Simulação de ataque DDOS através da ferramenta CORE network emulator

Clarel de Menezes Spies Filho, Luiza Daiane Rabuski

[1]Departamento de Computação – Universidade de Santa Cruz do Sul (UNISC)
Caixa Postal 96.815900 – Santa Cruz do Sul – RS – Brasil

{clarel,luiza}@mx2.unisc.br

Abstract. *This article presents the operation of the tool Common Open Research Emulator (CORE) to emulate computer networks, also explains the concepts about attack distributed of service negation (DDoS) and performs a simulation and analisys of an attack throught the tool.*

Resumo. *Este artigo apresenta o funcionamento da ferramenta Common Open Research Emulator (CORE) para emular redes de computadores, também explica os conceitos sobre ataque distribuído de negação de serviço (DDoS) e realiza a simulação e análise de um ataque através da ferramenta.*

1. Introdução

A ferramenta CORE (Common Open Research Emulator) proporciona um ambiente capaz de emular uma rede completa através da virtualização de equipamentos como hosts, hubs, switches, roteadores e meios de comunicação, entre outros. Esse tipo de ambiente também fornece uma estrutura para simular diferentes tipos de ataques, como os categorizados como ataque distribuído de negação de serviço, do inglês Distributed Denial of Service (DDoS) [Kumar et al. 2012].

O ataque DDoS tem como objetivo fazer com que o serviço da vítima sofra degradação da qualidade ou fique indisponível através da sobrecarga de tráfego de rede a partir de múltiplas fontes [Oliveira et al. 2007], visando a superar o limite de acessos simultâneos que o servidor alvo suporta, o que resulta na reinicialização ou travamento total do servidor. Faremos a simulação de um ataque distribuído de negação de serviço (DDoS) através da ferramenta CORE.

2. Common Open Research Emulator (CORE)

Common Open Research Emulator (CORE) é uma ferramenta que permite emular redes de computadores em uma ou várias máquinas. É possível criar uma representação real de rede de computadores, podendo também ser conectada a redes físicas, roteadores ou outras instâncias do aplicativo. Há um ambiente para execução de aplicações e protocolos reais, aproveitando a virtualização fornecida pelo sistema operacional Linux ou FreeBSD [Core-dev 2012].

O CORE é tipicamente usado para simulação de redes de pesquisa e protocolos de pesquisa, demonstrações, teste de aplicativos da plataforma, avaliação de cenários de redes, estudos de segurança e aumentar o tamanho das redes de teste físico [Core-dev 2012].

A ferramenta foi desenvolvida por um grupo da Boeing Divisão de Pesquisa e Tecnologia do Laboratório de Pesquisas da Marinha dos Estados Unidos. Consiste de uma interface gráfica GUI, que serve para desenhar as topologias de rede e configurar os equipamentos, tais como roteadores e hosts, ilustrada na figura abaixo [Core-dev 2012].

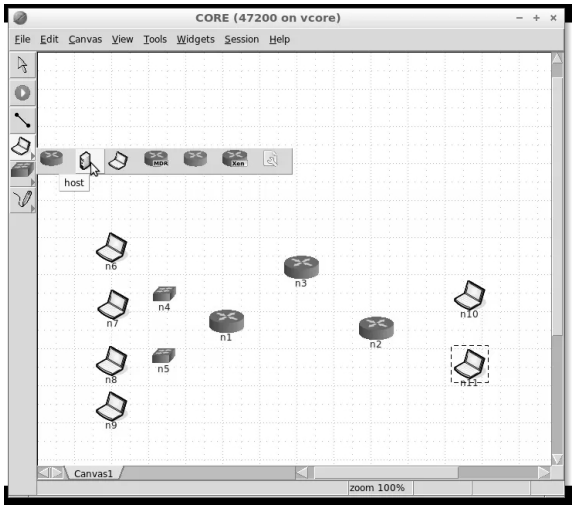


Figura 1. Adicionando nós ao simulador de rede CORE

A ferramenta tem dois modos de operação: Editar e Executar. Por padrão, é iniciada no modo de edição, onde é apresentada uma tela em branco e uma barra de ferramentas no lado esquerdo da tela. O usuário irá desenhar um cenário de rede utilizando as ferramentas do modo de edição e, em seguida, executar a simulação pressionando o botão verde Iniciar [Core-dev 2012].

2.1. Arquitetura

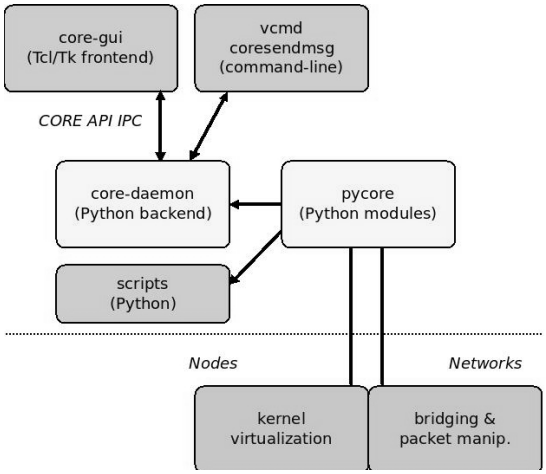


Figura 2. Arquitetura do CORE

Os principais componentes do CORE são mostrados na figura 2. O componente core-daemon (backend) administra sessões de emulação. Ele constrói redes emuladas usando o componente kernel virtualization para nodos virtuais e o componente bridging and packet manipulation para redes virtuais. O componente core-daemon é controlado

através da interface gráfica do usuário, o CORE-GUI (frontend). Ele utiliza módulos Python que podem ser importados diretamente por scripts (Python). Os componentes que usuário interage são os de cor azul: CORE-GUI (frontend), scripts (python) ou ferramentas de linha de comando (vcmd coresendmsg command-line) [Core-dev 2012].

O sistema é modular para permitir a mistura de componentes diferentes. O componente de redes virtuais, por exemplo, pode ser realizado com outros simuladores de rede e emuladores, tais como NS-3 e EMANE. A API do CORE é baseada em sockets para permitir a possibilidade de executar componentes diferentes em diferentes máquinas físicas [Core-dev 2012].

3. Distributed Denial of Service (DDoS)

A negação de serviço distribuída (DDoS) é uma tentativa de fazer com que um serviço online fique indisponível por sobrecarregá-lo com tráfego a partir de múltiplas fontes [1]. Em um ataque distribuído de negação de serviço, um computador denominado Mestre escraviza várias máquinas através de alguma vulnerabilidade encontrada nelas, podendo ter em seu comando até milhares de computadores e as fazem acessar um determinado recurso em um determinado servidor todos no mesmo momento [Oliveira et al. 2007].

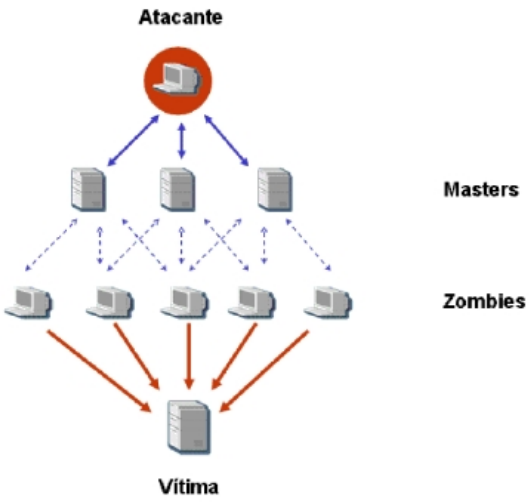


Figura 3. Ilustração de um ataque DDoS

Assim, todas as máquinas escravizadas acessam juntamente e de maneira ininterrupta o mesmo recurso de um servidor, impossibilitando-o de atender a qualquer pedido, já que há um número limitado de acessos que os servidores conseguem atender ao mesmo tempo. Um ataque pode simplesmente reiniciar o servidor ou causar travamento total dependendo do recurso que foi vitimado [Oliveira et al. 2007]. Veremos diferentes tipos de ataques que podem ser realizados.

3.1. SYN Flood

O objetivo do SYN Flood é consumir os recursos de rede e processamento da vítima. Basicamente ele tem o objetivo de enviar mais mensagens do que o servidor pode suportar. Estas mensagens possuem pacotes SYN que requisitam a abertura de uma conexão com

o alvo, o servidor por sua vez responde que aceita a conexão e fica ocioso esperando por uma resposta do cliente confirmando que a conexão foi estabelecida. O servidor fica sobrecarregado de conexões em espera, e fica impossibilitado de abrir novas conexões consideradas legítimas que seriam feitas pelos clientes do serviço [Orozco et al. 2014].

3.2. Ping Flood

Através deste ataque inundaremos a rede do servidor com a intenção de sobrecarrega-la através da função Ping com mensagens do protocolo ICMP. Esta função tem como objetivo testar se há conexão fim a fim, a máquina cliente envia uma mensagem echo request e espera por uma mensagem echo reply para verificar se o destino está acessível e o tempo de resposta [Orozco et al. 2014].

Através de várias máquinas executando o comando Ping para um mesmo destino, podemos sobrecarrega-lo com mensagens echo request e echo reply, podendo congestionar a rede e fazendo com que o servidor não consiga responder de forma eficiente ou de forma alguma a um cliente que esteja requisitando um serviço, dependendo do tamanho do ataque [Orozco et al. 2014].

Este ataque pode ser realizado facilmente como um ataque DoS, com apenas uma máquina enviando mensagens para o servidor. Através de parâmetros da funcionalidade Ping podemos definir um grande número de mensagens para ser enviado rapidamente, sem precisar esperar por uma resposta do servidor. Se a conexão do servidor tiver uma capacidade menor do que a do atacante, a rede não suportará a leitura e resposta de todas as mensagens recebidas da função Ping e negará serviço ao cliente que tentar requisita-lo [Orozco et al. 2014].

Para servidores de grandes sites onde a capacidade da rede excede em muito a de uma rede doméstica é necessário então um ataque DDoS. Será necessário diversos computadores executando o ataque para que a rede seja afetada.

3.3. UDP Flood

O ataque de UDP Flood é muito similar ao ataque realizado com Ping, ele tem como objetivo congestionar a rede do servidor através do envio de diversos pacotes UDP para o servidor. Os pacotes UDP são apenas mensagens que serão entregues e não respondem se foi enviado com sucesso ou não, para o cliente que originou a mensagem [Orozco et al. 2014].

4. Cenário para simulação

Nosso cenário de emulação montado no CORE é formado por quatro máquinas chamadas de zumbis para representarem os atacantes, uma máquina cliente e uma máquina servidor todos ligados a um Router e abstraindo sua rede interna para uma fácil compreensão. A rede interna do servidor teve a capacidade limitada em 64.000 Kbps para que possamos enchergar o efeito do ataque DDoS em um ambiente reduzido.

Apresentaremos como o cenário funciona em condições perfeitas, onde os zumbis não estão realizando ataques e apenas o cliente tenta acessar o serviço. Também apresentaremos o ambiente no momento em que os zumbis estão realizando o ataque e um cliente tenta acessar o serviço, para analisarmos o efeito causado pelo ataque DDoS.

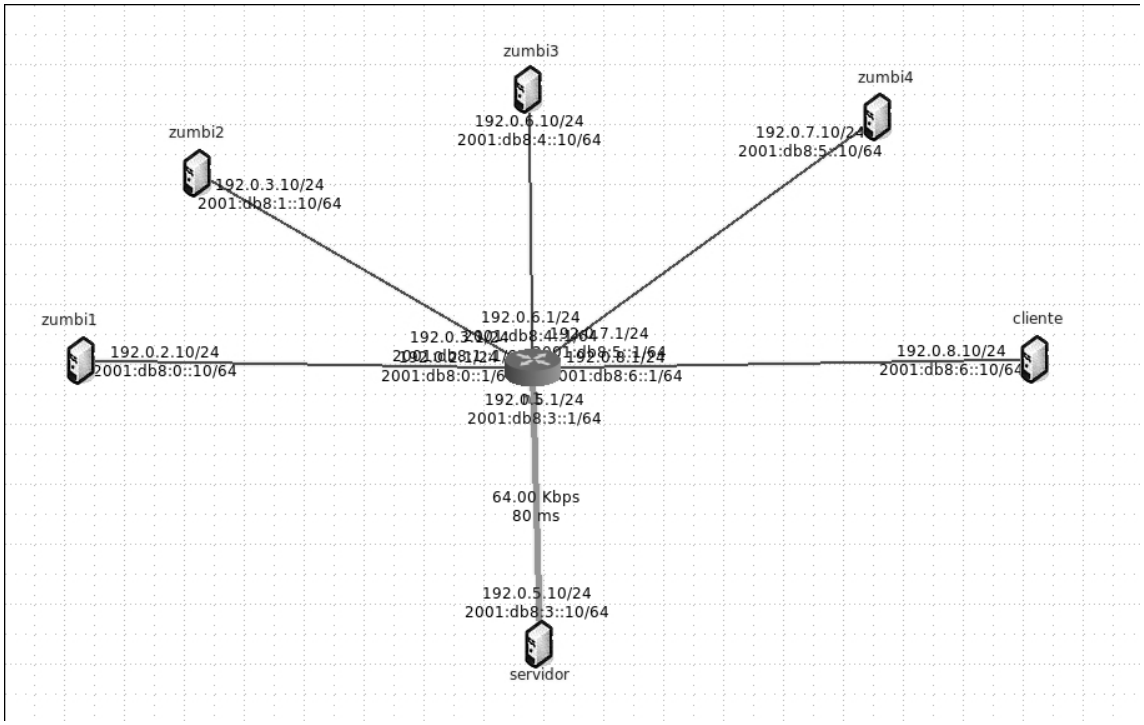


Figura 4. Cenário simulado na ferramenta CORE

4.1. Cenário sem ataques DDoS

Neste cenário apenas o cliente acessa o serviço do servidor, para testarmos se o mesmo está disponível testaremos o acesso através da ferramenta Ping para analisar a conexão fim a fim. O comando 'ping 192.0.5.10' será executado no cliente, onde 192.0.5.10 é o endereço do servidor.

```
root@cliente:/tmp/pycore.51480/cliente.conf# ping 192.0.5.10
PING 192.0.5.10 (192.0.5.10) 56(84) bytes of data.
64 bytes from 192.0.5.10: icmp_req=1 ttl=63 time=320 ms
64 bytes from 192.0.5.10: icmp_req=2 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=3 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=4 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=5 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=6 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=7 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=8 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=9 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=10 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=11 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=12 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=13 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=14 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=15 ttl=63 time=160 ms
64 bytes from 192.0.5.10: icmp_req=16 ttl=63 time=160 ms
^C
--- 192.0.5.10 ping statistics ---
16 packets transmitted, 16 received, 0% packet loss, time 15019ms
rtt min/avg/max/mdev = 160.235/170.298/320.512/38.788 ms
root@cliente:/tmp/pycore.51480/cliente.conf#
```

Figura 5. Resultado do teste Ping para o servidor em ambiente sem ataque

Podemos verificar através dos dados apresentados na imagem, que não houve nenhuma perda de pacotes durante as tentativas de sucesso. Houve 16 envios e 16 respostas com um tempo de resposta contínuo em 160ms. Podemos entender isto como se todo serviço que o cliente requisitou foi respondido com sucesso, em um tempo adequado.

4.2. Cenário com ataques DDoS

Neste cenário o cliente tentará acessar o servidor em meio a um ataque do tipo Ping Flood realizado por 4 máquinas zumbis. O comando executado é 'ping 192.0.5.10 -l 65500'. Onde 192.0.5.10 é o endereço do servidor e o -l 65500 representa o parâmetro preload.

Durante a tentativa de acesso ao serviço cada uma das máquinas zumbis estará enviando um comando Ping ao servidor com o parâmetro preload configurado 65500, ou seja, cada máquina ping enviará 65500 pacotes antes de aguardar pelo retorno das mensagens e assim congestionar a rede do servidor com mensagens echo request.

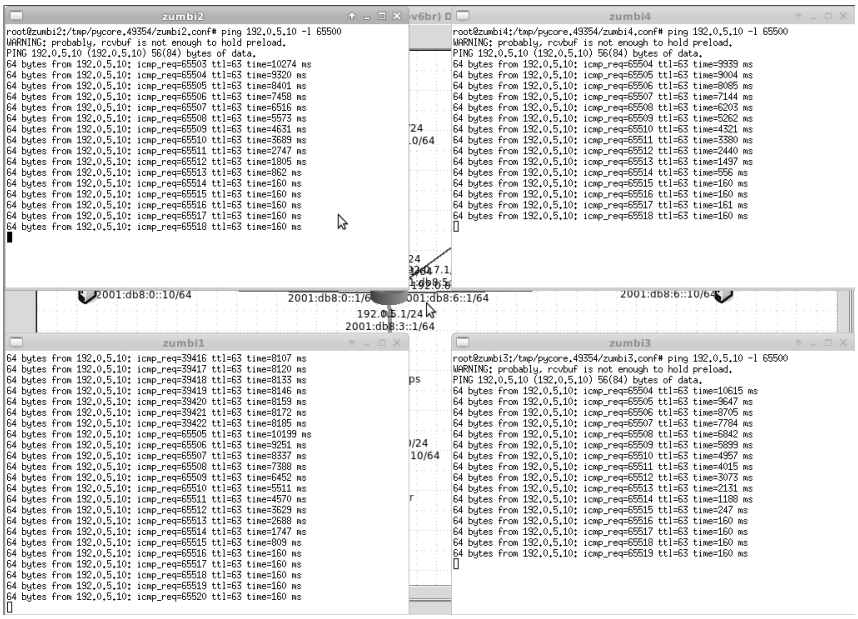


Figura 6. Máquinas zumbis sendo comandadas para atacar

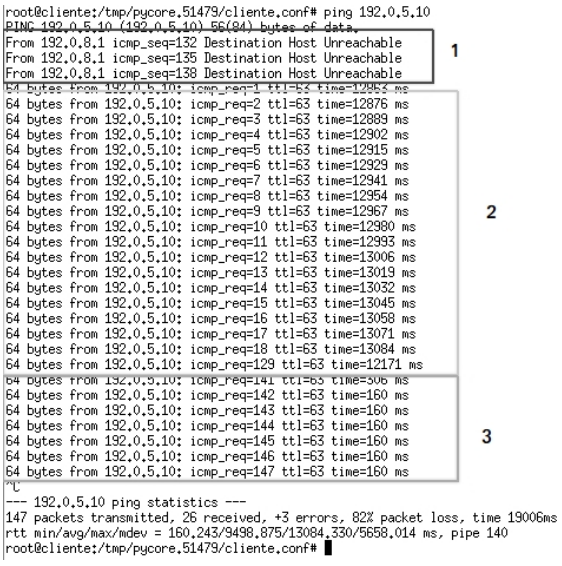


Figura 7. Resultado do teste Ping para o servidor em ambiente sofrendo ataque

Na Figura 7 podemos analisar o resultado do teste da ferramente Ping, no bloco 1 podemos ver que a conexão foi totalmente negada pelo servidor que seria no momento

em que as 4 máquinas zumbis estariam realizando os ataques simultâneamente. No bloco 2 foi realizado a comunicação de forma precária com um tempo grande para resposta do servidor, seria o momento em que algumas das máquinas já teriam termiado o envio dos 65500 pacotes e outras não.

No bloco 3 podemos ver o serviço já normalizado e com um tempo de resposta obtido igual aos obtidos no teste anterior, onde não havia nenhum ataque em andamento. A ferramenta também informa que houve entrega apenas de 26 dos 147 pacotes enviados e 3 erros durante a tentativa de conexão, que seriam representadas pelo bloco 1 da imagem.

5. Análise do ataque no Wireshark

Durante os ataques foi capturado os dados da rede com o uso da ferramenta Wireshark, uma das ferramentas analisadoras de protocolos de rede mais conhecidas. A ferramenta foi iniciado no modo conhecido como promíscuo, que captura todas as mensagens que foram transportadas pela rede a qual ela está conectada.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=1/256, ttl=64 (no response found!)
2	0.000019	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=1/256, ttl=64 (no response found!)
3	0.000027	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=1/256, ttl=63 (no response found!)
4	0.000224	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=2/512, ttl=64 (no response found!)
5	0.000232	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=2/512, ttl=64 (no response found!)
6	0.000236	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=2/512, ttl=63 (no response found!)
7	0.000245	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=3/768, ttl=64 (no response found!)
8	0.000248	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=3/768, ttl=64 (no response found!)
9	0.000250	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=3/768, ttl=63 (no response found!)
10	0.000257	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=4/1024, ttl=64 (no response found!)
11	0.000260	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=4/1024, ttl=64 (no response found!)
12	0.000262	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=4/1024, ttl=63 (no response found!)
13	0.000268	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=5/1280, ttl=64 (no response found!)
14	0.000271	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=5/1280, ttl=64 (no response found!)
15	0.000273	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=5/1280, ttl=63 (no response found!)
16	0.000279	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=6/1536, ttl=64 (no response found!)
17	0.000281	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=6/1536, ttl=64 (no response found!)
18	0.000284	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=6/1536, ttl=63 (no response found!)
19	0.000290	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=7/1792, ttl=64 (no response found!)
20	0.000292	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=7/1792, ttl=64 (no response found!)
21	0.000294	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=7/1792, ttl=63 (no response found!)
22	0.000301	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=8/2048, ttl=64 (no response found!)
23	0.000303	192.0.3.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=8/2048, ttl=64 (no response found!)

Figura 8. Dados da máquina zumbi2 capturados pelo Wireshark

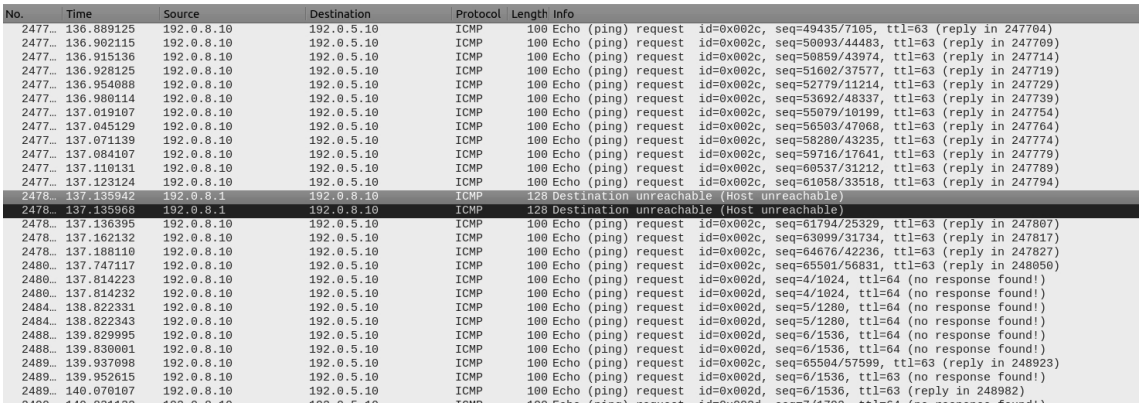
No.	Time	Source	Destination	Protocol	Length	Info
22938	1.602656	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22848/16473, ttl=63 (no response found!)
22939	1.602667	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22849/16729, ttl=64 (no response found!)
22940	1.602669	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22849/16729, ttl=64 (no response found!)
22941	1.602670	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22849/16729, ttl=63 (no response found!)
22942	1.602681	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22850/16985, ttl=64 (no response found!)
22943	1.602683	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22850/16985, ttl=64 (no response found!)
22944	1.602684	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22850/16985, ttl=63 (no response found!)
22945	1.602695	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22851/17241, ttl=64 (no response found!)
22946	1.602697	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22851/17241, ttl=64 (no response found!)
22947	1.602698	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22851/17241, ttl=63 (no response found!)
22948	1.602709	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22852/17497, ttl=64 (no response found!)
22949	1.602711	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22852/17497, ttl=64 (no response found!)
22950	1.602712	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22852/17497, ttl=63 (no response found!)
22951	1.602723	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22853/17753, ttl=64 (no response found!)
22952	1.602725	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22853/17753, ttl=64 (no response found!)
22953	1.602726	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22853/17753, ttl=63 (no response found!)
22954	1.602737	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22854/18009, ttl=64 (no response found!)
22955	1.602739	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22854/18009, ttl=64 (no response found!)
22956	1.602740	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22854/18009, ttl=63 (no response found!)
22957	1.602751	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22855/18265, ttl=64 (no response found!)
22958	1.602753	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22855/18265, ttl=64 (no response found!)
22959	1.602754	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22855/18265, ttl=63 (no response found!)
22960	1.602765	192.0.6.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002a, seq=22856/18521, ttl=64 (no response found!)

Figura 9. Dados da máquina zumbi3 capturados pelo Wireshark

Nas figuras 8 e 9 podemos observar os pacotes ICMP sendo enviados em grandes quantidades e em sequência para o endereço no servidor pelas máquinas zumbis 2 e 3, todas as mensagens sem respostas. Não foi apresentado a ilustração das demais máquinas zumbis pois apresentam o mesmo comportamento.

Na figura abaixo podemos analisar os dados capturados das mensagens do cliente. Veremos que em 2 momentos o Router (192.0.8.1) respondeu para o cliente que o destino

não pode ser encontrado, representando os momentos onde a ferramenta Ping apresenta como erro, como ocorreu no bloco 1 da Figura 7.



No.	Time	Source	Destination	Protocol	Length	Info
2477…	136.889125	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=49435/7105, ttl=63 (reply in 247704)
2477…	136.902115	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=50093/44483, ttl=63 (reply in 247709)
2477…	136.915136	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=50859/43974, ttl=63 (reply in 247714)
2477…	136.928125	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=51602/37577, ttl=63 (reply in 247719)
2477…	136.954088	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=52779/11214, ttl=63 (reply in 247729)
2477…	136.980114	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=53692/48337, ttl=63 (reply in 247739)
2477…	137.019107	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=55079/10199, ttl=63 (reply in 247754)
2477…	137.045129	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=56503/47068, ttl=63 (reply in 247764)
2477…	137.071139	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=58280/43235, ttl=63 (reply in 247774)
2477…	137.084107	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=59716/17641, ttl=63 (reply in 247779)
2477…	137.110131	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=60537/31212, ttl=63 (reply in 247789)
2477…	137.123124	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=61058/33518, ttl=63 (reply in 247794)
2478…	137.135942	192.0.8.1	192.0.8.10	ICMP	128	Destination unreachable (Host unreachable)
2478…	137.135968	192.0.8.1	192.0.8.10	ICMP	128	Destination unreachable (Host unreachable)
2478…	137.136395	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=61794/25329, ttl=63 (reply in 247807)
2478…	137.162132	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=63099/31734, ttl=63 (reply in 247817)
2478…	137.188110	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=64676/42236, ttl=63 (reply in 247827)
2480…	137.747117	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=65501/56831, ttl=63 (reply in 248050)
2480…	137.814223	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002d, seq=4/1024, ttl=64 (no response found!)
2480…	137.814232	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002d, seq=4/1024, ttl=64 (no response found!)
2484…	138.822331	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002d, seq=5/1280, ttl=64 (no response found!)
2484…	138.822343	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002d, seq=5/1280, ttl=64 (no response found!)
2488…	139.829995	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002d, seq=6/1536, ttl=64 (no response found!)
2488…	139.830001	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002d, seq=6/1536, ttl=64 (no response found!)
2489…	139.937098	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002c, seq=65504/57599, ttl=63 (reply in 248923)
2489…	139.952615	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002d, seq=6/1536, ttl=63 (no response found!)
2489…	140.070107	192.0.8.10	192.0.5.10	ICMP	100	Echo (ping) request id=0x002d, seq=6/1536, ttl=63 (reply in 248982)

Figura 10. Dados da máquina cliente capturados pelo Wireshark

6. Considerações finais

Após estudar e usar a ferramenta CORE para a realização do trabalho, verificamos a importância do mesmo para o estudo de assuntos relacionados a área de redes. Sem custo com equipamentos fomos capazes de montar nossa rede virtual da forma que precisávamos e realizar diversos testes de forma rápida e eficiente até obtermos o resultado esperado.

Simulamos um ataque DDoS com poucas máquinas, mas realizado com sucesso através da configuração de capacidade de rede do servidor. Para um próximo trabalho e mais estudo em cima da ferramenta CORE poderíamos criar scripts e no momento da execução da ferramenta automatizar a execução do Ping pelos zumbis, podendo assim criar uma enorme rede para simular um ataque DDoS mais próximo do que acontece com grandes serviços, onde poucas máquinas zumbis não são capazes de congestionar um serviço.

O trabalho realizado foi importante para vermos de forma prática e real como ocorre um ataque DDoS, e também conhecermos outras formas de como este ataque pode ser realizado. O Artigo e o ambiente da ferramenta CORE usado na simulação estão disponível em https://github.com/clarel/AtaqueDDoScomCORE para download.

Referências

Core-dev (2012). Core 4.8 documentation. Disponível em: `http://downloads.pf.itd.nrl.navy.mil/docs/core/core-html/intro.html`. Acesso em 18 maio 2016.

Kumar, A., Sharma, A. K., and Singh, A. (2012). Performance evaluation of centralized multicasting network over icmp ping flood for ddos. *International Journal of Computer Applications (0975–8887) Volume*.

Oliveira, E., Aschoff, R., Lins, B., Feitosa, E., and Sadok, D. (2007). Avaliação de proteção contra ataques de negaçãoo de serviço distribuídos (ddos) utilizando lista de ips confiáveis. *VII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*.

Orozco, A. M., Fernandes, A. P., and Costa, G. H. (2014). Simulação de syn flooding attackk no common open research emulator. *Revista Competência*, 7(1):161–173.