# COMP 4670

# Tutorials

## Network Attacks Part 3

Dr. Sherif Saad

# Agenda

- Network Mapping and Scanning.
- Deep Packet Inspections.
- Network Vulnerability Assessment.

# Network Mapping and Scanning

# NMAP Scanning Options

**NMAP** is a network mapping and scanning tool that could detect **live hosts**, **open ports**, **network services**, **OS** fingerprint and detect **network topology**.
There are several scanning options in NMAP:

1. Intense scan
2. Intense scan plus UDP
3. Intense scan plus all TCP ports
4. Intense scan, no ping
5. Ping scan
6. Quick scan plus
7. Quick traceroute
8. Regular scan
9. Slow Comprehensive

# Options in Nutshell

## Intense Scan

- **Command:** `nmap -T4 -A -v <target>`
- This scan has a reasonable run time and it scans known TCP ports only. It attempts to detect the OS type and the version of the running services.

## Intense Scan plus UDP

- **Command:** `nmap -sS -sU -T4 -A -v <target>`
- Perform regular intense scan plus scanning UDP ports (-sU) and to use SYN packets for TCP ports (-sS)

# Options in Nutshell

## Intense Scan, all TCP ports

- **Command:** `nmap -p 1-65535 -T4 -A -v <target>`
- This scan is a regular TCP intense scan but it scan all the possible ports (1-65535)

## Intense Scan, no ping

- **Command:** `nmap -T4 -A -v -Pn <target>`
- Assume that the target is up and that the target blocks ping request (e.g. firewall or packet filtering)

# Options in Nutshell

## Ping Scan

- **Command:** **nmap -sn  <target>**
- Do only a ping scan (ICMP echo request) there is no port scan.

## Quick Scan

- **Command:** **nmap -T4 -F <target>**
- Limit your scan to the top most common TCP ports (e.g. 22, 25, 21, 80, etc). It does not attempt to detect OS

# Options in Nutshell

## Quick Scan Plus

- **Command:** `nmap -sV -T4 -O -F -version-light <target>`
- It is a quick scan but will try to detect OS version and services type.

## Quick traceroute

- **Command:** `nmap -sn -traceroute <target>`
- Use this scan to determine hsts and routers in the target network.

# Options in Nutshell

## Regular Scan

- **Command: nmap <target>**
- TCP SYN scan for the 1000 common TCP ports and Ping to detect live host.

# Options in Nutshell

## Slow Comprehensive Scan

- **Command: nmap -sS -sU -T4 -A -v -PE -PP -PS80,443 -PA3389 -PU40125 -PY -g 53 -script “default or (discovery and safe)”<target>**
- This is scan s mainly for host detection assuming great effort has been done to hide live hosts. It is an intense scan plus UDP and other options. It tries to use TCP, UDP and SCTP protocols to detect live hosts.
- If the host is live it will try its best to find the host OS and running services. It also claim it is a DNS server (src port 53)

# Some NMAP Scanning Options

- -T4:
  - To control how fast is the scan.  It ranges from 0-5, where 5 is the fastest and 0 is the slowest. (0: paranoid 1: sneaky 2: polite 3: normal 4: aggressive 5: insane)
  - 1-2 being used for IDS evasion, 3 is the default and 4-5 is really quick scans.
- -A:
  - Instruct NMAP to try to detect the host OS version, traceroute  and services version
- -v:
  - Increased verbosity. This will give your extra information in the data outputted by Nmap.
- -sS:
  - TCP SYN connect scan
- -sU:
  - UDP scan

# Some NMAP Scanning Options

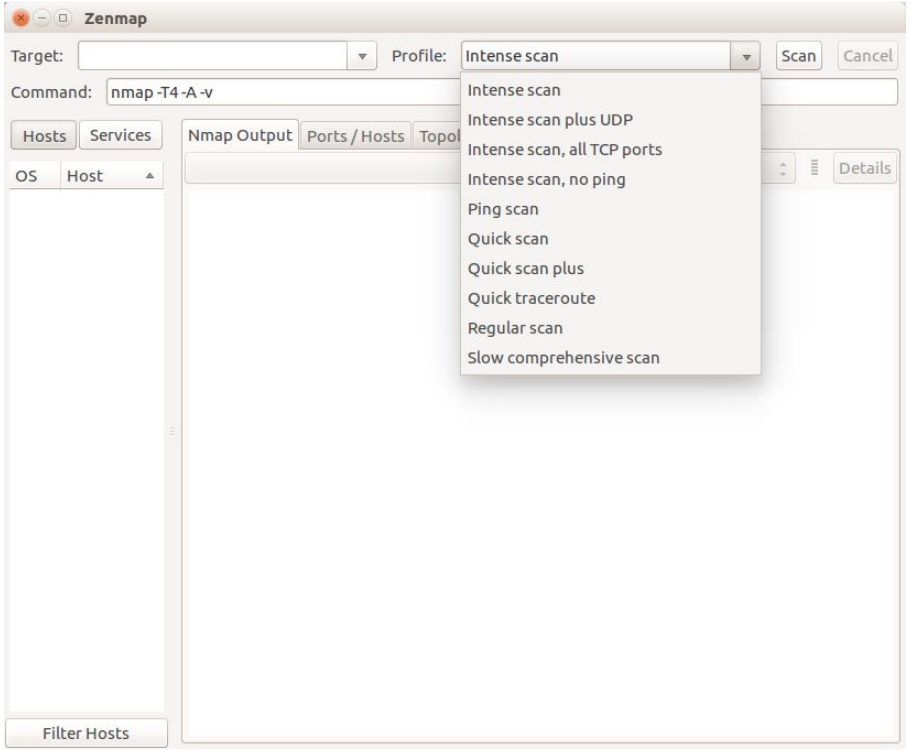
- -sN:
  - TCP Null scan, sends TCP packets with none of the TCP flags set in the packet. If the scan is returned a RST packet it means the port is closed, however if nothing is returned it is either filtered or open
- -sV:
  - Instruct the scan to try to detect the service version and type by sending additional packets to the open ports.
- -version-light:
  - Less intensive service probing. It will try to probe the service using best guesses
- -p:
  - To define a set of port to scan and probe
- -g:
  - Define what source port you want NMap to use (try to evade IDS and firewall)

# NMAP in actions

You should use the GUI version of Nmap.

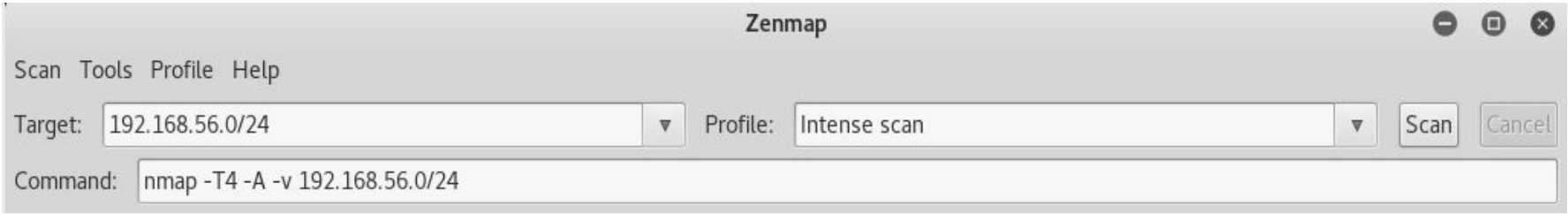
The UI allows specifying the target.

Select the scan type and edit the scan command if required.



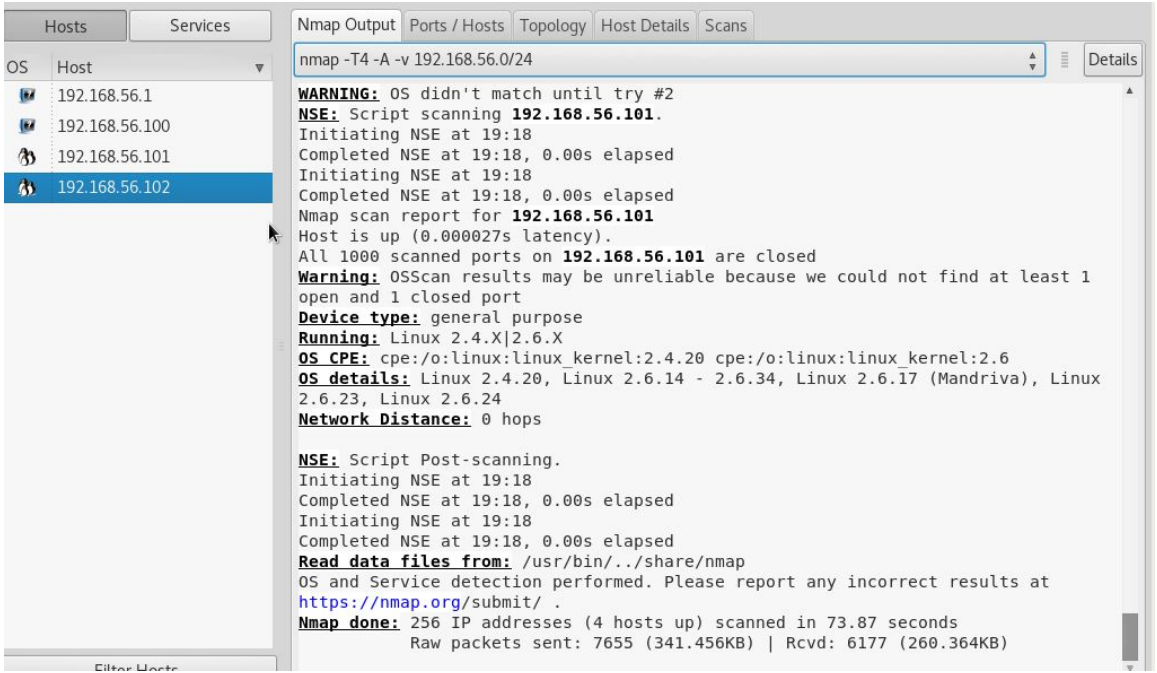
# NMAP Scan Example

We will scan the subnet **192.168.56.0/24** using an intense scan



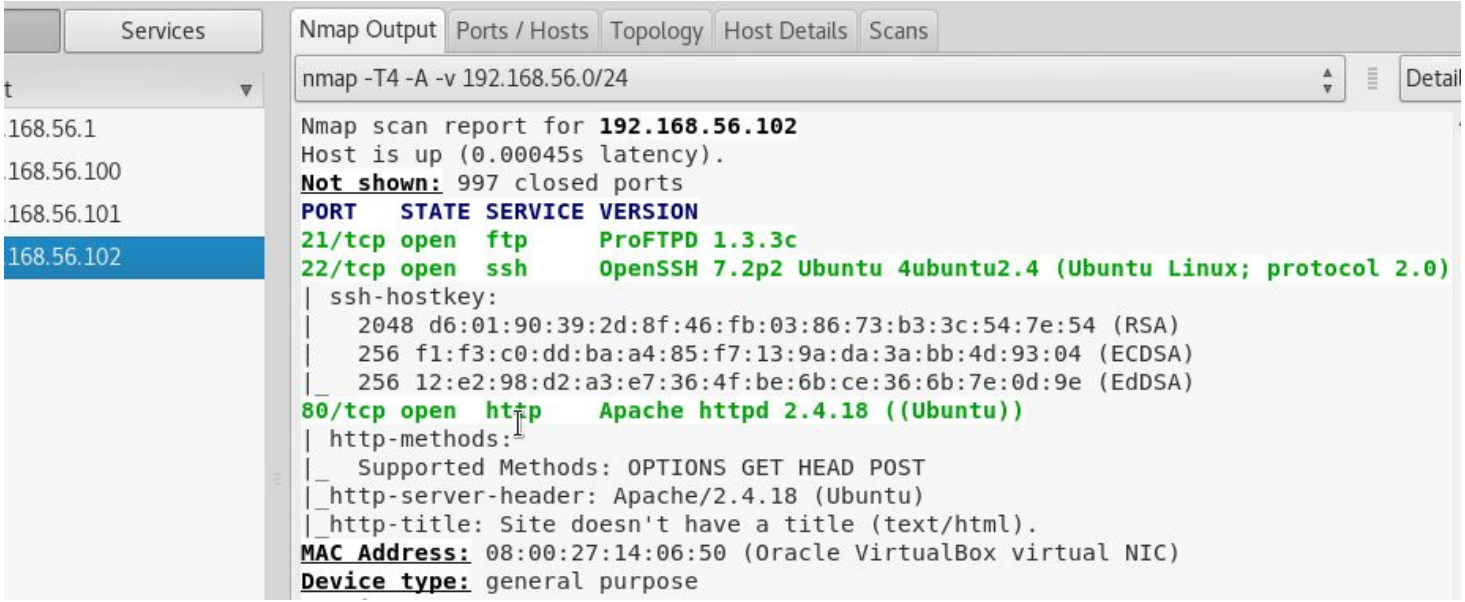
# NMAP Scan Example

When NMAP complete the scan. We will get a list of live host on the target subnet



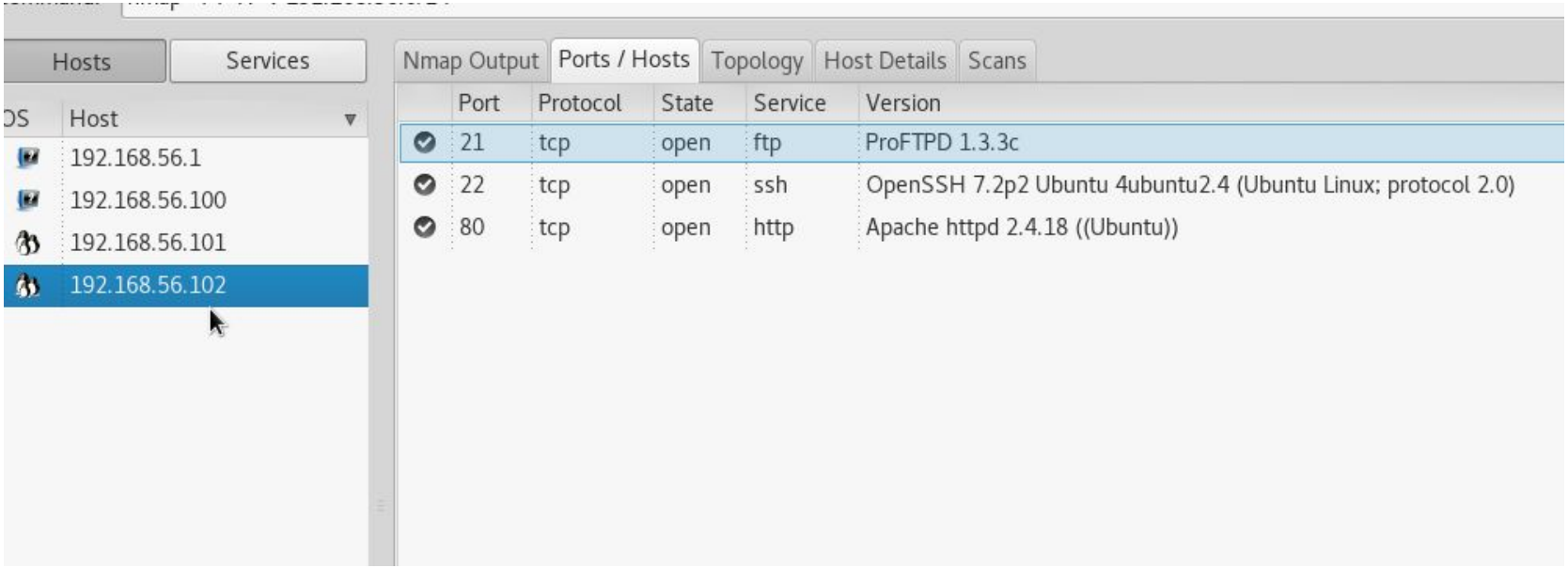
# NMAP Scan Example

When NMAP complete the scan. We will get a list of running service on every host



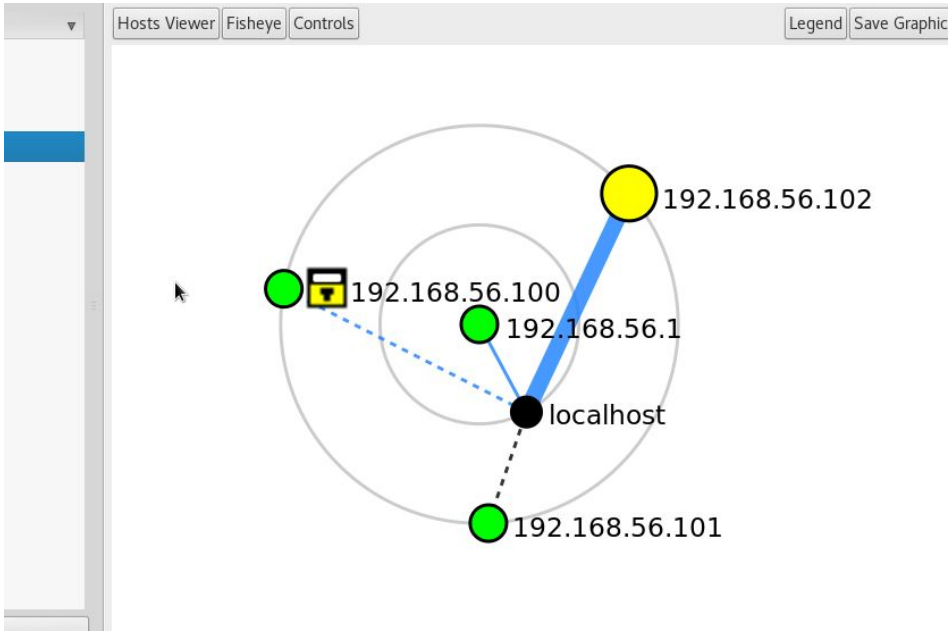
# NMAP Scan Example

When NMAP complete the scan. We will get a list of running service on every host, in addition the open/filtered/closed ports on every live host



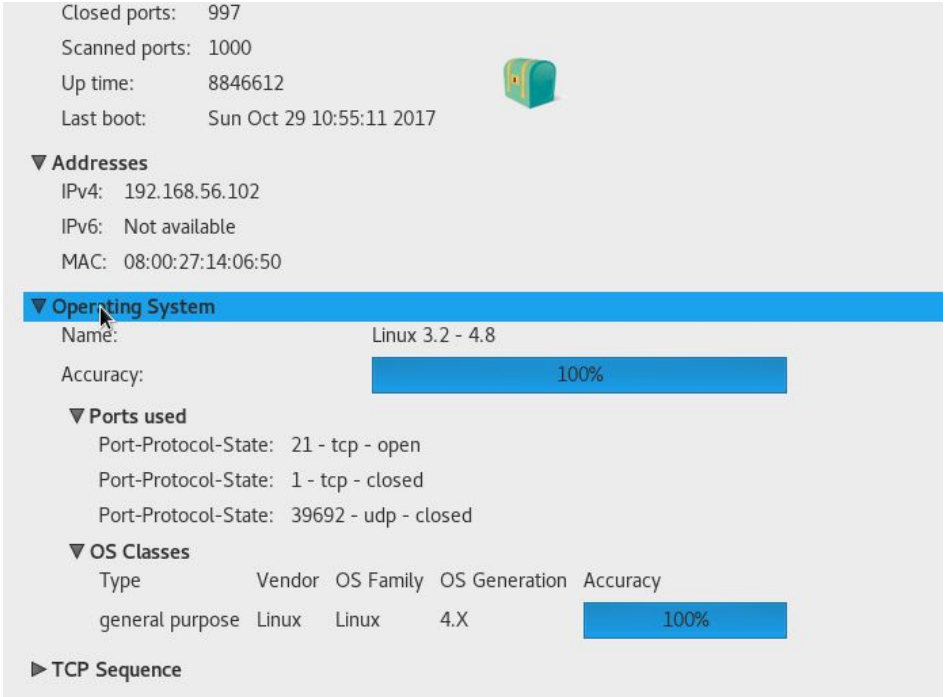
# NMAP Scan Example

When NMAP complete the scan. We will get the network topology (e.g. live host and how many hop they are far from the scanning machine (the machine that run NMAP)



# NMAP Scan Example

In addition if possible we will get OS version for each live node|host in the target network



# NMAP Scan Example

We could also check other scanning options like Intense scan all TCP post

# NMAP Scan Example

We could also check other scanning options like Intense scan all TCP post

```
Nmap scan report for 192.168.56.255 [host down]
Initiating SYN Stealth Scan at 19:35
Scanning 3 hosts [65535 ports/host]
Discovered open port 22/tcp on 192.168.56.102
Discovered open port 80/tcp on 192.168.56.102
Increasing send delay for 192.168.56.1 from 0 to 5 due to 14 out of 34 dropped
probes since last increase.
Increasing send delay for 192.168.56.102 from 0 to 5 due to 15 out of 36 dropped
probes since last increase.
Discovered open port 21/tcp on 192.168.56.102
Increasing send delay for 192.168.56.102 from 5 to 10 due to 12 out of 29
dropped probes since last increase.
Increasing send delay for 192.168.56.1 from 5 to 10 due to 48 out of 119 dropped
probes since last increase.
SYN Stealth Scan Timing: About 0.84% done
SYN Stealth Scan Timing: About 1.69% done; ETC: 20:35 (0:59:11 remaining)
SYN Stealth Scan Timing: About 3.64% done; ETC: 20:33 (0:56:06 remaining)
SYN Stealth Scan Timing: About 6.54% done; ETC: 20:31 (0:53:07 remaining)
```

# NMAP Additional Resources

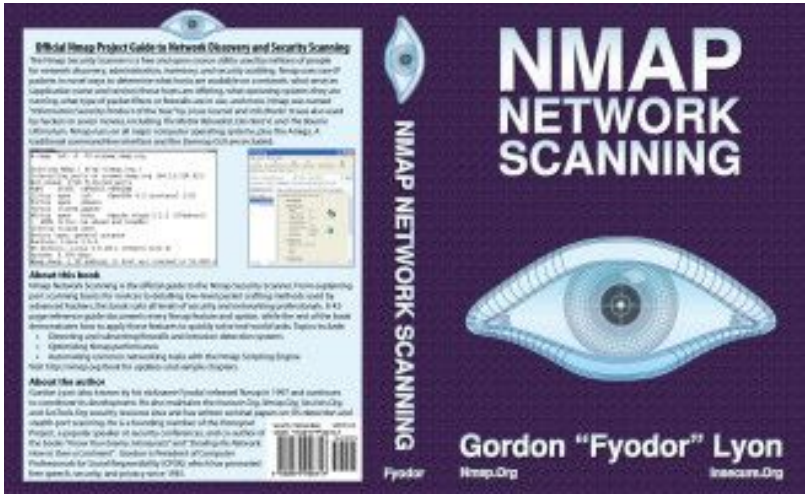
The Official Nmap Project Guide to Network Discovery and Security Scanning

By Gordon “Fyodor” Lyon

Book URL: http://nmap.org/book/

ISBN: 978-0-9799587-1-7

ISBN-10: 0-9799587-1-7



# Deep Packet Inspection

# What is Deep Packet Inspection (DPI)?

Deep Packet Inspection is also known as **complete packet inspection**.

A network traffic analysis technique that applies **packet filtering** to inspects the **data part** of the packet in addition to the packet header.

Deep Packet Inspection operates at the **application layer** of the TCP/IP stack to analyze the data in packets to identify protocol noncompliance, intrusions, spam, viruses or other predefined security flags.

Deep Packet Inspection is the combines techniques from network intrusion and prevention systems.

# Why Deep Packet Inspection?

DPI may be used by organizations for **Data Leak Prevention** (DLP). When an employee tries to send an email that contains confidential organization files

Deep Packet Inspection could help in detecting **advanced persistent threat** (APT).

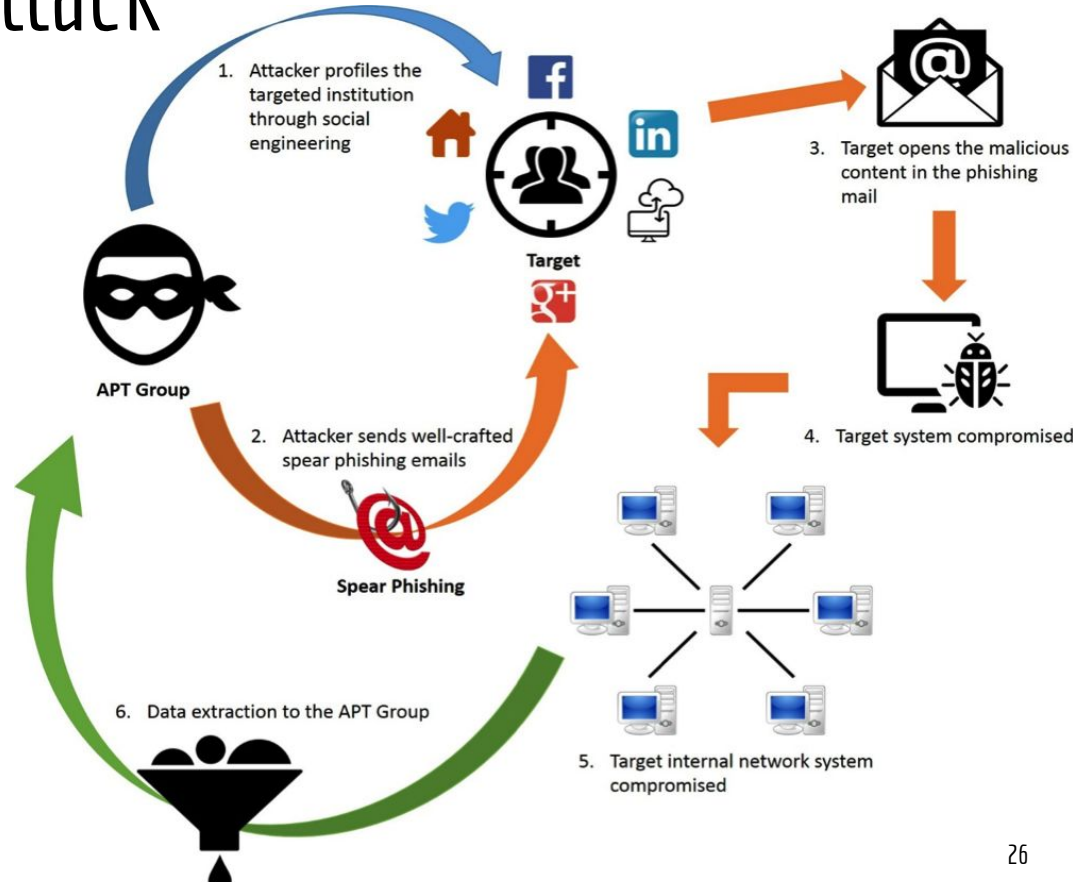
ART or **advanced persistent threat** is a network attack in which an unauthorized person or a group gains access to a network and stays there undetected for a long period of time.

# The Anatomy of APT Attack

The APT attack consist of 6 steps grouped into three main phases:

1. Infiltration
2. Expansion
3. Sabotage

DPI can detect the expansion and the sabotage by looking at the network traffic.



# Deep Packet Inspection with Wireshark


Wireshark is an **open source** tool for analyzing network traffic.

Wireshark enables network traffic **profiling** and **inspection**.

Wireshark is also a **network traffic sniffer** that can capture and record network packets over wired or wireless networks.

The best known packet analyzer and network sniffer that is commonly used by government agencies, enterprises, educational institutions.

Started by Gerald Combs in 1998.

# Why using Wireshark (packet sniffer)?

Provide **network traffic insights** by enabling data in motion recording and monitoring.

Gather and report **network statistics** about network usage

**Debug** and **troubleshooting** network problems and protocol implementation and configuration issue.

Detect **network intrusion** and **misuse** by internal and external users

Help in web filters, spam filters, and provide information to understand and inspect network intrusion and other malicious network activities.

# Wireshark (packet sniffer) Darkside

Could be used to execute man-in-the-middle, and packet injection attacks.

Gather network insights to design and carry out network attacks

Reverse engineer proprietary protocols used over the network.

Spy on other network users to collect sensitive information, such as login details or user cookies.

Could be used to hijack and compromise SSL/TLS communication depending upon encryption being used.

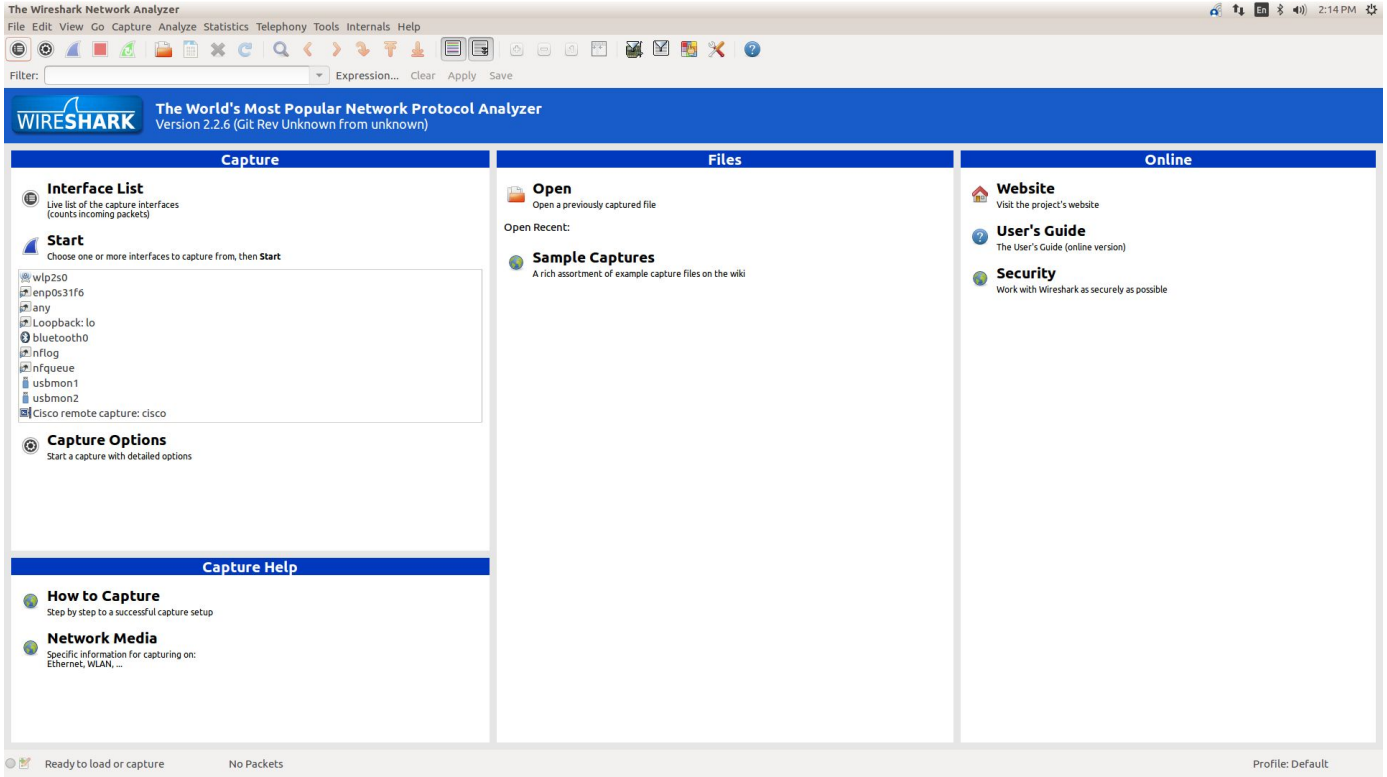
# Wireshark

Wireshark runs on almost all popular **platforms**. It runs on Windows, Linux, and macOS.


Wireshark is already installed on **Kali Linux** but if you prefer to use it on any other platform you could easily download the most recent version from https://www.wireshark.org/

**Note:** installing wireshark on your machine at work could be a violation to your company or organization policies.

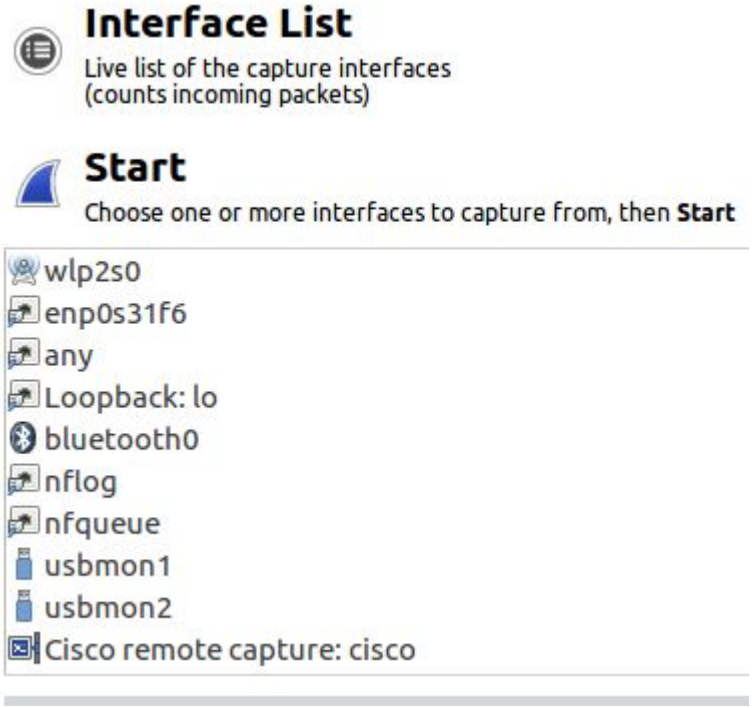
# Starting Wireshark



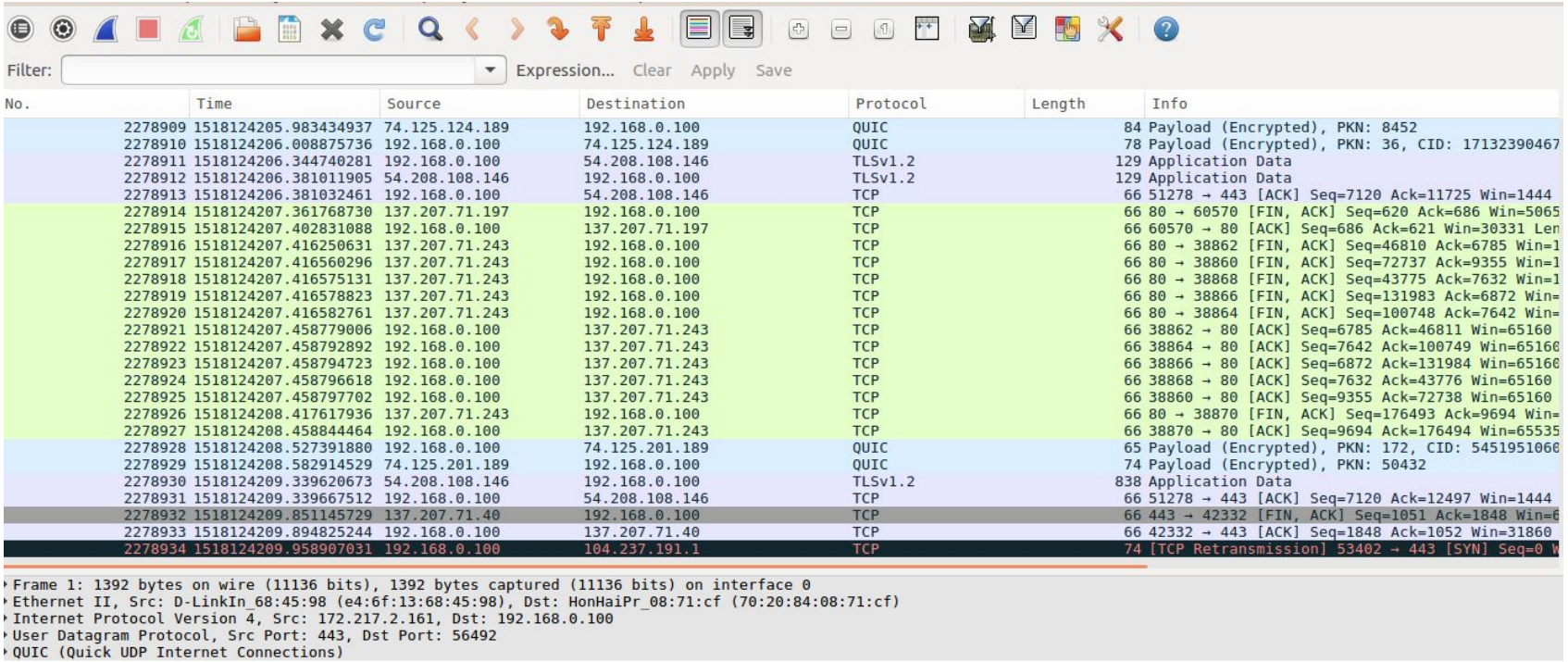
# Live Packet Capturing with Wireshark

To start capture live packets or live network traffic, you need to select the interface that Wireshark will start listening

If you can not see any interface. Then most likely you are running Wireshark with none-root privilege on Linux like OS or with a none-administrator account on Windows.



# Live Packet Capturing with Wireshark



# Wireshark In Nutshell

Simply select the network interface you want to monitor.

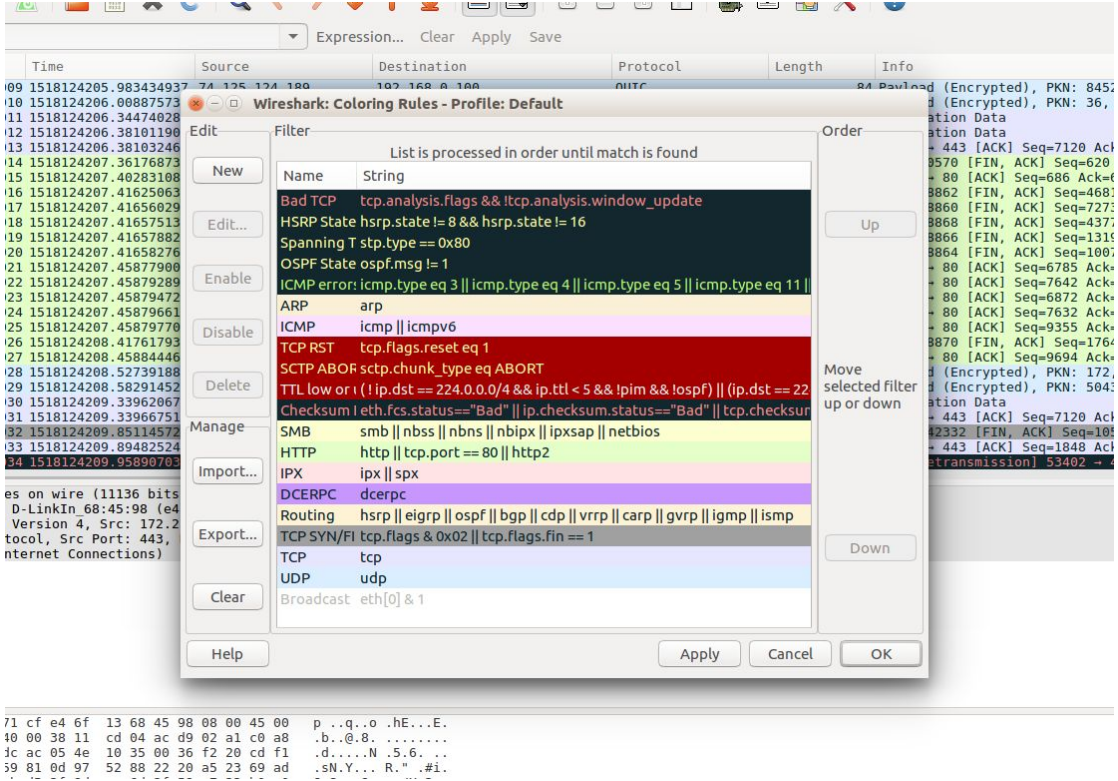
Use the toolbar on the top to start | stop| resume capture

Save the captured packets into a file (pcap file)

Open and load a file with captured packets.

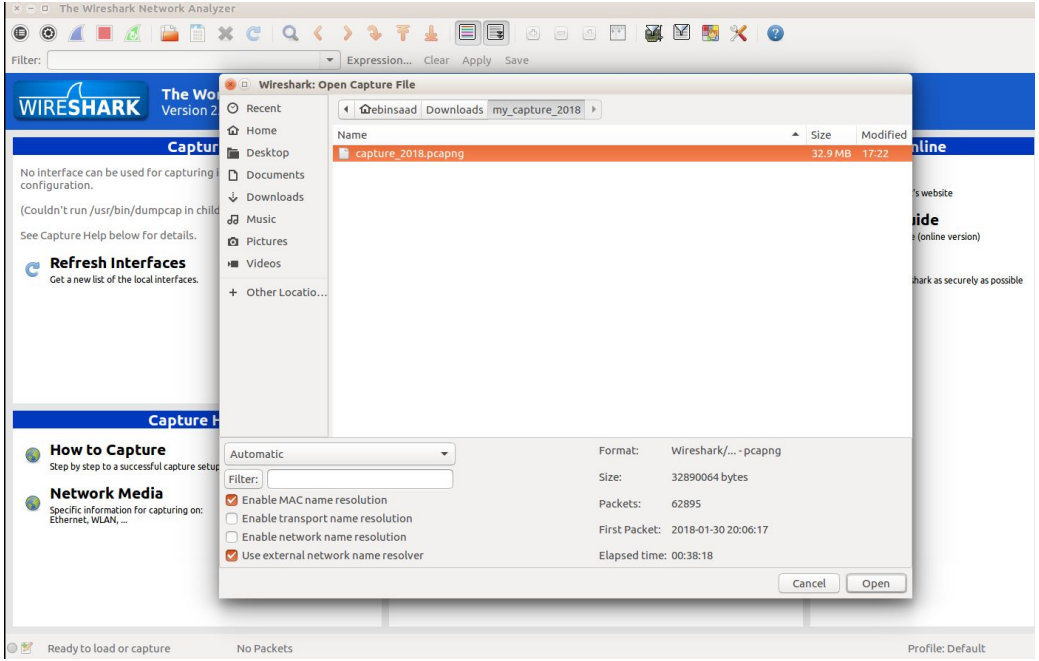
Inspect and analyze packets and traffic using powerful wireshark filters or write Lua script to automate complex analysis tasks.

# Colors in Wireshark



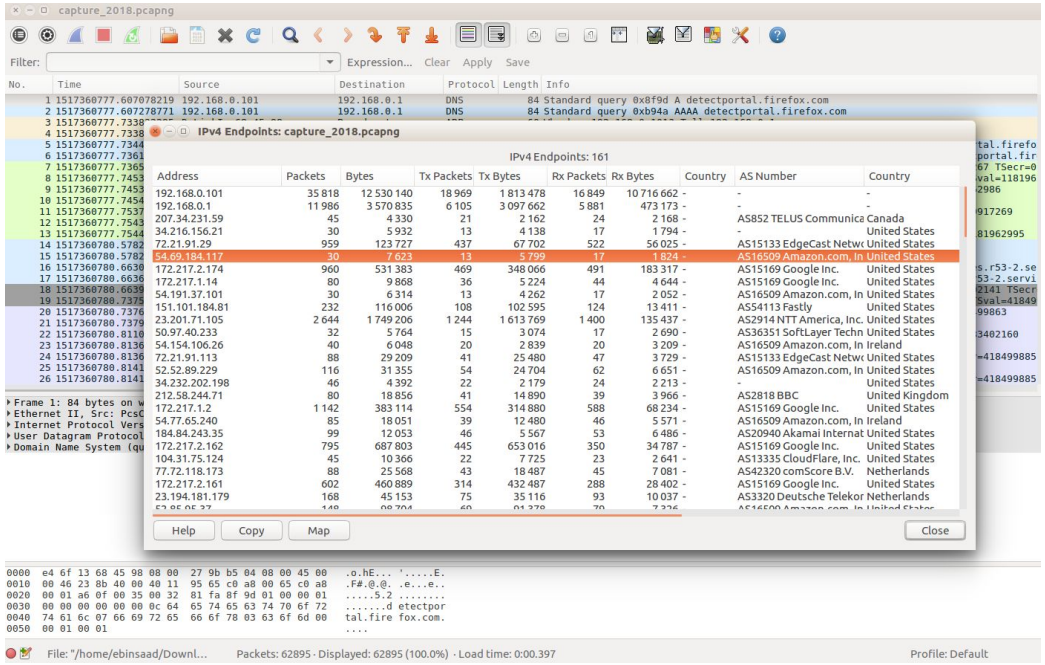
# Open a saved pcap file

From the file menu select open menu item



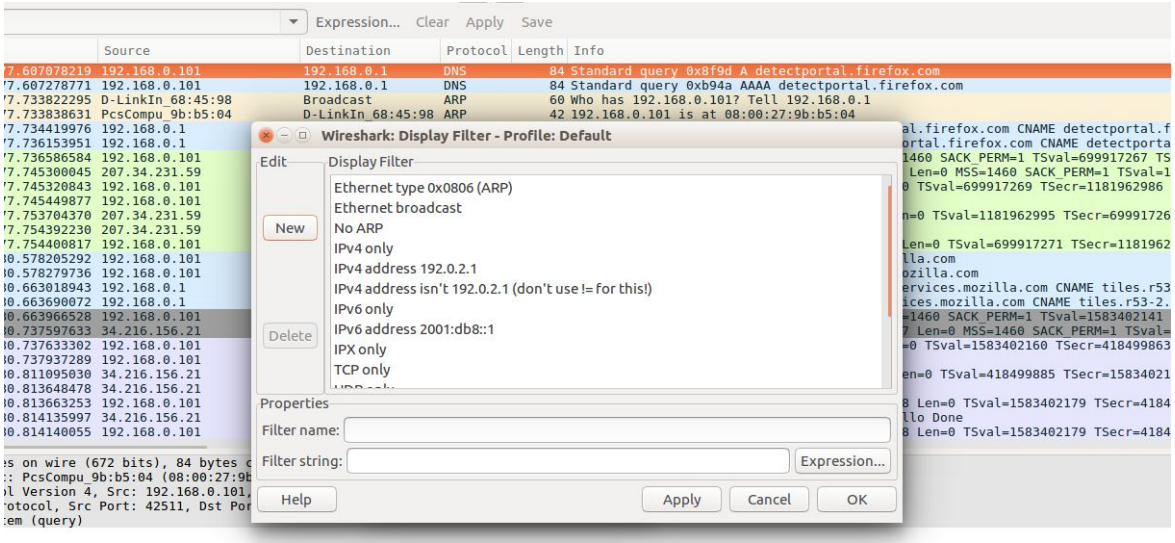
# Exploring PCAP file with Wireshark

To list of all IP addresses that were seen the network during the capture. Go to **"Statistics -> Endpoints"** and click on the IP tab.



# Wireshark Filters

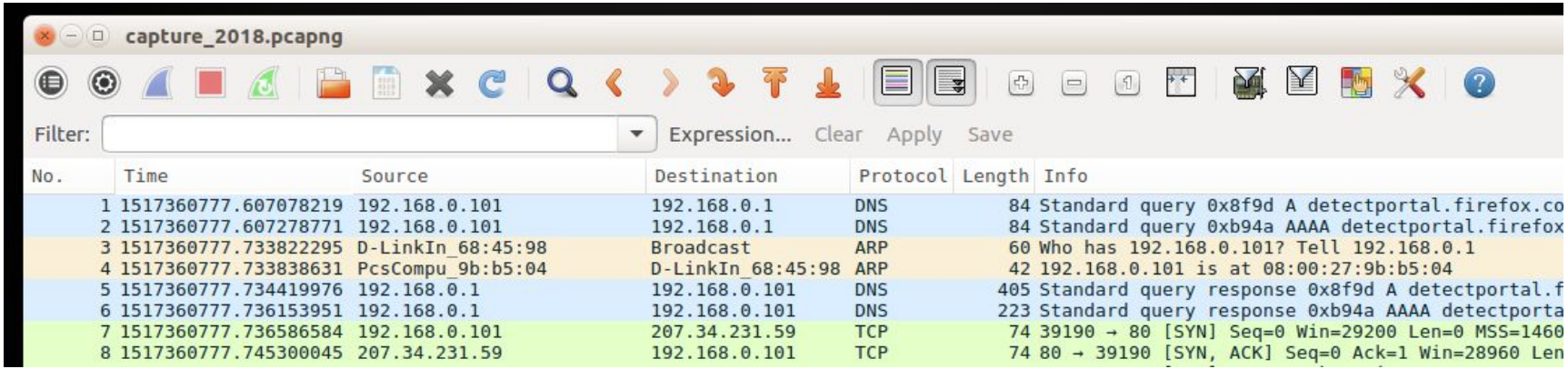
To list of all IP addresses that were seen the network during the capture. Go to **"Analyze -> Display Filters "** and click on the IP tab.



# Working with Wireshark Filters

Wireshark filters provide a powerful tool for network traffic inspections and analysis.

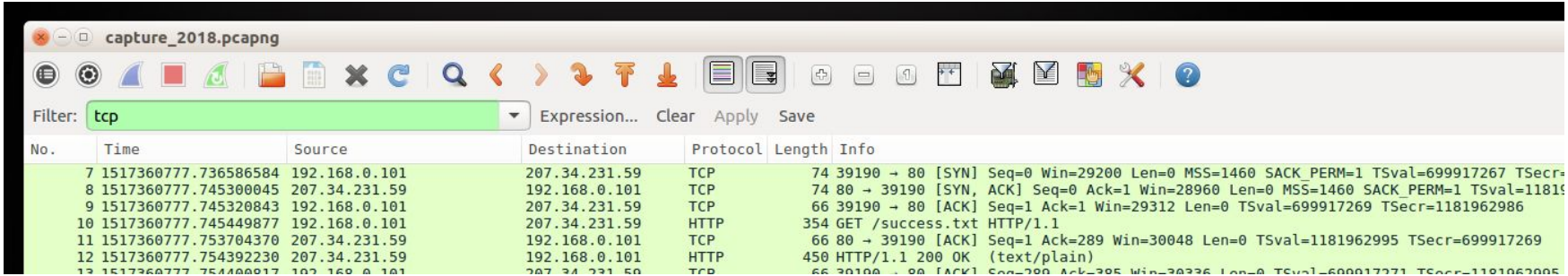
We could design simple and complex query to filter the captured traffic and focus our analysis  on specific IP address, port, protocol, network interface, etc




| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 1517360777.607078219 | 192.168.0.101 | 192.168.0.1 | DNS | 84 | Standard query 0x8f9d A detectportal.firefox.co |
| 2 | 1517360777.607278771 | 192.168.0.101 | 192.168.0.1 | DNS | 84 | Standard query 0xb94a AAAA detectportal.firefox |
| 3 | 1517360777.733822295 | D-LinkIn_68:45:98 | Broadcast | ARP | 60 | Who has 192.168.0.101? Tell 192.168.0.1 |
| 4 | 1517360777.733838631 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | 192.168.0.101 is at 08:00:27:9b:b5:04 |
| 5 | 1517360777.734419976 | 192.168.0.1 | 192.168.0.101 | DNS | 405 | Standard query response 0x8f9d A detectportal.f |
| 6 | 1517360777.736153951 | 192.168.0.1 | 192.168.0.101 | DNS | 223 | Standard query response 0xb94a AAAA detectporta |
| 7 | 1517360777.736586584 | 192.168.0.101 | 207.34.231.59 | TCP | 74 | 39190 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 |
| 8 | 1517360777.745300045 | 207.34.231.59 | 192.168.0.101 | TCP | 74 | 80 → 39190 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len |

# Working with Wireshark Filters

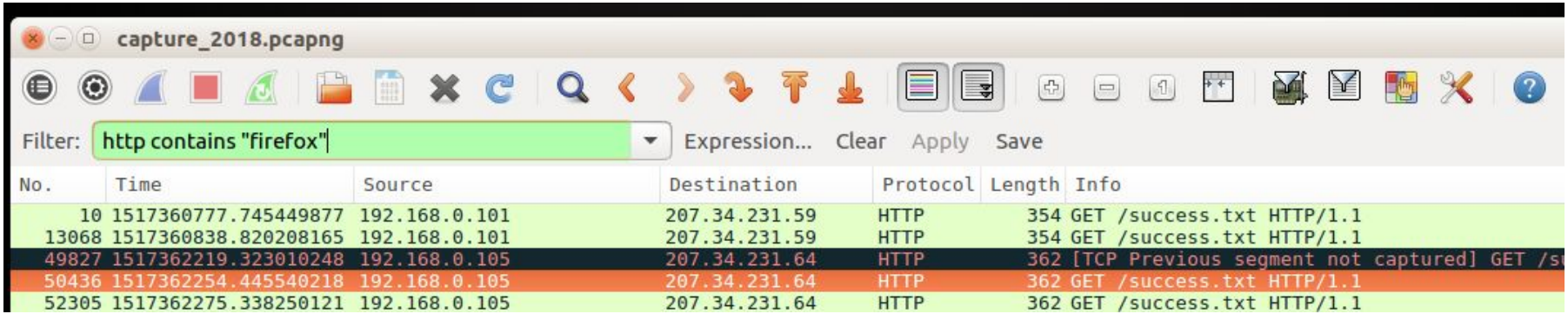
Display on TCP Packets, we simply type **tcp** in the filter textbox and hit enter. Note that wireshark filter support auto-complete



# Working with Wireshark Filters

Let us try another option let us display HTTP traffic contains specific text

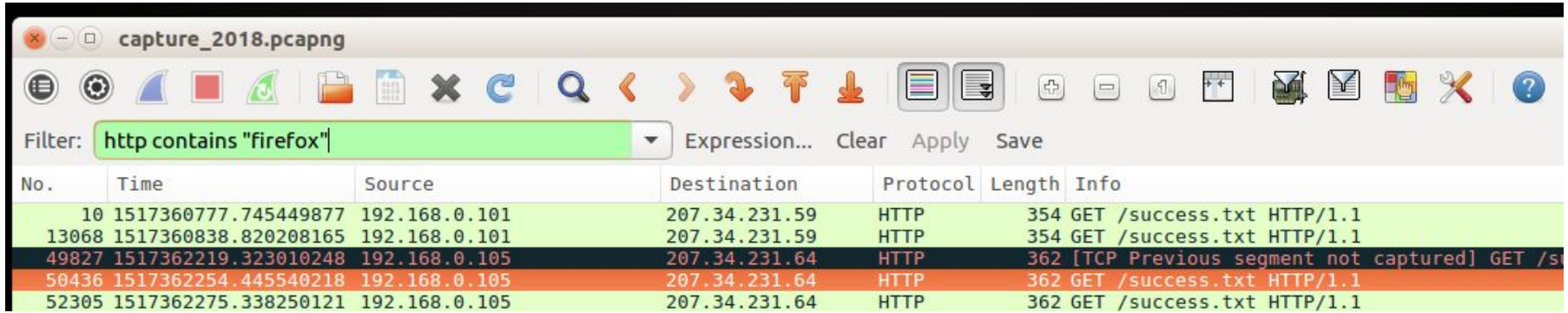
**`http contains "firefox"`**



# Working with Wireshark Filters

We can also filter based on IP addresses. For example display only the traffic between two IP addresses.

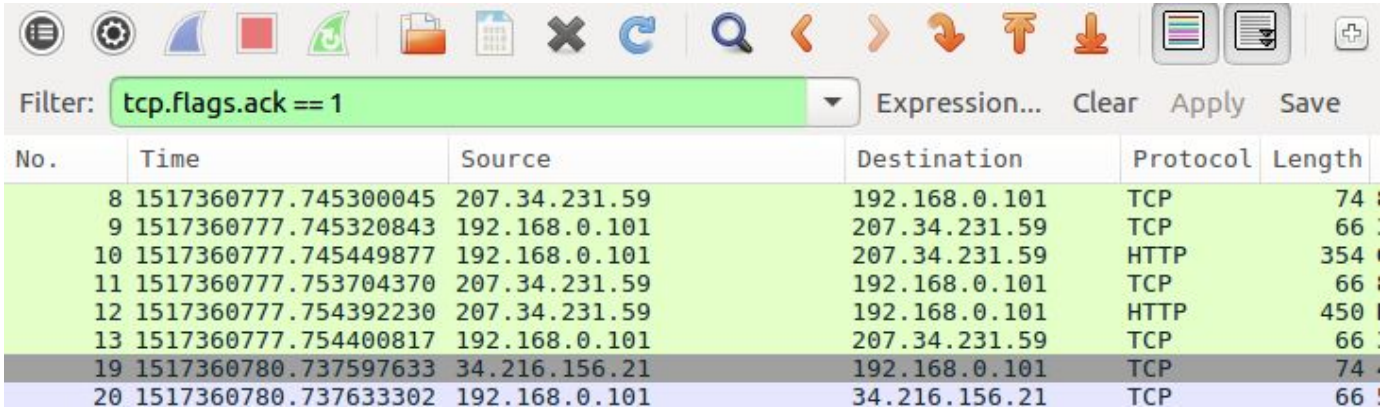
```
ip.src_host==192.168.0.101 and ip.dst_host==207.34.231.59
```



# Working with Wireshark Filters

We can also filter based on **protocol options**. For example let us display only the tcp packets where the ack flag is set to 1

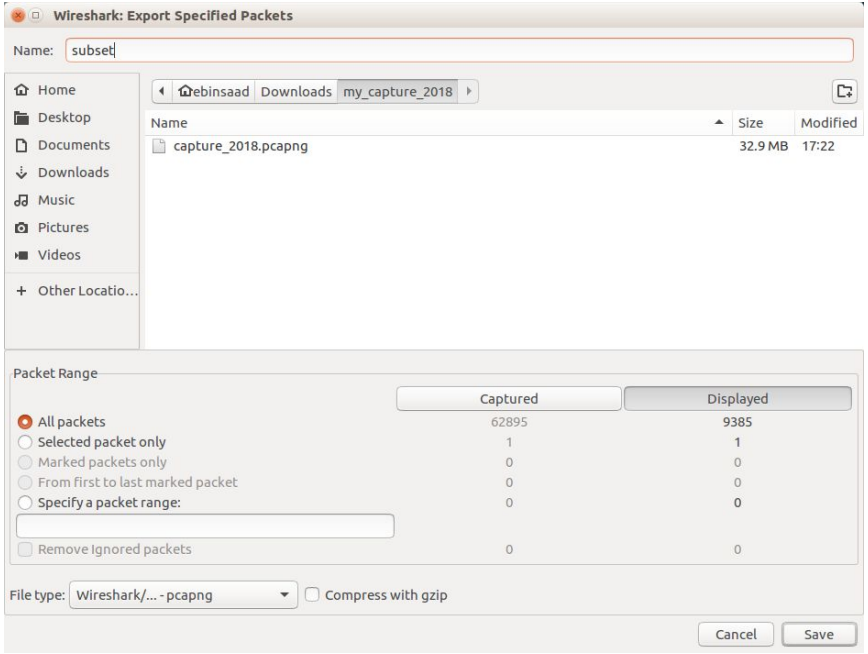
**`Tcp.flags.ack == 1`**




| No. | Time | Source | Destination | Protocol | Length |
|---|---|---|---|---|---|
| 8 | 1517360777.745300045 | 207.34.231.59 | 192.168.0.101 | TCP | 74 |
| 9 | 1517360777.745320843 | 192.168.0.101 | 207.34.231.59 | TCP | 66 |
| 10 | 1517360777.745449877 | 192.168.0.101 | 207.34.231.59 | HTTP | 354 |
| 11 | 1517360777.753704370 | 207.34.231.59 | 192.168.0.101 | TCP | 66 |
| 12 | 1517360777.754392230 | 207.34.231.59 | 192.168.0.101 | HTTP | 450 |
| 13 | 1517360777.754400817 | 192.168.0.101 | 207.34.231.59 | TCP | 66 |
| 19 | 1517360780.737597633 | 34.216.156.21 | 192.168.0.101 | TCP | 74 |
| 20 | 1517360780.737633302 | 192.168.0.101 | 34.216.156.21 | TCP | 66 |

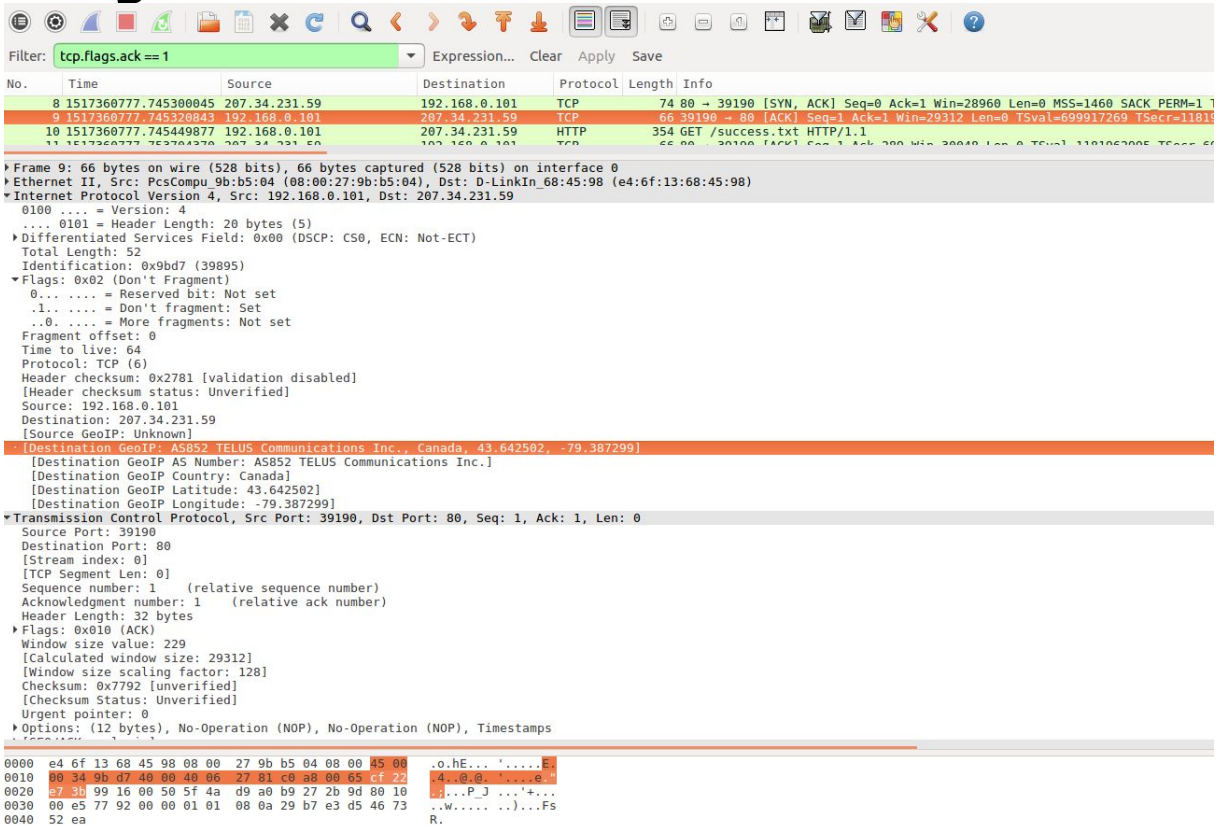
# Working with Wireshark Filters

We could save the filter results into a file. **Select File -> Export Specified packets**



# Wireshark Exploring Packet Details

When you select a **packet** you can see the **details** of the packet .



# Deep Packet Inspection with Wireshark

Download the file **capture_2018.pcapng** form the course site on blackboard.

Open the file using wireshark.

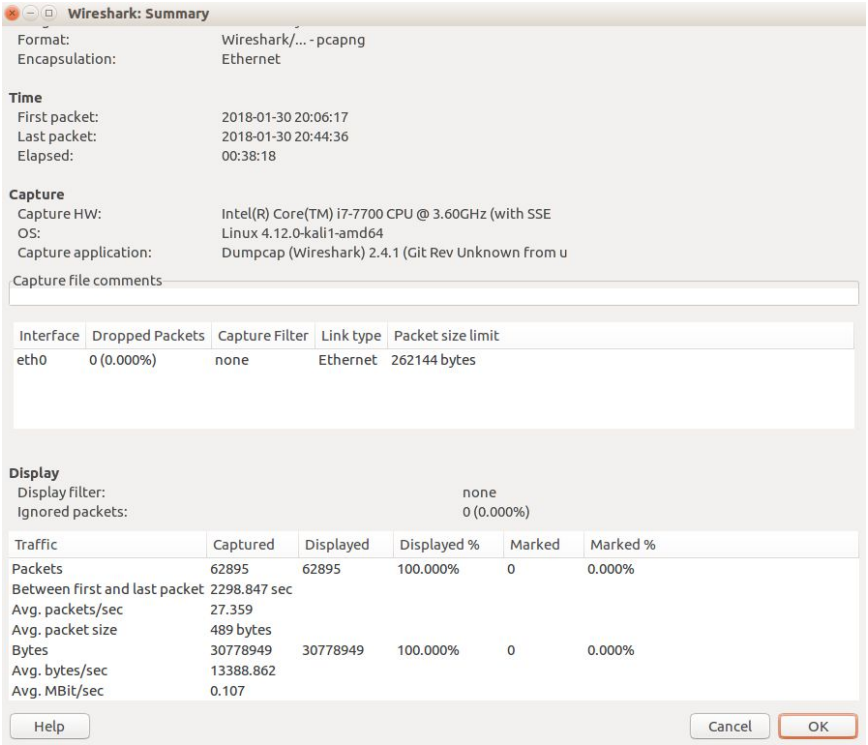
This file contains malicious activities try to:

- Identify the IP addresses involved in the activities.
- Their roles (who is the attacker and who is the victim)
- The type of the attacks, the outcome of the attacks and the severity of the attacks

# DPI with Wireshark: Traffic Statistics

First let us check the traffic summary from the Statistics.

What do you see? What metrics are there?



# DPI with Wireshark: Traffic Statistics

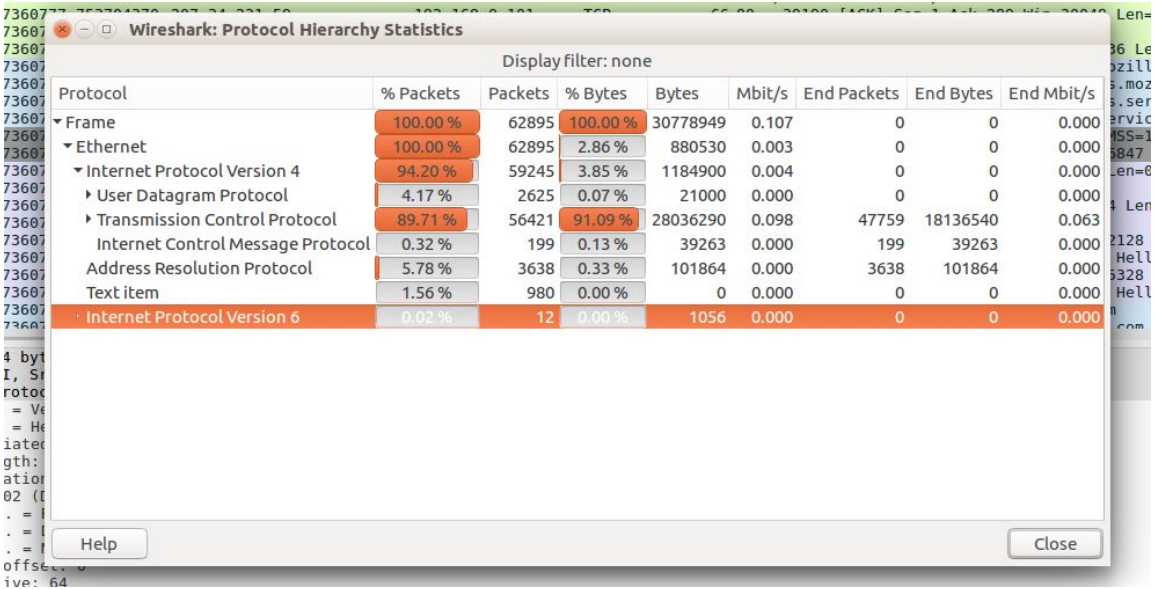
**Time**

| | |
|---|---|
| First packet: | 2018-01-30 20:06:17 |
| Last packet: | 2018-01-30 20:44:36 |
| Elapsed: | 00:38:18 |

| Traffic | Captured | Displayed | Displayed % | Marked | Marked % |
|---|---|---|---|---|---|
| Packets | 62895 | 62895 | 100.000% | 0 | 0.000% |
| Between first and last packet | 2298.847 sec | | | | |
| Avg. packets/sec | 27.359 | | | | |
| Avg. packet size | 489 bytes | | | | |
| Bytes | 30778949 | 30778949 | 100.000% | 0 | 0.000% |
| Avg. bytes/sec | 13388.862 | | | | |
| Avg. MBit/sec | 0.107 | | | | |

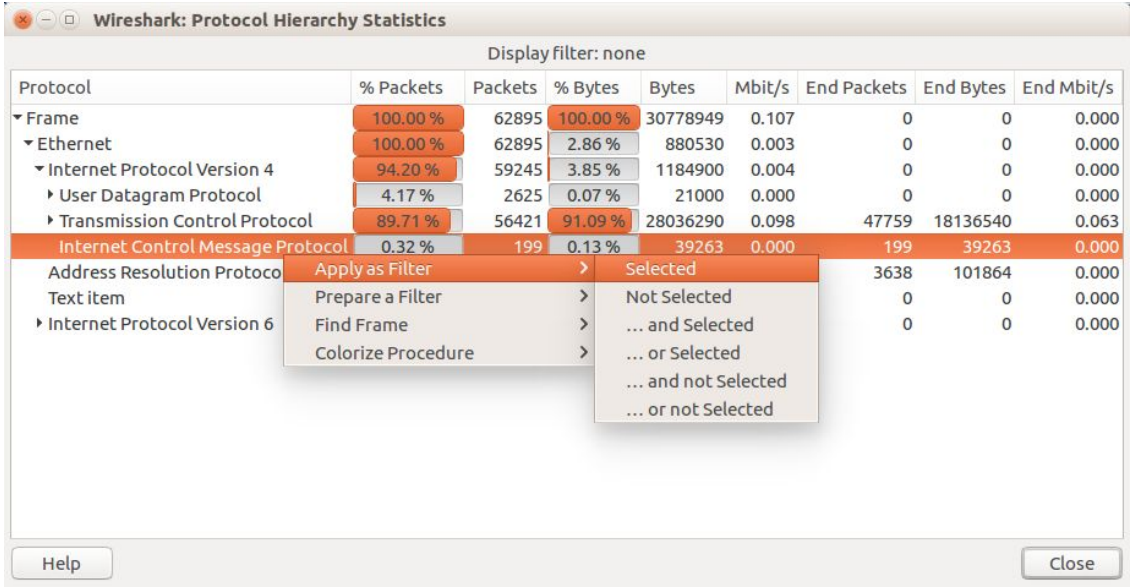
# DPI with Wireshark: Traffic Statistics

Go to **Statistics > Protocol** Hierarchy to see a list of all the protocols present, and a count of the number of packets and bytes for each protocol.



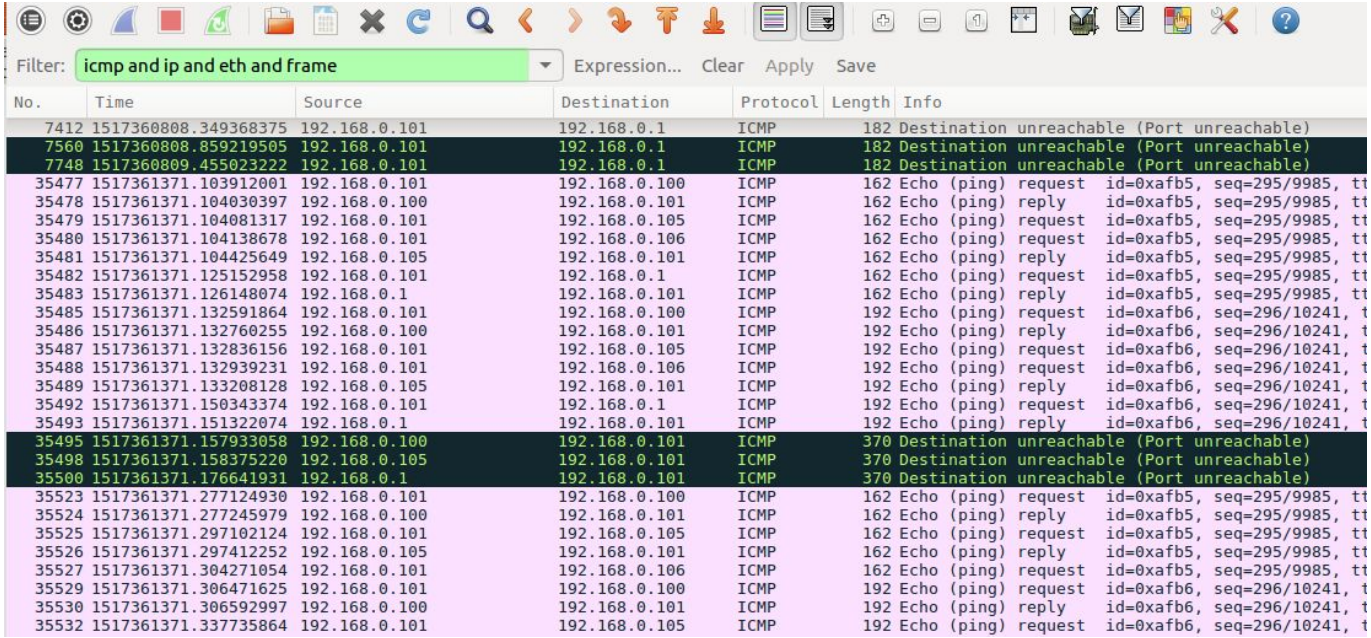
# DPI with Wireshark: Traffic Statistics

You can then right-click on a line and select **Apply As Filter > Selected** to see all the packets of that protocol type.



# DPI with Wireshark

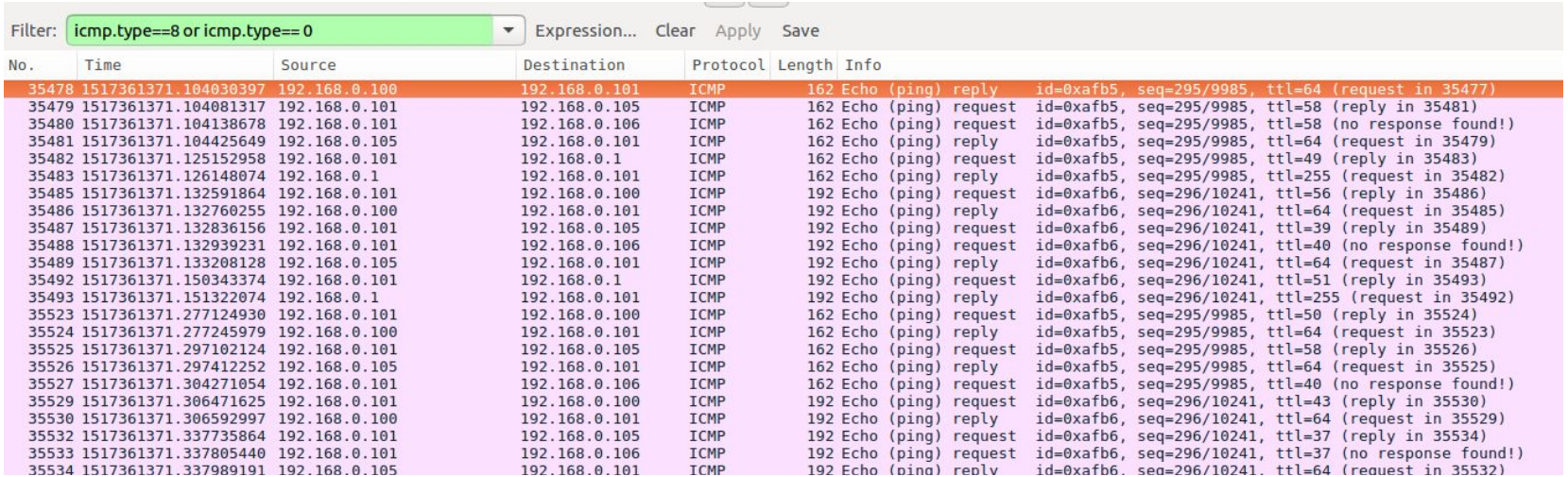
After applying the filter we can see many ICMP request to different hosts on the same local network




| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 7412 | 1517360808.349368375 | 192.168.0.101 | 192.168.0.1 | ICMP | 182 | Destination unreachable (Port unreachable) |
| 7560 | 1517360808.859219505 | 192.168.0.101 | 192.168.0.1 | ICMP | 182 | Destination unreachable (Port unreachable) |
| 7748 | 1517360809.455023222 | 192.168.0.101 | 192.168.0.1 | ICMP | 182 | Destination unreachable (Port unreachable) |
| 35477 | 1517361371.103912001 | 192.168.0.101 | 192.168.0.100 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, tt |
| 35478 | 1517361371.104030397 | 192.168.0.100 | 192.168.0.101 | ICMP | 162 | Echo (ping) reply id=0xafb5, seq=295/9985, tt |
| 35479 | 1517361371.104081317 | 192.168.0.101 | 192.168.0.105 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, tt |
| 35480 | 1517361371.104138678 | 192.168.0.101 | 192.168.0.106 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, tt |
| 35481 | 1517361371.104425649 | 192.168.0.105 | 192.168.0.101 | ICMP | 162 | Echo (ping) reply id=0xafb5, seq=295/9985, tt |
| 35482 | 1517361371.125152958 | 192.168.0.101 | 192.168.0.1 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, tt |
| 35483 | 1517361371.126148074 | 192.168.0.1 | 192.168.0.101 | ICMP | 162 | Echo (ping) reply id=0xafb5, seq=295/9985, tt |
| 35485 | 1517361371.132591864 | 192.168.0.101 | 192.168.0.100 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, t |
| 35486 | 1517361371.132760255 | 192.168.0.100 | 192.168.0.101 | ICMP | 192 | Echo (ping) reply id=0xafb6, seq=296/10241, t |
| 35487 | 1517361371.132836156 | 192.168.0.101 | 192.168.0.105 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, t |
| 35488 | 1517361371.132939231 | 192.168.0.101 | 192.168.0.106 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, t |
| 35489 | 1517361371.133208128 | 192.168.0.105 | 192.168.0.101 | ICMP | 192 | Echo (ping) reply id=0xafb6, seq=296/10241, t |
| 35492 | 1517361371.150343374 | 192.168.0.101 | 192.168.0.1 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, t |
| 35493 | 1517361371.151322074 | 192.168.0.1 | 192.168.0.101 | ICMP | 192 | Echo (ping) reply id=0xafb6, seq=296/10241, t |
| 35495 | 1517361371.157933058 | 192.168.0.100 | 192.168.0.101 | ICMP | 370 | Destination unreachable (Port unreachable) |
| 35498 | 1517361371.158375220 | 192.168.0.105 | 192.168.0.101 | ICMP | 370 | Destination unreachable (Port unreachable) |
| 35500 | 1517361371.176641931 | 192.168.0.1 | 192.168.0.101 | ICMP | 370 | Destination unreachable (Port unreachable) |
| 35523 | 1517361371.277124930 | 192.168.0.101 | 192.168.0.100 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, tt |
| 35524 | 1517361371.277245979 | 192.168.0.100 | 192.168.0.101 | ICMP | 162 | Echo (ping) reply id=0xafb5, seq=295/9985, tt |
| 35525 | 1517361371.297102124 | 192.168.0.101 | 192.168.0.105 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, tt |
| 35526 | 1517361371.297412252 | 192.168.0.105 | 192.168.0.101 | ICMP | 162 | Echo (ping) reply id=0xafb5, seq=295/9985, tt |
| 35527 | 1517361371.304271054 | 192.168.0.101 | 192.168.0.106 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, tt |
| 35529 | 1517361371.306471625 | 192.168.0.101 | 192.168.0.100 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, t |
| 35530 | 1517361371.306592997 | 192.168.0.100 | 192.168.0.101 | ICMP | 192 | Echo (ping) reply id=0xafb6, seq=296/10241, t |
| 35532 | 1517361371.337735864 | 192.168.0.101 | 192.168.0.105 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, t |

# DPI with Wireshark

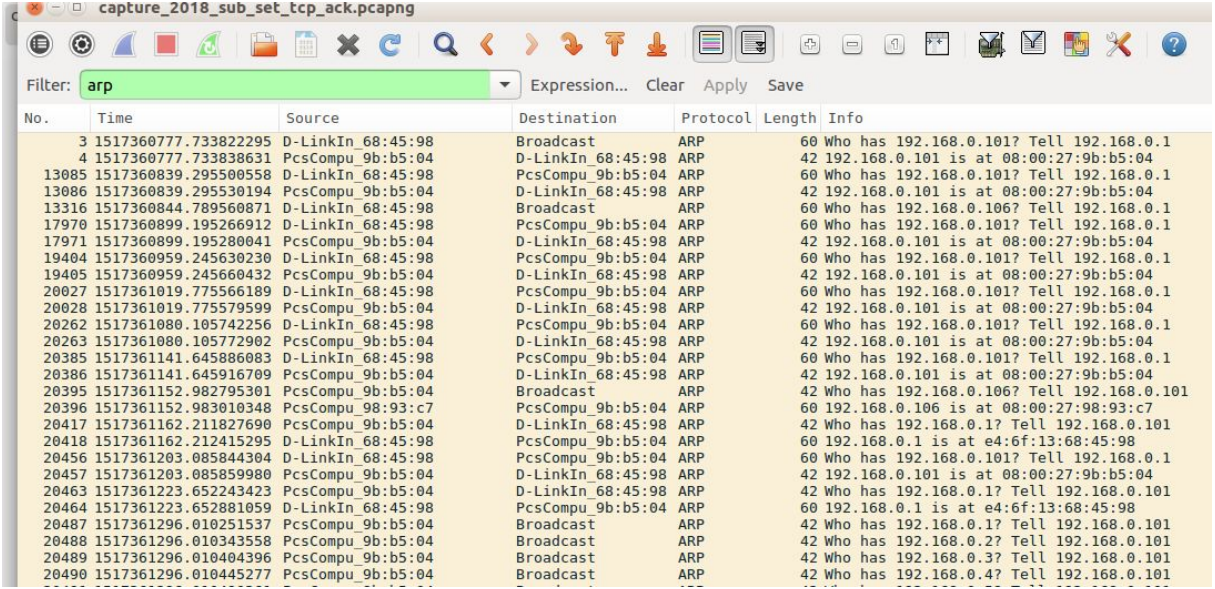
The ICMP traffic seems to be **Ping sweep** activities. We could investigate this possibility by applying another filter. icmp.type==8 or icmp.type==0




| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 35478 | 1517361371.104030397 | 192.168.0.100 | 192.168.0.101 | ICMP | 162 | Echo (ping) reply id=0xafb5, seq=295/9985, ttl=64 (request in 35477) |
| 35479 | 1517361371.104081317 | 192.168.0.101 | 192.168.0.105 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, ttl=58 (reply in 35481) |
| 35480 | 1517361371.104138678 | 192.168.0.101 | 192.168.0.106 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, ttl=58 (no response found!) |
| 35481 | 1517361371.104425649 | 192.168.0.105 | 192.168.0.101 | ICMP | 162 | Echo (ping) reply id=0xafb5, seq=295/9985, ttl=64 (request in 35479) |
| 35482 | 1517361371.125152958 | 192.168.0.101 | 192.168.0.1 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, ttl=49 (reply in 35483) |
| 35483 | 1517361371.126148074 | 192.168.0.1 | 192.168.0.101 | ICMP | 162 | Echo (ping) reply id=0xafb5, seq=295/9985, ttl=255 (request in 35482) |
| 35485 | 1517361371.132591864 | 192.168.0.101 | 192.168.0.100 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, ttl=56 (reply in 35486) |
| 35486 | 1517361371.132760255 | 192.168.0.100 | 192.168.0.101 | ICMP | 192 | Echo (ping) reply id=0xafb6, seq=296/10241, ttl=64 (request in 35485) |
| 35487 | 1517361371.132836156 | 192.168.0.101 | 192.168.0.105 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, ttl=39 (reply in 35489) |
| 35488 | 1517361371.132939231 | 192.168.0.101 | 192.168.0.106 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, ttl=40 (no response found!) |
| 35489 | 1517361371.133208128 | 192.168.0.105 | 192.168.0.101 | ICMP | 192 | Echo (ping) reply id=0xafb6, seq=296/10241, ttl=64 (request in 35487) |
| 35492 | 1517361371.150343374 | 192.168.0.101 | 192.168.0.1 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, ttl=51 (reply in 35493) |
| 35493 | 1517361371.151322074 | 192.168.0.1 | 192.168.0.101 | ICMP | 192 | Echo (ping) reply id=0xafb6, seq=296/10241, ttl=255 (request in 35492) |
| 35523 | 1517361371.277124930 | 192.168.0.101 | 192.168.0.100 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, ttl=50 (reply in 35524) |
| 35524 | 1517361371.277245979 | 192.168.0.100 | 192.168.0.101 | ICMP | 162 | Echo (ping) reply id=0xafb5, seq=295/9985, ttl=64 (request in 35523) |
| 35525 | 1517361371.297102124 | 192.168.0.101 | 192.168.0.105 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, ttl=58 (reply in 35526) |
| 35526 | 1517361371.297412252 | 192.168.0.105 | 192.168.0.101 | ICMP | 162 | Echo (ping) reply id=0xafb5, seq=295/9985, ttl=64 (request in 35525) |
| 35527 | 1517361371.304271054 | 192.168.0.101 | 192.168.0.106 | ICMP | 162 | Echo (ping) request id=0xafb5, seq=295/9985, ttl=40 (no response found!) |
| 35529 | 1517361371.306471625 | 192.168.0.101 | 192.168.0.100 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, ttl=43 (reply in 35530) |
| 35530 | 1517361371.306592997 | 192.168.0.100 | 192.168.0.101 | ICMP | 192 | Echo (ping) reply id=0xafb6, seq=296/10241, ttl=64 (request in 35529) |
| 35532 | 1517361371.337735864 | 192.168.0.101 | 192.168.0.105 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, ttl=37 (reply in 35534) |
| 35533 | 1517361371.337805440 | 192.168.0.101 | 192.168.0.106 | ICMP | 192 | Echo (ping) request id=0xafb6, seq=296/10241, ttl=37 (no response found!) |
| 35534 | 1517361371.337989191 | 192.168.0.105 | 192.168.0.101 | ICMP | 192 | Echo (ping) reply id=0xafb6, seq=296/10241, ttl=64 (request in 35532) |

# DPI with Wireshark

Let us try to check for **ARP sweep**. Where the attacker send ARP broadcast (for broadcast, destination MAC will be **0xff:ff:ff:ff:ff:ff**)




| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 3 | 1517360777.733822295 | D-LinkIn_68:45:98 | Broadcast | ARP | 60 | Who has 192.168.0.101? Tell 192.168.0.1 |
| 4 | 1517360777.733838631 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | 192.168.0.101 is at 08:00:27:9b:b5:04 |
| 13085 | 1517360839.295500558 | D-LinkIn_68:45:98 | PcsCompu_9b:b5:04 | ARP | 60 | Who has 192.168.0.101? Tell 192.168.0.1 |
| 13086 | 1517360839.295530194 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | 192.168.0.101 is at 08:00:27:9b:b5:04 |
| 13316 | 1517360844.789560871 | D-LinkIn_68:45:98 | Broadcast | ARP | 60 | Who has 192.168.0.106? Tell 192.168.0.1 |
| 17970 | 1517360899.195266912 | D-LinkIn_68:45:98 | PcsCompu_9b:b5:04 | ARP | 60 | Who has 192.168.0.101? Tell 192.168.0.1 |
| 17971 | 1517360899.195280041 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | 192.168.0.101 is at 08:00:27:9b:b5:04 |
| 19404 | 1517360959.245630230 | D-LinkIn_68:45:98 | PcsCompu_9b:b5:04 | ARP | 60 | Who has 192.168.0.101? Tell 192.168.0.1 |
| 19405 | 1517360959.245660432 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | 192.168.0.101 is at 08:00:27:9b:b5:04 |
| 20027 | 1517361019.775566189 | D-LinkIn_68:45:98 | PcsCompu_9b:b5:04 | ARP | 60 | Who has 192.168.0.101? Tell 192.168.0.1 |
| 20028 | 1517361019.775579599 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | 192.168.0.101 is at 08:00:27:9b:b5:04 |
| 20262 | 1517361080.105742256 | D-LinkIn_68:45:98 | PcsCompu_9b:b5:04 | ARP | 60 | Who has 192.168.0.101? Tell 192.168.0.1 |
| 20263 | 1517361080.105772902 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | 192.168.0.101 is at 08:00:27:9b:b5:04 |
| 20385 | 1517361141.645886083 | D-LinkIn_68:45:98 | PcsCompu_9b:b5:04 | ARP | 60 | Who has 192.168.0.101? Tell 192.168.0.1 |
| 20386 | 1517361141.645916709 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | 192.168.0.101 is at 08:00:27:9b:b5:04 |
| 20395 | 1517361152.982795301 | PcsCompu_9b:b5:04 | Broadcast | ARP | 42 | Who has 192.168.0.106? Tell 192.168.0.101 |
| 20396 | 1517361152.983010348 | PcsCompu_98:93:c7 | PcsCompu_9b:b5:04 | ARP | 60 | 192.168.0.106 is at 08:00:27:98:93:c7 |
| 20417 | 1517361162.211827690 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | Who has 192.168.0.1? Tell 192.168.0.101 |
| 20418 | 1517361162.212415295 | D-LinkIn_68:45:98 | PcsCompu_9b:b5:04 | ARP | 60 | 192.168.0.1 is at e4:6f:13:68:45:98 |
| 20456 | 1517361203.085844304 | D-LinkIn_68:45:98 | PcsCompu_9b:b5:04 | ARP | 60 | Who has 192.168.0.101? Tell 192.168.0.1 |
| 20457 | 1517361203.085859980 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | 192.168.0.101 is at 08:00:27:9b:b5:04 |
| 20463 | 1517361223.652243423 | PcsCompu_9b:b5:04 | D-LinkIn_68:45:98 | ARP | 42 | Who has 192.168.0.1? Tell 192.168.0.101 |
| 20464 | 1517361223.652881059 | D-LinkIn_68:45:98 | PcsCompu_9b:b5:04 | ARP | 60 | 192.168.0.1 is at e4:6f:13:68:45:98 |
| 20487 | 1517361296.010251537 | PcsCompu_9b:b5:04 | Broadcast | ARP | 42 | Who has 192.168.0.1? Tell 192.168.0.101 |
| 20488 | 1517361296.010343558 | PcsCompu_9b:b5:04 | Broadcast | ARP | 42 | Who has 192.168.0.2? Tell 192.168.0.101 |
| 20489 | 1517361296.010404396 | PcsCompu_9b:b5:04 | Broadcast | ARP | 42 | Who has 192.168.0.3? Tell 192.168.0.101 |
| 20490 | 1517361296.010445277 | PcsCompu_9b:b5:04 | Broadcast | ARP | 42 | Who has 192.168.0.4? Tell 192.168.0.101 |

# DPI with Wireshark

We can see that the Ip address **192.168.0.101** is performing **ARP sweep** trying to find the MAC address for every possible live host in the network

```
20507 1517361296.411459913 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.9? Tell 192.168.0.101
20508 1517361296.411687978 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.10? Tell 192.168.0.101
20509 1517361296.411925239 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.17? Tell 192.168.0.101
20510 1517361296.412134620 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.18? Tell 192.168.0.101
20511 1517361296.412305369 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.19? Tell 192.168.0.101
20512 1517361296.414838427 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.22? Tell 192.168.0.101
20513 1517361296.415037876 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.23? Tell 192.168.0.101
20514 1517361296.611755085 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.26? Tell 192.168.0.101
20515 1517361296.613989810 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.17? Tell 192.168.0.101
20516 1517361296.614036507 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.18? Tell 192.168.0.101
20517 1517361296.614062837 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.19? Tell 192.168.0.101
20518 1517361296.614105951 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.29? Tell 192.168.0.101
20519 1517361296.616263294 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.22? Tell 192.168.0.101
20520 1517361296.616313302 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.23? Tell 192.168.0.101
20521 1517361296.812354334 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.26? Tell 192.168.0.101
20522 1517361296.815788433 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.29? Tell 192.168.0.101
20523 1517361296.816145001 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.36? Tell 192.168.0.101
20524 1517361296.816273620 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.37? Tell 192.168.0.101
20525 1517361296.816382025 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.38? Tell 192.168.0.101
20526 1517361296.818933432 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.41? Tell 192.168.0.101
20527 1517361296.819108928 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.42? Tell 192.168.0.101
20528 1517361297.012886026 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.45? Tell 192.168.0.101
20529 1517361297.016300509 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.48? Tell 192.168.0.101
20530 1517361297.018979822 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.36? Tell 192.168.0.101
20531 1517361297.019124501 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.37? Tell 192.168.0.101
20532 1517361297.019203876 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.38? Tell 192.168.0.101
20533 1517361297.021787021 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.41? Tell 192.168.0.101
20534 1517361297.022024641 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.42? Tell 192.168.0.101
20535 1517361297.213202195 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.45? Tell 192.168.0.101
20536 1517361297.216353495 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.48? Tell 192.168.0.101
20537 1517361297.219033357 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.59? Tell 192.168.0.101
20538 1517361297.221323932 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.62? Tell 192.168.0.101
20539 1517361297.221388223 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.63? Tell 192.168.0.101
20540 1517361297.223666748 PcsCompu_9b:b5:04 Broadcast ARP 42 Who has 192.168.0.66? Tell 192.168.0.101
```

# DPI with Wireshark

When an attacker perform an ARP sweep it is possible that he either want to detect live hosts assuming some firewall block Ping  messages (ICMP request)

The other option is the attacker is planning for an ARP poisoning or ARP spoofing attack.

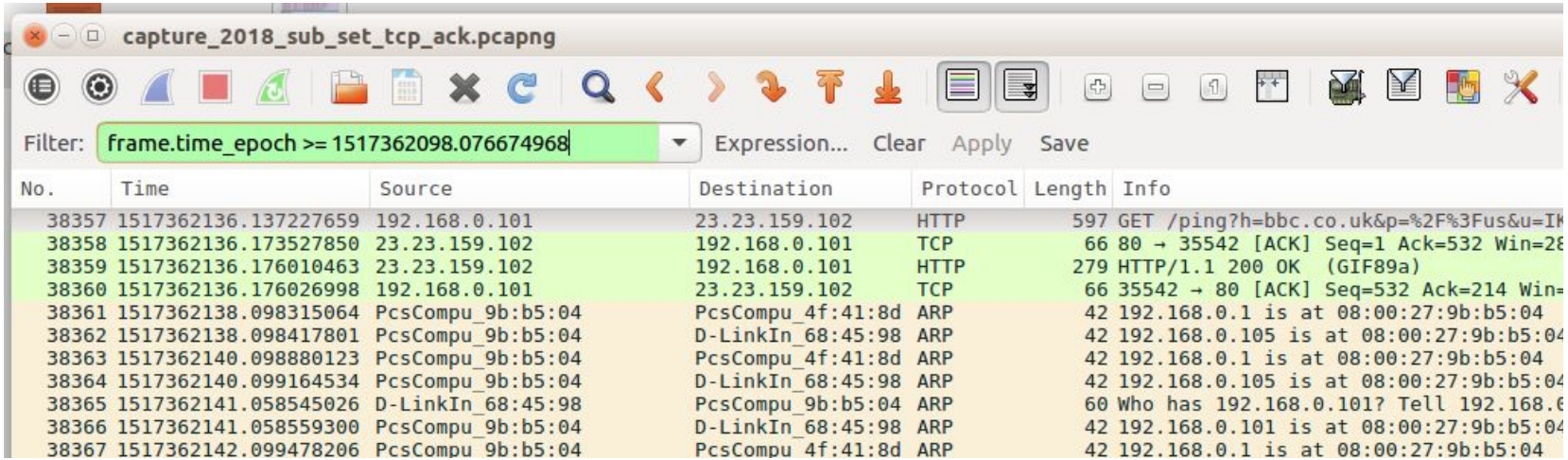
To check for **ARP spoofing** we will continue inspecting the ARP packets.

```
62399 1517362920.389754503 PcsCompu_9b:b5:04        PcsCompu_4f:41:8d ARP      42 192.168.0.1 is at 08:00:27:9b:b5:04
62400 1517362920.389864332 PcsCompu_9b:b5:04        D-LinkIn_68:45:98 ARP      42 192.168.0.105 is at 08:00:27:9b:b5:04
62407 1517362922.390289030 PcsCompu_9b:b5:04        PcsCompu_4f:41:8d ARP      42 192.168.0.1 is at 08:00:27:9b:b5:04
62408 1517362922.390499816 PcsCompu_9b:b5:04        D-LinkIn_68:45:98 ARP      42 192.168.0.105 is at 08:00:27:9b:b5:04
62409 1517362924.390821698 PcsCompu_9b:b5:04        PcsCompu_4f:41:8d ARP      42 192.168.0.1 is at 08:00:27:9b:b5:04
62410 1517362924.390891989 PcsCompu_9b:b5:04        D-LinkIn_68:45:98 ARP      42 192.168.0.105 is at 08:00:27:9b:b5:04
62411 1517362926.391326614 PcsCompu_9b:b5:04        PcsCompu_4f:41:8d ARP      42 192.168.0.1 is at 08:00:27:9b:b5:04
62412 1517362926.391563288 PcsCompu_9b:b5:04        D-LinkIn_68:45:98 ARP      42 192.168.0.105 is at 08:00:27:9b:b5:04
62413 1517362928.392064961 PcsCompu_9b:b5:04        PcsCompu_4f:41:8d ARP      42 192.168.0.1 is at 08:00:27:9b:b5:04
```

# DPI with Wireshark

We want to investigate what the attacker did after he executed the ARP spoofing. We could use the following filter

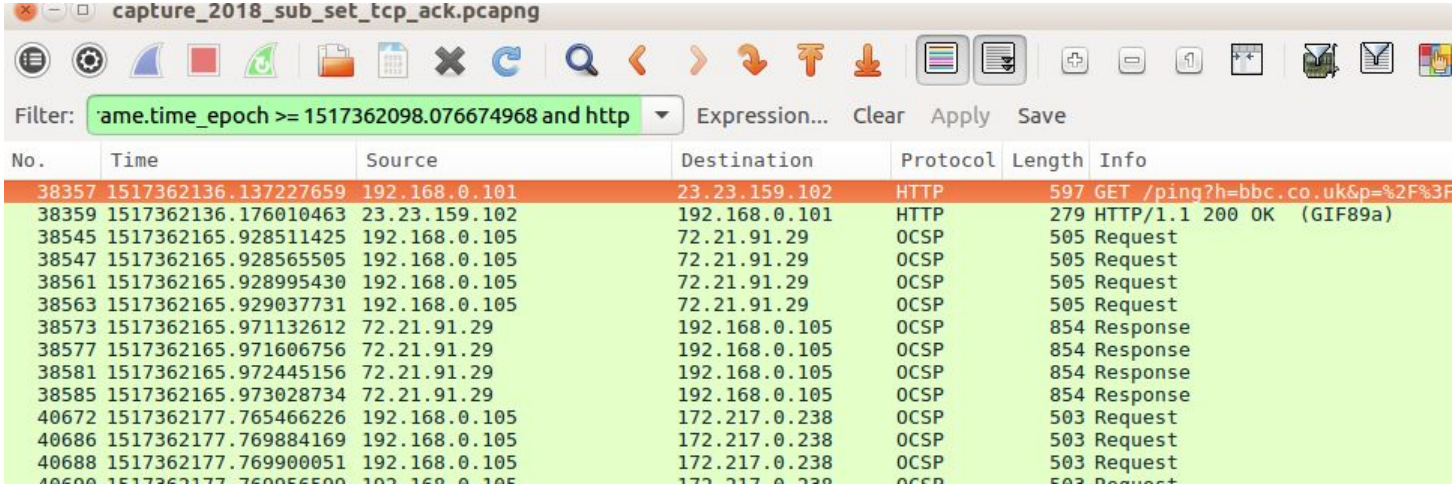
**frame.time_epoch >= 1517362098.076674968**



# DPI with Wireshark

We can see some HTTP traffic (unencrypted HTTP), to focus on those traffic we could update the filter by appending to it and http

**frame.time_epoch >= 1517362098.076674968 and http**

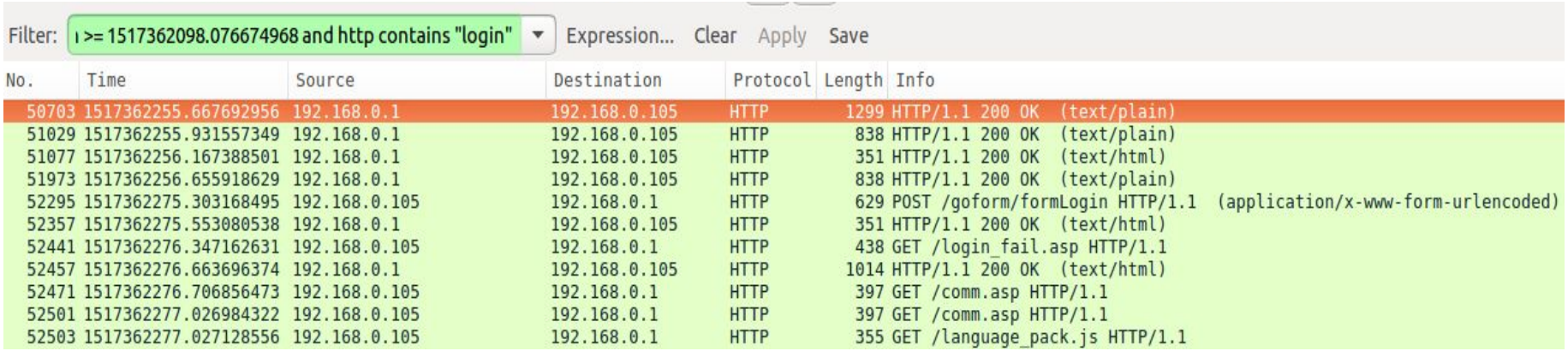



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 38357 | 1517362136.137227659 | 192.168.0.101 | 23.23.159.102 | HTTP | 597 | GET /ping?h=bbc.co.uk&p=%2F%3F |
| 38359 | 1517362136.176010463 | 23.23.159.102 | 192.168.0.101 | HTTP | 279 | HTTP/1.1 200 OK (GIF89a) |
| 38545 | 1517362165.928511425 | 192.168.0.105 | 72.21.91.29 | OCSP | 505 | Request |
| 38547 | 1517362165.928565505 | 192.168.0.105 | 72.21.91.29 | OCSP | 505 | Request |
| 38561 | 1517362165.928995430 | 192.168.0.105 | 72.21.91.29 | OCSP | 505 | Request |
| 38563 | 1517362165.929037731 | 192.168.0.105 | 72.21.91.29 | OCSP | 505 | Request |
| 38573 | 1517362165.971132612 | 72.21.91.29 | 192.168.0.105 | OCSP | 854 | Response |
| 38577 | 1517362165.971606756 | 72.21.91.29 | 192.168.0.105 | OCSP | 854 | Response |
| 38581 | 1517362165.972445156 | 72.21.91.29 | 192.168.0.105 | OCSP | 854 | Response |
| 38585 | 1517362165.973028734 | 72.21.91.29 | 192.168.0.105 | OCSP | 854 | Response |
| 40672 | 1517362177.765466226 | 192.168.0.105 | 172.217.0.238 | OCSP | 503 | Request |
| 40686 | 1517362177.769884169 | 192.168.0.105 | 172.217.0.238 | OCSP | 503 | Request |
| 40688 | 1517362177.769900051 | 192.168.0.105 | 172.217.0.238 | OCSP | 503 | Request |

# DPI with Wireshark

Is it possible that the victim send sensitive data over http. We could extend our filter to investigate the http payload for some keywords.

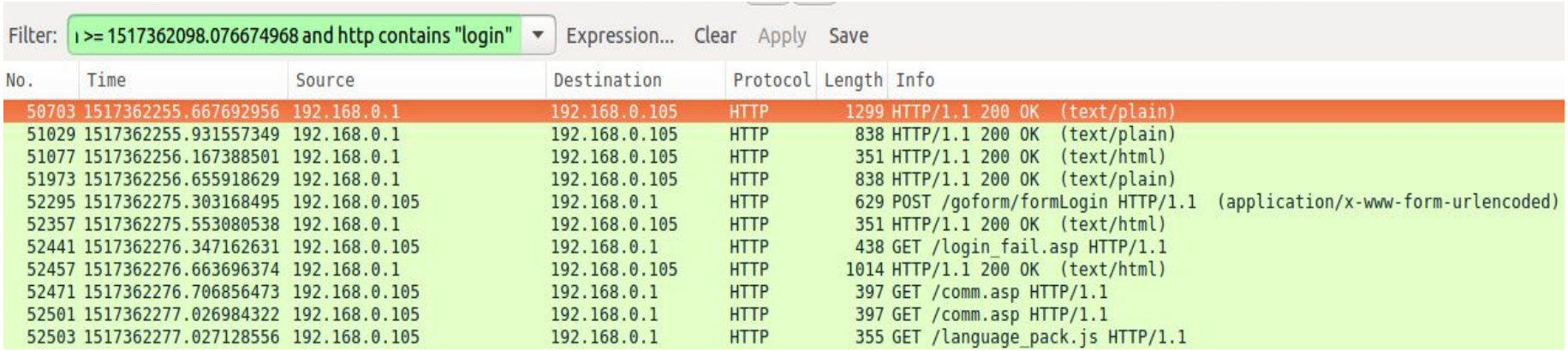
**frame.time_epoch >= 1517362098.076674968 and http contains "login"**




| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 50703 | 1517362255.667692956 | 192.168.0.1 | 192.168.0.105 | HTTP | 1299 | HTTP/1.1 200 OK (text/plain) |
| 51029 | 1517362255.931557349 | 192.168.0.1 | 192.168.0.105 | HTTP | 838 | HTTP/1.1 200 OK (text/plain) |
| 51077 | 1517362256.167388501 | 192.168.0.1 | 192.168.0.105 | HTTP | 351 | HTTP/1.1 200 OK (text/html) |
| 51973 | 1517362256.655918629 | 192.168.0.1 | 192.168.0.105 | HTTP | 838 | HTTP/1.1 200 OK (text/plain) |
| 52295 | 1517362275.303168495 | 192.168.0.105 | 192.168.0.1 | HTTP | 629 | POST /goform/formLogin HTTP/1.1 (application/x-www-form-urlencoded) |
| 52357 | 1517362275.553080538 | 192.168.0.1 | 192.168.0.105 | HTTP | 351 | HTTP/1.1 200 OK (text/html) |
| 52441 | 1517362276.347162631 | 192.168.0.105 | 192.168.0.1 | HTTP | 438 | GET /login_fail.asp HTTP/1.1 |
| 52457 | 1517362276.663696374 | 192.168.0.1 | 192.168.0.105 | HTTP | 1014 | HTTP/1.1 200 OK (text/html) |
| 52471 | 1517362276.706856473 | 192.168.0.105 | 192.168.0.1 | HTTP | 397 | GET /comm.asp HTTP/1.1 |
| 52501 | 1517362277.026984322 | 192.168.0.105 | 192.168.0.1 | HTTP | 397 | GET /comm.asp HTTP/1.1 |
| 52503 | 1517362277.027128556 | 192.168.0.105 | 192.168.0.1 | HTTP | 355 | GET /language_pack.js HTTP/1.1 |

# DPI with Wireshark

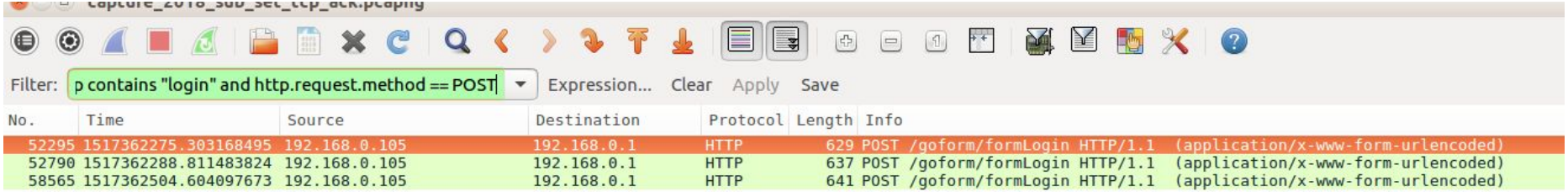
Since the filter return a nonempty results, then the HTTP contains a login keywork. But which packet??



# DPI with Wireshark

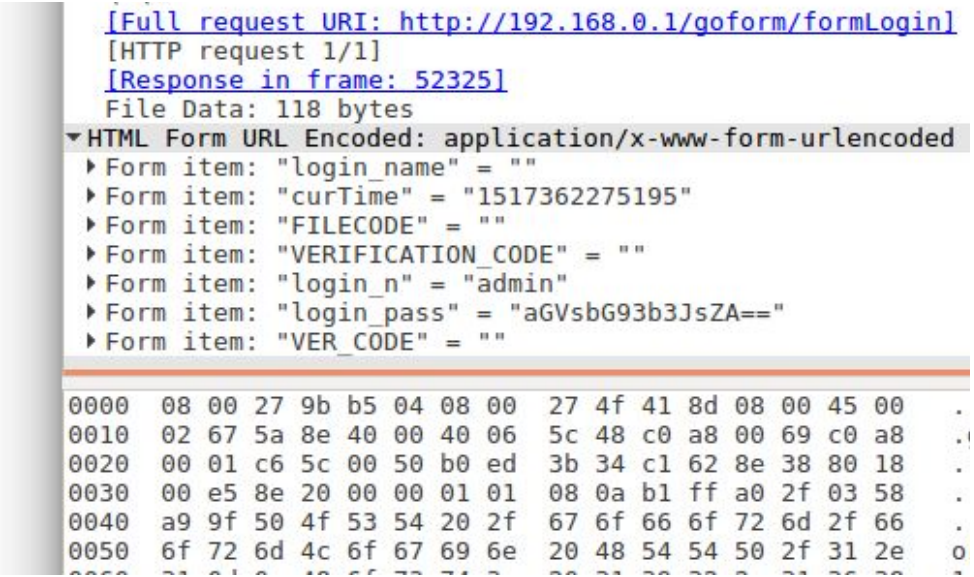
Most likely login data are sent over HTTP post method. So maybe we could update the filter to look for HTTP post

frame.time_epoch >= 1517362098.076674968 and http contains "login" and http.request.method == POST



# DPI with Wireshark

If we investigate the filtered packets we can see that the victim was trying to login to the network router with an admin account



# Deep Packet Inspection

What did we learn?

What are the challenges in DPI?

Could we automate the DPI process?

Could we use machine learning to improve DPI?

# Network Vulnerability Assessment

# What is Network Vulnerability Assessment?

The process of identifying and classifying any security vulnerability in computer networks or other communication systems.

It is the process of scanning, identifying and analyzing potential security holes.

it is an iterative process that should usually be conducted on a regular basis.

It is required when new software or hardware added to the network. Even with a software update, it is recommended to apply vulnerability assessment.

Network vulnerability could be an **active** or **passive** process.

# Passive Scanners

Passive Scanners will scan the network under investigation to collect and gather information about network nodes, running services, and network topology.

The scanner will analyze the collected data to identify potential security weakness.

Passive scanners use one or more network vulnerability database to identify the potential vulnerability.

It is essential to update network vulnerability databases otherwise the scanner will be outdated and provide inaccurate scanning results.

# Active Scanners

Active Scanners just like passive scanner they will scan the network under investigation to collect and gather information about network nodes, running services, and network topology.

The scanner will analyze the collected data to identify potential security weakness.

Based on the identified weakness the active scanner will try to exploit the identified weakness by attacking the network and monitor the network status.

Active scanners  not only report potential vulnerability but also attempt to verify the vulnerability by trying to exploit discovered vulnerability.

# Limitations and Issues with Network Scanners

Both active and passive scanner could produce false positive and false negative scan due to many reasons and they **can not** detect **zero-day exploit**

A **false positive** scan means a nonvulernable network asset( server, router, etc.) identified as vulnerable

A **false negative** scan means a vulnerable network asset ( server, router, etc.) identified as secure

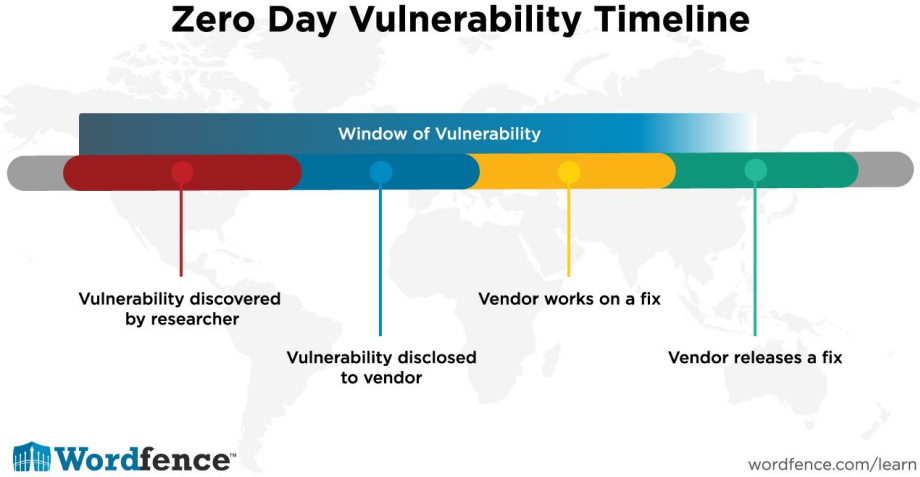
Both passive and active scan affect the network performance. In addition, the active scan could result in denial of services, disrupt regular network operations.

Qualified network **security engineers** need to review scanning results to detect potential false positive and false negative

# Zero-Day Exploit | Vulnerability

Is an attack that exploits a previously **unknown** security vulnerability.

A zero-day attack is also sometimes defined as an attack that takes advantage of a security vulnerability on the same day that the vulnerability becomes known.



# Nessus

Nessus is a **proprietary** vulnerability scanner developed by Tenable Network Security. It is **free of charge for personal use** in a non-enterprise environment.

Nessus is generally  a **passive** vulnerability scanner

Nessus **scan** for:

1. Vulnerabilities that allow a remote hacker to control or access sensitive data on a system (e.g. weak and default password)
2. Misconfiguration
3. Denials of service against the TCP/IP stack by using malformed packets
4. Security compliance (e.g PCI DSS = Payment Card Industry Data Security Standard)

You could use **Nessus Attack Scripting Language (NASL)** to write exploit and attacks

# OpenVAS

OpenVAS is a framework of several services and tools offering a comprehensive and powerful vulnerability scanning.

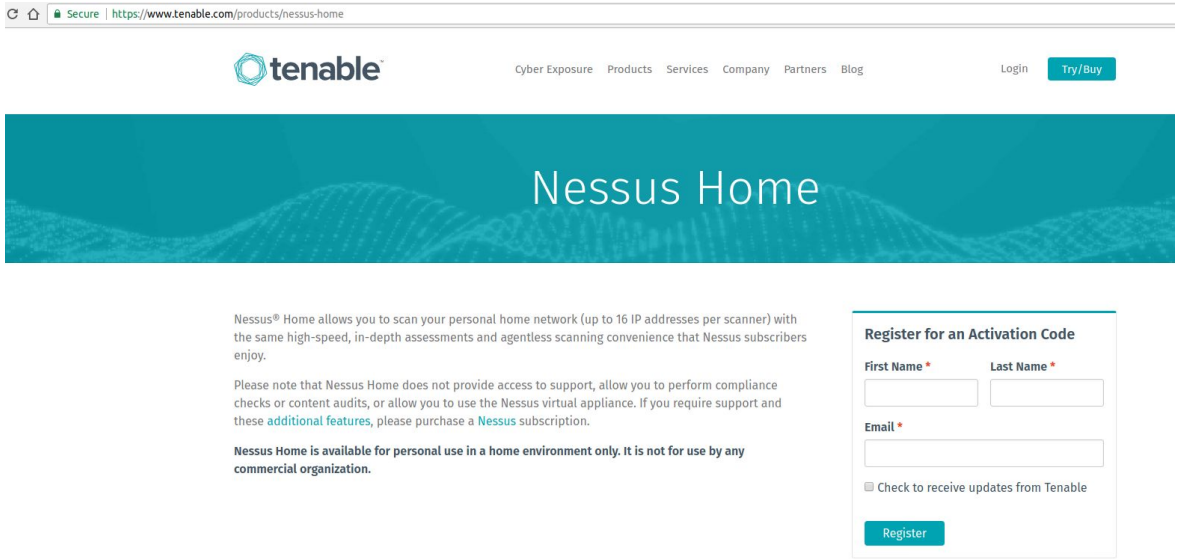
OpenVAS originally was a fork of the last free version of Nessus (2005)

OpenVAS is by default an **active** network vulnerability scanner even with the **none destructive scan** option. It is possible to crash a network services.

OpenVAS known vulnerabilities and mis-configuration using a large database **Network Vulnerability Tests** (NVTs)

# Using Nessus with Kali Linux

Go to https://www.tenable.com/products/nessus-home to register and download Nessus, you will receive an activation code by email



# Download the version that match your OS

## Nessus - 7.0.1

**Release Date**

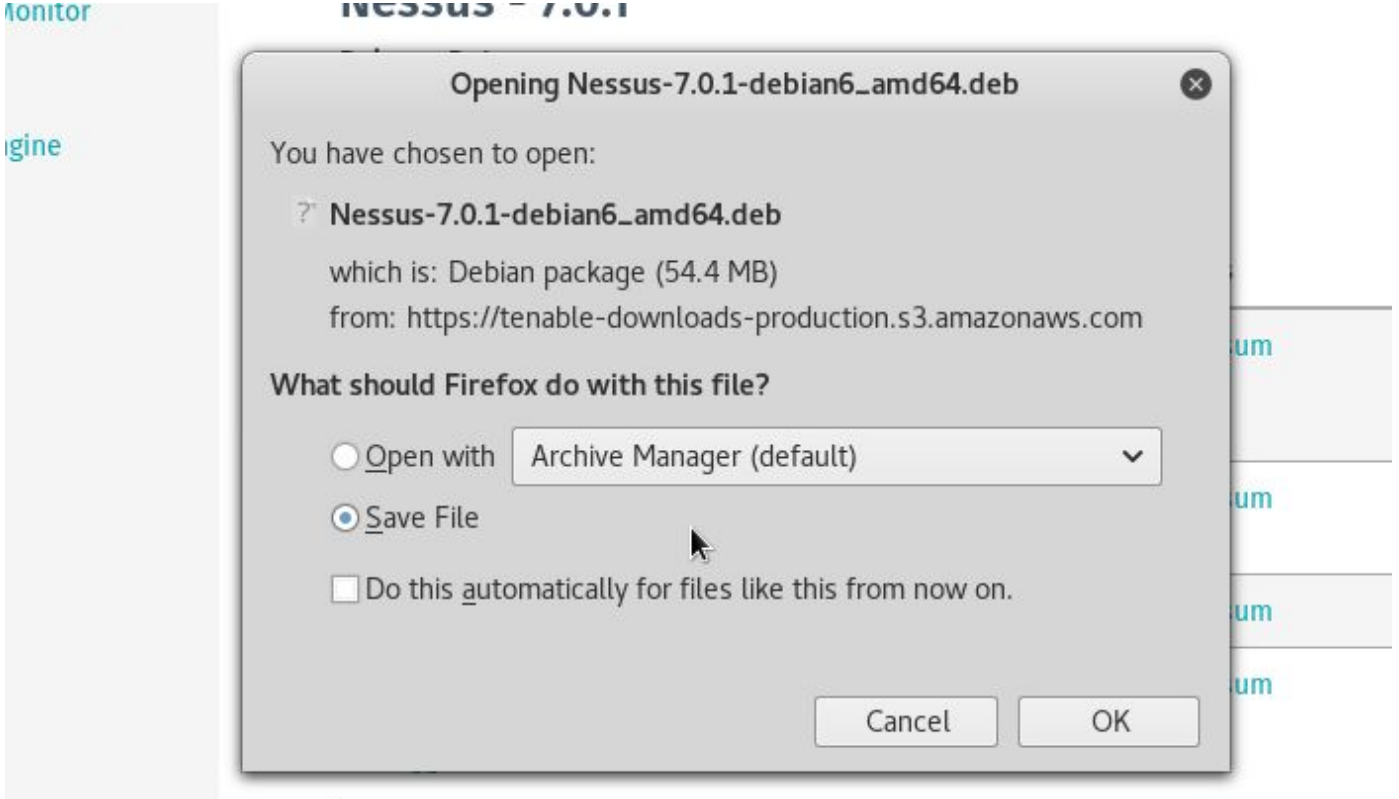
01/11/2018

**Release Notes:**

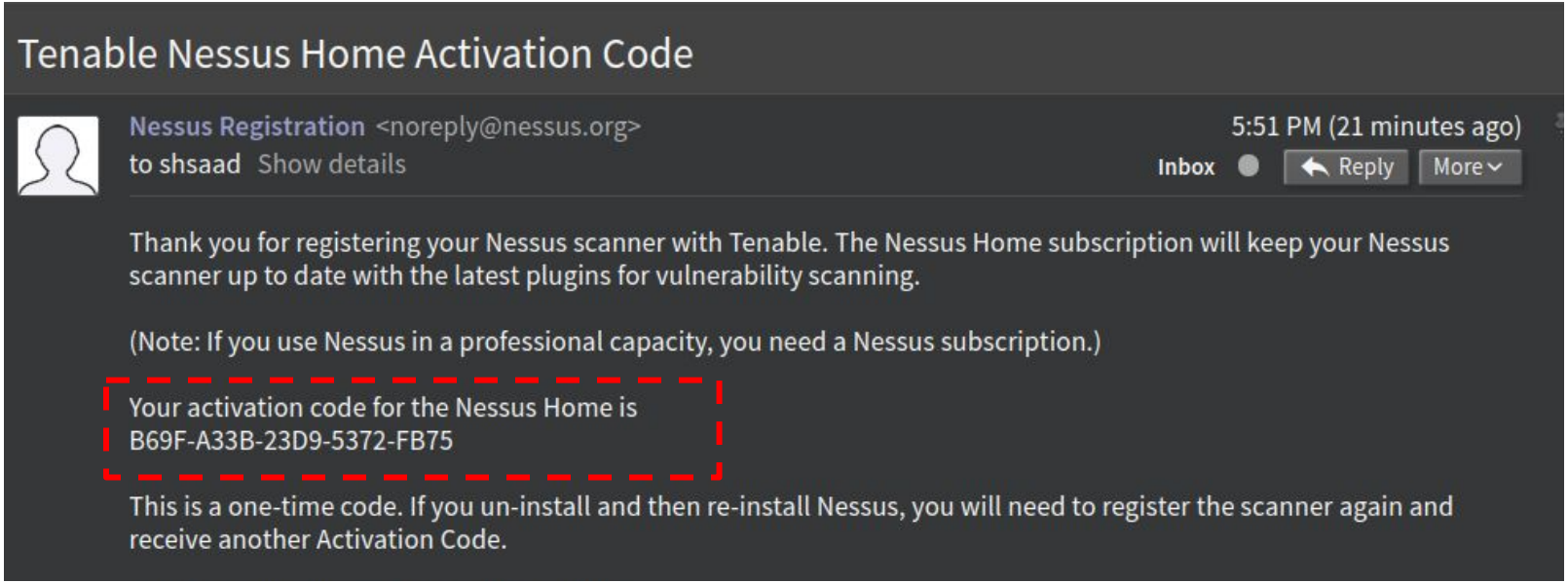
Nessus 7.0.1

| Name | Description | Details |
|---|---|---|
| Nessus-7.0.1-x64.msi | Windows Server 2008, Server 2008 R2*, Server 2012, Server 2012 R2, 7, 8, 10, Server 2016 (64-bit) | Checksum |
| Nessus-7.0.1-Win32.msi | Windows 7, 8, 10 (32-bit) | Checksum |
| Nessus-7.0.1.dmg | macOS (10.8 - 10.13) | Checksum |
| Nessus-7.0.1-debian6_amd64.deb | Debian 6, 7, 8 / Kali Linux 1 AMD64 | Checksum |

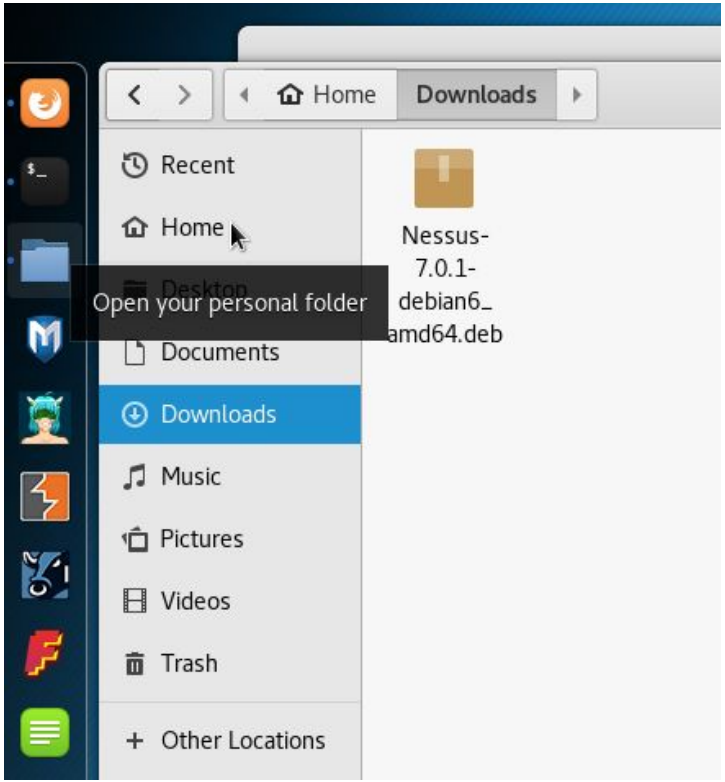
# Save the deb file to your disk

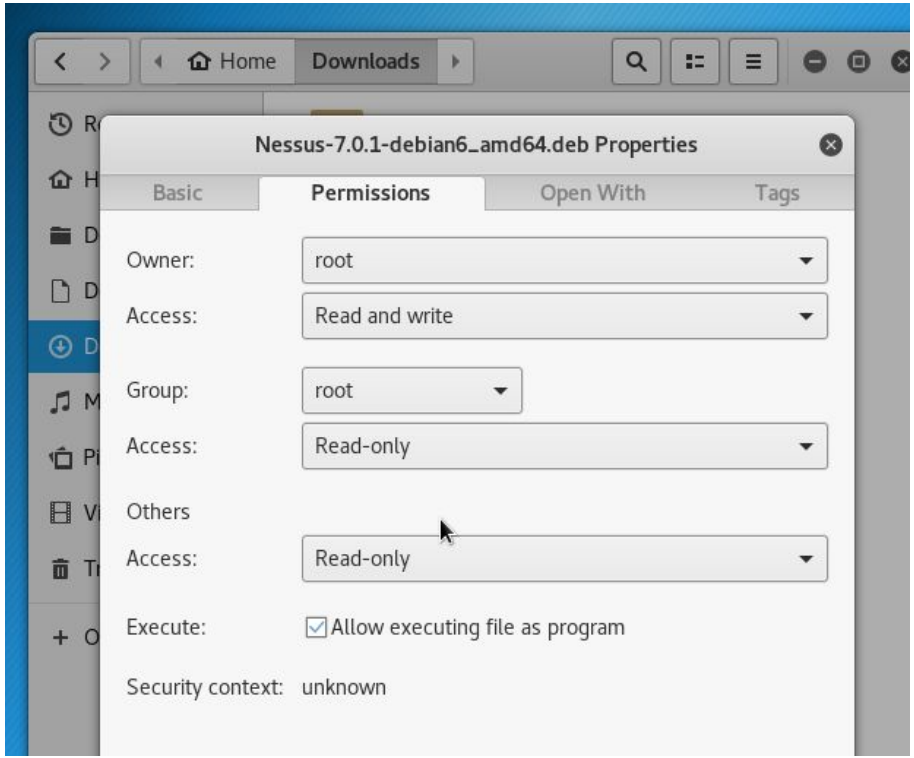


# Make sure you received the activation code



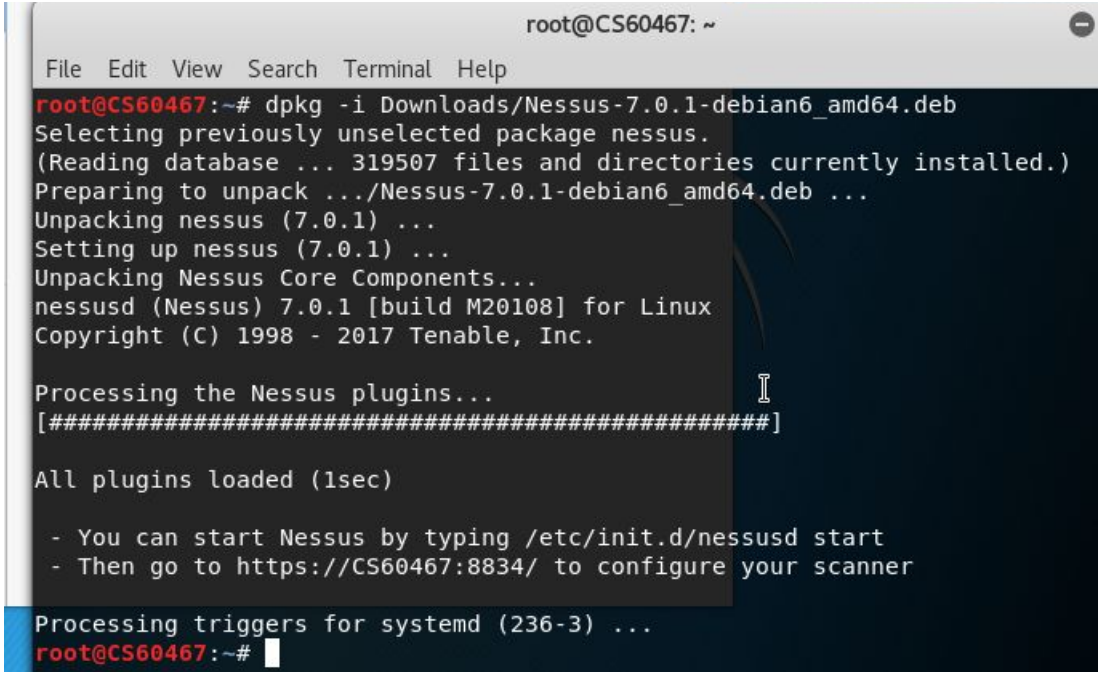
# Change the permission on the deb file



# Open the command line and run the installer

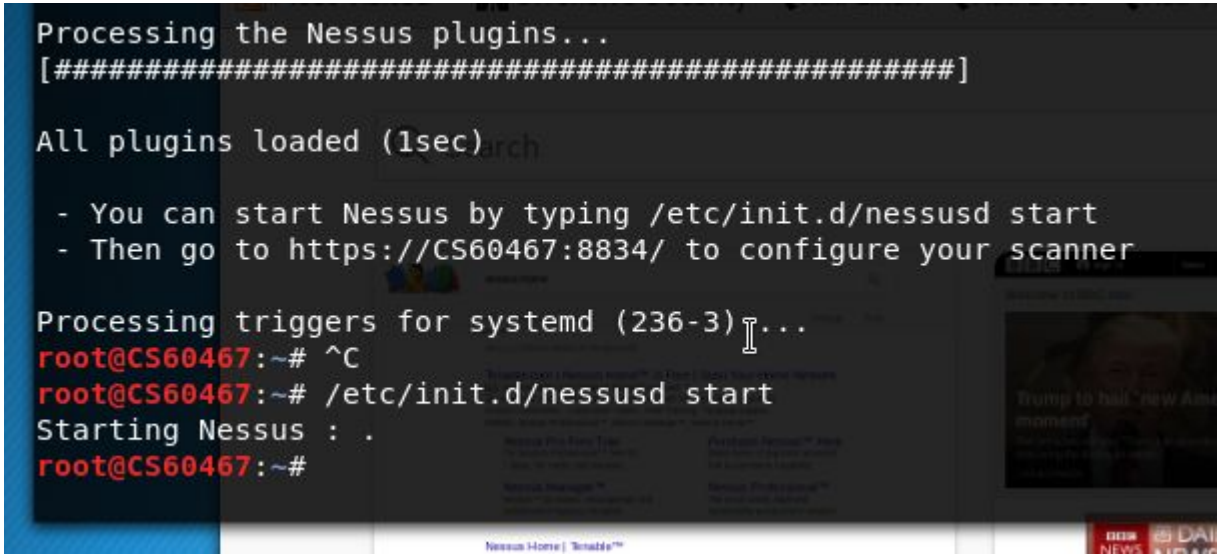
**dpkg -i Downloads/Nessus-7.0.1-debian6_amd64.deb**



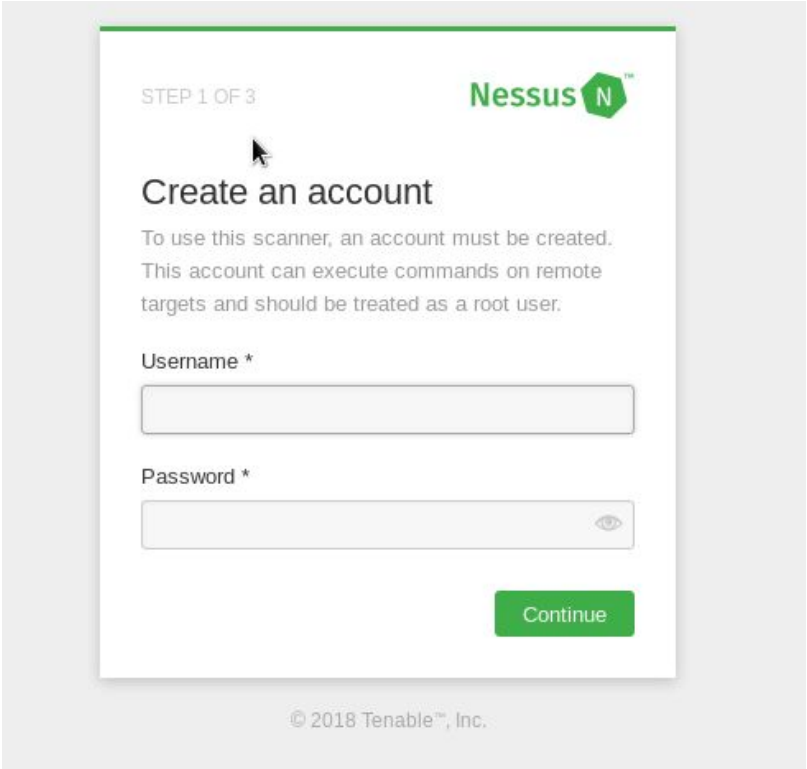
# Start the Nessus Server

**`/etc/init.d/nessusd start`**

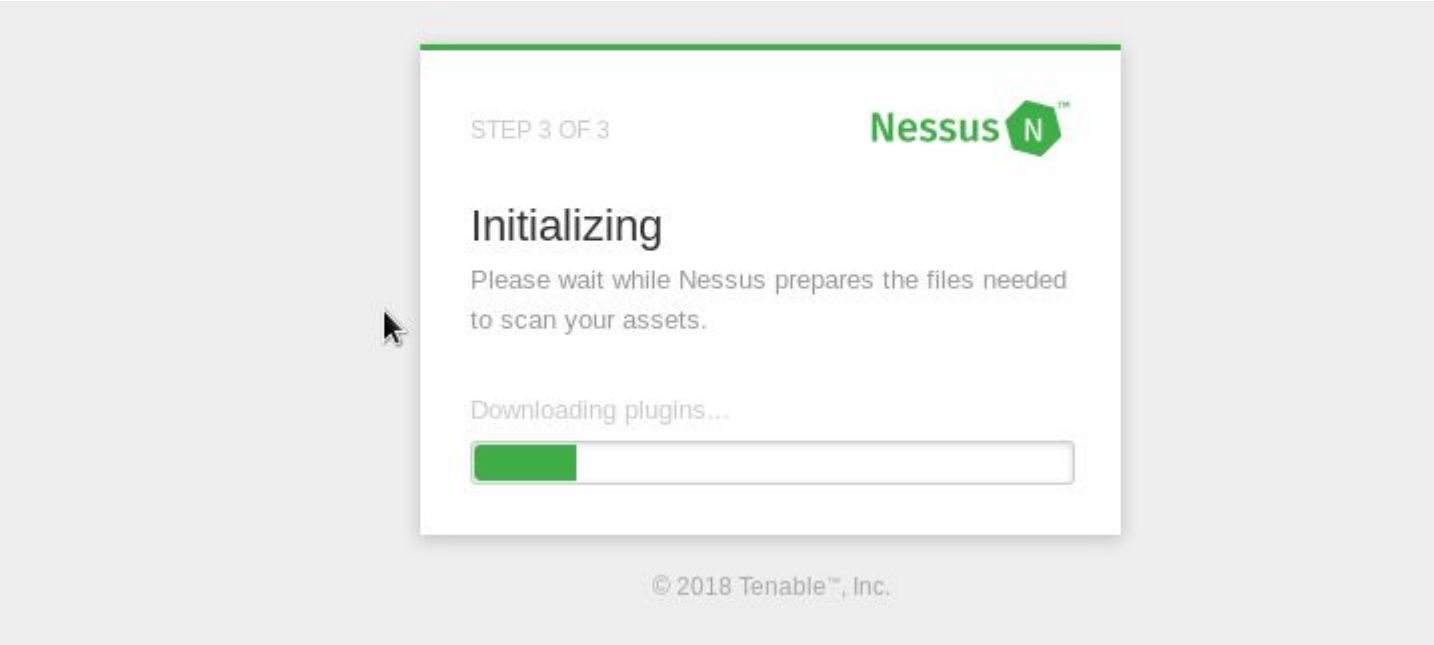
**Then open your process and go to https://<hostname>:8834**



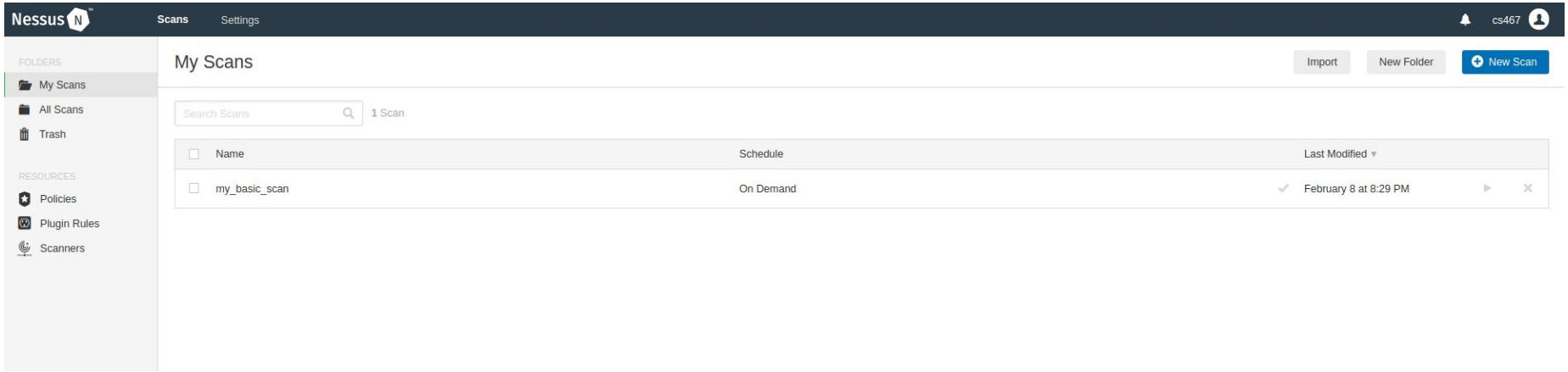
# Create and Account , Register, and Activate



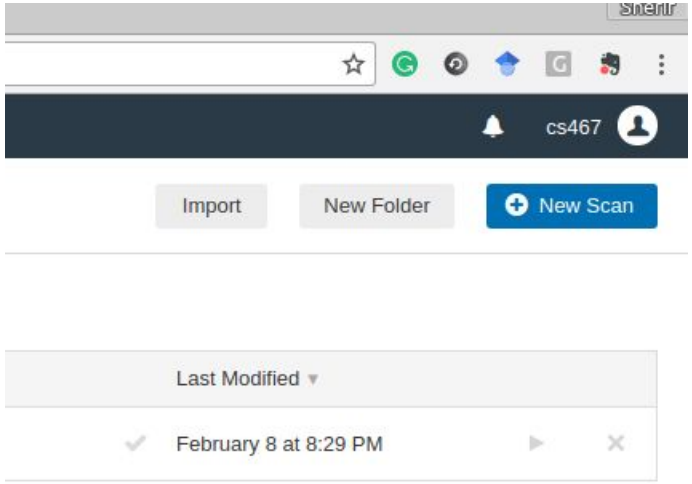
# Create and Account , Register, and Activate



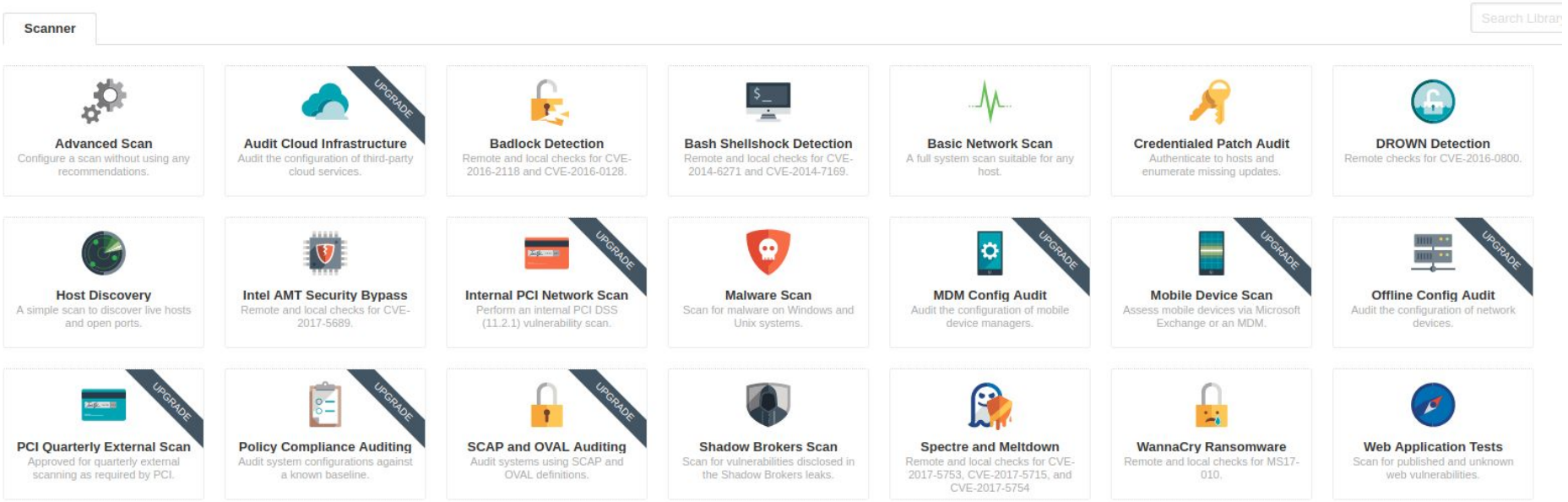
# Nessus Dashboard



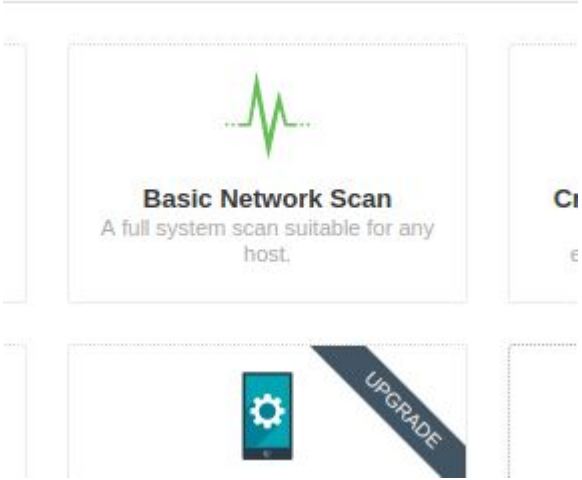
# Nessus Create a Scan

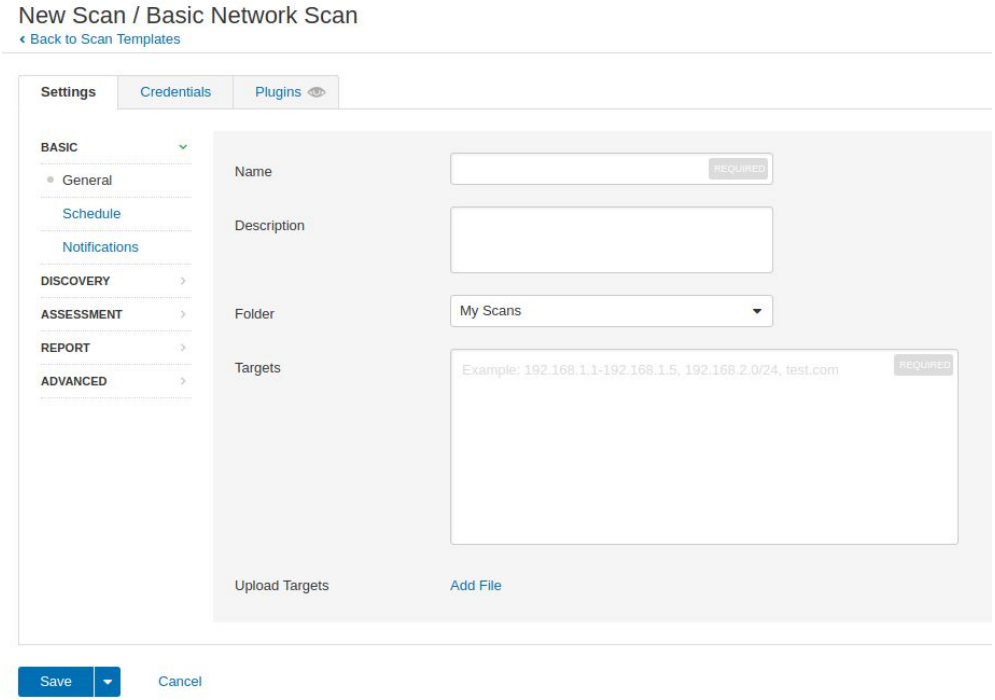


# Select the Type of the Scan

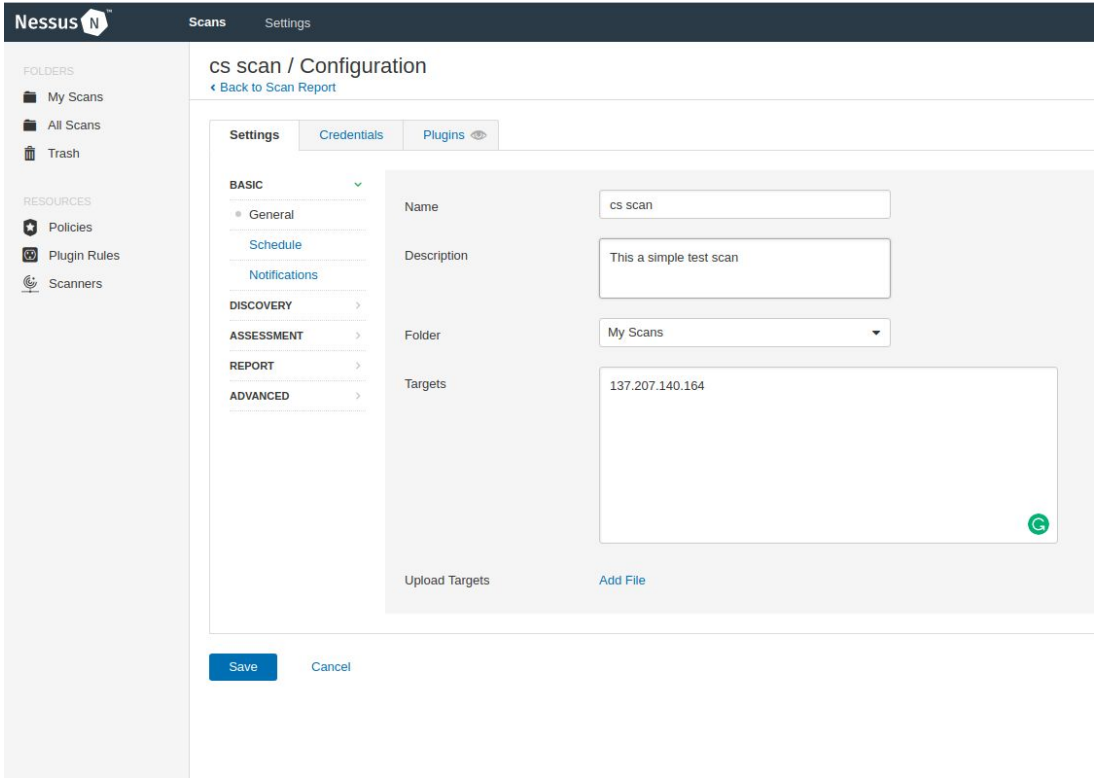


# Select Basic Network Scan

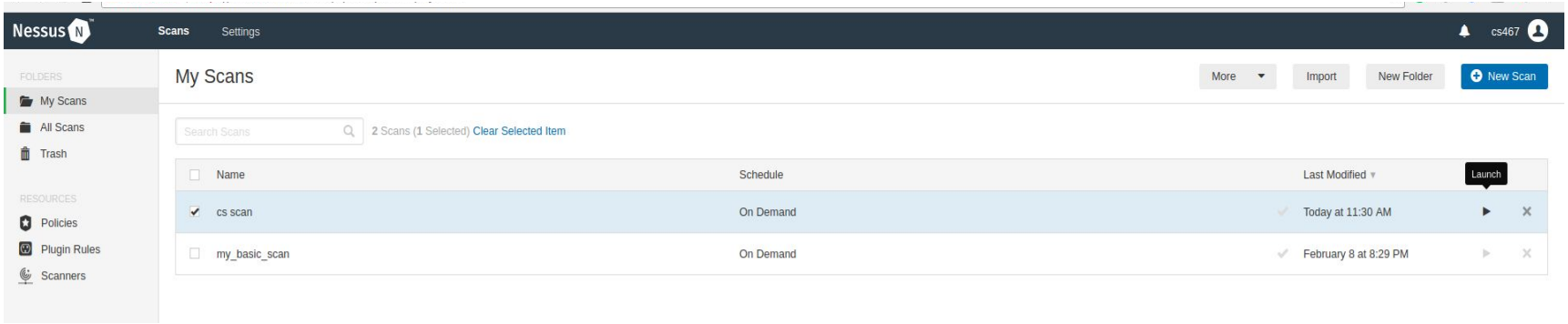


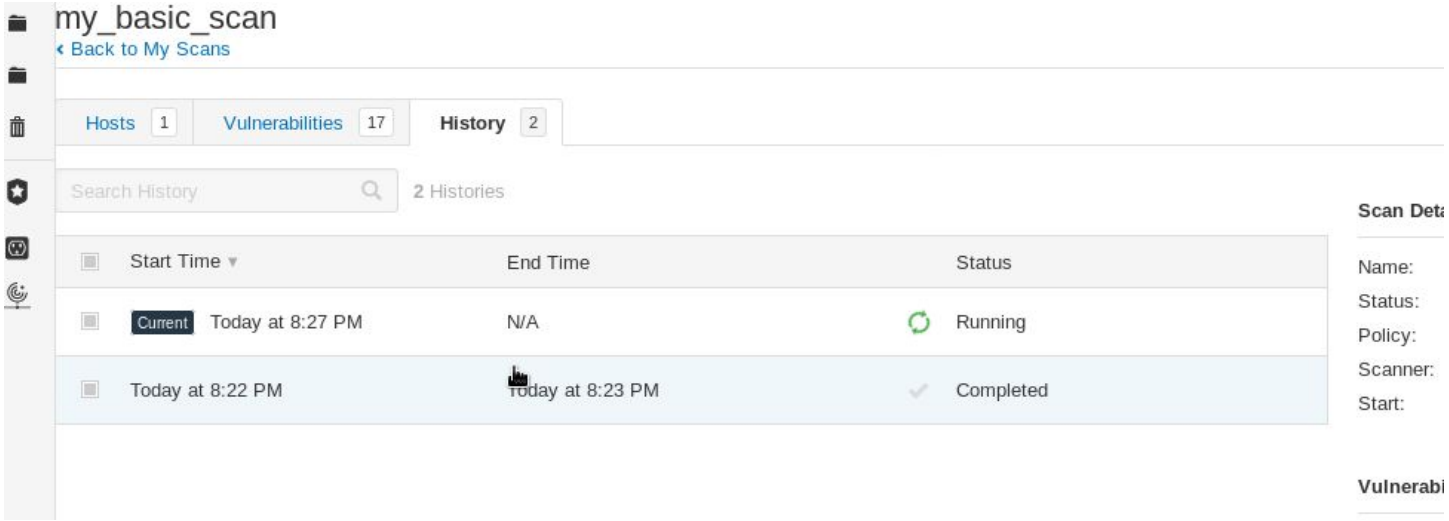
# Edit your Scan info and save it



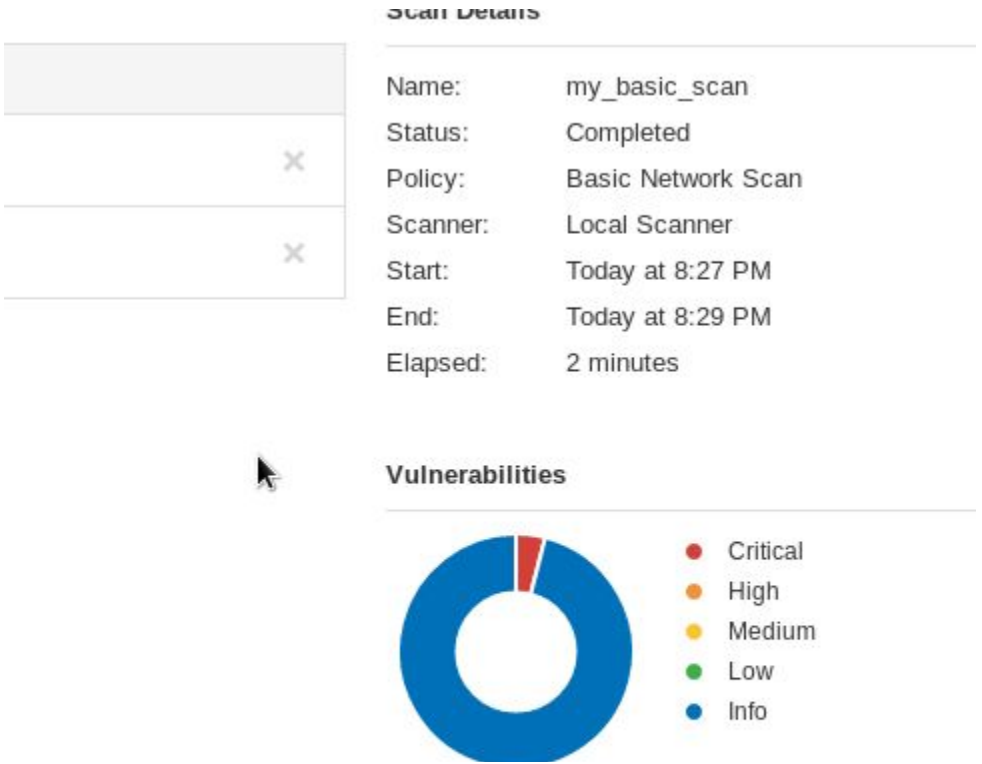
# Select and Launch your Scan



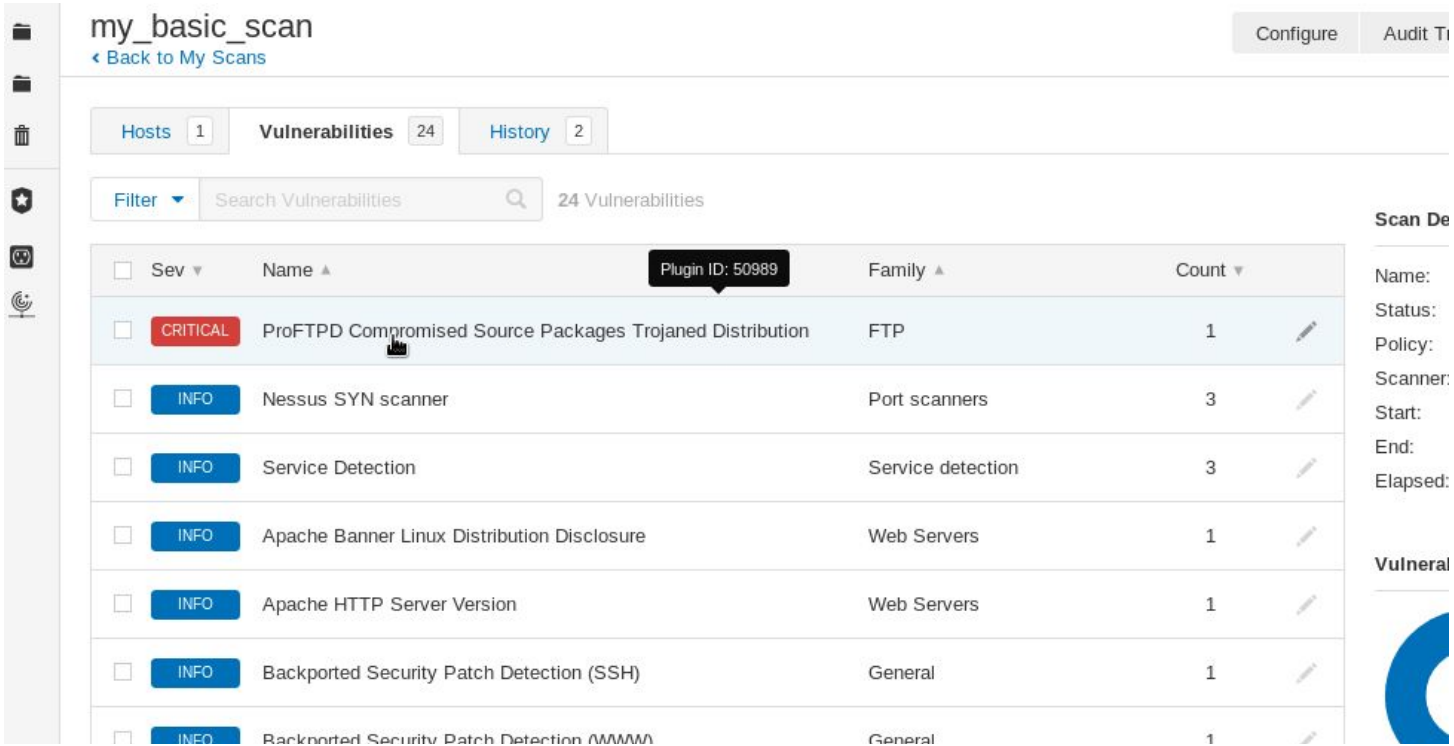
# Wait until the scan complete



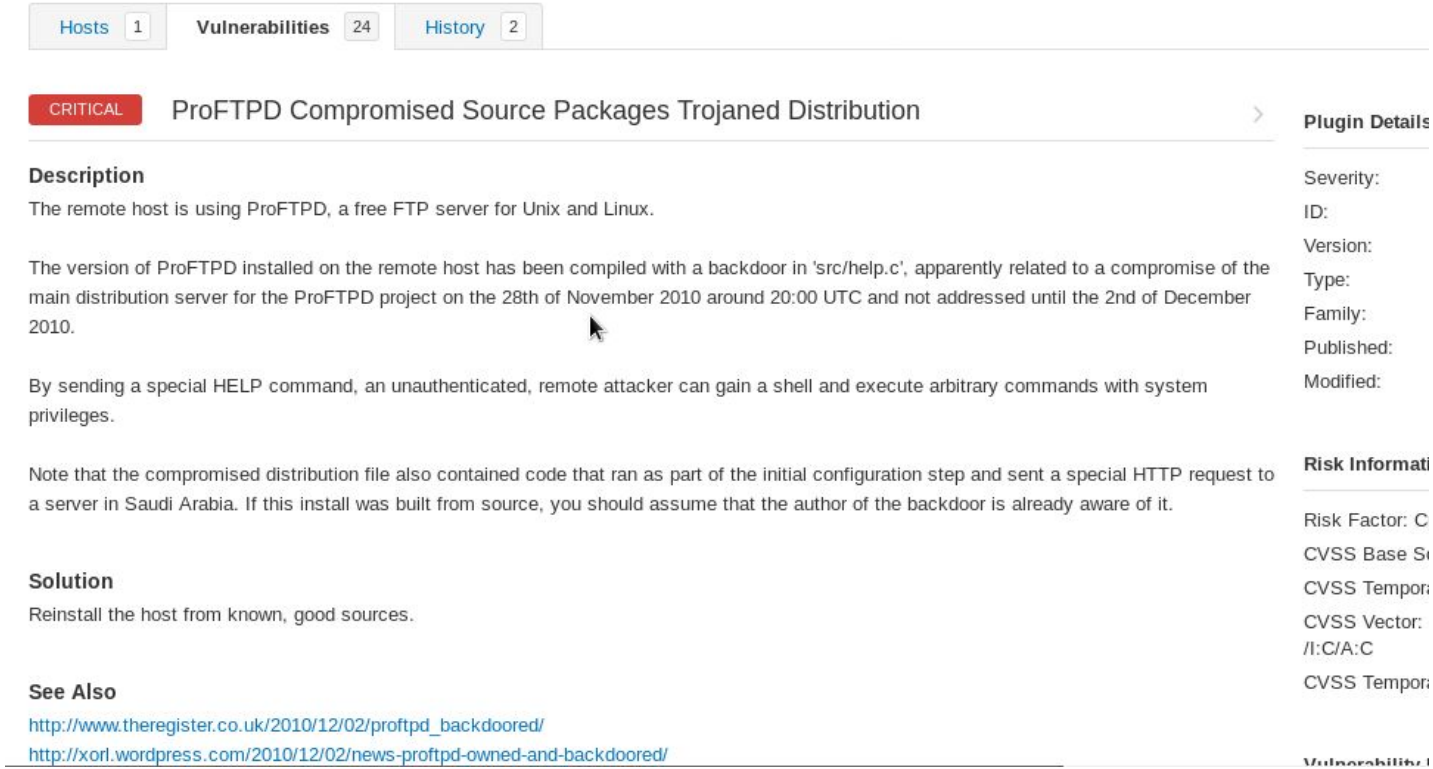
# When the Nessus Complete the Scan



# You can view the scanning result or export it



# Nessus report the problem and suggest solution



# Questions