ArchScan: Uma Proposta de Port Scan TCP/UDP

Article · October 2019

CITATIONS
0

READS
9

3 authors, including:

Diógenes Antonio Marques José
State University of Mato Grosso
10 PUBLICATIONS **3** CITATIONS

SEE PROFILE

ArchScan: Uma Proposta de Port Scan TCP/UDP

Karan Luciano Silva[1], **Douglas Teles de Oliveira**[1]
Diógenes Antonio Marques José[1]

[1]Faculdade de Ciências Exatas e Tecnológicas (FACET)
Universidade do Estado de Mato Grosso (UNEMAT)
Caixa Postal 92 – CEP 78390-000 – Barra do Bugres, MT – Brazil

{karanluciano1, douglastelesoliveira}@gmail.com, dioxfile@unemat.br

Abstract. *The Port Scans are programs that aim to identify which computers/services are active and thereby exploit their vulnerabilities. As a result, crackers uses these applications to hack computational systems. Besides that, these programs can be used to perform pentests and therefore help network administrators to strengthen the security of their services available on the network. Thus, the purpose of this article is to develop software capable of scanning TCP/UDP ports. The application, ArchScan (GitHub), was developed in Python and evaluated on a testbed. The evaluation took into account the following characteristics regarding the state of the ports: closed, open and filtered, and as a reference parameter Nmap was used. The results showed that ArchScan detects closed, open and filtered TCP/UDP ports. Also, in some cases, ArchScan scans were faster than Nmap scans.*

Resumo. *Port Scans são programas cujo objetivo é identificar quais computadores/serviços estão ativos e, dessa forma, explorar suas vulnerabilidades. Muitos Crackers utilizam estas aplicações para invadir sistemas computacionais. Todavia, este tipo de programa pode ser usado para realizar pentests e, portanto, ajudar administradores de redes a reforçar a segurança dos serviços disponibilizados na rede. Assim, o objetivo deste artigo é desenvolver um software capaz de escanear portas TCP/UDP. O aplicativo, ArchScan (GitHub), foi desenvolvido em Python e avaliado em um testbed. A avaliação levou em consideração as seguintes características, no que se refere ao estado das portas: fechadas, abertas e filtradas. Como parâmetro de referência foi utilizado o Nmap. Os resultados mostraram que o ArchScan detecta portas TCP/UDP fechadas, abertas e filtradas. Além disso, em alguns casos, as varreduras realizadas pelo ArchScan foram mais rápidas que as do Nmap.*

1. Introdução

Port Scan é um processo de varredura de portas de serviços de redes ativos, que usa os principais protocolos da camada de transporte *Transmission Control Protocol (TCP)* e *User Datagram Protocol (UDP)*. Segundo o *Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil (CERT)* [1], os *Port Scans* são notificações de varreduras em redes de computadores, com o intuito de identificar quais computadores estão ativos e quais serviços estão sendo disponibilizados por eles. Este tipo de *software* é amplamente utilizado por atacantes para identificar potenciais alvos, pois permite associar possíveis vulnerabilidades dos serviços habilitados em um computador. Estes serviços de redes são disponibilizados pelas aplicações através de *sockets*, o *socket* é um conceito de programação de redes *(API)* desenvolvido em 1983 pela universidade de Berkeley e ele possibilita que aplicações se comuniquem com a rede através da tupla ((`'IP'`, `Porta`)) (ex., ((`'192.168.118.2'`, `80`))). Muitos serviços (ex., portas) conhecidos são: *HTTP 80*, *HTTPS 443*, *DNS 53*, *SSH 22*, *Telnet 23*, *FTP 21*, *DHCP 67*, *DHCPv6 547*, etc[1].

A principal motivação para o desenvolvimento deste trabalho consiste no fato de que a maioria dos incidentes de segurança reportados pelo *CERT*, entre janeiro e dezembro de 2018 eram *Port Scan* (ex., mais de 57% deles), em função desta técnica ser um

[1]Para maiores detalhes sobre serviços consulte o arquivo `/etc/services` em sistemas *Linux*.

prelúdio dos ataques virtuais [1]. Há na literatura muitas ferramentas capazes de realizar o *Port Scan*, por exemplo, *Nmap/Zenmap*, *knocker* [2], *Nast* [3], entre outros. Todavia, para testar falhas específicas em serviços de redes e, ao mesmo tempo, evitar os módulos de detecção de *Port Scan* de alguns *Firewalls*, por exemplo, *PFsense*, é necessário desenvolver *Port Scans* específicos. Portanto, o propósito deste trabalho foi desenvolver um *software* capaz de escanear portas abertas e fechadas na rede. O aplicativo foi desenvolvido em *Python* e avaliado em um *testbed*. Os resultados mostraram que a aplicação consegue detectar serviços *TCP* e *UDP* abertos à conexão com precisão e confiabilidade em plataformas *Linux/Windows*.

O restante deste trabalho foi organizado como segue: a Seção 2 apresenta os trabalhos relacionados; a Seção 3 descreve, em detalhes, a proposta de *Port Scan* e a metodologia; na Seção 4 são apresentados os resultados e discussão; e finalmente na Seção 5 são apresentadas a conclusão e as possibilidades de trabalhos futuros.

2. Trabalhos Relacionados

Nesta seção são apresentados alguns programas que abrangem o tema *Port Scan*. Tais *softwares* são de suma importância na área científica e computacional em geral, e servem para exemplificar as funcionalidades do *Port Scan*.

*** *Nmap*[4]*:*** segundo Gordon Lyon [5], o *Nmap* é um utilitário de código aberto (licenciado) gratuito para descoberta de rede e auditoria de segurança. O *Nmap* foi planejado para varrer grandes redes, mas opera bem em *hosts* únicos. Além disso, o mesmo foi nomeado Produto de Segurança do Ano pelo *Linux Journal*, *Info World*, *LinuxQuestions.Org* e *Codetalker Digest*. Ele foi apresentado em doze filmes, incluindo *The Matrix Reloaded*, *Die Hard 4* , *Girl With the Dragon Tattoo* e *The Bourne Ultimatum*.

*** *Pscan:*** é um escaneador de portas *TCP* com funcionalidade parecida à do *Nmap*, ele, também, é capaz de descobrir o endereço *MAC* do destino e o Sistema Operacional. Dentre várias características, o *Pscan* é desenvolvido pelo grupo hacker *Blue Collar Union* e foi projetado inicialmente para *Windows*. Porém uma versão beta para *Linux* pode ser baixada no site da *SourceForge* sob a licença *GPLv2* [2].

*** *Wireshark:*** é o analisador de protocolo de rede mais utilizado no mundo. Ele permite ver o que está acontecendo na rede em um nível microscópico e é o padrão *de fato* (e muitas vezes *de jure*) em muitas empresas comerciais e sem fins lucrativos, agências governamentais e instituições educacionais [3]. Portanto, o *Wireshark* é uma das ferramentas mais completas para captura e análise de tráfego de rede.

3. Descrição da Proposta

3.1. Materiais e Métodos

As técnicas de pesquisas utilizadas neste trabalho foram a revisão bibliográfica e a pesquisa experimental, que tiveram como base referenciais teóricos que tratam de *sockets* [4]. Assim, para o desenvolvimento do aplicativo e execução dos testes, a metodologia foi dada por: **Softwares Utilizados:** Linguagem de programação *Python2.7*, *Nmap*,

[2] http://knocker.sourceforge.net/about.php.
[3] https://www.berlios.de/software/nast/.
[4] O *Nmap*, também, possui uma *GUI* chamada *Zenmap*.
[5] http://nmap.org.

Wireshark, Sistema Operacional *Arch Linux x86_64*/*Ubuntu 18.04 LTS x86_64* e *Socket TCP/UDP*; **Hardware para desenvolvimento:** *Intel i7 Q 740 (8) @ 1.734GHz, GeForce GT 425M, RAM 8GB DDR3* e *SSD 128GB + HD 500GB*; e **Testes:** rede local com dois computadores (cliente/servidor) que foi configurada como classe C 192.168.0.0/24, e para comparar os resultados foram utilizados os *softwares Nmap/Wireshark*.

O aplicativo foi desenvolvido na linguagem *Python* usando *sockets* variados, com e sem conexão. Sua arquitetura é composta de uma classe *ICMP* e treze funções, dentre elas a *tcp_scan()* (Figura 1(a)) e *udp_scan()* (Figura 1(b)) que são as duas principais, já que são as responsáveis por usar os *sockets*: *SOCK_STREAM(TCP)*, *SOCK_DGRAM(UDP)*. Além disso, a aplicação faz uso de várias bibliotecas, por exemplo, *socket* e *Tkinter*. A biblioteca *Tkinter* foi usada para desenvolver a *GUI* do sistema. A função *tcp_scan()* é

Algorithm 1 Função *scan_tcp()*.

Input: IP e Porta.
Output: Aberta ou fechada.

```
1: conn ← AF_INET(), SOCK_STREAM()
2: ret ← conn.connect (target, porta)
3: if (ret == 0) then
4:    SYN ACK (Porta aberta);
5: else if (ret ==11) then
6:    RST (Porta fechada);
7: end if
```

(a) *Algoritmo TCP Socket.*

Algorithm 2 Função *scan_udp()*.

Input: IP e Porta.
Output: Aberta ou fechada.

```
1: s ← AF_INET(), SOCK_DGRAM()
2: s.sendto("Teste teste", (IP, PORTA)
3: data, addr ← s.recvfrom(MAX_BYTES)
4: if (data >= 0) then
5:    Porta aberta;
6: else
7:    Porta fechada;
8: end if
```

(b) Algoritmo *UDP Socket*.

Figura 1. Algoritmos dos Códigos de *Sockets TCP/UDP*.

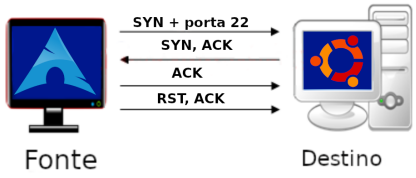
a responsável por varrer as portas *TCP* na rede, ela usa o *AF_INET*, que vai usar o *IP* do alvo a ser escaneado. Também usa o *socket SOCK_STREAM* (de fluxo), este por sua vez faz a conexão entre as entidades cliente-servidor, e usa o *socket SOL_SOCKET* com a opção *SO_LINGER* que quando ativo, caso ocorra um fechamento ou desligamento da conexão, não retornará erro até que todas as mensagens enfileiradas para o *socket* sejam enviadas com sucesso ou atinja o *timeout*.

3.2. Scan com a Porta Aberta *(TCP)*

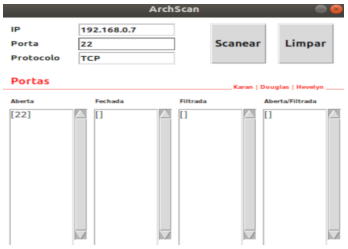
O *Scan TCP* procurará por portas *TCP* (ex., 22, 21, 23, 445) e verificará se a porta de escuta está aberta através do *3-handshake* entre a porta de origem e de destino. Assim, se a fonte enviar um pacote com o *flag SYN* ativo, então a porta de destino deve retornar um pacote com os *flags SYN+ACK* ativos. Se isso acontecer então o serviço está aberto. Além disso, a origem enviará mais dois pacotes um com *ACK* ativo e outro com os *flags RST+ACK* [5], conforme Figura 2(a). Por exemplo, executando a nossa proposta e analisando o resultado com *Wireshark* é possível observar a troca de mensagens, entre fonte e destino, na porta *TCP 22 (SSH)*, conforme supramencionado (Figuras 2(b) e 2(c)).

3.3. Scan com a Porta Fechada *(TCP)*

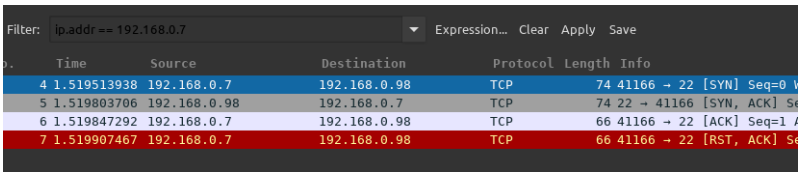
Com relação às mensagens *TCP*, quando a porta está fechada, as Figuras 3(a), 3(b) e 3(c) apresentam os *flags* do *3-handshake*, com o uso do *Port Scan* proposto e *Wireshark*, neste caso para a porta *TCP 23 (Telnet)*. Por exemplo, na Figura 3(a), a fonte envia um pacote *SYN* e o destino retorna os *flags RST+ACK*.



(a) *Scan TCP Open*

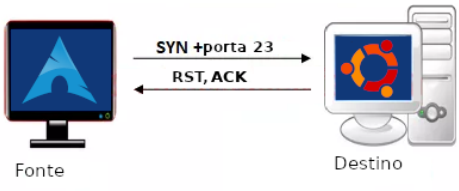


(b) *Resultado da pesquisa na porta 22 TCP aberta*

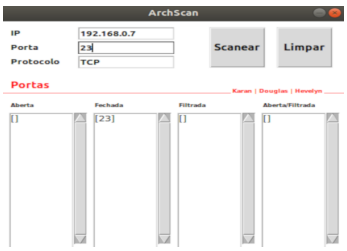


(c) Resultado *Wireshark* na porta 22 *TCP* aberta

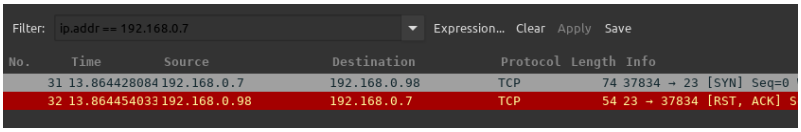
Figura 2. *Scan TCP* para portas abertas.



(a) *Scan TCP Closed*



(b) Resultado da pesquisa na porta 23 *TCP* fechada, usando o *Port Scan* proposto.



(c) Resultado das mensagens de controle *TCP* mostrados pelo *Wireshark* na porta 23 fechada.

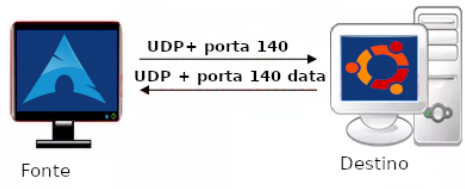
Figura 3. *Scan TCP* para portas fechadas.

3.4. Scan com a Porta Aberta *(UDP)*

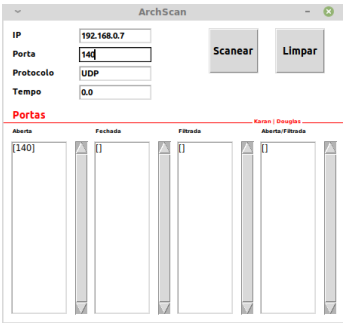
O protocolo *UDP* possibilita às aplicações acesso direto ao serviço de entrega de datagramas, como o serviço de entrega que o *IP* faz. Este protocolo é pouco confiável, sendo um protocolo não orientado à conexão. Não existem técnicas, sem a intervenção da programação[6], neste protocolo para confirmar se os dados chegaram ao destino corretamente. O *UDP*, assim como o *TCP* usa número de porta de origem e de destino de 16 bits (ex., 0-65535). A varredura *UDP* também é possível, embora haja desafios técnicos, pois não há equivalente a um pacote *TCP SYN* nele. No entanto, se um pacote *UDP* for enviado para uma porta que não esteja aberta, o sistema responderá com uma mensagem inacessível da porta *ICMP*. A maioria dos *scanners* de porta *UDP* usa esse método de

[6]Em implementações de *sockets C++* é possível ativar a confirmação de pacotes no *UDP* por meio do *flag MSG_CONFIRM* da função `sendto()`. Fonte: `https://linux.die.net/man/2/sendto`

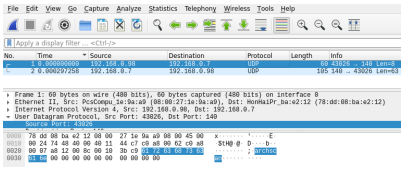
varredura que usa a ausência de uma resposta para inferir se uma porta está aberta. Todavia, se uma porta for bloqueada por um *Firewall*, esse método informará falsamente que a porta está aberta. Assim, se a mensagem de porta inacessível estiver bloqueada, todas as portas aparecerão abertas [6]. A Figura 1(b) apresenta o funcionamento do *Port Scan* proposto com o protocolo *UDP*. Todavia, a varredura *UDP* funciona enviando um pacote *UDP* para cada porta de destino, para algumas portas comuns, como 53 e 161, uma carga útil específica do protocolo é enviada para aumentar a taxa de resposta, um serviço responderá com um pacote *UDP*, provando que está aberto. Se nenhuma resposta for recebida após as retransmissões, a porta será classificada como *aberta—filtrada*. Isso significa que a porta pode estar aberta ou talvez os filtros de pacotes estejam bloqueando a comunicação [5], conforme Figura 4(a). Por exemplo, a Figura 4(c) mostra a máquina fonte (192.168.0.98) enviando à máquina destino (192.168.0.7) um pacote *UDP* na porta 140, com *Len* de 8 *Bytes* "ArchScan". Assim, a máquina destino retorna outro pacote com *Len* de 63 *Bytes*, isso significa que a porta está aberta. As Figuras 4(b) e 4(c) mostram a varredura em uma porta *UDP 140*, varredura feita pelo *Port Scan* proposto e verificada pelo *Wireshark*. A porta *UDP* usada (ex., 140) não é uma porta de serviços conhecida, portanto, para testá-la foi usado um *socket server* programado em *C++*, conforme Figura 4(d).



(a) *Scan UDP Open*



(b) Resultado da pesquisa na porta 140 *UDP* aberta.



(c) Resultado *Wireshark* na porta 140 *UDP* aberta.

(d) *Socket* Servidor *UDP*

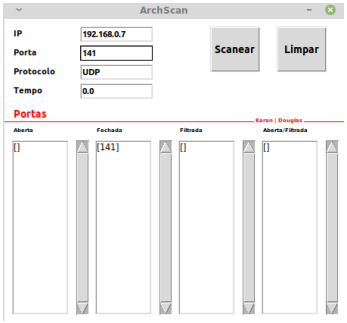
Figura 4. *UDP Scan* para portas abertas.

3.5. Scan com a Porta Fechada *(UDP)*

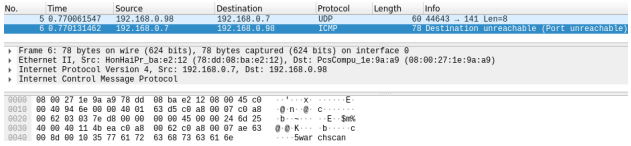
O próximo passo consiste em testar uma porta *UDP* fechada. Assim, a mesma técnica empregada com a porta *UDP* 140 é utilizada na porta 141 (Figuras 5(a), 5(b) e 5(c)).

Agora, como mostra a Figura 5(c), é enviado o mesmo tipo de pacote, porém, na porta 141, que está fechada. Quando a porta está fechada é recebido pela origem uma resposta *Internet Control Message Protocol (ICMP)*. Esse tipo de mensagem é, geralmente, enviada quando um pacote *IP* não consegue chegar ao seu destino ou quando o *gateway*

(a) *Scan UDP Closed*



(b) Resultado da pesquisa na porta 141 *UDP* fechada.



(c) Resultado *Wireshark* na porta 141 *UDP* fechada.

Figura 5. *UDP Scan* para portas fechadas.

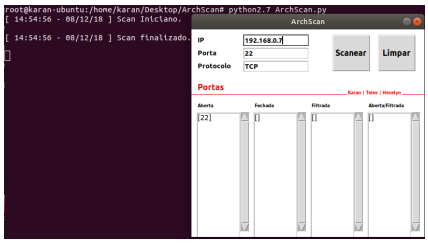
não consegue retransmitir os pacotes na frequência adequada. Portanto, este pacote *ICMP* indica que a porta 141 está fechada.

4. Resultados e Discussão

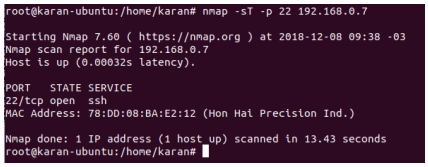
Para propósitos de simplificação, é apresentada a varredura apenas no nó 192.168.0.7. Neste contexto, o primeiro passo constituiu em executar o aplicativo, para isso foi necessário a instalação do *Python2.7* e a execução do aplicativo como *root*. Além disso, serviços de redes podem retornar à varredura três resultados (default): **Aberto:** significa que uma aplicação está ativamente aceitando conexões *TCP* ou pacotes *UDP* nesta porta. Encontrar esse estado é frequentemente o objetivo principal de um escaneamento de portas; **Fechado:** a porta está acessível, mas não há nenhuma aplicação ouvindo nela. Este estado pode ser útil para mostrar que um *host* está ativo em um determinado endereço *IP* (descoberta de *hosts*, ou *scan usando ping*) e como parte de uma detecção de SO. Pelo fato de portas fechadas serem alcançáveis, pode valer a pena escanear mais tarde no caso de alguma delas abrir; e **Filtrado:** não consegue determinar se a porta está aberta porque uma filtragem de pacotes impede que as sondagens alcancem a porta. A filtragem poderia ser de um dispositivo *Firewall* dedicado, regras de roteador, ou um *software* de *Firewall* baseado em *host*. Essas portas frustram os atacantes, pois elas fornecem poucas informações. Assim, foi observado, com a primeira varredura *TCP* (ex., *IP:* 192.167.0.7 e portas 22, 139, 145), que o *software* acertou com precisão as portas escaneadas na rede, levando em consideração o *Nmap* como ferramenta de precisão (Figuras 6(a), 6(b), 6(c) e 6(d)). Quanto ao tempo de execução o *software* desenvolvido neste artigo, *ArchScan*, obteve um resultado melhor do que o esperado, superando o tempo de *scan* do próprio *Nmap*, por exemplo, o *ArchScan* executa a varredura na porta 22 em 1s (Figura 6(a)) enquanto que o *Nmap* faz a mesma tarefa em 13.43s (Figura 6(b)).

Embora o *Nmap* tente produzir resultados precisos, todas as deduções são baseadas em pacotes devolvidos pelas máquinas-alvo ou *Firewalls*, que tem por objetivo aplicar

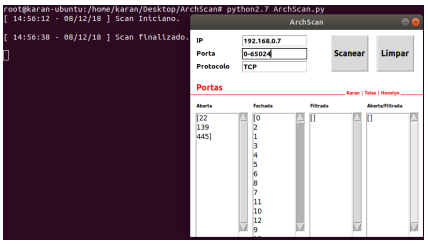
uma política de segurança a um determinado ponto da rede. O *Firewall* pode ser do tipo filtro de pacotes ou *proxy* de aplicações. Tais *hosts* podem ser não confiáveis e enviar respostas com o propósito de confundir ou enganar os *Port Scan's*. Quanto ao *scan* de todas as portas *TCP* na rede, o *ArchScan* obteve um resultado de tempo aceitável se comparado ao *Nmap*, 26s, Figura 6(c). Já o *Nmap* fez a mesma tarefa em 20.58s, Figura 6(d). Já, com relação às portas *UDP*, foram realizados *scan's* na faixa de portas 100-200. Neste contexto, o *ArchScan* levou 5.031s e apresentou a porta 140 aberta (Figuras 7(a) e 7(b)). Já o *Nmap*, na mesma tarefa, gastou 2.65s (Figura 7(c)).



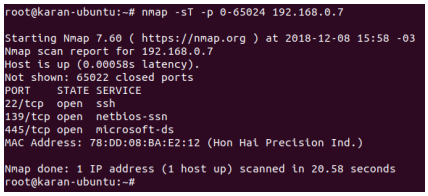
(a) *ArchScan* porta TCP 22



(b) Resultado do *Nmap*, porta 22 *TCP*.

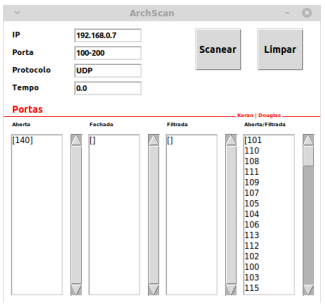


(c) Resultado do *ArchScan*, portas 0-65024 *TCP*

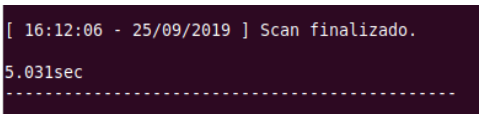


(d) Resultado do *Nmap*, portas 0-65024 *TCP*.

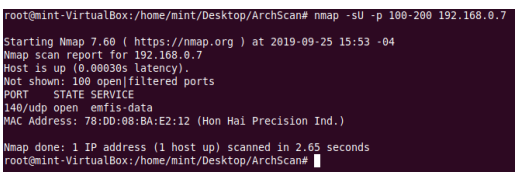
Figura 6. Comparação Scan *TCP* em faixa de portas 0-65024, ArchScan/Nmap.



(a) Resultado do *ArchScan*, portas 100-200 *UDP*.



(b) Tempo de execução do *ArchScan*, portas 100-200 *UDP*.



(c) Resultado do *Nmap*, portas 100-200 *TCP*.

Figura 7. Comparação *Scan UDP ArchScan*/*Nmap*.

Conforme demonstrado, o *ArchScan* consegue realizar as mesmas funções básicas

do *Nmap* (*scan TPC/UDP*). Em alguns casos específicos o *ArchScan* supera o *Nmap* em velocidade de execução.

5. Conclusão

Este trabalho apresentou um aplicativo, *Port Scan*, baseado no conceito de *sockets* e denominado *ArchScan*. Para avaliar a proposta foi utilizado o *Port Scan Nmap*, que serviu como ferramenta de comparação de varredura de portas permitindo, dessa forma, verificar a eficiência do *ArchScan*. O aplicativo proposto foi avaliado em um *testbed* com dois computadores e detectou com sucesso portas *TCP*/*UDP* abertas e fechadas. Além disso, o *ArchScan*, em algumas situações, conseguiu examinar uma faixa de portas mais rápido que o *Nmap* que é uma ferramenta de análise de redes consolidada.

Entre as principais contribuições deste trabalho está o fato do *ArchScan* ser totalmente desenvolvido utilizando *Software Livre*. Além disso, o desenvolvimento do mesmo permitiu a execução de uma atividade prática e de laboratório como um exercício didático da disciplina de sistemas distribuídos. Como trabalhos futuros, pode-se melhorar a eficiência do *ArchScan* na velocidade [7] dos *scans*, e também aplicar técnicas de envio de pacotes como *TCP NULL SCAN*[7] e *TCP FIN SCAN*[8]

Referências

[1] CERT.br, "Incidentes reportados ao cert.br, janeiro a dezembro de 2018." Disponível em: `https://www.cert.br/stats/incidentes/2018-jan-dec/tipos-ataque.html`, September 2019. Acessado em Setembro de 2019.

[2] V. Vieira, "Pscan: Uma alternativa ao nmap.." Disponível em: `https://sejalivre.org/pscan-uma-alternativa-ao-nmap/`, July 2011. Acessado em Setembro de 2019.

[3] G. Combs, "Wireshark." Disponível em `http://www.wireshark.org`, March 2019. Acessado em Setembro de 2019.

[4] W. R. Stevens, *Programação de Rede UNIX: API para soquetes de rede*. Bookman Editora, 2009.

[5] R. Chandel, "Understanding nmap scan with wireshark.." Disponível em: `https://www.hackingarticles.in/understanding-nmap-scan-wireshark/`, March 2017. Acessado em Setembro de 2019.

[6] J. Messer, *Segredos da cartografia de rede: Um guia completo para o nmap*. Professor Messer, 2007.

[7] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pp. 211–225, IEEE, 2004.

[7] Este tipo de *scan* ativa os *flags TCP* para zero. Dessa forma, se um *flag RST* é recebido, indica que a porta está fechada.

[8] Este tipo de *scan* envia um pacote com o *flag FIN* ativado para terminar uma conexão *TCP*. Portanto, o emissor do *FIN* espera elegantemente pelo término da conexão. Assim, se um *flag RST* é recebido, indica que a porta está fechada.