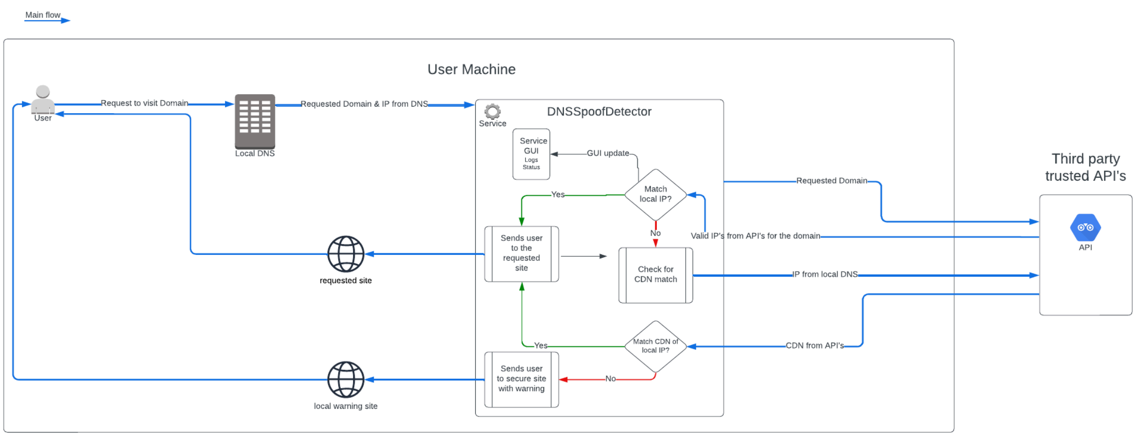
DNS Spoof Detector Tool - Documentation

Project by Barak Shalit 316222280 and Noa David 207465634

Tool objective:

The DNS Spoof detector tool's main objective is to protect users from a **DNS spoofing attack**, in which the user's machine local DNS was compromised and a domain name is now pointing to an IP address that leads to a malicious site made by the attacker.
This kind of attack is causing the user to visit a completely different site then the one he intended to.
By that, the user is vulnerable to several attacks and data theft that he is not aware of.
We believe that our tool does meets its purpose of protecting the user from DNS spoofing attack

Tool logic and flow:



Link to chart

The tool main flow works as follows:

1. On activation, DNSSpoof detector tool scan all user request to the web with tcpdump command on the user's machine.
2. User request to visit domain example.com
3. The tool catches the request form the tcpdump data and parses the requested domain of example.com.
4. The tool checks for the local DNS ip retrieved for the requested domain, example.com, for example 1.1.1.1
5. The tool then proceed to validate the domain for this ip with the following third party trusted API's:
    a. Google dns.
    b. network calc
    c. IP api
6. Each of the API's return the IP corresponded to the domain example.com and the tool checks for the local DNS ip among them (1.1.1.1)

7. If the local DNS ip was found in the responses from API's, and tool mark this request as valid and proceed to the next request.
8. If the ip was not found among the responses, the tool checks for CDN usage on this IP, and does a reverse nslookup on the ip with IP api (checks ISP for 1.1.1.1 and for every one of the returned IP's from the API's).
9. If one ISP from the API's matches the ISP of the local IP 1.1.1.1 (for example - "Akami technologies", then the tool marks the request as valid and proceeds to the next request.
10. Otherwise, the tool mark the request as invalid, mark this in the logs and APP GUI, and prompt the user to a secure site with warning message regarding the attack and the malicious domain, and guide him on how to deal with the attack with informational content (**user education**)

- **Our code is documented as well**

Main components and logic:

1. **Multithreading** - the tool uses multithreading to handle the mass amount of requests received from tcpdump, with each thread in charge of specific task (read raw tcpdump lines from a queue and parse them, read parsed lines and check them with the API's, write to log file, update GUI etc).
2. **GUI** - The tool has a GUI that shows all the scanned requests, a status for the current session, option to open logs file or to stop the detection.
3. **Logging** - all the scanned requests are saved to a log file that the user can later on analyze and search in, from all of his previous sessions.
4. **Round-Robin REST requests** - due to the mass amount of requests received with tcpdump, a logic to handle mass REST requests to the 3 API's had to be implemented for avoiding error codes 429 (too many requests). Web'e decided to go for a round-robin approach with retries mechanism and a caching logic to avoid multiple requests for the same domain and ip.

Challenges during development:

1. Performance issues:

During development we encountered with several performance issues in several points in our tool:

- Using python's **subprocess** to run local commands - several commands we were using line **nslookup** took a very long time to finish so we have to use different python tools to avoid it, like socket library.
- **I/O operation blockage** - we divided the tool logic to several areas on which a thread was in charge to handle - writing to log file, sending REST requests, parsing tcpdump raw input, updating the GUI etc.
- **REST requests 429 error code** - due to the mass amount of requests received with tcpdump, a logic to handle mass REST requests to the 3 API's had to be implemented for avoiding error codes 429 (too many requests). Web'e decided to go for a round-robin approach with retries mechanism and a caching logic to avoid multiple requests for the same domain and ip.

2. **Securly sending REST requests:**
   Due to the fact we are treating the local DNS as compromised, we had to think of a way to send a secure HTTPS requests to the third party API's (otherwise the response we might receive from them might be malicious as well) - to address this issue, we found the IP addresses of those services and used them in our requests - that way we have no need in our local DNS and can get a trusted response from them.

3. **Identification of CDN:**
   During the first phase of the development we were only checking the local DNS ip against the IP's returned from the API's, but this wasn't enough because several sites uses CDN to allocate their IP addresses, so a trusted site could have one ip from our local DNS but completely different IP's from the API's - to solve this problem we developed a reverse nslookup search on the IP to search for common CDN - this solved the problem and still enabled us to identify valid and invalid requests.

Strong and weak areas:

1. **Trustworthy** - during all of our tests, the tool had 100% success rate on discovering malicious sites! he didn't have false positives or false negatives during thousands requests he scanned during the development time - all the "real" sites were categorized as valid 100% of the time and as well of the malicious ones.
2. **Constant protection** - the tool is constantly running on the user machine and scanning all of his requests - such that the user doesn't need to bother checking every single one of his requests and can keep browsing securely.
3. **Friendly and interactive GUI** - our tool has a simple yet helpful user interface window to help the user operate the tool and receive important data.
4. **User education** - the tool not only protects the user from entering a malicious site with a warning message, but also guides him to an informative site in which the user can learn more on the attack and on how to deal with it.
5. **Performance** - although our tool works very fast after the changes we have made, it can still take **several seconds** before the tool will warn the user and stop him from entering the malicious site if the tcpdump queue is filled up with old requests - this is something that can be fixed with writing the tool in another language with better performance but we decided to stick with python for this mini project for simplicity reasons.
6. **API's communication with IP** - in the future the IP's of these API's might change and the tool will have to be updated with the new IP's - this is an easy fix.