Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Viabilidade de Ataque de Negação de Serviço explorando Perfect Forward Secrecy no SSL/TLS

Marco Antonio Marques Pinheiro

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. MSc. João José Costa Gondim

Brasília
2014

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Engenharia da Computação

Coordenadora: Prof. Ricardo Zelenovsky

Banca examinadora composta por:

Prof. MSc. João José Costa Gondim (Orientador) — CIC/UnB
Prof. Dr. Robson de Oliveira Albuquerque — ENE/UnB
Dr. Dino Macedo Amaral — Banco do Brasil

**CIP — Catalogação Internacional na Publicação**

Pinheiro, Marco Antonio Marques.

Viabilidade de Ataque de Negação de Serviço explorando Perfect Forward Secrecy no SSL/TLS / Marco Antonio Marques Pinheiro. Brasília : UnB, 2014.
44 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2014.

1. DDoS, 2. segurança, 3. ataques, 4. DoS, 5. SSL/TLS, 6. RSA ,
7. diffie & hellman, 8. cifras de segredo perfeito

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Viabilidade de Ataque de Negação de Serviço explorando Perfect Forward Secrecy no SSL/TLS

Marco Antonio Marques Pinheiro

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. MSc. João José Costa Gondim (Orientador)
CIC/UnB

Prof. Dr. Robson de Oliveira Albuquerque
ENE/UnB

Dr. Dino Macedo Amaral
Banco do Brasil

Prof. Ricardo Zelenovsky
Coordenadora do Curso de Engenharia da Computação

Brasília, 24 de dezembro de 2014

# Dedicatória

Experiência é um momento inerente à vida e vice-versa. Não há como presenciar um fenômeno, um aprendizado, um sorriso ou até mesmo um choro se não estiver disposto a viver plenamente. Contudo, assim, não há uma vida digna sem experiências. E, com o tão pouco que vivi, de todas as coisas que aprendi, corri atrás, arranquei dos livros da Biblioteca Central e absorvi dos professores da UnB, dedico este trabalho aos novos cientistas que queiram fazer de suas vidas uma vida cheia de experiências novas pois, viver não é só conhecer mas, também, passar conhecimento.

# Agradecimentos

Agradeço a Deus e aos meus pais e irmãos primeiramente pois todos passos na minha vida foi dado aos olhos e ao lado deles. Agradeço também a todos os professores que me ajudaram desde o início até minha formação universitária, em especial ao Professor Sergiu de Reno, Nevada, EUA, por ter me acolhido no exterior e ter me posto tanta sabedoria e cuidado, mostrando ser mais que um professor, um grande amigo; ao Professor Wilson por quem criei uma grande admiração e me mostrou que ser um grande profissional, e, por fim, ao Professor Orientador Gondim por ter me dado ensinamentos extraordinários e direcionando o meu sonho para realidade.

# Resumo

O uso do Perfect Forward Secrecy vem se tornando uma peça fundamental na segurança da inforação na Internet. O objetivo deste trabalho é analisar o impacto do uso do Perfect Forward Secrecy, mensurando sua eficácia e analisar a viabilidade de um ataque de negação de serviço explorando essa propriedade.

**Palavras-chave:** DDoS, segurança, ataques, DoS, SSL/TLS, RSA , diffie & hellman, cifras de segredo perfeito

# Abstract

The use of the Perfect Forward Secrecy has become an important element on the security of the information on the Internet. The aim of this project is to analyse the impact of the use Perfect Forward Secrecy, measuring its effectiveness and analyse the viability of a denial of service attack exploring this property.

**Keywords:** attack, exploits, DDoS, security, web servers,DoS, SSL/TLS, RSA , diffie & hellman, perfect foward secrecy

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

## 1.1 Rede de computadores e a *Internet*

Informação e comunicação são dois dos maiores problemas estratégicos encontrados para o sucesso de toda empresa. Enquanto a grande maioria utiliza de um substancial número de computadores e ferramentas de comunicação (telefones e celulares), muitas delas operam de forma isolada. Não é raro encontrar departamentos de empresas que não se comunicam ou que muita da informação necessária não seja de fácil acesso.

Para ultrapassar esses obstáculos de forma eficiente com o uso da tecnologia da informação, são necessárias redes de computadores. Elas são um novo tipo de paradigma de sistema de computadores geradas a partir da necessidade de fundir computadores e a comunicação. Somente com a ajuda de redes de computadores que uma comunicação sem fronteiras e um ambiente de informação podem ser construídos.

Redes de computadores permitem o usuário acessar programas remotos e base de dados de uma ou mais empresas. Elas permitem uma comunicação mais rápida que qualquer outro método. A *Internet* é um exemplo claro disso.

A *Internet* é a maior invenção dos últimos anos. Ela interconecta milhões de computadores, provendo uma comunicação global, armazenamento e infraestrutura de computação. Ela é muito famosa hoje em dia por satisfazer as pessoas com os mais variados serviços relacionados a diferentes campos. Quase tudo está disponível na *internet* nessa era de tecnologias avançadas. Nunca fomos bombardeados com tanta informação e ela nunca foi tão fácil de ser adquirida; a *internet* passou a ser a maior fonte de conhecimento e pesquisa das pessoas. Além disso, podemos, dentre milhares de coisas, pagar contas e comprar quase de tudo indo de *site* em *site* e escolhendo dentre as variedades de opções. Nada mais radicalizou tanto a cultura, a mídia, o comércio, o entretenimentoo e a comunicação como a *Internet*.

## 1.2 Objetivo e Motivação

No primeiro semestre de 2013, um ex-funcionário da *CIA*, Edward Snowden, revelou ao mundo programas de vigilância utilizados pelo governo norte-americano tanto para espionar sua própria população como outros países, incluindo personangens como presidentes europeus e latino-americanos. Essas revelações levaram ao debate sobre a segurança e privacidade das informações na rede e à sugestão do uso de uma propriedade no protocolo *HTTPS* para a maior proteção nas conexões seguras.

Essa propriedade, chamada de *Perfect Forward Secrecy*, ás vezes referenciada neste documento apenas pela sigla *PFS*, vem ganhando bastante destaque depois do escândalo envoltando a *NSA*. Sites que usam o *PFS* podem fornecer uma melhor segurança para os usuários em casos onde os dados cifrados estão sendo monitorados e gravados por alguma terceira parte. Isso acontece porque, sem o uso do *PFS*, qualquer pessoa que obtiver a chave secreta em qualquer momento no futuro pode usá-la para decifrar toda a informação armazenada anteriormente. [8]

O *Foward Secrecy* consiste em usar uma chave diferente para cada conexão, para isso usam-se *cipher suites* que usem do protocolo *Diffie & Hellman* efêmero, tanto na versão normal quanto na versão com curvas elípticas. Com uma chave diferente para cada conexão, o atacante teria que quebrar as chaves de cada conexão, o que seria computacionalmente inviável.

Mas o *Perfect Foward Secrecy* também envolve um maior custo computacional já que chaves devem ser geradas e negociadas a cada conexão, logo esse custo computacional extra também pode ser usado a favor de um ataque, como será mostrado nesse documento. Nos testes das ferramentas de ataque pretende-se testar o uso do *Perfect Foward Secrecy* para mostrar que seu uso não é a solução perfeita pois ela traz algumas desvantagens.

Este trabalho avalia o impacto do uso de *PFS*, comparando seu uso com outras formas de proteção, como *RSA* e avalia a viabilidade de ataques de negação de serviço que venham a explorar o possível impacto de processamento que o *PFS* acarreta. Mais especificamente, se avaliam os impactos, e por meio de uma ferramenta de prova de conceito se implementa um possível ataque de negação de serviço. Para isso, vários testes são realizados para avaliar a possibilidade de ataque e sua possível dinâmica. Pretende-se, com isso, ter um maior entendimento do problema e um ponto de partida para prevenir ou solucioná-lo.

## 1.3 Estrutura do Trabalho

Este primeiro capítulo fez uma breve introdução sobre o tema deste trabalho, abordando a importância da computação hoje em dia e seguindo para a rede de computadores. Foi

abordado também algumas questões sobre segurança que motivaram a realização deste documento.

O segundo capítulo faz uma revisão teórica, necessária para a compreensão dos conceitos básicos. Nele será abordado a definição, exemplos e a classificação dos ataques de negação de serviço.

O terceiro capítulo é dedicado ao protocolo *TLS/SSL*, com uma breve explicação sobre a importâcia e o funcionamento dele.

O quarto capítulo trata de detalhes da implementação e da preparação dos testes, assim como as ferramentas utilizadas durante sua realização. Será abordado também toda a descrição dos parâmetros escolhidos e da metodologia utilizada. Além disso, para cada testes será descrito os objetivos e resultados esperados.

O quinto capítulo mostra as tabelas e gráficos dos resultados dos testes realizados. Também será feita uma análise dos resultados, comparando as diferenças de tempo para as diferentes condições de ataque.

O sexto e último capitulo traz conclusões sobre este trabalho, possíveis medidas para previnir esses ataques e propostas para trabalhos futuros.

# Capítulo 2

# Ataques de negação de serviço

À medida que o uso da internet está crescendo assustadoramente, manter os sites e seus serviços disponíveis se tornou uma tarefa bem crítica. Bloquear a disponibilidade de um serviço da internet pode implicar em grandes perdas financeiras, no caso de ataques a sites de comércio como a Amazon, por exemplo. Esses ataques que tem como objetivo de tornar sites e serviços indisponíveis são chamados de Ataques de Negação de Serviço (DoS).

Como o objetivo inicial da *internet* era proporcionar uma rede aberta e escalável entre entidades educacionais, problemas de segurança eram as menores preocupações. Contudo, com o crescimento rápido da *internet*, o número de ataques nela também cresceu rapidamente. Então, é crucial deter ou minimizar os danos causados pelos ataques de negação de serviço.

Outra variação desses ataques é o chamado ataque de negação de serviço distribuídos (DDoS). Esse tipo de ataque requer a cooperação de vários sistemas finais, com um *botnet*[1], por exemplo. Os DDoS são mais agressivos, pois o tráfego acumulado de todas máquinas participantes no ataque pode ser muito maior que a capacidade de recursos da vítima. Esses ataques são, geralmente, mais difíceis de combater, pois é necessário identificar as máquinas atacantes e, assim, diferenciar tráfego legítimo de tráfego maldoso.

Esses ataques são motivados por várias razões, como ganhos econômicos e financeiros. Há tambem aqueles que praticam os ataques por vingança, em resposta a alguma injustiça sofrida. Os motivos variam muito, de pessoas que praticam ataques puramente por desafio intelectual; geralmente jovens *hackers* entusiasmados que querem mostrar suas habilidades aos outros até organizações terroristas bem treinadas com o intuito de atacar grandes setores do outros países.[12]

[1]Conjunto de computadores conectados à *Internet* usados para espalhar vírus ou realizar um ataque

## 2.1 Exemplos de ataques

### 2.1.1 TCP SYN Flooding

Quando um cliente quer iniciar uma conexão *TCP* com um servidor, requisitar uma pagina web, por exemplo, ele primeiramente envia uma mensagem *SYN* para este servidor. O servidor, por sua vez, responde a esta mensagem com outra do tipo *SYN-ACK*. O cliente, então, completa o handshake respondendo uma mensagem *ACK*. A partir daí a conexão entre o cliente e o servidor está aberta e dados podem ser trocados entre eles. O ataque surge no estado em que o servidor está esperando a última mensagem *ACK* do cliente para finalizar o *handshake*, depois de ter enviado o *SYN-ACK*. Neste estado, o servidor alocou memória para armazenar informações desta conexão quase-aberta. Essa memória não será liberada até o servidor receber o *ACK* do cliente ou até a conexão expirar. Realizando esse processo inúmeras vezes em pouco tempo, o servidor não terá como estabelecer novas conexões, e assim, não conseguirá prover seus serviços.

### 2.1.2 *UDP Flooding*

Neste ataque, vários pacotes *UDP* são enviados para a vítima. Já que pacotes *UDP* são mais simples que os pacotes *TCP*, esse tipo de ataque visa exaurir a largura de banda da vítima enviando centenas de pacotes *UDP* por segundo, fazendo, assim, o servidor não ter largura de banda para iniciar novas conexões com usuários legítimos.

### 2.1.3 *RUDY*

Mais conhecido como *R-U-Dead-Yet*, o *RUDY* é um ataque difícil de ser detectado, pois gera trafego a uma taxa baixa e de volume pequeno. Esse ataque explora vulnerabilidade no protocolo *HTTP* quando o browser usa o metodo *POST* para requisitar mensagens. Qualquer *website* que contenha algum tipo de formulário esta sucetível ao *RUDY*. Nesse tipo de ataque, são enviados apenas 1 *byte* de dados por pacote em intervalos aleatórios de tempo, sem deixar que ocorra *timeout* na conexão, fazendo com que o servidor não consiga mais receber novas conexões por conta das centenas de conexões abertas.

### 2.1.4 *SlowLoris*

Esse ataque, assim como o *RUDY*, eh um ataque difícil de ser detectado por sistemas padrões anti-DoS, pois gera tráfego a uma taxa baixa e de pequeno volume. Ele também explora vulnerabilidades no protocolo *HTTP*, fazendo com que toda requisição *HTTP* seja enviada sem a sequência de fim de linha, fazendo o servidor deixar a conexão aberta

e alocar recursos que estão esperando pela sequência de término. Um típico servidor web aloca recursos limitados para lidar com conexões abertas, pois essas conexões são, geralmente, pequenas e terminam rapidamente. *SlowLoris* explora esse comportamento e gera milhares de requisições por minuto, onde nenhuma dessas requisições são finalizadas. Isso consome os recursos disponíveis para conexões abertas, impedindo usuários legítimos de fazerem novas requisições.

## 2.2 Desafios de Defesa

Apesar do enorme esforço de pesquisadores e experts no assunto, ataques de negação de servico ainda é um problema não resolvido. Existem vários desafios técnicos e não-técnicos que precisam ser bem entendidos para projetar soluções que resolvam esse problema.[6]

A arquitetura da *internet* é baseada em vários princípios, como o roteamento e o repasse de pacotes e distribuição da gerencia de recursos. Enquanto o projeto desses princípios levaram a uma *internet* robusta, escalável e uma boa relação custo-eficácia que suporta diferentes tipos de redes e protocolos, isso também levou um ambiente desafiador para prevenção de partes maliciosas que buscam causar dano aos outros. Apesar do objetivo inicial da *internet* ser uma rede robusta na presença de ameaças externas, não houve uma preocupação aquivalente a respeito da possibilidade de ataque por usuários da própria *internet*. A visão dos projetistas da internet era que a comunidade de usuários fosse confiável, então medidas simples de segurança seriam suficientes para assegurar a proteção dela. [6]

Em adição aos desafios de arquitetura da *internet*, existem muitos outros desafios que tornam os ataques *DoS* difíceis de serem defendidos. É difícil distinguir requisições maliciosas de requisições legétimas. Mesmo que alguns comportamentos maliciosos forem encontrados por mecanismos de detecção, atacantes geralmente modificam características desses ataques para se esquivarem dessa detecção.[6]

## 2.3 Taxonomia dos ataques

Existem várias taxonomias dos ataques *DoS*, mas serão citadas apenas as principais.

### 2.3.1 Fraqueza Explorada

Os ataques de negação de serviço exploram diferentes fraquezas para derrubar suas vítimas. Eles podem ser classificados em duas classes: ataques semânticos e ataques de força bruta (também chamados de ataques de vulnerabilidade e de inundação, respectivamente).

#### Ataques semanticos

Ataques semânticos exploram alguma característica específica, algum protocolo ou *bug* em alguma aplicação instalada no computador da vítima para, então, consumir seus recursos. O ataque *Ping-of-Death* e o *TCP SYN* mencionados acima são exemplos de ataques de vulnerabilidade.[9]

#### Ataques de forca bruta

Por outro lado, os ataques de forca bruta miram negar o serviço de usuários legitimos invocando um grande volume de requisições aparentemente válidas e, assim, tentam exaustar algum recurso da vítima. Por exemplo, o *UDP Flooding* se utiliza desta técnica para atingir seu objetivo.[6] É interessante notar que ataques de força bruta necessitam gerar um tráfego substancialmente maior que ataques semânticos para causar dano a vítima.[9]

### 2.3.2 *Spoofing*

*IP spoofing* desempenha um papel importante em ataques de negação distribuídos. Técnicas de *spoofing* definem como o atacante escolhe o endereço de origem a ser mascarado em seu ataque.

#### Spoofing aleatorio

O atacante pode mascarar o endereço de origem simplesmente gerando números de 32 *bits* aleatoriamente para atribuir ao *IP*.

#### *Spoofing* na subrede

Nesta técnica, o atacante mascara o endereço de origem gerando aleatoriamente um endereço válido da subsede, ou seja, com o mesmo prefixo da subrede.

#### *Spoofing* fixo

Diferente das outras duas técnicas, o endereço mascarado é o endereço da vítima. Por exemplo, um atacante pode realizar um ataque de reflexão, daí todo pacote enviado será refletido para a vítima.

### 2.3.3 Alvo

Apesar da maioria dos ataques de negação de serviço focarem na exaustão de recursos da vítima, o alvo de fato da negação de serviço pode variar. Este alvo pode ser a aplicação, a rede ou a infraestrutura

#### Ataques de rede

Esse tipo de ataque desabilita o acesso a vítima sobrecarregando seus mecanismos de comunicação, consumindo toda a largura de banda. Um exemplo desse ataque é o *UDP flooding*. Todos os pacotes contém o endereço de destino do alvo sem se preocupar muito com seu conteúdo.

Nessa categoria também estão os ataques de reflexão e amplificação. Nos ataques de reflexão, atacantes geralmente enviam requisições forjadas ao invés de requisições diretas para os refletores, então esses refletores enviam suas respostas para a vítima e exaustam seus recursos. Nos ataques de amplificação, atacantes exploram serviços para gerar grandes mensagens ou múltiplas mensagens para cada mensagem que eles recebem para amplificar o tráfego em direção à vítima.

#### Ataque de aplicação

Um ataque de aplicação tem como alvo geralmente o servidor da vítima, impedindo usuários legítimos de usarem o serviço e comprometendo o recursos da máquina.[10] Se a vítima, contudo, tentar separar seus recursos para diferentes aplicações, outras aplicações e serviços da vítima podem ainda ser acessíveis para usuários. Por exemplo, um ataque na assinatura no servidor de autenticação pode enviar um grande volume de requisições de autenticação para exaustar os recursos da vítima de verificação de assinatura e derrubar o serviço todo. No entanto, se a vítima puder separar os recursos para as aplicações, ela ainda poderá ter recursos para outras aplicações que não necessitam de autenticação.[10]

#### Infraestrutura

Ataques de infraestrutura mira algum serviço distribuído que é crucial para o funcionamento global da *internet*. Por exemplo, ataques em servidores *DNS*, ataques a núcleos de roteadores, servidores de autenticação. Esse tipo de ataque só pode ser contido com a ação coordenada múltipla de vários participantes.[9]

## 2.4 Estratégias de Defesa

As estratégias dos variados mecanismos de defesa aos ataques *DoS* podem ser divididas em quatro categorias: prevenção, detecção, resposta e tolerância. O mecanismo de prevenção tenta eliminar a possibilidade dos ataques *DoS* ou prevenir os ataques de causarem dano significativo. A detecção monitora e analisa eventos em um sistema para descobrir tentativas maliciosas que visam causar a negação de serviço. Mecanismos de resposta são geralmente iniciados depois de ocorrer a detecção de um ataque e visa eliminar ou minimizar o impacto na vitima. Tolerância mira minimizar o dano causado por um ataque DoS sem ter a capacidade de diferenciar ações maliciosas de ações legítimas.[12]

### 2.4.1 Prevencao e Detecção

Prevenção da negação de serviço visa parar o ataque antes dele de fato causar dano. Muitos atacantes dependem do mascaramento do *IP* para esconder a origem do ataque. Mecanismos de filtragem são projetados para proibir ataques DoS com endereços de origem mascarados de chegarem ao alvo, rejeitando pacotes com *IP*s falsos. Ha também mecanismos de filtragem na rede que só permitem pacotes de saírem da rede se o endereço de origem estiver dentro do intervalo de *IP* esperado. [12]

Detecção é um importante passo antes de tomar providências contra ataques *DoS*. Simplesmente descobrir que um ataque está acontecendo é geralmente fácil, uma vez que a degradação da *performance* do alvo é facilmente notável. Por outro lado, identificar ações maliciosas é uma tarefa difícil. [12]

A maioria das técnicas de detecção focam em identificar o tráfego malicioso na rede e elas são geralmente implantadas perto do alvo. Contudo, diferenciar tráfego malicioso de tráfego legítimo na vítima pode não ser muito eficiente. Isso porque existem ataques maliciosos que consomem a largura de banda dos vários roteadores ao longo do tráfego até perto da vítima. Além disso, se o volume de ataque é grande o suficiente, filtrar o tráfego perto do alvo não ajuda a aliviar o congestionamento dos *links* perto dele. Devido aos roteadores serem elementos sem estado, não é possível adquirir informação a respeito da fonte do tráfego de maneira fácil. Técnicas de identificação da origem do ataque, como *traceback*, em que informações dos pacotes são usadas para reconstruir o caminho viajado, são propostas para contornar esse problema. [12]

### 2.4.2 Resposta

Técnicas de detecção de ataque e identificação da origem visam isolar o tráfego malicioso em tempo hábil, para que então ações futuras possam ser iniciadas para parar ou minimi-

zar o ataque. Os mecanismos de filtragem e limitador de taxa usam a caracterização do tráfego malicioso que é fornecido pelos mecanismos de detecção para filtrar ou limitar a taxa do fluxo do ataque. O limitador de taxa é geralmente usado em casos onde a detecção tem muitos falsos positivos ou não pode ser precisamente caracterizada. Se parte ou todo o fluxo de pacotes maliciosos podem ser precisamente distinguidos por mecanismos de detecção, então eles podem ser filtrados completamente. [12]

### 2.4.3 Tolerância

Mecanismos de tolerância miram minimizar o dano causado por ataques DoS sem conseguirem diferenciar comportamento malicioso de comportamento legitimo. A vantagem óbvia dos mecanismos de tolerância é que eles não dependem de mecanismos de detecção para identificar o ataque, e em alguns casos eles não sabem nem que o ataque está acontecendo. Isso ajuda muito onde a detecção de um ataque e a separação de trafego malicioso é difícil, ou quando a eficiência da detecção é baixa. Mecanismos de tolerância focam em minimizar o impacto do ataque e maximizar a qualidade do serviço provido durante o ataque. [12]

Construir tolerância a falhas no sistema é uma das principais abordagens para alcançar uma boa disponibilidade do serviço. Mecanismos de tolerância a falha podem ser implementados por *hardware* e *software*. Geralmente são alcançados replicando os serviços ou multiplicando os recursos usados pelos serviços. [12]

# Capítulo 3

# TLS/SSL

Algumas informações confidenciais não podem trafegar livremente pela *internet*, pois existe o risco dela ser interceptada ou de alguem se passar pelo destinatário ou da mensagem ser alterada. Para resolver esses problemas e permitir um fluxo de informação privada e autêntica, usa-se o protocolo TLS. [7]

O protocolo *TSL* eh o sucessor do *SSL* e serve justamente para garantir a integridade, sigilo e autenticidade de pacotes enviados pela *internet*. Este protocolo não está definido como uma camada da pilha *TCP/IP*, mas age como uma camada abstrata entre a camada de aplicação e a camada de transporte.

## 3.1 Criptografia simétrica e assimétrica

O conhecimento prévio de criptografia é fundamental para entender como funciona o protocolo *TLS/SSL*, pois seu uso está presente em todos os processos do transporte seguro de uma mensagem.

Criptografia assimétrica, como o *RSA*, utiliza duas chaves separadas, uma para cifrar e outra para decrifrar. Com esse tipo de criptografia, a chave usada para cifrar não pode ser usada para decrifrar. A única maneira de decifrar uma mensagem, do ponto de vista matemático, é usando a segunda chave do par. [11]

Uma organização que queira usar criptografia assimétrica pede um par de chaves a uma unidade certificadora, essa unidade irá guardar a chave pública gerada e a organização irá guardar a chave privada gerada. Toda vez que um usuário quiser se comunicar com essa organização, a unidade certificadora irá confirmar tal chave é, de fato, a chave pública da organização.

Na criptografia simétrica é usada a mesma chave para cifrar e decifrar mensagens. A criptografia simétrica é muito mais rápida computacionalmente, por isso elas são preferidas na hora de cifrar e decrifar grande quantidade de dados. Contudo, o problema desse tipo de

criptografia é que é difícil transportar seguramente a chave pelo meio a fim de estabelecer uma conexão segura.[11]

## 3.2 Assinatura digital

Uma das funções do *TLS/SSL* é garantir a autênticidade das partes envolvidas na comunicação. Esse processo é feito exigindo um certificado do servidor. Esse certificado é simplesmente um arquivo contendo informações da organização e sua chave pública.

Se alguém deseja se conectar seguramente ao servidor, ele enviará seu certificado ao cliente que, por sua vez, irá conferir junto à unidade certificadora se aquela chave pública enviada pelo servidor é, de fato, a chave pública verdadeira dele. Como somente quem possui a chave privada poderá decifrar o código, se o cliente conseguir se comunicar com o servidor, quer dizer que o servidor é quem ele diz ser.

## 3.3 *Cipher suite*

O protocolo *TLS/SSL* suporta um grande número de *cipher suites*. Uma *cipher suite* é uma coleção de algoritmos de criptografia simétrico e assimétrico usado pelos servidores para estabelecer uma comunicação segura. As *cipher suites* suportadas podem ser classificadas baseadas na força do algoritmo de criptografia, tamanho da chave, método de troca de chave e mecanismos de autenticação. Algumas *cipher suites* oferecem um melhor nível de segurança que outras, como é o caso das *cipher suites* que suportam o *PFS*. Existem mais de 200 *cipher suites* conhecidas. [4]

# Capítulo 4

# Implementação e Testes

Este capítulo aborda a preparação dos ataques e as ferramentas utilizadas. Será abordado também toda a descrição dos parâmetros de testes e da metodologia utilizada. Além disso, para cada testes será descrito os objetivos e resultados esperados.

## 4.1 Programa

Para a implementação do programa utilizado nos testes, foi escolhido a linguagem C devido a sua rapidez, deixando, assim, a sua execução rápida e eficiente. O programa utiliza basicamente sockets para realizar as conexões *TCP* e a biblioteca *Openssl*[2] para realizar as etapas do *handshake TLS/SSL* . A função *fork()* foi utilizada para incrementar o número de threads de acordo com as necessidades dos testes. Nenhuma outra *API* foi utilizada no programa a fim de não influenciar nas medições realizadas. O programa funciona de duas formas variáveis, de acordo com os parâmetros passados: o usuário pode escolher fazer o ataque utilizando ou não *Perfect Forward Secrecy* e também pode escolher o número de threads que será usadas no ataque (em exponenciais de 2, até 512 *threads*). Antes de prosseguir com o documento, vale ressaltar que este programa e trabalho tem propósitos apenas acadêmicos, sendo jamais utilizado indevidamente para fins maliciosos.

Primeiramente, as bibliotecas *SSL* são iniciadas e a *cipher suite* é setada de acordo com o parâmetro escolhido, ou seja, se o programa fará o ataque utilizando o *PFS* ou não. Depois das configurações iniciais, mais *threads* são criadas de acordo também com os parâmetros escolhidos pelo usuário. Então, inicia-se um *loop* infinito composto por basicamente três passos: conectar, fazer requisição e desconectar. O propósito deste tipo de ataque de negação é de exaustar o servidor com vários pedidos de conexão justamente para explorar as diferenças de tempo de resposta dele para diferentes *cipher suites* e também analisar o impacto do ataque de acordo com o número de threads utilizadas.

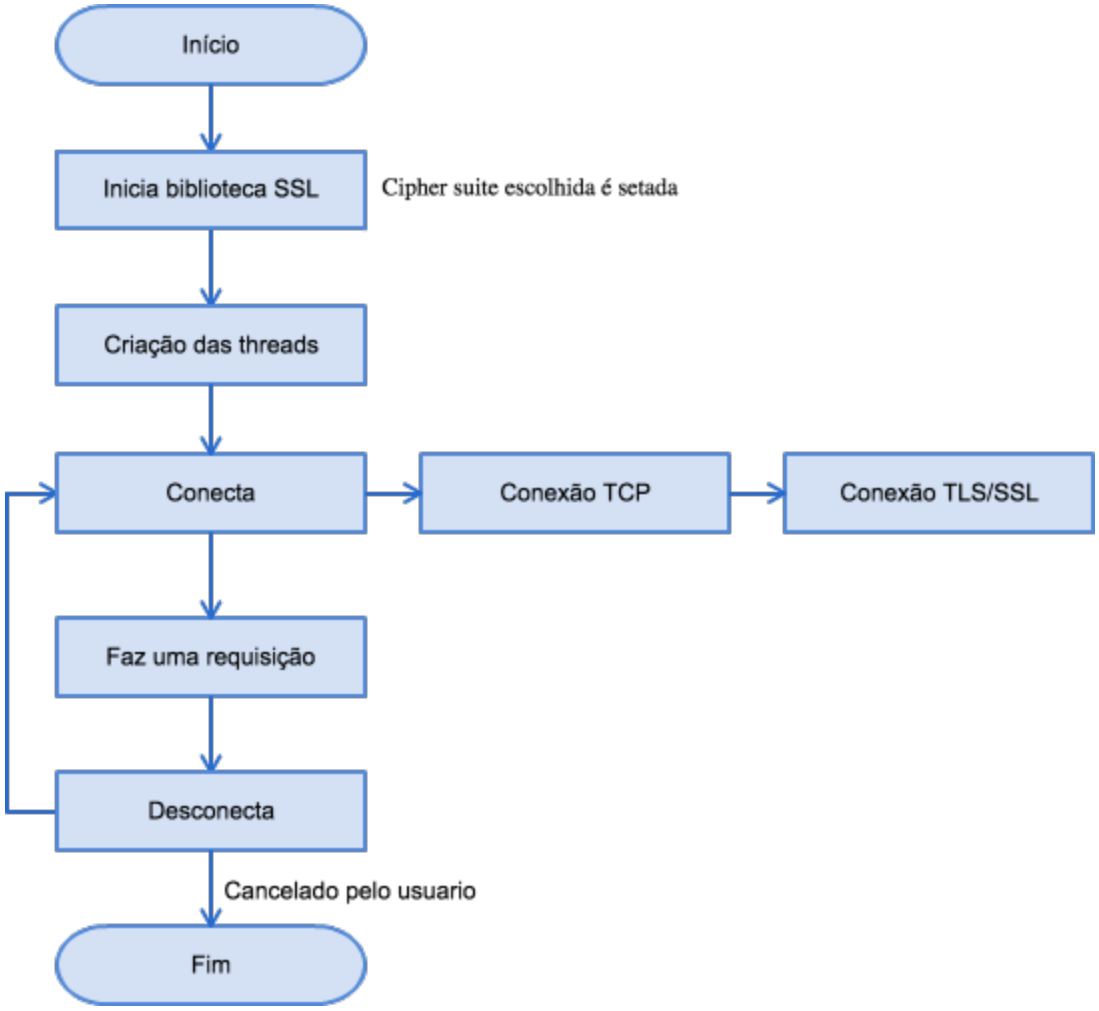


Figura 4.1: Fluxograma do programa utilizado no ataque..

## 4.2 Cliente e servidor

Existem 3 entidades envolvidas nesses testes: servidor, cliente e atacante. O servidor é o principal alvo desse ataque, o qual sofrerá negação parcial ou completa. O cliente é a simulação do usuário legítimo, que pode ser considerado uma vítima secundária do ataque, pois seu acesso ao servidor se encontrará comprometido. O atacante é o usuário que se passará por cliente fazendo o uso do programa de ataque.

Para fazer o monitoramento do servidor, foi utilizado uma própria ferramenta do Apache[1] chamada *server-status*. Com o uso desta ferramenta foi medido o uso da *CPU*, o número total de acessos, o tráfego total, o número de requisições por segundo e o número de requisições que estavam sendo processadas no momento da medição. Foi setado, também, nas configurações do servidor, a liberdade do cliente escolher a *cipher suite* desejada, fazendo, deste modo, que o atacante pudesse forçar as condições de ataque

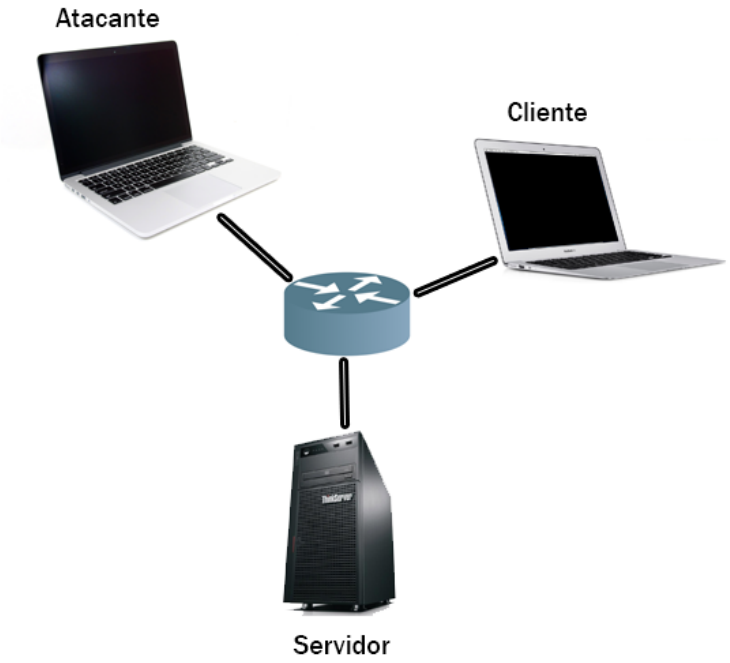


Figura 4.2: Estrutura dos testes.

desejadas[1].

A rede utilizada nos testes era uma rede *wireless* isolada, sem quaisquer outros pacotes percorrendo por ela.

Uma das medições realizadas foi feita pelo cliente, onde se media o tempo de resposta de uma requisição para diferentes variações de ataques. Essa simulação foi feita utilizando o comando

```
watch -n 10 time wget - -no-check-certificate https://IP_DA_VITIMA:443
```

com 30 amostras totalizando 5 minutos cada medição. Esse comando mede, de 10 em 10 segundos, o tempo de download de uma página *web*. Essas medições foram feitas para obter informação do impacto do ataque para diferentes números de *threads* e comparar a diferença de tempo quando o ataque era realizado com o *PFS* e quando era realizado sem essa propriedade. Cada vez que um teste era realizado, o servidor era reiniciado para garantir que os testes fossem feitos sob a mesma condição. Segue na tabela 5.1 a configuração das máquinas utilizadas nos testes.

[1]Neste programa só haviam 2 condições possíveis: com *PFS* ou sem *PFS*.

Tabela 4.1: Tabela com as configurações das máquinas utilizadas no ataque.

| **Servidor** | **Cliente** | **Atacante** |
|---|---|---|
| Desktop Ubuntu | MacBook Air i7 | MacBook Pro |
| Intel Core2 Duo 2.93GHz | 1.3 GHz Intel Core i5 | 2.3 GHz Intel Core i7 |
| 4GB Ram | 4 GB | 8GB Ram |

## 4.3 Objetivos e Metodologia Utilizada

Os testes realizados neste trabalho podem ser separados em 4 principais grupos: testes que mediam o tempo de resposta do servidor, testes que mediam o tempo de *handshake*, testes que avaliavam o impacto no servidor em relação ao número de *threads* e testes que avaliam o impacto no servidor em relação ao tempo decorrido do ataque.

O objetivo do primeiro conjunto de testes é medir o tempo de resposta do servidor dependendo do número de *threads* e da *cipher suite* utilizada e analisar as diferenças de tempo medidas. Espera-se observar um tempo de resposta do servidor cada vez maior à medida que cresce o número de threads utilizadas no ataque. Espera-se, também, observar um tempo de resposta maior quando o ataque utiliza o *PFS*, pois há um processamento adicional do servidor na etapa do *handshake*, o que acarreta numa maior demora neste processo. Nos testes que mediam o tempo de resposta do servidor, foram necessárias a máquinas do servidor, do atacante e do cliente, cuja simulação era dada pelo comando mencionado na sessão anterior. O teste foi dado pela seguinte forma: primeiro eram setados os parâmetros do ataque e então o programa era executado pelo atacante. Logo em seguida o comando de medição de tempo do cliente era iniciado e os resultados eram sendo salvos em um arquivo no próprio computador do cliente. Ao fim de 30 amostras, o comando era finalizado assim como o programa atacante e o servidor reiniciado. Esse processo era então refeito utilizando parâmetros diferentes na hora de rodar o programa atacante.

No segundo conjunto de testes foi medido o tempo de *handshake*. O objetivo desse conjunto de testes era de comparar e analisar o tempo de um handshake *TLS/SSL* utilizando a propriedade do *Perfect Forward Secrecy* e sem ela. Foram inseridas pequenas modificações no programa a fim de medir o tempo gasto no *handshake* quando o programa utiliza diferentes *cipher suites*. Foram adicionados contadores de ciclos de *clock* entre a função provida pelo *OpenSSL*[2] que realiza o *handshake* a fim de medir o tempo que o programa levava para realizar este processo. Foram tiradas 6 amostras de ciclos de *clock* para quando o *PFS* estava habilitado e outras 6 quando esta estava desabilitado. Dessas 6 amostras foi tirado a média[2] para determinar o tempo que o programa levava para realizar

[2]Foram feitas 3 amostras de 100 ciclos e 3 amostras de 1000 ciclos, sendo o resultado das últimas 3 amostras dividido por 10.

o processo de *handshake*.

O primeiro teste foi feito utilizando computadores diferentes para servidor e cliente, aproximando-se de um ambiente real. O objetivo desse teste era de analisar a diferença de tempo entre um *handshake* utilizando o *PFS* e um *handshake* sem *PFS* considerando o tempo que os pacotes levam para viajar de um computador ao outro passando pelo roteador. Devido ao processamento adicional do servidor quando o *PFS* está habilitado, espera-se observar um tempo maior no *handshake* quando essa propriedade está presente.

O segundo teste foi feito utilizando a máquina atacante tanto para atacar, ou seja, realizar os *handshakes*, como para ser servidor. Para isso, foi utilizado uma máquina virtual *VirtualBox*[3] com o sistema *Ubuntu*[5] para ser o servidor e todas as configurações feitas nela foram idênticas às feitas na máquina do servidor. O objetivo desse segundo teste era de analisar a diferença de tempo que levava para o *handshake* com *PFS* e um sem, desconsiderando o *delay* dos pacotes em seus percursos de um computador até o outro. Esperava-se observar uma diferença ainda maior no tempo.

O terceiro e o quarto conjunto de testes tinham como objetivo analisar o impacto de um ataque de negação de serviço num servidor. Este primeiro conjunto de testes visava analisar o ataque de acordo com o número de *threads* usadas pelo atacante utilizando o *PFS*. A metodologia utilizada nesse conjunto de testes é analoga àquela utilizada no primeiro, porém o próprio servidor era utilizado para gerar os dados necessário para a análise, por meio do módulo *server-status*, dispensando, desta forma, o cliente. Os valores considerados nessa análise são aqueles providos pelo servidor ao final de 5 minutos de ataque. Foram obtidos informações sobre o total de acessos no servidor, tráfego total, requisições por segundo, requisições sendo processadas e o uso da *CPU*. Espera-se, nesses testes, observar funções crescentes à medida que aumenta o número de *threads*, pois elas são responsáveis por potencializar o ataque.

O último conjunto de testes tem também como objetivo analisar o impacto de uma ataque de negação de serviço num servidor, mas diferente dos testes anteriores, esses agora visam analisar o impacto em relação ao tempo decorrido do ataque. As informações utilizadas para montar os gráficos desses testes foram as mesmas utilizadas no conjunto de testes anterior, porém as medidas em todos os tempos foram levadas em consideração. São esperados valores mais altos para ataques que utilizaram maior número de *threads* e funções crescentes com o tempo.

# Capítulo 5

# Resultado e Análise

Neste capitulo será mostrado as tabelas e gráficos dos resultados dos testes explicados no capítulo anterior. Também será feita uma análise dos resultados, comparando as diferenças de tempo para as diferentes condições de ataques.

## 5.1 Tempo de resposta do servidor

Este teste foi realizado para analizar o impacto do uso do *PFS* em um servidor. Para isso, foram feitas medições de tempo de *download* de uma simples página do servidor.

Na tabela a seguir está o resultado das medições de tempo realizadas pelo cliente. A coluna em verde representa a medição de tempo feita pelo cliente quando o ataque não está acontecendo. As seguintes colunas mostram a medição de tempo feita pelo cliente para as diferentes variações de ataque na seguinte ordem: uso do *PFS* e crescente uso exponencial de threads (1, 2, 4,. . . ,512) e depois o mesmo crescente uso de *threads* sem o *PFS*. As linhas representam as medições feitas a cada 10 segundos e a última representa a média de tempo de todas as outras acima.

No gráfico 5.2 conseguimos pra ter uma melhor visualização do tempo gasto para cada requisição do cliente nas mais variadas condições de ataque. O eixo $X$ representa os diferentes usos do programa atacante e o eixo $Y$ representa o tempo gasto de uma requisição do cliente em segundos. Percebe-se que, à medida que o número de threads cresce exponencialmente, o tempo de resposta do servidor também cresce em proporções similares, evidenciando a importância do número de *threads* utilizadas durante um ataque.

Infere-se do gráfico 5.2 que, para poucas *threads* praticamente não existe influência no tempo de resposta do servidor, pois ele consegue escalonar todas as requisições sem muito prejuízo para o cliente. Percebe-se, também, que em todas variações possíveis de número de *threads*, sempre que a propriedade do *PFS* estava presente, houve uma maior demora na resposta do servidor, ou seja, a presença dessa propriedade causa um impacto muito

| | SEM ATAQUE | PFS1 | PFS2 | PFS4 | PFS8 | PFS16 | PFS32 | PFS64 | PFS128 | PFS256 | PFS512 | RSA1 | RSA2 | RSA4 | RSA8 | RSA16 | RSA32 | RSA64 | RSA128 | RSA256 | RSA512 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0s | 0.05 | 0.04 | 0.05 | 0.07 | 0.1 | 0.29 | 0.5 | 0.62 | 1.41 | 2.15 | 2.99 | 0.04 | 0.04 | 0.07 | 0.06 | 0.07 | 0.18 | 0.53 | 1.1 | 1.04 | 1.7 |
| 10s | 0.04 | 0.04 | 0.04 | 0.08 | 0.12 | 0.32 | 0.47 | 0.74 | 1.38 | 2.51 | 3.97 | 0.04 | 0.04 | 0.07 | 0.08 | 0.09 | 0.18 | 0.65 | 1.21 | 1.17 | 1.79 |
| 20s | 0.04 | 0.04 | 0.05 | 0.09 | 0.08 | 0.13 | 0.32 | 0.78 | 1.42 | 2.18 | 4.04 | 0.05 | 0.05 | 0.06 | 0.08 | 0.09 | 0.28 | 1.04 | 1.44 | 2.12 | 2.99 |
| 30s | 0.04 | 0.05 | 0.05 | 0.06 | 0.15 | 0.23 | 0.7 | 0.52 | 1.47 | 2.15 | 3.53 | 0.05 | 0.04 | 0.1 | 0.13 | 0.11 | 0.24 | 1.08 | 1.15 | 2.19 | 3.1 |
| 40s | 0.05 | 0.04 | 0.04 | 0.07 | 0.12 | 0.14 | 0.36 | 1.03 | 1.66 | 2.54 | 6.16 | 0.04 | 0.05 | 0.13 | 0.14 | 0.08 | 0.49 | 0.64 | 1.25 | 2.17 | 2.1 |
| 50s | 0.04 | 0.05 | 0.05 | 0.08 | 0.13 | 0.17 | 0.53 | 0.88 | 1.53 | 3.13 | 3.06 | 0.04 | 0.06 | 0.08 | 0.09 | 0.1 | 0.24 | 1.24 | 1.37 | 2.13 | 2.17 |
| 1min | 0.04 | 0.04 | 0.05 | 0.06 | 0.13 | 0.17 | 0.43 | 0.61 | 2.21 | 3.2 | 4.36 | 0.04 | 0.05 | 0.09 | 0.07 | 0.13 | 0.37 | 0.85 | 1.28 | 3.15 | 2.33 |
| 1min e 10s | 0.05 | 0.05 | 0.05 | 0.05 | 0.08 | 0.18 | 0.51 | 1.05 | 1.58 | 3.61 | 2.92 | 0.05 | 0.05 | 0.07 | 0.14 | 0.09 | 0.51 | 1.23 | 1.54 | 2.27 | 2.29 |
| 1min e 20s | 0.05 | 0.05 | 0.04 | 0.07 | 0.12 | 0.27 | 0.61 | 0.99 | 1.64 | 2.79 | 3.1 | 0.05 | 0.05 | 0.08 | 0.14 | 0.13 | 0.42 | 0.84 | 1.15 | 2.25 | 2.25 |
| 1min e 30s | 0.04 | 0.04 | 0.05 | 0.08 | 0.15 | 0.19 | 0.32 | 1.08 | 1.7 | 2.58 | 2.94 | 0.04 | 0.06 | 0.07 | 0.09 | 0.12 | 0.66 | 0.88 | 1.52 | 2.42 | 5.45 |
| 1min e 40s | 0.05 | 0.04 | 0.05 | 0.07 | 0.08 | 0.25 | 0.47 | 1.14 | 1.51 | 2.38 | 66.52 | 0.05 | 0.04 | 0.06 | 0.11 | 0.13 | 0.5 | 1.07 | 1.41 | 2.33 | 5.22 |
| 1min e 50s | 0.05 | 0.05 | 0.06 | 0.06 | 0.11 | 0.17 | 0.46 | 1.03 | 1.72 | 2.21 | 3.08 | 0.04 | 0.06 | 0.07 | 0.1 | 0.1 | 0.42 | 1.12 | 1.4 | 2.44 | 9.23 |
| 2min | 0.04 | 0.04 | 0.06 | 0.08 | 0.11 | 0.19 | 0.74 | 1.04 | 1.68 | 3.26 | 10.17 | 0.05 | 0.05 | 0.07 | 0.07 | 0.11 | 0.53 | 0.83 | 1.58 | 2.5 | 2.51 |
| 2min e 10s | 0.04 | 0.04 | 0.05 | 0.07 | 0.12 | 0.14 | 0.42 | 1.02 | 1.83 | 3.37 | 17.85 | 0.04 | 0.05 | 0.09 | 0.15 | 0.13 | 0.38 | 1.45 | 1.52 | 3.37 | 5.28 |
| 2min e 20s | 0.04 | 0.05 | 0.05 | 0.07 | 0.15 | 0.15 | 0.42 | 0.64 | 1.73 | 5.27 | 3 | 0.05 | 0.05 | 0.07 | 0.08 | 0.14 | 0.43 | 1.21 | 1.5 | 2.17 | 5.32 |
| 2min e 30s | 0.05 | 0.04 | 0.05 | 0.05 | 0.13 | 0.32 | 0.84 | 1.08 | 1.66 | 3.33 | 3.09 | 0.05 | 0.04 | 0.08 | 0.09 | 0.11 | 0.33 | 1.05 | 1.55 | 2.38 | 2.28 |
| 2min e 40s | 0.04 | 0.05 | 0.06 | 0.07 | 0.11 | 0.26 | 0.48 | 1.78 | 1.63 | 2.3 | 2.95 | 0.04 | 0.05 | 0.05 | 0.11 | 0.15 | 0.31 | 1.17 | 1.68 | 2.21 | 5.83 |
| 2min e 50s | 0.05 | 0.05 | 0.04 | 0.07 | 0.09 | 0.16 | 0.95 | 0.94 | 1.61 | 2.67 | 88.41 | 0.04 | 0.05 | 0.06 | 0.19 | 0.16 | 0.34 | 0.99 | 1.64 | 2.15 | 2.32 |
| 3min | 0.04 | 0.04 | 0.05 | 0.08 | 0.09 | 0.18 | 0.55 | 1.01 | 2.05 | 3.2 | 2.96 | 0.04 | 0.06 | 0.08 | 0.11 | 0.17 | 0.41 | 0.93 | 1.51 | 2.32 | 2.31 |
| 3min e 10s | 0.04 | 0.04 | 0.06 | 0.13 | 0.17 | 0.16 | 0.83 | 2.01 | 1.7 | 2.39 | 4.1 | 0.05 | 0.06 | 0.06 | 0.1 | 0.13 | 0.33 | 1.01 | 1.4 | 2.3 | 17.27 |
| 3min e 20s | 0.05 | 0.04 | 0.05 | 0.09 | 0.14 | 0.11 | 0.68 | 1.56 | 1.67 | 5.33 | 4.15 | 0.05 | 0.05 | 0.06 | 0.09 | 0.12 | 0.41 | 0.84 | 1.48 | 2.23 | 2.09 |
| 3min e 30s | 0.05 | 0.05 | 0.05 | 0.09 | 0.08 | 0.16 | 0.65 | 7.61 | 1.63 | 3.38 | 3.07 | 0.04 | 0.05 | 0.06 | 0.11 | 0.11 | 0.49 | 1.35 | 1.54 | 2.22 | 3.38 |
| 3min e 40s | 0.05 | 0.05 | 0.05 | 0.09 | 0.12 | 0.21 | 0.55 | 1.05 | 1.45 | 2.58 | 2.99 | 0.05 | 0.05 | 0.07 | 0.13 | 0.11 | 0.38 | 0.98 | 1.47 | 2.1 | 2.38 |
| 3min e 50s | 0.05 | 0.06 | 0.06 | 0.1 | 0.07 | 0.17 | 0.72 | 1.22 | 1.62 | 2.73 | 3.1 | 0.05 | 0.05 | 0.07 | 0.12 | 0.12 | 0.51 | 1.02 | 1.44 | 2.24 | 2.28 |
| 4min | 0.05 | 0.05 | 0.06 | 0.07 | 0.17 | 0.23 | 0.62 | 1.13 | 2.59 | 2.19 | 9.94 | 0.05 | 0.06 | 0.06 | 0.09 | 0.14 | 0.64 | 1.46 | 1.56 | 2.32 | 2.35 |
| 4min e 10s | 0.05 | 0.05 | 0.08 | 0.06 | 0.12 | 0.18 | 0.81 | 1.85 | 1.64 | 2.53 | 3.93 | 0.05 | 0.05 | 0.05 | 0.1 | 0.21 | 0.43 | 0.83 | 1.52 | 2.37 | 3.46 |
| 4min e 20s | 0.05 | 0.05 | 0.06 | 0.06 | 0.14 | 0.22 | 0.78 | 0.81 | 1.94 | 3.25 | 3.1 | 0.05 | 0.05 | 0.06 | 0.09 | 0.12 | 0.44 | 0.66 | 1.79 | 2.41 | 9.31 |
| 4min e 30s | 0.04 | 0.05 | 0.06 | 0.06 | 0.1 | 0.22 | 1.01 | 0.96 | 1.66 | 2.56 | 3.42 | 0.05 | 0.06 | 0.06 | 0.13 | 0.18 | 0.53 | 1.39 | 1.25 | 3.36 | 2.32 |
| 4min e 40s | 0.04 | 0.06 | 0.05 | 0.08 | 0.11 | 0.17 | 0.75 | 1.39 | 1.78 | 2.37 | 3.02 | 0.04 | 0.05 | 0.05 | 0.2 | 0.13 | 0.58 | 1.22 | 1.66 | 2.5 | 2.44 |
| 4min e 50s | 0.05 | 0.05 | 0.06 | 0.08 | 0.13 | 0.16 | 0.33 | 1.11 | 1.99 | 2.45 | 3.07 | 0.04 | 0.05 | 0.08 | 0.13 | 0.13 | 0.35 | 1.59 | 1.58 | 2.16 | 2.27 |
| 5min | 0.04 | 0.05 | 0.04 | 0.06 | 0.1 | 0.31 | 0.32 | 0.96 | 1.68 | 2.56 | 3.03 | 0.05 | 0.05 | 0.07 | 0.08 | 0.12 | 0.49 | 1.22 | 1.41 | 3.38 | 2.54 |
| | 0.04516129 | 0.0464516 | 0.0522581 | 0.0741935 | 0.1167742 | 0.2003226 | 0.5848387 | 1.2787097 | 1.7022581 | 2.8758065 | 9.0974194 | 0.0454839 | 0.0506452 | 0.0709677 | 0.1096774 | 0.1235484 | 0.4129032 | 1.0441935 | 1.4483871 | 2.3345161 | 3.8245161 |

Figura 5.1: Tabela com o tempo de resposta de cada requisição feita pelo cliente de acordo com o número de *threads* e da *cipher suite* escolhida.

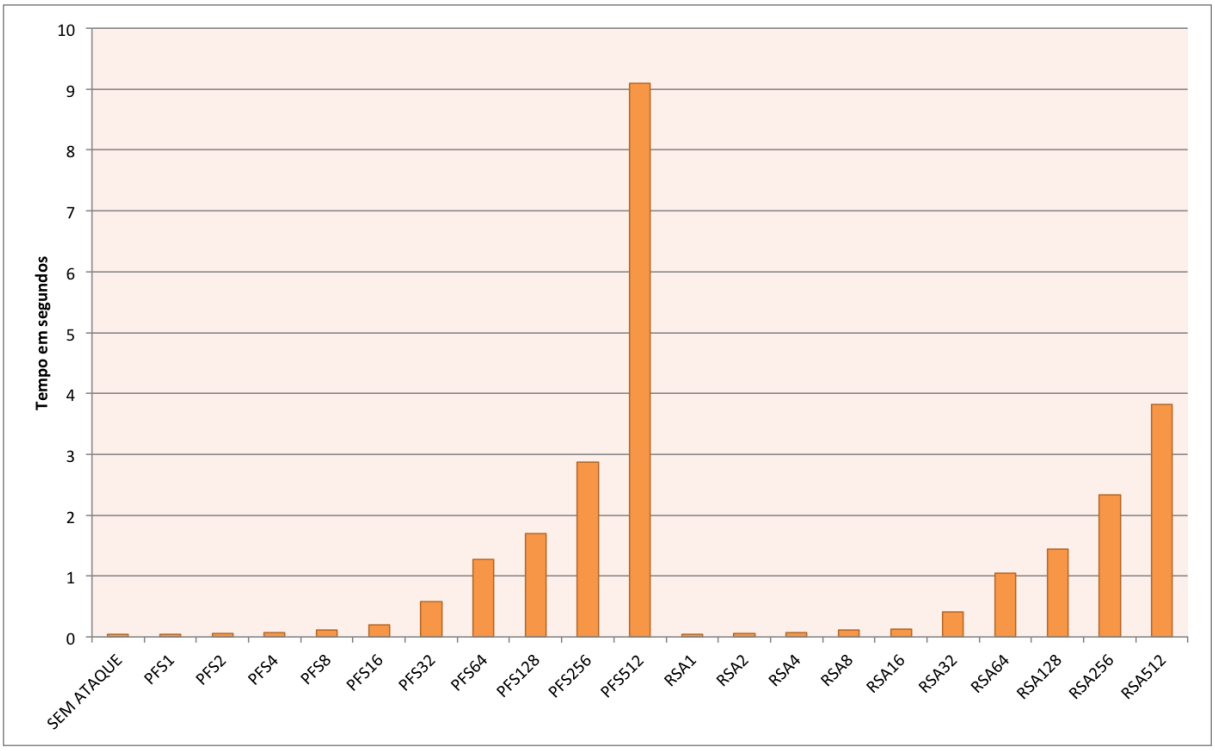


Figura 5.2: Gráfico mostrando o crescimento exponencial do tempo de resposta da diferença entre ataques usando e não usando o PFS.

maior no tempo de resposta do servidor ao cliente legítimo. Essa diferença torna-se mais notável a medida que cresce o número de *threads*, pois há momentos em que o servidor está completamente ocupado e não consegue atender as requisições do cliente, tornando o serviço temporariamente indisponível.

Outra importante característica que se pode inferir desses dados, melhor representado

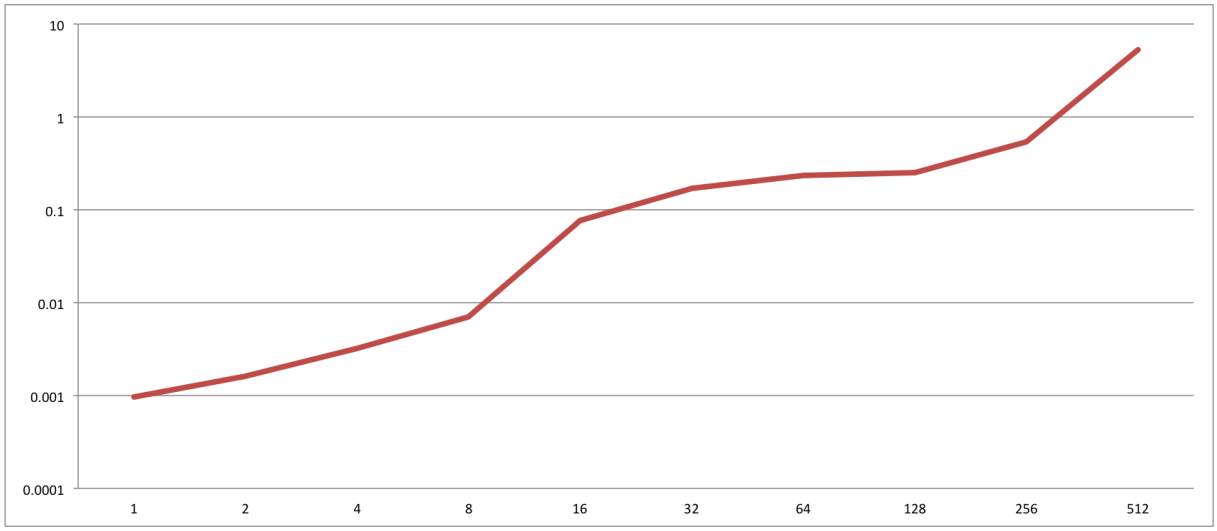


Figura 5.3: Gráfico mostrando a diferença de tempo entre um ataque com e sem *PFS* de acordo com o número de *threads*..

no gráfico 5.3, é que a diferença entre o tempo de resposta de um ataque utilizando o *PFS* e outro não, para um mesmo número de threads, cresce exponencialmente também. No gráfico 5.3 o eixo $X$ representa o número de *threads* e o eixo $Y$ representa a diferença de tempo de resposta do servidor com o *PFS* pela diferença de tempo sem o *PFS* em escala logarítmica. A curva, neste gráfico, que se aproxima de uma reta, indica a presença de um aumento exponencial do tempo de resposta entre um ataque que utiliza o *PFS* e outro que não utiliza, com o aumento do número de *threads*.

O gráfico 5.4 foi feito a partir das medições de tempo do ataque usando *RSA* e mostra a evolução dos tempos de resposta no cliente ao longo do tempo. É importante notar que o eixo $Y$ também está em escala logarítmica e que a distância quase que equidistantes entres as curvas das *threads* evidencia um aumento exponencial no tempo de resposta do servidor no cliente com o aumento exponencial no número de *threads* usadas no ataque. Nota-se, também, que o tempo de resposta do servidor, apesar de aparentemente não seguir um padrão ao longo do tempo, aumenta em todas as variações de ataque nos primeiros 10 ou 20 segundos em que o programa está rodando. Isso se deve de que, nesse momento, o ataque chegou ao máximo de sua capacidade de consumo dos recursos do servidor, não conseguindo degradá-lo mais.

## 5.2 Tempo do *handshake*

Este teste foi realizado para comparar o tempo de um handshake *TLS/SSL* utilizando a propriedade do *Perfect fForward Secrecy* e sem ela.

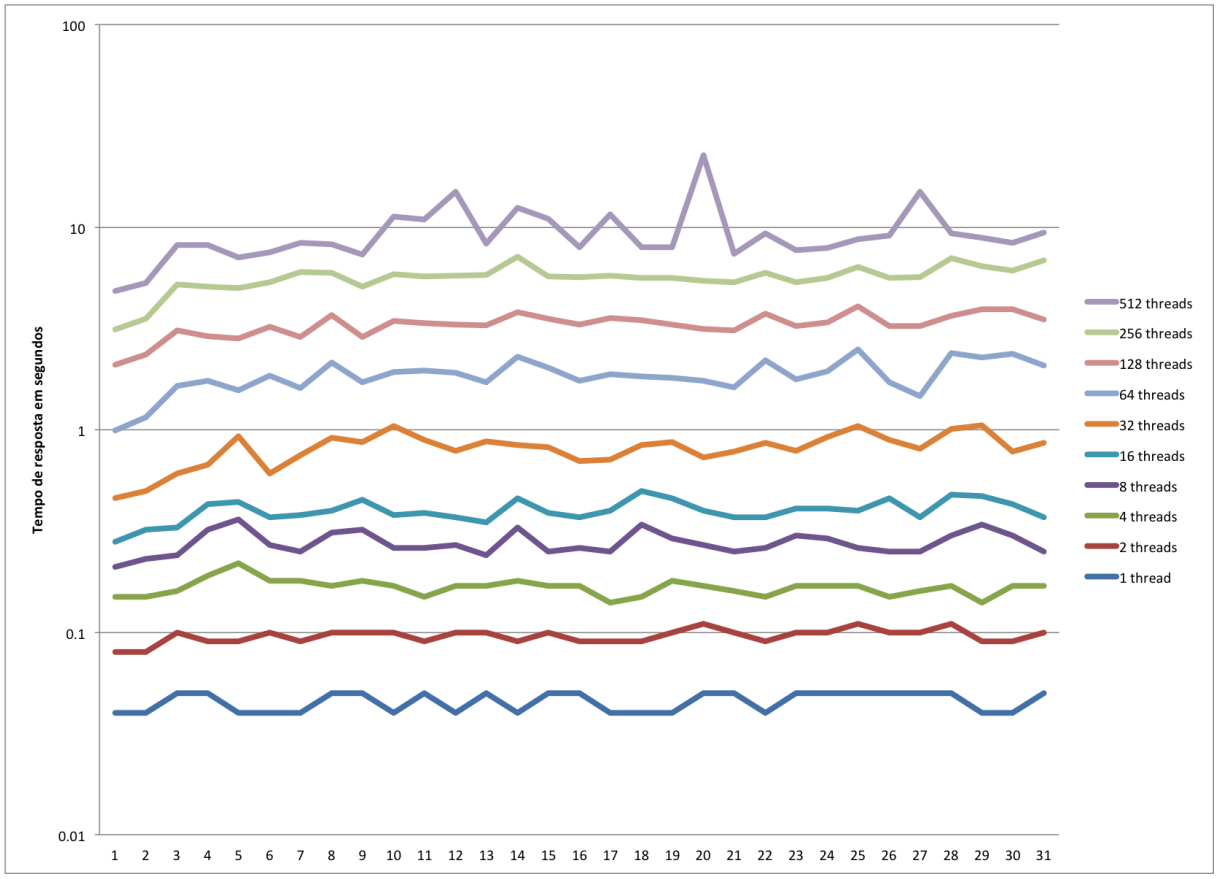


Figura 5.4: Gráficos com a evolução dos tempos de resposta no cliente ao longo do tempo..

Pelo gráfico 5.5 percebe-se que o *handshake* utilizando o *PFS* demora cerca de 7.4 vezes mais que o mesmo sem o *PFS*. Essa demora se deve ao fato de no *handshake* com *PFS* existir um processamento adicional do servidor para se calcular um parâmetro usado no método *Diffie-Hellman*, que é enviado do servidor ao cliente no pacote *ServerKeyExchange*. Para poucas requisições, essa diferença de tempo pode não fazer tanta diferença, mas em ambientes reais, onde *websites* recebem milhares de pedidos de conexão a cada minuto, essa diferença de tempo pode sim interferir no desempenho do site.

O segundo teste foi feito utilizando um computador para o cliente e uma máquina virtual no mesmo computador para o servidor a fim de amenizar a influência do tempo na troca de pacotes.

Percebe-se pelo gráfico 5.6 que, nessa situação, a demora de um handshake com *PFS* é cerca de 11.6 vezes maior que um *handshake* sem *PFS*. Isso ocorre porque o transporte de pacotes entre máquinas virtuais em um mesmo computador é mais rápido que em computadores diferentes. Vale ressaltar que essa é uma diferença nominal de tempo, pois em ambientes reais existem vários roteadores no caminho, o que causa uma diminuição na diferença de tempo.

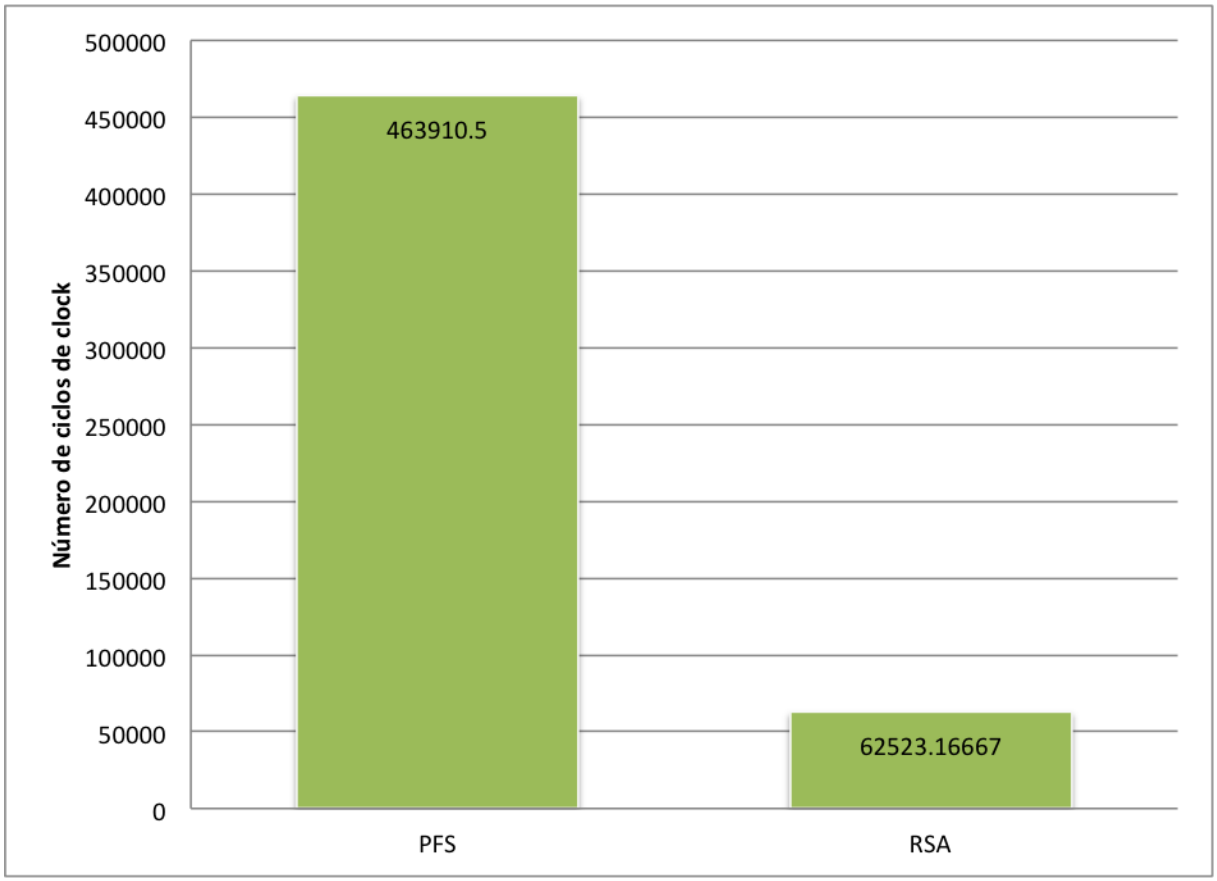


Figura 5.5: Gráfico mostrando o número de ciclos de *clock* que levaram o processador a realizar 100 *handshakes* usando e não usando o *PFS* com o servidor e atacante utilizando máquinas diferentes..

## 5.3 Impacto no servidor em relação ao número de threads

Estes testes foram feitos para comparar o impacto de negação de serviço no servidor de acordo com o número de *threads* usadas pelo atacante. Neles foram obtidos as informações de status do servidor após 5 minutos de ataque para variados números de *threads*.

A figura 5.7 mostra que o total de acessos ao servidor é crescente de acordo com o aumento de *threads* no ataque. Essa é uma evidência clara de que o servidor está sofrendo uma degradação cada vez maior, pois ele terá de alocar cada vez mais recursos para permitir esse crescente número de acessos.

A figura 5.8 mostra que o tráfego total é crescente de acorco com o aumento de threads no ataque, assim como acontece no número de acessos. Esse gráfico também evidencia uma degradação no servidor cada vez maior, pois existe um limite máximo de banda que pode trafegar pelo servidor até que seus pacotes sejam descartados.

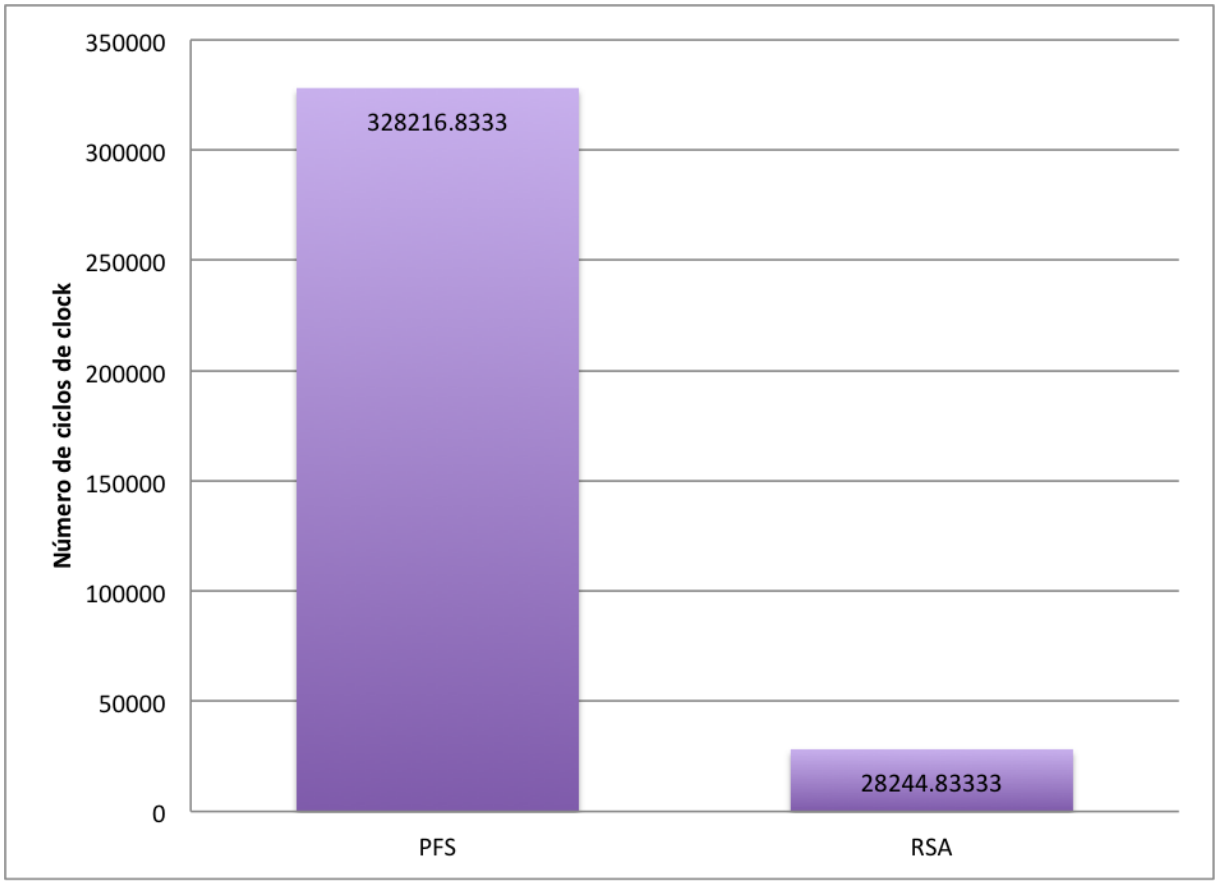


Figura 5.6: Gráfico mostrando o número de ciclos de clock que levaram o processador a realizar 100 *handshakes* usando e não usando o *PFS* com o servidor usando uma máquina virtual no mesmo computador atacante..

A figura 5.9 mostra que o número de requisições por segundo é crescente até por volta de 8 threads usadas no ataque. Após isso, este número se mantém em uma faixa constante provavelmente devido à saturação do servidor, na faixa de 175 requisições por segundo, no caso desse servidor.

A figura 5.10 mostra que o número de requisições sendo processadas é crescente até por volta de 128 *threads* usadas no ataque. Após isso, o servidor provavelmente entra em saturação e consegue processar apenas um máximo de 150 requisições por segundo. As requisições não processadas provavelmente ou entram num *buffer* de espera ou são descartadas, provocando uma degradação maior do serviço.

A figura 5.11 mostra que o uso da CPU no servidor cresce exponencialmente com o aumento exponencial de threads usadas no ataque. Contudo esse valor tende a chegar ao seu máximo, 200%, a partir de 32 *threads*, onde seu valor atinge 192%. A partir desse momento acontece uma degradação maior do serviço, pois o servidor não consegue processar todas as requisições que estão chegando. O carregamento da CPU supera 100%, pois a

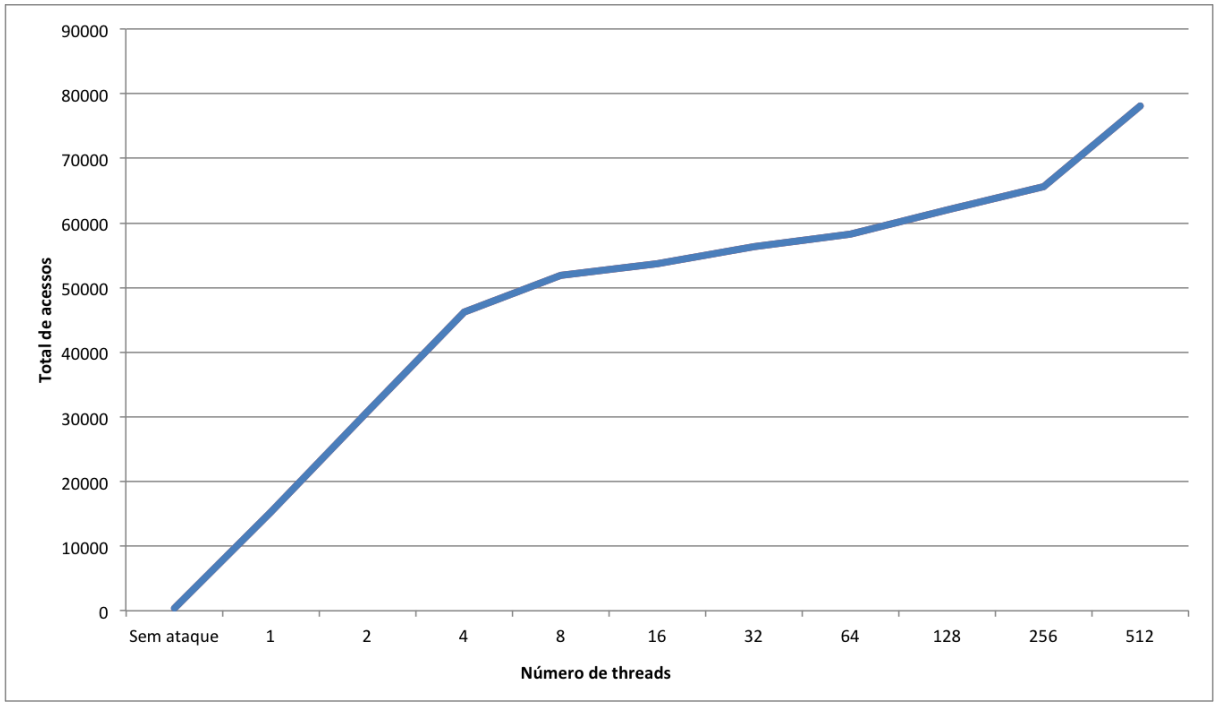


Figura 5.7: Gráfico do número total de acessos ao servidor de acordo com o número de *threads*..

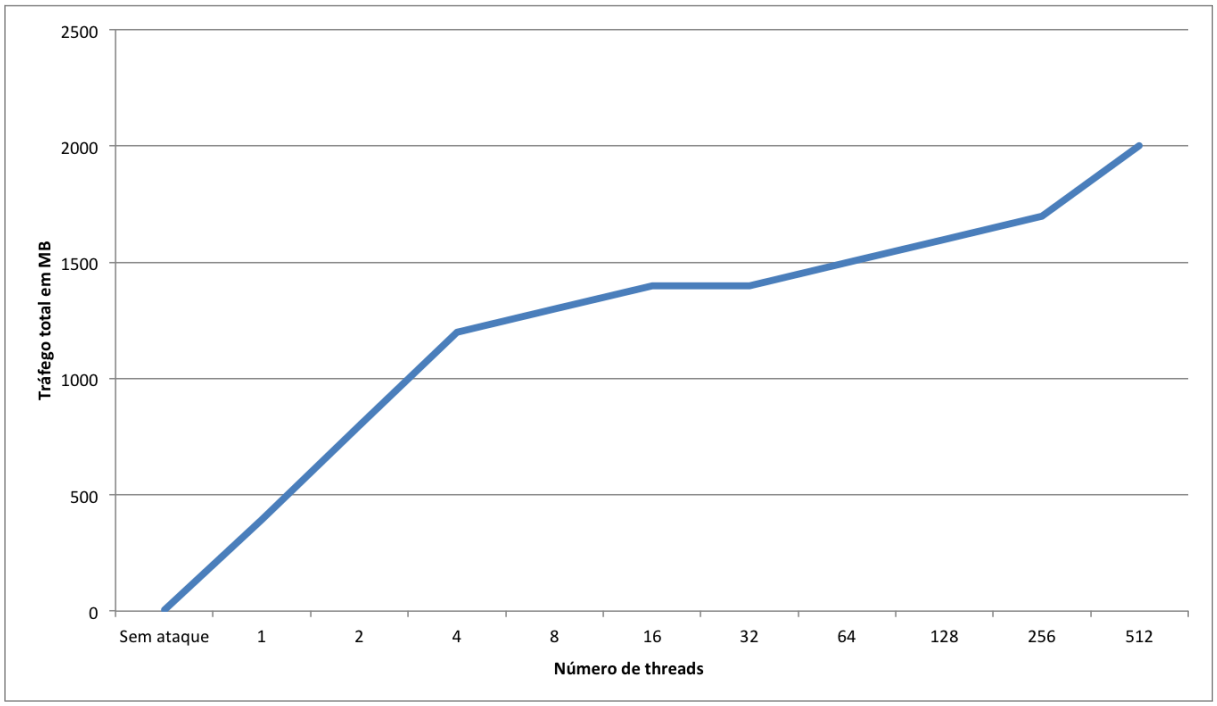


Figura 5.8: Gráfico do tráfego total pelo servidor de acordo com o número de *threads*..

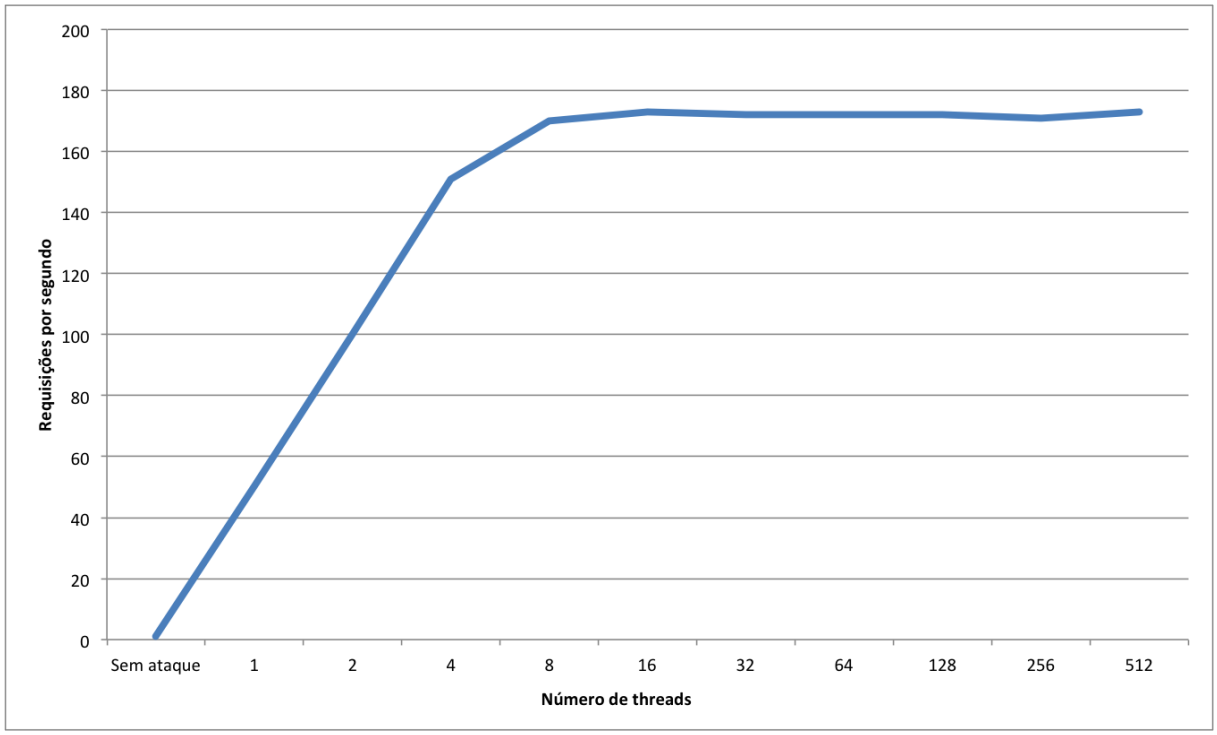


Figura 5.9: Gráfico com o número de requisições por segundo de acordo com o número de *threads*..

cada core do processador do servidor é atribuido uma capacidade de 100%, totalizando 200% no caso deste servidor devido ao seus 2 *cores*.

Pelos gráficos mostrados, nota-se que a degradação do servidor vai acontecendo aos poucos e não só por um, mas por vários motivos. Com 8 *threads*, nota-se que o servidor entra em saturação em relação ao número de requisições que ele consegue captar. A partir de 32 *threads* atacantes é a vez da *CPU* alcançar seu limite de processamento, degradando mais o serviço e, consequentemente, aumentando o tempo de resposta do servidor. Com 128 *threads* atacando, o número de requisições sendo processadas atinge seu máximo, contribuindo mais ainda para a degradação do serviço, pois requisições não atendidas prontamente ou entrarão para uma fila de espera ou serão descartadas.

## 5.4 Status do servidor de acordo com o tempo

Este teste foi feito para comparar o impacto do tempo na negação de serviço do servidor em relação ao tempo decorrido do ataque.

A figura 5.12 mostra que o número total de acessos cresce linearmente com o tempo com uma inclinação cada vez maior à medida que cresce o número de threads usadas

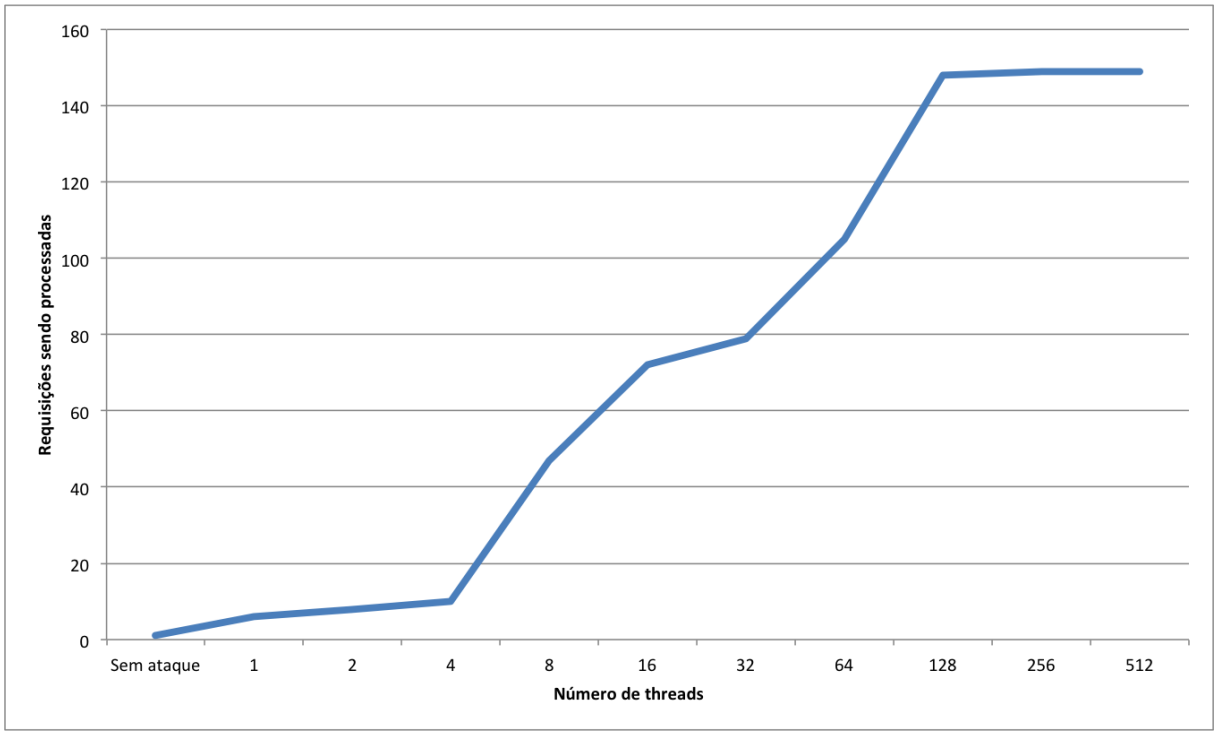


Figura 5.10: Gráfico com o número de requisições sendo processadas no momento da medição de acordo com o número de *threads*..

no programa atacante. Essas funções crescentes confirmam a ideia de que o ataque está causando efeito durante todo o tempo em que o programa está rodando.

A figura 5.13 mostra que, assim como o número de acessos, o tráfego total no servidor também cresce com uma inclinação cada vez maior a medida que o cresce o número de *threads*. Nota-se, contudo, a existência de patamares para elevados números de *threads*. Isso ocorre por causa da precisão dada pelo *server-status* do Apache quando o tráfego ultrapassa 1GB. Infere-se, entretando, que o crescimento do tráfego com o passar do tempo seja linear.

A figura 5.14 mostra que o uso da *CPU* do servidor mantém uma taxa constante para os diferentes números de *threads* usadas pelo programa atacante. Percebe-se que a partir de 32 *threads* atacando, como analisado em testes anteriores, o valor da porcentagem atinge um nível limite máximo próximo a 200%. Percebe-se também, pelo gráfico, que nos primeiros 20 segundos que o programa está rodando os diferentes ataques já atingem seu nível máximo de carregamento da CPU, caracterizando um ataque rápido.

A figura 5.15 mostra que, assim como no caso do uso da CPU, a partir de um certo número de *threads*, o servidor chega ao máximo de sua capacidade. Neste caso, 8 *threads* apenas são necessárias para o servidor chegar nesse limite. Também nota-se, assim como no gráfico da CPU, que já nos primeiros 20 segundos do programa rodando, os diferentes

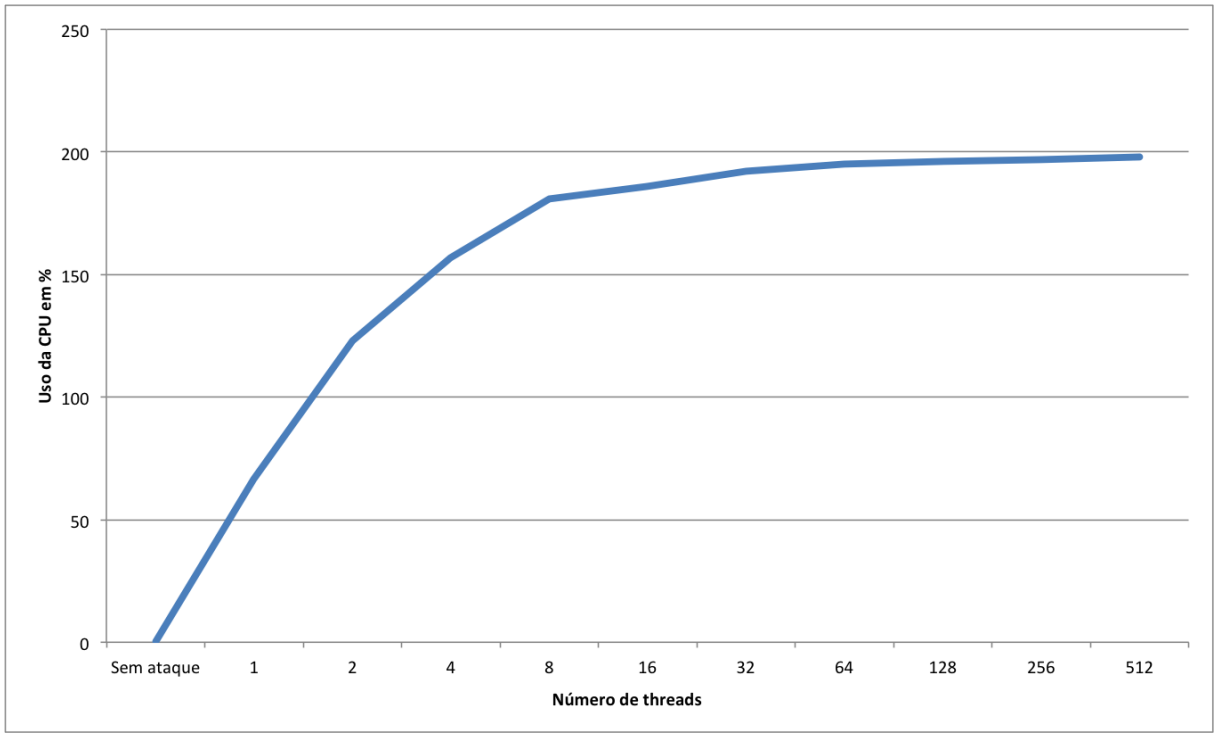


Figura 5.11: Gráfico com o carregamento da CPU de acordo com o número de *threads*..

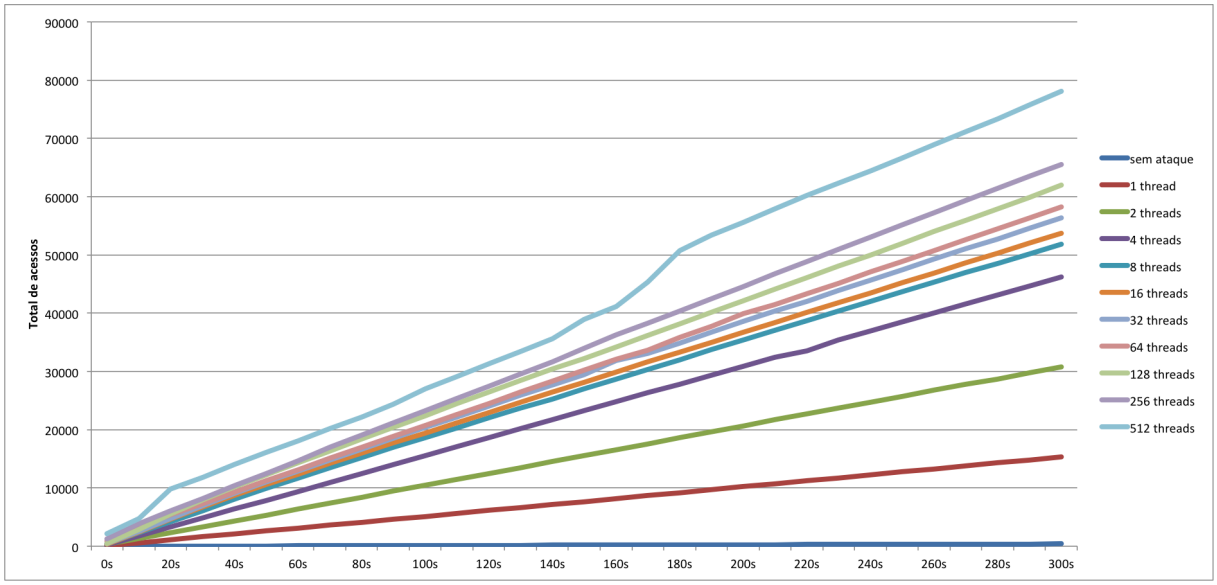


Figura 5.12: Gráfico com o total de acessos ao servidor de acorco com o número de threads.

ataques atingem seu nível máximo de potência, reinforçando a ideia de que este é um ataque rápido.

O último gráfico 5.16 talvez seja inconclusivo a primeira vista, mas mostra que para a partir de ataques com 128 *threads* o número de requisições sendo processadas no momento

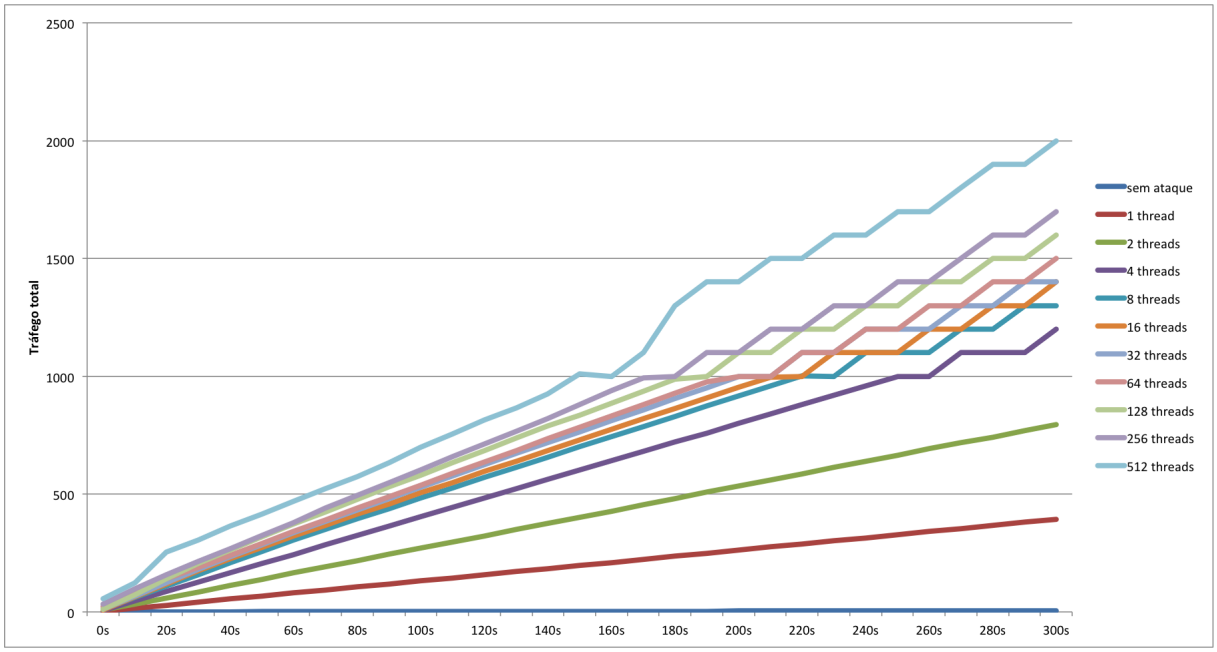


Figura 5.13: Gráfico com o tráfego total de acordo com o número de threads.

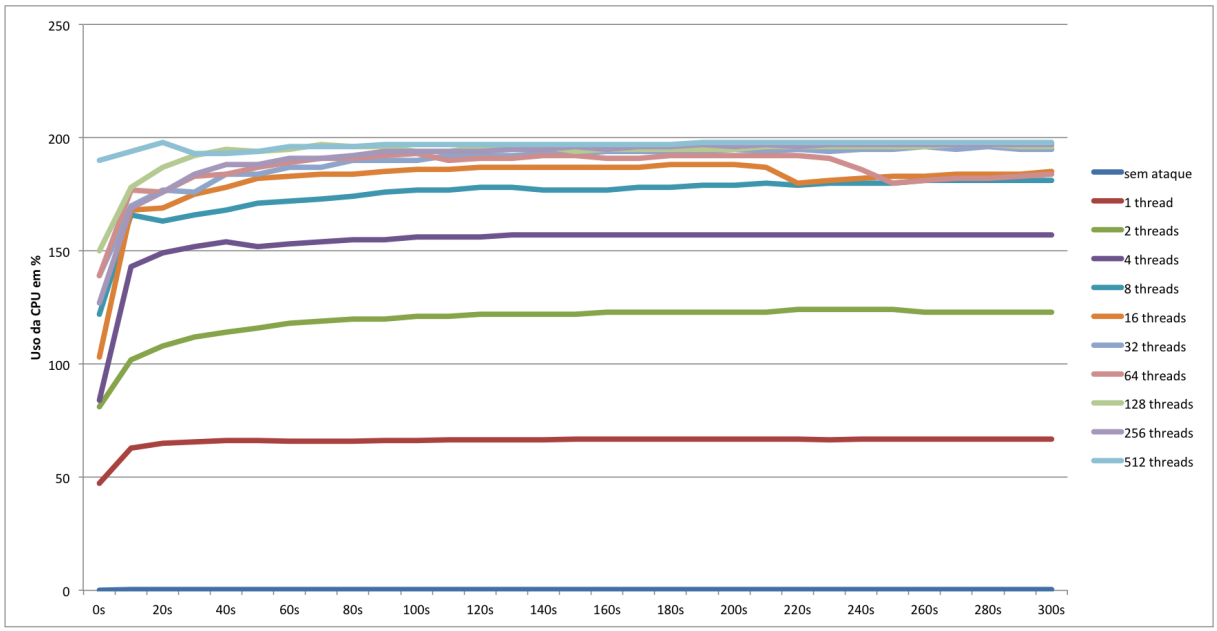


Figura 5.14: Gráfico do uso da CPU do servidor de acordo com o número de threads.

da medição atinge um valor máximo que o servidor consegue processar.

Todos os gráficos dos testes indicam que, com o aumento do número de *threads* usadas no programa, aumenta-se a eficiência do ataque. Vários recursos do servidor são comprometidos com o ataque proposto nesse trabalho à medida que cresce o número de threads, contribuindo para um maior nível de degradação do serviço.

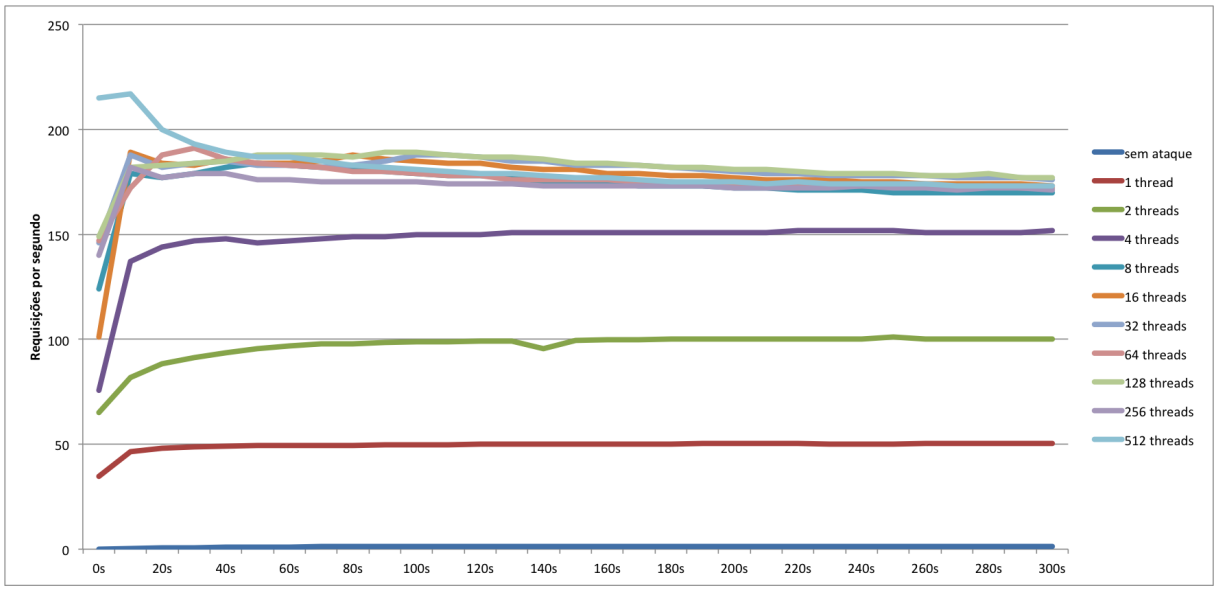


Figura 5.15: Gráfico com as requisições por segundo de acordo com o número de threads.

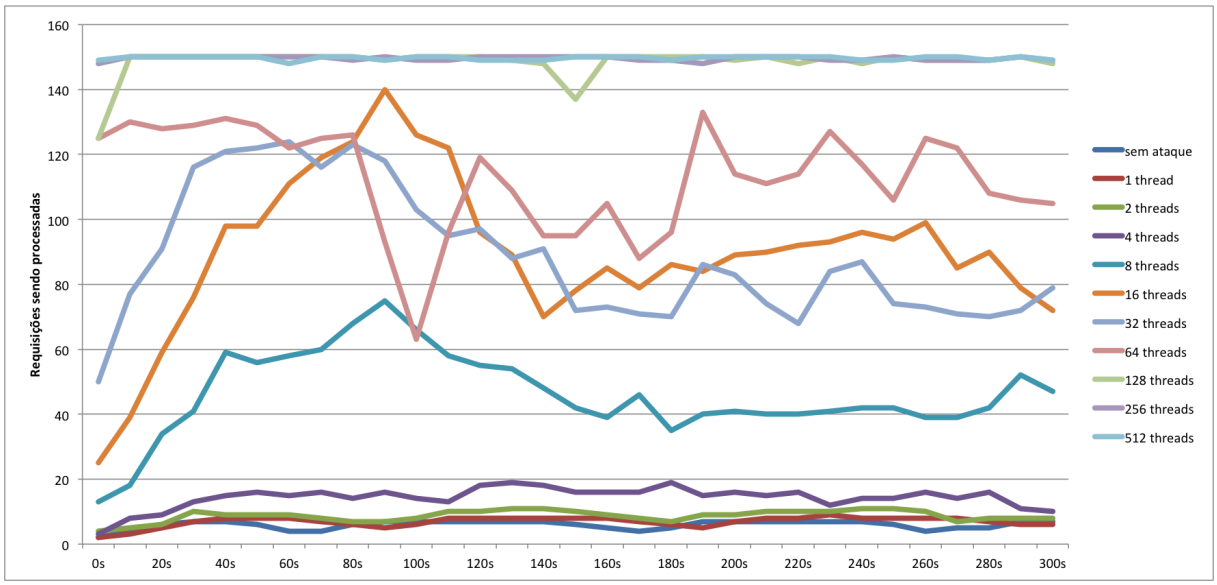


Figura 5.16: Gráfico com as requisições sendo processadas de acordo com o número de threads.

# Capítulo 6

# Conclusão

## 6.1 Segurança e Eficiência

Os resultados obtidos nos testes se mostraram bastante coerentes com o esperado e ressaltam o perigo desse tipo de ataque. Um computador apenas é capaz de negar parcialmente o serviço de um servidor de pequeno porte fazendo apenas simples requisições a ele. Vale ressaltar, também, que muitos sites atualmente nao contam com servidores robustos capazes de aguentar ataques como esse. O desenvolvimento de programas mais complexos para ataques pode deixar o estrago ainda maior.

Além disso, todos os testes foram feitos em ambientes controlados e isolados. Em ambientes reais, ainda há a uma demora maior na resposta dos pacotes devido a rede de nós que separam cliente e servidor. Um servidor real quase sempre possui uma aplicação mais robusta que irá consumir mais dele, potencializando, assim, um possível ataque. Todos esses fatores contribuem para um ataque mais eficaz e devastador.

O uso da *cipher suite* adequada deve ser um ponto importante na hora de configurar um servidor, pois elas possuem tempos de resposta variados. É tudo uma questão segurança e eficiência que devem ser colocadas numa balança. *Cipher suite* que utilizam o *Perfect Forward Secrecy* contribuem para uma maior segurança do servidor, mas demandam um processamento adicional dele, ocasionando num tempo cerca de 7 vezes a mais no *handshake* quando comparado a um servidor sem essa propriedade. Por outro lado, *cipher suite* que não utilizam o *Perfect Forward Secrecy* tem uma resposta do servidor muito mais rápida, tornando-o mais resistente a ataques de negação de servico, mas fica mais vulnerável a roubos de informações confidenciais.

## 6.2 Trabalhos futuros

Observando os resultados, conclui-se que é possível aprimorar mais ainda esses ataques para deixá-los mais devastadores. Essas ferramentas podem ser trabalhadas, por exemplo, para se montar um ataque de negação de serviço distribuído, aumentando drasticamente seu poder de ataque. Outra alternativa seria explorar falhas no protocolo *TLS/SSL*, potencializando o poder de ataque do programa.

Uma análise mais precisa desses ataques pode ser construída realizando os testes em circunstâncias mais realistas das que foram feitas neste trabalho, como um servidor com maior limite de conexões e com várias aplicações rodando nele. Os testes dariam resultados que mais se aproximariam de ataques reais, fazendo com que soluções mais eficientes poderiam ser estudadas para combater esses ataques.

Existem, contudo, algumas medidas capazes de amenizar o impacto de um ataque. O uso de um *firewall* bloqueando requisições do atacante com base no *IP* é uma medida bastante simples e eficaz. Contudo, não é suficiente no caso de ataques distribuídos, onde várias fontes são utilizadas para atacar o servidor, tornando a tarefa de bloquear os *IP*s mais difícil e abrindo a possibilidade de se bloquear tráfego legítimo.

Outra possível medida para amenizar o ataque seria de aumentar os recursos do servidor para que eles consigam operar de maneira rápida mesmo em meio de um ataque. Essa solução, no entando, pode exigir um alto investimento estrutural e financeiro, tornando-a, assim, muitas vezes inviável.

Ataques de negação de serviço nao podem ser meramente resolvidos com um simples produto, mas sim com uma aprofundada análise de todos elementos da computação, redes e sistemas, incluindo o design, a implementação e a manuntenção, para assegurar que todas medidas sejam tomadas para reduzir pontos de falhas e resistir aos ataques.

# Referências


[1] The apache software foundation. http://www.apache.org/. 15

[2] Openssl project. https://www.openssl.org/. 14, 17

[3] Oracle vm virtualbox. https://www.virtualbox.org/. 18

[4] The sprawl. https://www.thesprawl.org/research/tls-and-ssl-cipher-suites/. 13

[5] Ubuntu: The leading os for pc, tablet, phone and cloud. http://www.ubuntu.com/. 18

[6] MEHMUD ABLIZ. Internet denial of service attacks and defense mechanisms. people.cs.pitt.edu/ mehmud/docs/abliz11-TR-11-178.pdf. 7, 8

[7] T. Dierks and E. Rescorla. The transport layer security protocol version, 2008. 12

[8] PARKER HIGGINS. Pushing for perfect forward secrecy, an important web privacy protection. https://www.eff.org/deeplinks/2013/08/pushing-perfect-forward-secrecy-important-web-privacy-protection. 2

[9] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. http://www.eecis.udel.edu/ sunshine/publications/ccr.pdf. 8, 10

[10] PhD. Qijun Gu. Denial of service attacks. s2.ist.psu.edu/paper/DDoS-Chap-Gu-June-07.pdf, 2007. 9

[11] Sally Vandeven. Ssl/tls: What's under the hood. http://www.sans.org/reading-room/whitepapers/authentication/ssl-tls-hood-34297. 12, 13

[12] Saman Taghavi Zargar. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. http://d-scholarship.pitt.edu/19225/1/FinalVersion.pdf. 5, 10, 11