

Forum:[Hack Forums](#)

Thread: [c++ CLE Web-Server Stresser/Flooder statically-compiled](#)

nucleo_type09 - April 28, 2024, 6:49 am

recently i've seen that there is an increased number of DDoS attacks on websites, so i decided to write a program that does kind of test the robustness of your server or applications under heavy load conditions. it is itself kind of a DoS tool but made if for simulating such attack not to create one :), also most of DoS/DDoS attacks do not establish https connections, but this one does, here's a little description about it took from my git repo where you can find it...

PS: NOT WINDOWS COMPATIBLE, i know this will piss off most people but it is already huge by itself on linux only OS(+2k LoC) so i didn't want to add more to it, as i also thought that many people using this kind of tools might also have linux as OS or at least vBox -> Linux setup.

Program Description: Genesis (NetFlooder) leverages multi-threading and parallel execution methodologies to concurrently dispatch HTTP requests to designated target endpoints. Each thread operates autonomously, decoupled from the main execution, facilitating independent operation and enabling exponential scaling of throughput, thus achieving unparalleled velocity in request dispatch. The program is designed to simulate flooding requests to specified addresses for educational purposes. It allows users to test the robustness of their web servers or applications under heavy load conditions. However, it should only be used in controlled environments with proper authorization and consent.

This program uses a command-line-like execution, where all inputs are supplied from the command line. For every attack, it will save a copy of the result into a file named the same as the target address in the "scan_result" folder as html file is it will easier later on to dissect and analyse. If the file already exists, it will not be overwritten. If you want to save a new result, you must delete the file first. "main.cpp" is the source file, and "main" is the executable. "scanner.cpp" contains the class and related functions/methods, while the "include" directory contains the "scanner.cpp" header file. "compile.sh" is the shell script you can use for re-compiling. The Program sets default headers to simular the exact behavior of a browser sending a request, headers such ["User-Agent:", "Accept:", "Accept-Encoding:", "Accept-Language:", "Sec-Ch-Ua:", "Sec-Ch-Ua-Mobile:", "Sec-Ch-Ua-Platform:", "Sec-Fetch-Dest:", "Sec-Fetch-Mode:", "Sec-Fetch-Site:", "Sec-Fetch-User:", "Upgrade-Insecure-Requests:", "Cache-Control:", "Content-Type:"], i decided to poison some of those headers because when i tried to curl without some of them, Cloudflare servers would reject the request for some reason, but when used, everything works as expected. The Program Picks a random User-Agent from 10/11 options and binds the corresponding Sec-* fields, additional headers Sec-x are injected into the payload to prevent cloudflare DDos attack mitigation, as some CloudFlare Servers rely on some of the following headers to implement security. static headers are:

Cache-Control
Accept
Accept-Language
Accept-Encoding
User-Agent
Accept
Accept-Encoding
Accept-Language

Sec-Ch-Ua
Sec-Ch-Ua-Mobile
Sec-Ch-Ua-Platform
Sec-Fetch-Dest
Sec-Fetch-Mode
Sec-Fetch-Site
Sec-Fetch-User
Upgrade-Insecure-Requests
Cache-Control
Content-Type

IMPORTANT: The executable was compiled using statically linked libraries for Curl operations. Therefore, the Curl library is not required on your system, as it was statically compiled already. This means everything related to the Curl library is already incorporated into the executable "main.cpp". All you need to do is execute it.

Note: You can use "./main -help" to get all the possible instruction flags available. The instruction set (IS) will be already displayed every time you execute it, but calling "-help" will display more in-depth pseudo-execution and examples. There is also a file named "url_list.txt" located in the src directory which includes some of the urls that r actually ok for this kind of testing, but not suggested to target those for persistent testing anyway... you can replace or add any address you want within "src/url_list.txt".

Compilation Stage

The shell script "compile.sh" facilitates the compilation process for the program. It first checks for the presence of required libraries such as libcurl, libidn2, librt, libdl, libz, libgssapi_krb5, libkrb5, libssl, and libcrypto on the system where the program is being compiled. If any of these libraries are not installed, the script automatically installs them. Once all required libraries are available, the script proceeds to statically compile the program with curl static linking, ensuring seamless execution across different systems.

you can also type -help after ./main to get instructions help...

PoC

this test was performed on my machine.

On github where it is hosted, you will find all the parameters and flags used for the execution, such as speed, rounds, timeout, verbose etc...

repo: <https://github.com/nucleoType98/NetFlood.git>

i do want to say that under some conditions like a speed less than 50000 microseconds for let's say 1000 rounds on my system causes some timeout operations, might be because im using a not really powerful laptop.