Snort SP beta 2 is able to analyze more than 90% of the traffic on average, and more than 60% during peak hours.

In one month, 38 of those rules triggered 592 alerts related to confirmed security incidents. The worst performing rules of the remaining set are listed as Appendix B.2.

Most of the incidents detected triggered alerts repeatedly until the cases were closed. This confirmed the fact that it was not critical to analyze all traffic, because the alerts were likely to repeat themselves after some time.

Interestingly, 96% of those alerts were generated by Emerging Threats rules, and only 4% were generated by VRT rules. The set being the most productive was *ET TROJAN*, and there were very few alerts from the *attack-response* sets.

One probable explanation of these differences is that ET is community driven, whereas VRT rules are written by a team of Sourcefire experts. Being internationally spread, the community is probably much more efficient in writing rules detecting all kind of threats they have seen on their network, while Sourcefire team focuses on known exploits based on security bulletins. This doesn't mean that ET rules are better, but that they seem to be more oriented on detecting malware, whereas VRT rules seem to be more oriented on detecting known exploited vulnerabilites.

7.3 Attacks

7.3.1 Introduction

This set of rule tries to group all rules indicating that an attack is in progress. As any other big organization or company, the main problem with this set is that CERN is constantly under attack, and therefore there are constantly hundreds of alerts triggered by Snort.

Snort's attack coverage is very wide. There are rules aimed at detecting specific vulnerabilities, rules analyzing abnormal use of a protocol, rules detecting brute force attempts, rules detecting abnormal traffic, etc. Table 8 of Appendix A.2 lists all the files initially included in this set.

While incoming attacks are known and taken care of, outgoing attacks are much more interesting and could indicate compromized hosts.

To generate such a set, a program was written to "reverse" rules, so that they would consider the CERN Public Network as the potential source of attacks.

Both attack sets (normal and reversed) were deployed, and the same iterative process was started. Each alert was analyzed and the source rule disabled in case of false positive or unwanted alert.

However the amount of alerts was huge[3], and after three weeks of intensive sorting and processing the idea was abandoned (see details in the next sections).

The number of alerts had been considerably reduced, but in the three weeks these sets have been running, no useful alert had been seen.

The list of modifications done to this set is proposed as Appendix B.3.

7.3.2 Normal attacks

Deploying these files gave unusable results due to the large amount of alerts.

The most interesting thing to notice is that the vast majority of these alerts are informational, and not very useful in an environment such as the CERN. Such alerts include: ping of Windows hosts, access to the Google calendar service, data posted to a web form, link sent via MSN. These are only a few examples of informational alerts detected by this set.

There also are a lot of rules detecting known vulnerabilities, often of more than five years old. Interestingly, a lot of them trigger a lot of alerts on normal traffic. CERN mail servers, for example, were constantly triggering six different "overflow attemps" on perfectly normal traffic.

[3] more than 2'000'000 alerts per day, not counting rule *2001022* which was triggering 50'000 alerts per second

Having a timestamp on the rules could allow to easily deactivate old and deprecated rules. Unfortunately this field does not yet exists.

There also are a lot of rules targeted at specific web servers and applications. However for these rules to be useful, it is mandatory to know which webserver runs which operating system, and which webserver hosts which web application. This knowledge is difficult to have in a very large network with hundreds of web servers, and blindly enabling everything creates way too many false positives or unwanted alerts.

Snort also offer some DDOS rules detecting brute force attemps, scans, or blind large scale attacks against the network, but there are other means to detect these threats (Chapter 4).

Another interesting fact is that there are hundreds of rules detecting perfectly harmless and normal traffic and classified as “attempts”. The most noticeable rules doing this are the “ping” rules. There are tens of rules detecting all kind of pings. One quick example is the rule *480*, which is labeled ”ICMP ping speedera”, which is triggered by normal windows update behavior.

In a network of reasonable size these rules could provide useful information about incoming attacks. They just seem to be not fit to be used at CERN.

7.3.3 Reversed attacks

Using reverse attack rules did not really give more conclusive results. There were a lot of unforeseen consequences. The main results for this set are given below.

Web attacks There are eight rule files targeted at web attaks, each being specific to a web server, to a specific type of traffic, or to known web vulnerabilities.

All of the web alerts seen with the reversed set were legitimate. The main problem with those were the search engines. Request containing potentially malicious strings were always triggered by legitimate users querying search engines.

To illustrate this with a very simple example, imagine that a user is trying to insert something in a database. This user is very likely to query google for “INSERT INTO (...)”. This query will be posted in the URL via the GET method, and Snort’s SQL injection rules will think this is an injection attempt, and trigger an alert.

The inefficiency of these reversed web rules seems to be mainly due to the search engines. One could imagine to create exceptions for all the known search engines IPs, but unfortunately there is no such list and there will probably never be. Even if it existed, the list would be too long for Snort; Snort being slow to process IP lists in rules.

There were also a lot of rules triggering when a user accesses a potentially dangerous directory such as */cgi-bin* or */viewtopic.php*. Obviously a lot of websites meet these requirements on the web, therefore generating unwanted alerts.

Specific protocols attacks The reverse rules analyzing the SMTP, POP, IMAP, FTP, and other protocols did not give more conclusive results. For example, the only hosts triggering SMTP attack alerts were CERN mail servers, which were obviously not attacking anyone.

In one day, there were more than 80 IPs “attacking” the IMAP protocol, 90 “attacking” the POP protocol, and 38 “attacking” the FTP protocol.

A sample of these alerts was chosen and studied, and there was no confirmed case of attack. All these alerts seemed to have been triggered by perfectly standard software and traffic.

7.3.4 Rule Scoring

In order to improve the selection of rules, a new strategy was devised to try to sort the rules and keep only the interesting alerts.

The idea was to compute a “rule score” depending on each rule complexity, classtype, and specificities. If the rule score was above a threshold, then the rule would be kept, and if it was below the rule would be dismissed.

Here are the different factors that were measured by the rule analyzer:

- The number of bytes it is checking. The more the better.
- The placement of these bytes. Looking for a byte at a specific position is obviously better than looking for a byte anywhere in a packet.
- The number of ports concerned by the rule. The more specific the better.
- The packet size. If the rule specifies a packet size, it is obviously better than checking all traffic.
- Penalty for certain classtypes. Certain very specific rules are only classified as “not-suspicious” or “icmp-event”. To dismiss them as well some keywords were associated a penalty value to dismiss them.
- Other features, such as flowbits, or PCREs, which both improve a rule.

After some trial and error process while trying to find the ideal weights for the different parameters, it seemed that there was no direct correlation between complexity of a rule and its usefulness.

Starting again from scratch, by putting all the weights to 0 except the “content” score, thus reducing significantly the dimension space of the problem, traffic was gathered and alerts analyzed. Figure 2 presents the number of alerts opposed to the complexity of the rules triggering alerts over a day. There seem to be no direct relation between the complexity of a rule and the number of alerts it triggers.

Figure 3 presents the the number of CERN IPs triggering a rule opposed to the complexity (the score) of the rules. Here again, there seem to be no direct relation between the number of IPs triggered by a rule and its complexity. Also, this plot does not reflect the “usefulness” of the rules. Rules with high scores that could be selected by this process were mainly unneeded and informational rules.

As there seemed to be no direct correlation between the score of a rule and its efficiency/usefulness, this idea was abandoned too.

The python code computing the scores is joined as Appendix E.1.

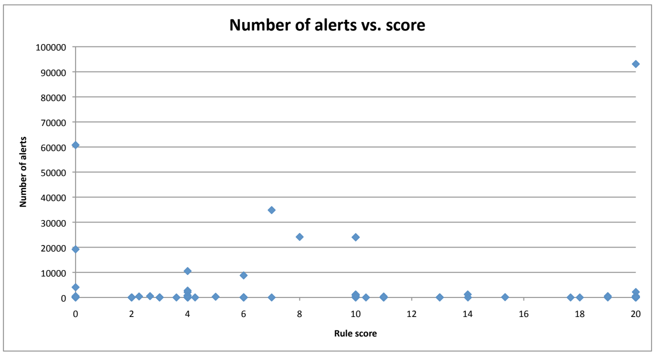

Fig. 2: Number of alerts vs score

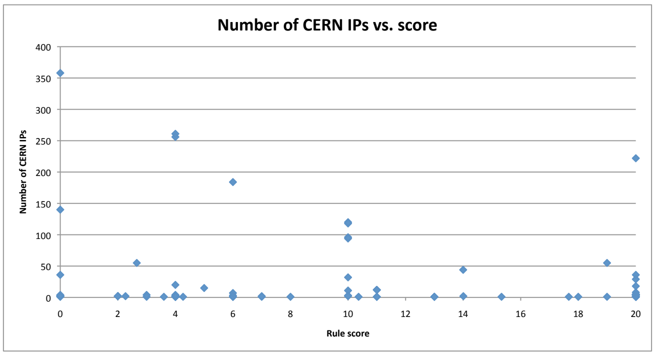

Fig. 3: Number of IPs vs score

7.3.5 Finally

As a good candidate set was not found for attack and reversed attack rules, the opposite approach was chosen for them. All attack rule was disabled, and only very few selected ones were finally enabled.

In June 2009, there were many new 0-day exploits that were discovered. Specific signatures detecting them quickly became available, and they were deployed.

To achieve good results in detecting attacks, it is recommended to know beforehand which attacks to look for, and deploy the corresponding rules accordingly.

7.4 Conclusions of the Evaluation

There are many different rules and not all are as useful as they seem.

The performance of a rule depends on how well it is written. Some rules have a very large impact on the traffic drop statistics, and they resource consuming rules are not always the expected ones.

In a large corporate network with unmanaged hosts, there will be a lot of unexpected traffic and protocols detected.

Due to its encryption, Skype really poses a problem in a large corporate network, where it triggers many unrelated alerts. It chooses a port randomly and sends random traffic to it. This random traffic, given enough time, will always trigger rules with low complexity.

Emule/ED2K is impossible to properly detect with the default rules when Skype is also present on the network. Most of Emule rules are not complex enough to be useful.

Rules cannot be deployed as they are and require modifications before being useful. There are rules that need to be modified to suit the environment, and other that need to be disabled. Deploying untouched rules produces way too many alerts to give useful results.

A set of rules detecting compromized and infected hosts has been proposed and tested thoroughly on the CERN Public Network.

It was much more difficult to obtain a clean set detecting attacks, either incoming or outgoing. Without knowing exactly what to look for, attack rules are too noisy to be useful.

There seem to be no correlation between a rule complexity and its usefulness. A tool was written to evaluate the rules based on their content, but the correct balance of weights has not been found to successfully use this tool to generate the wished ruleset.

8 Rules Management

8.1 The Problem

It has been seen that there are many different sources for finding rules that are regularly updated, and that rules cannot be deployed as they are. Many rules need to be disabled or modified by hand before they can be deployed, and these modifications have to be re-done every time a new ruleset becomes available.

So there is a need to properly handle the new set releases while keeping the specific modifications done to the sets.

8.2 Existing Tools

There are different tools available to help network administrators with their Snort sensor's administration.

The most noteworthy is Oinkmaster [20]. Oinkmaster is a very powerful Perl script that can do almost everything from a configuration file. Rules can be disabled, enabled, modified, added and deleted. The tool takes one or more untouched rule sets from the Internet, modifies it according to the configuration file, and generates a rule set that can be instantly deployed. The main advantage of using such a tool instead of disabling the rules one by one by hand, is that when a new set is available from the source, there is no need to re-apply all the modifications to this new set. The script does it automatically.

A sample Oinkmaster configuration file is proposed as Appendix C.2. It takes as input the VRT and ET sets (5.2.1); discards all the files classified as non-compromized or non-policy; disables rules that were triggering false positives at CERN or that were too resource consuming, modifies a few rules to increase their performance or comply with CERN needs; and generates output files containing only rules of the "compromized" category, and the rules related to the CERN policies.

A few other tools worth mentioning are "Dumb Pig" [21], that parses a snort rule-set, and depending on command line options, and recommends "fixes" for unperfect Snort rules, and "Pulled Pork" [22], which is a very promising replacement for Oinkmaster.

Dumb Pig can provide useful information on home-made or poorly written rules. It does a meta-analysis of the rules, to detect if they include all the necessary information that a rule should contain. It will propose fixes if it finds incoherences. However, given that rules are written by experts and checked by the community, its usefulness for commonly available rules is very limited.

Pulled Pork, on the other hand, provides moreless the same functionalities as Oinkmaster. However it is still in an early development phase; the project began in May 2009. Therefore this tool was not evaluated in this paper. Oinkmaster still remains the reference tool in this area.

8.3 Update Process

With the help of Oinkmaster, the update process is simple. The user only has to download the two rules sets tarballs from VRT and Emerging Threats, and run Oinkmaster on those sets.

The Oinkmaster configuration file is relatively easy to follow. It starts by defining where to find the rule sets, and then lists all the modifications that have to be done to these sets. The script is then called with two parameters, the location of the configuration file, and the output directory where it should put the final rules.

With the appropriate openings in the firewall, Oinkmaster could even automatically download the latest tarballs via HTTP, FTP, or even SSH.

After its run, Oinkmaster provides a short summary of the rules that were added/deleted/modified since the last update that can easily be reviewed by hand.

A sample report is included as Appendix C.1

The whole process could be put in a cron job and run automatically every day or week.

8.4 Oinkmaster Configuration

Here is a short description of Oinkmaster's most common options. Its configuration file is split in three different parts. The first part starts by defining where to find the rules tarballs and some other constants. The second tells Oinkmaster which files are irrelevant in those tarballs, and the last part applies modifications to the remaining rules.

For this last part, there are a few statements that need to be known:

disablesid:
: This is the most common. It completely disables a rule that is by default enabled in the tarballs.

enablesid:
: This can enable some rules that are by default disabled in the tarballs.

modifysid:
: This one offers the greatest flexibility, and allows to do pretty much anything with the rules. The standard syntax is *modifysid 1234 "foo" | "bar"*. This will replace foo with bar in rule 1234. Complex regular expressions can be used in these statements. It is important to note that this modification is applied only once for each rule. For a modification to be applied twice on a rule (to remove a keyword for example), it is needed to duplicate the statement (ie. with *modifysid 1234,1234*).

localsid:
: This one marks a rule as "modified locally". If a new version of the rule is downloaded, it will not be enabled, and the old one will be kept unharmed.

To reverse all the alert rules, for example, the following statements can be used:

```
modifysid * "^alert (\S+) (\S+) (\S+) (\S+) (\S+)" | \
"alert ${1} ${5} ${3} ${4} ${2}"
modifysid * "msg(\s?):\"" | "msg:\"REVERSED "
```

This will switch the local and remote addresses, and change to message alert to reflect the change.

Once the configuration file is complete, there are two parameters that need to be passed to Oinkmaster to execute the update process:

-C
: This will tell Oinkmaster where to find the configuration file. If it is omitted Oinkmaster will try to use */etc/oinkmaster.conf*.

-o
: The ouput directory. This is followed by the path where the rules are to be put.

The aforementioned functionalities make Oinkmaster an efficient and flexible tool to manage Snort rules.

9 Snort Optimizations

9.1 Using Snort SP: Multiple Analyzers

Snort SP claims to be much more performant than its predecessor due to multithreading. However, by default, Snort SP does not take full advantage of this possibility.

The Snort SP platform is built with three different layers, the source, the engine, and the analyzers (see Chapter 4). The source is responsible for capturing the traffic, and handing it to the engine. The engine preprocesses the traffic, and gives it to the analyzers, which, only then, will try to match the traffic with the enabled rules.

Due to the very large amount of traffic seen at CERN's Public Network, one analyzer is not enough to try and match all the wanted rules with every pre-processed packet. A proposed optimization is to split the rules in different analyzers, so that each of them is able to handle the flow of traffic and work on the traffic in parallel.

This is done by modifying the LUA configuration file in such a way that multiple analyzers are created, and attached to the engine. Each analyzer has its own configuration file, telling it what rules should be enabled.

The benefits of such a configuration are obvious, the amount of traffic that can be analyzed increases significantly with each new analyzer. With most high-end processors having now 16 cores, this architecture allows up to 14 analyzers (one thread per core).

The only disadvantage is that this has to be done manually and that Snort can not automatically split the rules in an optimal way. It also adds some overhead to the analysis.

An example configuration script is provided as Appendix D.1

Unfortunately, there is no documentation on those possibilities at the time of this writing.

9.2 Libpcap Modification

As seen in 9.1, there is only one thread in Snort SP that captures the traffic and feeds the Snort SP engine.

To capture the traffic, Snort relies on the libpcap library [23]. Before Snort can access the data, it has to be copied many times between the NIC, the kernel, and

Snort. On a high speed network, this consumes a lot of CPU cycles.

To vastly improve the packet capture performance, the default libpcap library can be replaced by another version radically changing the way data is passed around before entering the Snort engine.

Phil Wood's libpcap [24] takes full advantage of the Linux kernel options, and uses a MMAP ring buffer. With this new library, the data is immediately copied from the NIC to some memory space where Snort can access it. The use of this library greatly reduces the number of packets dropped.

To install Snort SP with this new libpcap the following steps are needed: Download and compile the new libpcap (there is no need to install it); and then build the Snort SP platform (not the analyzer) with the following flags: *–with-libpcap-libraries=/path/to/pcap/ –with-libpcap-includes=/path/to/pcap/*

9.2.1 Tests

The idea of improving the packet capture performance came from a simple observation: At CERN, during peak hours, on our test platform, even with no rule active and no preprocessor in Snort, around 10% of the traffic was reported as "dropped". Enabling the basic preprocessors increased this number to 15-20%.

The test platform is a custom built computer with a 16 core Intel Xeon E5472 with 16GB of RAM running 32-bit Scientific Linux CERN 5 (SLC5) [25], which is based on RedHat Enterprise Linux 5, and an Intel 10Gbit/s card to capture traffic.

Extended tests have been done concerning this modification.

For the first one, there were no Snort rules active, and only the "Stream 5" and the "http_inspect" preprocessors were enabled with their default configuration.

The plots in Figures 4 and 5 show the difference in packet loss before and after the modification. The amount of traffic was similar during the two experiments, but the percentage of dropped packets was not. We can see that when the number of captured packets exceeds 6M/minute, the default libpcap in unable to handle the traffic and starts loosing packets. The amount of packets dropped seems to be proportional to the traffic. With the "ringed" libpcap Snort seems unaffected (or at least less affected) by peaks in traffic. The CPU usage also dropped from around 170% to around 135% with the new library (it is more than 100% because of the multithreading on a multicore system).

To further check the improvements, the same experiment was conducted deploying some rules. The deployed rules are the "untouched" sets from VRT and Emerging Threats. Figure 6 show the percentage of packet loss per default set. We see a clear improvement in performance with this new library. Here again the amount of traffic was similar during the two experiments, and in all cases but one the results are significantly better with the new libpcap.

9.3 Snort Performance Profiling

To better understand Snort rules, the developers provided a very useful option: the performance profiling tool.

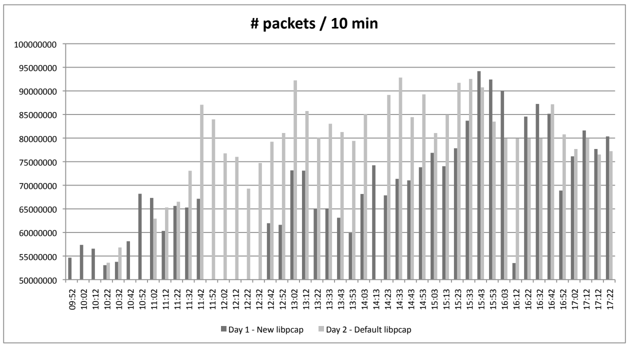

Fig. 4: Amount of traffic for the two measurements. Note that no data was collected between 11:42 and 12:32 on day 1.

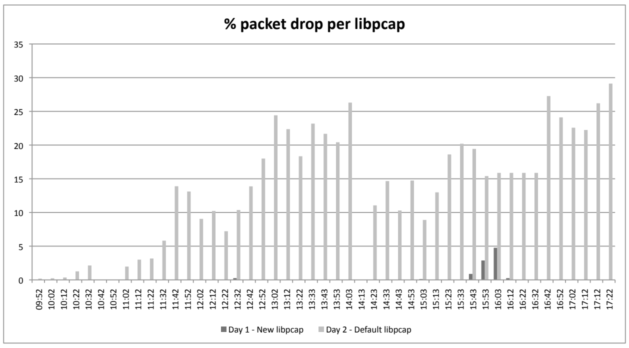

Fig. 5: % of packet drop for each libpcap

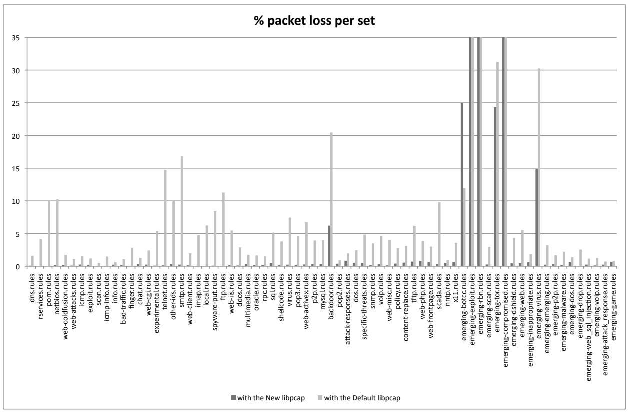

Fig. 6: % of packet drop per set with both libpcap

To use it, Snort and its engine must be built with the *–enable-perfprofiling* option. The tool should then be enabled in the engine configuration file, for example with *config profile_rules: print all, sort total_ticks*. This will print a list of all the rules that have been checked at least once during Snort's run, and sort them by total CPU time.

This will print some very useful information about the rules resource consumption when Snort exits. A sample report is available as Appendix D.2. Snort's manual mentions that the output of the performance profiling module can be printed to a file. However this option did not seem to work with our Snort SP version.

This option was very useful to evaluate the different rules, and to find which ones are consuming the most resources. Strangely, the most expensive rules were not always the expected ones.

Running Snort with this option does not seem to significantly impact snort performance. In fact no difference was noticed with and without it.

Here is an exemple of an unexpected expensive rule:

```
alert tcp $EXTERNAL_NET 1024: -> $HOME_NET 1024: (msg:"ET
TROJAN Beizhu/Womble/Vipdataend Controller Keepalive";
flowbits:isset,ET.vipde; flow:established,from_server; dsize:1;
content:"d"; classtype:trojan-activity; reference:(...);
sid:2008335; rev:6;)
```

This rule relies on a flowbit, and if the flowbit is set, then checks if the packet is of size 1, and that the byte is “d”. This rule was reported as one of the most time consuming. A lot of rules with FlowBits showed the same behavior. It seems that Snort spends a lot of time checking for FlowBits.

A few other things that Snort seems to not like in terms of CPU time is IP filtering, Perl regular expressions (PCRE), and sliding windows in all traffic (ie. checking for a series of bytes anywhere in all packets.

All IP-based rules (rules that were not relying on content, but only on a list of known remote IPs) performed very badly on our Snort sensor. This is probably due to the way Snort analyzes the traffic. The use of netflow data to do the exact same thing a posteriori seems to be much more efficient, as Snort can analyze much more traffic for details that netflow data does not contain.

PCRE on traffic were also slowing considerably the sensor. A few rules were modified to check for specific strings before checking the PCREs. Adding checks reduced the overall CPU time consumption. It is recommended to always check for the maximum possible specific strings before checking for PCRE, and to only validate the findings with the PCRE.

An example of a rule that has been modified in this way is the following:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"
(...)"; flow:to_server,established; content:"User-Agent\:";
nocase; pcre:"/User-Agent\:[^\n]+DEBUT\.TMP/i"; sid:2003427;
rev:3;)
```

This rule was running a PCRE check for all web requests. By simply adding *content:“DEBUT.TMP”;* before the PCRE, the load was significantly reduced. After the modification only packets containing both strings were checked with the PCRE.

The final rule becomes:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"
(...)"; flow:to_server,established; content:"User-Agent\:";
nocase; content:‘‘DEBUT.TMP"; nocase;
pcre:"/User-Agent\:[^\n]+DEBUT\.TMP/i"; sid:2003427; rev:3;)
```

10 Conclusion

This thesis presented Snort SP capabilities as an IDS in a large corporate network. The two main aspects of the software configuration were covered in this document: The *rules* and the *performance* problems. Several options to improve Snort’s performance have been proposed.

The bottleneck on a high-speed network with the default installation is the packet capture library. A replacement has been proposed and its efficiency has been shown. With its default configuration and no rule active, our Snort instance was dropping more than 15% of the traffic. This number was reduced close to zero with the new library.

To further improve the amount of traffic that can be analyzed by Snort SP, its ability to use multiple analyzers on the same traffic has been tested, and it’s efficiency is undeniable as it allows to multiply the number of rules deployed without increasing the packet loss ratio.

This paper also addressed different problems that arise when trying to deal with the rules. There is a clear lack of classification among all the available rules, and it is difficult to choose the ones that will be useful in a particular environment.

An in-depth analysis of all the available rules was done, and the rules have been re-classified in three main categories: policies, compromized, and attacks.

For the first two categories, a working set is proposed and tested on the CERN Public Network. These sets have been deployed, and have significantly improved the efficiency at detecting infected hosts, while reducing the number of false positives.

The problem of the coexistence of Skype and Emule rules has been addressed and some optimizations to bittorrent’s detection have been proposed.

Concerning the attacks, it seems that adapting the set to a very large environment requires a tremendous amount of work and extensive knowledge of the network, which is clearly not possible in a very large and heterogenous corporate network.

Once the rules have been chosen, there is no easy way to update the rulesets while keeping the changes intact. To take care of the update process, it is proposed to use the “Oinkmaster” software, which allows Snort’s administrators to easily and seamlessly update their snort rules.

10.1 Outlook

There is a clear lack of classification in the default rulesets available. It has been seen that a better classification will help many users to get the best out of their Snort sensor.

An additional field “date” on all the rules would significantly help on selecting current events rules and dismiss all the deprecated rules.

Another classification system, such as a tagging system, could also help in selecting rules if it were available.

With the correct weights, the rule score system proposed in Chapter 7.3.4 could also be of use. Unfortunately they have not been found.

Another possibilty to improve Snort’s performance could be to recompile Snort with another optimized compiler. Snort is currently built with GCC, but the Intel compiler has been known to produce faster code on Intel machines [26].

A Additional Data

A.1 Classtypes

Table 5 presents the default classtypes available in Snort with their description.

Classtype	Description
not-suspicious	Not Suspicious Traffic
unknown	Unknown Traffic
bad-unknown	Potentially Bad Traffic
attempted-recon	Attempted Information Leak
successful-recon-limited	Information Leak
successful-recon-largescale	Large Scale Information Leak
attempted-dos	Attempted Denial of Service
successful-dos	Denial of Service
attempted-user	Attempted User Privilege Gain
unsuccessful-user	Unsuccessful User Privilege Gain
successful-user	Successful User Privilege Gain
attempted-admin	Attempted Administrator Privilege Gain
successful-admin	Successful Administrator Privilege Gain
rpc-portmap-decode	Decode of an RPC Query
shellcode-detect	Executable code was detected
string-detect	A suspicious string was detected
suspicious-filename-detect	A suspicious filename was detected
suspicious-login	An attempted login using a suspicious username was detected
system-call-detect	A system call was detected
tcp-connection	A TCP connection was detected
trojan-activity	A Network Trojan was detected
unusual-client-port-connection	A client was using an unusual port
network-scan	Detection of a Network Scan
denial-of-service	Detection of a Denial of Service Attack
non-standard-protocol	Detection of a non-standard protocol or event
protocol-command-decode	Generic Protocol Command Decode
web-application-activity	access to a potentially vulnerable web application
web-application-attack	Web Application Attack
misc-activity	Misc activity
misc-attack	Misc Attack
icmp-event	Generic ICMP event
kickass-porn	SCORE! Get the lotion!
policy-violation	Potential Corporate Privacy Violation
default-login-attempt	Attempt to login by a default username and password

Tab. 5: Snort's description of the classtypes

A.2 File mappings

These tables present how the different available files are split into the three categories: policies, compromized, and attacks. (Tables 6, 7 and 8)

File
chat.rules
content-replace.rules
policy.rules
porn.rules
voip.rules
multimedia.rules
p2p.rules
chat.rules
emerging-game.rules
emerging-inappropriate.rules
emerging-p2p.rules
emerging-policy.rules

Tab. 6: Files containing policies rules

File
emerging-virus.rules
emerging-malware.rules
emerging-attack_response.rules
ddos.rules
virus.rules
backdoor.rules
spyware-put.rules
attack-responses.rules
shellcode.rules

Tab. 7: Files containing compromized rules

File
emerging-exploit.rules
emerging-scan.rules
emerging-web.rules
emerging-dos.rules
emerging-web_sql_injection.rules
emerging-voip.rules
dns.rules
rservices.rules
web-coldfusion.rules
icmp.rules
exploit.rules
scan.rules
finger.rules
web-cgi.rules
telnet.rules
smtp.rules
web-client.rules
imap.rules
ftp.rules
web-iis.rules
ddos.rules
oracle.rules
rpc.rules
sql.rules
shellcode.rules
pop3.rules
web-activex.rules
mysql.rules
dos.rules
specific-threats.rules
snmp.rules
voip.rules
web-misc.rules
web-php.rules
web-frontpage.rules
scada.rules
nntp.rules

Tab. 8: Files containing attacks rules

A.3 CERN IRC Rules

This file contains all the IRC rules deployed at CERN. They should catch any instance of the IRC protocol, except for the hosts that have pass rules.

```
#+##############################################################################
#                                                                              #
# File: cern-irc.rules                                                         #
#                                                                              #
# Description: IRC rules for CERN                                              #
#                                                                              #
#-##############################################################################

# define the ports where external IRC servers could run

portvar IRC_PORTS ![25,80,110,119,443,2401,8080]


#+++############################################################################
#                                                                              #
# IRC detection rules                                                          #
#                                                                              #
#---############################################################################

# note: to avoid performance problems, we only check the first 256 bytes

alert tcp any $IRC_PORTS <> any $IRC_PORTS ( \
  msg:"IRC DCC CHAT command"; \
  flow:established; \
  content:"PRIVMSG"; offset:0; depth:256; \
  pcre:"/^((\x3a[^\x00\x20\r\n]+\x20+)?\w+(\x20[^\x00\r\n]*)?\r?\n)*?(\x3a[^\x00\x20\r\n]+\x20+)?\
  PRIVMSG\x20+\S+\x20+\x3a\x01X?DCC\x20+CHAT\x20/is"; \
  classtype:policy-violation; \
  sid:3584031; rev:4; )

alert tcp any $IRC_PORTS <> any $IRC_PORTS ( \
  msg:"IRC DCC SEND command"; \
  flow:established; \
  content:"PRIVMSG"; offset:0; depth:256; \
  pcre:"/^((\x3a[^\x00\x20\r\n]+\x20+)?\w+(\x20[^\x00\r\n]*)?\r?\n)*?(\x3a[^\x00\x20\r\n]+\x20+)?\
  PRIVMSG\x20+\S+\x20+\x3a\x01X?DCC\x20+SEND\x20/is"; \
  classtype:policy-violation; \
  sid:3584032; rev:4; )

alert tcp any $IRC_PORTS <> any $IRC_PORTS ( \
  msg:"IRC CTCP command"; \
  flow:established; \
  content:"PRIVMSG"; offset:0; depth:256; \
  pcre:"/^((\x3a[^\x00\x20\r\n]+\x20+)?\w+(\x20[^\x00\r\n]*)?\r?\n)*?(\x3a[^\x00\x20\r\n]+\x20+)?PRIVMSG\x20+\S+\x20+\x3a\x01/is"; \
  classtype:policy-violation; \
  sid:3584021; rev:4; )

alert tcp any $IRC_PORTS <> any $IRC_PORTS ( \
  msg:"IRC CTCP reply"; \
  flow:established; \
  content:"NOTICE"; offset:0; depth:256; \
  pcre:"/^((\x3a[^\x00\x20\r\n]+\x20+)?\w+(\x20[^\x00\r\n]*)?\r?\n)*?(\x3a[^\x00\x20\r\n]+\x20+)?NOTICE\x20+\S+\x20+\x3a\x01/is"; \
  classtype:policy-violation; \
  sid:3584022; rev:4; )

alert tcp any $IRC_PORTS -> any $IRC_PORTS ( \
  msg:"IRC NICK command"; \
  flow:established; \
  content:"NICK"; offset:0; depth:256; \
  pcre:"/^((\x3a[^\x00\x20\r\n]+\x20+)?\w+(\x20[^\x00\r\n]*)?\r?\n)*?(\x3a[^\x00\x20\r\n]+\x20+)?NICK\x20/is"; \
  classtype:policy-violation; \
  sid:3584011; rev:4; )

alert tcp any $IRC_PORTS -> any $IRC_PORTS ( \
  msg:"IRC JOIN command"; \
  flow:established; \
  content:"JOIN"; offset:0; depth:256; \
  pcre:"/^((\x3a[^\x00\x20\r\n]+\x20+)?\w+(\x20[^\x00\r\n]*)?\r?\n)*?(\x3a[^\x00\x20\r\n]+\x20+)?JOIN\x20/is"; \
  classtype:policy-violation; \
  sid:3584012; rev:4; )

alert tcp any $IRC_PORTS <> any $IRC_PORTS ( \
  msg:"IRC PRIVMSG command"; \
  flow:established; \
  content:"PRIVMSG"; offset:0; depth:256; \
  pcre:"/^((\x3a[^\x00\x20\r\n]+\x20+)?\w+(\x20[^\x00\r\n]*)?\r?\n)*?(\x3a[^\x00\x20\r\n]+\x20+)?PRIVMSG\x20/is"; \
  classtype:policy-violation; \
  sid:3584013; rev:4; )
```

B Detailed results

B.1 For the “Compromized” set

B.1.1 Simple rules

The following rules were disabled because they were producing too many false positives due to their simplicity. Most of them are not checking enough bytes, and others are trying to detect common patterns, that are present in legit traffic.

141 :
checks for ”host” in traffic on one port. Often triggers false positives

152 :
This rule checks for three bytes on two ports. There has been false positives with linux build computers.

248 :
This one should detect a DDOS tool checking for “>”. But rsync data has been triggering alerts.

1292 :
Looks for the string “Volume serial number”, which has 146’000 google results, therefore 146’000 false positives on the web.

2123 :
Should detect a cmd.exe banner, but has only triggered by e-mails on mail servers.

5321, 5322, 5323 :
These rules should detect the sober worm, however all the alerts logged were simple TIME protocol to NIST server, which is legit traffic.

6031, 6033 :
These two rules should detect some trojan, but all alerts logged were triggered by the GRIDFTP protocol.

7672 :
This rule looks for the string “connected” on one port. Often triggers false positives.

8361 :
This should detect the black curse backdoor, which is a windows trojan, but this rule has been triggered by 3 linux computers in one day.

10442 :
Same as above, this should detect a windows worm, but has only detected linux computers. It checks for 5 bytes in all traffic.

2000040 :
This rule detects the string “up.exe” sent to some ports. This rule triggered two different false positives at CERN, where the user was uploading files via FTP. The file “setup.exe” would trigger an alert, which is of course unwanted.

2003555 :
It should detect Windows malware, but triggers regularly on clean windows computers. Only checks for a 6 bytes pattern.

2007594 :
This rule detects User-Agents starting by “Mz”. There has been false positives with the Symantec Liveupdate service, and some broadcasting korean server.

2007711 :
This rule only checks two bytes in UDP traffic.

2007840 :
This rules looks for http traffic with “Shell” as User-Agent. However in all the cases we detected, this was triggered by the MSN “Shell” client, which is no malware.

2007964, 2007963, 2007962 :
These rules are very weak, and look for two very common bytes in traffic.

2008056 :
This rule checks only two bytes, there has been many false positives, some come from Apple iDisk service.

2008103, 2008104, 2008105, 2008106, 2008107, 2008108, 2008109, 2008110 :
These rules do not rely on content. All packets on 1 port with a specific size trigger an alert.

2008468, 2008469 :
LDPinch, rule is complex, but many false positives with known websites.

2008547 :
This rule should detect trojan binaries. However it triggers also on normal downloads, often from clubic.com.

2009031 :
Should have detected malware, but triggers on some french ad server.

2009292 :
This rule should detect C&C responses, but it is triggered instead by axis network cameras.

2009522 :
This rule should detect when a fake gif is passed many arguments via its URI. However it seems that all the cases but one that were detected were false positives.

B.1.2 Unneeded

These rules were disabled because they did not provide any useful information for CERN.

518, 520, 1444 :
These rules trigger when they detect TFTP traffic.

721 :
This one triggers when it finds file attachments with bad extension (exe, chm, bat, ...). Such attachments are rejected by CERN mail servers.

1200 :
Looks for ”Invalid URL” in http traffic.

1201 :
Triggers when a HTTP 403 forbidden reply is received. This is not a threat.

12077 :
This looks for c99shell command requests. There are a lot of incoming requests, but there is no way to sort between successful and attempted attacks. The number of false positives outnumbers the real cases (zero found over one month).

2000345, 2000348, 2000347, 2000352 :
These detect IRC messages on non standard ports (nick change, join, privmsg, and dns). This is covered by CERN IRC rules.

2000562 :
Detects file attachments, which are normal.

2001689 :
This rule looks for potential bots scanning for SQL server. This rule does not report compromized hosts, and there are hundreds of bots at any time trying to scan the CERN Public Network from outside.

2001795 :
This rule triggers when an IP is sending more than 30 mail per minute towards CERN.

2001920 :
Looks in all SMTP traffic and catches gif.exe in incoming mail. But all .exe attachments are rejected by CERN mail servers.

2002322 :
This rule looks at all incoming MSN messages containing links ending in ".php". Any link to "index.php" would trigger an alert.

2002323 :
This rule detects exe files sent via msn.

2002894, 2002892, 2002895, 2001919 :
These rules detect viruses incoming or outgoing via SMTP. The only IPs triggering these alerts are CERN mail servers, which already drop incoming and outgoing viruses.

2003484 :
This rule should be called "malformed http request" instead of "virus...". It is triggered regularely by linux computers.

2007866 :
This one tries to detect gadu-gadu, which is not a trojan.

2008221, 2008222 :
And these two detects incoming potential phishing e-mails.

2008333, 2007774 :
Detects the "swizzor" adware, which is not a trojan.

2008411 :
This rule looks for people sending e-mails with "The Bat" mail client and having attachments.

2008576 :
Looks for tinype windows executables. Has triggered many times on legit remote hosts hosting normal files.

2009345 :
Triggers when a web server replies 401 unauthorized. This does not mean that a computer has been compromized.

2009346 :
This does not report a compromized host. It detects http bruteforce (many 401 errors during a short period).

B.1.3 Resource consuming rules

These rules were commented because they consumed a lot of resources, and their usefulness was discussable. The performance was evaluated using Snort's performance profiling tool. Here is the list of the disabled rules and the reasons:

7101 & 7103 :
Dependant on 7102. No use if 7102 is disabled.

7102 :
It detects a Spyware dated from 2004. And the load is very high.

7716 :
7715 has been modified to trigger an alert instead.

7761 :
This rule was the winner in times of resource consumption. It tracks a malware dated 2004. Any antivirus should detect it.

13509 :
13508 is complex enough and has been modified to trigger an alert to reduce load. Therefore the modified version of 13508 already covers this threat.

2002031 :
IRC - potential download or upload. IRC is covered by the CERN rules.

2002032 :
IRC - potential bad command. IRC is covered by the CERN rules.

2003176 :
Detects a packet of 4 null bytes. The load is very high, and it detects a mail-spreading worm of 2006.

2003380 :
Looks for a suspicious User-Agent. It checks for PCRE on all User Agent strings, and the rule is impossible to modify to improve its performance.

2003427 :
RxToolbar. Very high load and it is only adware, so no real threat.

2007583 :
Looks for User-Agent "IEbar". Induces a very high load.

2008178 :
2008177 is already covering this threat and it is consuming too much resources.

2008335 :
2008334 is already covering this threat and it is consuming too much resources.

2009026 :
: 2009025 is already covering this threat and it is consuming too much resources.

B.1.4 Modifications Done to the Remaining Rules

There were also a few rules that were modified in order to increase performance or reduce dependencies. Here is the summary of the changes.

2003427, 2007583 :
: These two signatures were modified in order to increase the system performance. Both signatures were checking if the packet contained a "User-Agent", and if found were trying to match it using PCRE. Both were modified to also check for the specific User-Agent before doing the PCRE check. The additions were respectively "content:'DEBUT.tmp'" "and "content:'iebar'.

7118 :
: This rule is new, and uses a token that Snort SP beta 2 does not understand: "http_header". It has been removed from the rule.

  This modification will be removed with a new Snort SP version.

2008335, 2009026, 2003176 :
: These rules were modified to improve the performance. These 3 rules check for a flowbit, then if the flowbit is found check that the packet has a certain size. The order of these two instructions was reversed.

13508, 7715:
: These two rules were modified to produce alerts. They were initially created to set a flowbit that another rule would check, but these other rules were disabled to increase performance. These rules are complex enough, and are reliable enough to create alerts. The "noalert" keyword was removed from them.

498 :
: This rule checked all traffic for root, uid 0. It was modified to only trigger an alert on outgoing traffic.

B.2 Worst Performers for the Remaining “Compromized” Set

The results below present the worst performing rules for the “Compromized” remaining rule set. This set is split in 2 threads using the multiple analyzers modification.

```
Thread 1:
- emerging-virus.rules

Rule Profile Statistics (all rules)
==========================================================
   Num      SID GID     Checks   Matches    Alerts           Microsecs  Avg/Check  Avg/Match Avg/Nonmatch
   ===      === ===     ======   =======    ======               =====  =========  ========= ============
     1  2008730   1 1826668853         0         0           105796457        0.1        0.0          0.1
     2  2009291   1  629264853         0         0            76022043        0.1        0.0          0.1
     3  2007585   1 1134700181         0         0            72686573        0.1        0.0          0.1
     4  2003175   1 1303177316       370         0            72206368        0.1        0.1          0.1
     5  2008245   1  925971873       285         0            51205263        0.1        0.1          0.1
     6  2009081   1   22238944         0         0            34261723        1.5        0.0          1.5
     7  2003427   1   22239491         0         0            26843863        1.2        0.0          1.2
     8  2008182   1   21506124         0         0            15259640        0.7        0.0          0.7
     9  2008452   1   23038075         0         0            14064426        0.6        0.0          0.6
    10  2008493   1   21189840         0         0            14006085        0.7        0.0          0.7
    11  2008546   1   22878926         0         0            13905275        0.6        0.0          0.6
    12  2009450   1   21066530         0         0            13872456        0.7        0.0          0.7
    13  2008482   1   23076539         0         0            13841727        0.6        0.0          0.6
    14  2008580   1   25073430         0         0            13274268        0.5        0.0          0.5
    15  2009351   1   32386660         0         0            12952928        0.4        0.0          0.4
    16  2009458   1   22200961         0         0            12910578        0.6        0.0          0.6
    17  2009521   1   21902878         0         0            12790560        0.6        0.0          0.6
    18  2009299   1   23383335         0         0            12608802        0.5        0.0          0.5
    19  2009531   1   21605641         0         0            12608399        0.6        0.0          0.6
    20  2008194   1   20425773         0         0            12380037        0.6        0.0          0.6
    21  2009300   1   21064288         0         0            11864142        0.6        0.0          0.6
    22  2008639   1   20359221         0         0            11850647        0.6        0.0          0.6
    23  2009374   1   20860333         0         0            11785922        0.6        0.0          0.6
    24  2009519   1   20775807         0         0            11686315        0.6        0.0          0.6
    25  2008377   1   21954815         0         0            11645410        0.5        0.0          0.5
    26  2008461   1   21950965         0         0            11443947        0.5        0.0          0.5
    27  2009389   1   20944548         0         0            11309687        0.5        0.0          0.5
    28  2008317   1   21020113         0         0            11262052        0.5        0.0          0.5
    29  2009526   1   21320081         0         0            11200685        0.5        0.0          0.5
    30  2008329   1   22065578         0         0            11183437        0.5        0.0          0.5


Thread 2:
- ddos.rules
- emerging-attack_response.rules
- virus.rules
- attack-responses.rules
- backdoor.rules

Rule Profile Statistics (all rules)
==========================================================
   Num      SID GID     Checks   Matches    Alerts           Microsecs  Avg/Check  Avg/Match Avg/Nonmatch
   ===      === ===     ======   =======    ======               =====  =========  ========= ============
     1     7723   1  168413564         0         0           362226779        2.2        0.0          2.2
     2     6396   1   22239491         0         0            53247314        2.4        0.0          2.4
     3     5320   1   23122000         0         0            46377813        2.0        0.0          2.0
     4    12661   1   22239491         0         0            42201974        1.9        0.0          1.9
     5     7751   1  348634798         0         0            26308135        0.1        0.0          0.1
     6     6140   1  112509659     30505         0            22423457        0.2        0.1          0.2
     7     7786   1  125158823    154282         0            18693660        0.1        0.1          0.1
     8     6401   1  160723890         0         0            17989233        0.1        0.0          0.1
     9    12166   1  169493945         0         0            16930946        0.1        0.0          0.1
    10     7067   1   66019574         0         0            14358016        0.2        0.0          0.1
    11     6030   1   85784136    302088         0            12031188        0.1        0.1          0.1
    12    12146   1   85784136    256202         0            11158196        0.1        0.1          0.1
    13     7610   1   70121360      2228         0            10803136        0.2        0.1          0.2
    14     6298   1   79796274         0         0             9790261        0.1        0.0          0.1
    15     6027   1   50314179         0         0             8375946        0.2        0.0          0.2
    16     7693   1   70529642         5         0             8327883        0.1        0.1          0.1
    17     7636   1   69094908         0         0             8118030        0.1        0.0          0.1
    18     7715   1  189330186         0         0             7710739        0.0        0.0          0.0
    19     7606   1   70410947     13650         0             6839717        0.1        0.1          0.1
    20    13654   1   21823328         0         0             6768424        0.3        0.0          0.3
    21     7072   1   56720486         0         0             6374444        0.1        0.0          0.1
    22    13856   1   22239491         0         0             4446687        0.2        0.0          0.2
    23     7657   1   33103205         0         0             4246800        0.1        0.0          0.1
    24     7612   1   61357304         0         0             4243033        0.1        0.0          0.1
    25    13942   1   22730983         0         0             4210179        0.2        0.0          0.2
    26     6023   1   22239495         0         0             3990634        0.2        0.0          0.2
    27     7077   1   22239702         0         0             3975854        0.2        0.0          0.2
    28     7656   1   32477973       820         0             3965702        0.1        0.1          0.1
    29     7648   1   20592169        38         0             3068717        0.1        0.1          0.1
```

B.3 For the "attacks" set

Here a list of all the SIDs that were disabled and the reason. All rules starting by 7 are the reverse version of the same number without it.

All the files listed in Table 8 of Appendix A.2 were initially enabled and the following rules disabled. Rules are listed in the order they were disabled.

Note that even after having disabled all these rules this set was still producing a lot of unwanted alerts.

2001022:
Detects fragmented packets... 50000 alert per second.

486:
"ICMP Destination Unreachable Communication with Destination Host is Administratively Prohibited".

480:
"ICMP ping speedera" This is normal windows update behavior.

8428:
Https traffic with some flags not set. The flags are probably not set due to Snort's packet drop.

485:
"ICMP Destination Unreachable Communication Administratively Prohibited"

882:
URI contains "Calendar".

466:
"ICMP L3retriever Ping". This is apparently normal with windows

1394:
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" in payload.. should detect shellcode noop, but is contained in many normal images.

13514:
Triggered when the words "update" and "set" are present on the same line

2002922:
"ET POLICY VNC Authentication Successful". This is allowed at CERN.

2002912, 2002913, 2002914, 2002915, 2002916, 2002917, 2002918, 2002919, 2002920, 2002921, 2002923, 2002924 :
Real VNC stuff, induces high load, and rules are too targeted.

2001090:
Triggers when javascript is found on a web page but not enclosed by the appropriate "javascript" tag. Induces a very high load.

2001091:
Same for Visual Basic

2001092:
Same for Microsoft Access shell code.

13819:
Triggers when "Accept-Language" is more than 100 bytes. This is often the case.

1852:
Triggers when the file robots.txt is accessed. The purpose of this file is to be accessed.

2001674:
This rule triggers on "POST" requests containing "http://". There has been many false cases.

469:
Triggers on ICMP type 8 (echo request). This is normal.

11974:
Triggers on all traffic on one port when the size is smaller than 11 bytes.

823:
Triggers on all "cvweb.cgi" accesses.

2001219:
SSH scan. Can be detected by other means.

2003068:
SSH scan outbound. Can be detected by other means.

2517:
"IMAP PCT Client_Hello overflow attempt" . 16739 alerts were triggered by a perfectly normal xchange server.

72517:
Same rule reversed.

1042:
Triggers when the string "Translate: F" is in http requests.

71042:
Same reversed.

2000536, 2000537,2000538, 2000540, 2000543, 2000544, 2000545, 2000546:
Those are NMAP scans. There are too many of them and not much can be done to prevent them.

2003099:
Triggers when a null byte is found in the URI of a web request. There were more than 4000 alerts per day coming from lots of IPs.

72003099:
Same reversed.

483:
Triggers on ICMP traffic containing "AAAAAAAA". Cyberkit triggers those alerts.

10995:
SMTP possible BDAT DoS attempt. 3620 alerts on CERN mail server on one day.

710995:
And its reverse.

13948:
"DNS large number of NXDOMAIN replies - possible DNS cache poisoning". Triggers on normal traffic.

72001621:
Potential PHP SQL injection attack. "Potential".

895:
Triggers when "/redirect" is found in URLs. Why not?

7895:
Same reversed.

1968, 1998, 1999, 2000, 2229, 72229:
Normal ".php" file access (such as viewtopic.php).

all SIDs having "web-application-activity" as classtype:
They provided no real information about attacks and were informational.

712007:
"401 Unauthorized" in SIP/2.0 protocol. Many alerts.

853:
When "/wrap" is contained in url. Lots of normal web requests.

7853:
Same reversed.

2329:
SQL overflow attempt , checks only 2 bytes on any ports.

72329:
Same reversed.

2002851:
FTP LIST without login. There are lots of FTPs.

72007873:
Triggers on GET requests for files ending in .exe, .bat, .dll, ...

1156:
When "////////" is contained in a packet. Reported as "apache directory disclosure attempt".

478:
Triggered by ICMP Broadscan Smurf Scanner. Not much can be done about it.

72002997:
Reversed ET WEB PHP Remote File Inclusion (monster list http). We don't really care if an external website is potentially vulnerable.

8440:
Too many alerts / IPs to be usable.

72002992, 72002993, 72002994, 72002995:
Triggers on 10 mail connections in 2 minutes outgoing.

11969:
VOIP-SIP inbound 401 unauthorized message.

1288:
"/_vti_bin/" request.

78734:
"REVERSED WEB-PHP Pajax arbitrary command execution attempt" . Triggers on normal traffic with google servers.

web-misc.rules :
This file contained too many rules not suited for a large environment.

2006445:
Triggers when "SELECT" and "FROM" were found in a packet. There were many false positives.

72006445:
Same reversed.

72001087:
Reversed "ET WEB-MISC cross site scripting attempt to execute Javascript code". No use for remote sites.

web-frontpage.rules:
Mainly alerts related to accesses to some files. Too many unwanted alerts.

7969:
reversed "WEB-IIS WebDAV file lock attempt". This is a normal feature of webdav.

All rules having "access" in their name in all the web* files:
Access rules did not provide useful information.

Whole sets of rules:

- emerging-web.rules
- emerging-web_sql_injection.rules
- emerging-voip.rules
- web-coldfusion.rules
- web-cgi.rules
- web-client.rules
- web-iis.rules
- web-activex.rules
- snmp.rules
- voip.rules
- web-misc.rules
- web-php.rules
- web-frontpage.rules

474:
ICMP traffic containing "—00 00 00 00 00 00 00 00—". Why not?

2002995:
Potential IMAP scan. Can be detected with some other means.

2006546:
SSH bruteforce. Can be detected with other means.

2006435:
SSH bruteforce. Can be detected with other means.

73072:
Imap status overflow attempt. Always triggered by legitimate traffic.

2590:
Smtp mail from overflow attempt. Idem.

2183:
“Content transfer encoding” overflow attempt. Always triggered by legitimate traffic.

713513:
Reversed web traffic containing ”insert * into”. Triggered by search engine requests.

2003:
SQL worm propagation attempt. Triggered by 52 cern machines. No virus found on some of those. Probably false positives.

2050:
SQL overflow attempt. Checks for one byte in all packets of size bigger than 100 bytes to port 1434. Lots of false positives.

72250:
Checks for POP3 user with % at the end. Triggered by legitimate traffic... (OVH and some physics lab in Japan)

13512:
Checks for “exec master” SQL in traffic. Lots of false positives.

13513:
Checks for “insert into” SQL statement in traffic. Lots of false positives.

713695:
“Reversed Real Helix server 2002 vulnerability”. This reversed alerts is triggered by legitimate rtsp akamai servers.

C Rules Management

C.1 Oinkmaster Sample Report

This is report provided by Oinkmaster after a run. It provides a summary of the modifications apported to the rules: tells you which one have been modified, which ones have been disabled, etc.

It even warns you when a file is added or deleted, so that the changes can be reflected in the snort configuration file.

This is very useful to merge the local changes apported to the rules, with the new rules provided by different sources.

```
[***] Results from Oinkmaster started 20090721 15:52:29 [***]

[+++]          Added rules:          [+++]

     -> Added to backdoor.rules (1):
        alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"BACKDOOR Infector.1.x"; flow:established,from_server; content:"WHATISIT";\
        metadata:policy balanced-ips drop, policy connectivity-ips drop, policy security-ips drop; reference:arachnids,315;\
        reference:cve,1999-0660; reference:nessus,11157; classtype:misc-activity; sid:117; rev:10;)

[///]     Modified active rules:     [///]

     -> Modified active in emerging-attack_response.rules (1):

        old: alert tcp any any -> any any (msg:"ET ATTACK_RESPONSE Bindshell2 Decoder Shellcode"; content:"|53 53 53 53 53 43 |";\
             content:"|66 53 89|"; distance:0; classtype:shellcode-detect; reference:url,doc.emergingthreats.net/2009246;\
             reference:url,www.emergingthreats.net/cgi-bin/cvsweb.cgi/sigs/ATTACK_RESPONSE/ATTACK_RESPONSE_Common_ShellCode;\
             sid:2009246; rev:1;)
        new: alert tcp any any -> any any (msg:"ET ATTACK_RESPONSE Bindshell2 Decoder Shellcode"; \
             content:"|53 53 53 53 53 43 53 43 53 FF D0 66 68|"; content:"|66 53 89 E1 95 68 A4 1A|"; distance:0;\
             classtype:shellcode-detect; reference:url,doc.emergingthreats.net/2009246; \
             reference:url,www.emergingthreats.net/cgi-bin/cvsweb.cgi/sigs/ATTACK_RESPONSE/ATTACK_RESPONSE_Common_ShellCode;\
             sid:2009246; rev:2;)

[---]         Disabled rules:        [---]
    None.

[---]         Removed rules:         [---]

     -> Removed from backdoor.rules (1):
        alert udp any any -> $HOME_NET 3344 (msg:"A nice rule"; flow:to_server; content:"logged in"; metadata:policy\
        security-ips drop; reference:arachnids,83; classtype:misc-activity; sid:234162; rev:6;)

[*] Non-rule line modifications: [*]
    None.

[*] Added files: [*]
    None.
```

C.2 Oinkmaster Sample Configuration File

This is the proposed Oinkmaster configuration file. It contains rules detecting compromized hosts, as well as rules detecting CERN policy violations.

```
# $Id: oinkmaster.conf,v 1.132 2006/02/02 12:05:08 andreas_o Exp $ #

url = file://snortrules-snapshot-2.8.tar.gz
url = file://emerging.rules.tar.gz

path = /bin:/usr/bin:/usr/local/bin

# Files in the archive(s) matching this regular expression will be
# checked for changes, and then updated or added if needed.
# All other files will be ignored. You can then choose to skip
# individual files by specifying the "skipfile" keyword below.
# Normally you shouldn't need to change this one.
update_files = \.rules$|\.config$|\.conf$|\.txt$|\.map$


#######################################################################
# Files to totally skip (i.e. never update or check for changes)      #
#                                                                     #
# Syntax: skipfile filename                                           #
# or:     skipfile filename1, filename2, filename3, ...               #
#######################################################################

# Ignore local.rules from the rules archive by default since we might
# have put some local rules in our own local.rules and we don't want it
# to get overwritten by the empty one from the archive after each
# update.
skipfile local.rules
skipfile VRT-License.txt

# The file deleted.rules contains rules that have been deleted from
# other files, so there is usually no point in updating it.
skipfile deleted.rules

# We skip all files except those belonging to the "compromized" set

skipfile bad-traffic.rules
skipfile chat.rules
skipfile content-replace.rules
skipfile dns.rules
skipfile dos.rules
skipfile emerging-botcc-BLOCK.rules
skipfile emerging-botcc.excluded
skipfile emerging-botcc.rules
skipfile emerging-compromised-BLOCK.rules
skipfile emerging-compromised.rules
skipfile emerging-dos.rules
skipfile emerging-drop-BLOCK.rules
skipfile emerging-drop.rules
skipfile emerging-dshield-BLOCK.rules
skipfile emerging-dshield.rules
skipfile emerging-exploit.rules
skipfile emerging-inappropriate.rules
skipfile emerging-malware.rules
skipfile emerging-policy.rules
skipfile emerging-rbn-BLOCK.rules
skipfile emerging-rbn.rules
skipfile emerging-scan.rules
skipfile emerging-sid-msg.map
skipfile emerging-sid-msg.map.txt
skipfile emerging-tor-BLOCK.rules
skipfile emerging-tor.rules
skipfile emerging-voip.rules
skipfile emerging-web.rules
skipfile emerging-web_sql_injection.rules
skipfile emerging.conf
skipfile emerging.rules
skipfile experimental.rules
skipfile exploit.rules
skipfile finger.rules
skipfile ftp.rules
skipfile icmp-info.rules
skipfile icmp.rules
skipfile imap.rules
skipfile info.rules
skipfile misc.rules
skipfile multimedia.rules
skipfile mysql.rules
skipfile netbios.rules
skipfile nntp.rules
skipfile oracle.rules
skipfile other-ids.rules
skipfile policy.rules
skipfile pop2.rules
skipfile pop3.rules
skipfile porn.rules
skipfile rpc.rules
skipfile rservices.rules
skipfile scada.rules
skipfile scan.rules
skipfile shellcode.rules
skipfile smtp.rules
skipfile snmp.rules
skipfile specific-threats.rules
skipfile spyware-put.rules
skipfile sql.rules
skipfile telnet.rules
skipfile tftp.rules
skipfile voip.rules
skipfile web-activex.rules
skipfile web-attacks.rules
skipfile web-cgi.rules
skipfile web-client.rules
skipfile web-coldfusion.rules
skipfile web-frontpage.rules
skipfile web-iis.rules
skipfile web-misc.rules
skipfile web-php.rules
skipfile x11.rules



# Also skip snort.conf by default since we don’t want to overwrite our
# own snort.conf if we have it in the same directory as the rules. If
# you have your own production copy of snort.conf in another directory,
# it may be really nice to check for changes in this file though,
# especially since variables are sometimes added or modified and
# new/old files are included/excluded.
skipfile snortsp.conf



##########################################################################
# SIDs to modify after each update (only for the skilled/stupid/brave).  #
# Don’t use it unless you have to. There is nothing that stops you from  #
# modifying rules in such ways that they become invalid or generally     #
# break things. You have been warned.                                    #
# If you just want to disable SIDs, please skip this section and have a  #
# look at the "disablesid" keyword below.                                #
#                                                                        #
# You may specify multiple modifysid directives for the same SID (they   #
# will be processed in order of appearance), and you may also specify a  #
# list of SIDs on which the substitution should be applied.              #
# If the argument is in the form something.something it’s regarded       #
# as a filename and the substitution will apply on all rules in that     #
# file. The wildcard ("*") can be used to apply the substitution on all  #
# rules regardless of the SID or file. Please avoid using #comments      #
# at the end of modifysid lines, they may confuse the parser in some     #
# situations.                                                            #
#                                                                        #
# Syntax:                                                                #
#   modifysid SID "replacethis" | "withthis"                             #
# or:                                                                    #
#   modifysid SID1, SID2, SID3, ... "replacethis" | "withthis"           #
# or:                                                                    #
#   modifysid file "replacethis" | "withthis"                            #
# or:                                                                    #
#   modifysid * "replacethis" | "withthis"                               #
#                                                                        #
##########################################################################


#to improve performance
modifysid 2003427 "content:\"User-Agent\\\:\"; nocase;" | "content:\"User-Agent\\\:\"; nocase; content:\"DEBUT.TMP\"; within:10;"
#modifysid 2007583 "content:\"User-Agent\\\:\"; nocase;" | "content:\"User-Agent\\\:\"; nocase; content:\"iebar\"; within:10;"

#http_header not implemented on snortspbeta2
modifysid 7118,7118 "http_header;" | ""

#dsize before flowbit to improve performance
modifysid 2008335 "flowbits:isset,ET.vipde; flow:established,from_server; dsize:1;" | "dsize:1; flowbits:isset,ET.vipde; \
flow:established,from_server;"
modifysid 2009026 "flowbits:isset,ET.vipdataend; flow:established,to_server; dsize:1;" | "dsize:1; flowbits:isset,ET.vipdataend; \
flow:established,to_server;"
modifysid 2003176 "flowbits:isset,BEposs.warezov.challenge; flow:established,from_server; dsize:4;" | "dsize:4; \
flowbits:isset,BEposs.warezov.challenge; flow:established,from_server;"

#the two following statements removes "noalert" from rules that were not producing alerts but only setting flowbits.
#Rules reading the flowbits have been disabled
modifysid 13508 "flowbits:noalert;" | "" #generate an alert, because 13509 has been disabled below
modifysid 7715 "flowbits:noalert;" | "" #generate an alert, 7716 has been disabled because of resource consumption

# rule detecting root uid0 in traffic. Modified so that only outgoing triggers an alert
modifysid 498 "any any -> any any" | "\$HOME_NET any -> any any"

########################################################################
# SIDs that we don’t want to update.                                   #
# If you for some reason don’t want a specific rule to be updated      #
# (e.g. you made local modifications to it and you never want to       #
# update it and don’t care about changes in the official version), you #
# can specify a "localsid" statement for it. This means that the old   #
# version of the rule (i.e. the one in the rules file on your          #
# harddrive) is always kept, regardless if the official version has    #
# been updated. Please do not use this feature unless in special       #
# cases as it’s easy to end up with many signatures that aren’t        #
# maintained anymore. See the FAQ for details about this and hints     #
# about better solutions regarding customization of rules.             #
########################################################################

# Example to never update SID 1325.
# localsid 1325



########################################################################
# SIDs to enable after each update.                                    #
# Will simply remove all the leading ’#’ for a specified SID (if it’s  #
# a multi-line rule, the leading ’#’ for all lines are removed.)       #
# These will be processed after all the modifysid and disablesid       #
# statements. Using ’enablesid’ on a rule that is not disabled is a    #
# NOOP.                                                                #
#                                                                      #
# Syntax:  enablesid SID                                               #
# or:      enablesid SID1, SID2, SID3, ...                             #
########################################################################

# Example to enable SID 1325.
# enablesid 1325

enablesid 2003427


########################################################################
# SIDs to comment out, i.e. disable, after each update by placing a    #
# ’#’ in front of the rule (if it’s a multi-line rule, it will be put  #
# in front of all lines).                                              #
#                                                                      #
# Syntax:  disablesid SID                                              #
# or:      disablesid SID1, SID2, SID3, ...                            #
########################################################################

#Disabled for performance reasons:
disablesid 7101,7103 # gwboy, dependant on 7102
disablesid 7102    # gwboy. spyware from 2004. high load...
#disablesid 7694   #Exception backdoor
#disablesid 7723   #Wollf remote manager
disablesid 7761          # 3x more time consuming than second most consuming rules. track down anal ftp, which dates from 2004. Any antivirus should d
disablesid 13509   # 13508 is complex enough and has been modified to trigger an alert (13509 is the most resource consuming rule)
disablesid 2002031 # IRC - potential DL or UL. IRC should have been detected already
disablesid 2002031 # very high load all packets sometimes with pcre. irc should have been detected sooner
disablesid 2002032 # IRC - potential bad command. idem
disablesid 2003176 # Warezoft trojan - packet of 4 null bytes - very high load - mail-spreading worm of 2006 -> protected
disablesid 2003380 # Suspicious User-Agent - pcre on all UA, difficult to modifysid efficiently
disablesid 2003380 # does pcre on ALL UAs... disabled
disablesid 2003427 # RxToolbar UA - Very high load and only adware
#disablesid 2007583 # iebar spyware. reenabled
disablesid 2008178 # TROJAN Ceckno Keepalive from Controller - covered by 2008177
disablesid 2008335 # 2008334 already covers this threat
disablesid 2009026 # 2009026 already covers this threat


#Disabled because they provide no real useful information
disablesid 221     #TFN probe, attack, not compromized
disablesid 518     # TFTP normal usage
disablesid 520     # TFTP normal usatge
disablesid 721     # File attachments with bad extension (exe chm bat ...)
disablesid 1200    # "Invalid URL" in http
disablesid 1201    # 403 forbidden
disablesid 1444    # TFTP normal usage
disablesid 12077   # c99shell command request. a lot of incoming requests.
disablesid 2000345,2000348,2000347,2000352,2002363 # IRC normal messages, and potential bad command. Should be detected by own IRC rules
disablesid 2000562 # File attachments
disablesid 2001689 # potential bot scanning for sql server. who cares. Only checks traffic on one port with no content...
disablesid 2001795 #more than 30 mail from per minute towards cern
disablesid 2001919 #incoming virus by mail
disablesid 2001920 # disabled because catches gif.exe in incoming mail
disablesid 2002322 # Matched all msn links with php...
disablesid 2002323 #exe file sent via msn
disablesid 2002892 # Virus smtp inbound
disablesid 2002894 # Virus smtp inbound
disablesid 2002895 # Virus smtp outbound, always MX ips
disablesid 2003484 #should be called "malformed http request" instead of virus... triggered by linux machine
disablesid 2007774 # swizzor adware
disablesid 2007866 # gadu-gadu is imo no trojan
disablesid 2008221 #same
disablesid 2008222 #incoming phishing e-mail asprox
disablesid 2008333 # swizzor adware
disablesid 2008411 #mailer the-bat attachment: - useless...
disablesid 2008576 # tinype win executables. remote hosts seem legit.
disablesid 2009345 # 401 unauthorized.
disablesid 2009346 # not a compromized machine. detects http bruteforce (some 401 errors during a short period)

#Disabled because of low complexity, or high FP rate
disablesid 141    # checks for "host" in traffic on 1 port.
disablesid 152    # 3 bytes, 2 ports, lxbuild FP
disablesid 248    #ddos tool checking for > probably rsync data, false positives 08/05
disablesid 1292    #"Volume serial number" 155’000 google results
disablesid 1811    #Many false positives - string "uname" in ssh traffic
disablesid 2123    #cmd.exe banner... triggered by e-mails on MX
disablesid 5321    # Simple TIME protocol to NIST server (should have detected sober)
disablesid 5322    # Simple TIME protocol to NIST server (should have detected sober)
disablesid 5323    # Simple TIME protocol to NIST server (should have detected sober)
disablesid 6031    # FKWP trojan -> Really GRIDFTP
disablesid 6033    # FKWP trojan -> Really GRIDFTP
disablesid 7672        # "connected" on traffic on 1 port
disablesid 7716    # 7715 has been modified to trigger the alert, too much resource consuming
disablesid 8361    #BACKDOOR black curse 4.0 runtime detection - windows trojan - triggered by 3 different linux machines 6 chars out of 6
disablesid 10442   #windows worm detected on linux machine. false positive. only 5 bytes anywhere
disablesid 2000040 # up.exe sent to ftp. FPs
disablesid 2003555 # false positive, linux machine, only 6 bytes
disablesid 2007594 # "User-agent: Mz" -false positives on symantec live update and bbs korean server (mnet.com, cafe.naver.com)
disablesid 2007711 # Only 2 bytes... udp...
disablesid 2007840 # UA: Shell - triggered by MSN shell client
disablesid 2007962 #worse
disablesid 2007963 #idem
disablesid 2007964 #2 bytes OK in 2 bytes packets vipdataend
disablesid 2008056 # idisk.mac.com FP
disablesid 2008103,2008104,2008105,2008106,2008107,2008108,2008109,2008110 # does not rely on content, all packets on 1 port with a specific size
disablesid 2008468,2008469 # LDPinch, too many FPs with known websites
disablesid 2008547 # Trojan Packed Binary - legit download from ftp.clubic triggers alert
disablesid 2009031 #not armitage loader request, but ads.clicmanager.fr...
disablesid 2009292 # c&c response, triggered by axis cameras
disablesid 2009522 # Checkin to fake GIF. too many false positives, long discussion by mail with GoD


#Those are the emule rules that trigger way too many FPs to be useful for CERN when Skype is around.
#Chapter 7 of the thesis explains this in detail.
disablesid 2003316,2003310,2003317,2003308,2003309,2003311,2003312,2003313,2003314,2003315,2003318,2003319,2003320,2003321,2003322,2003323,
disablesid 2003324,2587,2000330,2000332,2000333,2001295,2001296,2001297,2001299
# The only emule rule interesting to us is 2001298
enablesid 2001298

#This is a bittorent rule triggering too many unwanted alerts and dependent on another rule (2181)
disablesid 2000334
#This is a bittorrent rule detecting the same machines with less alerts
enablesid 2181
```

D Improvements to Snort

D.1 Sample Script to Create Mutiple Analyzers

Below is a sample of the configuration file creating two analyzers for a Snort instance.

```
opttab1={
  conf="/opt/snort/etc/snortsp1.conf",
  dynamic_engine_lib="(...)",
  dynamic_preprocessor_lib_dir="(...)",
  l="/opt/snort/log/current"
}
opttab2={
  conf="/opt/snort/etc/snortsp2.conf",
  dynamic_engine_lib="(...)",
  dynamic_preprocessor_lib_dir="(...)",
  l="/opt/snort/log/current"
}
function init ()
  eng.new("e1")
  eng.add_analyzer({
    engine=engine_id,
    analyzer="a1", order=1,
    module=snort_module,
    data=opttab1, bpf=""
  })
  eng.add_analyzer({
    engine=engine_id,
    analyzer="a2", order=2,
    module=snort_module,
    data=opttab2, bpf=""
  })
end
```

Upon initialization, Snort SP creates a new engine, named “e1”, and attaches two analyzers. Each having a separate options array.

D.2 Rule Profiling Sample Report

The following report was obtained by deploying all the P2P rules for a few minutes. The two first Emule rules (that did not trigger any alert), consumed more resources that all the other rules combined.

```
Num      SID GID     Checks   Matches     Alerts           Microsecs  Avg/Check  Avg/Match Avg/Nonmatch
===      === ===     ======   =======     ======               =====  =========  ========= ============
  1  2003322   1   66774824         0          0            10767116        0.2        0.0          0.2          0
  2  2003321   1   66774824         0          0             9232964        0.1        0.0          0.1          0
  3     5999   1    7166994       346          0              803038        0.1     1380.8          0.0          0
  4  2008595   1    4240390         0          0              539240        0.1        0.0          0.1          0
  5  2006379   1      37844         0          0               41184        1.1        0.0          1.1          0
  6     5998   1     395476       153          0               38001        0.1        0.1          0.1          0
  7  2003310   1     104496        25          0               35600        0.3        0.0          0.3          0
  8  2003320   1     112767        22          0               33693        0.3        0.0          0.3          0
  9  2003317   1     101584        28          0               33141        0.3        0.0          0.3          0
 10  2003319   1     102202         0          0               32280        0.3        0.0          0.3          0
 11  2003323   1     130301         0          0               28617        0.2        0.0          0.2          0
 12  2003313   1     105045         8          0               28320        0.3        0.0          0.3          0
 13  2003315   1     110124         7          0               27923        0.3        0.1          0.3          0
 14  2009098   1     105197         0          0               14819        0.1        0.0          0.1          0
 15  2003308   1     107289         0          0               13289        0.1        0.0          0.1          0
 16  2009099   1     211941         0          0               13223        0.1        0.0          0.1          0
 17  2003309   1     107598         0          0               11093        0.1        0.0          0.1          0
 18  2003318   1     104127         0          0                8458        0.1        0.0          0.1          0
 19  2003311   1     104130         3          0                8367        0.1        0.0          0.1          0
 20  2003316   1     104510         1          0                7710        0.1        0.0          0.1          0
 21  2003312   1     100854         0          0                6563        0.1        0.0          0.1          0
 22  2002814   1       9339         0          0                5774        0.6        0.0          0.6          0
 23     5693   1       3157       120          0                4102        1.3        0.1          1.3          0
 24  2000333   1      38105         0          0                2080        0.1        0.0          0.1          0
 25  2000332   1      38105         0          0                2080        0.1        0.0          0.1          0
 26  2008581   1       1304      1286          0                1761        1.4        0.0         95.6          0
 27  2008583   1        647         3          0                1615        2.5        0.1          2.5          0
 28     3680   1       1218         0          0                1079        0.9        0.0          0.9          0
 29  2001185   1       1197         0          0                 981        0.8        0.0          0.8          0
 30  2008582   1       1291         0          0                 954        0.7        0.0          0.7          0
 31    12691   1        211         0          0                 514        2.4        0.0          2.4          0
 32  2000340   1        421         0          0                 221        0.5        0.0          0.5          0
 33    12211   1        379         0          0                 188        0.5        0.0          0.5          0
 34  2007727   1         84         2          0                 159        1.9        0.1          1.9          0
 35  2001299   1        310         0          0                 114        0.4        0.0          0.4          0
```

E Tools

E.1 Python Script Used to Compute a Rule Score

This script computes a score for each rule contained in the file passed as first argument. A rule will be placed either in keep.rules or in dismiss.rules, depending on a fixed threshold.

In the below version, only the content is used to compute a general score. This is very easily modifiable with the different weight on the first lines.

```
import re
import os
import sys

# Regexp for a one-line rule
ruleline = re.compile('^alert (?P<proto>\S+) (?P<src>\S+) (?P<srcport>\S+) (?P<way>\S+) (?P<dst>\S+) (?P<dstport>\S+) (?P<payload>[^\n]+)')

# We create two output files. One where the kept rules will be put, and one where the dismissed rules will be put
of = open("keep.rules", 'a')
dismiss = open("dismiss.rules", 'a')

# For each of the input file (given as argument), if it is a line
for line in open(sys.argv[1], "r"):
   try:
        m = ruleline.match(line)
        if m is not None:

          # Here we define the weights of the different function parameters.
          # The final score will be the sum of all these weight multiplied by their respective scores

          # In this version, only the rule content is used
          weightClasstype= 0
          weightPacketSize= 0
          weightNumberOfPorts= 0
          weightContent= 1
          weightIPs= 0
          weightFlowBits = 0
          weightMessage = 0
          weightPcre = 0

          # The scores that will multiply with the weights (above). All initialized to 0.
          scoreNumberOfPorts = 0   #from 0 (all ports) to 1 (1 port)
          scoreContent = 0    #from 0 when no content checked to 20 depending on the sum of bytes + bytes^2/depth, \
          #=2 if 1 byte is checked, =2, if 1 byte at specific position, =10 if 10 bytes, =20 if 10 bytes at specific position
          scorePacketSize = 0      #from 0 (no size) to 1 (exact size)
          scoreIPs = 0        #unimplemented yet
          scoreClasstype = 0  #remove unwanted classtypes with negative score
          scoreFlowBits = 0   #Increase confidence in rules having flowbit set
          scoreMessage = 0    #Reduce confidence when the message contains some keywords
          scorePcre = 0       #Increase confidence when a rule has a PCRE

          #Split the rule's payload to get individual elements
          payload = (m.group("payload")).replace(" ","").split(';')

          # We start computing scores in the order proposed above

          # Number of ports. This one was the most difficult to implement due to the large variety of possiblities
          # in specifying ports. It can be a single port (80), a port range (80:100), an unbounded port range (80:),
          # a list of ports (80,81), a combination of the above (80-,22), or a variable ($HTTP_PORTS), and all the above negated (!80:100)
          # The coding is not very nice, because each of the above case was added one by one... But it works.

          if m.group('srcport') == "any" and m.group('dstport') == "any":
               scoreNumberOfPorts = 0;
               port = "any"
          elif m.group('srcport') != "any":
               port = m.group('srcport')
          elif m.group('dstport') != "any":
               port = m.group('dstport')
          if port.find("!") != -1:
               isReversed = 1
          else:
               isReversed = 0
          if re.match('^[\d]+$',port)!=None:
               scoreNumberOfPorts = 1
          elif port.find(",") != -1:
               port = port.replace("[","").replace("]","")
               nbport=0
               ports = port.split(",")
               for p in ports:
                    if p.find(":") != -1:
                         p2 = p.split(":")
                         #because rules can be written with :1024 instead of 1:1024
                         if p2[0] == "":
                              p2[0] = 1
                         if p2[1] == "":
                              p2[1] = 65535
                         if not isReversed:
                              nbport += (int(p2[1]) - int(p2[0]))
                         else:
                              nbport += 65535-(int(p2[1]) - int(p2[0].strip("!")))
                    elif re.match('^[\d]+$',p)!=None:
                         nbport += 1

               scoreNumberOfPorts = (65535.0 - nbport) / 65535.0
```

```
elif port.find(":") != -1:
     port = port.replace("[","").replace("]","")
     ports = port.split(":")
     #because rules can be written with :1024 instead of 1:1024
     if ports[0] == "":
          ports[0] = 1
     if ports[1] == "":
          ports[1] = 65535
     if isReversed:
          scoreNumberOfPorts = (int(ports[1]) - int(ports[0].strip("!"))) / 65535.0
     else:
          scoreNumberOfPorts = (65535 - (int(ports[1]) - int(ports[0]))) / 65535.0
elif port.find("$") != -1:#port is a variable from config file, usually HTTP_SERVERS, will round the value to 5 ports
     scoreNumberOfPorts = (65535 - 5.0) / 65535.0
print "score ports is ", scoreNumberOfPorts, "(*", weightNumberOfPorts,")"


# The score is then computer for the packet size, which is given by: (1518.0-(BIGGEST-SMALLEST))/1518.0;
# This gives a score of 1 if the port is specific, 0 if any.

for word in payload:
     if word.find("dsize:") != -1:#packet has a size
          word = word.split(":")[1]
          if word.find("<>") != -1:
               numbers = word.split("<>");
               scorePacketSize = ( 1518.0 - (int(numbers[1])-int(numbers[0])) ) / 1518.0;
          elif word.find("<") != -1:
               scorePacketSize = ( 1518.0 - int(word.strip("<")) ) / 1518.0;
          elif word.find(">") != -1:
               scorePacketSize = ( 1518.0 - (1518 - int(word.strip(">")))) / 1518.0;
          else:
               scorePacketSize = 1;
          break;
print "score size is", scorePacketSize, "(*",weightPacketSize,")"

# The PCRE score. set to 1 if a PCRE is found.
for word in payload:
     if word.find("pcre:") != -1:
          scorePcre = 1
print "score pcre is",scorePcre*weightPcre

# The Flowbits score, set to 1 if a flowbit is read.
for word in payload:
     if word.find("flowbits:") != -1:
          flowbits = word.split(":")[1].strip("\"")
          if flowbits.find("isset,") != -1:
               scoreFlowBits = 1
print "score flowbits is", scoreFlowBits*weightFlowBits

# The Message score, if "POLICY" is found then the score is set to 1
for word in payload:
     if word.find("msg:") != -1:
          msg = word.split(":")[1].strip("\"")
          if msg.find("POLICY") != -1:
               scoreMessage = 1
print "score message is", scoreMessage*weightMessage

# And now the big part, the content score. There can be many content statements, each can be followed by a depth.
# The total score is given by:  Sum (  # bytes checked +  (# bytes checked ^ 2 / depth) ). If 10 bytes are to be found anywhere
# in the packet, the score will be 10, if those 10 bytes are at an exact position, the score will be 20.
# The score is maxed to 20 at the end.

hasContent = False
length = 0.0
for word in payload:
     if word.find("content:") != -1:
          #not the first content. If true, then some content already encountered but not summed, so we sum it with no depth.
          if hasContent:
               scoreContent = scoreContent + length
               hasContent = False

          content = word.split(":",1)[1].strip("\"").replace("\\","")
          length = 0.0
          bytes = False #0 = char, 1 = bytes
          for char in content:
               if char is "|":
                    bytes = not bytes
                    continue
               if bytes:
                    length = length + 0.5
               else:
                    length = length + 1.0
          hasContent = True
     elif word.find("depth:") != -1 and hasContent:
          hasContent = False
          depth = word.split(":")[1]
          scoreContent = scoreContent + length + length * length / int(depth)

if hasContent:
     scoreContent = scoreContent + length
     hasContent = False

print "Score total content is", scoreContent,
if scoreContent > 20:
     scoreContent=20
     print "maxing to 20",
print
```

```
          # And now the Classtype score. If it is mentioned below, the score will be set to 1
          for word in payload:
               if word.find("classtype:") != -1:
                    classtype = word.split(":",1)[1]
                    if classtype.find("web-application-activity") != -1:
                         scoreClasstype = 1
                    elif classtype.find("misc-activity") != -1:
                         scoreClasstype = 1
                    elif classtype.find("web-application-attack") != -1:
                         scoreClasstype = 1
                    elif classtype.find("not-suspicious") != -1:
                         scoreClasstype = 1
                    elif classtype.find("icmp-event") != -1:
                         scoreClasstype = 1
                    elif classtype.find("suspicious-filename-detect") != -1:
                         scoreClasstype = 1
          print "score Classtype is", scoreClasstype


          # We get the SID of the current rule
          sid = 0
          for word in payload:
               if word.find("sid:") != -1:
                    sid = word.split(":",1)[1]


          # We compute the rule final score
          totalScore = weightNumberOfPorts * scoreNumberOfPorts + weightPacketSize * scorePacketSize + weightContent * scoreContent \
          + weightIPs * scoreIPs + weightClasstype * scoreClasstype + weightFlowBits * scoreFlowBits + scoreMessage * weightMessage \
          + weightPcre * scorePcre


          # We add the score to the rule message for convenience
          print "Total Score of the rule is", totalScore
          line = line.replace("msg:\"","msg:\" "+str(totalScore)+" ")

          # We place the rule in one or the other file depending on the score.
          # In this version, the limit is set to 7, so that 3 bytes at specific position are dismissed, but 4 kept.
          if totalScore >= 7:
               of.write("#"+str(totalScore)+"\n"+line)
          else:
               dismiss.write("#"+str(totalScore)+"\n"+line)
   except Exception:
     print "ErrorLine",line
of.close()
dismiss.close()
```

E.2 Plot the Rules Dependencies

E.2.1 Script

This scripts opens an alert file. Analyzes all the alerts, and if some rule depends on another links them on the produced graph.

```
import re
import os
import gv
import sys


# Min number of IPs for considering the set when plotting the dependency graph.
# Explained in detail below
MIN_THRESHOLD = 2


# These are the two regexp to match alerts. The first one extracts the SID and message, and the second one the IPs, time and ports.

snortline1 = re.compile('^\[\*\*\] \[(?P<what>\d+):(?P<sid>\d+):(?P<rev>\d+)\] (?P<descr>.+) \[\*\*\]$')
snortline2 = re.compile('^\d{2}.\d{2}.\d{2}-\d{2}:\d{2}:\d{2}.\d{6} (?P<ip1>[\d\.]+):(?P<port1>\d+) -> (?P<ip2>[\d\.]+):(?P<port2>\d+)$')

# Various data structures to store alerts, sorted by IP or by SID

# Number of alerts per SID
count = dict() # count[sid]

# Stores which SIDs triggered alerts for a given IP
perip = dict() #

# Stores which IPs triggered alerts for a given SID
persid = dict()

#
name = dict()

# Opens the alerts file
for line in open("alert.current", "r"):
     m = snortline1.match(line)
     if m is not None:
          sid = m.group('sid')
          rev = m.group('rev')
          descr =  m.group('descr')
     else:
          m = snortline2.match(line)
          if m is not None:

               # At this point, we have the two interesting lines for an alert, and therefore all the necessary data.


               # We are only interested in IPs belonging to the CERN.
               ip1 = m.group('ip1')
               if not ip1.startswith('128.141.') and not ip1.startswith('137.138.') and not ip1.startswith('128.142.'):
                    ip1 = -1
               ip2 = m.group('ip2')
               if not ip2.startswith('128.141.') and not ip2.startswith('137.138.') and not ip2.startswith('128.142.'):
                    ip2 = -1

               port1 = m.group('port1')
               port2 = m.group('port2')
               name[sid] = descr

               # All the variables are set, we can use them !

               # We start by incrementing the counter for the current SID
               if sid not in count:
                    count[sid] = 0
               count[sid] = count[sid]+1;

               # And for each ip, we store which sids are active
               if ip1 != -1:
                    if ip1 not in perip:
                         perip[ip1] = set()
                    perip[ip1].add(sid)
               if ip2 != -1:
                    if ip2 not in perip:
                         perip[ip2] = set()
                    perip[ip2].add(sid)

               #and for each rule, we store the ips
               if sid not in persid:
                    persid[sid] = set()
               if ip1 != -1:
                    persid[sid].add(ip1)
               if ip2 != -1:
                    persid[sid].add(ip2)



# We print the ordered number of alerts per SID
pairs = [(v, k) for (k, v) in count.iteritems()]
pairs.sort()
highestcount = 0
for (k,v) in pairs:
     print k,v,name[v]
     highestcount = k



# This can be used to print all SIDs that each IP triggered
#keys = perip.keys()
#keys.sort()
#for key in keys:
#    print key,': ',
#    v = sorted(perip[key])
#    for v2 in v:
#         print v2,
#    print

# We print the number of IPs triggered by each SID
# For some specific SIDs, the list of IPs can be printed. In this case only 2001298 will print details.
print 'Printing number of ips per snort id :'
keys = persid.keys()
keys.sort()
for key in keys:
     print key,': ',len(persid[key])
     if key == '2001298':
          v = sorted(persid[key])
          for v2 in v:
               print v2,',',
          print


# And now the big plotting part:
# We get the active SIDs
keys = persid.keys()
keys.sort()

# We setup the plot
G = gv.digraph('G')
N = gv.protonode(G)
E = gv.protoedge(G)

gv.setv(G, 'rankdir', 'LR')
gv.setv(G, 'center', 'true')
gv.setv(G, 'nodesep', '0.05')
gv.setv(N, 'shape', 'box')
gv.setv(N, 'width', '0')
gv.setv(N, 'height', '0')
gv.setv(N, 'margin', '.03')
gv.setv(N, 'fontsize', '8')
gv.setv(N, 'fontname', 'helvetica')
gv.setv(E, 'arrowsize', '.4')

nodes = {}

# We check all SID against each other.
# If a set of IPs (for a given SID) is a subset of the other (for the other SID)
# We create a dependency between the two

# This is valid only if there are at least MIN_THRESHOLD IPs. If there are less then we do not plot.

for leftkey in keys:
     left = persid[leftkey]
     for rightkey in keys:
          right = persid[rightkey]
          if right <= left and rightkey != leftkey and len(right)>=MIN_THRESHOLD:
               nodeName = leftkey
               nodeName += name[leftkey]#leftkey
               nodeName += '--'
               nodeName += str(count[leftkey])
               n = gv.node(G, nodeName)
               nodeName = rightkey
               nodeName += name[rightkey]#rightkey
               nodeName += '--'
               nodeName += str(count[rightkey])
               n2 = gv.node(G, nodeName)
               gv.edge(n, n2)

# We save the file as dev.jpg
gv.layout(G,'dot')
gv.render(G,'jpeg','dev.jpg')
```

E.2.2 Sample result

Figure 7 presents a sample plot produced by the above script on P2P rules with CERN alerts. In this example we can clearly see that on that day Skype rule *5998* is sufficient to detect all Skype instances, as all other Skype rules depend on it. The same holds for bittorrent traffic, where the set produced by the 2655 alerts of rule *2181* was more complete than the one produced by the 57452 alerts of rule *2000334*. We also see that *2008581* is the best choice to detect bittorrent's DHT. The number on the right of the cell is the number of alerts counted for every SID. There are a few cells that do not have enough data to produce usable results.

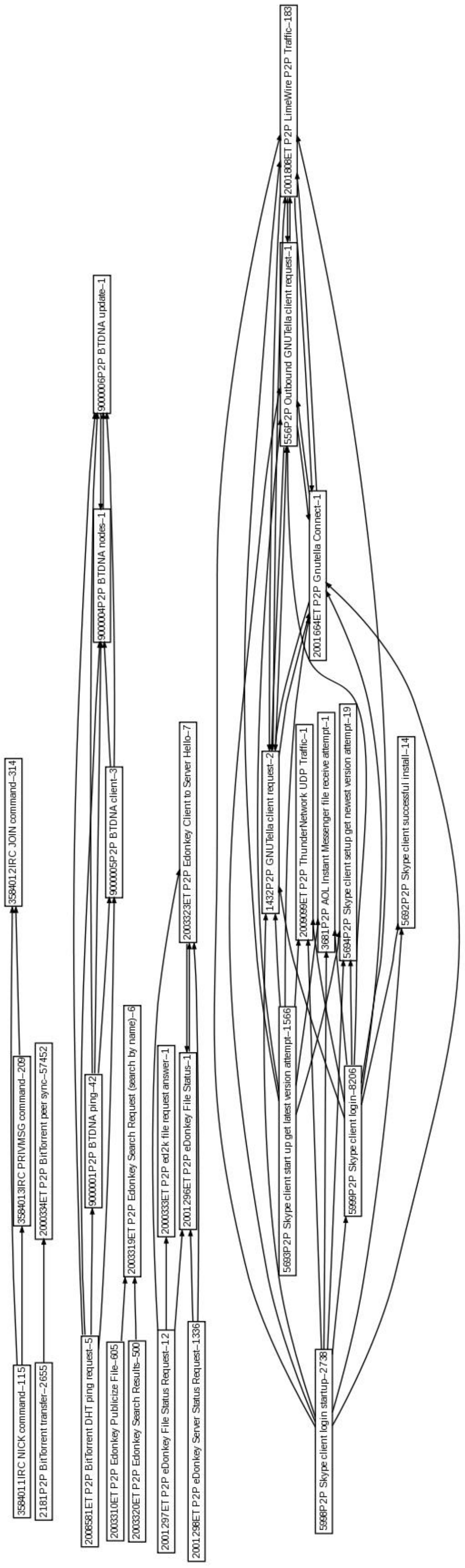
3584011IRC NICK command–115
2181P2P BitTorrent transfer–2655
3584013IRC PRIVMSG command–209
2000334ET P2P BitTorrent peer sync–57452
3584012IRC JOIN command–314
2008581ET P2P BitTorrent DHT ping request–5
9000001P2P BTDNA ping–42
9000005P2P BTDNA client–3
9000004P2P BTDNA nodes–1
9000006P2P BTDNA update–1
2003310ET P2P Edonkey Publicize File–605
2003320ET P2P Edonkey Search Results–500
2003319ET P2P Edonkey Search Request (search by name)–6
2001297ET P2P eDonkey File Status Request–12
2000333ET P2P ed2k file request answer–1
2001298ET P2P eDonkey Server Status Request–1336
2001296ET P2P eDonkey File Status–1
2003323ET P2P Edonkey Client to Server Hello–7
5998P2P Skype client login startup–2738
5693P2P Skype client start up get latest version attempt–1566
5999P2P Skype client login–8206
1432P2P GNUTella client request–2
2009099ET P2P ThunderNetwork UDP Traffic–1
3681P2P AOL Instant Messenger file receive attempt–1
5694P2P Skype client setup get newest version attempt–19
5692P2P Skype client successful install–14
2001664ET P2P Gnutella Connect–1
556P2P Outbound GNUTella client request–1
2001808ET P2P LimeWire P2P Traffic–183

Fig. 7: Dependencies between rules

References

[1] "CERNs mission." http://public.web.cern.ch/public/en/About/Mission-en.html, cited June 2009.

[2] "Intrusion detection system." http://en.wikipedia.org/wiki/Intrusion_detection_system, cited June 2009.

[3] "Intrusion Detection Systems (IDS)." http://www.windowsecurity.com/articles/Intrusion_Detection_Systems_IDS_Part_I__network_intrusions_attack_symptoms_IDS_tasks_and_IDS_architecture.html, cited June 2009.

[4] "OSI model." http://en.wikipedia.org/wiki/OSI_model, cited August 2009.

[5] Y.-T. Chan, C. Shoniregun, and G. Akmayeva, "A netflow based internet-worm detecting system in large network," in *Digital Information Management, 2008. ICDIM 2008. Third International Conference on*, pp. 581–586, Nov. 2008.

[6] "Snort homepage." http://www.snort.org, cited July 2009.

[7] "The Programming Language Lua." http://www.lua.org/about.html, cited June 2009.

[8] "Snort 3.0 Architecture Series Part 2: Changes and Betas." http://securitysauce.blogspot.com/2008/08/snort-30-architecture-series-part-2.html, cited August 2009.

[9] P. Garcia-Teodoroa, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, pp. 18–28, 2009.

[10] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 4, pp. 56–76, 2008.

[11] A. A. Papaioannou, "Non-convex Neyman-Pearson classification," Master's thesis, École Polytechnique Fédérale de Lausanne, 2009.

[12] "Sans internet storm center; cooperative network security community - internet security." http://isc.sans.org/, cited july 2009.

[13] "Snort Users Manual." http://www.snort.org/assets/82/snort_manual.pdf, April 2009.

[14] L. Etienne, "A short Snort rulesets analysis," tech. rep., CERN CERT, 2009.

[15] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing skype traffic: when randomness plays with you," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 37–48, 2007.

[16] S. A. Baset and H. G. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–11, 2006.

[17] E. Freire, A. Ziviani, and R. Salles, "Detecting skype flows in web traffic," in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pp. 89–96, April 2008.

[18] F. D. P. Biondi, "Silver Needle in the Skype." Black Hat Europe'06, Amsterdam, the Netherlands, Mar. 2006.

[19] D. B. Y. Kulbak, "The eMule Protocol Specification." DANSS, Hebrew University of Jerusalem, Jan. 2005.

[20] "Oinkmaster." http://oinkmaster.sourceforge.net, cited June 2009.

[21] "Dumbpig - Automated checking for Snort rulesets." http://leonward.wordpress.com/2009/06/07/dumbpig-automated-checking-for-snort-rulesets/, cited July 2009.

[22] "Pulled Pork." http://code.google.com/p/pulledpork/, cited July 2009.

[23] "tcpdump/libpcap public repository." http://www.tcpdump.org/, cited August 2009.

[24] "Phil Wood's libpcap." http://public.lanl.gov/cpw/, cited May 2009.

[25] "Scientific Linux CERN 5." http://linux.web.cern.ch/linux/scientific5/, cited May 2009.

[26] "Comparing Linux Compilers." http://www.coyotegulch.com/reviews/linux_compilers/index.html, cited August 2009.

[27] "CERN in a nutshell." http://public.web.cern.ch/public/en/About/About-en.html, cited June 2009.

[28] W. Zhenqi and W. Xinyu, "Netflow based intrusion detection system," *MultiMedia and Information Technology, International Conference on*, vol. 0, pp. 825–828, 2008.