Realtime HTTP Monitoring with Nmap NSE

Pre-Approval Project 2 - Write an NSE Script
Instructor - Dimple Chauhan
Liam Salazar-Endara | 11/08/2925

Contents

Agenda

1. Introduction - Overview of Nmap and the Nmap Scripting Engine
2. Functions & Libraries - Explanation of NSE Script Functions and Libraries used
3. Script Walkthrough - Step-by-step explanation of our HTTP Title Grabber Script
4. Execution & Output - How to run the script and review the results
5. Conclusion - Key takeaways and future improvements

What is Nmap ?

Who uses it?

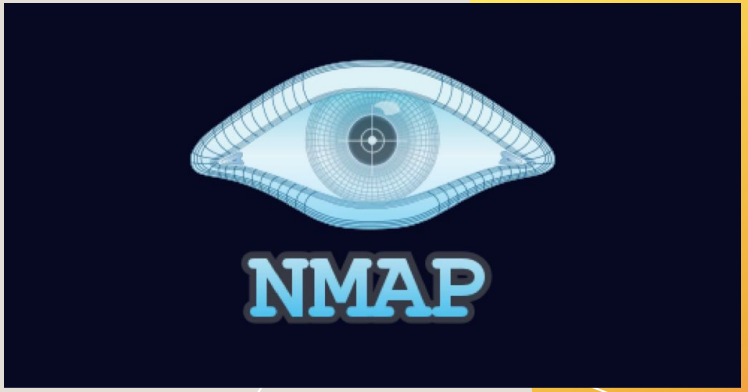
- Nmap, short for Network Mapper, is a powerful and free network scanning tool used by network administrators, security analysts, and cybersecurity professionals.
- It is popular in both enterprise environments and educational settings because it provides a quick and reliable way to understand network structure and identify potential issues.

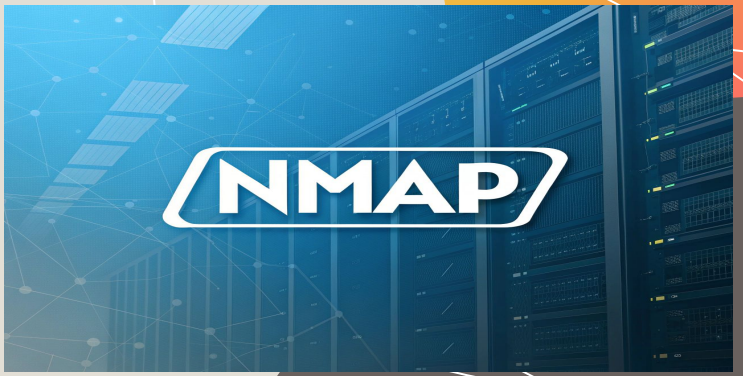
What is it used for?

- Its primary job is to detect live hosts on a network, scan their ports, identify the services running on those ports and make educated guesses about the operating system in use.
- Additionally, Nmap can help map network topologies, check for unauthorised devices, and perform security audits.

How is it used?

- Nmap is widely used for both defensive and offensive security purposes.
- For instance, a system administrators might use it to monitor networks for misconfigured or vulnerable systems., while penetration testers use it to identify potential entry points in a controlled environment.
- Its versatility comes from the combination of port scanning, service detection and scripting capabilities through NSE.





What is Nmap Scripting Engine (NSE)?

- The Nmap Scripting Engine (NSE) is an integrated feature of Nmap that allows users to write custom scripts in the Lua programming language.
- These scripts extend the functionality of Nmap far beyond its default capabilities, letting users perform specific tasks automatically instead of manually scanning.

How is it used?

- NSE scripts can automate a wide range of network tasks that would otherwise be repetitive, time consuming or impossible with a basic port scan. For example, scripts can:
    - Gather detailed information about services running on a host
    - Check for misconfigurations or weak security settings
    - Extract data from web pages or network protocols
    - Identify vulnerabilities or compliance issues

What are the benefits of NSE Scripting?

- Flexible and powerful - Scripts can range from safe information-gathering to advance vulnerability detection or even exploitation.
- Wide selection of prebuilt scripts - Nmap includes hundreds of ready-to-use scripts in categories like discovery, safe, vuln, auth, and exploit.
- Easy to write your own - With Lua, you can quickly create a custom script tailored to your specific needs.
- Time Saving - Automates repetitive checks across multiple hosts or networks, making large-scale scanning more efficient.
- Consistent output - Scripts standardise results, which makes reporting and analysis easier.

Script Purpose

- My project's script is designed to connect to a target web server and repeatedly send HTTP GET requests at timed intervals.
- Instead of capturing just a single snapshot of the server's response, the script polls the server multiple times, recording the HTTP status code, latency, and response size with each iteration
- Each result is timestamped, producing a real-time trace of the server's behavior over time.
- This approach makes the script different from typical NSE discovery scripts.
- Where most NSE scripts return one piece of information and stop, this script shows how a service behaves dynamically, which can reveal short-lived issues such as delays, outages, or sudden changes in response size.
- Because it only uses standard HTTP requests (no intrusive probing), the script is safe to run against public websites or internal organizational servers.
- It falls under the "discovery" category in NSE, meaning it is intended for information gathering and monitoring rather than vulnerability exploitation.

In practical use, this script could help administrators:

- **Monitor website availability**
  - Confirm that a site consistently responds with status code 200 OK.
  - Spot issues like temporary downtime or high latency.
- **Measure performance trends**
  - Track latency values to identify if a web server is responding slower than expected.
  - Detect intermittent spikes in response times that may indicate server load or network problems.
- **Verify service consistency**
  - Ensure the response size remains stable — a sudden change in length might mean content updates, misconfiguration, or an error page.
- **Automate simple health checks**
  - Use the script as part of an automated scan to confirm critical sites are up and behaving correctly.
  - Reduce manual effort by providing structured, timestamped output in Nmap's results.
- **Support troubleshooting**
  - When users report that "the website is sometimes slow or down," administrators can run this script to observe the behavior in real time.

NSE Libraries Used

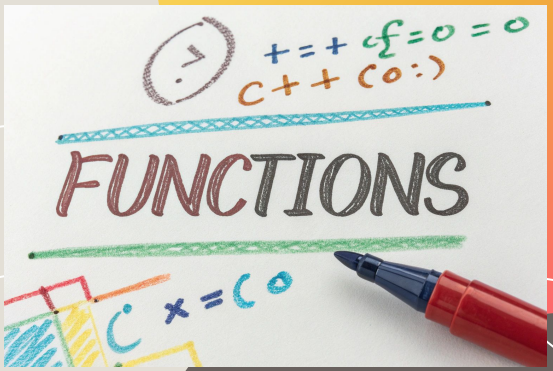
- NSE comes with many built-in Lua libraries that simplify the process of writing scripts.
- These libraries provide ready-made functions for common tasks, so script writers don't have to code everything from scratch.
- In our script, we use 2 key libraries:
    - http - This library handles making HTTP requests and retrieving the HTML content of web pages. It allows our script to easily fetch the homepage of a target server.
    - stdnse - This library provides helper functions for formatting and output, making the results more readable and organised in Nmap's scan report.

- By leveraging these libraries, we are able to write a fully functional script in just a few lines of code, instead of having to implement all the underlying networking and parsing functionality ourselves.
- This makes the NSE scripting both faster and more reliable.

Script Functions (Part 1)

- The portrule function acts as a filter, telling Nmap when the script should run.
- Here, it ensures the script only executes on open TCP ports, which are likely to host HTTP services.
- Prevents the script from running on irrelevant or closed ports.
- Improves scan efficiency which avoids wasting time on hosts/ports that can't respond.
- Adds safety avoiding unnecessary traffic to non-HTTP services.
- It is modular allowing you to modify this rule to target only port 80/443 or a range of web servers.
- The action function is the core of the script — it defines what the script actually does.
- In this script, it:
    a. Reads script arguments (iter, interval, path)
    b. Loops iter times, sending HTTP GET requests to the target URL
    c. Records timestamp, HTTP status, latency, and response length for each iteration
    d. Collects all results in a table
    e. Formats the results for Nmap output using stdnse.format_output

- This is where the "realtime polling" happens — the feature that makes this script unique.
- Handles errors gracefully: if the server doesn't respond, it records "NO RESPONSE".
- Modular and reusable: can be adapted for other monitoring tasks (e.g., polling other paths or services).
- Provides human-readable output, ready for presentation or analysis.



Script Functions (Part 2)

In more complex NSE scripts, helper functions handle repetitive tasks. In this simplified script:

nmap.clock_ms()

- Measures request start/end time to calculate latency in milliseconds.
- Important for real-time monitoring and performance analysis.

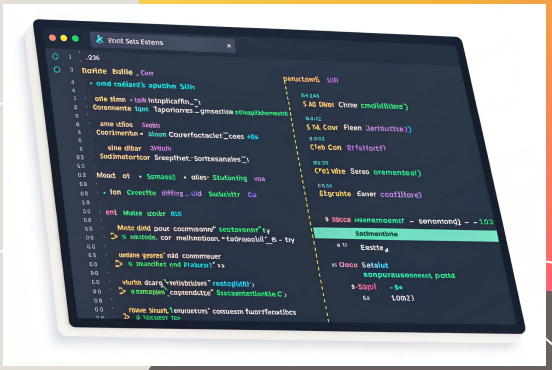
os.date()

- Generates a human-readable timestamp for each poll.
- Crucial for showing when each observation occurred.

table.insert(output, …)

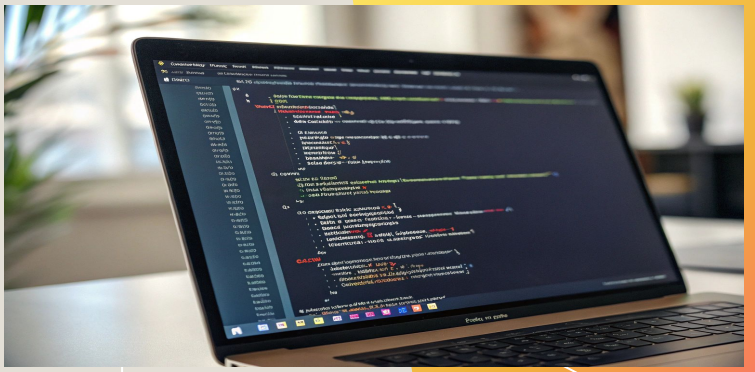
- Adds each iteration's result to an output table, preserving order.
- Makes formatting and returning results to Nmap simple.

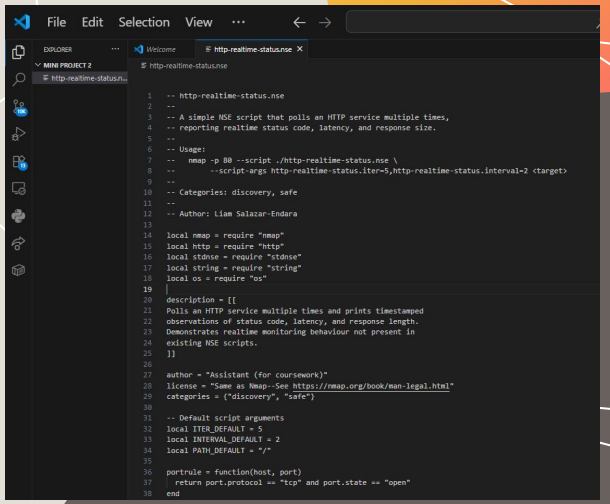
- They make the action function cleaner and easier to read.
- Allows the script to scale or add new features without rewriting core logic.
- portrule and action define the when and what of your NSE script:
  - portrule → ensures the script runs only on relevant targets (TCP open ports).
  - action → defines the polling logic, real-time observations, and output formatting.
- Helper functions support the action by measuring time, formatting timestamps, and storing results.
- This structure makes the script safe, modular, reusable, and suitable for demonstrations or this project.



Script Code (Part 1)

- Created a project folder and saved the script as *.nse (Lua/NSE file).
- Top-of-file comments explain purpose, usage, categories and author — important for documentation.
- require statements load NSE libraries: nmap, http, stdnse, string, os.
- description, author, license, categories provide machine-readable metadata for Nmap/NSE.
- Default constants (ITER_DEFAULT, INTERVAL_DEFAULT, PATH_DEFAULT) set sane behaviour if no args given.
- portrule ensures the script only runs on open TCP ports (avoids irrelevant runs).
- These lines perform *setup* — preparing environment, safety, and discoverability before the scanning logic.





Script Code (Part 2)

The action function is the main logic of the script, executed for each host/port that passes the portrule.

Retrieves user-provided script arguments:

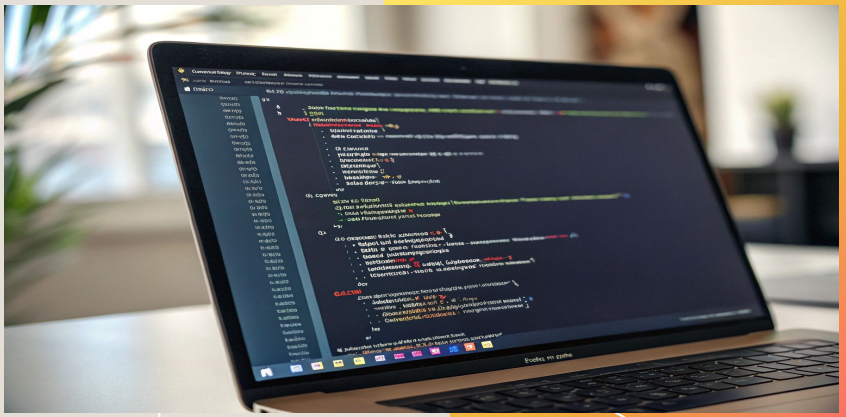
- iter → number of observations (default 5)
- interval → seconds between polls (default 2)
- path → URL path to poll (default /)

Prepares an output table to collect timestamped results.

Loops 1 → iter:

- Records start time (nmap.clock_ms())
- Sends HTTP GET request to target using http.get
- Records end time and calculates latency in seconds
- Extracts status code and response length; handles missing response gracefully
- Generates timestamp using os.date
- Adds formatted line to output table showing: timestamp, status, latency, length
- Waits interval seconds before next iteration

After all iterations, formats and returns output via stdnse.format_output() for clean Nmap display.



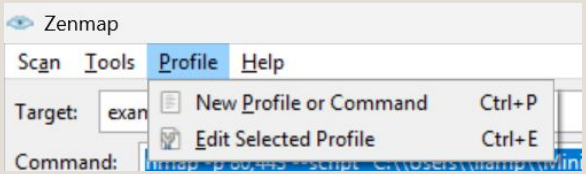
```
action = function(host, port)
  local iter = tonumber(stdnse.get_script_args("http-realtime-status.iter")) or ITER_DEFAULT
  local interval = tonumber(stdnse.get_script_args("http-realtime-status.interval")) or INTERVAL_DEFAULT
  local path = stdnse.get_script_args("http-realtime-status.path") or PATH_DEFAULT

  local output = {}
  table.insert(output, string.format("Realtime HTTP polling (path=%s, iterations=%d, interval=%ds)", path, iter, interval))

  for i=1, iter do
    local start_time = nmap.clock_ms()
    local resp = http.get(host, port, path)
    local end_time = nmap.clock_ms()

    local latency = (end_time - start_time) / 1000.0
    local status = resp and resp.status or "NO RESPONSE"
    local length = resp and #resp.body or 0

    local timestamp = os.date("%Y-%m-%d %H:%M:%S")
    table.insert(output, string.format("[%s] code=%s, latency=%.3fs, length=%d", timestamp, status, latency, length))

    if i < iter then
      stdnse.sleep(interval)
    end
  end

  return stdnse.format_output(true, output)
end
```

Script Execution

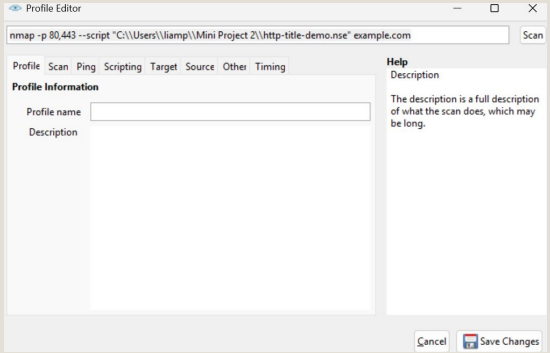
- Before running the Script, the requirement for running the script is that we have Nmap installed in our system available by the Zenmap GUI.
- Then you are going to need to type the correct command in the command prompt in a similar fashion to how it is shown below:

```
nmap -p 80 --script "C:\\Users\\liamp\\Mini Project 2\\http-realtime-status.nse" --script-args http-realtime-status.iter=5,http-realtime-status.interval=2 example.com
```

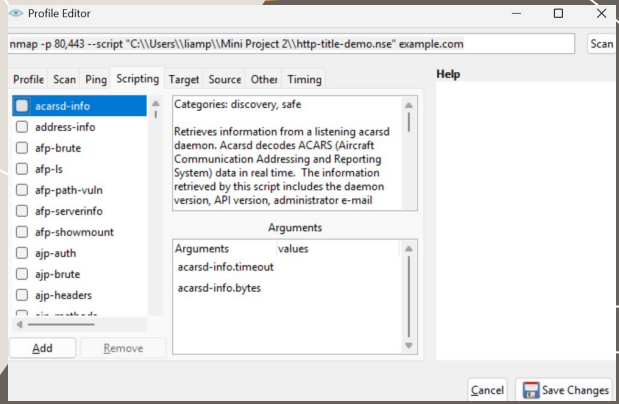
- After typing the command, we are going to head to the toolbar of Zenmap above and we are going to select the option of profile then New Profile or Command:



- Then type a profile name of your choice:



After typing the profile name head to the option of “Scripting” then click on the sub option of “Add” and then look for the folder which contains your nse file and select it then click “Open”. After doing so the final thing that is needed is to “Save Changes”. Finally you are going to exit the “Profile Editor” and run the option of “Scan” on Zenmap.

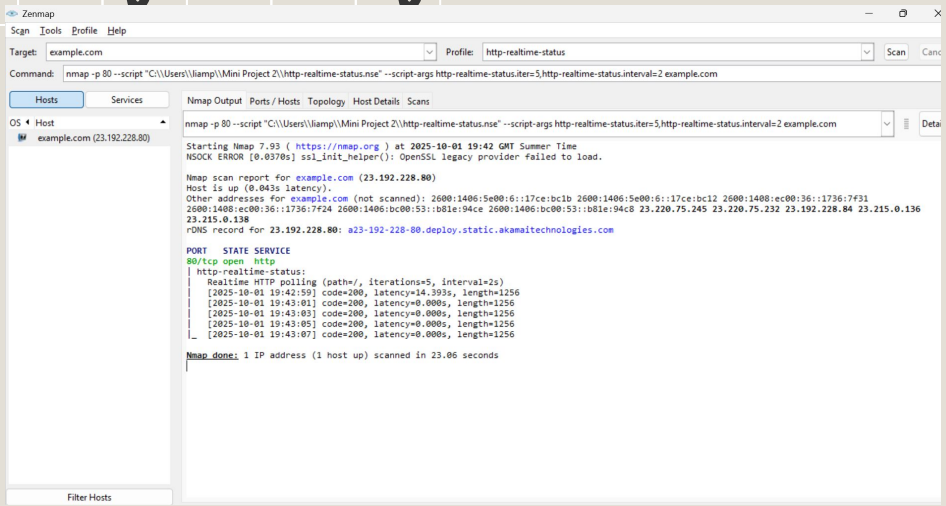


Understanding the Command

```
nmap -p 80 --script "C:\\Users\\liamp\\Mini Project 2\\http-realtime-status.nse" --script-args http-realtime-status.iter=5,http-realtime-status.interval=2 example.com
```

- Runs Nmap to probe a target HTTP service (port 80).
- --script "…http-realtime-status.nse" → executes our custom NSE script (located on disk).
- Purpose of the script is to poll the target repeatedly and produce a timestamped, per-request trace.
- http-realtime-status.iter=5 → perform 5 observations (iterations) during the run.
- http-realtime-status.interval=2 → wait 2 seconds between each observation (controls “realtime” spacing).
- Each observation reports timestamp, HTTP status code, latency, and response length.
- The output is human-readable and appears in the host’s script block inside Nmap results.
- It demonstrates transient behavior (downtime, slow responses, short-lived errors, caching).
- Unlike standard NSE HTTP scripts return a single snapshot — this one provides a time-series trace.
- You can safely run against localhost / lab VM or authorized hosts only (legal and ethical reminder).
- You can change the server state while it runs to show realtime changes.
- Via curl, we can ensure the script path is correct, and watch total runtime ≈ iter * interval.

Script Output



1. Latency (latency=14.393s)
    - Time taken for the HTTP GET request.
    - First request may be slower due to DNS resolution or connection setup. Subsequent requests are faster (0.000s here, possibly cached or rapid measurement).

2. Response Length (length=1256)
    - Number of bytes in the HTTP response body.
    - Helps detect changes in content over time if the site updates.

- Final line (|_)
    - Nmap syntax indicating the end of the script output block.

- Shows Nmap version and scan start time.
- The NSOCK ERROR is a warning about OpenSSL on Windows — it does not affect script functionality because this script doesn't rely on OpenSSL.
- Confirms that port 80 (HTTP) is open.
- The script will only run on this port, controlled by the portrule.
- Header line
    - "Realtime HTTP polling (path=/, iterations=5, interval=2s)"
    - Shows the URL path polled, number of iterations, and time between polls.

- Observation lines
    - Each line corresponds to one iteration of the HTTP request.
    - Format: [timestamp] code=<status>, latency=<seconds>, length=<bytes>

1. Timestamp ([YYYY-MM-DD HH:MM:SS])
    - Records exact time of each poll. Shows script is producing "realtime" output.

2. HTTP Status Code (code=200)
    - Confirms server responded successfully.
    - If server had been down or returned an error, this would show code=404, 500, or "NO RESPONSE".

Conclusion

- In conclusion, this project demonstrates how Nmap and its Scripting Engine (NSE) can be extended beyond one-time information gathering to provide real-time monitoring capabilities.
- We focused on building a safe and lightweight discovery script that polls an HTTP server repeatedly, recording status codes, latency, and response sizes with timestamps. This unique approach transforms a single scan into a time-based view of server behavior, helping administrators and testers observe availability, performance, and consistency in a structured way.
- The project shows how NSE's modular structure — with a portrule to define scope and an action function to implement logic — makes it possible to design customized tools for specific operational needs.
- Looking ahead, this script could be enhanced to handle HTTPS polling, log additional headers such as Server or Set-Cookie, or even export results in CSV/JSON formats for automated monitoring systems.
- Overall, this work highlights the flexibility and power of NSE scripting, demonstrating how even simple scripts can be adapted into practical tools for network administration and monitoring.