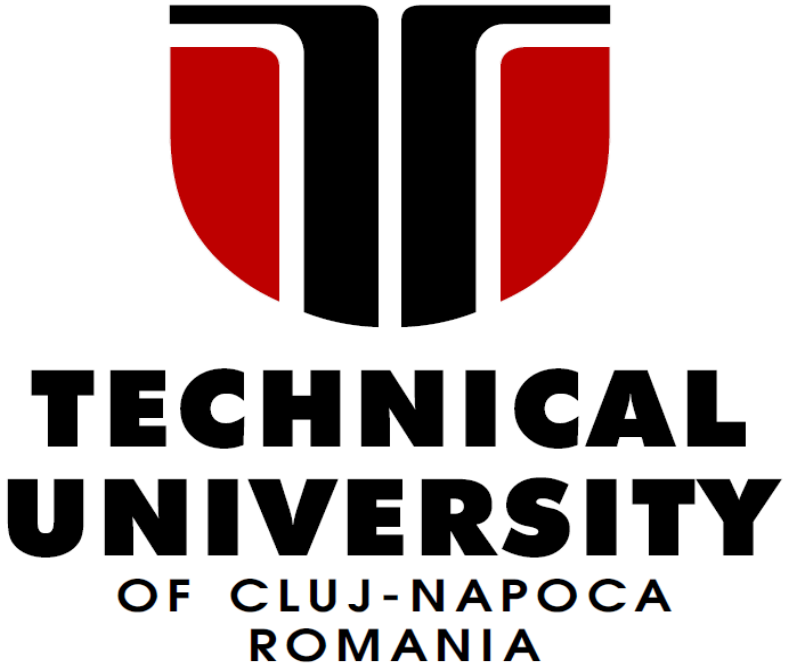

Proiect ASO 2024: Management-ul unui botnet

Documentatie - Etapa 2

Iamnitchi Bogdan - Grupa 30461

Administrarea Sistemelor de Operare
Facultatea de Automatica si Calculatoare
2024-2025

Cerinte rezolvate

Această etapă presupune dockerizarea infrastructurii Mirai, formată din trei containere: `cnc` (serverul de comandă și control), `patient-zero` (primul sistem infectat), și `victim` (o victimă a atacului). Soluția se bazează pe crearea unei orchestrări eficiente cu **Docker Compose** pentru a simula comportamentul malware-ului într-un mediu controlat.

1. Containerul `cnc`

- Acest container găzduiește serverul de comandă și control (C&C), care gestionează atacul și trimite comenzi către sistemele infectate.
- Serverul C&C trebuie configurat pentru a comunica cu containerele `patient-zero` și `victim`.
- Se asigură deschiderea porturilor necesare pentru comunicare.

2. Containerul `patient-zero`

- Acesta reprezintă primul sistem infectat, dar **nu conține malware-ul în sine**.
- După pornire, acest container:
  - Așteaptă ca serverul C&C să devină funcțional.
  - Descărcă malware-ul direct de la serverul C&C.
  - Rulează malware-ul descărcat, conectându-se la serverul C&C pentru comenzi ulterioare.
- Asigură o simulare realistă a procesului de infecție inițială.

3. Containerul `victim`

- Acest container simulează o victimă tipică a atacului, fiind configurat ca un server Linux.
- Conține un serviciu Telnet vulnerabil cu credențiale slabe, care sunt incluse în lista încercată de malware-ul Mirai.
- După ce este identificată și infectată, acest container se conectează la serverul C&C, contribuind la botnet-ul creat.

Orchestrarea cu Docker Compose

Toate cele trei containere sunt configurate să comunice între ele printr-o rețea virtuală Docker definită în fișierul `docker-compose.yml`. Aceasta permite simularea procesului complet al atacului într-un mediu controlat și izolat.

Probleme întâlnite și modul de rezolvare

Să zic sincer, partea asta de dockerizare a fost mai complicată decât m-am așteptat. Una dintre cele mai mari bătăi de cap a fost să configurez rețeaua și IP-urile între containere. Inițial, containerele nu prea se "vedeau" între ele, deși foloseam rețeaua definită în `docker-compose`. A trebuit să mă joc cu setările, să verific dacă am greșit pe undeva, și până la urmă am reușit să le fac să comunice.

Altă problemă mare a fost sincronizarea între `bot` și serverul `cnc`. Serverul `cnc` se deschidea mai greu și `bot` încerca să se conecteze înainte ca `cnc` să fie gata. Am incercat sa fac un healthcheck pe container-ul de cnc dar nu a functionat, am rezolvat utilizan un delay pe bot

Când am ajuns la partea cu `victim`, și acolo am avut ceva probleme. Telnet-ul era destul de dificil de configurat, dar până la urmă am găsit o soluție. În plus, trebuia să mă asigur că Mirai poate scana și infecta corect victima, iar asta a implicat și câteva teste până să fiu sigur că funcționează.

Să mai zic și de faza cu directoarele din Dockerfile, care mi-a dat și ele de furcă. E puțin frustrant să te prinzi unde te afli exact în timpul build-ului, mai ales când folosești `COPY`. La început am tot greșit căile și ori nu copiam fișierele corect, ori ajungeau în alte directoare decât mă așteptam. Mi-a luat ceva timp să-mi dau seama cum funcționează cu adevărat `WORKDIR` și cum influențează asta unde ajung fișierele copiate.

Pe scurt, dacă nu setezi bine `WORKDIR`, fișierele copiate cu `COPY` se duc în locuri random (sau cel puțin așa pare 😅). A trebuit să verific de mai multe ori unde ajung fișierele în container și să ajustez constant calea din `COPY`. Chestia asta chiar te încurcă, mai ales când trebuie să te asiguri că aplicația ta găsește tot ce îi trebuie ca să ruleze.

Pe lângă toate astea, am descoperit că e o idee bună să folosești comanda `docker build` cu `--no-cache` ca să nu te bazezi pe cache-ul Docker-ului, care poate să păstreze greșelile de la build-urile anterioare. Dar și asta m-a prins nepregătit la început.