SDMay19-39: Iseage Traffic Generator

Dustin Ryan-Roepsch, Matthew Vanderwerf, Josh Wallin, Ethan Williams

Background

The cyber defense competition allows students studying security to practice their skill.

Blue Team - Students who have set up services (e.g. web servers) that they need to keep running and protect.

Green Team - Volunteers who verify status of services

Red Team - Penetration testers who attempt to disrupt/ gain access to the servers controlled by the Blue team

ISEAGE - Internet-Scale Event and Attack Generation Environment, internet simulation

Problem Statement

Our project addresses two problems:

1. The green team checks the status of the blue teams servers at regular intervals. Outside these times, the blue team can assume that any traffic they receive is malicious traffic. This enables strategies for the blue team that wouldn’t be feasible in a realistic scenario, like blanket IP banning.
2. In a classroom setting, a professor attempting to teach important software like intrusion detection systems or firewalls doesn’t have realistic malicious traffic to let the students play with.

Requirements

High-Level Requirements

R1. The system shall obscure the Red and Green teams' traffic, to limit the effectiveness of blanket IP banning.
R2. The system shall produce useful traffic for a classroom setting, such as to trigger responses from Intrusion Detection Systems (IDS).
R3. Each type of traffic shall be configurable such that, for example, a task that will perform an ssh attack should be able to be run with different password lists.

Low-Level Requirements

R4. The traffic generator shall accept a list of target IP addresses.
R5. The traffic generator shall be reconfigurable with respect to the attack/traffic types.
R6. The traffic generator shall be reconfigurable without requiring a restart.
R7. The traffic generator shall accept a configuration file.
R8. The traffic generator shall consist of a task producer and a group of task consumers.
R9. The traffic generator shall appropriately rewrite source addresses to obfuscate packet origins.1
R10. The traffic generator shall produce both normal (i.e., non-attack) traffic and attack traffic.
R11. The producer node of the traffic generator shall execute on a virtual machine within the ISEAGE network.
R12. The consumer nodes shall execute within Docker containers housed on the ISEAGE network.

Non-functional requirements

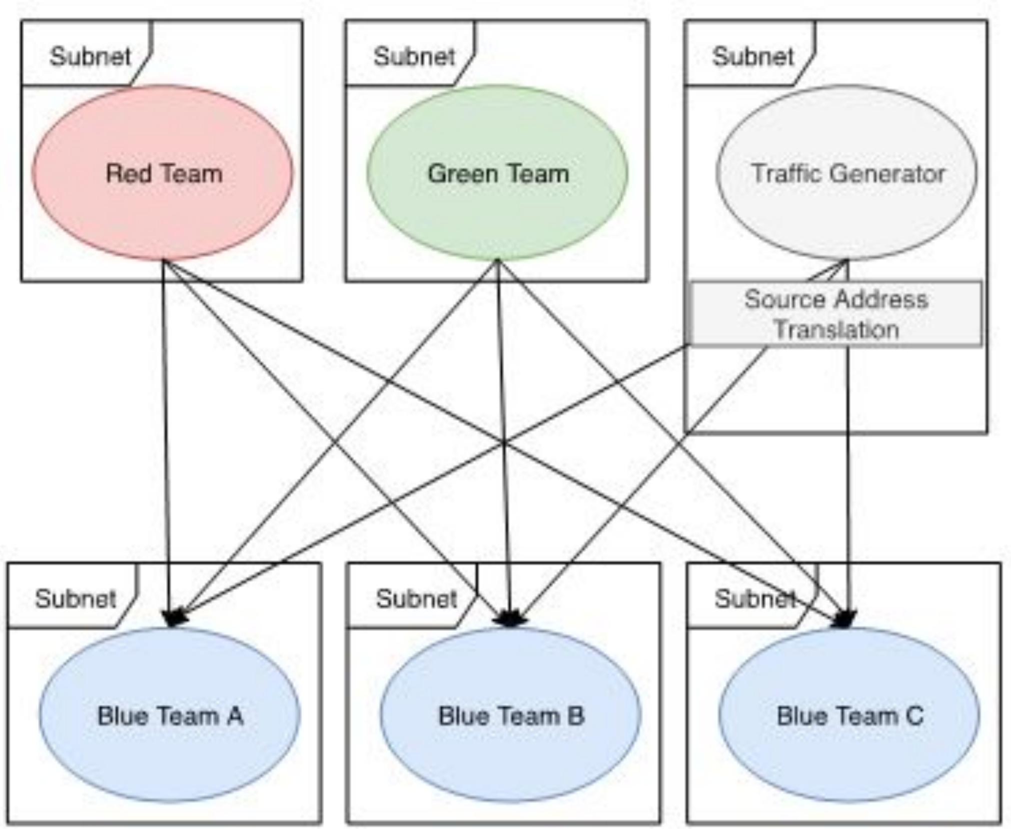
R1. The design shall scale to support a full cyber defense competition.

R2. All software libraries employed shall be licensed such that their use is permitted in both a classroom and competition setting.

R3. The design and implementation shall follow all relevant and reasonable standards, as encountered during their elaboration.

R4. The product shall be sufficiently secured such that the client can reasonably assume outside parties will not have access to critical system settings

Design Diagram



Design

Producer/Consumer Architecture

User Interface- Producer will read from configuration file to build network tasks

Inserts tasks into queue for particular target

Consumer for corresponding target continually reads from queue and executes task, rewriting the source address beforehand

The ISEAGE environment communicates responses back to the appropriate consumer despite altered source addresses

Design Analysis

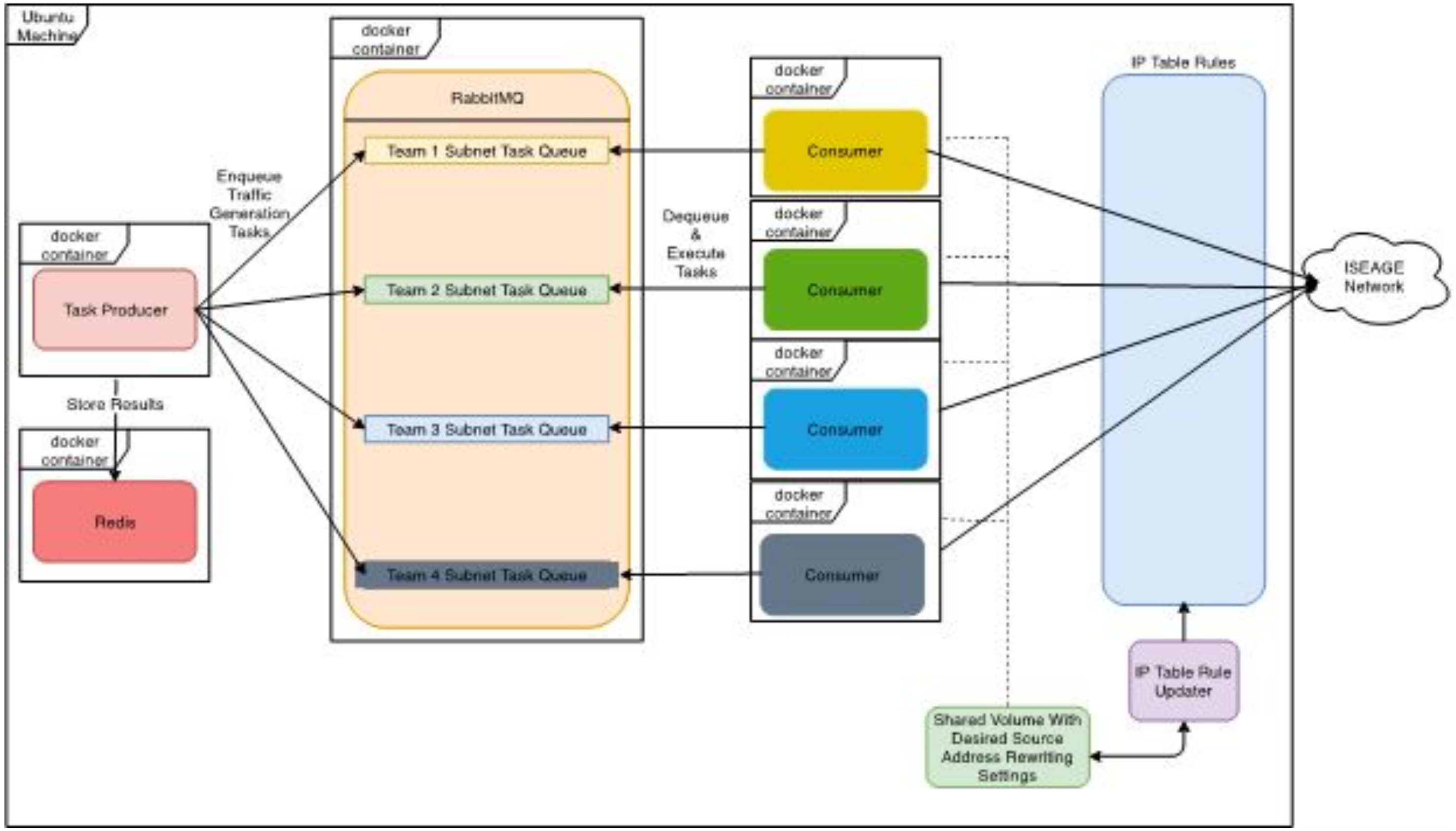
Strengths:

- Efficient
- Scalable
- Simple source rewriting
- Easy test automation

Weaknesses:

- Too many targets could be a stress on system resources
- Not set up to receive return traffic
- Lacks complex traffic types

Implementation Diagram



Software Used

Docker- Our containerization technology which makes it easy to develop and pass off to ISEAGE

RabbitMQ- Our task runner which will contain tasks built by the producer and executed by a consumer to send the traffic to the target

Python- Used for the producer and consumer implementations as well as task and configuration objects

Software Used (cont.)

WGET- Tool installed to consumer containers to facilitate HTTP traffic generation

SNORT- IDS with database containing specification of malicious packets

SSH- An attack type which will try a list of commonly used passwords against the target machine and executed using a Python library.

Technical Challenges

- Source address rewriting went through several iterations before it worked.
- Getting the ISEAGE environment set up initially posed to be harder than we thought, we spent time setting up the environment on ESXI machines from scratch but ended up using a prebuilt solution provided by our client
- Docker wasn’t as “drop in” as we expected for the ISEAGE environment - we spent a lot of time developing outside of the ISEAGE environment because we didn’t have one setup, banking on docker running the same everywhere.

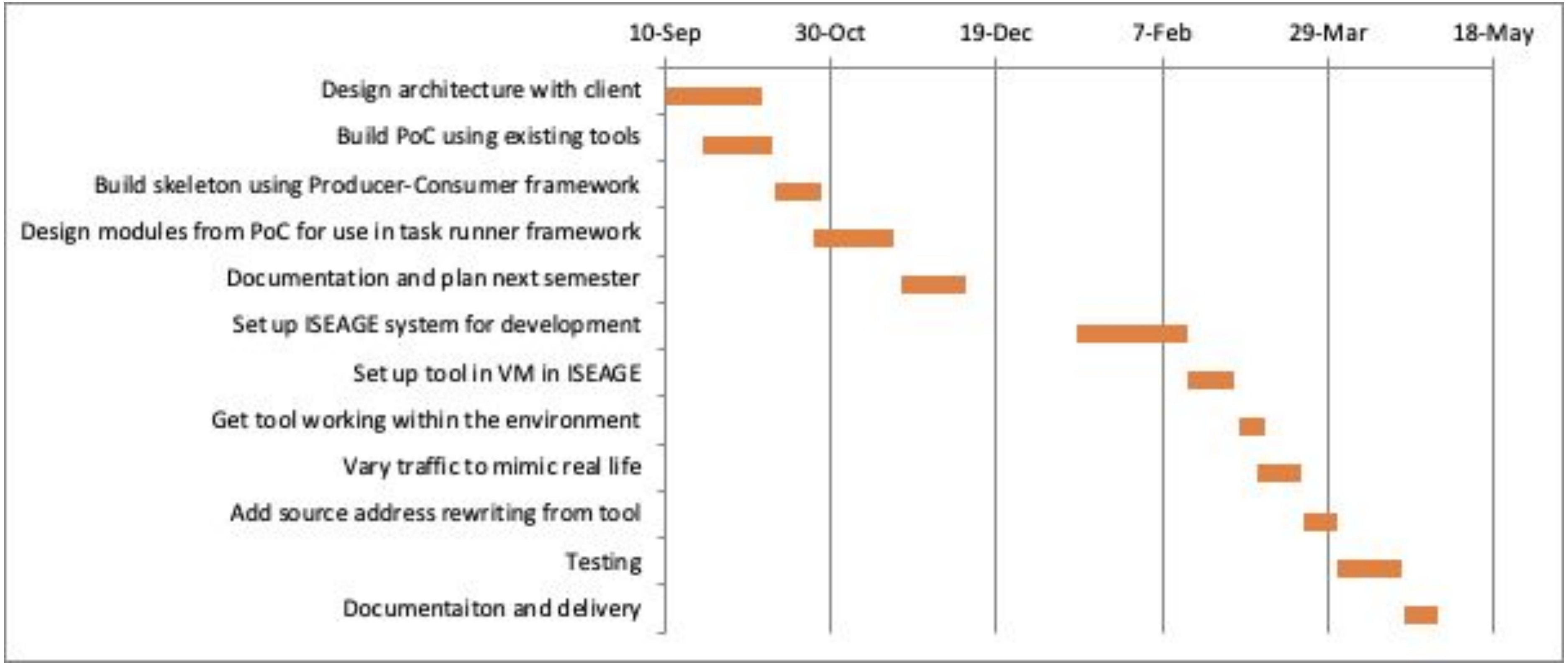
Testing/Evaluation

- A docker-compose configuration file is used to spin up containers for the following testing dependencies:
  - Apache (HTTP Request Testing)
  - Ubuntu (SSH Protocol Testing)
  - Snort (Malicious Packet Detection Testing)
- Environmental variables are inherited from the machine that the containers are spun up on
  - Enables more simplified local and integration testing
  - Important ISEAGE variables such as $HTTP_PROXY, $HTTPS_PROXY can be inherited automatically
    - Failure to have these variables in ISEAGE would result in no internet connectivity (outside ISEAGE)

Testing/Evaluation (cont.)

- Test Cases
  - WGET tests iterating over a list of URLs, requests sent to Apache container
    - Successful if it receives 200 OK responses
  - SSH tests iterating over a dictionary of username/passwords, pointed at Ubuntu container
    - Successful if appropriate username/passwords successfully establish a secure shell
  - Common malicious packet generation
    - Successful if Snort container detects the malicious packets

Project Schedule



Risks & Mitigation Techniques

This tool could be used as an easily scalable botnet that could be deployed on the open internet against any targets

Considering this, we limited the exposure of the tool’s development by:

- Developing in a private repository on Iowa State’s network
- Testing locally and in the ISEAGE environment
- Handing off all the code to the ISEAGE team whose source code is already private

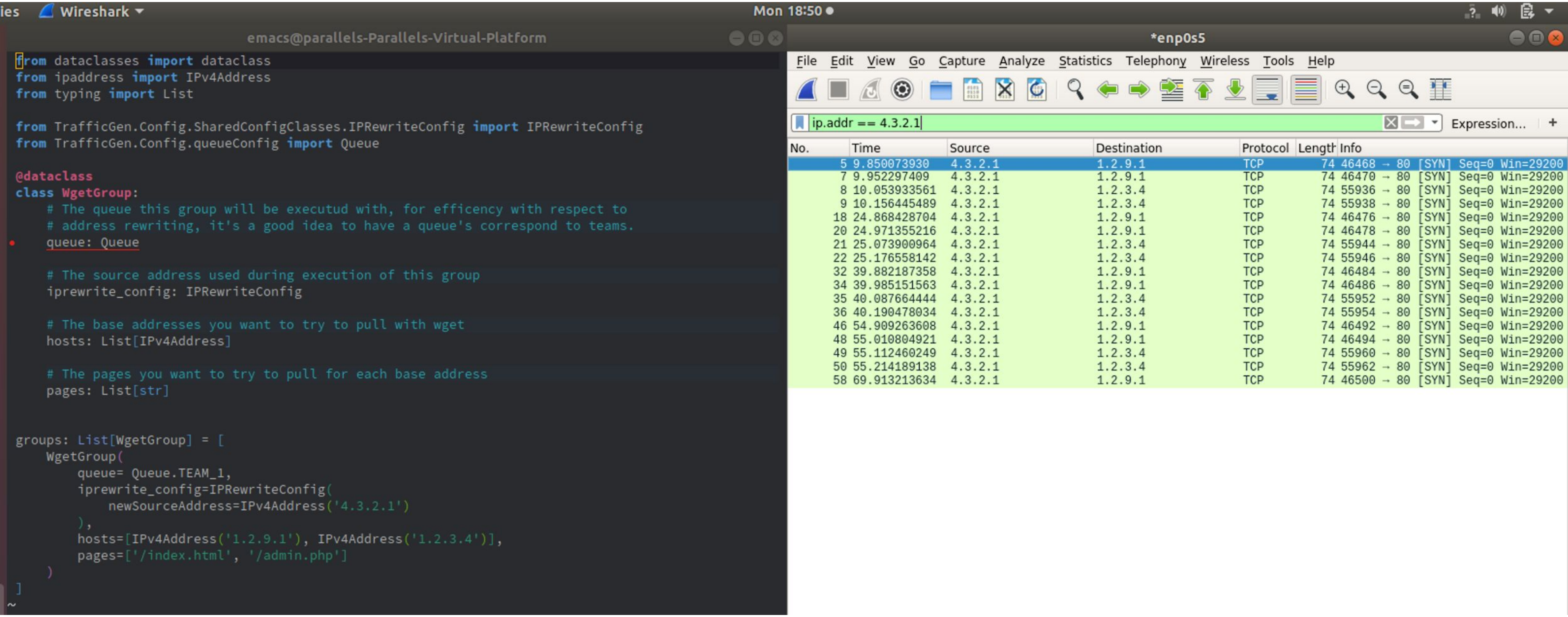
Conclusion

We have several deliverables for our project including

- Configurable tool for sending web traffic to multiple targets
- Rewritable source addresses to obfuscate traffic source
- WGET, SNORT, and SSH traffic types
- Dockerized build for easy scalability and testing
- Virtual Machine to be loaded into ISEAGE environment

Future Work

- Getting return traffic into the tool considering modified source addresses
- Adding more traffic types to more accurately reflect the real internet
- Making the scheduling configuration more robust
- Adding scheduling presets for different use cases
- Automatic deployment to the competition network
- Documentation for how each of the technologies work within the tool


ies
Wireshark
Mon 18:50
emacs@parallels-Parallels-Virtual-Platform
*enp0s5
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
+

```python
from dataclasses import dataclass
from ipaddress import IPv4Address
from typing import List

from TrafficGen.Config.SharedConfigClasses.IPRewriteConfig import IPRewriteConfig
from TrafficGen.Config.queueConfig import Queue

@dataclass
class WgetGroup:
    # The queue this group will be executud with, for efficency with respect to
    # address rewriting, it's a good idea to have a queue's correspond to teams.
    queue: Queue

    # The source address used during execution of this group
    iprewrite_config: IPRewriteConfig

    # The base addresses you want to try to pull with wget
    hosts: List[IPv4Address]

    # The pages you want to try to pull for each base address
    pages: List[str]


groups: List[WgetGroup] = [
    WgetGroup(
        queue= Queue.TEAM_1,
        iprewrite_config=IPRewriteConfig(
            newSourceAddress=IPv4Address('4.3.2.1')
        ),
        hosts=[IPv4Address('1.2.9.1'), IPv4Address('1.2.3.4')],
        pages=['/index.html', '/admin.php']
    )
]
~
```

```
ip.addr == 4.3.2.1
```

```
No.   Time          Source      Destination  Protocol  Lengtł Info
    5 9.850073930   4.3.2.1     1.2.9.1      TCP         74 46468 → 80 [SYN] Seq=0 Win=29200
    7 9.952297409   4.3.2.1     1.2.9.1      TCP         74 46470 → 80 [SYN] Seq=0 Win=29200
    8 10.053933561  4.3.2.1     1.2.3.4      TCP         74 55936 → 80 [SYN] Seq=0 Win=29200
    9 10.156445489  4.3.2.1     1.2.3.4      TCP         74 55938 → 80 [SYN] Seq=0 Win=29200
   18 24.868428704  4.3.2.1     1.2.9.1      TCP         74 46476 → 80 [SYN] Seq=0 Win=29200
   20 24.971355216  4.3.2.1     1.2.9.1      TCP         74 46478 → 80 [SYN] Seq=0 Win=29200
   21 25.073900964  4.3.2.1     1.2.3.4      TCP         74 55944 → 80 [SYN] Seq=0 Win=29200
   22 25.176558142  4.3.2.1     1.2.3.4      TCP         74 55946 → 80 [SYN] Seq=0 Win=29200
   32 39.882187358  4.3.2.1     1.2.9.1      TCP         74 46484 → 80 [SYN] Seq=0 Win=29200
   34 39.985151563  4.3.2.1     1.2.9.1      TCP         74 46486 → 80 [SYN] Seq=0 Win=29200
   35 40.087664444  4.3.2.1     1.2.3.4      TCP         74 55952 → 80 [SYN] Seq=0 Win=29200
   36 40.190478034  4.3.2.1     1.2.3.4      TCP         74 55954 → 80 [SYN] Seq=0 Win=29200
   46 54.909263608  4.3.2.1     1.2.9.1      TCP         74 46492 → 80 [SYN] Seq=0 Win=29200
   48 55.010804921  4.3.2.1     1.2.9.1      TCP         74 46494 → 80 [SYN] Seq=0 Win=29200
   49 55.112460249  4.3.2.1     1.2.3.4      TCP         74 55960 → 80 [SYN] Seq=0 Win=29200
   50 55.214189138  4.3.2.1     1.2.3.4      TCP         74 55962 → 80 [SYN] Seq=0 Win=29200
   58 69.913213634  4.3.2.1     1.2.9.1      TCP         74 46500 → 80 [SYN] Seq=0 Win=29200
```