ESTUDO DE CASO:ANÁLISE DE ATAQUE DE NEGAÇÃO DE SERVIÇO SLOWLORIS

Helder C Almeida P[1], Jose M Trindade Jr[2]

[1]Estudante do Curso Superior de Tecnologia em Sistemas para Internet – IFTO. e-mail: <juniorifto2@gmail.com>

Resumo: Conforme a internet e seus serviços vão se tornando cada vez mais sofisticados, as alternativas de conseguir prejudicar esses serviços também evoluem na mesma proporção, ataques de negação de serviço são uma das maiores ameaças da internet, esses ataques geralmente destinados a organizações, têm como objetivo fazer com que usuários sejam impedidos de acessar algum serviço em um servidor *WEB* fazendo com que a organização dona deste serviço sofra um prejuízo enorme. Neste artigo é realizado um estudo de caso em um ambiente virtualizado, simulando um ambiente real com o servidor *Apache* para um tipo específico de ataque classificado como *Slowloris*, este é um ataque de negação de serviço na camada de aplicação utiliza de uma vulnerabilidade no protocolo *HTTP*, simulando um trafego lento e sobrecarregando o servidor com conexões falsas, impedindo acesso ao serviço, para isso, iremos utilizar o software *Slowhttptest* para simular este ataque e verificar se o servidor é comprometido, após isso iremos utilizar um servidor com um módulo de segurança instalado e repetir o ataque para observar os resultados e determinar se o módulo teve sucesso ao impedir o ataque.

Palavras–chave: *ADoS, Apache, Antiloris, DoS, Slowloris, HTTP*

1 INTRODUÇÃO

Uma empresa que fornece serviços *web* que precisa funcionar 24 horas e 7 dias por semana, segurança e disponibilidade são essenciais, no entanto, existem diversos tipos de ameaças que podem prejudicar essa empresa, como os ataques de negação de serviço (*DoS*). Esse tipo de ataque visa deixar o serviço indisponível ou com a qualidade reduzida.

Ataques de negação de serviço são divididos em três tipos: Denial of service(*DoS*), Distributed Denial of service(*DDoS*), e o ultimo denominado de Application Denial of Service(*ADoS*). Os ataques *ADoS* têm como alvo os protocolos da camada de aplicação, como o *HTTP, SMTP, FTP* e etc, esse tipo de ataque tem como objetivo, inviabilizar a aplicação propriamente dita, fazendo com que o ataque não seja necessariamente um ataque volumétrico(NASCIMENTO et al., 2018).

Segundo (NASCIMENTO et al., 2018) Os ataques *ADoS* são difíceis de detectar porque o atacante estabelece uma conexão *TCP* ou *UDP* normal com a máquina alvo, mesmo com um hardware robusto do lado da vítima não significa que ela estará segura, porque o alvo de um ataque *ADoS* é a aplicação em si em vez dos recursos de processamento, rede ou armazenamento.

O objetivo desse artigo é fazer uso de um ataque para inviabilizar o acesso a um serviço a usuários legítimos, documentar como o ataque funciona e possível solução para mitigar ou mesmo refutar o ataque. Este artigo está divido da seguinte forma: 1-Introdução; 2 – protocolo TCP; 3 – protocolo *HTTP*; 4 – Classificação de Ataques; 4 – Ataques de Negação de Serviço; 5 – *Slowloris*; 5 – 6 – Ferramenta *Slowhttptest*; 7 – Realizando um Ataque Simples; 8 – Realizando um Ataque Massivo Resultados; 9 – Conclusão.

2 PROTOCOLO TCP

Transmission Control Protocol(*TCP*) é um protocolo orientado a conexão, ou seja, requer que uma conexão lógica seja estabelecida entre os dois processos antes que os dados sejam trocados, fazendo um uso de um mecanismo chamado *three-way handshake* (cumprimento de três vias), ele basicamente requer que ambos os lados inicializem e mantenham certas informações de status para manter o fluxo de dados entre eles. Na prática, se aceita, uma conexão *TCP* exige que o servidor reserve recursos, feito a priori durante o processo de *handshake*(POSTEL, 1981).

A conexão *TCP* é *full-duplex*, ou seja, tanto o servidor quanto o cliente conseguem enviar e receber dados ao mesmo tempo, e caso um dos lados queira finalizar a conexão, cada *host* deve finalizar a conexão independentemente, enviando um pacote com sinalizador (*flag*) FIN ativada quando terminar de enviar seus dados, dessa forma, quando o protocolo *TCP* recebe um pacote com a *flag* FIN, ela informará a aplicação envolvida na comunicação que o outro *host* encerrou a transmissão de dados.

Outra *flag* muito importante no protocolo *TCP* é o método *push*(PSH), o protocolo *TCP* por sua natureza, espera que a pilha de dados esteja cheia ao enviar o pacote para o destino, com o método *push* é possível contornar esse pré-requisito.

Para garantir a entrega dos dados, sempre que um pacote é enviado, um contador se inicia, e espera-se que uma resposta do lado do cliente seja enviada confirmando a entrega desse, caso a confirmação seja recebida, o contador é interrompido, caso contrário acontece um *timeout*, e o *TCP* retransmite o pacote considerado perdido.

3 PROTOCOLO HTTP

Enquanto o protocolo *Internet Protocol*(*IP)* e *TCP* trabalham com a parte de endereçamento e confiabilidade respectivamente, um protocolo de camada de aplicação, o Hypertext Transfer Protocol(*HTTP*), trabalha com a parte do conteúdo a ser entregue ao destino, ele atua na camada de aplicação do modelo *Open System Interconnection*(*OSI)*, permitindo que seja possível solicitar um conteúdo disponível em um *website* ou enviar dados através de formulários(NIELSEN et al., 1999).

Quando um usuário acessa um site via um browser, ele usa o protocolo *HTTP* para poder requisitar o conteúdo do site, solicitando os arquivos necessários como uma imagem, ou um arquivo html. Para fazer isso o protocolo *HTTP* utiliza comandos para realizar ações específicas, são chamadas de métodos, existem diversos métodos, mas os principais, são GET e POST.

Segundo (NIELSEN et al., 1999), o método GET é usado para fazer uma requisição de um dado recurso específico, como por exemplo solicitar uma página a um servidor *web*.

O protocolo *HTTP* é também, orientado á conexão, para que uma conexão *HTTP* seja estabelecida, primeiramente é necessário estabelecer uma conexão *TCP*.

Após o estabelecimento de uma conexão *TCP*, uma solicitação *HTTP* pode ser realizada via browser a um servidor web, por padrão destinado a porta 80, porém outras portas como a 8000 ou a 8080 podem também ser usadas, esta solicitação enviada, consiste em diretivas de texto e é formada em três partes, a linha de status, o cabeçalho *HTTP*, e o conteúdo da requisição, cada um desses campos possuem informações essenciais para a comunicação.

4 ATAQUES DE NEGAÇÃO DE SERVIÇO

Um ataque de negação de serviço pode ser caracterizado como um ataque projetado para impossibilitar um computador de fornecer serviços. Em um computador ou servidor, existem recursos limitados para atender ás solicitações dos usuários, como memoria, poder de processamento, espaço em disco, e até a disponibilidade da rede, o objetivo desse ataque é sobrecarregar esses recursos, impedindo o servidor de atender a uma solicitação legítima de usuário que tenta acessar serviços da rede(FARIA et al., 2021);(NASCIMENTO et al., 2018).

4.1 Ataques ADoS

Ataques *ADoS(Application layer DoS)* são ataques de negação de serviço que ocorrem na camada "topo" do modelo OSI, ou seja, na camada de aplicação, usando protocolos como por exemplo o *HTTP*. Esse tipo de ataque se aproveita do comportamento dos protocolos para conseguir neutralizar um serviço. Como o ataque usa a própria estrutura ou comportamento de um protocolo legítimo isso torna mais difícil a sua detecção(BEITOLLAHI; DECONINCK, 2012).

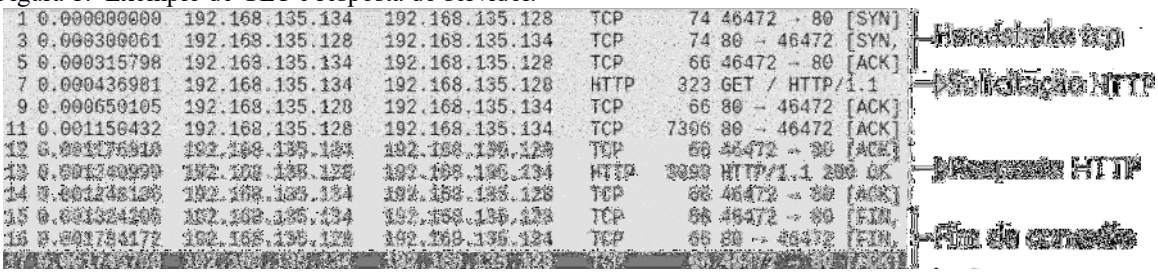
Diferenciar um tráfego malicioso de um tráfego normal, pode ser um problema, pois em um ataque como o *slowloris* o atacante faz solicitações legalmente legítimas para o servidor(BEITOLLAHI; DECONINCK, 2012).

5 SLOWLORIS

O *Slowloris* é um ataque de negação de serviço que tem como objetivo tornar um serviço indisponível se aproveitando do tempo de *timeout* de um servidor *web* através da abertura de várias conexões *HTTP* usando solicitações com dados parciais, forçando o servidor a ficar à espera de mais dados, dessa forma quando um usuário tenta acessar o servidor, ele não consegue, porque todas as conexões disponíveis estão ocupadas pelo atacante.(RSNAKE, 2009)

Em uma solicitação e resposta *HTTP* normais, o cliente envia um *HTTP* GET para solicitar um recurso do servidor, e o servidor responde 200 OK e envia o recurso exigido ao cliente, como na figura 1(HELALAT, 2017).

Figura 1: Exemplo de GET e resposta do servidor.



Fonte: HELALAT, 2017

O servidor *web*, por natureza, precisa que a mensagem seja completamente recebida antes de ser processada, se o cabeçalho *HTTP* GET não estiver completo, o servidor presume que o cliente esteja com uma conexão lenta; portanto o servidor mantém os recursos alocados aguardando o restante do cabeçalho GET. O invasor não enviará o GET completo; apenas envia cabeçalhos falsos lentamente para evitar o tempo de expiração da conexão e manter o servidor ocupado(HELALAT, 2017).

Figura 2: Solicitação legitima *HTTP*.

```
Hypertext Transfer Protocol
> GET /xampp/ HTTP/1.1\r\n
  Host: 192.168.198.128
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0)
  Gecko/20100101 Firefox/45.0\r\n
  Accept: text/html,application/xhtml+xml,application/
  xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  Cache-Control: max-age=0\r\n
  \r\n
```

Fonte: HELALAT, 2017

Um cabeçalho *HTTP* geralmente é composto de múltiplas linhas, o final de uma linha no cabeçalho *HTTP* é definido pelos caracteres \r \n, enquanto \r \n \r \n significam o fim do cabeçalho e o começo do corpo da mensagem. A figura 2 nos mostra o uso do \r \n em uma solicitação *HTTP* legítima(HELALAT, 2017).

Figura 3: Série de pacotes que geram um cabeçalho *HTTP* incompleto

```
GET /xampp/ HTTP/1.1
Host: 192.168.198.128
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/
4.0; SLCC2)                                   First header not closed
Referer: https://github.com/shekyan/slowhttptest/
X-UqlXz8Vw0Ro4Fr57j2LhwK: uScp2Kw9cdIzSELE2aqx   second header not closed
X-zuQqZ: QSpDAS              third header not closed
X-yF9LVkBXobnq: zg        forth header not closed
X-iQb6cxsYxrFbyJC4LLRrp: tbsHaF8ubg82baoHwbx1  Fifth header not closed
97a53454c4532617178 0d0a
582d7a7551715a3a20515370444153 0d0a
582d7946394c566b42586f626e713a207a67 0d0a
582d695162366378735978724662794a43344c4c5272703a20746273486146387562673833
262616f4877627831 0d0a   not a complete combination of "0d0a0d0a"
```

Fonte: HELALAT, 2017

A figura 3 mostra uma série de cabeçalhos *HTTP* que são transmitidos a cada 10 segundos para o servidor para assemelhar a uma solicitação *HTTP* incompleta. É possível detectar esse truque quando é observada a informação em hexadecimal que mostra que os pacotes semelhantes não tem um \r \n \r \n completo(HELALAT, 2017).

O atacante usa esse recurso do cabeçalho *HTTP* para lançar o ataque, iniciando uma solicitação *HTTP*, sem finalizar a solicitação, ou seja, enviando somente o \r \n, isso fará com que o servidor fique esperando pelo resto do pacote e além disso irá criar conexões o suficiente para ocupar todas as filas de conexões, impedindo um usuário legítimo de se conectar ao servidor(HELALAT, 2017).

6 SLOWHTTPTEST

Para fazermos os testes no servidor apache, usaremos uma ferramenta chamada Slow*http*test. É uma ferramenta de simulação para testar a vulnerabilidade dos servidores apache a ataques do tipo ADoS, e implementa o *slowloris*(SHEKYAN, 2013).

Esta ferramenta é interessante pois ela consegue gerar relatórios após o ataque onde é possível ver em qual momento do teste o servidor teve seus serviços interrompidos, além disso é possível determinar algumas variáveis no ataque, como a quantidade de conexões e o tempo do ataque.

Figura 4: Exemplo de linha de comando do *slowhttptest*

```
root@ubuntu:/home/atacante# slowhttptest -c 50000 -H -g -o slowloristest -i 5 -r 200 -t GET -u http://192.168.10
.110 -x24 -p3
```

Fonte: Elaborada pelo autor, 2021

Para utilizar a ferramenta (figura 4), é preciso conhecer os parâmetros que são usados para configurar o ataque, os principais parâmetros são: **-c** mostra o número de conexões que o ataque vai gerar. **-H** informa o tipo do ataque entre os 4 disponíveis, nesse caso o *slowloris*. **-g** é usado para criar estatísticas e gerar a página de relatório. **-o** é usado para salvar as estatísticas geradas pelo -g em um diretório. **-i** determina o intervalo de dados a serem enviados em segundos. **-r** determina o "verbo" a ser usado se for em ataques do tipo "*slowheader*" deixar como padrão *GET* se for do tipo "*slowbody*", deixar como *POST*. **-u** o endereço do servidor a ser atacado. **-p** determina o tempo de *timeout* de uma conexão "sonda" depois que o servidor fica inacessível(SHEKYAN, 2013).

7 REALIZANDO UM ATAQUE SIMPLES

Para realizar os experimentos deste artigo um ambiente, foi configurado com quatro máquinas virtuais, duas dessas máquinas são servidores, a primeira com uma instalação padrão do *apache*, e a segunda, com o *apache* e o módulo de segurança *antiloris* instalado, a terceira máquina virtual fazendo o papel do atacante, e a última máquina fazendo o papel de um cliente legítimo. Os servidores terão 4 gb de memória, e as outras duas máquinas terão 2gb, todas as quatro máquinas estão sendo virtualizadas em um computador com processador i5-9400f com 16gb de memória.

A princípio realizamos um ataque simples ao servidor apache com o intuito de descrever o comportamento do *slowloris* e coletar dados. Nesse ataque foi realizado apenas uma conexão

maliciosa, não sendo o suficiente para interromper o serviço de um servidor *web*. Coletou-se o fluxo de pacotes com a ferramenta *wireshark*, onde poderemos analisar como o *slowloris* se comporta.

De acordo com a figura 5, quando executamos o ataque, a ferramenta abre duas conexões, vamos chamá-las de conexão A e B, a conexão A, é a conexão não maliciosa, solicita uma página para o servidor usando o comando *GET*, logo em seguida o servidor responde ao *GET* da conexão A com o status 200(linha 8) confirmando o envio, depois disso o próprio software envia um *FIN, ACK* e fecha a conexão A. O servidor responde com um *FIN, ACK* também, e a conexão é fechada, tudo isso em menos de um segundo.

Figura 5: Exemplo de linha de comando do *slowhttptest*

Fonte: Elaborada pelo autor,2021

Na conexão B, ocorre o handshake com o servidor e logo em seguida o suposto cliente envia um pacote para o servidor usando a flag *PSH*, a conexão B fica silenciosa e somente após 5 segundos do envio do PSH o suposto cliente envia novamente outro PSH para o servidor, até atingir 20 segundos após o primeiro *PSH*, tempo limite do servidor *web* para processamento do recebimento e um requerimento, ocorrendo dessa forma um *timeout*, desencadeando por parte do servidor o envio de uma sinalização de encerramento da conexão. Na prática o servidor *web* não recebe a totalidade do cabeçalho *HTTP* durante todo tempo da conexão, recebendo pacotes com a terminação \r\n e não com \r\n\r\n no entretanto, durante todo esse período, um recurso (thread) do servidor *web* ficou exclusivamente a disposição desta conexão maliciosa.

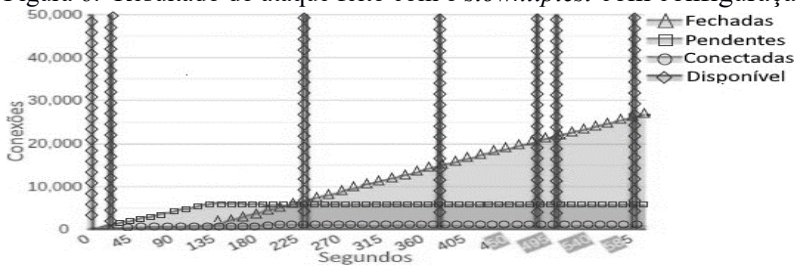
Conforme foi analisado, a conexão A, solicita uma página para o servidor somente para detectar se o servidor ainda é capaz de responder solicitações legítimas. Portanto uma página é solicitada de 3 em 3 segundos sendo respondida pelo servidor e encerrando a conexão. O ataque é verdadeiramente realizado pela conexão B que de 5 em 5 segundos envia uma flag *PSH* para o servidor, somente para manter a conexão aberta o maior tempo possível, causando assim uma negação

de serviço para solicitações legítimas.

8 REALIZANDO UM ATAQUE MASSIVO

O *Apache* foi escolhido para ser o alvo do ataque apenas para reduzir o escopo do experimento, trabalhos futuros irão ser feitos utilizando outros servidores como teste, foram utilizadas 50000 conexões (definidas no parâmetro -c na figura 4) e foi escolhido um tempo máximo de 10 minutos(-t), com o intervalo de envio de dados a cada 5 segundos(-i), o tempo de *timeout* por padrão do servidor apache é de 20 segundos, então cada conexão maliciosa criada pelo ataque só consegue manter o servidor ocupado por 20 segundos, depois disso a conexão sofre *timeout* e é aberta outra conexão no lugar.

Figura 6: Resultado do ataque feito com o *slowhttptest* com configuração padrão.



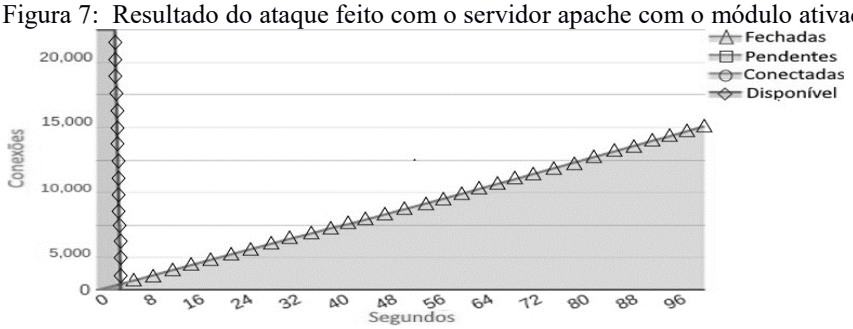
Fonte: Elaborada pelo autor, 2021

Na figura 6 mostra um gráfico gerado pela própria ferramenta, nele é possível ver o impacto que o ataque teve no servidor, existiram somente poucos momentos em que o servidor ficou disponível em intervalos de mais de 2 minutos. Em todo o tempo de ataque o servidor ficou inacessível para acessos legítimos.

8 .1 TÉCNICAS PARA MITIGAR UM ATAQUE SLOWLORIS

Na tentativa de mitigar o ataque, foi utilizado um módulo chamado *antiloris*, com o módulo instalado, observa-se que o gráfico do resultado do ataque muda completamente. Conforme mostrado na figura 7, o gráfico mostra que durante o ataque o servidor ficou disponível somente no início do ataque, mas para entendermos o que aconteceu, precisamos que entender como o módulo se comporta.

Figura 7: Resultado do ataque feito com o servidor apache com o módulo ativado



Fonte: Elaborada pelo autor, 2021

O módulo *antiloris* limita o número de conexões em que o mesmo *ip* pode enviar ao servidor, portanto o módulo bloqueia até mesmo as conexões que a ferramenta envia para verificar se o servidor está ativo, dessa forma o gráfico mostra que o servidor não se encontrava disponível, mas ele só não estava disponível para o *ip* da máquina atacante, no entanto, isso pode se tornar um problema, pois em um ataque vindo de um *ip* externo, o bloqueio desse *ip* irá impactar todos os usuários legítimos que estão por traz desse endereço IP, por exemplo em um cenário de *network address translation*(*NAT)*.

9 CONCLUSÃO

Ataques *slowloris* são difíceis de evitar, mesmo fazendo configurações minuciosas no servidor não foi possível mitigar totalmente o ataque. O apache é configurável com relação a quantidade de atendimento concorrente, mas isso é limitado, por mais que aumente a robustez de um servidor, o *slowloris* com pouco recurso de hardware consegue causar uma grande demanda para o servidor.

O grande trunfo de ataques *slowloris* é a simplicidade do ataque, mas que é capaz de causar grande dor de cabeça aos profissionais de segurança. Em nosso experimento o ataque se mostrou eficiente e a neutralidade do mesmo não se mostrou eficaz.

Em trabalhos futuros pretende-se estudar o comportamento do *slowloris* fazendo uso do protocolo *IPv6*, bem como fazer experimentos em outros servidores *web* e pesquisar novas formas de mitigar um ataque *slowloris*.

REFERÊNCIAS

BEITOLLAHI, Hakem; DECONINCK, Geert. Tackling application-layer DDoS Attacks. **Procedia Computer Science**, *[S. l.]*, v. 10, p. 432–441, 2012. DOI: 10.1016/j.procs.2012.06.056.

FARIA, Vinicius; GONÇALVES, Jéssica; SILVA, Camila; VIEIRA, Gabriele; MASCARENHAS, Dalbert. Ferramenta para Detecção e Contenção de Ataques Slowloris. *[S. l.]*, p. 183–196, 2021. DOI: 10.5753/sbseg.2019.13971.

HELALAT, Seyed Milad. An Investigation of the Impact of the Slow HTTP DOS and DDOS attacks on the Cloud environment. *[S. l.]*, 2017.

NASCIMENTO, Jáder Lincoln; HENRIQUE, Paulo; CONCEIÇÃO, Coelho; RIOS, Vinícius De Miranda. de ataque http do tipo slowloris. *[S. l.]*, p. 75–80, 2018.

POSTEL, Jon. RFC 793 - Transmission Control Protocol. *[S. l.]*, p. 91, 1981. Disponível em: https://rfc-editor.org/rfc/rfc793.txt.

RSNAKE. **Slowloris HTTP DoS**. 2009. Disponível em: https://web.archive.org/web/20150426090206/http://ha.ckers.org/slowloris. Acesso em: 28 nov. 2021.

SHEKYAN. **SlowHTTPTest Package Description**. 2013. Disponível em: https://tools.kali.org/stress-testing/slowhttptest. .