

1. Intrusion Detection

- 1.1 Introduction**
- 1.2 Snort Introduction**
- 1.3 Snort in the network**
- 1.4 Snort Installation**
- 1.5 Snort Configuration**
- 1.6 Starting Snort**
- 1.7 Snort Rules**
- 1.8 Example Rule**
- 1.9 3rd party tools**

1.1 Intrusion Detection - Introduction

“the art of detecting inappropriate, incorrect, or anomalous activity”

- IDS provides:

- Monitoring and analysis of user and system activity;
- Auditing of system configurations and vulnerabilities;
- Assessing the integrity of critical system and data files;
- Operating system audit ;

- Classify by operation mode:

- Host Intrusion Detection Systems - HIDS;
- Network Intrusion Detection Systems - NIDS;
- Network Node intrusion Detection Systems - NNIDS;

- Classify by how they work:

- Knowledge Based Intrusion Detection Systems;
- Behavior (Anomalous) Based Intrusion Detection Systems;

1.1 Intrusion Detection - Introduction

Host Based IDS:

- protocol analyzer
 - logsurfer
 - fwlogwatch
 - logwatch
 - logsentry (ex. Logcheck)

- file integrity
 - Tripwire
 - AIDE

- right management / Kernel + Process Level
 - LIDS – Linux Intrusion Detection-System
 - SNARE

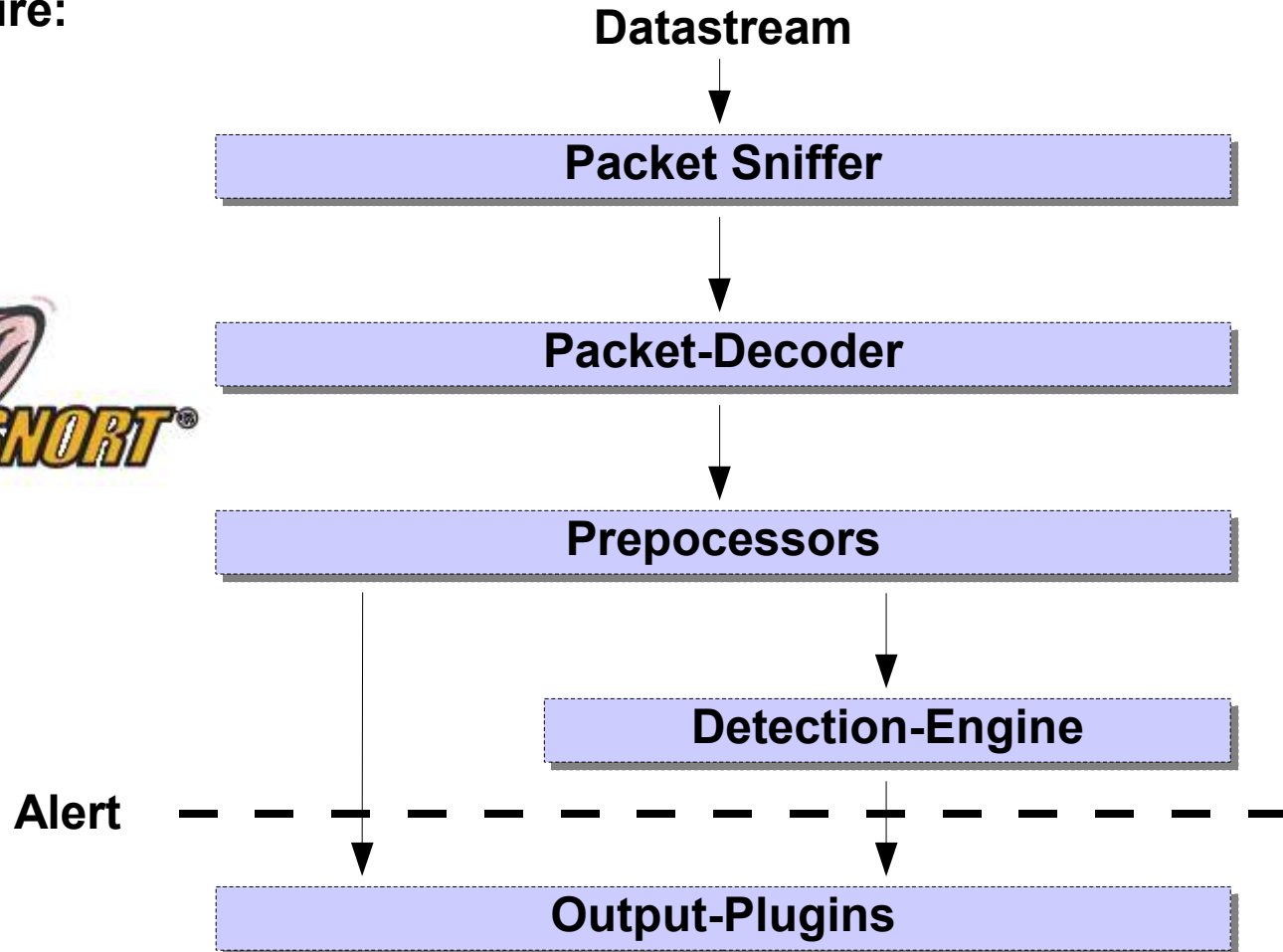
1.1 Intrusion Detection - Introduction

Network Based IDS:

- A network-based ID system monitors the traffic on its network segment as a data source;
- placing the network interface card in promiscuous mode to capture all network traffic;
- looking at the packets on the network as they pass by some sensor;
- sensor can only see the packets that happen to be carried on the network segment it's attached to;
- Packets are considered to be of interest if they match a signature;
- Three primary types of signatures are string signatures, port signatures, and header condition signatures;

1.2 Intrusion Detection – Snort Introduction

Architecture:



1.2 Intrusion Detection – Snort Introduction

Packet Sniffer:

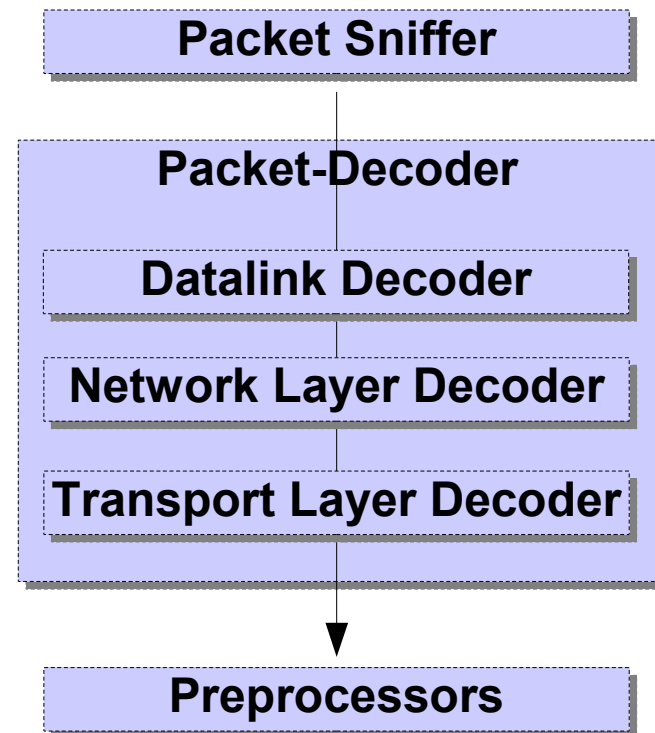
- Snort required „libpcap“
- read raw traffic is important;

Packet-Decoder:

- brings a structure to the raw data;
- works transparent;

Preprocessors:

- modular based;
- “normalizing” the traffic, de-fragment, clean up and bring into got order;
- do first small analysis and may generate an alert;



1.2 Intrusion Detection – Snort Introduction

Detection Engine:

- heart of Snort, performs 2 activities:
 - read signatures during startup;
 - pattern matching;

- Signature

- head

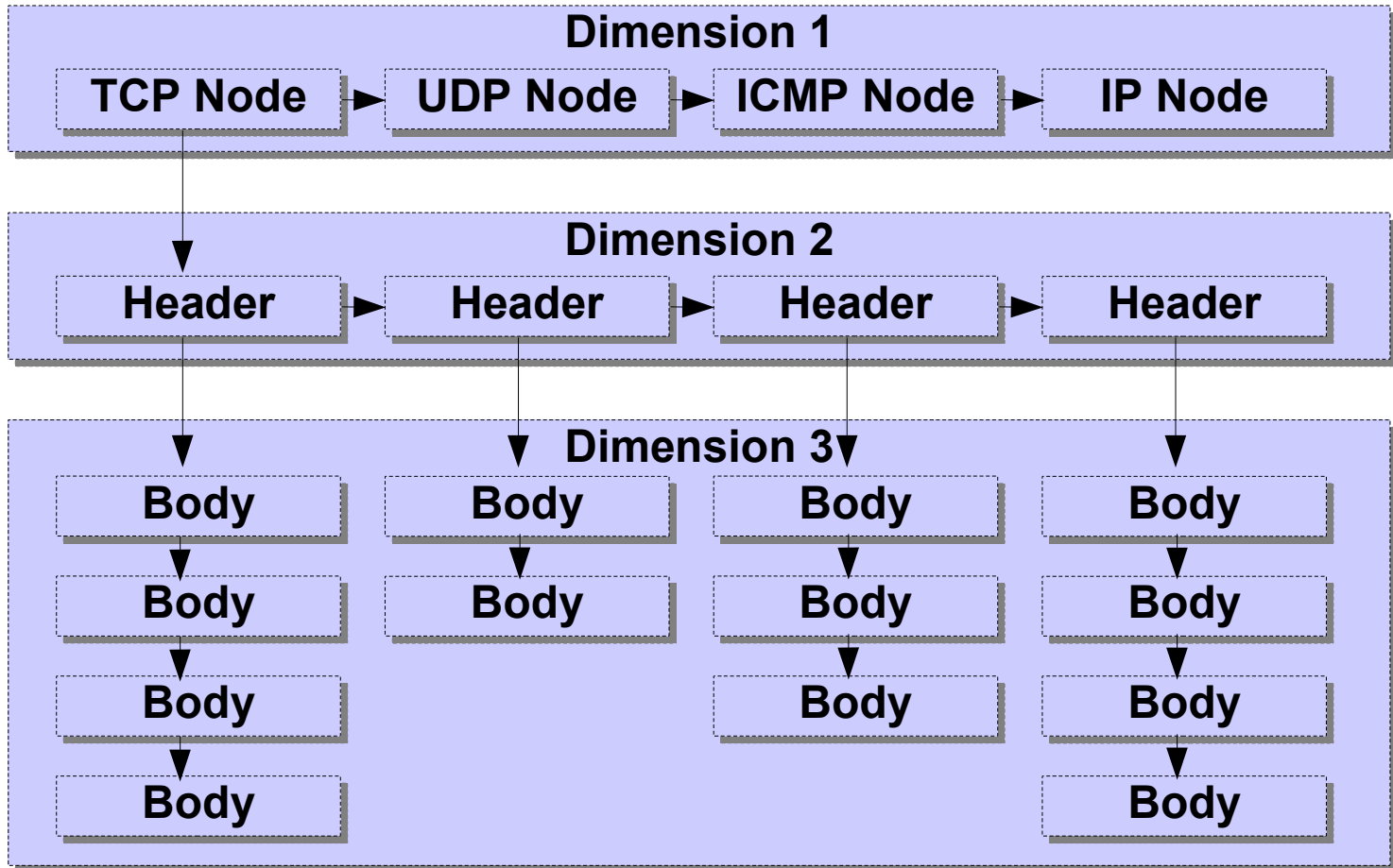
- ```
alert tcp $EXTERNAL_NET any -> $SMTP 25
```

- body

- ```
(msg:"EXPLOIT sniffit overflow"; flags: A+;  
content: "from|3A 90 90 90 90 90 90 90 90 90 90 90|";  
nocase; dsize: >512; reference:bugtraq,1158;  
reference:cve,CAN-2000-0343; reference:arachnids,273;  
classtype:attempted-admin; sid:309; rev:2;)
```

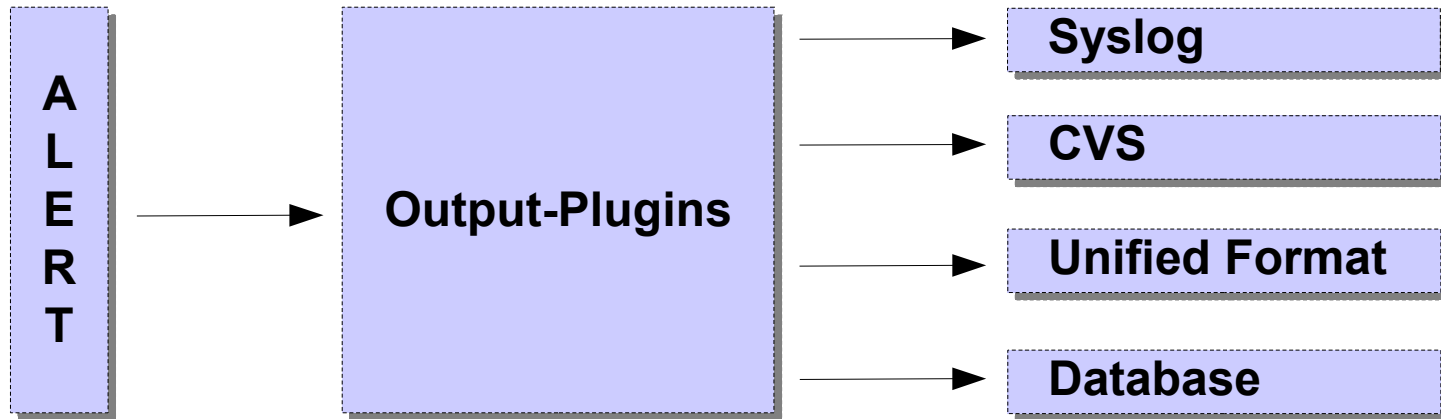
1.2 Intrusion Detection – Snort Introduction

Detection Engine:

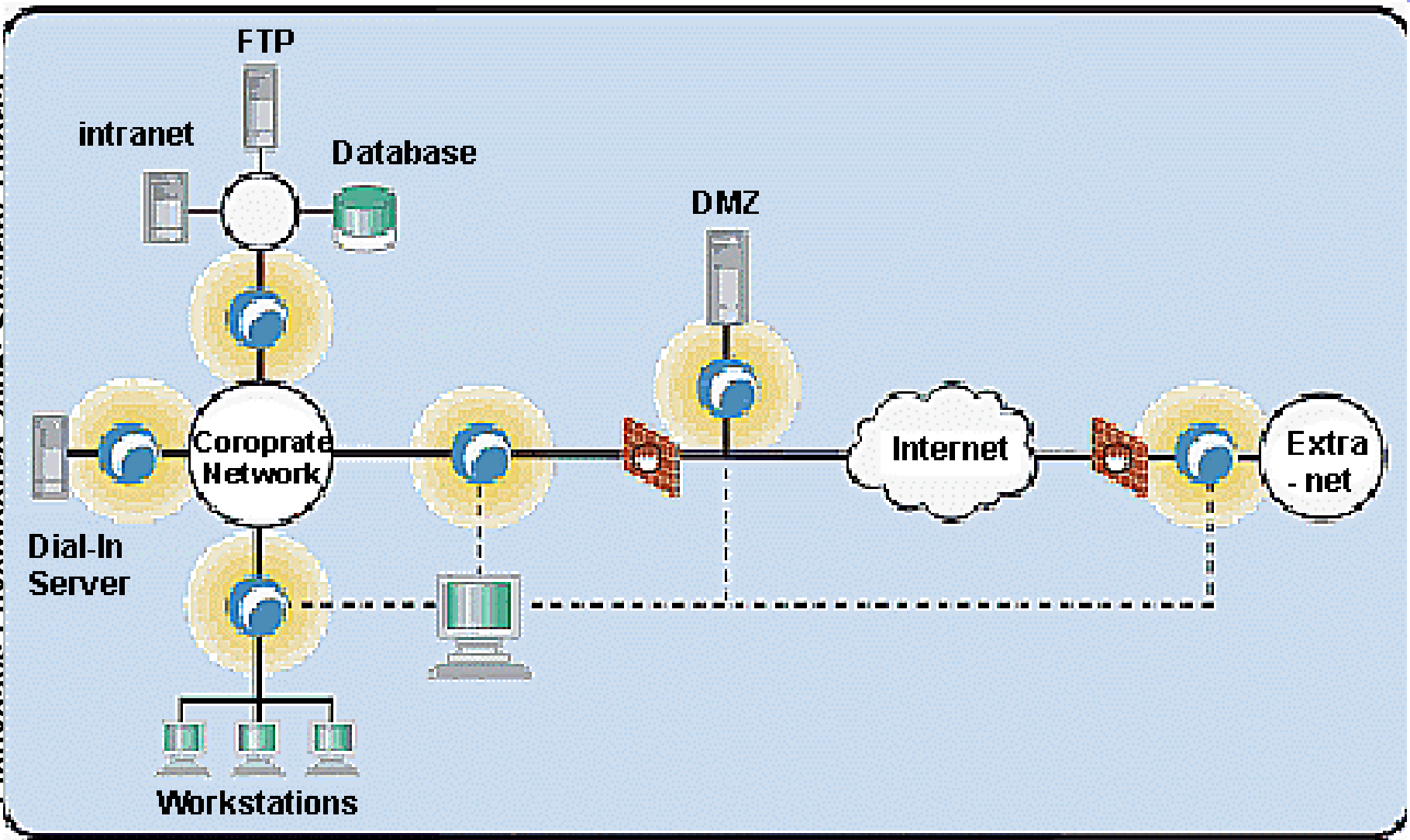


1.2 Intrusion Detection – Snort Introduction

Output-Plugins:

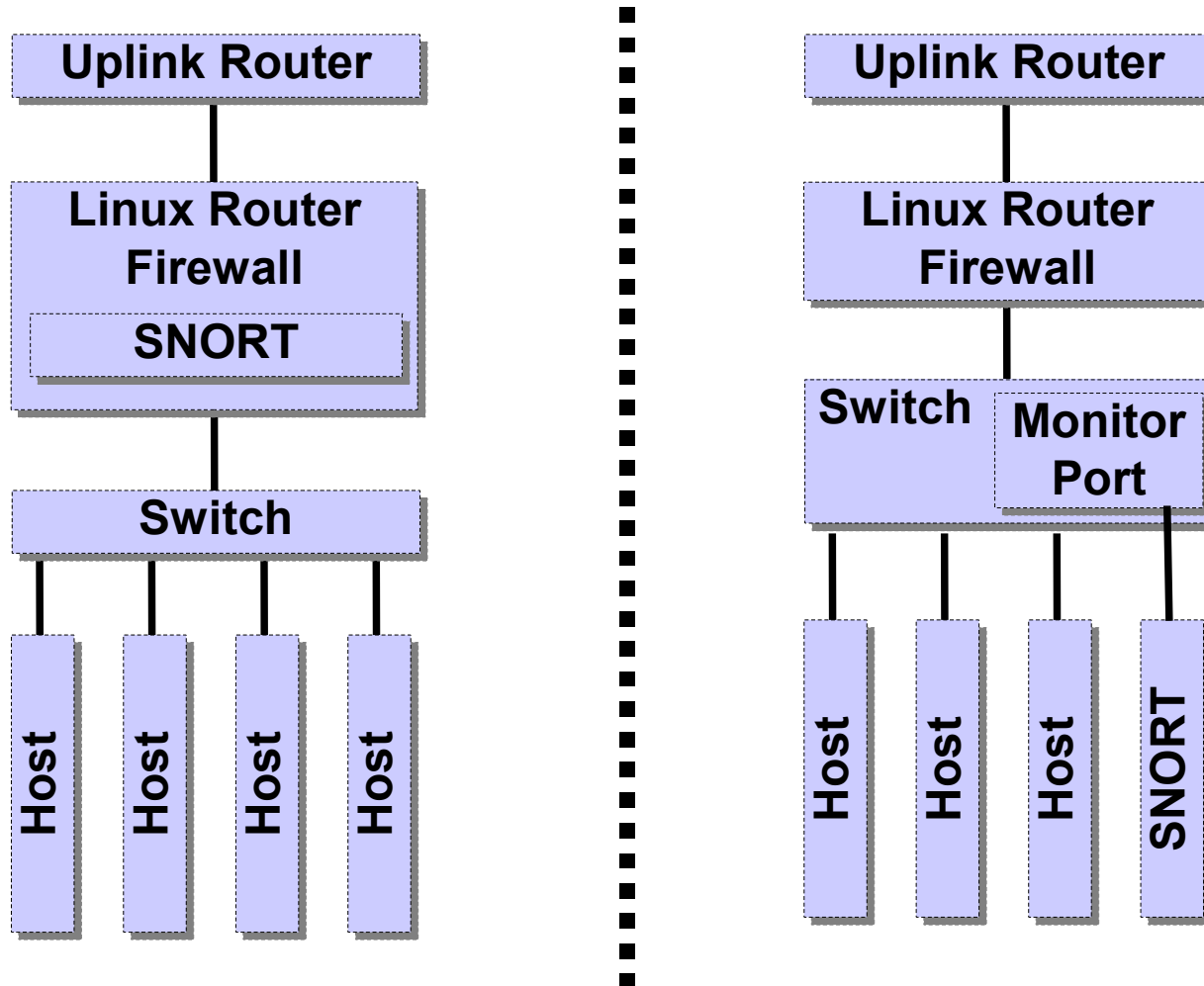


1.3 Intrusion Detection – Snort in the network



1.3 Intrusion Detection – Snort in the network

Linuxdays Luxembourg 2005: Security Tutorial



1.3 Intrusion Detection – Snort in the network

- Sensor without IP-Address;
- Sensor with 2 Network Cards;

Topology:

- Snort on Firewall / Router;
- Snort with multiple Network Card without IP Address on Monitor Port;
- Multiple independent Snort Sensor on the Switches;
- Distributed Intrusion Detection System with a single LogHost;
- Distributed IDS with a single LogHost on a separated network;

1.4 Intrusion Detection – Snort Installation

```
apt-get install snort
  - libnet0
  - libpcap0.8
  - snort
  - snort-common
  - snort-rules-default
  (- snort-mysql)
  (- snort-pgsql)
  (- webmin - snort)
  (- oinkmaster)    Snort Rules Manager
  (- acidlab)       Analysis Console for Intrusion Detection
```

```
/usr/sbin/snort           Snort IDS
/usr/sbin/snort-stat      Tool for Statistics

/var/log/snort            Logfiles
/var/log/snort/alert      Alerts

/etc/snort                Configuration Files
/etc/snort/rules          Rules Database

/etc/init.d              Starting Snort during Startup
```

1.5 Intrusion Detection – Snort Configuration

1. General settings (/etc/snort/snort.conf /etc/snort/snort.debian.conf):

```
config set_gid: snort
config set_uid: snort
config interface: eth0

var HOME_NET [10.35.0.0/16,192.168.1.0/24]
var EXTERNAL_NET any
```

Finetuning: Configure your server lists.

```
var DNS_SERVERS $HOME_NET
var SMTP_SERVERS $HOME_NET
var HTTP_SERVERS $HOME_NET
var SQL_SERVERS $HOME_NET
var TELNET_SERVERS $HOME_NET
var SNMP_SERVERS $HOME_NET

var HTTP_PORTS 80
var SHELLCODE_PORTS !80
var ORACLE_PORTS 1521
var AIM_SERVERS[64.12.24.0/23,64.12.28.0/23,64.12.161.0/24,
64.12.163.0/24,64.12.200.0/24,205.188.3.0/24,205.188.5.0/24,
205.188.7.0/24,205.188.9.0/24,205.188.153.0/24,205.188.179.0/24,
205.188.248.0/24]

var RULE_PATH ../rules
```

1.5 Intrusion Detection – Snort Configuration

2. Preprocessor:

```
# defragmentation of IP-Packets
preprocessor frag2

# prepare TCP-Packets composite to a data stream
preprocessor stream4: detect_scans
preprocessor stream4_reassemble: both, ports all

# normalise HTTP-Data; RPC-Data; Back Orifice; Telnet & FTP;
preprocessor http_inspect: global iis_unicode_map unicode.map 1250
preprocessor http_inspect_server: \
    server default profile all ports { 80 8080 }
preprocessor rpc_decode: 111 32771
preprocessor bo
preprocessor telnet_decode

# portscan; arpspoofing; statistics
preprocessor flow: stats_interval 0
preprocessor flow-portscan
preprocessor arpspoof
preprocessor arpspoof_detect_host: 192.168.0.1 ab:bc:a1:00:fa:1a
preprocessor perfmonitor: time 300 events flow file \
    /var/log/snort/snort.stats pktcnt 10000
```

1.5 Intrusion Detection – Snort Configuration

3. Output-Plugins:

```
# general format
output <name_of_plugin>: <configuration_options>

# defragmentation of IP-Packets
output alert_syslog: LOG_AUTH LOG_ALERT
output alert_syslog: host=hostname:port, LOG_AUTH LOG_ALERT

# log packets in binary tcpdump format
output log_tcpdump: /var/log/snort/tcpdump.log
```

1.5 Intrusion Detection – Snort Configuration

4. Rules:

```
# classification
/etc/snort/snort.conf
    include classification.config
    include reference.config

/etc/snort/classification.config
```

```
config classification: system-call-detect,A system call was detected,2
config classification: tcp-connection,A TCP connection was detected,4
config classification: trojan-activity,A Network Trojan was detected, 1
```

Name

Description

Class 1-4

```
/etc/snort/classification.config
```

```
config reference: bugtraq    http://www.securityfocus.com/bid/
config reference: cve        http://cve.mitre.org/cgi-bin/cvename.cgi?name=
config reference: arachNIDS  http://www.whitehats.com/info/IDS
```

1.5 Intrusion Detection – Snort Configuration

4. Rules:

```
# including the rules
/etc/snort/snort.conf
...
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-php.rules
...

# example rules
/etc/rules/web-cgi.rules
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-CGI HyperSeek hsx.cgi directory traversal attempt";
flow:to_server,established; uricontent:"/hsx.cgi";
content:"../../" ; content:"%00"; distance:1;
reference:bugtraq,2314; reference:cve,2001-0253;
reference:nessus,10602; classtype:web-application-attack;
sid:803; rev:11;)
```

1.5 Intrusion Detection – Snort Configuration

5. Other configuration files

gen-msg.map

- used by Preprocessor based alerts
- maps Processor Generator ID || Signature-ID || MESSAGE
- used by Barnyard for graphical representation of results

sid-msg.map

- used by rules based alerts
- maps SID || Message || optional Reference 1 || optional Reference 2 || ...
- all Generator ID equals to 1
- used by Barnyard for graphical representation of results

unicode.map

- includes all Signs of all alphabets
- used by Preprocessor „http_inspect“

1.6 Intrusion Detection – Starting Snort

```
# -T start snort in testing mode
# -t start snort in chroot jail
# -c read this configuration file
snort -c /etc/snort/snort.conf -t /var/log/snort -T

# -D start snort in Daemon mode
snort -c /etc/snort/snort.conf -t /var/log/snort -D

(create Start/Stop Scripts under /etc/init.d/)
```

1.7 Intrusion Detection – Snort Rules

- to many default rules will provide to much false positives;
- to many rules may cause packet lost
- adapt the rule set to your individual needs
- create your own rules
- a rule is divided into 2 parts: Body and Header

1.7 Intrusion Detection – Snort Rules

Rule Header:

```
tcp $HOME_NET 23 -> any any
```

Action:

alert: logging and send an alert

log: logging; no alert

pass: bypass this packet, don't investigate

activate: send alert and process assigned tests

dynamic: must be called from an activate rule

Protocoll: (IP, ICMP, UDP, TCP, any)

Src-IP Src-Port:

Direction: (-, <>) <- is not allowed

Dst-IP Dst-Port

Example Rules:

```
log any any any <> any any                   # log all packets
alert any any any -> 192.168.0.1 23           # alert telnet to router
alert tcp any !80 -> any any               #
log tcp 10.0.0.0/24 any <> 10.0.1.0/24 any
log tcp [10.10.10.1,10.10.10.2,10.10.11.0/24] any \
-> [10.1.1.1-10.1.1.222] any
log upd any 1:1023 -> any 1024:65535
```

1.7 Intrusion Detection – Snort Rules

Rule Body:

- Investigate the payload of a packet;
- is separated of the body by a <space>
- is enclosed in ()
- Syntax: parameter:"value"
- parameter a separated by a semicolon
- example:

```
alert tcp any any <> any any \  
      (parameter1:"value1"; parameter2:"value2";)
```

Categories of Parameter:

- Meta-Data: Information about this rule
--
- Payload: Content of the packets
- Non-Payload
- Post-Detection: action on match

1.7 Intrusion Detection – Snort Rules

Categories of Parameter:

- Meta-Data: Information about this rule

```
-- msg:"<Message>";
-- reference:<id system>,<id>; \
    # reference:URL,http://www.whitehats.com/info/IDS
-- sid:<Identification Number>;
-- rev:<Integer>          # Versionnumber of the rule
-- classtype:<Classification>; # classification.config
-- priority:<Integer>
```

- Payload: Content of the packets

```
-- content: [!] "<String>";
    --- the most important parameter at all;
    --- search in ASCII and in Binary Data||
    --- example: (content:"|1010 111F AEFEE|");
    --- example: (content:"HTTP");
-- nocase;
-- rawbytes;          # no pre-processor
-- depth / offset / distance / within: # number of bytes
-- uricontent:[!]<String>; # system32/cmd.exe?/c+ver
-- -- --
```

1.7 Intrusion Detection – Snort Rules

Categories of Parameter:

- Non-Payload

```
-- flowbits:<Option>[,<state>];          # assign a status \
      to packets which could be used in alter rules
-- fragoffset:[<|>]<Value>;
-- ttl:[[<Value>-]><=><Value>;
-- top:[!]>Value>;
-- id:<Value>;
-- ipopts:<rr|eol|nop|ts|sec|lsrr|ssrr|satid|any>;
-- fragbits:[+-*]<[MDR]>;
-- dsize / flags / flow / seq / ack / window
-- -- --
```

- Post-Detection: action on match for more flexibility

```
-- logto:"Filename";
-- session:<printable|all> # follow a plain text session
-- block          # shut down a session
-- warn           # send warning message to a browser
-- resp / msg / proxy / tag / session / host
-- -- --
```

1.8 Intrusion Detection – Example Rule

Create an example rule

```
vi /etc/snort/rules/misc.rules

alert tcp any any -> $HOME_NET 22 \
msg:"ATTENTION! Remote configuration attempt"; \
content:"SSH"; depth:128; \
classtype:misc-attack; sid:1000001; rev:2;)

/etc/init.d/snort force-reload
```

sid < 100 reserved for later use
100 < sid < 1.000.000 public rules
sid > 1.000.000 private use

1.9 Intrusion Detection – 3rd party tools

3th party Products:

- Oinkmaster
- Barnyard
- ACID (Analyse Console for Intrusion Database)
 - PHP Scripts for Apache Webserver
 - present database entrys
 - komplex search requests
 - statistiks
 - <http://www.andrew.cmu.edu/user/rdanyliw/snort/snortacid.html>
- MySQL
- Apache
- OpenSSH
- OpenSSL
- stunnel
- Syslog-NG
- NTP

2. Vulnerability Scanning

- 2.1 Introduction**
- 2.2 Nessus Introduction**
- 2.3 Nessus Installation**
- 2.4 Nessus Plugins**
- 2.5 NASL**
- 2.6 Knowledge Base**
- 2.7 Bring it together**
- 2.8 Scanning networks**
- 2.9 The results**

2.1 Vulnerability Scanning - Introduction

Two types of testing:

- Network Based testing;
- Host Based testing;

Three types of approaches:

- Outsider approach;
- Insider approach;
- Hybrid approach;

Limits of vulnerability scanning:

- find only common vulnerabilities;
- false positives;
- false negatives;
 - is not able to find errors in applications;
 - inadequate security checks on user input;
 - cannot replace a security team;

2.1 Vulnerability Scanning - Introduction

Typical process of a vulnerability test:

- detecting of “Live Hosts”;
- identifying of the “Live Hosts”;
- enumerating open ports;
- identifying the service off an open port;
- identifying the application running this service;
- searching and identifying vulnerabilities;
- reporting the found vulnerabilities;



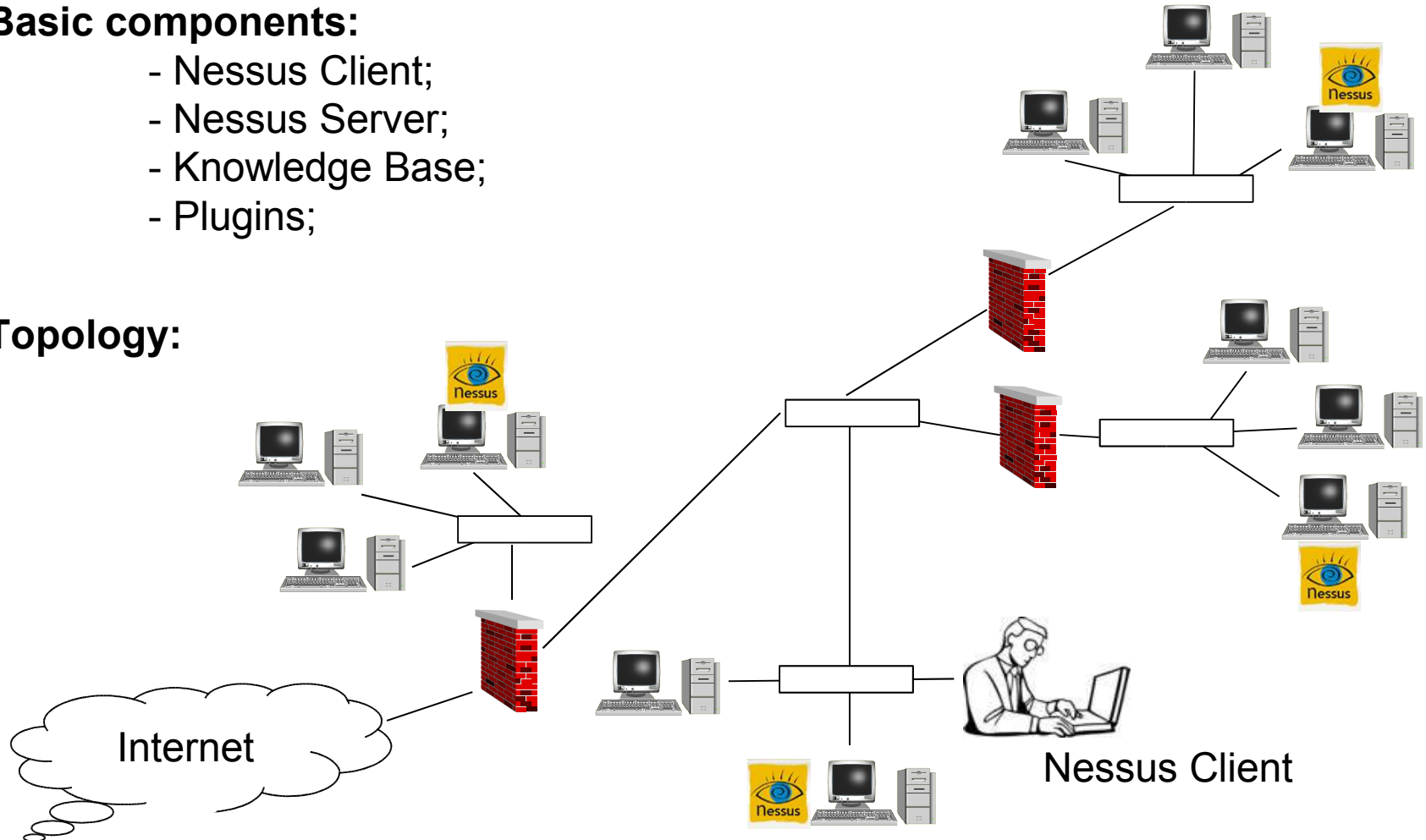
Nessus Open Source
Vulnerability Scanner Project

2.2 Vulnerability Scanning - Nessus Introduction

Basic components:

- Nessus Client;
- Nessus Server;
- Knowledge Base;
- Plugins;

Topology:



2.3 Vulnerability Scanning – Nessus Installation

Installing Nessus from source:

- Prerequisites Software:
 - GTK 1.2 (for Nessus GUI Client);
(<ftp://ftp.gimp.org/pub/gtk/v1.2>)
 - OpenSSL (for Client/Server communication):
(www.openssl.org)
- Download the latest version from a mirror:
(<http://ftp.nessus.org/nessus/nessus-2.2.2a/src/>)
 - nessus-libraries
 - libnasl
 - nessus-core
 - nessus-plugins

2.3 Vulnerability Scanning – Nessus Installation

```
tar -xzf nessus-libraries-2.2.2a.tar.gz
tar -xzf libnasl-2.2.2a.tar.gz
tar -xzf nessus-core-2.2.2a.tar.gz
tar -xzf nessus-plugins-2.2.2a.tar.gz
```

```
cd nessus-libraries
./configure --enable-cipher --with-ssl=/usr/lib/
make
su
make install
exit
```

(DO NOT RUN configure OR make AS ROOT)

```
echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/bin/X
11
export PATH=$PATH:/usr/local/bin
echo $PATH >>/etc/profile

grep /usr/local/lib /etc/ld.so.conf
echo "/usr/local/lib" >>/etc/ld.so.conf
ldconfig
```

2.3 Vulnerability Scanning – Nessus Installation

```
cd ../libnasl
./configure
make
su
make install
exit

cd ../nessus-core
./configure
make
su
make install
exit

cd ../nessus-plugins
./configure
make
su
make install
exit
```

2.3 Vulnerability Scanning – Nessus Installation

```
apt-get install nessusd
  - libnasl2
  - libnessus2
  - nessus-plugins
  - nessusd
  - openssl
```

```
apt-get install nessus
  - libgd-gif1
  - nessus
```

<code>/usr/sbin/nessusd</code>	Nessus Server
<code>/usr/sbin/nessus-****</code>	Nessus Server Tools
<code>/var/lib/nessus/CA</code>	CA- and Server certificate
<code>/var/lib/nessus/private/CA</code>	CA- and Server Private key
<code>/var/lib/nessus/users/<user></code>	Knowledge Base; User specific PlugIns
<code>/var/lib/nessus/plugins</code>	Plugins
<code>/var/log/nessus</code>	Logfiles of the Nessus Server
<code>/etc/nessus</code>	Configuration File of the Server
<code>/etc/init.d/</code>	Start Scripts
<code>/usr/bin/nessus</code>	Nessus Client
<code>/usr/bin/nessus***</code>	Nessus Client Utilities

2.3 Vulnerability Scanning – Nessus Installation

Create the Nessus Server Certificate:

```
nessus-mkcert
```

Create a user account:

```
nessus-adduser  
    <User Name>  
    --  
    --  
    default accept  
    <ctrl>d
```

```
-----  
accept|deny ip/mask  
default accept|deny  
-----  
alternative:  
nessus-mkcert-client  
-----
```

Update for the latest plugins

```
nessus-update-plugins
```

2.3 Vulnerability Scanning – Nessus Installation

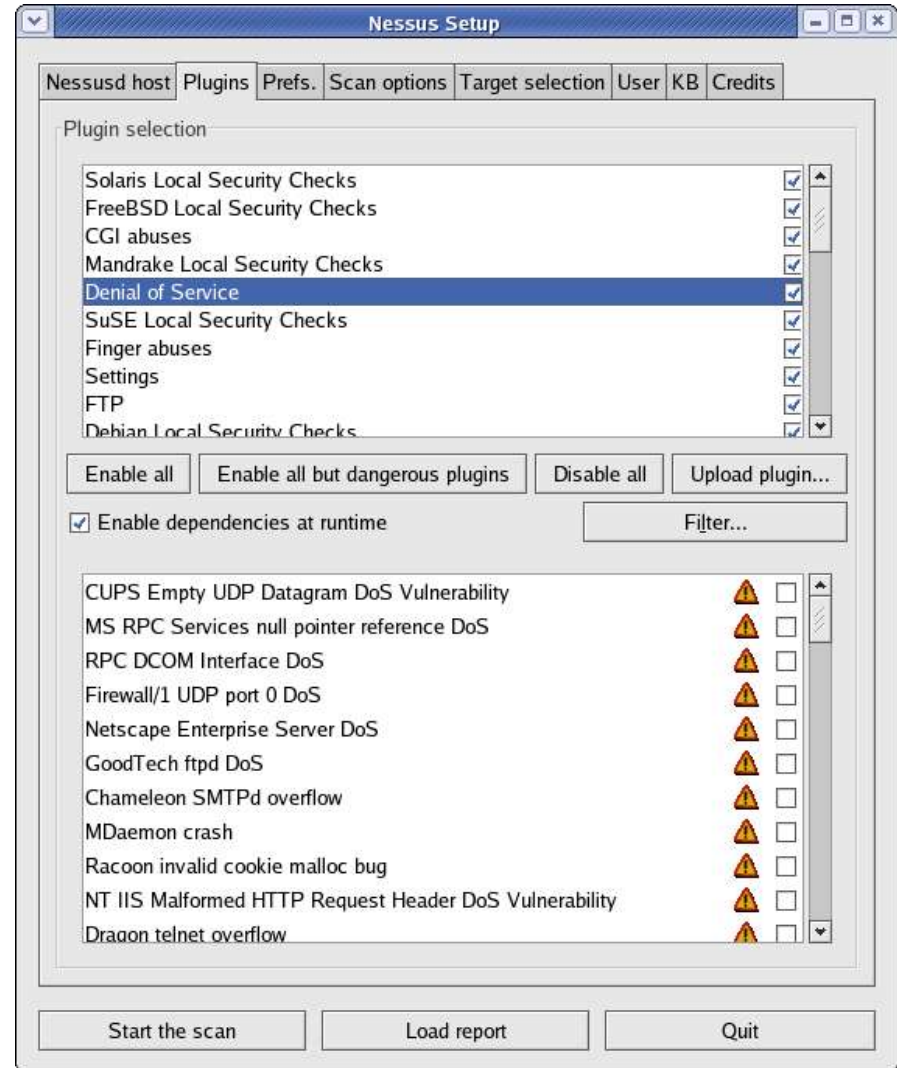
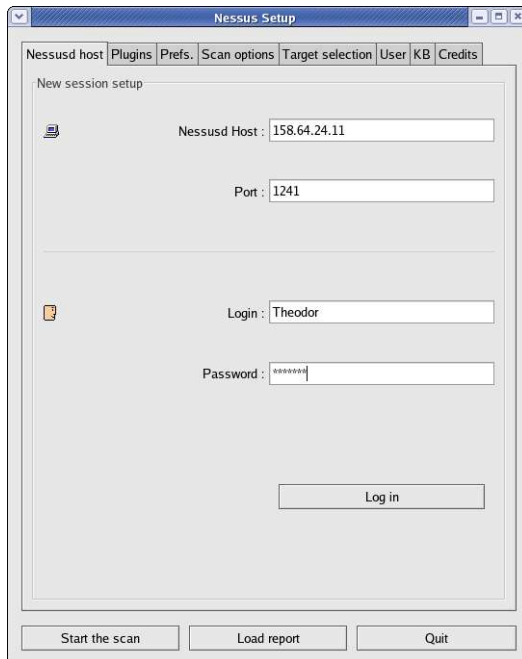
Start Nessus Server:

```
nessusd -D
```

Start Nessus Client:

```
nessus
```

Connect / Login:



2.4 Vulnerability Scanning – Nessus Plugins

Update Plugins:

- Direct Feed (commercial from Tanable Network Security);
- Registered Feed (free registration by eMail);
- GPL Feed (Plugins from the community);

working with Plugins:

- Plugins are organized into functional families;
- Selecting a family displays all of them;
- use checkbox to enable / disable;
- “Enable all” - turns on all the Plugins;
- “Enable all but dangerous”;
- “Disable all” - turns off all the Plugins;
- “Upload plugin” - brings in a new Plugin;

2.5 Vulnerability Scanning – NASL

Work with the official Plugin Page:

- www.nessus.org;
- Investigate Example Plugin “CGI abuse/cgiform” (ID = 10552);
- “Filter...” -> using the Plugin-Filter to search for specific Plugins;

NASL (Nessus Attack Scripting Language):

- Separate the scanning engine “nessusd” and “security checks”;
- provides modular any easy extensible architecture;
- First approach Plugins was shared libraries in “C”;
- > NASL

2.5 Vulnerability Scanning – NASL

Goals of NASL:

- each script must be self-contained (in one file);
- easy to use for the end-user;
- easy to learn for contributor;
- small memory footprint;
- designed for network security checks;
- strong security;
- easy to modify and extend;
- support multiple languages;

2.5 Vulnerability Scanning – NASL

Structure of a NASL Script:

- Section 1 (Script Description):
 - contains Code which is used by the Nessus engine:
 - Plugin Identifier;
 - short name of the Plugin;
 - dependencies;
 - controls how and when the Plugin is launched;
 - infos for the Client interface;
 - infos for the final report;
- Section 2 (Script Body):
 - test for a vulnerability and produce relevant data:
 - first Nessus run every Plugin once, to build a dependency tree;
 - provide a list of available Plugins to the Client;
 - during the scan each Plugin is launched once for each target;

2.5 Vulnerability Scanning – NASL

The Description Section:

- script_id(<number>)
- script_name(english:"<name>")
- script_description(english:"<description>")
- script_summary(english:"<name>")
- script_category(<category>)
(ACT_INIT, ACT_SCANNER, ACT_SETTINGS, ACT_GATHER_INFO,
ACT_ATTACK, ACT_MIXED_ATTACK, ACT_DESTRUCTIVE_ATTACK,
ACT_DENIAL, ACT_KILL_HOST, ACT_FLOOD, ACT_END)
- script_copyright(english:"<copyright>")
- script_family(english:"<family>")
(Backdoors, CGI abuses, Cisco, Denial of Service, FTP,)
- script_bugtraq_id(ID1, ID2, ...)
- script_cve_id("CVE-XXXX-YYYY", "CVE-XXXX-YYYY", ...)
- script_xref(name:<name>. value:<value>)
- script_timeout(timeout)
- script_dependencies("name1.nasl", "name2.nasl", ...)
- script_require_ports(port1, port2, ...)
- script_require_keys(key1, key2, ...)
- script_exclude_keys(key1, key2, ...)

2.6 Vulnerability Scanning – Knowledge Base

What is the Knowledge Base:

- a list of information about a host being tested;
- allows Plugins / Tests to share information;
- save the KB to disk for use in future scans;
- one Plugin can use data gathered by a previous running Plugin;
- decreasing the number of interactions with each host;
- making the development of plugins easier;
- every process receive its own copy of the current KB;
- 2 process running in parallel cannot communicate with each other;
- no LOCK / UNLOCK mechanisms required;
- path: `/usr/local/var/nessus/users/[username]/kbs/[target IP]`
- check "enable KB saving" to keep a copy of the KB after the scan is completed;

2.6 Vulnerability Scanning – Knowledge Base

Example:

```
1103817672 3 Launched/12288=1
xxxxxxxxxxx 3 Ports/tcp/22=1
xxxxxxxxxxx 3 Ports/tcp/25=1
xxxxxxxxxxx 1 Known/tcp/22=ssh
xxxxxxxxxxx 1 Services/ssh=22
1103821492 3 Ports/tcp/80=1
xxxxxxxxxxx 1 Known/tcp/80=www
xxxxxxxxxxx 1 Services/www=80
1103821557 1 SentData/11032/INFO/1=The following directories
were discovered:\n/cgi-bin, /icons, /manual, /perl, /
statistics, /stats\n\nWhile this is not, in and of itself, a
bug, you should manually inspect \nthese directories to ensure
that they are in compliance with company\nsecurity standards\n
```

Example CGI Test

this test must not run against a host, if there is no WWW service running

2.6 Vulnerability Scanning – Knowledge Base

Explanation:

1. Time in milliseconds since 1970
2. type
 - 1 ARG_STRING string
 - 2 ARG_PTR
 - 3 ARG_INT integer
 - 4 ARG_ARGLIST
 - 5 ARG_STRUCT
3. <name of KB entry> = <value>
4. new line \n

Caveats:

- services which are launched after the initial scan are missed in the future;
- access to the KB for others is dangerous;

2.6 Vulnerability Scanning – Knowledge Base

using the Knowledge Base

- enables Knowledge Base savings on the Nessus Server
 - start to write to disk
 - do not read the KB for tests
- Test all hosts
- Test hosts that have been tested in the past
- Tests hosts that never been tested
- Reuse the KB about hosts for test
 - enables reading from KB

the following checkboxes correspond directly to the Nessus plugin categories:

- ACT_SCANNER
- ACT_GATHER_INFO
- ACT_MIXED_ATTACK, ACT_ATTACK;
ACT_DESTRUCTIVE_ATTACK
- ACT_DENIAL, ACT_KILL_HOST, ACT_FLOOD

2.7 Vulnerability Scanning – Bring it together

1. Host Detection:

- goal, to identify all alive hosts
- 1th approach: do an ICMP Ping (Preferences)
- 2nd approach: do a TCP Ping (Preferences)
- 3rd approach: enable both
- 4th approach: disable both

2.7 Vulnerability Scanning – Bring it together

2. Port Detection:

- based on the list of alive host
- identify open ports
- results stored in the KB as “service/known“ entries
- detect the type of service behind each open port
- Plugins do the job
 - find_service.nes
 - find_service2_nasl
- using fingerprint techniques
- example: send “GET / HTTP/1.0” and examines the response
- send predefined strings like “help” and analyse the response
- report to KB: “Known/tcp/80 Service/www”
- this kind of information allows to start tests on non-standard ports;

2.7 Vulnerability Scanning – Bring it together

3. Information Gathering:

The Plugins in this category do:

- Service queries
- Application Fingerprinting
- Remote version Analysis

4. Vulnerability Fingerprinting:

- after finishing the information gathering process
- attack Plugins are classified:
 - ACT_ATTACK
try to circumvent defense without effect on the system
 - ACT_MIXED_ATTACK
if „save checks“ is off -> could damage the system
 - ACT_DESTRUCTIVE_ATTACK
just executed if „save checks“ is off

2.7 Vulnerability Scanning – Bring it together

5. Denial of Service Testing:

all tests must be passed step by step, to make it possible to say which test was successful

- ACT_DENIAL
 - send exploit to a service
 - try to connect again
 - if service down -> vulnerability reported
- ACT_FLOOD
 - send constant stream of data
 - could use large amounts of bandwidth
- ACT-KILL-HOST
 - always be executed at last;
 - try to kill the OS itself

2.8 Vulnerability Scanning – Scanning networks

- dont scan from one single point
- distributed scanning
- average of 3 - 5 high-risk vulnerabilities per host
- average of 10 low-risk vulnerabilities per host
- makes 750 high risk and 4.500 low risk alerts in a class C network
- first scan most critical systems only
- average of 3 - 5 high-risk vulnerabilities per host
 1. Centralized servers (DNS, mail, file, database)
 2. Financial servers and workstations
 3. Management servers and workstations
 4. Servers and workstations containing marketing data and plans
 5. Sales servers and workstations
 6. All remaining hosts
- Prior Notification of System Administrators is very important

2.8 Vulnerability Scanning – Scanning networks

- define goals, what you like to see
 - what can an employee do
 - what can somebody without username:password do
 - what will an insider see
 - what will an attacker from the internet see
 - what will somebody who compromise a system in the DMZ see
- make sure the network does not suffer from being scanned
 - unresponsive applications,
 - high CPUusage
 - high memory usage
 - traffic congestion.
- optimize the test for your network - mirror traffic with a sniffer
 - "Number of checks to perform at the same time"
 - "max_threads"
- use machines with known vulnerabilities to see the positive alert
- use honeypot with known vulnerabilities to test positive alerts
- set up a test-Lab to prepare testst and see what happens

2.8 Vulnerability Scanning – Scanning networks

- Trapps:

DHCP Environment

-> KB aout of Sync with the actual Network State

-> turn on the "Designate host by their MAC address"

-> only usefull on a LAN

Web Servers Load Balancing

2.9 Vulnerability Scanning – The Results

- viewing results using the Nessus GUI Client for X
- saving and exporting to other formats
- creating reports
- understanding reported vulnerabilities
- factors that can affect scanner output
 - Plugin selection
 - resolved dependencies
 - safe checks
 - ping the remote host
 - portscanner settings
 - valid credentials
- False Positives
 - What are False Positives
 - What is the problem with False Positives

3. User Mode Linux

3.1 Introduction

3.2 Installation

3.3 Running

3.1 User Mode Linux - Introduction

Running Linux inside itself:



- patching the kernel;
- running the kernel as a standard software inside user-space;
- full powered kernel with scheduler and memory management;
- need the Host kernel for hardware support;
- from host point of view its a normal process;
- supports block-devices, networking and X;

3.1 User Mode Linux - Introduction

Applications:

- Virtual hosting
 - build multiple virtual systems on a single physical host;
 - used for workshops;

- Testing
 - `rm -rf /`
 - test new kernel parameters

- Sandbox / Jail
 - Honeypot
 - test new applications

- Virtual networking / routing
- Software development

3.1 User Mode Linux - Introduction

Conditions:

- Kernel
 - patched and self compiled vanilla kernel;
 - download precompiled kernel from www.usermodelinux.org
 - installation from distribution;

- Root filesystem
 - a complete filesystem (/etc /bin /var etc.);
 - packed into an image file;
 - find filesystems at the UML- Website;
 - install from distribution;
 - create your own filesystem;

3.2 User Mode Linux - Installation

**user-mode-linux.sourceforge.net/
packages.debian.org/user-mode-linux**

```
apt-get install user-mode-linux  
  /usr/lib/uml/*  
  /usr/bin/linux  
  /root/.uml
```

```
apt-get install uml-utilities  
  /usr/sbin/tunctl  
  /usr/lib/uml/*  
  /usr/bin/uml_*  
  /var/run/uml-utilities/*  
  /etc/default/uml-utilities  
  /etc/network/*/uml-utilities  
  /etc/init.d/uml-utilities  
  /etc/passwd /etc/shadow /etc/group
```

3.2 User Mode Linux - Installation

Manual installation

1. download the latest UMP-Patch
2. download the matching kernel
3. make directory and unpack the kernel into it

```
mkdir ~/uml
cd ~/uml
tar -xjvf linux-2.4.0-prerelease.tar.bz2
```
4. apply the patch

```
cd ~/uml/linux
bzcat uml-patch-2.4.0-prerelease.bz2 | patch -p1
```
5. Run your favorite config

```
make xconfig ARCH=um / config ARCH=um / menuconfig ARCH=um
make linux ARCH=um
make modules ARCH=um
```

3.3 User Mode Linux - Running

Create a Swap Partition

```
# dd if=/dev/zero of=swap_fs seek=256 count=1 bs=1M  
# mkswap -f swap swap_fs
```

Starting UML:

```
# linux ubd0=Debian-3.0r0.ext2 ubd1=swap_fs umid=Debian
```

