Scanner de Redes em Python

Gustavo Moura de Sá[1], Rafael Florindo de Mello[1], Eduardo Minghini Sales da Silva[1], Bruno Tavares da Cunha[1]

[1]Instituto Federal de Educação Ciência e Tecnologia de São Paulo – Câmpus Salto
Caixa Postal 13325-047– Salto– SP – Brasil

moura.sa@aluno.ifsp.edu.br, edu.minghini.s.silva@gmail.com,
bruno.tavares@aluno.ifsp.edu.br , rafaelflorindomello@gmail.com

Abstract. *Network scanners are crucial in the identification and analysis of devices, playing essential roles in both administration and cybersecurity. Deployed in corporate environments, they facilitate the management of active devices, optimizing operations. In cybersecurity, they excel in audits and proactive threat detection, identifying vulnerabilities through port scans. Given the current cyber challenges, network scanners emerge as indispensable tools, promoting more secure environments. Our goal is to develop and demonstrate an educational scanner in a controlled environment, utilizing the ICMP echo request technique and Python language libraries.*

Resumo. *Os scanners de rede são cruciais na identificação e análise de dispositivos, desempenham funções essenciais na administração e segurança cibernética. Empregados em ambientes corporativos, facilitam a gestão de dispositivos ativos, otimizando operações. Na segurança cibernética, destacam-se em auditorias e detecção proativa de ameaças, identificando vulnerabilidades por meio de varreduras de portas. Diante dos desafios cibernéticos atuais, os scanners de rede emergem como ferramentas indispensáveis, promovendo ambientes mais seguros. Nosso objetivo é desenvolver e demonstrar um scanner educativo em um ambiente controlado, utilizando a técnica de ICMP echo request e bibliotecas da linguagem Python .*

1. Introdução

Numa era dominada por ecossistemas digitais interconectados, a necessidade de fortalecer a segurança de redes nunca foi tão evidente. A onipresença de ameaças cibernéticas exige ferramentas sofisticadas para proteger informações e infraestrutura. Dentro desse contexto, os scanners de rede em Python surgem como instrumentos vitais, desempenhando papéis cruciais tanto na administração de redes quanto na cibersegurança. Essas ferramentas, projetadas para identificar e analisar dispositivos dentro de uma rede, oferecem utilidade multifacetada. Desde otimizar a gestão de dispositivos ativos em ambientes corporativos até detectar proativamente vulnerabilidades por meio de varreduras minuciosas de portas. No campo da cibersegurança, os scanners de rede se destacam como peças-chave na defesa contra ameaças cibernéticas emergentes.

Em resposta a essa necessidade premente, nosso projeto embarca numa jornada para explorar as complexidades dos scanners, visando não apenas compreender sua importância, mas contribuir ativamente para sua evolução. Este estudo destaca o papel

duplo desses scanners: otimizar as operações de rede e reforçar as defesas de cibersegurança. Ao explorarmos suas aplicações em diversos contextos, desde redes corporativas até auditorias de cibersegurança, o foco do projeto é o desenvolvimento e demonstração de um scanner educativo em Python. Este scanner, meticulosamente elaborado, passará por testes em um ambiente controlado para assegurar não apenas aplicabilidade prática, mas também uma implementação segura dessas tecnologias. Ao combinar insights teóricos com aplicações práticas, nosso esforço busca aprimorar a compreensão e utilização dos mapeadores de rede.

2. Trabalhos Relacionados

[Arkin Ofir, 1999], apresenta o conceito técnico de um scanner de redes e diversas técnicas aplicadas para este mesmo tema. Como no nosso projeto, este artigo aborda diversas ferramentas que utilizam a mesma técnica (ICMP sweep / echo request) e suas peculiaridades em ambientes *UNIX* e *Windows*.

Em [Black Hat Python, 2015], o autor desenvolve diversos códigos na linguagem Python, todos alinhados com a Cibersegurança e, dentre eles, está o código de scanner de redes no qual foi utilizado *sockets* puros, sem o intermédio de ferramentas e bibliotecas, e a técnica de scan *ICMP echo request*.

3. Metodologia

A metodologia adotada para o desenvolvimento do scanner de rede baseado na técnica "ICMP echo request" é descrita nesta seção. A implementação foi realizada utilizando a linguagem de programação Python, aproveitando bibliotecas específicas para manipulação de pacotes de rede e operações relacionadas. A técnica em questão envolve o envio de pacotes ICMP echo request para determinar a disponibilidade de hosts na rede-alvo.

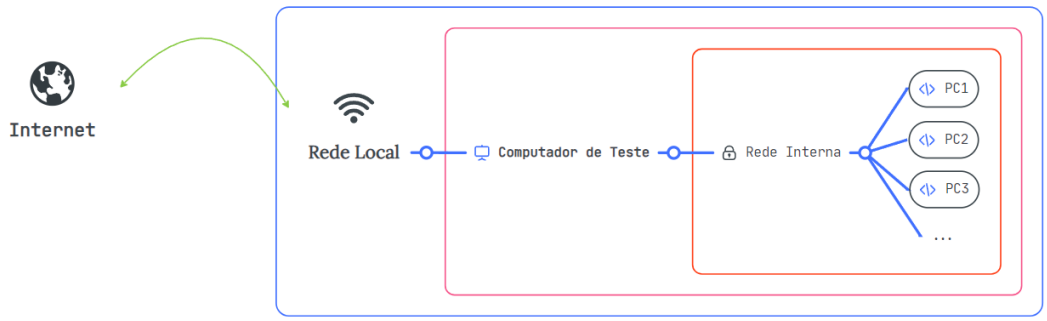

Figura 1. Diagrama da Rede utilizada para testes

O código implementa um scanner que verifica a disponibilidade de portas em hosts dentro de uma determinada subnet. O processo é iniciado pelo envio de pacotes ICMP echo request para os hosts-alvo. Ao receber respostas ICMP do tipo 3 (destino inalcançável), o scanner identifica a presença de hosts ativos na rede. Em seguida, verifica-se a disponibilidade de portas específicas, conforme listadas em "PORTAS_PRINCIPAIS".

```
PORTAS_PRINCIPAIS = [1,3,4,6,7,9,13,17,19,20,21,22,23,24,25,26,30,32,33,37,42,43,49,53,70,79,80,81,82,83,84,85,88,89,90,99,100,106,109,110,111,113,

if sys.argv[1] == "--help":
    print "USAGE => python2.7 scanner.py <HostIP> <Subnet>\n"
    print "Example: python2.7 scanner.py 10.0.0.143 10.0.0.0/24\n"
else:

    # host to listen on
    host   = sys.argv[1]

    # subnet to target
    subnet = sys.argv[2]

    # magic we'll check ICMP responses for
    magic_message = "SCANNER2.0"

    def port_scan(host, porta):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(1)
        if s.connect_ex((host,int(porta))) == 0:
        print "      [*] Porta %s [TCP] aberta\n" % porta


    def multi_process(host, port_scan, porta):
        l = multiprocessing.Process(target=port_scan, args=(host,str(porta)))
        l.start()


    def scan_porta(host):
        for p in PORTAS_PRINCIPAIS:
            multi_process(host, port_scan, p)

    print "================================================================================\n"
    result = pyfiglet.figlet_format("Super Scanner")
    print(result)
    print "================================================================================\n"


def find_SO(ip_header):
    if ip_header.ttl < 65:
    print """
    ============================================================================================================
    =                                       Sistema Operacional: Linux                                         =
    ============================================================================================================
    """
    elif ip_header.ttl == 128:
    print """
    ============================================================================================================
    =                                       Sistema Operacional: Windows                                       =
    ============================================================================================================
    """
def udp_sender(subnet,magic_message):
    #time.sleep(5)
    sender = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    for ip in IPNetwork(subnet):
    try:
        sender.sendto(magic_message,("%s" % ip, 65212))
    except:
        pass

class IP(Structure):
    _fields_ = [
    ("ihl",     c_ubyte, 4),
    ("version",     c_ubyte, 4),
    ("tos",      c_ubyte),
    ("len",      c_ushort),
    ("id",      c_ushort),
    ("offset",      c_ushort),
    ("ttl",      c_ubyte),
    ("protocol_num",      c_ubyte),
    ("sum",      c_ushort),
    ("src",      c_uint32),
    ("dst",      c_uint32)
    ]

    def __new__(self, socket_buffer=None):
    return self.from_buffer_copy(socket_buffer)

    def __init__(self, socket_buffer=None):

    # mapeia constantes do protocolo aos seus nomes
    self.protocol_map = {1:"ICMP", 6:"TCP", 17:"UDP"}

    #enderecos ip visiveis aos seres humanos
    self.src_address = socket.inet_ntoa(struct.pack("@I",self.src))
    self.dst_address = socket.inet_ntoa(struct.pack("@I",self.dst))

    #protocolo legivel a seres humanos
    try:
        self.protocol = self.protocol_map[self.protocol_num]
    except:
        self.protocol = str(self.protocol_num)
```

```
class ICMP(Structure):

    _fields_ = [
    ("type",         c_ubyte),
    ("code",         c_ubyte),
    ("checksum",     c_ushort),
    ("unused",       c_ushort),
    ("next_hop_mtu", c_ushort)
    ]

    def __new__(self, socket_buffer):
    return self.from_buffer_copy(socket_buffer)

    def __init__(self, socket_buffer):
    pass

# create a raw socket and bind it to the public interface
if os.name == "nt":
    socket_protocol = socket.IPPROTO_IP
else:
    socket_protocol = socket.IPPROTO_ICMP

sniffer = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket_protocol)

sniffer.bind((host, 0))

# we want the IP headers included in the capture
sniffer.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

# if we're on Windows we need to send some ioctls
# to setup promiscuous mode
if os.name == "nt":
    sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)


# start sending packets
t = threading.Thread(target=udp_sender,args=(subnet,magic_message))
t.start()

try:
    while True:

    # read in a single packet
    raw_buffer = sniffer.recvfrom(65565)[0]

    # create an IP header from the first 20 bytes of the buffer
    ip_header = IP(raw_buffer[0:20])

    #print "Protocol: %s %s -> %s" % (ip_header.protocol, ip_header.src_address, ip_header.dst_address)

    # if it's ICMP we want it
    if ip_header.protocol == "ICMP":

        # calculate where our ICMP packet starts
        offset = ip_header.ihl * 4
        buf = raw_buffer[offset:offset + sizeof(ICMP)]

        # create our ICMP structure
        icmp_header = ICMP(buf)

        #print "ICMP -> Type: %d Code: %d" % (icmp_header.type, icmp_header.code)

        # now check for the TYPE 3 and CODE 3 which indicates
        # a host is up but no port available to talk to
        if icmp_header.code == 3 and icmp_header.type == 3:

            # check to make sure we are receiving the response
            # that lands in our subnet
            if IPAddress(ip_header.src_address) in IPNetwork(subnet):

                # test for our magic message
                if raw_buffer[len(raw_buffer)-len(magic_message):] == magic_message:
                    print "[*] Host Up => %s" % ip_header.src_address
                    find_SO(ip_header)
                    scan_porta(ip_header.src_address)
# handle CTRL-C
except KeyboardInterrupt:
    # if we're on Windows turn off promiscuous mode
    if os.name == "nt":
    sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)
```

Figura 2. Implementação do Scanner em Python2.7

Além disso, o código incorpora elementos de segurança, como a detecção do sistema operacional do host por meio da análise do valor TTL no cabeçalho IP. A detecção é baseada na premissa de que sistemas Linux e Windows possuem valores TTL típicos diferentes.

Por fim, o código emprega multiprocessing para otimizar o processo de varredura de portas, utilizando múltiplos processos simultaneamente. O uso de threads também é

evidente na função udp_sender, responsável pelo envio assíncrono de mensagens UDP para os hosts da subnet, aprimorando a eficiência do scanner. Este scanner oferece uma abordagem eficaz para identificar hosts ativos e suas respectivas portas abertas em uma rede, contribuindo para a avaliação da segurança e integridade da infraestrutura de rede.

3.1 Bibliotecas Utilizadas:

socket: Utilizada para comunicação de baixo nível, permitindo a criação de sockets e a manipulação de conexões de rede.

os: Utilizada para realizar chamadas ao sistema operacional, necessárias para configurações específicas, como ativar o modo promíscuo em ambientes Windows.

struct: Usada para a manipulação de dados binários, essencial para a construção e análise de pacotes de rede.

threading: Utilizada para criar threads independentes, otimizando a execução simultânea de tarefas.

time: Usada para introduzir atrasos controlados, garantindo a sincronização adequada entre partes do código.

netaddr: Essencial para operações relacionadas a endereços IP e subnet

```
import socket
import os
import struct
import threading
import time
from netaddr import *
from ctypes import *
import sys
import pyfiglet
import multiprocessing
```

Figura 3. Bibliotecas

4. Resultado e Discussão

A execução do scanner resultou na identificação de hosts ativos na rede-alvo, juntamente com informações sobre o sistema operacional e as portas abertas em cada host. Os resultados são apresentados a seguir:

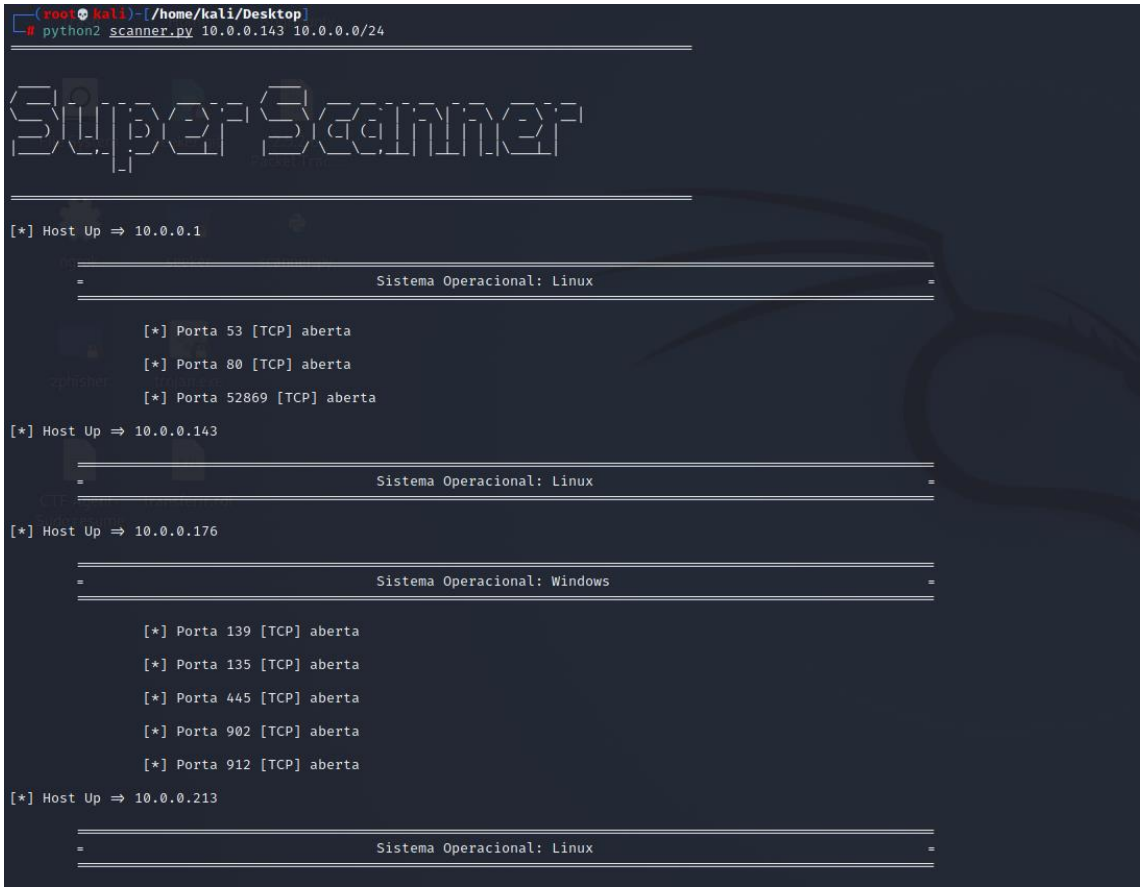

Figura 4. Resultado do Scan

A análise dos resultados revela uma variedade de sistemas operacionais coexistindo na rede monitorada, destacando a diversidade da infraestrutura. A identificação dos sistemas operacionais foi baseada na análise do valor TTL nos pacotes ICMP, conforme detalhado na seção de Metodologia.

Além disso, a enumeração das portas abertas fornece insights sobre a superfície de ataque potencial. No caso do Host 10.0.0.176, um sistema Windows, diversas portas conhecidas associadas a serviços como NetBIOS (portas 135, 139, 445) estão abertas, indicando possíveis pontos de vulnerabilidade.

É importante observar que a presença de uma porta aberta não implica automaticamente em uma vulnerabilidade. A análise de cada serviço associado a essas portas é crucial para determinar o risco real para a segurança da rede.

Os resultados obtidos fornecem uma base sólida para a continuidade da análise de segurança, permitindo a implementação de medidas proativas para fortalecer a infraestrutura contra potenciais ameaças. A abordagem adotada neste estudo destaca-se pela eficácia na identificação de hosts ativos e na enumeração de portas, contribuindo para a avaliação abrangente da postura de segurança da rede.

5. Conclusão e Considerações Finais

A realização deste estudo demonstrou a eficácia do scanner de rede implementado, baseado na técnica de "ICMP echo request", como uma ferramenta valiosa para avaliação da segurança de redes. A identificação de hosts ativos, sistemas operacionais e portas abertas proporcionou uma visão abrangente da infraestrutura de rede, permitindo a detecção de potenciais pontos de vulnerabilidade.

A diversidade de sistemas operacionais identificados ressalta a importância da heterogeneidade na infraestrutura de rede e a necessidade de abordagens de segurança adaptáveis. A detecção de sistemas Windows e Linux, fundamentada na análise do valor TTL nos pacotes ICMP, destaca-se como um elemento crucial para a compreensão do ambiente monitorado.

A enumeração de portas abertas ofereceu insights valiosos sobre a superfície de ataque potencial, especialmente no contexto de sistemas Windows, onde portas associadas a serviços sensíveis foram identificadas. A interpretação desses resultados, contudo, demanda uma análise mais aprofundada, considerando não apenas a presença de portas abertas, mas também a natureza e configuração dos serviços associados.

As considerações finais destacam a importância de integrar o scanner de rede como parte de uma estratégia abrangente de segurança. A análise contínua da postura de segurança, aliada à implementação de medidas corretivas e preventivas, é essencial para mitigar possíveis ameaças e fortalecer a resiliência da rede.

Cabe ressaltar que a interpretação dos resultados deve ser realizada por profissionais de segurança qualificados, considerando o contexto específico da infraestrutura e a natureza dos serviços em execução. A abordagem adotada neste estudo proporciona uma base sólida para futuras investigações e aprimoramentos na detecção e resposta a ameaças em ambientes de rede. Em suma, a combinação de técnicas eficazes e análises criteriosas contribui para o fortalecimento contínuo da segurança cibernética em organizações e ambientes corporativos.