Federal University of Rio Grande do Sul
Institute of Informatics

Pedro Egidio Menegaz Paganela

Study and Development of a Network Based Intrusion Detection System

Computer Engineering Graduation Work

Prof. Dr. Alexandre Carissimi
University Advisor

Porto Alegre, December 2011.

ACKNOWLEDGMENTS

I would like to thank my advisor Alexandre Carissimi, for all support and patience in this project, Michael Rigoni that guided me in the development over the Mancala Network Controller and Sérgio Cechin for being always helpful when needed. Finally, I would like to thank all the people from UFRGS that, directly or indirectly, helped me to arrive where I am now.

SUMMARY

ABSTRACT

In the 90's the majority of threats performed against computer systems were made by teenagers influenced by their curiosity. They had as objective just to show their capabilities exploiting computer systems. As good example from this time, we can take the famous Hacker Kevin Mitnick [KEVIN BIO, 1996]. However, in the last years, computer systems are facing a new type of threats, where in general they are more organized, sophisticated and with devastating consequences [REUTERS, 2007] [MARKOFF, 2009][STUXNET, 2011] .

Since that the threats in the last years are becoming more complex, simple Firewalls are not enough to secure a computer network system. For this reason, Intrusion Detection Systems [SCARFONE ET AL., 2007] are becoming more and more used by corporations, as an extra protection against these threats.

In this paper we will present a study about a new Network Based Intrusion Detection System (NIDS) solution that improves the detecting capabilities of the principal NIDS solution in the market [SNORT, 2011]. This solution uses known and new detection methodologies developed in this work. Finally, the developed NIDS will be integrated to the Mancala Network Controller Framework [MANCALA, 2011], to be used as a security module to network corporations.

Keywords: Network Security, System Security, NIDS, IDS, threat, vulnerability, Mancala Network Controller

RESUMO

Na década de 90, a maioria das ameaças contra sistemas de informação eram executadas por jovens influenciados por sua curiosidade. Esses jovens tinham como objetivo somente mostrar suas capacidades frente a os sistemas de computação de sua época. Como um exemplo deste tempo, podemos pegar o livro que conta a história do famoso hacker Kevin Mitnick [KEVIN BIO, 1996]. Todavia, nos últimos anos os sistemas de informação estão se tornando alvo de ameaças mais sofisticadas, organizadas e normalmente com consequências devastadoras [REUTERS, 2007] [MARKOFF, 2009][STUXNET, 2011] .

Desde que essas novas ameaças estão se tornando mais complexas, simples firewalls não são suficientes para proteger os sistemas de informação hoje em dia. Por essa razão, Sistemas de Detecção de Intrusão [SCARFONE ET AL., 2007] estão se tornando cada vez mais usados em sistemas de corporações, sendo uma proteção extra contra essas ameaças.

Neste artigo será apresentado o estudo e o desenvolvimento de uma nova solução de Sistema de Detecção de Intrusão baseada no tráfico da rede (NIDS). O NIDS desenvolvido aumenta a capacidade de detecção do principal NIDS do mercado [SNORT, 2011], usando métodos de detecção conhecidos e também novos, desenvolvidos nesse trabalho. Finalmente, o NIDS desenvolvido será integrado com o framework Mancala Network Controller [MANCALA, 2011], sendo usado como um módulo de segurança para redes de empresas.

Palavras-chave: Segurança de Redes, Segurança de Sistemas, NIDS, IDS, vulnerabilidades, Mancala Network Controller

1. INTRODUCTION

Twenty years ago, few companies operated with an internal computer network. Today, in the Internet age, these structures are necessaries to the majority of corporations. While these networks are a need, networks grow in complexity, attacks are also getting more specialized [STUXNET, 2011][ZEUS, 2011]. For these reasons, the corporation system security should not be linked only to a simple solution, like a firewall [NETFILTER, 2011], but it should also use other security methods, as detection systems.

One of the most popular approaches related with threat detection in computer networks is the Network Intrusion Detection Systems (NIDS) [SCARFONE ET AL., 2007]. The general functionality of an NIDS is, based in the information collected from the networks' traffic, detect if a possible threat is happening.

1.1 Objectives

The objective of this final project is, based in the Mancala Network Controller Framework [MANCALA, 2011], to develop a more powerful NIDS solution that using new approaches, it detects threats that were not detected before by the principal NIDS solution in the market, Snort [SNORT, 2011].

1.2 Organization

This work is divided in 6 chapters, including this one. The next chapter (chapter 2), we will present the NIDS solutions state of the art, including a brief study in commercial NIDS solutions and what can be improved. In the chapter 3, we will present the threats detected by our IDS solution. In the chapter 4, we will present a brief explanation about the NIDS architecture and how it was integrated with the Mancala Network Controller Framework. In the chapter 5, we will present the evaluation method used and how we proven that our solution works. Finally, in the chapter 6, we will finish with a small conclusion about this work and possible future improvements to the solution.

2. STATE OF THE ART

The majority of threats that we find in the wild use as vector computer networks [SHNEIER, 2011][MARKOFF, 2009][LULZSEC, 2011]. For this reason, an approach that is being more and more used is the Network Intrusion Detection System (NIDS). Exist many solutions in the wild and to understand which one is the state of the art, we will analyze the principal ones. The analysis will be divided in 2 groups, the open source solutions and the commercial solutions.

There are not many known open source NIDS solutions, but it does not mean that they are not good solutions. The open sources solutions studied here are Snort [SNORT, 2011] in the chapter 2.1 and Suricata [OISF, 2011] in the chapter 2.2. However, There are numerous commercial solutions presented by different vendors, as for example: Cisco [CISCO, 2011], IBM [IBM, 2011], McAffe [MCAFEE, 2011], SourceFire [SOURCEFIRE, 2011], CheckPoint [CHECKPOINT, 2011], etc.

In the study of the commercial solutions, since that the student does not have access to the full NIDS commercial frameworks, a basic analysis based in manuals, demos and even marketing videos was performed by the student over solutions of important vendors of the security area. Also, it is important to highlight that all the studied solutions supports IPS (Intrusion Prevention System), for this reason they are called exclusively IPS.

2.1 Snort

Snort [SNORT, 2011] is the baseline when we talk about NIDS. The snort architecture, in a simplified point of view, can be resumed in the figure 2.1. As can be seen in the figure 2.1, the first step in the packet analysis is the capture of it. This action is performed mainly by PCAP and NFQ modules.

The snort architecture, in a simplified point of view, can be resumed in the figure 2.1. As can be seen in the figure 2.1, the first step in the packet analysis is the capture of it. This action is performed mainly by PCAP and NFQ modules.

PCAP is a API for capturing network traffic developed by the Tcpdump Team [TCPDUMP, 2011] to linux. This is the default mode when we are working in NIDS mode.

NFQ is a framework that provides packet manipulation in real time in the kernel level at Linux systems. Netfilter Queue [NETFILTER, 2011] is a Netfilter module that captures packets in real time in the kernel level and sends to the user level to be analyzed and judged. This is the most used mode to analyze packets in Intrusion Prevention System mode at a Linux System.

After the packet capture, the packet will pass through the preprocessor engine to prepare it for the rule engine (like gathering fragmented packets, organizing them by sessions, etc).

This engine is formed, as the name indicates, by a group of different preprocessors. Each preprocessor have a specific functionality, many times related with the organization/classification of a packet (e.g. sessions, fragmented packets), or even features related with stateful analysis (e.g. port scans). Below we can see a list with the main preprocessors presented in the Snort 2.9.0.4 analyzed in this study:

- Frag3: treat fragmented packets (avoid evasion attacks)
- Stream5: organize the packets by sessions (including UDP "sessions") [RFC 768, 1980]
- sfPortScan: detect TCP port/sweep scans and badly UDP scans [RFC 793, 1981]
- HTTP inspect: detect some invalid/malicious HTTP traffic
- ARP spoof preprocessor: a bad ARP spoof detector [RFC 826, 1982]
- Sensitive Data preprocessor

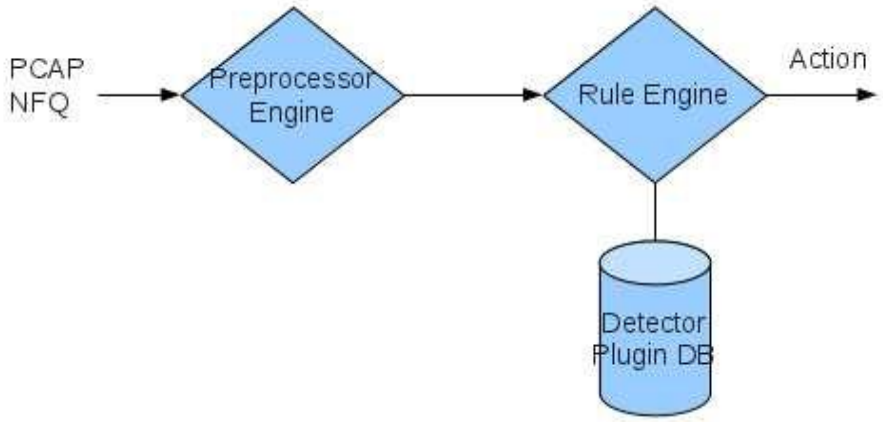

Figure 2.1 – Simplified Snort architecture.

Finally, in the last part of the packet processor, we have the Rule Engine, that is the main module inside the architecture. This engine triggers actions based on conditions. A rule is formed by the follow structure:

```
<action> <ip(s) source> <port(s) source> -> <ip(s) destination> <port(s) destination> (<rule options> ; sid :<unique ID from this rule>;)
```

IP and port fields are self-explanatory. Action field is where the action trigged by the rule is defined. The possible actions in NIDS mode is Alert, log and pass. Since the scope of this work is the detection, not the prevention, we will not detail the IPS action modes.

The rule options can be as variable as possible and many times we use Detection Plugins in the rule options to detect more complex, and sometimes stateful events. A list of the main supported detection plugin options:

- Raw Content modifiers: search for a bytes' pattern in the packet.
- HTTP Content Modifiers: search for patterns in HTTP fields [RFC 2616, 1999].
- PCRE: content modifier to support regular expressions.

Also, there are some options not direct related with payloads that can be considered metadata. A list of them can be found below:

- msg: defines the message related with an alert action.
- sid: unique ID of a rule.
- rev: revision number.
- flow: flow direction (e.g. to server).
- reference: a reference to the threat/detection.
- threshold: see below.

Threshold is a special option. It gives a stateful capability to the rule. In this condition you need to pass 4 fields: type (limit, threshold or both), track (by source or destination) count and seconds. This option will be satisfied when the number of times that this alert (based in a track) is satisfied is equal or bigger to count in an interval define by seconds. Also, type defined is the alerts will be triggered until the threshold is satisfied or if it will only be alerted when the threshold is satisfied. In the figure 2.2 we can see a good example of rule.

```
alert udp $HOME_NET 53 -> $EXTERNAL_NET any (msg:"DNS dns response for rfc1918 10/8 address detected"; \
flow:from_server; content:"|00 01 00 01|"; content:"|00 04 0A|"; within:3; distance:4; fast_pattern; \
metadata:policy security-ips alert, service dns; reference:url,www.faqs.org/rfcs/rfc1918.html; \
classtype:policy-violation; sid:13249; rev:3;)
```

Figure 2.2: Rule to detect DNS Internal IP Leak

In this example, if a DNS packet that comes from the internal network is not resolving a DNS query to an internal network IP in the range 10.0.0.0/8. This alert added with other 2 similar alerts related with internal network IPs (192.168/16 and 172.16/14) are able to detect DNS Internal IP Leakage.

2.2.1 Snort Detection Capabilities

The company Source Fire [SOURCEFIRE, 2011], that founded Snort and is the main signatures supporter, giving a big rules set that is frequently refreshed. These rules can be divided in groups of detection threats, as for example: FTP rules, Exploit rules, P2P rules, C&C Botnet rules, etc. Also, there is the open source group called Emerging Threats [EMERGING THREATS, 2011], also known as ET, that has a big set of rules available for free, with a big support from the Snort enthusiastic community. Snort focuses its detection capability in the follow topics: remote application exploitation detection, malicious software and address detection.

"An application exploit is a piece of software, a chunk of data, or sequence of commands that takes advantage of a vulnerability in order to cause unintended

or unanticipated behavior to occur on computer software" This sentence was taken informally from the Wikipedia [WIKIPEDIA, 2011] exploit page and resumes the meaning of remote application exploitation.

There are numerous methods that can be used to exploit an application; some classical examples are listed below:

- Buffer Overflow
  - Stack based
  - Heap based
- angling pointers
- Uncontrolled format string
- Integer Overflow

The detection of remote application exploitation is the strongest characteristic of Snort that is able to detect numerous threats of this genre. Also, their signatures have a constant refresh, making possible to detect new known vulnerabilities.

Another point that looked relatively strong in Snort is the detection of malicious software and malicious address (mainly by rules from ET). Malicious software is detected by the traffic that it generates. Malicious software is a program with a harmful and undesired behavior. Normally the victim that is running this software does not know of its existence and/or does not want to run this software. This kind of software is normally design to illegal purposes. Also, the definition from Wikipedia to Malware: "Malware, short for malicious software, consists of programming (code, scripts, active content, and other software) designed to disrupt or deny operation, gather information that leads to loss of privacy or exploitation, gain unauthorized access to system resources and other abusive behavior".

To the detection of malicious software, Emerging Threats publishes signatures to many malicious programs traffic. These signatures are based in any specific traffic that normally only this Malware generates. Sometimes, it can give false positives (e.g. Malware does a HTTP GET with the content owned), but if we have multiple alerts of the same Malware signatures, it pretty much means that it is actually the Malware.

Addresses are defined as malicious when they are related with suspicious or malicious behavior in the network. The main detection method is based in to know IPs, networks or URLs that are associated with malicious traffic. The main sources of malicious addresses used by ET are:

- Malicious addresses by DSHIELD [DSHIELD, 2011]
- Russian Business network hosts
- C&C servers addresses by ShadowServer [SHADOWSERVER, 2011]
- Malicious addresses by C.I. Army [CI ARMY, 2011]
- ET compromised addresses

DSHIELD website classifies periodically the networks that present a high malicious traffic frequency. The addresses are updated regularly. Russian business networks are famous in the security domain for being source of malicious acts, mainly related with

Botnets, for this reason was added in the malicious addresses detection. ShadowServer keeps a list of many Botnet C&C servers, based in their studies. C.I. Army uses a methodology based in distributed sentinels that study device behaviors in the network and judge if an IP is doing some malicious activity. Finally there are many known compromised hosts in the network that Emerging Threats keeps a list of them.

Also, it is important to highlight that Snort does not only detect these types of threats, but it certainly has focus in these types of threats, mainly in remote application exploitation detection, that is the strongest point in Snort.

2.2 Suricata

Suricata is an open source NIDS developed by the Open Information Security Foundation (OISF) [OISF, 2011]. Its development started in 2009, but it is experimenting a fast growing and soon can become a real Open Source option. OISF defends Suricata saying that it is the next generation of NIDS. However what this study saw is that its structure is incredibly similar to Snort, but with less features. One important advantage is that it is multi-threaded, while Snort engine doesn't support threads. Suricata supports almost all Snort rules and present a good HTTP engine (this engine makes easier create rules to detect behaviors over HTTP packets). Even being able to perform a good part of the Snort functions, it is still too young, while Snort is mature and well accepted by the security community.

2.3 SourceFire IPSx

The first commercial product presented in this chapter, this solution from SourceFire is called IPSx. The engine used by this system is Snort, including the rules developed by a team from the company SourceFire, called VRT [VRT, 2011]. This NIDS also includes an easy to use graphical interface that automates the control of many structures of an NIDS solution: sensors configuration, reports about attacks or hosts and email alerts. As a conclusion, this NIDS is Snort with a beautiful graphical interface to manage some parts of the NIDS and create reports.

2.4 SourceFire NGIPS

NGIPS is a more robust NIDS solution from SourceFire. As the IPSx solution, it uses Snort as the NIDS Engine and it has a good graphical interface to manage the solution, but it also has other characteristics. It has a host profiling based detection system, in other words, it applies a specific rules' set to a type of host automatically.

A feature in special makes this NIDS a great tool, the capability to auto-tuning. In few words, tune a NIDS is to configure this NIDS to a specific computer network, affecting mainly the detections based in anomalies in the network (i.e. anomaly in the frequency of a behavior). What makes this special is that tuning a NIDS is not a easy work for a system administrator (or even a security specialist), for this reason it becomes a handful feature in this NIDS. Finally, it is a great tool, but it is still lies over the Snort engine.

2.5 Cisco IPS

The worldwide known company in the computer network area also has an IPS solution. The engine is similar to snort, i.e. it uses stateless rules/signatures to detect threats.

This solution has bigger and better explained rules' sets, each rule has a risk value defined, i.e. each rule is evaluated by probability of false positive/negative detection versus impact of the threat (e.g. a rule that has almost null false positives/negatives and has a critical impact of the system, this rule will have a high risk value). In this solution the rules are created in a higher abstraction level, making easier to create rules, but also sometimes it makes more restrictive (i.e. you cannot work in the byte level). Also, this NIDS obviously has an easy integration with CISCO devices, making easier its use in networks full of CISCO equipment (that is normally the case of many companies).

Finally, in a more technical point of view, it is hard to define if it detects more/better threats than Snort due to the lack of information available.

2.6 IBM Proventia IPS

The biggest Information Technology company in the world also has a NIDS. To be exact, the IBM Proventia IPS encloses 4 functions:

- Classic Firewall
- IPS (NIDS)
- OS Events (i.e. more deep analysis over the log files)
- Buffer Overflow exploit detection (host based)

The NIDS part of this solution uses also stateless signature based detection. The rule syntax in special is incredibly similar to the snort one, as can be seen in the IBM Proventia rule below:

```
alert tcp any any -> any any (msg:"Yahoo accessed";
content:"yahoo"; nocase; sid:5000;)
```

This solution in an overall just presents a more complex solution that includes a NIDS solution that is incredibly similar to Snort.

2.7 CheckPoint IPS-1

CheckPoint is a big company from Israel specialized in system security that also has a NIDS solution. As all the solutions studied until now, it also has a signature based engine. However, it has a big improvement, the capability to create stateful signatures using finite state machine of rules (i.e. a digraph of rules that activate themselves). A bad point in this engine is its ugly graphical interface, but at least functional. Also, in the same way of the other commercial IPS studied, it is practically impossible to compare in a technical point of view which one is the best.

2.8 NIDS Solutions Comparison

In the table 2.1 we can see the characteristics comparison of all the solutions studied here.

	Open Source	Stateless Signatures	Stateful Signatures	Good GUI	Auto Tuning	Host Profiling	Good Rules Docs	Extra Functions
Snort	■	■						
Suricata	■	■						
IPSx		■		■				
NGIPS		■		■	■	■		
CISCO IPS		■		■			■	
IBM Proventia		■						■
CheckPoint IPS		■	■					

Table 2.1 – NIDS solutions comparison.

To complete the analysis presented in the table 2.1, we also present an analysis made by a security company called NSSLABS [NSSLABS4, 2011]. Each year, this company presents reports of a supposed fair, clear and technical comparison of "who detects more threats" between the main IPS in the market. The last full technical comparison (2010) between the main IPS in the market , the first place was given to SourceFire IPS 3D 4500 sensor [NSSLABS1, 2011][NSSLABS2, 2011] [NSSLABS3, 2011].

2.9 Final Considerations

Based in this informal comparison made by the work author, his experience in the security area and the technical comparison from NSSLabs, it was not hard to conclude that even being a totally free IPS, or in our case NIDS, Snort is *de facto* the best cost over benefits NIDS in the market. However, many things need to be improved in Snort.

Snort has a big support to detect remote application exploitation and malicious software and addresses, but it lacks in detection capability in many other types of threats, as for example: threats that exploit protocols or network architectures, threats to VoIP Phones, threats to companies' policies, etc.

Also, Snort is an awesome tool when related to stateless detection (i.e. analyzing only one packet). Rules can be made easily to analyze deeply any packet's content, but Snort is still really poor to stateful analysis. The majority of the stateful analysis performed by Snort are made in a Preprocessor level (e.g. port scan detector preprocessor), but these analysis are in general weak and noiseful, needing to be well tuned to work properly.

Finally, some existing solutions in Snort are not well made. As a good example is the Arpspoof preprocessor. This preprocessor only supports static network based detection (i.e it is necessary to write manually all the IP-MAC relations to detect a possible ARP spoofing (see 3.1.1 for details), while tools in the wild like Arpwatch [ARPWATCH, 2011] does also dynamic network based detection (i.e. it learns the relation of IP-MAC and test for possible IP Flip-Flop effects). In the static mode, it will detect ARP spoofing attacks, but it doesn't save the packets related with the ARP Spoofing, making almost useless to know that someone is attacking when it cannot even say who is performing the attack.

3. DEVELOPED THREAT DETECTIONS

In this chapter we will present all the threat detections implemented in this work, improving the actual capabilities of Snort. There will be 4 types of menaces treated here: internal network threats, SIP VoIP phone threats, policy based threats and profile based threats.

Since that Snort does not provide a good detection capability to threats based in protocols and network architectures, and also does not provide a good detection capability to VoIP threats. For this reason, in the chapter 3.1 and chapter 3.2 we will treat these both types of menaces. In the chapter 3.3 we will present an important type of threats that are not very well treated by Snort, called policy based threats. In the chapter 3.4 will be presented a new concept of detection threat methodology, called profile based threat. Finally, in the chapter 3.5, a resume will be presented, explaining in few words what were developed and the main characteristics of each detection construction.

3.1 Internal Network Threats

In this we will describe some threats that exploit flaws in protocols and network architectures in a local network and how to detect the threat.

3.1.1 ARP Spoofing

ARP spoofing (or ARP poisoning) is the act to send ARP answers with a fake response. There are 3 possible different attacks that can be done, exploiting the lack of secure in the ARP protocol. Considering to the attack explanations the follow machines connected to a switch:

Alice IP:192.168.56.1 MAC: aa:aa:aa:aa:aa:aa

Bob IP:192.168.56.2 MAC: bb:bb:bb:bb:bb:bb

Eve IP:192.168.56.3 MAC: cc:cc:cc:cc:cc:cc (the evil machine)

In a good case, if Alice want to communicate with 192.168.56.2, she will send an ARP packet saying “who has 192.168.56.2 says to 192.168.56.1”. This message will be send in MAC broadcast (i.e. ff:ff:ff:ff:ff:ff), Bob and Eve will receive the message, since that Eve doesn't own 192.168.56.2, it will drop the packet silently, while Bob will see the packet and will answer with a “192.168.56.2 is at bb:bb:bb:bb:bb:bb. The weakness here is that, when Alice receives a message, there is no guarantee that the message is correct, so Alice will always believe in this as true.

Suppose now that Eve answers the question saying "192.168.56.2 is at cc:cc:cc:cc:cc:cc" and that this message arrives before or overwrite (depends the Arptable politic) the good address. At this moment Alice will believe that Eve is the owner of 192.168.56.2, i.e. Eve impersonates Bob.

In a second case, suppose now that Eve answers the question saying "192.168.56.2 is at dd:dd:dd:dd:dd:dd". Since that this address doesn't exist, no one (again, depend of

the switch politics) will receive the packet (including the true destination), causing a DoS.

In a third case, Eve sends to Alice a message "192.168.56.2 is at cc:cc:cc:cc:cc:cc" and sends to Bob "192.168.56.1 is at cc:cc:cc:cc:cc:cc". From this moment, Eve impersonates both Alice and Bob, so any traffic between them will arrive to Eve, that after seeing or modifying the traffic, can forward to the true destination. This attack is an opened door to numerous other attacks based in Man-in-the-Middle.

All the attacks presented have the same basic attack approach, so the detection for one of them normally should work to all of them. Based only in a passive network probing, we can see if an IP is suffering a MAC Flip Flop, i.e. in a small time interval, the MAC is oscillating between 2 different MACs. This solution can give false positives/negatives. This implementation was based in the Arpwatch tool solution [ARPWATCH, 2011].

3.1.2 DNS Rebinding

One important security policy in browsers is the same-origin policy, where a script cannot access another hostname:port that it is not related with its scope (i.e. the address that launched it). Also, it is important to highlight that an address resolution (DNS query) can be resolved to numerous IPs (e.g. www.google.com) and the same-origin will accept these IPs as they are from the same scope. Exploiting this policy, the DNS Rebinding attack solves an address (e.g. evil.mywebsite.com) to the true address but also to the attacked IP (public or private), making possible the connection redirection.

The DNS rebinding attack based in small TTLs was for many years an important threat, because with that an attacker has access to the internal network from outside using the victim as a "proxy". Nowadays the DNS rebinding based in an internal DNS doesn't work due to DNS Pinning method (i.e. browser pins the first DNS response and doesn't accept the new one) and the policy of always use an internal IP first when receiving public and private IPs in a DNS resolution.

In 2010, Craig Heffner presented a new attack using DNS Rebinding [HEFFNER, 2010]. This new attack profits of a bad policy in routers that accept connections in an interface with an IP of another interface. So the attack rebinds to a Public Router IP and access this IP from the internal interface.

The easiest way to detect the Heffner's attack is to test if an external IP is solving to our public router/gateway IP. A simple rule was developed to be able to detect this attack in the NIDS module.

3.1.3 DNS Tunneling

Using DNS tunneling applications such as iodine [IODINE, 2011], an attacker can gain external access by encapsulating its traffic into DNS traffic. The attacker has at least to needs before the attack: be authoritative for (at least) 1 DNS zone and install a tunneling tool on the server that is authoritative for the zone.

This attack is common in networks where you have access to the internal network, but you don't have access to the external network. However, in many cases, the internal DNS server is still accepting requests to the exterior.

Suppose an attacker authoritative to a domain (e.g. myevildomain.com), if he is inside the victim's network he can do a DNS query to "hello.myevildomain.com". It will go to the internal DNS server of the network, since that it will not be in the DNS cache from the server, it will ask in the exterior to the DNS query resolution. Supposing again that no one has this address in a cache (the DNS tunneling tools will try to guarantee a unique subdomain request per time) it will arrive in the DNS server authoritative to this domain (that is controlled by the attacker). The DNS server will answer, for example, a CNAME with the address hi.myevildomain.com, it will returns to the internal Server DNS that will forward to the attacker inside the network.

One method of detecting DNS tunneling is by performing statistical anomaly detection on the network. The main characteristic is a high flow of DNS queries to the same top domain. For this reason, the NIDS module uses the frequency method based in the IP source and in the top domain (e.g. myeviltopdomain.com) with different subdomains (e.g. mysubdomain.myeviltopdomain.com). Also, all the primaries top domains are in a whitelist (e.g. .com, .fr, .gov) and almost all the second domains with the same structure (e.g. .com.br, .gov.fr).

3.1.4 DNS Internal IPs Leakage

The resolution of address to internal IPs from the exterior should never happens, being considered a threat due to the fact that it leaks internal network structure information. The internal DNS server, due to a misconfiguration, answers to an external DNS Query request with the internal IP related to the query. After this, the attacker just needs to do a DNS query request to a vulnerable address and he receives this related internal IP.
The detection used analyses the DNS responses from the internal DNS servers to the external network, if it contains at least an internal IP, it will trigger an alert.

3.1.5 DNS Cache Poisoning

This threat is characterized when a DNS server receives a fake answer to a query that will be stored in its cache. When a DNS server does a query to an address, it will wait for a response where the follow answer values match:

- Query ID (16 bits)
- Source query port (16 bits supposing all of them)
- Query request
- The authority in the additional sections is in the top domain
- The first good answer received is accepted

If an attacker can predict all these values, he can fake an answer that will be accepted by the DNS server. The main security characteristic in DNS response

authentication was the random Query ID that needs to match in the query and in the answer. The pool of possible random Query ID is of size of near 65536, for this reason we could say that with a flood of 40 responses it could be hard to guess the correct value. However, Dan Kaminsky presented an attack at Black Hat USA 2008 [KAMINSKY, 2008] how to own and entire zone even with the random Query ID method.

Suppose that a client does a request to iamainvalisubdomain.google.com, it will arrive at some moment to a DNS server authoritative to the google.com zone (since that it will not be in ANY cache) and since that it doesn't exist, it will be answered as inexistent, but it will have an additional information saying that the authoritative server is ns1.google.com - 216.239.32.10 to this domain. The DNS server will keep this information as true (the exploited point by Dan's attack) and the next time the resolution will go directly to the address of ns1.google.com 216.239.32.10.

The Dan's attack tries to poison the authoritative nameserver IP, gaining control of entire google.com zone. The attack follows like that:

- Attacker send request to invalidrandom().google.com resolution request to the victim's DNS server
- Attacker flood with random Query NIDS the victim's DNS server with fake answers saying: domain doesn't exist, but the authoritative DNS server to google.com is ns1.google.com - 50.50.50.50 (the evil IP)
- Repeats the process until guess correctly the query ID (you can because none of the requested addresses are in the cache)

With automatized tools the guessing time is around 10 seconds, after that the attacker owns an entire zone (e.g. .fr, google.com, etc). This attack also does not work anymore due to the massive mobilization of DNS vendors and open source communities.

The problem is the small pool of possibilities related with the Query ID, so the solution is to increase the number the possibilities. The way used today is the use of a random source port (16bits), making the new pool of size 65k * 65k = 4G, becoming really harder to guess the right values. Thankfully to the massive cooperation between DNS server providers and the Kaminsky intelligence, this attack does not work anymore.

The only well-known way to apply a DNS cache poisoning in 2011 is if you can eavesdrop (i.e. capture) the traffic between the victim's DNS Server and the external victim's network. Having the DNS query packet, you have all the authentication values to give a valid fake answer. Since that Eavesdrop is a passive behavior, the only way to stop such attack, is to stop the vector that allowed the eavesdropping. Some vectors can be found in the chapter 3.1.1 and 3.1.7..

3.1.6 DHCP Exhaustion

DHCP Exhaustion (also known by DHCP Starvation) is a threat when an attacker exhausts all the pool of available IPs in a DHCP server [RFC 2131, 1997].

The first method to exhaust the DHCP server pool is flooding the network with DHCP discovers to consume all the available IPs in the server. This works because after a discover packet, the DHCP server will offer an IP to the MAC that sent the discover packet and wait a possible request time, making the offered IP unavailable to anyone else in this interval. To detect this threat, we used anomaly based detection, i.e. if the quantity of DHCP discovers passes a frequency limit, an alert is trigged.

The second method uses a more intelligent and stealth attack. Instead of only to make an avalanche of discover packets, this attack allocates all the offered IPs to all the created spoofed MACs. A good tool to perform this attack is the Metasploit [METASPLOIT, 2011] module called Digininja [DIGININJA, 2011]. The detection in this case is based in the frequency of allocated IPs in a switch's port, if the number of allocated IPs passes a value, an alert is trigged. The detection of this threat is only possible when the Mancala Network framework has access to the Switch configuration.

3.1.7 Rogue DHCP Server

A rogue server is an unauthorized server that coexists in a network with the authorized server. In the moment that a rogue DHCP sever starts to answer discovers and requests of others machines in the network, it can lie about all the information configured by the DHCP server, including gateway and DNS servers.

If a rogue DHCP server lies about the gateway, it will be able to apply a man-in-the-middle between the victim and the gateway. It is only necessary that the DHCP server answers the DHCP discover and request with the correct values from the authorized server, except by the gateway, giving his own IP. The same thing can be used to make a man-in-the-middle in the DNS server.

It is possible to deny the service to any part in the network that the DHCP answer can be forged, for example: the Gateway and the DNS servers. The attacker gives an invalid value of gateway or DNS server, making it unavailable.

The NIDS developed in this work uses two methods to detect rogue DHCP servers. The first method analyses the DHCP answers packets to see if the MAC origin matches with the authorized DHCP server MAC. It can be evaded if the rogue DHCP server spoofs its MAC address.

The second method used to detect rogue DHCP server uses a discover packet with spoofed MAC address, time to time, if 2 DHCP answers are received, it means that exist a rogue DHCP server.

3.1.8 Port Scanning

Port scanning is the act of probe a host or multiple hosts to find opened ports. Usual methods can be used to detect if a port is opened, the main ones can be found in the tool Nmap[NMAP, 2011].

A port scan can be based in the protocol TCP or UDP. In the TCP case, way to detect opened ports is to abuse some characteristics of the TCP flags. The most usual case is

the SYN scan, where the attacker sends a SYN packet to a specific port against a victim, if it answer with a SYN/ACK, it means that the port is open (i.e. starting the handshake), if the victim answers with a RST (i.e. port closed in Linux systems) or if it does not answer, it means that it is filtered, or it cannot reach the destination. Of course, this one is the simplest port scan TCP based, but we have others that will not be detailed here, as for example: NULL Scan, Xmas Scan, FIN Scan, ACK Scan, etc. The UDP scan depend a little bit more of the service in the application layer that the port is related.

The detection to this attack was not developed because Snort already has a good pre-processor to detect scans. This preprocessor can detect both probe types presented: TCP and UDP (also ICMP [RFC 792, 1981], but it does not fit in port scan).

3.2 SIP VoIP Threats

VoIP phones SIP based are almost a standard to companies' telephony. It happens due to low cost calls and easy implementation of these systems that only depend of a simple IP network infrastructure. Nevertheless, in the standard implementation, the SIP VoIP Phones presents numerous vulnerabilities based in the protocol infrastructure, network architecture and firmware flaws, as will be presented next. The lecture of this vulnerabilities are depended of a previous knowledge in the SIP and RTP protocols, that can be find in details in their respective RFCs [RFC 3261, 2002][RFC 3550, 2003].

3.2.1 SIP Register Credentials Guessing Flood

An attacker can apply a brute-force attack against the SIP Registar to find an registered user credential. In the figure 3.1 is represented the authentication method of an user to its respective Registar. As can be seen, first the user Alice tries to register herself in the Registar, as an answer the Registar sends that she needs to authenticate herself, sending a challenge based in a Nonce (abbreviation of number used once) and a digest algorithm, as for example MD5 [RFC 1321, 1992]. After that, Alice will compute the Nonce together with a secret (e.g. a password) with the specified digest algorithm and send back to the Registar, if the digest is correct, the Registar sends a OK, else it sends a message of forbidden (error 403).

Since that the number of times that an user tries to register himself in the Registar is not tested, it is easy to see that someone can just apply a brute-force in this authentication method, guessing the credentials of the victim, until find the correct secret. However this method is slow and definitely works only to weak passwords (i.e. less than 5 characters or easily guessable).

When the authentication fails, the Registar sends a 403 (forbidden) SIP message to the attacker, so if we have many 403 messages (or 4xy in general) to some IP in a small time interval, it means that this IP is maybe trying to crack an user password. This detection based in the frequency of 403 messages can give false positives (as any frequency based detection), being dependent of the constant trigger value, defined in the detection as 10 403 messages in 1 second from the same attacker.

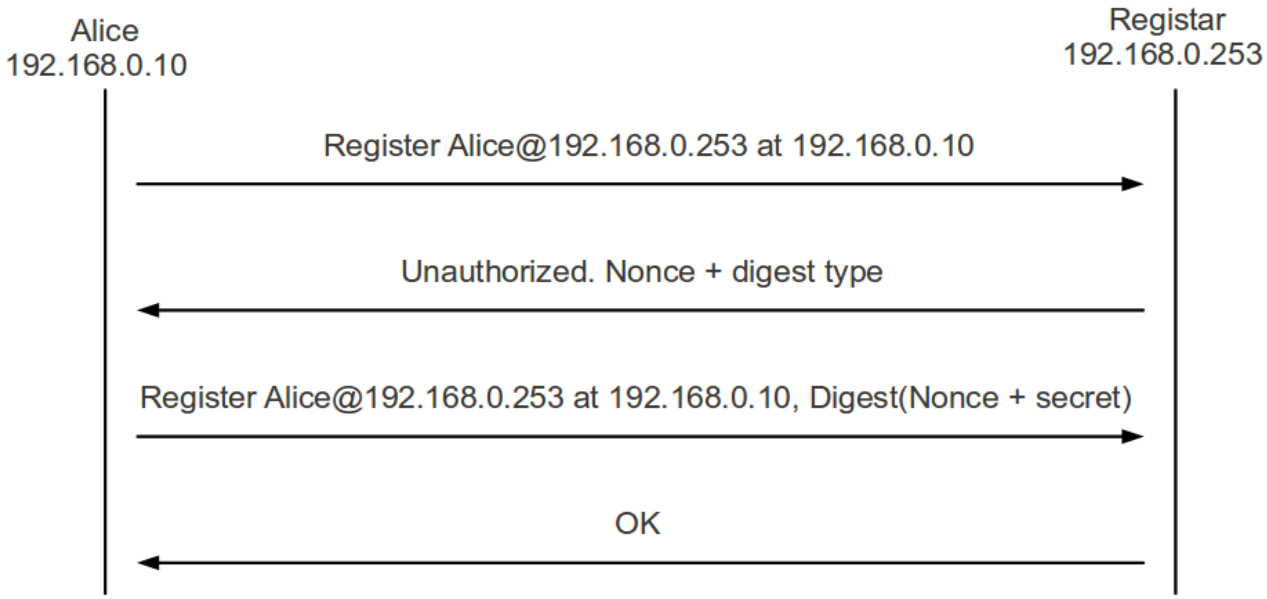

Figure 3.1 – User Alice authenticate herself to a SIP Registar.

3.2.2 MD5 Hashes Eavesdropping

If we have access to a sample of nonce/hash from the victim, we can try to reverse the hash guessing the credentials locally. If the attacker is able to probe the victim traffic, he can use the tools sipdump and sipcrack, found in Backtrack [BACKTRACK, 2011], to capture the Nonce/hash relation and finally crack it. This attack is definitely more effectively than the Registar brute-force due to the fact that the brute-force is made locally.

To apply this threat, the attacker need to have access to the relation nonce/hash, the main method have access to this is to eavesdrop the victim VoIP traffic. Since that eavesdrop is a passive act, we need to detect how it was eavesdropped. See chapters 3.1.1 and 3.1.7 for methods that can be vectors to this attack.

3.2.3 SIP VoIP Fuzzing

Fuzzing is a technique to test software, often in an automated way, to involve providing invalid, unexpected, or random data to the inputs of the respective software. In this case, SIP VoIP Fuzzing is the Fuzzing technique applied over the VoIP firmware responsible to deal with the SIP traffic. A study presented in the Misc magazine [MISC, 2011] edition 39 shows how vulnerable VoIP phones firmware are over SIP Fuzzing attacks. Even being from the end of 2008, not much changed over the firmware security policies.

Automated attacks can be applied easily with the advent of some tools like SIP PROTOS [PROTOS, 2011], that can be found in the Backtrack [BACKTRACK,

2011] operational system. It is important to highlight, that a Fuzzing attack can be responsible by the follow threats (or at least be a vector to):

- Unstable behavior
- Denial of Service / Firmware Crash
- Arbitrary Code Execution

To detect this attack, many rules created by VRT that detects invalid SIP packets can be used to detect Fuzzing attacks. Due to the incomplete nature of the RFC SIP, and sometimes even the disrespect of this norm by VoIP phone vendors, some of these rules can give some false positives

3.2.4 SIP Invite Flood

The invite SIP message is the equivalent of SYN to a SIP session, i.e. it starts the SIP session negotiation. When an attacker sends an excessive quantity of invite to a certain VoIP phone, this one will consume its resources, causing a possible Denial of Service.

This attacks is detected with a frequency based solution, i.e. if an IP is sending multiple SIP invite packets in a small time interval to the same IP destination, it will be considered a Denial of Service attempt.

3.2.5 Eavesdropping Call Sessions

If the media transport is made by the RTP protocol (that is almost always the case), the call session is passing in clear, so we just need to sniff them and transform it in an audio file.

Since that eavesdrop is a passive act, we need to detect how it was eavesdropped. See chapters 3.1.1 and 3.1.7 for methods that can be vectors to this attack.

3.2.6 Information Leak from Configuration files

When activated, normally a VoIP phone will try to download a configuration file from the server using TFTP. So, if we sniff configure file's name downloaded by the VoIP phone, we can retrieve many sensitive information getting this file from the server.

Since that eavesdrop is a passive act, we need to detect how it was eavesdropped. See chapters 3.1.1 and 3.1.7 for methods that can be vectors to this attack.

3.2.7 Redirection Attacks

In SIP, a proxy or UA can respond to an INVITE request with a 301 Moved Permanently or 302 Moved Temporarily response. The initiating UA should use the value in the Contact header line to locate the moved user. The 302 response will also include an Expires header line that communicates how long the redirection should last. If an attacker is able to monitor for (or is an MITM) the INVITE request, he can respond with a redirection response. It can be an attack that can open ports to

new and more dangerous attacks, like tricking the caller into communication with a rogue User Agent for example.

Detecting 301 or 302 responses can alert want this threat happens. Of course, it can give false positives, since that you could have a trusted 301/302 answer in your network. The better solution to detect this threat is to avoid eavesdrop, avoiding for example man-in-the-middle attacks (see 3.1.1 and 3.1.7).

3.2.8 SIP Devices Scanning

SIP devices scanning is the act of probe a host or multiple hosts to find SIP based devices (e.g. VoIP devices). You can find a good SIP scanner in the auditing VoIP device tools called SIPVicious [SIPVICIOUS, 2011].

To detect this attack, we used a frequency based solution, i.e. if an IP is sending multiple SIP options packets in a small time interval, it will be considered a scan. To improve: detect if the destination is different to each packet (a true scan) and add the possibility to be other types of scan (e.g. invite scan).

3.2.9 SIP Enumeration Scanning

Many Registar implementations (e.g. Asterisk 1.6.2 [ASTERISK, 2011]) when receive a REGISTER message answer or with 401 – Unauthorized (if the user exist) or with 404 – Not Found (user does not exist). This behavior clearly leaks information about the existing users in the Registar, making possible to enumerate the existent users in the Registar. The module swvar from the auditing VoIP device tools called SIPVicious [SIPVICIOUS, 2011] offers this feature.

The detection based in frequency is the solution used to detect this threat. If an IP is sending more SIP Register packets to an IP destination than the trigger value (5 tries 1 minute), it will be considered an SIP Enumeration Scanning attempt.

3.3 Policy Based Threats

Some traffic behaviors can be not exactly associated with the classical point of view of threats, but it can be against the corporation network policy. It is very common in corporations to deny some specific type of services that, for numerous reasons, disturb the company environment, as for example, p2p programs to share files. Due to this need, the NIDS module also supports a module to detect the most common undesired behaviors that are presented in the follow chapters.

3.3.1 Chat

This engine is responsible for detecting the main social networks and chat programs. The list of detected chat networks can be seen below:

- Facebook
- GaduGadu
- Google Talk
- Google IM

- MSN
- Yahoo! IM
- IRC based software
- Skype
- ICQ
- AIM
- Jive

The detection engine used existing signatures in Snort, signatures from Emerging Threats and signatures developed by the work's author. The signatures used by Snort are in general based in the detection of certain parts of its traffic (e.g. IRC has a very specific traffic that can be easily detected, of course, if not encrypted). However, the signatures developed by the work's author are based in the DNS resolution of those services (e.g. if www.facebook.com is asked in a DNS query, we detect it and block).

3.3.2 P2P

This engine is responsible for detecting the main P2P programs in the wild. The list of detected of P2P programs are listed below:

- ABC
- Ares
- Vuze/Azureus
- Bittorrent
- Gnutella
- eDonkey
- Kaaza
- GnucDNA
- LimeWire
- Manolito
- Morpheus
- OctoShape
- Pando
- SoulSeek
- ThunderNetwork
- Emule
- uTorrent

In this engine exists both signatures from Snort communities and signatures developed by the work's author. In the same way of the Chat engine, the existing rules use characteristics from the protocols to detect the certain type of service. Finally, the author also developed rules based in their DNS resolution, when existent.

3.3.3 Online Games

This engine is responsible for detecting the main online games running in the network. The list of detected online games can be seen below:

- Alien Arena
- StarCraft
- Brood War
- Diablo
- Diablo 2
- Warcraft 2
- Warcraft 3
- Battle.net general traffic
- World of Warcraft
- Guild Wars
- Trackmania general traffic

Exactly the same way of the other engines presented (chat and p2p), the games existent signatures are based in special characteristics of their traffics that due to lack of the time of the author of this work, he did not study deeply to see these points in details.

Also, there are rules developed in this work, this rules are based in the DNS resolution of URL addresses related with these services (e.g. battle.net).

3.3.4 Inappropriate Content

This engine is responsible to detect sensible words/phrases related with pornography.

The detection is based in sensitive words, normally related with pornography searching, including words well known to be related with pedophilia. The author included many signatures to detect words that he considered related with pornography searching.

3.3.5 TOR Network

This engine is responsible to detect machines that are using the service TOR [TOR, 2011] to hide their traffic (and maybe do things that they should not be able to do).

This engine works only with existent rules from Emerging Threats. It used a regularly refreshed database of border nodes IPs from the TOR network, so if some traffic is detected coming from or going to one of these addresses, it will trigger an alert of TOR use.

3.3.6 NAT

This engine is responsible to detect devices that are implementing a NAT in the network.

The detection, developed by the work's author, is based in the IP field TTL behavior. This method is based in 2 assumptions:

- Different Operational Systems normally have different default TTLs
- NAT decreases TTL value (as a router)

Based in these assumptions a tool can apply over a probed traffic the follow rule: if a IP just does flip- flops of TTL default values in a small time interval, it probably means that this IP is running a NAT. Tha was the method implemented over the NIDS module. However, this method can give false positives (e.g. traceroute traffic). Also, due to the stateful nature of this detection (same thing to ARP Spoofing and DNS Tunneling), it is not used Snort as engine, but Perl scripts.

3.3.7 General

This engine detects different services that do not fall in the other categories described before. A non- exhaustive list of detected services is listed below:

- PCAnywhere
- Google Desktop
- Teredo Traffic
- Nintendo Wii generated traffic
- XBOX generated traffic
- Microsoft Office generated traffic
- AOL ToolBar
- Megaupload download service

This is the rest of signatures from ET and VRT that do not fall in any other category presented before. The majority of them are based in special characteristics of each service that is present in its traffic.

3.4 Profile Based Threats

An informal definition that the report author gives to classical threats is "things that cannot arrive independent of the attacker profile". In other words, when a device starts flooding a Ipv4 local network with fake ARP answers, it doesn't care if the attacker is a printer, a SIP VoIP phone or a PC, it will be considered a threat. However, imagine that there is a device in the network that is doing a DNS resolution to www.facebook.com, if it is a PC, normally it will look correct, but what happens if the device is defined as a printer, it will probably be an unexpected and undesired behavior, indicating that something wrong is happening in the network.

The NIDS module uses the MCN engine (that in real time create profiles of all the active devices in its visible network) to apply personalized rules to a specific profile.

Defining a profile to a diversified group of devices (i.e. what these devices can and cannot do) can be a hard task, due to the lack of a "perfect" pattern between all of them. However some devices groups, as an example SIP VoIP Phones, have a kind

of strict behavior and to these kinds of devices, a detection policy based in the type of device is welcome.

In this topic it will be presented profiles that were created in the NIDS Module. It is important to highlight that a profile usually is not perfect (since that there isn't a formal specification about the respective device type), so it is not impossible to find a device that does not fit with the profile definition.

3.4.1 SIP VoIP Phones Profile

It was defined that a SIP VoIP Phone can only do the follow tasks:

Client Side:

- SIP at port 5060-5070 (most used 5060 and 5061)
- RTP: UDP ports 16384-32767[RFC 3550]
- FTP (21) and TFTP (69)

Server Side:

- SIP at port 5060-5070 (most used 5060 and 5061)
- RTP: UDP ports 16384-32767[RFC 3550]
- Webserver (80, 443)
- SSH (22)
- Telnet (23)

So, any device classified as a Sip VoIP Phone could in theory have only the behaviors described below, if any traffic from/to this device is outside this definition, it will trigger an alert.

To automatically relate IPs to devices, we used the Mancala Network Framework (that has automatically host profiling), after it, we just created rules testing if one of these devices are presenting such traffic, as described in the profile. Of course, since that there is no perfect standard related with SIP VoIP Phone devices, it can give some false positives.

3.4.2 Another Profiles

Due to the limited time of the work any other profile was created. However, there are some profiles that are highly recommended to exist: Printers, DHCP server, DNS Server, PBX Server, etc.

3.5 Final Considerations

Both types of classical threats presented before, Snort had not a big detection capability, but with this work were developed numerous detections to the known attacks related with these 2 topics in the last years. In the table 3.1 we can see a resume of all the developed detections.

	Entirely Developed	Improved an existent solution	Use only the Snort Solution	Not dangerous anymore	Based in an external solution	vector is detected
ARP Spoofing						
DNS Rebinding – Small TTL Attack						
DNS Rebinding – Heffner Attack						
DNS Tunneling						
DNS Internal IPs Leakage						
DNS Cache Poisoning – Kaminsky Attack						
DNS Cache Poisoning - Eavesdropping						
DHCP Exhaustion – Discover Flood						
DHCP Exhaustion – Spoofed MAC Association						
Rogue DHCP Server						
Port Scanning						
SIP Register Credentials Guessing						
MD5 hash crack - eavesdropping						
SIP VoiP Fuzzing						
SIP Invite Flood						
Eavesdropping Call Sessions						
Information Leak from Config Files						
Redirection Attacks						
SIP Devices Scanning						
SIP Enumeration Scanning						
Chat						
P2P						
Online Games						
Inappropriate Content						
TOR Network						

NAT						
General						
SIP VoIP Phones Profile						

Table 3.1 – Classical Threats Detection Treated

Even developing many types of threat detection, there are many other types that could be treated, as for example: web application threats, the microcontroller application threats and new profiles to profile based detection. The detection of these threats is recommended as future work.

4. NIDS MODULE DEVELOPMENT AND INTEGRATION

In this chapter it will be presented the needs to the software development, its structure and finally the followed process to its development.

4.1 NIDS Module Overview

Before develop the module, some constraints need to be respected to the module development be consistent with the pattern applied by the Mancala Networks Development Team.

The development language was Perl (the language used by the Mancala Network Controller), including some specific libraries (e.g. Rose to handle files) and determined functions. Also, the development was totally object oriented.

The IDS module should also follow the structure of all the other modules in the Mancala Network Controller, as for example: use templates for database, develop test codes, create views (to the CLI module), library codes, makefiles, debian package configuration, dependencies, configuration template files, etc. Also, to store and organize the module, GIT [GIT, 2011] was used.

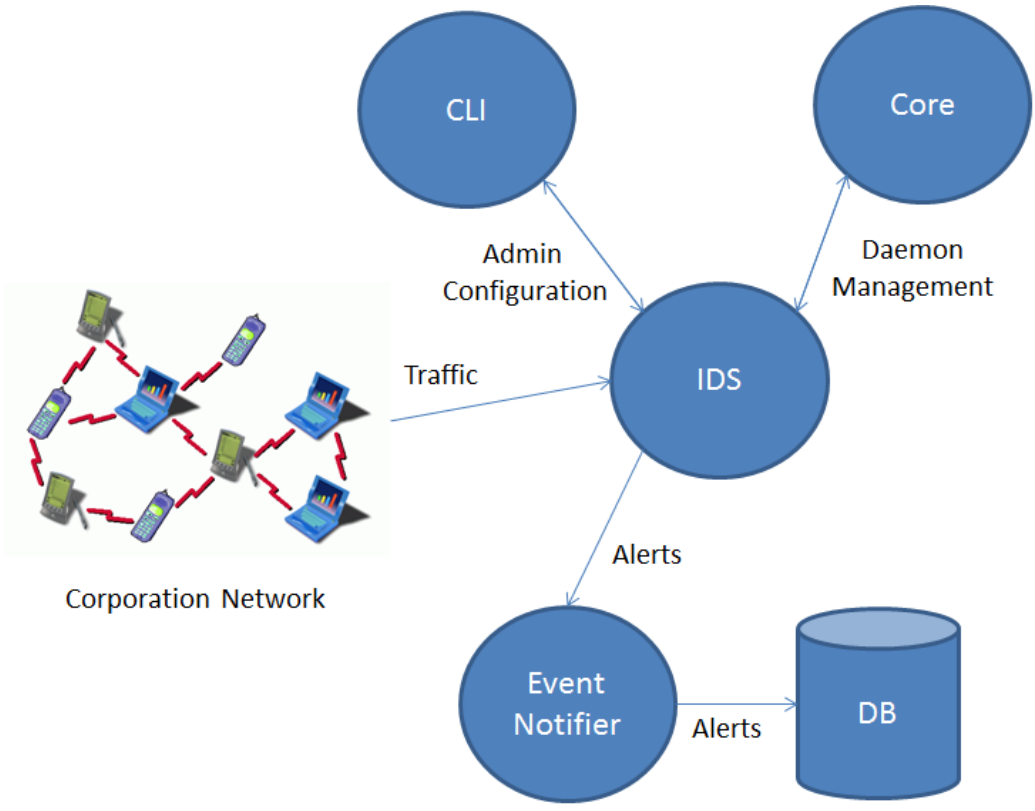

Figure 4.1 – NIDS Module Integration Architecture Simplified

```
mcn ids> ?

ids: manage the IDS service.

AVAILABLE COMMANDS:
  config    manage the configuration.
  detect    manage IDS detection configuration.
  disable   disable the IDS service.
  enable    enable the IDS service.
  network   manage IDS network configuration.
  status    display the IDS service status.
  topology  manage IDS topology configuration.


Type 'help  <command>' to get more details about a specific command.
Type 'exit' to exit from the current module or from the Command Line Interface.
Type 'end'  to exit from all modules.

DESCRIPTION:

The IDS (Intrusion Detection System) service listens to the network to detect
numerous threats or behaviours that can be harmful.

Examples of Detected Threats:
- Internal Network Structure Threats:
        - ARP Spoofing (used to MitM attacks)
        - DHCP Rogue (used to MitM attacks)
        - DHCP Starvation (DISCOVER flood based)
        - DNS External Leakage
        - Network Discovery (i.e. Scans)
        - DNS Tunneling
        - NAT Detector
- SIP VoIP Phones
        - Network Discovery (i.e. SIP scans/enumeration)
        - DoS (fuzzing based, flood based)
        - Credentials Cracking (register based)
- Malware Traffic
        - Botnet Command and Control
        - Trojans
        - Viruses
        - Worms
- Vulnerability Exploitation
- Possible Backdoors Detection
- Malicious Addresses (Extrated from ET sources)

Examples of detected undesired behaviors:
- Chat Traffic
- P2P Traffic
- Tor Traffic (proxy network)
- Innapropriate Traffic
- Games Traffic
- Nat Traffic
COMMAND CONTEXT:
ids

mcn ids>
```

Figure 4.2 – NIDS Module CLI Interface, main menu.

```
mcn ids topology> ?

topology: manage IDS topology configuration.

AVAILABLE COMMANDS:
  add     add one or multiple addresses to a topology type.
  config  manage the configuration.
  del     delete one or multiple addresses of a topology type.
  mod     modify one or multiple addresses of a topology type.
  show    show the IDS topology.


Type 'help  <command>' to get more details about a specific command.
Type 'exit' to exit from the current module or from the Command Line Interface.
Type 'end'  to exit from all modules.

DESCRIPTION:

This sub command is used to manage the IDS topology. The topology defines some
special device types. When a device is defined as a type in the topology, a
respective set of policies will be applied to it. It is important to highlight
that the devices added here should be devices that are expected to act like the
profile that it was added (e.g. add only the DNS servers that you want that
exist in your network to dns_servers).
Other devices will be checked not to act as the that type of server.

Examples:
#set the respective IPs and dhcp_servers as dns_servers device
topology mod dns_servers dhcp_servers 10.10.0.200 10.10.0.201
#add all 10.20.0.0/24 except 10.10.0.32 IPs to sip voip phones
topology add sip_voip_phones 10.20.0.0/24 not 10.10.0.32

COMMAND CONTEXT:
ids topology

mcn ids topology>
```

Figure 4.3 – NIDS Module CLI Interface topology menu.

In the figure 4.1 can be seen the module integration with the Mancala Network framework. All the alerts generated by the module are sent to the Event Notifier module using a respective syntax. These alerts are eventually saved in the database. The responsible to manage the enable/disable system from the NIDS module, including saving all the correct configuration files in the good place, is the Core module.

The CLI is a module responsible to give a hierarchical text based interface, used to the configuration of the NIDS module. In the annexed figure 4.2 can be seen a sample of the NIDS module CLI interface, also, in the annexed figure 4.3 can be seen a sample of the topology configuration show command output.

Using the CLI, it can be configured the follow values:

- Home Network
- Interface
- DNS, DHCP, HTTP and PBX Servers
- SIP VoIP Phones
- Devices to detect
- Chat and social networks
- P2P
- Games
- Inappropriate content
- NAT

It is important to highlight that all of these configuration values are optional, making the module almost “plug-and-play”. However, the home network and the interface values are highly recommended to be configured manually.

4.2 Traffic from the Corporation Network

The NIDS module solution presented is incorporated in the machine that runs the MCN framework. All the analyzed traffic must be redirected to the port that the NIDS probes. In the case that the company that is interested in the module has a small or medium network, a SPAN port [SPAN, 2011] that redirects the traffic to the NIDS could be enough. However, in big networks, it is not feasible to redirect all the traffic to just one port located in one place of the network, that is why that in the section, future plans it will be presented some improvements that could be applied over the NIDS module.

5. NIDS EVALUATION

In this topic we will present the methods used to evaluate the NIDS module and the new detection methods created. The tests can be divided in 3 groups, the software test (see 3.2.1), the software integration with the Mancala Network Controller Framework and the most important, the detection methods tests that uses an entire virtual machines laboratory.

5.1 Unit Tests

The first tests performed, when writing codes, are used to make sure that each subroutine works as it should do. Ensure that given a set of input arguments, the function has the expected behavior and return the correct answer is the fastest way to detect an error in source code. If primary blocks of code do not work, more sophisticated pieces of code build from basics blocks can't work. These tests are called Unit Tests.

Frameworks to do Unit Tests are very dependent of the coding language. The module Test::More written by Michael G Schwern [SCHWERN, 2011] was used during this project to perform unit tests in Perl. In the Figure 5.1 we can see an example of unit tests.

```
#test the delete command
ok($server->del($this_host),"check del command");
is($server->find("test_host"),undef,"check host 'test_host'
deleted");
```

Figure 5.1 – Extract of an user test command.

5.2 Integration Tests

The goal of integration testing is to check that all the individual modules work properly together when integrated in the Network Controller application. It is complementary to unit testing which checks modules with a finer grain in isolation. Integration tests are based on the Virtualization technology.

The basic mechanism of integration tests is relatively straightforward. A virtual computer is emulated using a virtualization solution [CPAN, 2011][KVM, 2011] and a basic Ubuntu operating system installed. Then every components of the Network Controller is installed one-by-one and tested. This is the easiest way to get a clean computer, and to test installation from scratch, without any pre-existent software. This emulated computer is the closest to the production environment – computer on which the product will be installed and shipped to the customer. Actually development environment – computers where the software is developed – is the very different from the production environment, that's the reason why this step is mandatory.

5.3 Virtualized Lab Environment

The company Mancala Networks has a set of servers that implement an automatic laboratory to tests, where we can create routines to each machine (clients, attackers, servers, switches, routers, etc) in a way to simulate threats. This laboratory allowed validating the created detections.

In the figure 5.2 we can see the virtual laboratory structure. The virtualized laboratory make available some high level scripts to automatize the tests, including return the machines to a "safe state" and configure them to different configurations or roles inside this virtualized network. Below a list of all the detections that were tested:

- ARP Spoofing
- DNS Tunneling
- DNS Leakage internal IP
- DNS Internal IP Rebinding
- DNS Internal IPs Leakage
- NAT
- SIP Enumeration Scan
- SIP Invite DoS
- SIP Devices Scan
- SIP Register credentials guessing flood
- SIP Fuzzing with SIP PROTOS
- Rogue DHCP Server
- DHCP Starvation Discover Flood Based
- DHCP Rogue
- Chat DNS resolutions
- P2P DNS resolutions
- Games DNS resolutions
- Inappropriate Content
- NAT
- SIP VoIP Phone Profile

All the tests were applied every night for one month in this laboratory. Also, we tested this module in a real environment (the Mancala Networks network).

The majority of detections created worked to their respective threats. However, the ones based in the frequency of a behavior (e.g. if there is 300 DNS responses queries to the same top domain, it is a DNS Tunneling) presented a high sensibility to false positives and negatives.

The problem related with frequency based detections (or also known by anomaly based detections) is that they are sensitive to the environment that they analyze. Nevertheless, all the frequency based detections have the possibility to be tuned (i.e. to adapt the detection trigger value to the respective environment).

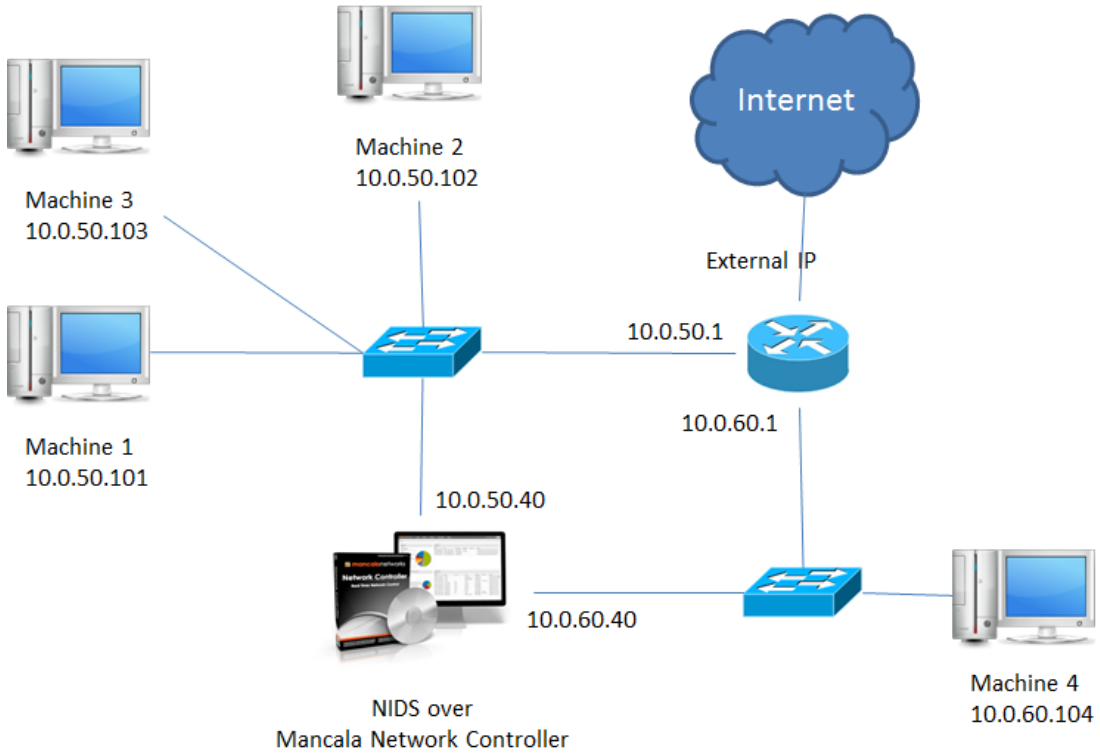

Figure 5.2 – Integration Laboratory Tests Architecture

In the next chapters we will present 4 examples from the Integration Laboratory tests, Facebook DNS resolution (unique signature), the ARP spoofing detection (stateful analysis), the SIP register credentials guessing flood (frequency based) and the SIP VoiP phone profile detection (profile based detection test).

5.4 Facebook DNS Resolution Test

This detection is from the CHAT engine (see chapter 3.3.1). It is a single rule that detects if a DNS Query packet is asking for the resolution www.facebook.com.

In this test, first the Machine 1 generates 3 DNS Queries to the follow addresses:

- www.myfacebook.com
- www.facebookc.om
- www.acebook.com

All these tests were performed to see if the signature was with the correct URL address. In the second part of the test, the machine tries to solve www.facebook.com. This test should trigger an alert.

False positives and negatives were not detected in our tests. However, it is not the perfect way to detect if someone is accessing Facebook, there are many other ways to evade this detection (e.g. have directly the server IP address or access it using a

proxy), but based in the proposition of this rule, it detects perfectly what it proposes: to detect DNS resolutions of www.facebook.com.

5.5 ARP Spoofing Detection Test

This test first analyze the detection capability and in a second moment the false positives and negatives. The Machine 2 does an ARP request of the MAC address of 10.0.50.103, the machine 3 and the machine 1 answer. Since that 2 different IPs are claiming the same MAC, it will trigger an alert of ARP Spoofing.

The second part of the test see if the release time from the detector is respected (i.e. when 1 IP is associated with one MAC, this association has a time-to-live of 5 minutes). So, the Machine 2 does an ARP request asking who has the IP 10.0.50.103, the machine 3 answer with its MAC. 4 minutes later, the Machine 3 flushes its ARP Table and does ask again, the machine 3 does not answer (the network interface is put down), and the Machine 1 answers. This behavior should trigger an alert. The third part tests the release time after 5 minutes, in the same way of the part 2, where in this case this should not trigger an alert.

There are false positives and negatives in this detection. A false positive happens when a machine is configured statically with the same IP of another machine in the network, both machines will not be lying about their IPs, but it will cause many ARP Spoofing alerts. In this case, it is interesting the false positive, because, the majority of the cases, there is no interest of having 2 machines with the same IP in the same network.

The main responsible by false positives and negatives is the release time value. If we put a small release time, many false negatives will happen (just apply the spoofing a little bit later the good answer), but if we put a big release time, we will have many false positives when new machines enter in the network, using IPs that were used before by another machine.

Finally, this detection, mainly due to the fact of being a little bit more complicated than the other presented in the chapter 5.4, it presents a certain quantity of false positives and negatives, but in the majority of the cases, the best solution is tune the release value to the respective network needs.

5.6 SIP Register Credentials Guessing Flood Test

To apply this test, the machine 1 uses to attack a tool from Backtrack called Sipcrack [BACKTRACK, 2011]. The functionality of Sipcrack is: it generates a REGISTER command claiming be an user to the attacked REGISTAR, in consequence the REGISTAR will answer with a challenge (i.e. a string to be used in the hash), the machine 1 answers with a guessed password plus the challenge, all together hashed, if the hash matches, the registration is successful, else it will answer with a forbidden message (code 403).

Since that the detection is based in the frequency of forbidden messages, the quantity of false positives (a normal user answering with a wrong password) and the quantity

of false negatives (a guessing flood below the trigger value) are direct related with the frequency trigger value.

The tests were made over and below the defined trigger frequency value (i.e. 10 messages in 1 minute). The tests below the trigger value, did not trigger an alert, giving a false positive. However, if an user has a random alphanumeric password of at least 6 characters, it will take the average of at least 25.10^8 tries, making the attack expend more or less 494 years to be successful. Of course, if the password is easily guessable, the attack can be reduced to hours or days with the use of a dictionary. False negatives were not detected in the tests.

5.7 SIP VoIP Phone Profile Detection Test

The first part of the test, it sees the existence of false negatives and true threats. In the second step, it sees the false positives. The tests were made using simple Perl scripts to open UDP and TCP sockets, respecting the profile created (see chapter 3.4). To simulate the VoIP device, the machine 1 was defined manually in the NIDS as a VoIP device. No false positives were generated.

The second part of this test was to see if false positives were created. The generated traffics from Machine 1 were:

- TCP destination ports 80 and 443
- UDP destination and source port 53

These ports were chosen because they are the most usual ports in a regular machine and normally not blocked by firewalls. No false negatives were generated.

This test presents a certain trust in the detection, but does not prove that the profile is correct. The profile's tests are more interesting in a real network with real VoIP phones.

6. CONCLUSION

A good security solution, to corporation systems, is a need nowadays. Between the existing solutions, the Network Intrusion Detection System (NIDS) presents an extra layer in the company system security.

The objective of this work was present a NIDS solution that has as objective to improve the detection capabilities from the state of the art NIDS solution in the wild. The developed solution used the engine Snort, defined as the state of the art in the studied NIDS solutions.

The final solution implemented improved the detection capabilities in some threat types, as internal network threats and VoIP threats. However, not only these classical threats are now detected by this new NIDS solution, but also different types of threats, as policy threats, avoiding users undesired behaviors inside a company or even a new approach that judges devices not only based in their IP, but also in their device type (i.e. role in the network). All the new detections were tested multiple times in the Integration Laboratory from the company Mancala Network, and now it is being used by their commercial product, Mancala Network Controller.

The NIDS module is already a stable and a usable module. Nevertheless, there are many improvements that could be applied. The main ones will be discussed here.

One of the main improvements in the NIDS module presented compared with other solutions (inclusive Snort) is the profile based detection approach. As discussed in the chapter 3.4, this approach detects all the traffic in a transport level (i.e. TCP and UDP) that it is not expected to the defined profile. As a concrete example, was the SIP VoIP Phones profile, presented in the chapter 3.4.1. It is strongly recommended to create profiles to Printer, DHCP, DNS, SIP Proxy and SIP Registar servers.

A second point is that many rules are incredibly time sensitive, i.e. 1 week after their release, they can be already considered old, useless and even wrong (e.g. blacklist rules). For this reason, it is vital to have a system to update rules. When treated directly with the Snort NIDS, there is a good solution called PulledPork [PULLEDPORK, 2011]. However, the NIDS modules use modified VRT rules, modified ET rules and rules created in this work, making necessary an adaption of the existent solution.

A third proposition is the creation of detections against web application threats. Some example of attacks that are highly recommended the creation of detection methodologies: SQL injection, cross site request forgery, cross site scripting, file inclusion, poison null byte and server side includes.

Finally, the existent solution only supports one NIDS sensor that is in the same machine where is running the Mancala Network Controller (MCN). In small networks (50 devices or less) it is not a real problem, because the traffic can be easily redirected to an interface in the MCN machine using a switch SPAN port for

example. However, if it is a large network, it is necessary to use a distributed solution to the NIDS module. It is highly recommended to a future release.

REFERENCES

[ARPWATCH, 2011] http://www.securityfocus.com/tools/142, last access: June 2011

[ASTERISK, 2011] http://www.asterisk.org , last access: August 2011

[BACKTRACK, 2011] www.backtrack-linux.org, last access: August 2011

[CHECKPOINT, 2011] http://www.checkpoint.com/, last access: August 2011

[CI ARMY, 2011] http://ciarmy.com, last access: August 2011

[CISCO, 2011] http://www.cisco.com, last access: August 2011

[CPAN, 2011]http://search.cpan.org/~mschwern/Test-Simple-0.94/lib/Test/More.pm, CPAN Test::More, last access: August 2011

[DIGININJA, 2011] www.digininja.org, last access: August 2011

[DSHIELD, 2011] http://www.dshield.org, last access: August 2011

[ET, 2011] http://www.emergingthreats.net, last access: August 2011

[FREERADIUS, 2011] http://freeradius.org, last access: August 2011

[GIT, 2011] http://git-scm.com, last access: August 2011

[HEFFNER, 2010] http://www.blackhat.com/html/bh-us-10/bh- us-10-briefings.html, How to hack millions of routers by Craig Heffner, Black Hat USA, 2010

[IBM, 2011] http://www.ibm.com, last access: August 2011

[IODINE, 2011] http://code.kryo.se/iodine/, last access: June 2011

[KAMINSKY, 2008] www.blackhat.com/html/webinars/kaminsky- DNS.html, Black Hat USA 2008 Kaminsky presentation, 2008

[KEVIN BIO, 1996] Tsutomu Shimomura and John Markoff. "Takedown: The Pursuit and Capture of Kevin Mitnick, America's Most Wanted Computer Outlaw-By the Man Who Did It". Hyperion, 1996.

[KVM, 2011] http://www.linux-kvm.org/page/Main_Page, last access: August 2011

[LULZSEC, 2011] http://www.bbc.co.uk/news/technology-13787229, last access: November 2011

[MANCALA, 2011] http://www.mancalanetworks.com/en/products/, last access: June 2011

[MARKOFF, 2009] Markoff, John (2009-01-22). "Worm Infects Millions of Computers Worldwide". New York Times. Retrieved 2009-04-23.

[MCAFEE, 2011] http://www.mcafee.com, last access: August 2011

[METASPLOIT, 2011] www.metasploit.com, last access: August 2011

[MISC, 2011] – Misc magazine website, August 2011, http://www.miscmag.com

[NETFILTER, 2011] http://www.netfilter.org/, last access: June 2011

[NMAP, 2011] http://sectools.org, last access: June 2011

[NSSLABS1, 2011] http://www.nsslabs.com/research/network- security/network-ips/sourcefire-3d-4500-q4-2010.html, Sourcefire 3D 4500 NSSLabs report, last access: August 2011

[NSSLABS2, 2011] www.clm.com.br/produtos/sourcefire/pdf/NSS- Labs-Network-IPS-Group-Test-Sourcefire-3D4500-Test-Results.pdf, Sourcefire 3D 4500 NSSLabs report for free, last access: August 2011

[NSSLABS3, 2011] www.checkpoint.com/campaigns/intrusion-prevention-system/index.html, Comparison of CheckPoint with others vendors website, last access: August 2011

[NSSLABS4, 2011] www.nsslabs.com NSSLabs website, last access: August 2011

[OISF, 2011] http://www.openinfosecfoundation.org/, last access: June 2011

[PERL, 2011] http://www.perl.org/, June 2011

[PROTOS, 2011] https://www.ee.oulu.fi/research/ouspg/PROTOS_Test-Suite_c07-sip, last access: August 2011

[PULLEDPORK, 2011] http://code.google.com/p/pulledpork, last access: August 2011

[REUTERS, 2007] Jim Finkle (July 17, 2007). "Hackers steal U.S. government, corporate data from PCs". *Reuters*. Retrieved November 17, 2009.

[RFC 134-135, 1987] http://tools.ietf.org/html/rfc1034 and http://tools.ietf.org/html/rfc1035, Domain Name System RFCs (just the 2 basic ones), November 1987

[RFC 768, 1980] http://tools.ietf.org/html/rfc768, August 1980

[RFC 792, 1981] http://tools.ietf.org/html/rfc792, , September 1981

[RFC 793, 1981] http://tools.ietf.org/html/rfc793, September 1981

[RFC 826, 1982] http://tools.ietf.org/html/rfc826, November 1982

[RFC 1321, 1992] http://tools.ietf.org/html/rfc1321, April 1992

[RFC 2131, 1997] http://tools.ietf.org/html/rfc2131, March 1997

[RFC 2616, 1999] http://tools.ietf.org/html/rfc2616, June 1999

[RFC 3261, 2002] http://tools.ietf.org/html/rfc3261, June 2002

[RFC 3550, 2003] http://tools.ietf.org/html/rfc3550 , July 2003

[SCARFONE ET AL., 2007] Scarfone, Karen; Mell, Peter (February 2007). Guide to Intrusion Detection and Prevention Systems (IDPS). Computer Security Resource Center (National Institute of Standards and Technology) (800-94).

[SCHWERN, 2011] http://search.cpan.org/~mschwern/Test-Simple-0.98/lib/Test/More.pm, last access: June 2011

[SHADOWSERVER, 2011] http://www.shadowserver.org, last access: August 2011

[SHNEIER, 2011] http://www.schneier.com/blog/archives/2011/07/is_there_a_hack.html, July 2011

[SIPVICIOUS, 2011] http://blog.sipvicious.org, last access: August 2011

[SNORT, 2011] http://www.snort.org/, last access: June 2011

[SOURCEFIRE, 2011] http://www.sourcefire.com/, last access: June 2011

[SPAN, 2011] www.cisco.com/en/US/products/hw/switches/ps708/products_tech_note09186a00805c612.shtml, Example of Switch SPAN port from Cisco switches, last access: August 2011

[STUXNET, 2011] "Last-minute paper: An indepth look into Stuxnet". Virus Bulletin. http://www.virusbtn.com/conference/vb2010/abstracts/LastMinute7.xml.

[TCPDUMP, 2011] http://www.tcpdump.org/, last access: June 2011

[TOR, 2011] www.torproject.org , last access: June 2011

[VRT, 2011] http://www.snort.org/vrt, last access: June 2011

[WIKIPEDIA, 2011] http://en.wikipedia.org, last access: June 2011

[ZEUS, 2011] http://www.net-security.org/malware_news.php?id=1811, last access: November 2011