CND Assessment 2 - 2023/24

Prepared Question

version 1.01

Course Code:	Course Title:	Course Leader:
UC3CND10	Computer Network Defence	Barry Irwin

Changelog:

- v1.01 - Minor typos corrected. Note filename clarification in S10.

This course examines ways in which computer systems can be defended against Cyber Attacks and Threats. During the course students will investigate a number of approaches, techniques and tools for defending computer systems from a variety of threats, covering technical, procedural and physical counter-measures.

This document is the prepared question you should work on in the coming weeks for submission during the examination. The examination timing assumes that work on this activity is complete. There is not time in the examination for you to complete this task.

Title	Contribution to the Course	Due
Prepared Question	25% of EXAM	29/11/2023

Contents

Prepared Question: Practical Task

Date Set:	Due Date:	Word Count:	Contribution:	Hours:
12/10/2023	29/11/2023	-	25% of Examination	20

The pre-prepared exam question for the Final exam assessment for the Computer Network Defence (UC3CND10) course consists of the following tasks.

Background

Maintaining the security of computer systems and defending them against threats is a continuous ongoing process and not an event. A system deemed secure one day can be compromised the next though compromise of other systems with which it has a trust relationship or via an unknown vulnerability. The regular auditing and validation of computing systems is an important part of ensuring the ongoing security. This can become quite challenging as the number of systems being monitored grows. Fortunately it is possible to automate much of this. Automation ensures that it happens regularly, and also allows the checking to happen without requiring staff time. In a typical organisation, the outputs of auditing scripts are sent though to SOC systems where they are then undergo a triage process.

Task overview

Your task is to demonstrate the application of your skills and understanding of a number of concepts that have been covered in the course. This is to be done though the development of **a single** script that can be run on a standard server system, as we have used during the course. This script is to be used to aid the SOC Threat hunting team in checking for emerging threats, and gathering information for further processing. The specific requirements are detailed below.

You are required to develop the script using the `bash` shell, on an Ubuntu 22.04 server system. For your shell script, only those tools covered in the course, or part of the base Ubuntu system at installation can be used. You may assume that all tools are all correctly installed and available. If you have any queries you should clarify these with the course team by **14h00 1 November 2023 (1/11/2023)**.

The scripts will be executed in a virtual environment and the output validated against a known set of changes made on the examination system. There is **not** an option for installing additional packages, modules or software. The execution environment is a standard Ubuntu 22.04 LTS system (as used the course) that will have the latest security patches applied, along with installations of all software directly used in the tutorials.

This is an **individual** task, and you are to submit your own **individual** solutions to the problems.

You should develop your script on the correct platform. Notably, using Kali for your script development very will likely result in serious problems with the final execution.

Task definition

The definition of the tasks required in the script is the core of what you are required to implement. While not explicitly listed in these, you need to consider other 'good practice' when developing your solutions. This would include, but is not limited to:

- Appropriate Formatting and layout.
- Clear definition of the authorship, purpose, inputs and outputs of the script.
- Validation and error checking
- Other surrounding feedback and reporting of the scripts activities as it runs, beyond the few specified outputs defined in the tasks.

The output **must** all be *plain text* and formatted suitable to be passed as direct input to the mail command (as would happen with a task executed in system crontab). This specifically means that fancy colours ANSI /terminal scripted text must not be used as this causes problems with automated parsing.

The user/installation guide for the script should be simple, but complete and sufficient for an junior administrator to follow and successfully install and configure the script. This is **not** intended to be a full user guide with screenshots of every step. Be very clear what is part of the script and what is to be done by the administrator. The script should **not** contain its own installer.

Threat Hunting Script

Your organisation's internal The Threat Hunting Team (THT) publishes a daily list of Indicators of Compromise (IoC) on their internal system. You have been asked to develop a script that can be used to search for these IoC's along with undertaking other related security checks on the organisations Ubuntu Servers. These are required to run with no interaction or human input.

The purpose of your script is to execute checks using this IOC file as defined below, along with generating some logs and a report. The outputs of the script are to be uploaded to a central server on completion. The script will be run via `crontab` between 2 and 3am every day. Output should be kept to a minimum, and limited to statements of pass/fail for each of the checks, as this output will be sent via email, and should be easy to parse for a receiving SOC system.

Further details should be contained in the report file that is uploaded. An example direct calling of the script from the command prompt would be:

```
./tth-IOC.sh https://iocserve.int.org/tth logs.tth.loc.org uploader123
```

Operating Requirements

You should take note of the following as they relate to the environment

K1 Scripts, specific tools and logs relating to this script all reside and operate in `/opt/security`. This is a **read-only** file-system for most of the time to minimise any risk of these being changed.

K2 The server url path can change. This must **not** be hard coded in the script but should be taken from the **first** command line parameter. This is **always** an HTTPS url, and the protocol must be checked to be as such.

K3 The server makes use of certificate issued by a public CA

K4 The IoC file is named based on the day it is created using the format `IOC-`*`yyyymmdd`*`.ioc`

K5 The files are digitally signed by the Threat Hunting Team using GPG as: `IOC-`*`yyyymmdd`*`.gpg`

K6 The public keys needed for validating these signatures are already loaded on the servers.

K7 The IoC file and accompanying signature are contained in the directory pointed to by the url in K2.

K8 The remote server for uploading the report is defined as the **second** command line parameter.

K9 The user identity to be used for access to the upload server is defined as the **third** command line parameter.

K10 The servers only have SSH as a login mechanism.

K11 The environment should be restored one the completion of the script. ie. the script must return the system to the condition it was in before execution.

Functional Requirements

In its operation, your script **must** undertake the following. Use these as a checklist to build up the functionality of your script. For headings for the sectipon they should be descriptive, but start with the relevant **S** value below.

S1 Use the working directory of `/opt/security/working`. Any downloads and/or temporary files must be placed here.

S2 Download the daily IoC file from the URL specified on the command line (first parameter)

S3 Validate the integrity of the IoC file. If it fails see S6.

S4 Validate that the date-stamp in the file is correct. If it fails see S6.

S5 Use the hashes provided to confirm that the `validate` and `strcheck` tools are correct. If either fails see S6. These are used to help defend against possible rootkits. Further Details are available on 7.

S6 **(FAILURE OPTION)** If either the above items fail the script must generate an appropriate error message to STDOUT stating what has failed, and exit at this point Ensure that point K11 is addressed.

S7 For the remainder, the script should handle errors as gracefully as possible, report and log them and continue. The working directory should be preserved as `/opt/security/errors/error-yyymmmdd.tgz` and then included in the upload if present.

S8 Details of the IoC file are given on 7. The script should process this as follows:

- For each **IoC** listed check if it appears in the specified directory. This should must the `validate` command as defined above.
- For each **string** listed check if it occurs in files in the specified directory. This must use the `strcheck` command from above.
- Any matches for the above tasks should be written to a temporary log file in the working directory.
- Any matches should **also** produce the following line to STDOUT, replacing *filename* with the appropriate value:
  `WARN: IOCHASHVALUE filename` or `WARN: STRVALUE filename`
- You should undertake these checks in the most efficient way as possible.

S9 The script then needs to collect, and check some information about the local system and append this to its report along with appropriate headings:

- Currently listening ports (with **no** DNS/port name resolution) and log as file `listeningports` .
- Current firewall rules (with **no** DNS/port name resolution) and log as `firewall` .
- Validate that all files installed in `/sbin`, `bin`, `/usr/sbin`, `/usr/bin` and `/usr/lib` match the valid hashes in the system package database. Any that do not match should be logged to the file `binfailure` and included in the report under an appropriate heading.
- Report **files** in `/var/www` (and subdirectotries) that have been **created** in the last 48 hours, and list **any** SUID/GID files in the same path regardless of modification time.
- Ensure that file systems mounted on `/var/www/images` and `/var/www/uploads` are set as non executable (i.e. scripts cannot run from them). If not, this should be corrected and a warning issues to STDOUT and the report, along with a copy of the system configuration for mounting filesystems.

S10 On completion of the above, the script must merge the gathered files together into the report as appropriate. This should be named `iocreport-`*`yyyymmmdd`*`.txt`. The generated files must be included with the text report into an appropriate archive file called `hostname-tth-`*`yyymmdd`*`.tgz`. There must be a *detached gpg* signature generated before copying both to a remote system. The following are important:

- The keyid to use is `tht2023@tht.noroff.no`
- *hostname* is the name of the host the script is **executing on**.

- In all cases *yyymmmdd* is derived from the date of execution using the YEAR, MONTH and DAY. Values must be zero-padded as required.

S11 The copy process to the central server needs to satisfy the following:

- The copy to the central server is to be done using `rsync` over ssh.
- The user identity passed on the command line must be used. The corresponding ssh identity (key) can be found in `/opt/security/`*`useridentity`*`.id` .
  useridentity will be replaced with what is passed to the script.
- The name of the upload server is taken from the relevant command line parameter passed to the script.
- Files must be copied to the directory `/submission/`*`hostname`*`/`*`YYYY`*`/`*`MM`*`/` on the remote system using the same *hostname* above. The detached signature should be put in the same place. *YYYY* and *MM* are respectively the YEAR and MONTH (1-12) of the date on which the script is executed. Value less than 10 should be zero padded.

S12 After copying is complete, you need to execute appropriate commands on the **remote server** to validate the backup.

- This validation is to be based on a pgp signature of the backup.
- You may assume all keys are loaded on the appropriate systems.

S13 Once successfully complete, the script must report the name, size of the upload (in MB) and sha256hash, along with a final line output stating:
`TTH IoC Check for `*`hostname YYYYMMDD-HH:mm`*` OK`
Ensure to replace *hostname* **and** the *YYYYMMDD-HH:mm* (timestamp) with the appropriate values at the time of execution.

S14 Should any process fail, you must report an error. The error message must be on a new line and starting with start with:
`FAILED `*`Sx-hostname`*` timestamp:`
Where *hostname* is the hostname of the system the script is running on, and a timestamp as described above. Sx is the script requirement step that contained the failure. This would allow for the output to be easily processed, categorised and acted on at the SOC.

S15 The script should be appropriately automated to run daily as specified. All specified output (other than what is directed to be in the report or specific files) must be written to STDOUT, in plain text, where the standard `crond` system will capture it and report via email.

S16 Prior to exit the script should take appropriate measures to clean up after itself, and return the environment to a clean state.

In **addition** to the script, you must include brief instructions on what setup/configuration is needed to get your script working and set up on a system. This includes appropriate entries relating to system configuration to ensure it runs in an automated manner at a specified time. This must be provided in a **pdf** file. This **must** be accompanied by and an overview of the logic and flow of the script.

IoC file format

The format of the IoC file is is as follows (lines starting with a # are comments and should be ignored in processing):

- Datestamp - this should match the filename
- Hashes to validate the two special tools
- A list of IOCs. Each line starts with IOC and is followed by a sha256 cryptographic hash and a directory. The files in this directory and all bellow should be checked against the HASH.
- String based IoC's follow. These are either strings or regular expressions that may indicate problems. They should be run against all files in the specified directory (and any sub-directories).

An example layout is below. *sha256hash* and *directory* would be replaced by appropriate values.

```
# Datestamp this should match the filename
DATE
# The hash value here should be used to check the integrity of the validation tools
# /opt/security/bin/validate and /opt/security/bin/strcheck
VALIDATE sha256hash
STRCHECK sha256hash
# IOC  values to check
IOC sha256hash directory
# Strings follow.  These are strings  that may indicate  problems
STR string directory
```

An example IoC file is shown below.

```
# Datestamp this should match the filename
20231012
# The hash value here should be used to check the integrity of the validation tools
# /opt/security/bin/validate and /opt/security/bin/strcheck
VALIDATE 5730f0e6112870ca638a21167e670502ef7fd0fffc2d438c0420e5ac63ac4c6e
STRCHECK 73abb4280520053564fd4917286909ba3b054598b32c9cdfaf1d733e0202cc96
# IOC  values to check
IOC de9f83707e8eb38b2028d6f4330f6b5c19a3afac49bb63c7eb8a6ff5e565487a  /
IOC ac2bec8f1f09a99571924f6f4ff3075348bc8edfa4859d77292ea37d5edf8014 /var/www/uploads
IOC 888e275738cf32583ee1e9bd3c40d753a46352d2e7bd37779cbf578f942be0fb    /var/www
# Strings follow.  These are strings  that may indicate  problems
STR string directory
STR IFZvbHVtZSBpbiBkcml2ZS /var/www
STR PSEXECscv /data/share/windows
STR "/eval\(|rot13\(/"  /var/www
STR "/r0nin|m0rtix|upl0ad|r57shell|phpshell|void\.ru/"  /var/www
```

Validation tools
The two specialist files `validate` and `strcheck` are just statically linked versions of `sha256sum` and `grep` respectively. They are functionally identical to the standard commands, and are a direct drop-in replacement for the normal tools. Take care that these will need to be accessed via the **full path** as they will **not** be in the system search for commands defined by `$PATH` in the shell.

Test Data

Suitable example test data and sample input will be made available on Moodle well before the submission date. This will be indicative of the types of data to be provided, and used in the assessment. The assessment server will use **different** data conforming to the same format(s). It is the candidates responsibility to consider the generation of additional test cases from this data in order to fully validate their implementation. Scripts emulating output of the sample data will be considered incorrect.

Submission Guide

Your submission must be packaged as a **single *.tgz* (tar-gzip) file** and needs to contain the following files in its root (i.e. **not** in a sub-directory):

- Your Hunting Script - named **tth-*xxxx*.sh**
- A detached pgp/gpg signature file for the script. This **must** be signed using the key **you** provided earlier in the course. These must be named as the script name with a `.sig` appended at the end. For example **tth-*xxxx*.sh.sig**
- An accompanying PDF file containing installation guides/instructions - named ***xxxx*.pdf**

In **all** cases, the placeholder ***xxxx*** must be replaced by the portion of *your* email address **before** the `@stud.noroff.no`.

Take note of the following points relating to the submission of your files. Ensure you have tested your work in a clean environment before submission.

1. Your `.tgz` file **must be named** `CND-XXXX-HUNTER.tgz`.
2. Incorrectly named files will **not** be processed further.
3. Files of other archive types will **not** be processed further.
4. Incorrectly named scripts will **not** be executed by the test system.
5. Incorrectly named PDF files will **not** be extracted and evaluated.
6. Scripts are expected to be able to execute and take parameters as specified.
7. Scripts that fail to execute will **not** be evaluated further.
8. All submissions will be put though testing for AI and similarity.

You must retain a complete copy of your work until the grading process is completed. Should further clarification of your script be needed, you may be required to attend an oral interview. This will be during the marking period.

Prepared Question: Marking and Assessment

Marks for the assignment are gained for the work you complete in terms of its depth. This is not designed to be a superficial assignment.

Process

An overview of the marking process is as follows:

1. tgz files are downloaded from Moodle and processed based on a class list of eligible students.
2. tgz file is copied to a clean instance of the test server, and extracted. The signatures are verified for the script based on your provided keys. If valid, testing will proceed.
3. The script turn is executed (run) on the test environment with parameters provided.
4. There will be several environment configurations the script is run against, to check error handling and cases as described in the specifications. In each case, the output of the script to STDOUT is captured. Where commands are called on a remote server, those are also captured along with output.
5. Recorded output is named appropriately for each case and student, and copied to a central system.
6. The review will compare the output provided against expected output/actions for the test case(s). This will be done using a combination of programmatic and manual inspection. These form the sensor guidelines.
7. On successful completion of testing, Output reports and the submitted PDF are manually reviewed, and graded.
8. Any attempt to damage or adversely impact the test execution environment will result in a zero.

Guiding overview

As a general guidance of expectations:

- Work that is awarded an **A** will show an *in-depth understanding* of the task assigned, with a clear ability to evidence learning and development and individual exploration beyond the material provided. The work clearly demonstrates critical engagement with materials and concepts related to the task and stays upon topic. No grammatical or typographical errors will be present and formatting is consistent, clear and appropriate. Code is well commented, concise, and demonstrates excellent use of methods.
- Work that is awarded a **B** will follow the above guidance, containing no fundamental errors, and few (if any) omissions or inaccuracies, but does not evidence an in-depth understanding or critical engagement with materials. Work clearly demonstrates components *beyond the minimum required*.
- Work that is awarded a **C** will follow the above guidance, but contain some errors, omissions or inaccuracies, but does not strongly evidence an in-depth understanding and/or critical engagement with materials. Elements of commentary and error checking/handling should be present.
- Work that is awarded a **D** will mostly follow the above guidance, containing significant errors, omissions or inaccuracies, although the work may be lacking supporting evidence or descriptions, with a focus more upon the descriptive. Code will be functional but may be lacking elements and/or error checking and/or comments.
- Work that is awarded a **E** will more or less follow the above guidance, compliant to a minimal functional competence level, although it will contain errors or inaccuracies or lack detailed discussion, and will be lacking supporting evidence. Code should still run although with limited capabilities.
- Work that does not pass (**F**) will generally not follow the above guidance, be of a poor standard, will not demonstrate an understanding of the technical concepts, and/or will not provide adequate evidence of engagement, and/or will have significantly poor formatting, layout grammar and presentation. Failure for script to execute.

Weightings

Satisfying the requirements as defined above accounts for a maximum of 75% of the assessment grade for this question. The remainder of the grade for this assessment is based on appropriate formatting of your output, error checking and readability (including comments) of the provided code as well as the accompanying PDF and evaluated as per the guidelines above. The **relative** weightings of individual components are shown in Tables 1.

Table 1: Marking Weightings

Item	***Test***	**%**
1	S1 Working directory	3
2	S2 IOC Download	3
3	S3 IOC Validation	3
4	S4 Timestamp	1
5	S5 Validation tools	8
6	S6 Hard termination	2
7	S8 IOC	10
8	S8 String	10
9	S9 Ports	3
10	S9 Firewall	3
11	S9 Validation	6
12	S9 New files	6
13	S9 Permissions	4
14	S10 Bundle creation	5
15	S11 Upload	6
16	S12 Validation	4
17	S13 Completion Message	2
18	S14 Error handling	5
19	S15 Automation	5
20	S16 Cleanup	8
	Total	**100**

This assignment will be marked out of:	A-F
Contribution to the final course mark:	25% of Examination
Estimated number of hours to be spent on this assignment:	20

A separate grade will not be provided for this question. The final assessment grade and final course grade composed of this and any other assignment(s) will be recorded in the primary student information system.