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

Ataque de negação de serviço por reflexão amplificada explorando Memcached

Igor F. Miranda

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

Orientador
Prof. Dr. João José Costa Gondim

Brasília
2019

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

Ataque de negação de serviço por reflexão amplificada explorando Memcached

Igor F. Miranda

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

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

Prof. Dr. Marcelo Grandi Mandelli
CIC/UnB

Prof. Dr. Marcos F. Caetano
CIC/UnB

Prof. José Edil Guimarães de Medeiros
Coordenador do Curso de Engenharia da Computação

Brasília, 25 de março de 2019

Dedicatória

Dedico esse trabalho a todos aqueles que estiveram ao meu lado me ajudando e apoiando nessa minha graduação.

Agradecimentos

Agradeço ao Professor Dr. Gondim por ter me disponibilizado seu tempo e conhecimento. Agradeço aos meus amigos pela força e palavras de incentivo que me deram. Agradeço ao meu amigo João Apolinário por ter me disponibilizado seu computador para a realização dos testes desse trabalho. Agradeço aos meus amigos João Versiani, Angelita Torres e Gabriela Romano pela ajuda na revisão do trabalho. Agradeço o meu amigo Andrei Buslik pela ajuda nos momentos difíceis que tive na UnB. E por último e não menos importante agradeço aos meus pais Valdemir Miranda e Kátia Fernandes por todo apoio, ajuda e carinho que me dão.

Resumo

Ataques de negação de serviço (DoS) representam uma grande ameaça para a estabilidade da internet e são uma preocupação constante dos profissionais de segurança da informação. Esse tipo de ataque tem como objetivo interromper o acesso de usuários legítimos a um determinado serviço. Muitas vezes o atacante se utiliza de máquinas expostas na internet para refletir e amplificar o ataque, e essa subcategoria de ataque de negação de serviço é o foco do estuda desse trabalho. Aqui são apresentados os conceitos básicos do ataque de negação de serviço e a análise da utilização da aplicação Memcached na realização de um ataque DoS refletido. De maneira secundária iremos apresentar a ferramenta desenvolvida para a realização dos teste apresentados.

Palavras-chave: Memcached, negação de serviço, negação de serviço distribuída, amplificação, reflexão

Abstract

Denial of service attacks presents a very serious threat to the stability of the internet and is a major concern for security professionals. This type of attack is intended to stop legitimate users from accessing a particular service. Often the attacker uses machines exposed on the internet to reflect and amplify the attack, and this subcategory of denial of service attack will be the focus of this study. We will present the basic concepts of DoS attack and analyze the potential of a Memcached server in performing a reflected DoS. In a secondary way we will present the tool used to performe tests.

Keywords: Memcached, denieal of service, distributed deniel of service, amplification, reflection

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACID Atomicity, Consistency, Isolation, Durability.

BSoD Blue Screen of Death.

CLI Command-line user interface.

DARPA Defense Advanced Research Projects Agency.

DDoS Distribuited Denial of Service.

DNS Domain Name System.

DoD Department of Defense.

DoS Denial of Service.

ICMP Internet Control Message Protocol.

IRC Internet Relay Chat.

OOB Out of Band.

RUDY R-U-Dead-Yet.

Capítulo 1

Introdução

A *Internet* tem início em meados de 1969, como ARPANET em uma pesquisa financiada pela Defense Advanced Research Projects Agency (DARPA), uma agência do Department of Defense (DoD) dos Estados Unidos da América. No final desse ano, a ARPANET foi lançada e conectava quatro universidades. Em 1973 a primeira conexão internacional da ARPANET foi feita. Em 1982 o conceito de *Internet* começou a surgir e, junto com ele, novos protocolos surgiram para padronizar a comunicação entre essa redes como os protocolos IP e TCP, que continuam sendo os protocolos base da *Internet* até hoje [1].

Essa tecnologia revolucionou a troca de informações, transformando-a em algo fácil, rápido e eficiente, pavimentando o caminho para a sociedade globalizada e estando hoje presente em ambientes essenciais de que uma nação necessita. Outra grande revolução que a *internet* trouxe para a sociedade pós-moderna foi a criação de um novo ambiente de conflito. Antes de seu surgimento, os conflitos se passavam apenas na esfera física, onde grande movimentações do inimigo podem ser descobertas, possibilitando movimentos de antecipação. E além disso evidências físicas podem ser deixadas, possibilitando seu rastreio e talvez sua captura. Com o surgimento da *internet* os conflitos passaram a existir também na esfera digital, que é um ambiente sem fronteiras, com maior anonimato e a um custo muito menor que um conflito na esfera física. Os ataques que ocorrem em conflitos na esfera digital são denominados como ciberataques.

Apesar de a *internet* criar essa nova esfera de conflito, não é nela que reside o perigo dos ciberataques. A maior ameaça está na enorme quantidade de dispositivos vulneráveis conectados a ela. Novas vulnerabilidades em softwares executados por esses dispositivos são sempre descobertas e, uma vez apresentada ao público, podem vir a comprometer uma imensa quantidade de dispositivos [1]. Outro fator importante é que a sofisticação dos ciberataques tem crescido, enquanto a expertise necessária para executá-los tem diminuído. Isso é explicado com o advento de ferramentas para a preparação, gerenciamento

e execução desses ataques. A figura 1.1 apresenta essa sofisticação dos ciberataques pela expertise do atacante.

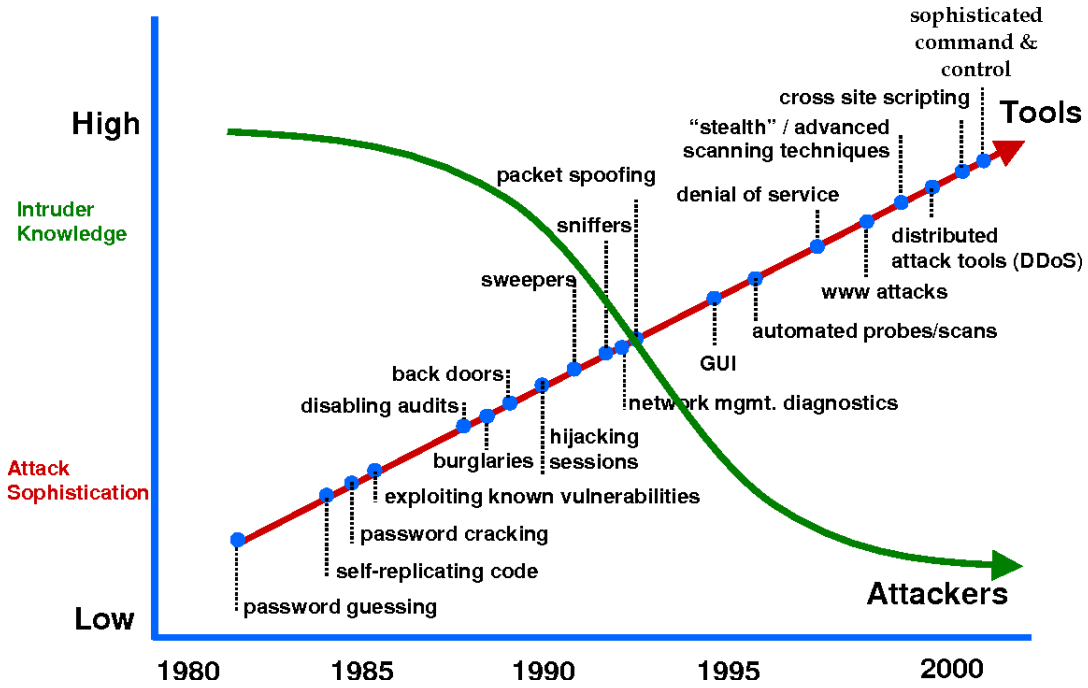

Figura 1.1: Sofisticação do ataque pela expertise do atacante [1]

As motivações para se efetuar um ataque na esfera digital não diferem das motivações de ataques na esfera física, sendo eles políticos, econômicos e socioculturais [2].

Com toda essa disponibilidade e facilidade para a execução de ciberataques somadas à dependência da *internet* que se tem hoje, tornou-se algo rotineiro a ocorrência de ciberataques em todo o mundo.

1.1 Motivação

Com toda essa dependência que se tem atualmente da *internet* na troca de informações para a tomada de decisão, era inevitável o surgimento de ataques que bloqueassem os serviços disponibilizados por ela. Um ciberataque amplamente utilizado para cumprir esse objetivo é o ataque de negação de serviço.

Esse ataque consiste em esgotar os recursos de uma máquina ou infraestrutura , impedindo que a vítima consiga disponibilizar seus serviços na rede ou acessar outros. Para efetuar o ataque, o atacante pode utilizar uma única fonte ou distribuir o trabalho em várias fontes para aumentar o seu poder computacional e anonimato, transformando o ataque em um ataque de negação de serviço distribuído.

Esse tipo de ataque não possui uma dificuldade muito alta para sua implementação. Entretanto ele pode causar um grande estrago, podem trazer prejuízos enormes a uma pessoa, empresa ou nação a um custo muito baixo.

1.2 Justificativa

No dia 28 de fevereiro de 2018, o serviço do Github ficou inoperante das 17:21 às 17:26 UTC e intermitentemente indisponível das 17:26 às 17:30 UTC devido a um ataque de negação de serviço distribuído que alcançou um pico de 1.35Tbps, enviando 126.9 milhões de pacotes por segundo. O serviço disponibilizado por eles é utilizado no mundo inteiro por várias pessoas e empresas, com sua disponibilidade sendo de crítica importância para seus usuários [3].

Quatro dias depois, foi registrada pela Arbor Network a mitigação de um outro ataque a uma empresa norte-americana não divulgada que chegou a um pico de 1.7Tbps, sendo o maior ataque de negação de serviço registrado até hoje [4] [5].

Esses dois incidentes utilizaram servidores Memcached expostos a *internet* como refletores/amplificadores para efetuar o ataque e nos mostram o estrago que tal ataque pode fazer, com muitas vezes o atacante pedindo um pagamento para cessar o ataque.

1.3 Objetivos

O objetivo da monografia é desenvolver uma ferramenta que sirva como uma plataforma para a implementação de ataques de negação de serviço para viabilizar uma análise do Memcached como um refletor em um ataque de negação de serviço.

A ferramenta desenvolvida irá facilitar a implementação de futuros ataques, evitando o retrabalho na implementação de um injetor de pacotes e de uma Command-line user interface (CLI). O desenvolvedor irá se preocupar em desenvolver apenas o mirror para o seu ataque e fazer pequenas alterações no código da ferramenta para incluí-lo.

Vale ressaltar que esse trabalho possui apenas fins acadêmicos, com o autor e seu orientador não se responsabilizando por qualquer uso inapropriado da ferramenta cujo código se encontra sob guarda do autor e seu orientador.

1.4 Organização do texto

Esse trabalho está organizado na seguinte forma:

- O capítulo 2 aborda o fundamento teórico necessário para o entendimento do trabalho;

- O capítulo 3 apresenta a ferramenta desenvolvida;
- O capítulo 4 apresenta os dados obtidos em laboratório e a análise desses dados;
- O capítulo 5 apresenta a conclusão e as propostas para trabalhos futuros.

Capítulo 2

Referencial Teórico

Nesse capítulo iremos apresentar os conceitos fundamentais para entender um ataque de negação de serviço, a aplicação Memcached e seus protocolos. Além disso, também será apresentado como efetuar um ataque de reflexão amplificada utilizando o Memcached.

2.1 Ataque de negação de serviço

Um ataque de negação de serviço (Denial of Service (DoS)) tem como objetivo principal interromper o acesso de usuários legítimos a serviços ou recursos compartilhados. Esses ataques podem ocorrer em uma variada gama de contextos, de sistemas operacionais a serviços baseados em rede [6].

Peng et al.[6] divide os os ataques DoS em dois tipos. O primeiro compreende os ataques que visam derrubar o sistema da vítima, enquanto o segundo tipo inclui os ataques que utilizam um elevado fluxo de tráfego na vítima, visando ocupar todos seus recursos.

Entretanto, Specht et al.[7] apresenta uma classificação mais detalhada para os ataques DoS, como apresentada na Figura 2.1. Ele divide em dois grupos: esgotamento de largura de banda e esgotamento de recurso. A primeira categoria engloba os ataques que visam saturar a largura de banda da vítima com um tráfego não desejado impedindo assim o tráfego legítimo chegue ao *host*. Essa categoria é dividida em duas subcategorias: direto e refletido/amplificado. A categoria esgotamento de recurso tem o objetivo de esgotar ou encerrar os recursos computacionais da vítima, fazendo com que o *host* vítima não consiga processar requisições para usuários legítimos. É dividida em duas subcategorias: ataque por exploração de protocolo e vulnerabilidade. Na seção 2.2 iremos detalhar melhor essa classificação.

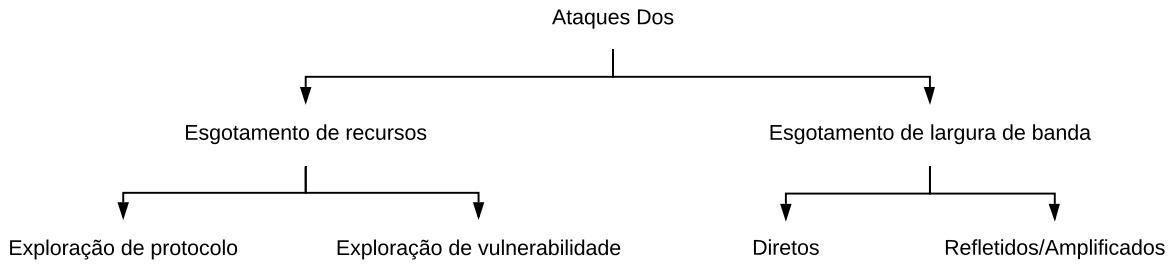

Figura 2.1: Classificação dos ataques de negação de serviço

2.2 Classificação dos ataques DoS

2.2.1 Esgotamento de recurso

No ataque de negação de serviço por abuso de protocolo, o atacante utiliza uma falha ou característica do protocolo para lançar o ataque. Specht et al.[7] divide esse tipo de ataque em exploração de protocolo e exploração de vulnerabilidade.

Exploração de protocolo

Quando o atacante utiliza essa abordagem, ele tem como objetivo utilizar uma característica do protocolo para exaurir os recursos do alvo.

Podemos utilizar como exemplo o ataque TCP SYN *flooding*. Nesse ataque o atacante explora o *three-way handshake* do protocolo TCP, enviando vários pedidos SYN para a vítima com o endereço de IP de origem spofado, ou seja, um IP que não corresponde ao IP da fonte. A vítima irá alocar os recursos necessários para estabelecer a conexão TCP, enviará um pacote SYN ACK e irá esperar por um pacote ACK. Como o IP do pacote SYN foi spoofado, não teremos a conclusão do *three-way handshake*, fazendo com que o recurso fique alocado por um longo período. Se o atacante conseguir enviar um elevado numero de pacotes TCP SYN, alocando assim um elevado número de conexões, ele pode conseguir exaurir os recursos do alvo efetuando assim o ataque de negação de serviço SYN flooding [8].

Apesar desse ataque enviar um elevado número de pacotes, que é a característica dos ataques de esgotamento de largura de banda, o seu objetivo principal é exaurir os recursos do alvo. Para não causarmos alguma confusão, iremos apresentar o ataque R-U-Dead-Yet (RUDY), que explora o protocolo HTTP.

O RUDY explora uma fraqueza do protocolo HTTP o qual foi originalmente desenhado para prover serviços para usuários com um baixo tráfego de rede . Ele é utilizado do fato que a operação de HTTP POST permite que a conexão fique aberta por um tempo indeterminado no caso onde os dados chegam a uma taxa muito lenta. O ataque

começa com o envio de um pacote legítimo de requisição HTTP POST com campo *content length* possuindo um valor absurdamente grande. Após esse passo inicial, o atacante deve começar a injetar pequenos pedaços do dado a uma taxa muito baixa. Isso fará com que o servidor aloque uma grande quantidade de recursos por causa do campo *content length* ser uma valor muito grande por um longo período, já que os dados chegam a uma taxa muito baixa. Se o atacante conseguir alocar uma grande quantidade de recursos da vítima, ele poderá ter um ataque bem sucedido[9].

Exploração de vulnerabilidade

Esse tipo de ataque explora uma falha na implementação de um protocolo ou até mesmo o próprio protocolo. Podemos tomar como exemplo uma falha na implementação de como Windows NT lidava com pacotes Out of Band (OOB), que causava um Blue Screen of Death (BSoD) [10].

Outro exemplo de ataque que se enquadra nessa categoria é o *Ping of death*. Esse ataque explorava uma falha na implementação do protocolo Internet Control Message Protocol (ICMP) em alguns sistemas, que ao receber um pacote ICMP ECHO maior que 65536 bytes causava um buffer overflow, podendo levar a sua reinicialização[11].

2.2.2 Esgotamento de largura

Os ataques de negação de serviço volumétricos têm como objetivo saturar a largura de banda da vítima. Podem ser divididos em duas categorias: diretos e refletidos/amplificados [12]. É indispensável para o atacante utilizar uma arquitetura distribuída para efetuar um ataque bem-sucedido nessa categoria de ataque.

Ataques diretos

Esse tipo de ataque consiste em enviar grande número de pacotes diretamente para a vítima com o objetivo de saturar sua largura de banda. Os ataque UDP flood e ICMP flood são exemplos de ataques de negação de serviço dessa subcategoria. Nesses ataques, deve-se enviar para a vítima uma grande quantidade de pacotes UDP ou ICMP, tendo como objetivo saturar a largura de banda da vítima.

Ataques refletidos/amplificados

Uma forma de o atacante conseguir mascarar a fonte do ataque é utilizando um máquina que consiga retransmitir os pacotes do atacante para a vítima. Essa máquina, que funciona como mais um agente do ataque, é chamada de refletor.

Possíveis refletores são máquinas na rede que utilizam um protocolo que gere um pacote resposta a partir de um estímulo e que possibilite o spoofing dos pacotes.

Uma máquina com o protocolo TCP pode ser explorada como um refletor, por exemplo. Nesse ataque, deve-se enviar pacotes TCP SYN com o endereço spoofado da vítima e enviá-los para o refletor. O refletor irá responder com pacotes SYN-ACK, que serão encaminhados para a vítima. É importante perceber que nesse ataque o refletor será vítima de um ataque TCP SYN *flooding*, então é importante o atacante utilizar vários refletores para não derrubar o refletor e também para não saturar sua largura de banda [13].

Um servidor de Domain Name System (DNS) mal configurado, deixado em sua configuração padrão, ou um servidor de Domain Name System (DNS) público que possui a intenção de disponibilizar seus serviços para a internet são outros exemplos de possíveis refletores. Para se executar o ataque, deve-se enviar requisições DNS para o refletor que parecem ser legítimas; entretanto, elas estarão spoofadas com o endereço da vítima. Ao receber essa requisição, o servidor DNS irá responder para a vítima, já que o pacote recebido possui seu endereço [14].

É importante observar que em ambos os casos os refletores podem gerar repostas maiores que as requisições enviadas pelo atacante, causando uma amplificação do ataque pelo refletor. Essa característica do refletor de também se comportar como um amplificador do ataque é excelente no ponto de vista do atacante. Além de possuir um intermediário que irá trazer uma camada a mais de anonimato para a fonte do ataque, também irá amplificar seu tráfego gerado, requisitando assim menos de sua infraestrutura para um ataque bem-sucedido.

Para medir a eficiência de uma amplificador, o cálculo utilizado é dividir o tamanho do pacote de resposta (*response packet*) pelo tamanho do pacote de requisição (*request packet*), como apresentado no Cáculo 2.1. Esse cálculo nos retorna o fator de amplificação do refletor. Esse fator indica diretamente qual será o impacto de cada refletor no ataque: quanto maior o fator de amplificação melhor é o refletor [14].

$$Amplificação = \frac{tamanho(resposta)}{tamanho(pedido)} \tag{2.1}$$

Voltando para o exemplo do servidor DNS, a partir de uma pacote de requisição de 60 bytes podemos conseguir gerar um pacote resposta de até 4000 bytes, gerando uma fator de amplificação de 66 [14].

Outro exemplo de refletor com um alto fator de amplificação são servidor Memcached mal configurados expostas a internet. O estudo do Memcached como refletor é o foco do estudo desse trabalho e será abordado na seção 2.4.

2.3 Ataque de negação de serviço distribuído

Devido ao fato de as atuais infraestruturas possuírem uma alta taxa de largura de banda e um grande poder de processamento, ficou inviável para o atacante exaurir os recursos de um determinado alvo com uma única máquina. Para contornar essa barreira, os atacantes passaram a utilizar mais fontes de ataque para alcançar um maior poder computacional.

Essas fontes de ataque geralmente são máquinas comprometidas que passam a ser controladas pelo atacante. Para utilizar essas máquinas como fonte, o atacante deve primeiro comprometê-las com um *malware* para ter persistência no acesso à máquina comprometida, que passará a ser um bot. Esses bots serão coordenados pelo atacante para lançar o ataque ao mesmo tempo, como apresentado na Figura 2.2. Esse tipo de ataque é chamado de Ataque de Negação de Serviço Distribuído (Distribuited Denial of Service (DDoS)) [15].

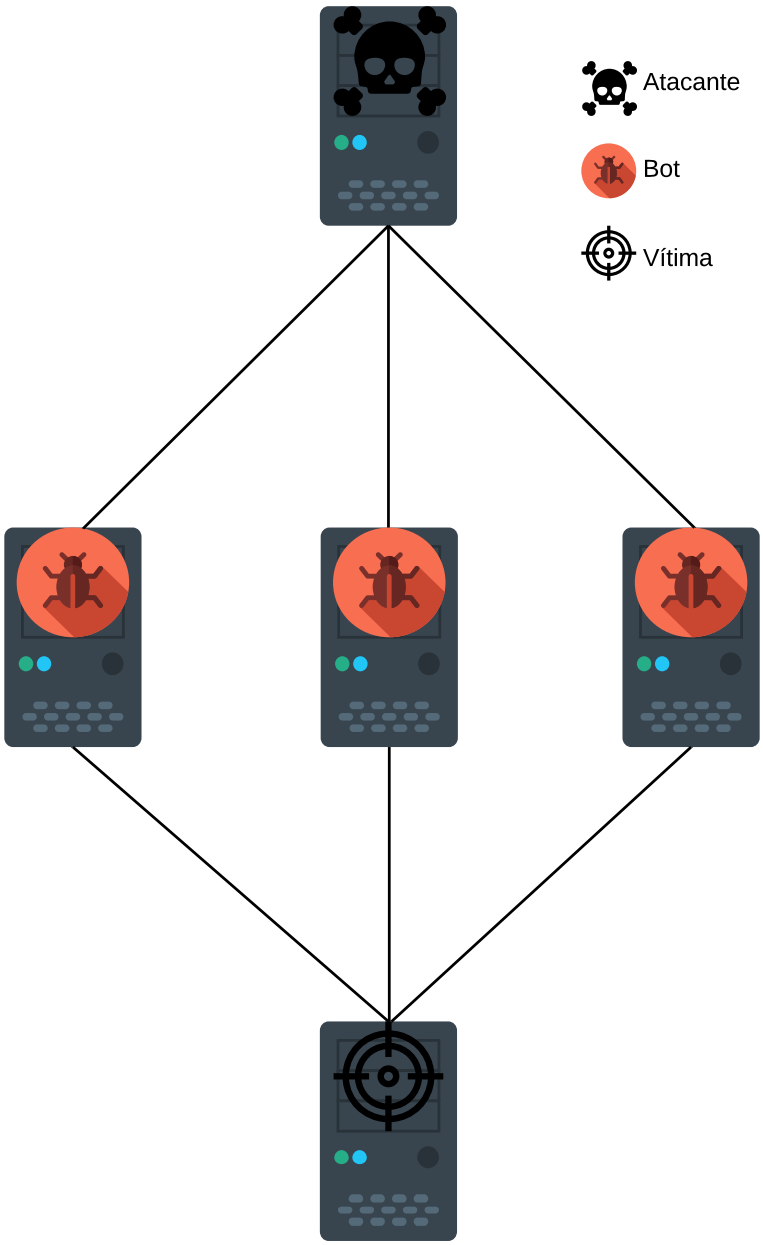

Figura 2.2: Ataque de negação de serviço distribuído

Quanto mais bots se têm, mais difícil será coordená-los no ataque. Para resolver esse problema, algumas arquiteturas surgiram, dando origem às botnets. Alomari etal.[9] apresenta 3 arquiteturas de Comando e Controle (C&C) para uma botnet: Agente-Executor, Internet Relay Chat (IRC) e Web-based.

Na arquitetura Agente-Executor (Figura 2.3), os bots são divididos em dois grupos. Os agentes que irão fazer o ataque e os executores que são responsáveis por coordenar os agentes. Para iniciar o ataque, o atacante necessita apenas enviar os dados necessárias para os executores coordenarem o ataque a uma determinada vítima em uma determinada

hora. Nessa arquitetura, também é possível ter uma rede em que os agentes podem se comunicar com qualquer um dos executores, dando uma maior resiliência à arquitetura.

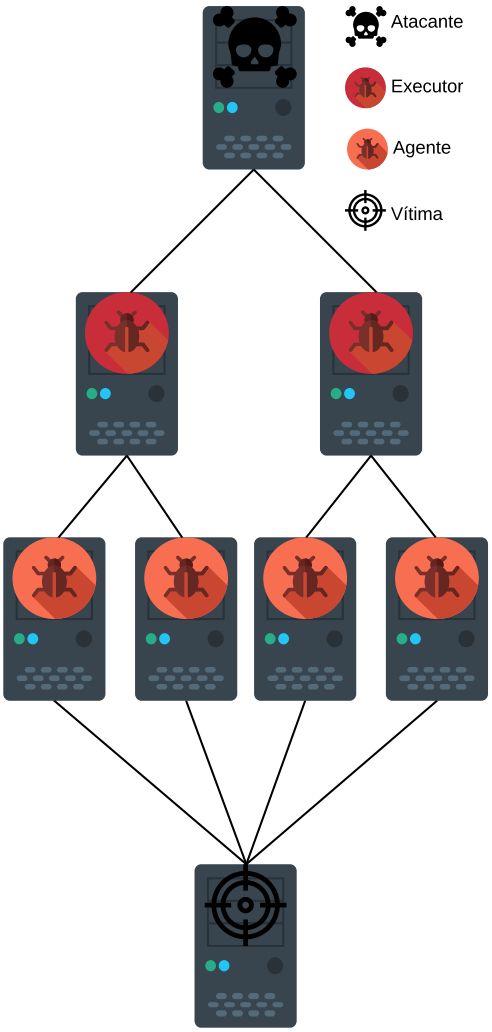

Figura 2.3: Arquitetura Agente-Executor

A arquitetura IRC tem um canal IRC como a ligação do atacante aos executores (Figura 2.4). A vantagem dessa arquitetura em relação a Agente-Executor é que o atacante utiliza um tráfego legítimo, utilizando as portas servidor IRC para iniciar o ataque. Além disso, os servidores IRC costumam ter um elevado tráfego, dando um maior anonimato para o atacante. Outra vantagem é que o sistema não precisa armazenar a lista de escravos, já que ele tem esse log no IRC. O atacante pode obter por utilizar vários servidores IRC para uma maior resiliência da botnet.

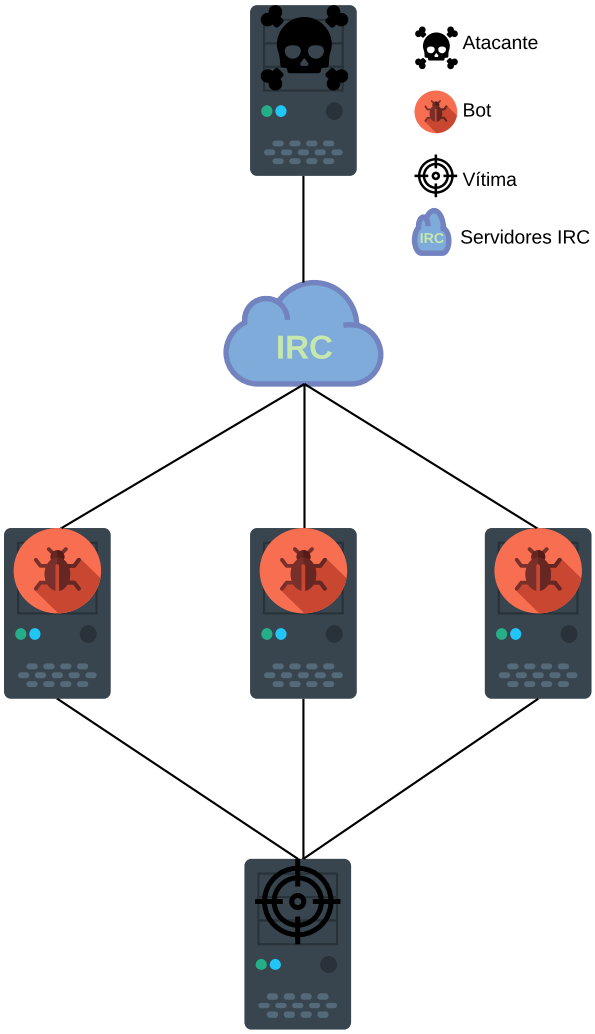

Figura 2.4: Arquitetura IRC

O modelo Web-based é bem parecido com o IRC, a diferença é que ele utiliza uma aplicação web para a comunicação do atacante com os bots. Essa arquitetura possui algumas vantagens em relação a IRC, como uma maior flexibilidade para a implementação de scripts mais complexos, o uso do HTTP/HTTPS e um uso mais simplificado.

2.4 Ataque de reflexão amplificado explorando Memcached

O Mencached é um sistema distribuído de caching em memória de chave-valor. Sua proposta veio para resolver o problema da dificuldade de escalar um banco de dados devido as garantias para manter às propriedades Atomicity, Consistency, Isolation, Durability (ACID). Ele gerencia o caching dos resultados de chamadas de banco de dados,

APIs ou qualquer outro dado e é amplamente utilizado para armazenar dados que são frequentemente requisitados, diminuindo assim as requisições ao banco de dados.

Exitem dois protocolos que podemos utilizar para a comunicação cliente servidor no Memcached, são eles: protocolo de texto e protocolo binário. A seguir, iremos explicar esses dois protocolos, focando nos comandos SET/GET e STATS que são os utilizados para efetuar o ataque de reflexão amplificada Memcached.

2.4.1 Protocolo de texto

Nesse protocolo, todo comando começa com o nome do comando que se deseja executar, seguido pelos argumentos, se necessário, separados pelo carácter espaço. Todo comando é case-sensitive, em caixa baixa e deve terminar com os caracteres \r\n [16]. Os argumentos obrigatórios de uma comando estão entre os caracteres '< >' e os argumentos opcionais estão entre colchetes '[ ]'.

Armazenado um dado

Para armazenar um conjunto chave-valor no servidor Memcached, utilizaremos o comando 'set'. Esse comando possui a seguinte forma:

```
set <key> <flags> <exptime> <bytes> [noreply]\r\n<data block>\r\n
```

- key: a chave que será correspondente ao valor armazenado;
- flags: Valor de 16 bits que o servidor irá armazenar junto com o dado armazenado
- exptime: tempo que o dado ficará armazenado no servidor. 0 significa que o dado não expira;
- bytes: número de bytes no campo data block;
- noreply: parâmetro opcional que instrui o servidor a não enviar um resposta ao cliente;
- data block: dado que será armazenado no servidor memcached.

Caso o noreplay seja omitido no comando set, a resposta do servidor para o cliente será uma das seguintes:

- STORED\r\n: indica sucesso;
- NOT_STORED\r\n: dado não foi armazenado.

Requisitando um dado

Para requisitar um dado do servidor Memcached, devemos utilizar os comandos 'get' ou 'gets'. Esses comandos possuem a seguinte forma:

```
get <key>\r\n
gets <key>\r\n
```

O argumento key corresponde a uma ou mais chaves separadas pelo carácter espaço.

A resposta do servidor segue a seguinte estrutura:

```
VALUE <key> <flags> <bytes> [<cas unique>]\r\n<data block>\r\n
```

- key: a chave armazenada;
- flags: flag armazenada;
- bytes: número de bytes no campo data block;
- cas unique: inteiro de 64 bits que identifica o conjunto chave-valor;
- data block: dado armazenado no servidor memcached.

Caso o servidor não retorne nenhuma resposta, significa que o dado não foi armazenado.

Estatísticas do servidor Memcached

Para requisitar estatísticas gerais e a configuração do servidor Memcached, podemos utilizar o comando stats. Esses comandos possuem a seguinte forma:

```
stats\r\n
```

Utilizando o protocolo UDP

Ao utilizar o protocolo UDP, cada comando possuirá um cabeçalho de 8 bytes como apresentado na Figura 2.5. Todo *request* deve conter apenas um datagrama, mas a resposta pode conter um ou mais datagramas.

Byte	0	1
0	Request ID	
2	Sequence number	
4	Número total de datagramas	
6	Reserved	

Figura 2.5: Cabeçalho UDP protocolo de texto

O campo Request ID deve ser fornecido pelo cliente. Normalmente, será um valor crescente monotonicamente a partir de uma semente aleatória, mas o cliente é livre para usar qualquer valor. A resposta do servidor irá conter o mesmo Request ID escolhido pelo cliente. O cliente utiliza esse campo para identificar respostas que possam ser de pedidos pendentes. Quaisquer datagrama com um Request ID desconhecido é provavelmente uma resposta atrasada a uma solicitação anterior e deve ser descartada.

O valor do campo Sequence number varia de 0 a n-1, onde n é o número total de datagramas na mensagem. O cliente deve concatenar os payloads dos datagramas para uma dada resposta em ordem numérica de sequência. A resposta completa terá o mesmo formato de uma resposta enviada com o protocolo TCP(incluindo terminação \r\n sequências).

O campo Reserved é reservado para uso futuro e deve ter o valor 0.

2.4.2 Protocolo binário

Os pacotes binários Memcached possuem um campo obrigatório que é o cabeçalho e mais três campos que são opcionais , são eles Command-specific extras, Key, e Value. Como apresentado na Figura 2.6. [17]

Campo	Tamanho
Cabeçalho (Obrigatório)	24 bytes
Command-scpecific extras (Opcional)	Tamanho no campo 'Extra length' do Cabeçalho.
Key (Opcional)	Tamanho no campo 'Key length' do Cabeçalho.
Value (Opcional)	Total body length - (Extra length + Key length)

Figura 2.6: Estrutura geral do pacote Memcached

Os comandos possuem dois tipos de cabeçalho para os pacotes, um para o requisições (pacote *request*) Figura 2.7, e outro para respostas (pacote *response*) Figura 2.8. Os dois possuem nove campos com a única diferença entre eles é a existência do campo vbucket id ou Status.

Cabeçalho request

Byte	0	1	2	3
0	Magic	Opcode	Key length	
4	Extras length	Data type	vbuvket id	
8	Total body length			
12	Opaque			
16	CAS			
20				

Figura 2.7: Estrutura do pacote *request* Memcached

Cabeçalho response

Byte	0	1	2	3
0	Magic	Opcode	Key length	
4	Extras length	Data type	Status	
8	Total body length			
12	Opaque			
16	CAS			
20				

Figura 2.8: Estrutura do pacote *response* Memcached

- Magic: identificação da versão do protocolo. Para cada versão do protocolo, é usado um valor para os pacote request o outro para o response;
- Opcode: identificação do comando a ser executado (Figura 2.9);
- Key length: tamanho em bytes do campo Key;
- Extras length: tamanho em bytes do campo Command-specific extras;
- Data type: reservado para uso futuro.
- vbucket id: o bucket virtual para o comando;
- Status: status da resposta, diferente de zero se ocorrer um erro (Figura 2.11);
- Total body length: tamanho em bytes da soma dos tamanhos dos campos Extras, Key e Value;
- Opaque: Dado que será copiado de volta na resposta;

- CAS: Data version check.

Na Figura 2.9 apresentamos os Opcodes dos comandos Get, Set e Stat. O comando Get busca por uma chave e retorna o valor, possui campo opcional Key para o *request* e para o *response* possui Key e Value. O comando Set armazena um dado (chave-valor) no servidor, possui campo opcional key, value e extra. E o comando Stat retorna os dados de status do servidor, não possui campo opcional.

Opcodes

Byte	Descrição
0x0000	Get
0x0001	Set
0x0010	Stat

Figura 2.9: Códigos campo Opcode (apenas os utilizados no ataque)

Response status

Byte	Descrição
0x0000	Sem erro
0x0001	Chave não encontrada
0x0002	Chave existe
0x0003	Valor muito longo
0x0004	Argumentos invalidos
0x0005	Item não armazenado
0x0006	Incre/Decr de um valor não numérico
0x0007	Vbucket pertence a outro servidor
0x0008	Erro na autenticação
0x0009	Autenticação continua
0x0081	Comando desconhecido
0x0082	Falta de memória
0x0083	Não suportado
0x0084	Erro interno
0x0085	Ocupado
0x0086	Falha temporária

Figura 2.10: Códigos do campo Status

Toda comunicação é iniciada por um pacote de *request* enviado pelo cliente, e o servidor irá responder com 0 ou múltiplos pacotes para cada pedido. Se o campo Status de um pacote *response* é diferente de 0, o corpo do pacote irá conter a mensagem de erro. Se for zero o campo Opcode irá definir o layout da estrutura do pacote.

Efetuando o ataque

O Memcached não possui controle de acesso, não sendo desenhado para operar com suas portas expostas para a internet. Porém, algumas empresas vêm utilizando o Memcached com suas portas abertas para a internet sem os devidos cuidados básicos para evitar o uso de seus servidores como refletores.

Para ser possível utilizar um servidor Memcached como refletor/amplificador é necessário que o servidor Memcached tenha o protocolo UDP habilitado e exposto a internet para qualquer endereço da rede.

Existem duas formas de explorar o servidor como refletor/amplificador. A primeira, e mais simples, é forjar um pacote Stat *request* Memcached com o endereço da vítima no campo de endereço da fonte do pacote IP. O servidor irá responder para a vítima com um pacote Stat *response*.

O segundo método exige um certo preparo do servidor antes de iniciar o ataque. O atacante deve saber qual o valor máximo permitido para os valores armazenados. Para obter esse dado, o atacante deve utilizar pacotes Stat para encontrar um tamanho máximo que podemos armazenar no valor. Após escolher os melhores refletores, o atacante deve armazenar os dados de chave-valor em cada refletor, respeitando os limites encontrados no passo anterior para ter certeza que o dado foi armazenado. Como o dado de chave-valor armazenado o ataque pode começar, para isso o atacante deve enviar pacotes Get *request* forjados com o endereço IP da vítima. Isso implicará que o servidor Memcached irá responder com um pacote Get *response* para a vítima.

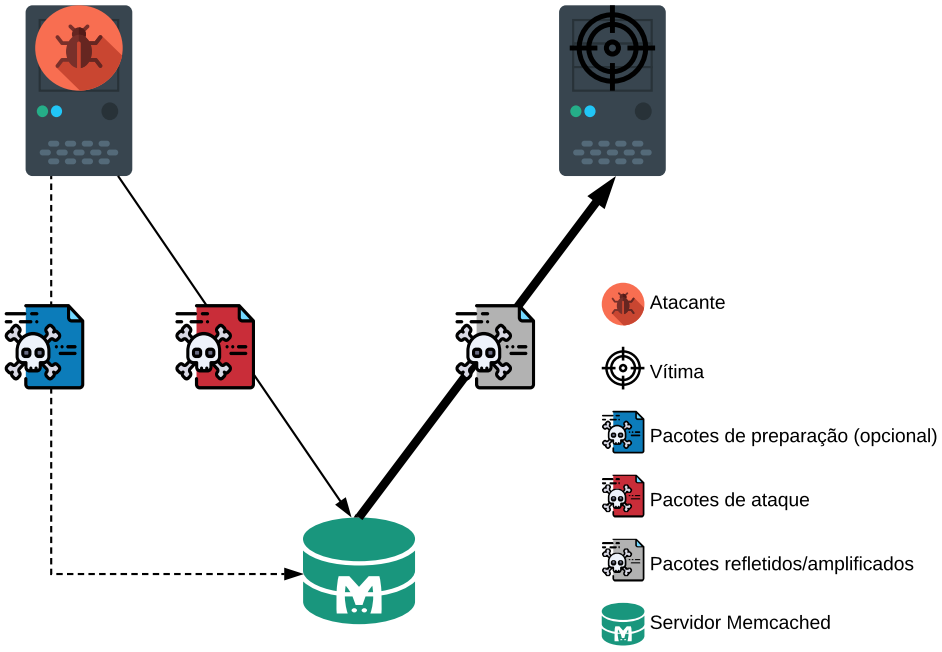

Figura 2.11: Uso do refletor Memcached

Para calcular a eficiência do ataque, iremos utilizar a razão do tamanho do pacote de resposta pelo de requisição, como apresentado na equação 2.2. Esse valor nos dará a amplificação do ataque e quanto maior for essa valor mais eficiente é o ataque. Iremos considerar nos cálculos os cabeçalhos IP e UDP já que eles fazem parte do pacote que será enviado na rede e existiu um esforço computacional para gerá-los.

$$Amplificação = \frac{Resposta}{Requisição} \tag{2.2}$$

Para calcular o fator de amplificação do método GET/SET, iremos considerar a fase do uso do pacote SET como uma preparação. O cálculo do fator de amplificação utilizará apenas os pacotes GET. Assim, temos a equação 2.3 como o fator de amplificação do segundo método.

$$Amplificação = \frac{Ip + Udp + Header + Key + Value}{Ip + Udp + Header + Payload} = 1 + \frac{Value}{Ip + Udp + Header + Payload} \tag{2.3}$$

O ataque de negação de serviço utilizando Memcached tem um grande potencial para disponibilizar uma alta amplificação. No capítulo 4 iremos apresentar os testes e análise do uso de um servidor Memcached em um ataque DoS.

Capítulo 3

Linderhof

Nesse capítulo será apresentado o Linderhof, que foi a ferramenta implementada para efetuar o ataque. Iremos detalhar a arquitetura, funcionalidades e como adicionar novos ataques ao Linderhof. A ferramenta tem o objetivo de facilitar o desenvolvimento de ataques DoS refletidos.

3.1 Arquitetura

A arquitetura do Linderhof foi projetada de forma modular. Possui três módulos como mostra a Figura 3.1, Oryx (interface), Commander e Netuno(injetor de pacotes). Cada *engine* periférica do Linderhof não impacta o módulo Commander, sendo possível utilizar qualquer outra interface e injetor.

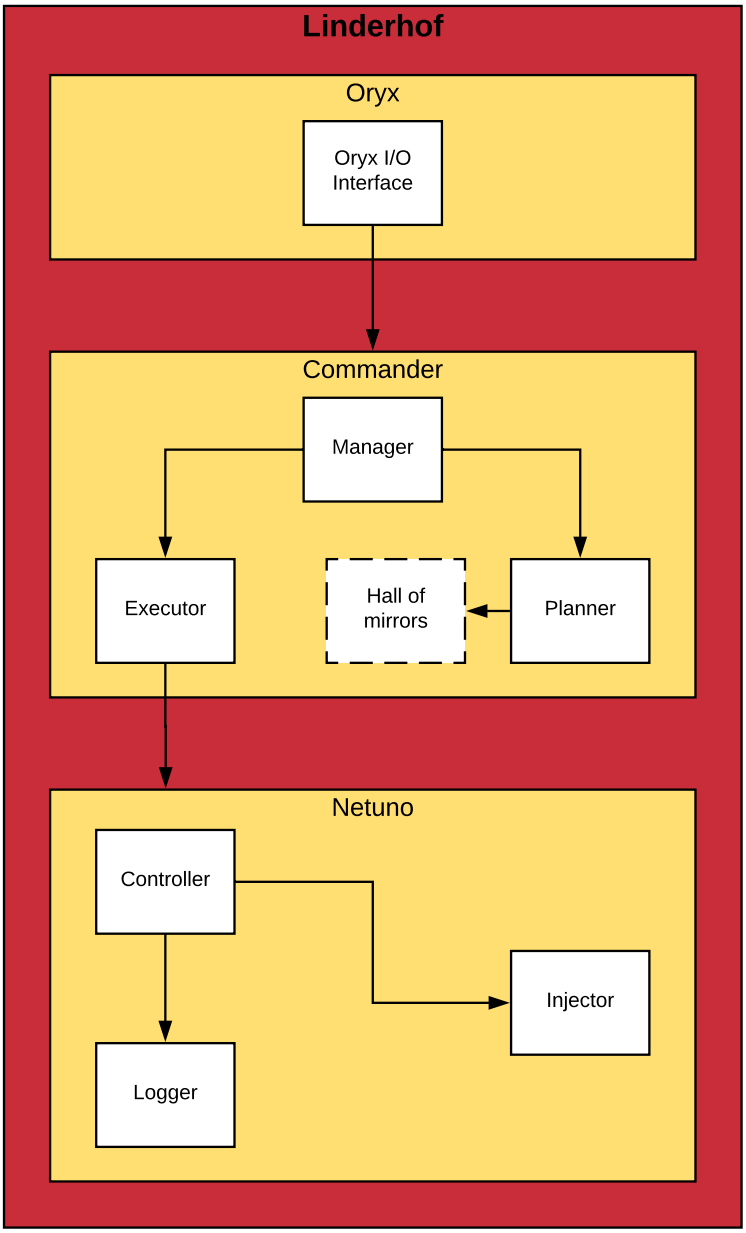

Figura 3.1: Arquitetura Linderhof

3.1.1 Módulo Oryx

O módulo Oryx é a *engine* de interface de usuário do Linderhof. Seu trabalho principal é criar a estrutura de *draft* do ataque que será executada. Os argumentos que podem ser passados para a interface e são apresentados na Figura 3.2. Ao terminar de montar a estrutura *draft*, o Módulo Oryx o encaminha para o Módulo Commander.

Nome	Opção longa	Opção curta	Opção obrigatória	Argumento obrigatório	Valor padrão	Descrição
ARG_MIRROR	mirror	m	Sim	Sim	Não possui	Espelho que será utilizado no ataque
ARG_TARGETIP	target	t	Sim	Sim	Não possui	IP do alvo
ARG_AMPLIFIERIP	amplifier	a	Sim	Sim	Não possui	IP do amplificador
ARG_TARGPORT	targport	g	Não	Sim	80	Porta do alvo
ARG_AMPPORT	amport	p	Não	Sim	Porta padrão do mirror	Porta do amplificador
ARG_LEVEL	level	l	Não	Sim	1	Level do ataque
ARG_TIMER	timer	c	Não	Sim	Infinito	Tempo do ataque
ARG_LOGFILE	log	f	Não	Sim	NULL	Nome do arquivo de log
ARG_INCREMENT	inc	i	Não	Sim	FALSO	Ataque incremental

Figura 3.2: Opções da interface Oryx

3.1.2 Módulo Commander

O módulo Commander é o núcleo da ferramenta. Ele é responsável por criar o planejamento do ataque de acordo com *draft* passado pelo módulo superior e chamar o injetor Netuno com os argumentos necessários para executar o ataque. Esse módulo é subdividido em 4 submódulos: Manager, Planner, Executor e Hall of mirrors.

O Hall of mirrors contém as funções de chamada para cada ataque, chamamos essas funções de mirror. Os mirrors têm a finalidade de fazer toda a preparação para o ataque e chamar o injetor para executar o ataque quando tudo estiver preparado. Essa preparação consiste em criar todos os pacotes para o injetor e fazer toda a preparação dos refletores antes do ataque.

O módulo Manager gerencia o Planner e o Executor, funcionando como uma ponte entre eles. Além disso, ele tem a responsabilidade de validar o *draft* recebido e fazer a inicialização da ferramenta como setar os *handler* de sinal, adicionar as Linux *capabilities* necessárias e funções de error.

O Planner é responsável por montar a estrutura de plano de ataque de acordo com o mirror desejado. Essa estrutura contém o tipo do mirror, a função de chamada do mirror, e os dados necessário para executar a função de chamada do ataque.

O Executor tem a única responsabilidade de executar o mirror.

3.1.3 Módulo Netuno

O módulo Netuno é a engine de injeção do Linderhof. Ele foi implementado para conseguir uma injeção de pacotes constante no ataque. É dividido em três submódulos, Controller, Injector e Logger.

O Injector tem a responsabilidade de criar e destruir as *threads* injetoras. Cada injetor possui um bucket que corresponde à quantidade de pacotes que ele deve enviar. O Controller é responsável por controlar a taxa de injeção de cada injetor através da atualização do bucket dos injetores de acordo com o intensidade do ataque desejada e encerrar o ataque de acordo com o timer passado pelo usuário. O Netuno tem a opção de executar um ataque incremental, nesse tipo de ataque ele irá aumentar o nível a uma frequência que foi passado pelo usuário. O Logger faz apenas o log do ataque em execução, que pode ser apresentado no terminal ou salvo em um arquivo.

A intensidade do ataque corresponde à quantidade de pacotes que o Netuno deve enviar por segundo, e essa parâmetro é passado através do argumento nível do Oryx. A quantidade de pacotes por segundo respeita a seguinte fórmula:

$$10^{L-1} \tag{3.1}$$

Onde L corresponde à intensidade do ataque deseja. Resultando na seguinte tabela:

Tabela 3.1: Intensidade por pacotes/segundo

Level	**Pacotes/segundo**
1	1
2	10
3	100
4	1000
5	10000
6	100000
7	1000000
8	10000000
9	100000000
10	1000000000

É importante ressaltar que esse parâmetro indica o fluxo de pacotes desejado no ataque. Isso não significa que a máquina atacante conseguirá gerar a quantidade de pacotes desejada e nem que a quantidade de pacotes gerados chegará ao amplificador.

3.2 Adicionando um novo mirror

Como dito anteriormente, um mirror é responsável por fazer toda a preparação do ataque, como forjar os pacotes para o injetor e fazer a preparação do amplificador para o ataque. Após finalizar a preparação, ele deve chamar o injetor para iniciar o ataque.

Para adicionar um novo mirror ao Linderhof, devemos fazer algumas pequenas alterações nos módulos Oryx e Controller. A seguir, iremos apresentar quais são essa alterações.

1. **Criar função de chamada do mirror**

   A função de chamada do mirror deve seguir o protótipo a seguir:

   ```
   int ExecuteMirrorNameMirror( void * p_arg );
   ```

   Sendo `p_arg` os argumentos necessários para executar o mirror.

   Os arquivos do mirror devem ficar na pasta `src/linderhof/hom/nome_do_mirror`.

   A função de chamada será declarada em `src/linderhof/hom/nome_do_mirror.h`.

2. **Adicionar mirror ao Linderhof planner**

   (a) Adicione seu mirror no enum MirrorType em src/include/venus.h

   (b) Atualize o switch-case da função Planner para construir o LhfPlan do seu mirror. O ponteiro atkData deve conter os argumentos que serão utilizados pelo mirror.

3. **Adicionar mirror a engine UI Oryx**

   Na função src/interface/interface.c:parserAttackOpt adicione a construção do draft do mirror.

3.3 Considerações finais

Nesse capítulo foi apresentado o *design*, funcionamento do Linderhof e como adicionar novos mirrors a ele. De modo geral, a ferramenta tem o objetivo para facilitar o desenvolvimento de ataques DoS refletidos, retirando o retrabalho de implementação dos módulos de injeção e interface.

No capítulo 4 utilizaremos o Linderhof no papel de atacante para viabilizar a simulação de uma ataque DoS refletido Memcached em laboratório.

Capítulo 4

Ambiente experimental e resultados

Nesse capítulo iremos apresentar os dados coletados em laboratório junto com a análise que obtivemos do comportamento do servidor Memcached no ataque. De forma geral, os testes seguiram a metodologia descrita em Gondim et al. [18] na avaliação de ataques DDoS.

4.1 Coleta de dados

Todas as capturas foram feitas em um ambiente controlado, utilizando o analisador de protocolos de rede TShark para fazer a captura e análise do tráfego de rede. A configuração das máquinas é apresentada no Tabela 4.1. O roteador utilizado foi o ASUS RT-AC1900 de 1Gbit/s e a versão do Memcached utilizada foi a 1.5.10 .

Cada nível do ataque tem uma duração de 5 segundos. Para obter o valores de bytes e pacotes de um nível específico foi feita a média aritmética de 5 em 5 segundos da estatística retornado pelo TShark de cada captura.

Na tabela 4.1 apresentamos as máquinas utilizadas em laboratório com suas respectivas configurações.

Tabela 4.1: Configuração das máquinas utilizadas

	Processador	Ram	Placa de rede	SO
Sputnik	Intel Core i5-6200U	12G	1Gbit/s	Manjaro Linux 18
Spitfire	AMD Ryzen 7 1700	16G	1Gbit/s	Manjaro Linux 18
Orion	Intel Core i7-6700	16G	1Gbit/s	Manjaro Linux 18

Para termos uma melhor análise do comportamento do Memcached no ataque e para minimizar o residual que a diferença de hardware pode deixar na análise, optamos por

fazer o rodízio das máquinas entre atacante, amplificador e vítima, resultando em seis possíveis configurações.

Como a maioria dos servidores Memcached que estão expostos na internet estão com a configuração padrão, optamos por fazer as mínimas alterações possíveis no amplificador para o teste. A única alteração feita foi a abertura da porta 11211 no protocolo UDP, já que na última versão do Memcached essa porta não vem mais aberta por padrão.

Para cada configuração de ataque, iremos apresentar os dados coletados para os dois ataques possíveis no Memcached o STATS e o GET/SET, utilizando o protocolo de texto Memcached. Para cada ataque são apresentados dois gráficos: um para os bytes e outro para os pacotes que cada máquina enviou e recebeu e outro gráfico com amplificação para cada nível de ataque. A seguir, são apresentadas essas tabelas para cada configuração.

Configuração 1

Essa configuração se caracteriza por ter o refletor e vítima mais fortes e a atacante mais fraco.

No método GetSet, tivemos um máximo de 49.01 bytes/pacote chegando no amplificador no nível 4 e um máximo de 1426.86 bytes/pacote saindo do amplificador no nível 1 como apresentado na tabela 4.5, apesar de o atacante saturar apenas no nível 7 como mostra os gráficos 4.1 e 4.2.

Já para o método Stats, tivemos um máximo de 46.03 bytes/pacote chegando no amplificador no nível 4 e um máximo de 989.04 bytes/pacote saindo do amplificador também no nível 4 como apresentado na tabela 4.8, apesar de o atacante saturar apenas no nível 7 como mostra os gráficos 4.3 e 4.4.

Tabela 4.2: Configuração dos ataques

Config	Sputnik	Spitfire	Orion
1	Atacante	Refletor	Vítima
2	Vítima	Atacante	Refletor
3	Refletor	Vítima	Atacante
4	Atacante	Vítima	Refletor
5	Refletor	Atacante	Vítima
6	Vítima	Refletor	Atacante

GetSet

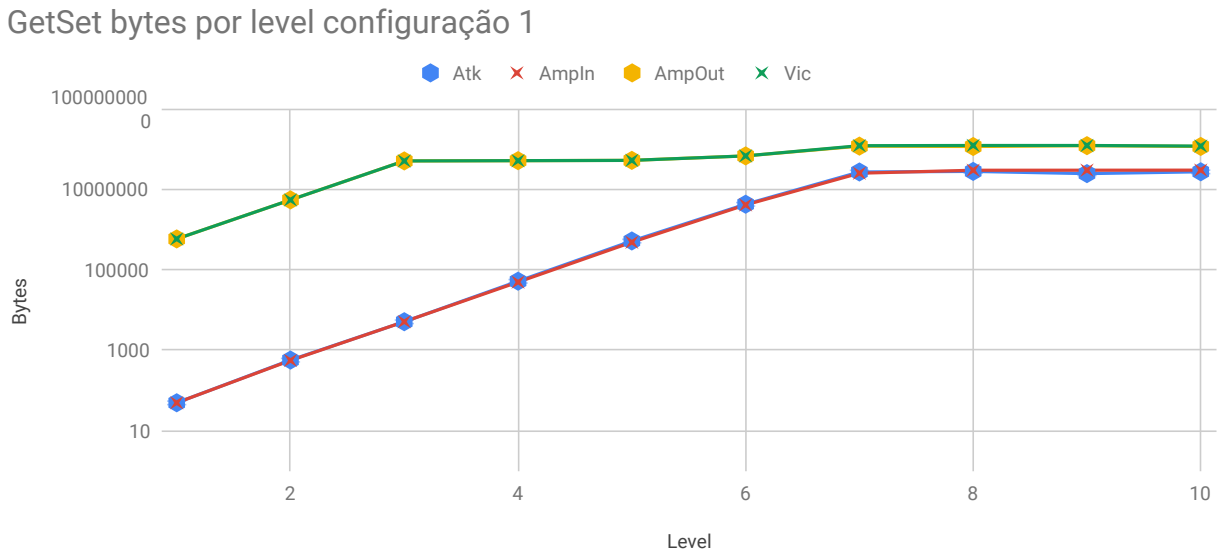

Figura 4.1: GetSet bytes por level configuração 1

Tabela 4.3: Quantidade de bytes para cada nível no método GetSet configuração 1

Nível	Atk	AmpIn	AmpOut	Vic
1	49	49	575026	575311
2	450	441	5369840	5368412
3	3959	3969	49996525	49997667
4	40180	38180	50777571	50777571
5	401800	373989	51638208	51638239
6	3295426	3171877	66699352	66699352
7	20861289	19453764	116760067	119557012
8	21567036	23041201	114175952	121100734
9	19000004	23097022	119827783	121046858
10	21093539	23181625	114683043	116953396

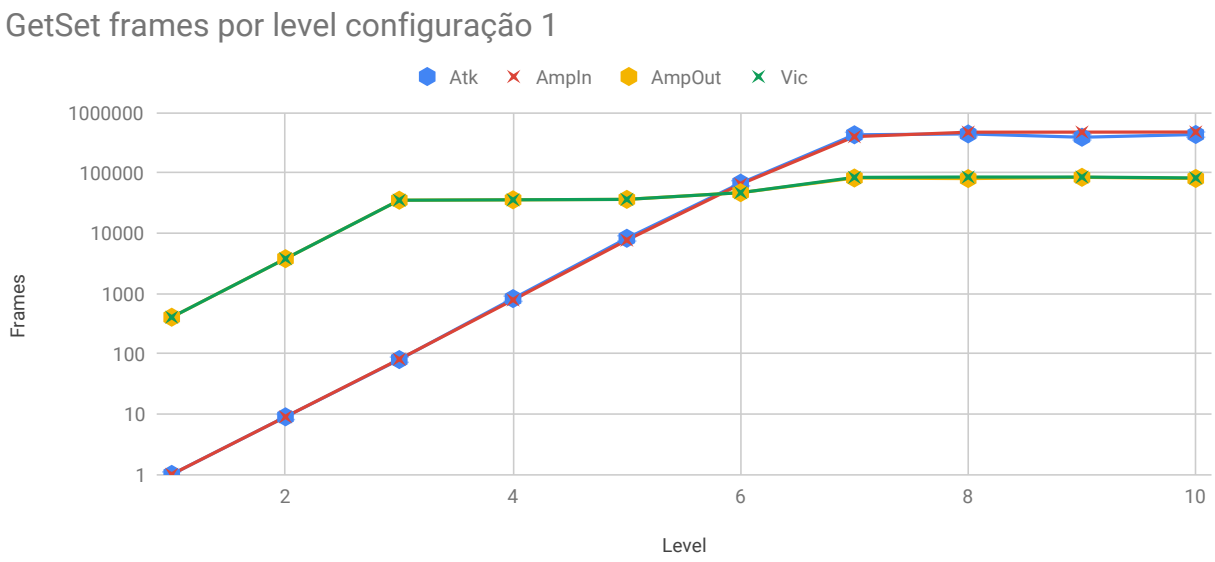

Figura 4.2: GetSet frames por level configuração 1

Tabela 4.4: Quantidade de pacotes para cada nível no método GetSet configuração 1

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	403	403
2	9	9	3763	3762
3	80	81	35041	35042
4	820	779	35588	35588
5	8200	7632	36192	36192
6	67253	64732	46748	46748
7	425740	397015	81834	83794
8	440143	470228	80023	84876
9	387755	471367	83984	84839
10	430480	473094	80377	81969

Tabela 4.5: Razão de bytes por pacotes para cada nível no método GetSet configuração 1

Nível	Input	Output
1	49.0	1426.86
22 2	49.0	1427.01
3	49.0	1426.8
4	49.01	1426.82
5	49.0	1426.79
6	49.0	1426.79
7	49.0	1426.79
8	49.0	1426.79
9	49.0	1426.79
10	49.0	1426.81

Stats

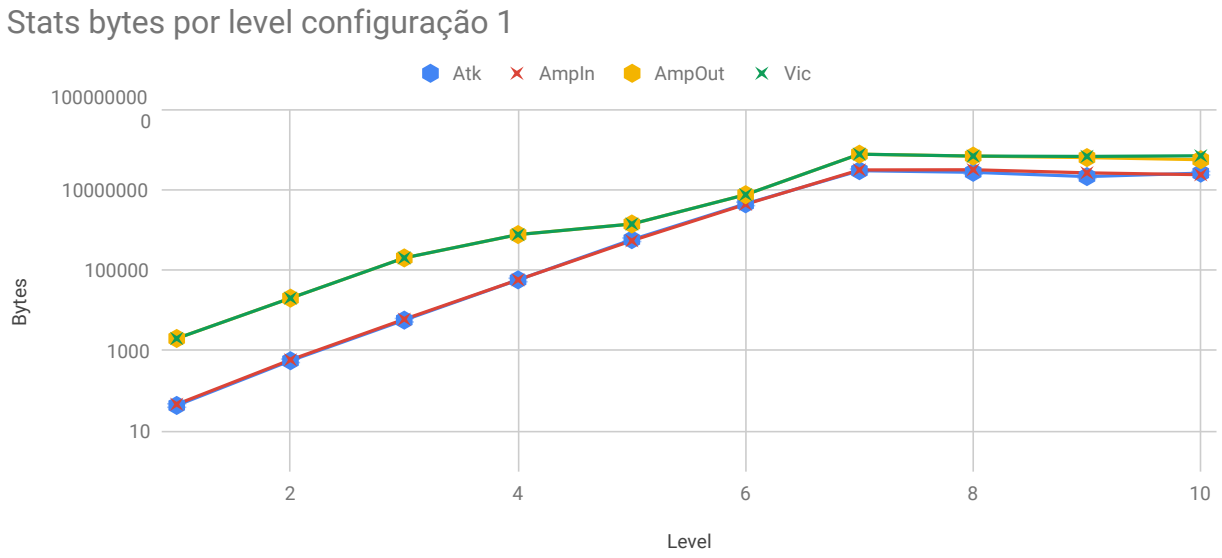

Figura 4.3: Stats bytes por level configuração 1

Tabela 4.6: Quantidade de bytes para cada nível no método Stats configuração 1

Nível	Atk	AmpIn	AmpOut	Vic
1	43	46	1976	1976
2	430	460	19760	19760
3	4300	4600	197600	197600
4	43000	43911	751670	751670
5	430000	410706	1378457	1378457
6	3349416	3305054	7374036	7374036
7	22310756	23652992	74979715	74989595
8	20320871	23895022	67225891	67222334
9	15911324	20037213	61656634	66000392
10	19375972	18002192	54702513	68442158

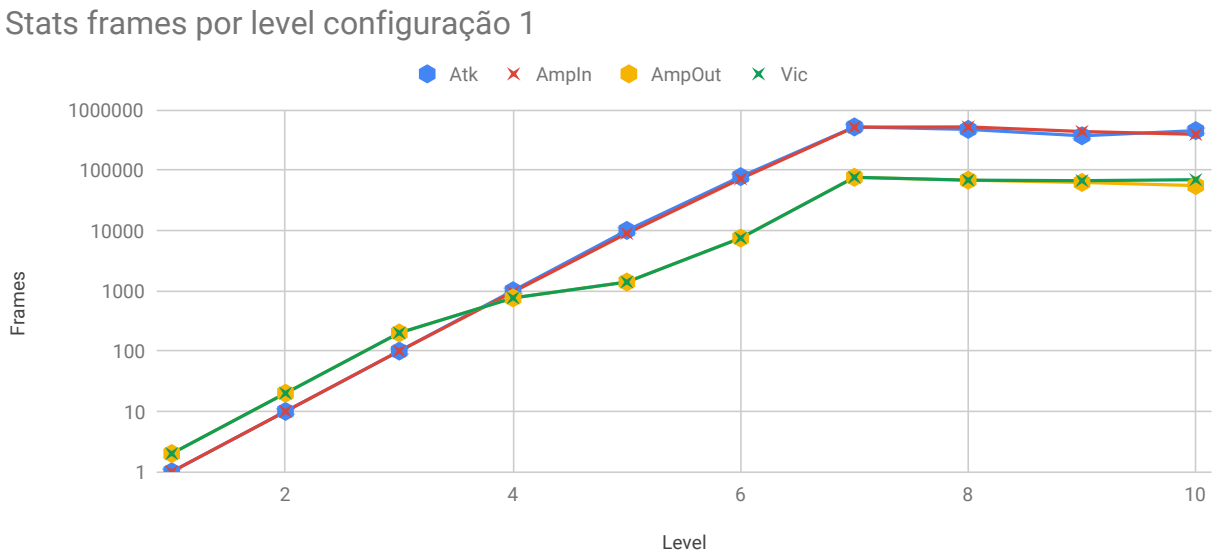

Figura 4.4: Stats frames por level configuração 1

Tabela 4.7: Quantidade de pacotes para cada nível no método Stats configuração 1

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	2	2
2	10	10	20	20
3	100	100	200	200
4	1000	954	760	760
5	10000	8928	1395	1395
6	77893	71849	7463	7463
7	518854	514195	75890	75900
8	472578	519457	68042	68038
9	370030	435591	62374	66768
10	450604	391352	55339	69238

Tabela 4.8: Razão de bytes por pacotes para cada nível no método Stats configuração 1

Nível	Input	Output
1	46.0	988.0
2	46.0	988.0
3	46.0	988.0
4	46.03	989.04
5	46.0	988.14
6	46.0	988.08
7	46.0	988.01
8	46.0	988.01
9	46.0	988.5
10	46.0	988.5

Configuração 2

Essa configuração se caracteriza por ter o refletor e atacante mais fortes e a vítima mais fraca.

No método GetSet, tivemos um fluxo de 49 bytes/pacote chegando no amplificador e um máximo de 1426.8 bytes/pacote saindo do amplificador no nível 7 como apresentado na tabela 4.12, apesar de o atacante saturar apenas no nível 8 como mostra os gráficos 4.5 e 4.6.

Já para o método Stats, tivemos um máximo de 46.04 bytes/pacote chegando no amplificador no nível 4 e um máximo de 1006.96 bytes/pacote saindo do amplificador

também no nível 4 como apresentado na tabela 4.15, apesar de o atacante saturar apenas no nível 7 como mostra os gráficos 4.7 e 4.8.

Tabela 4.9: Configuração dos ataques

Config	Sputnik	Spitfire	Orion
1	Atacante	Refletor	Vítima
2	Vítima	Atacante	Refletor
3	Refletor	Vítima	Atacante
4	Atacante	Vítima	Refletor
5	Refletor	Atacante	Vítima
6	Vítima	Refletor	Atacante

GetSet

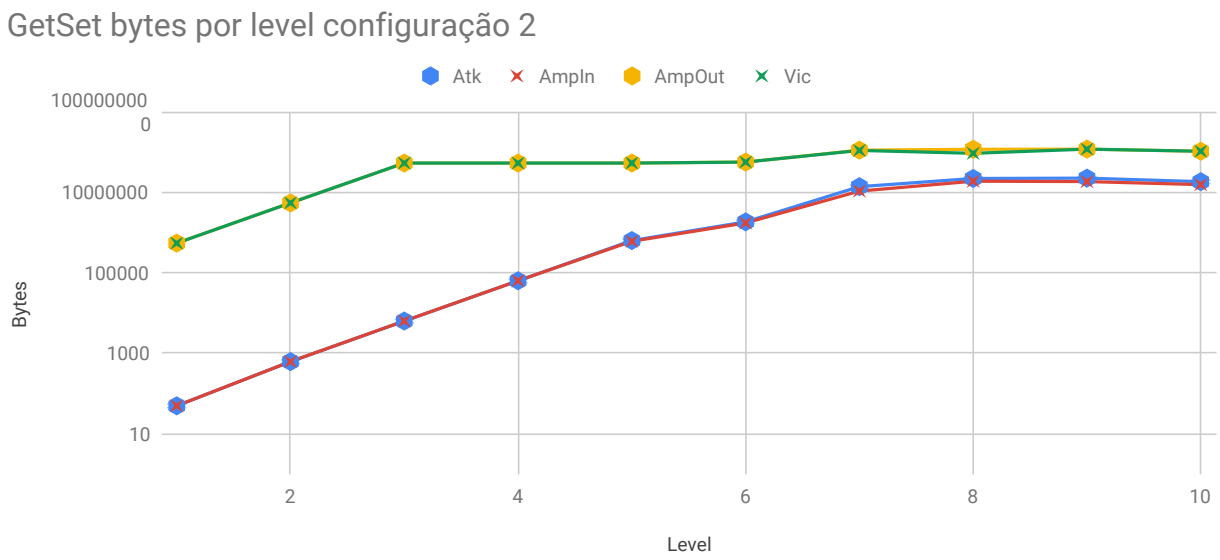

Figura 4.5: GetSet bytes por level configuração 2

Tabela 4.10: Quantidade de bytes para cada nível no método GetSet configuração 2

Nível	Atk	AmpIn	AmpOut	Vic
1	49	49	537898	537898
2	490	490	5378980	5378980
3	4900	4900	52714004	52714004
4	49000	49000	52714004	52714004
5	490000	469988	52714004	52714004
6	1429075	1342796	55726232	55726232
7	10722954	8324972	110311919	108611596
8	17131487	14646021	115955672	92165923
9	17469323	14349515	116961210	116966636
10	14270211	12054529	103709213	103726380

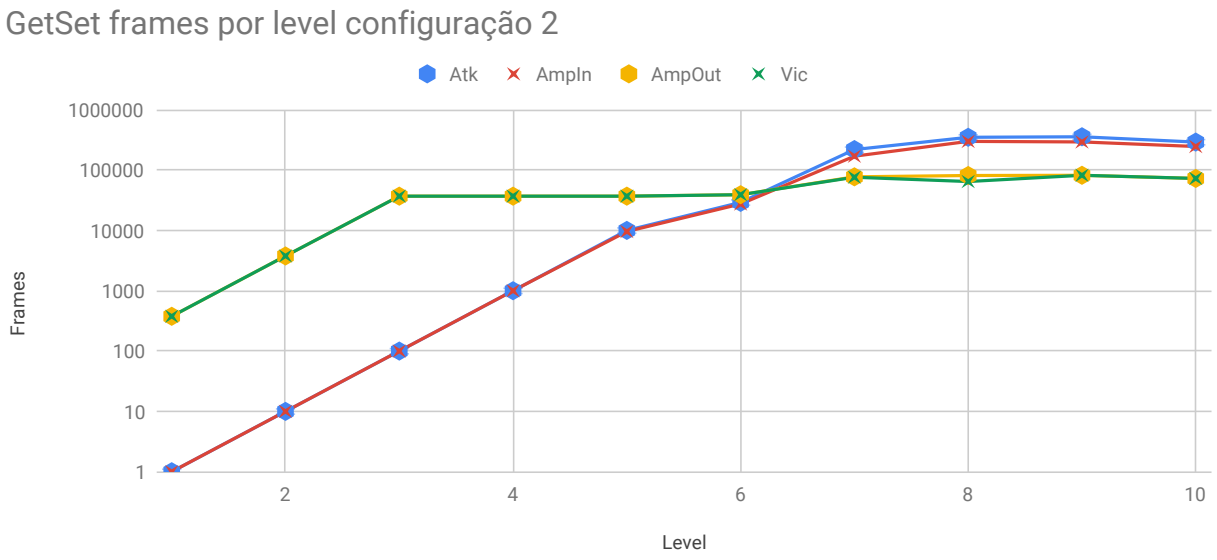

Figura 4.6: GetSet frames por level configuração 2

Tabela 4.11: Quantidade de pacotes para cada nível no método GetSet configuração 2

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	377	377
2	10	10	3770	3770
3	100	100	36946	36946
4	1000	1000	36946	36946
5	10000	9591	36946	36946
6	29164	27404	39057	39057
7	218835	169897	77314	76123
8	349622	298898	81270	64597
9	356516	292847	81975	81979
10	291228	246010	72687	72699

Tabela 4.12: Razão de bytes por pacotes para cada nível no método GetSet configuração 2

Nível	Input	Output
1	49.0	1426.79
2	49.0	1426.79
3	49.0	1426.79
4	49.0	1426.79
5	49.0	1426.79
6	49.0	1426.79
7	49.0	1426.8
8	49.0	1426.8
9	49.0	1426.79
10	49.0	1426.79

Stats

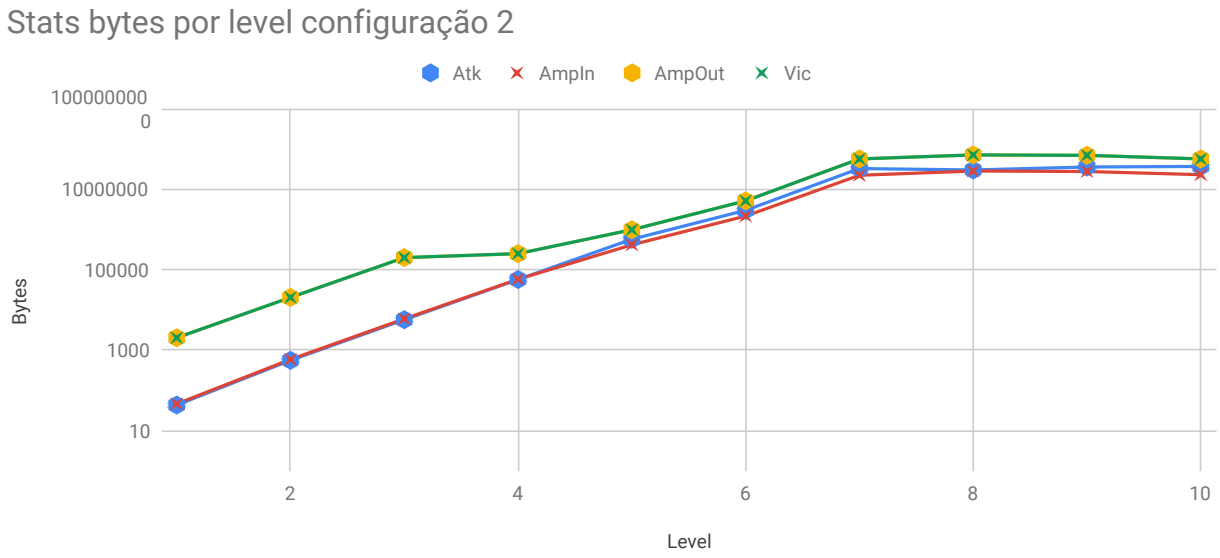

Figura 4.7: Stats bytes por level configuração 2

Tabela 4.13: Quantidade de bytes para cada nível no método Stats configuração 2

Nível	Atk	AmpIn	AmpOut	Vic
1	43	46	2009	2009
2	430	460	20090	20090
3	4300	4600	196882	196882
4	43000	44334	246705	246705
5	430000	317878	969543	969543
6	2254713	1648768	5066698	5066698
7	24641468	17011407	55848592	55848592
8	22708265	21762968	70242274	70242676
9	26920124	21118296	68595238	68596476
10	27807214	17616822	55864332	55862724

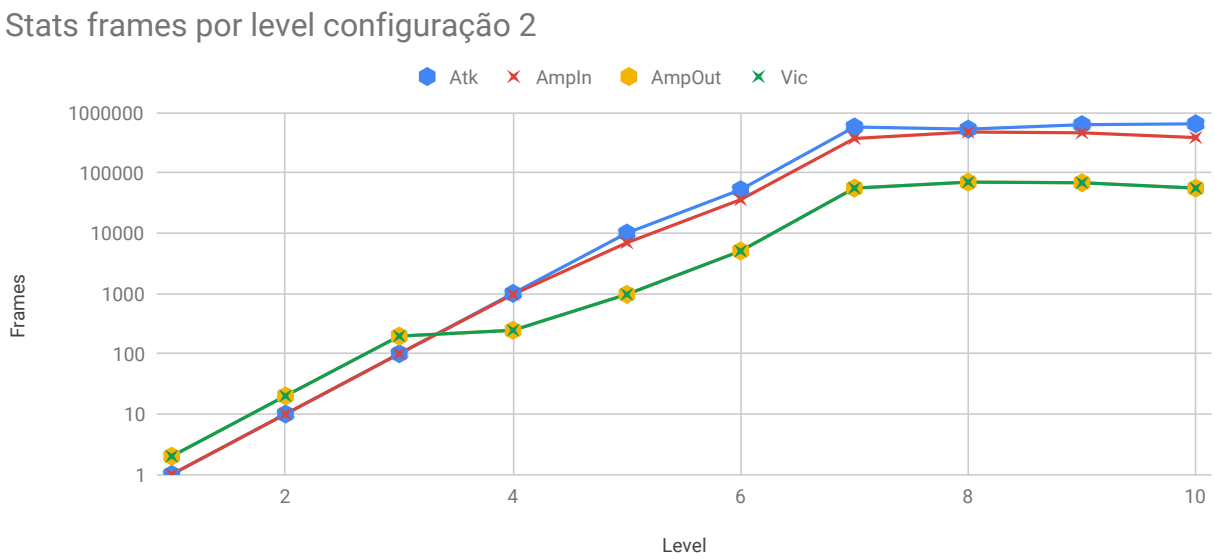

Figura 4.8: Stats frames por level configuração 2

Tabela 4.14: Quantidade de pacotes para cada nível no método Stats configuração 2

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	2	2
2	10	10	20	20
3	100	100	196	196
4	1000	963	245	245
5	10000	6910	965	965
6	52435	35842	5044	5044
7	573057	369813	55598	55598
8	528099	473108	69927	69928
9	626049	459093	68279	68280
10	646679	382974	55586	55584

Tabela 4.15: Razão de bytes por pacotes para cada nível no método Stats configuração 2

Nível	Input	Output
1	46.0	1004.5
2	46.0	1004.5
3	46.0	1004.5
4	46.04	1006.96
5	46.0	1004.71
6	46.0	1004.5
7	46.0	1004.51
8	46.0	1004.51
9	46.0	1004.63
10	46.0	1005.01

Configuração 3

Essa configuração se caracteriza por ter o atacante e vítima mais fortes e o refletor mais fraco.

No método GetSet tivemos um máximo de 49.31 bytes/pacote chegando no amplificador no nível 3 e um máximo de 1426.8 bytes/pacote saindo do amplificador no nível 7 como apresentado na tabela 4.19, com o atacante saturando no nível 7, como mostram os gráficos 4.9 e 4.10.

Já para o método Stats tivemos um fluxo de 46 bytes/pacote chegando no amplificador 5 e um máximo de 1000.53 bytes/pacote saindo do amplificador também no nível 9 como apresentado na tabela 4.22, apesar de o atacante saturar apenas no nível 7 como mostram os gráficos 4.11 e 4.12.

Tabela 4.16: Configuração dos ataques

Config	Sputnik	Spitfire	Orion
1	Atacante	Refletor	Vítima
2	Vítima	Atacante	Refletor
3	Refletor	Vítima	Atacante
4	Atacante	Vítima	Refletor
5	Refletor	Atacante	Vítima
6	Vítima	Refletor	Atacante

GetSet

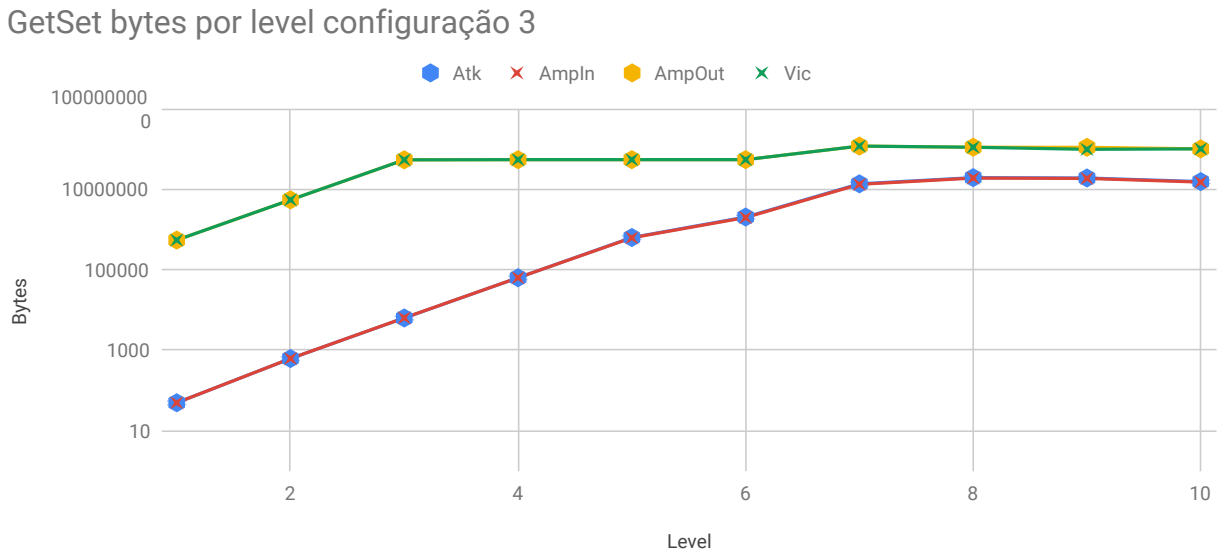

Figura 4.9: GetSet bytes por level configuração 3

Tabela 4.17: Quantidade de bytes para cada nível no método GetSet configuração 3

Nível	Atk	AmpIn	AmpOut	Vic
1	49	49	537898	537898
2	490	490	5378980	5378980
3	4900	4931	53251902	53251902
4	49000	49000	53789800	53789800
5	490000	481591	53789800	53789800
6	1583366	1536365	53789800	53789800
7	10597269	10292881	117033364	117003948
8	15106856	14712171	109427211	109424446
9	14802116	14437732	109087336	97138300
10	12039545	11685353	99793507	99824352

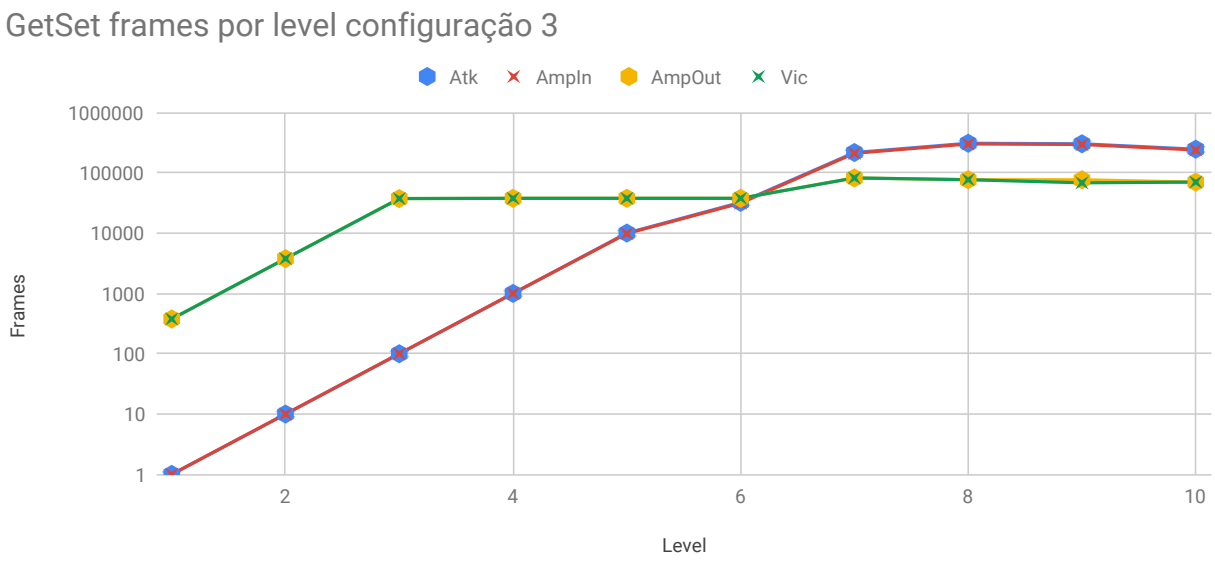

Figura 4.10: GetSet frames por level configuração 3

Tabela 4.18: Quantidade de pacotes para cada nível no método GetSet configuração 3

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	377	377
2	10	10	3770	3770
3	100	100	37323	37323
4	1000	1000	37700	37700
5	10000	9828	37700	37700
6	32313	31354	37700	37700
7	216270	210058	82025	82005
8	308303	300248	76695	76693
9	302084	294647	76456	68082
10	245705	238476	69943	69964

Tabela 4.19: Razão de bytes por pacotes para cada nível no método GetSet configuração 3

Nível	Input	Output
1	49.0	1426.79
2	49.0	1426.79
3	49.31	1426.79
4	49.0	1426.79
5	49.0	1426.79
6	49.0	1426.79
7	49.0	1426.8
8	49.0	1426.78
9	49.0	1426.8
10	49.0	1426.78

Stats

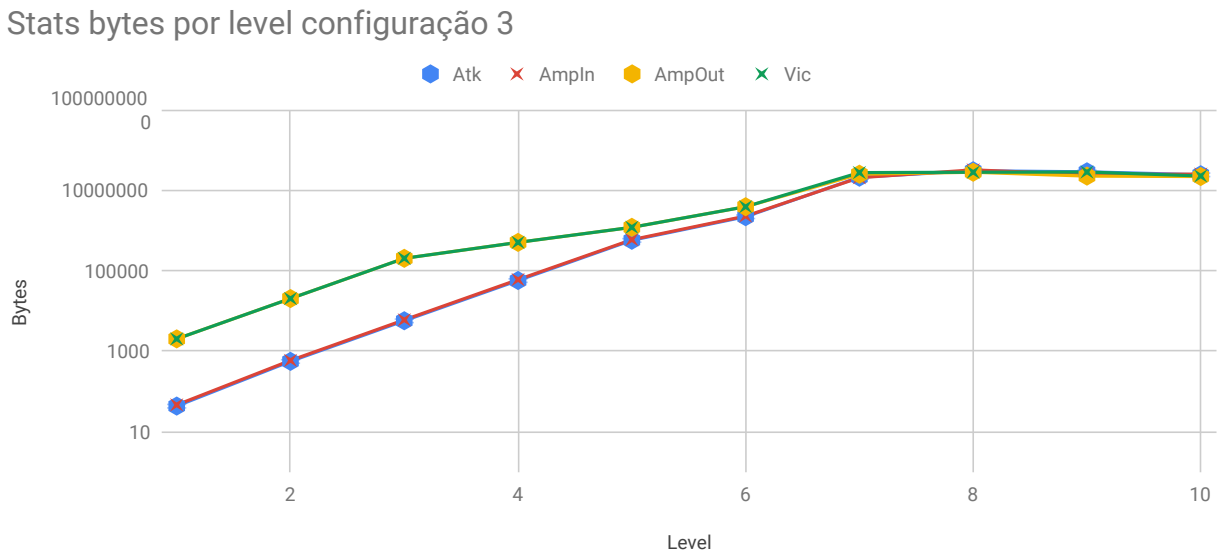

Figura 4.11: Stats bytes por level configuração 3

Tabela 4.20: Quantidade de bytes para cada nível no método Stats configuração 3

Nível	Atk	AmpIn	AmpOut	Vic
1	43	46	2000	2000
2	430	460	20000	20000
3	4300	4600	200000	200000
4	43000	46000	498000	498000
5	430000	451802	1180000	1180000
6	1658630	1744752	3840000	3840000
7	15905450	15749765	24701428	27060000
8	23007674	24588564	27634939	27635225
9	21660295	18847340	22152727	27989702
10	18306605	19269970	21627264	22317553

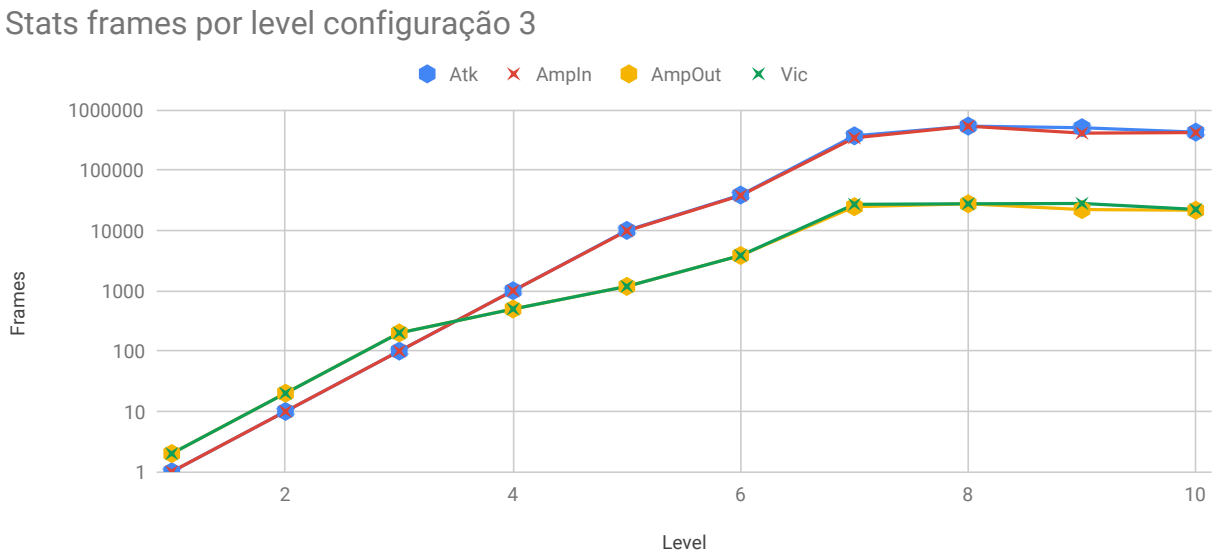

Figura 4.12: Stats frames por level configuração 3

Tabela 4.21: Quantidade de pacotes para cada nível no método Stats configuração 3

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	377	377
2	10	10	3770	3770
3	100	100	37323	37323
4	1000	1000	37700	37700
5	10000	9828	37700	37700
6	32313	31354	37700	37700
7	216270	210058	82025	82005
8	308303	300248	76695	76693
9	302084	294647	76456	68082
10	245705	238476	69943	69964

Tabela 4.22: Razão de bytes por pacotes para cada nível no método Stats configuração 3

Nível	Input	Output
1	46.0	1000.0
2	46.0	1000.0
3	46.0	1000.0
4	46.0	1000.0
5	46.0	1000.0
6	46.0	1000.0
7	46.0	1000.02
8	46.0	1000.29
9	46.0	1000.53
10	46.0	1000.52

Configuração 4

Essa configuração se caracteriza por ter o refletor e vítima mais fortes e a atacante mais fraco, similar à configuração 1.

No método GetSet, tivemos um máximo de 51.11 bytes/pacote chegando no amplificador no nível 2 e um máximo de 1427.02 bytes/pacote saindo do amplificador no nível 1 como apresentado na tabela 4.26, apesar de o atacante saturar apenas no nível 7 como mostram os gráficos 4.13 e 4.14.

Já para o método Stats, tivemos um máximo de 46.01 bytes/pacote chegando no amplificador no nível 4 e um máximo de 1006.5 bytes/pacote saindo do amplificador

também no nível 4 como apresentado na tabela 4.29, apesar de o atacante saturar apenas no nível 7, como mostram os gráficos 4.15 e 4.16.

Tabela 4.23: Configuração dos ataques

Config	Sputnik	Spitfire	Orion
1	Atacante	Refletor	Vítima
2	Vítima	Atacante	Refletor
3	Refletor	Vítima	Atacante
4	Atacante	Vítima	Refletor
5	Refletor	Atacante	Vítima
6	Vítima	Refletor	Atacante

GetSet

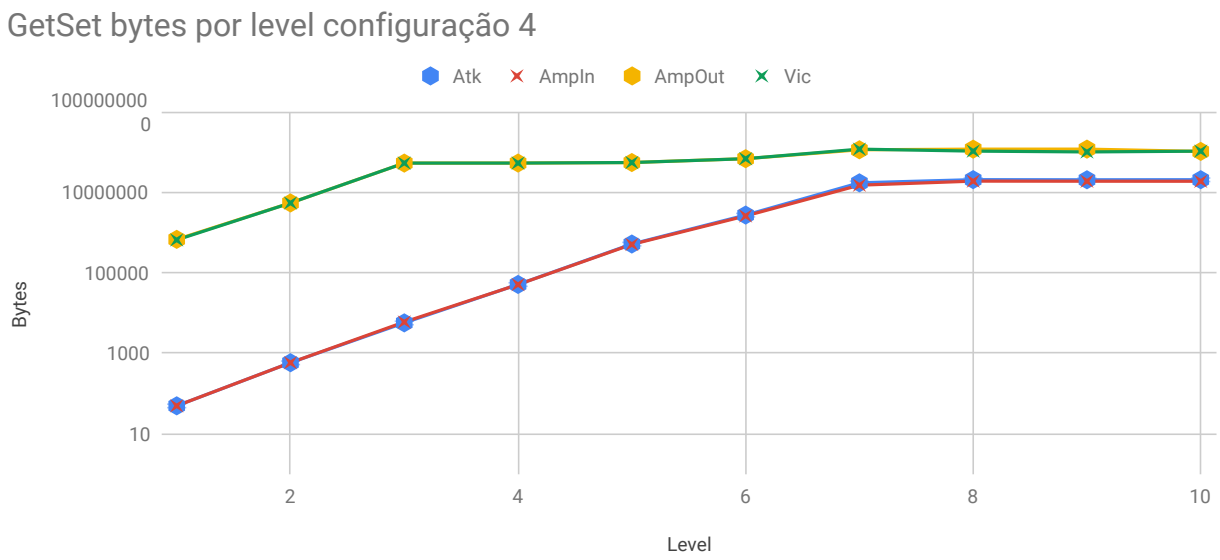

Figura 4.13: GetSet bytes por level configuração 4

Tabela 4.24: Quantidade de bytes para cada nível no método GetSet configuração 4

Nível	Atk	AmpIn	AmpOut	Vic
1	49	49	667846	647282
2	460	460	5369555	5359844
3	4410	4635	52593480	52632893
4	39719	39494	52714004	52704864
5	401800	392803	54327698	54327698
6	2108764	2005266	67667568	67667568
7	13351275	11664940	112696857	116954059
8	16016042	14724186	116956355	104482925
9	15902930	14646119	117019187	99736296
10	15968100	14608193	102724248	104136961

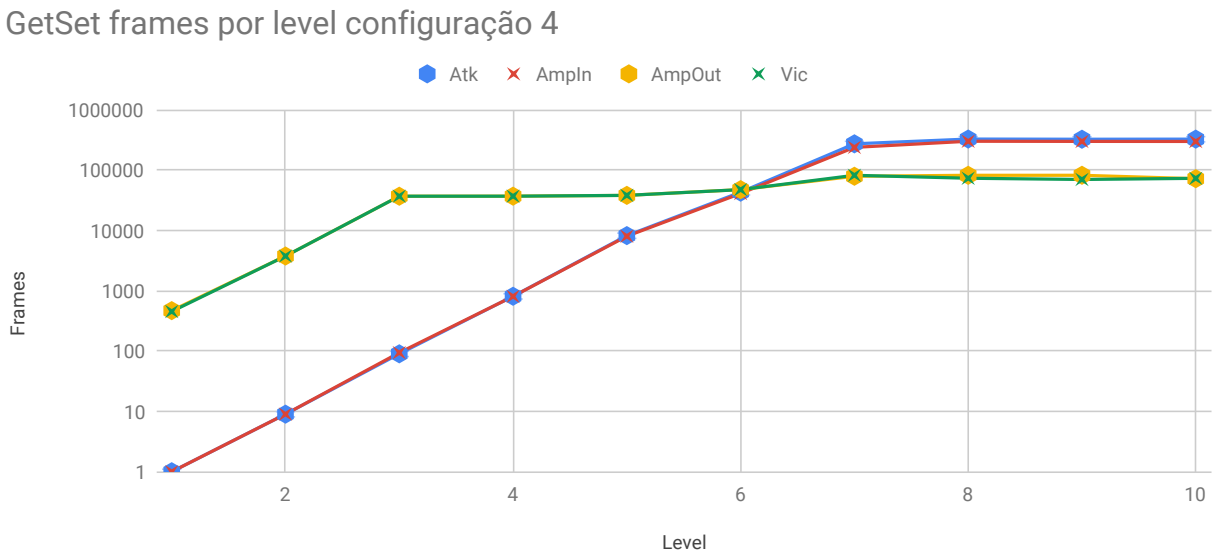

Figura 4.14: GetSet frames por level configuração 4

Tabela 4.25: Quantidade de pacotes para cada nível no método GetSet configuração 4

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	468	453
2	9	9	3763	3756
3	90	94	36861	36889
4	810	806	36946	36939
5	8200	8016	38077	38077
6	43036	40923	47426	47426
7	272475	238060	78986	81970
8	326858	300493	81972	73229
9	324549	298900	82016	69903
10	325879	298126	71997	72987

Tabela 4.26: Razão de bytes por pacotes para cada nível no método GetSet configuração 4

Nível	Input	Output
1	49.0	1427.02
2	51.11	1426.93
3	49.31	1426.81
4	49.0	1426.79
5	49.0	1426.79
6	49.0	1426.8
7	49.0	1426.8
8	49.0	1426.78
9	49.0	1426.78
10	49.0	1426.79

Stats

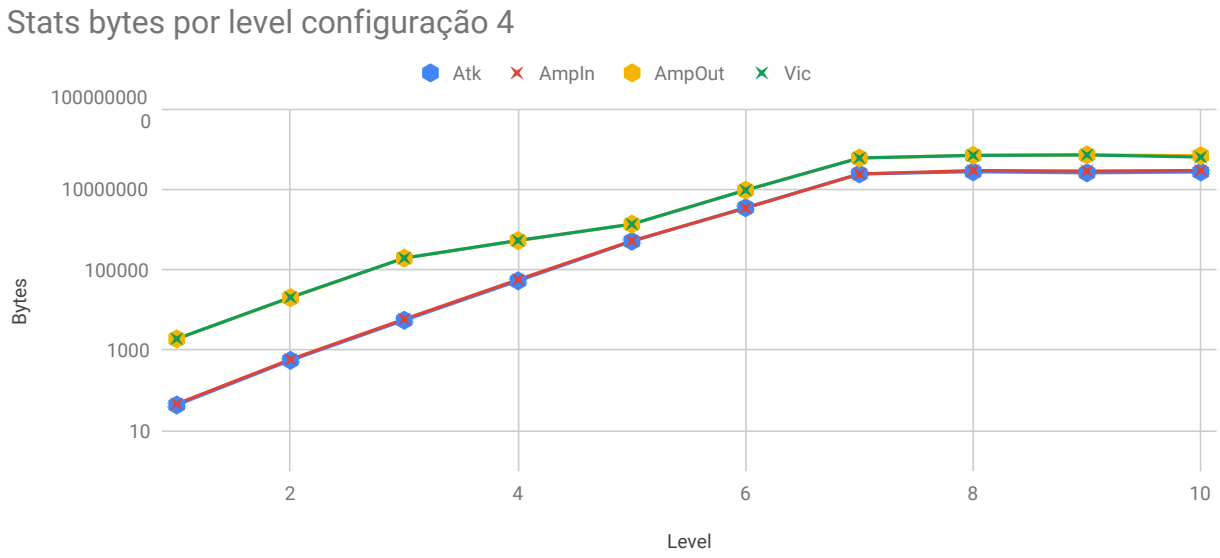

Figura 4.15: Stats bytes por level configuração 4

Tabela 4.27: Quantidade de bytes para cada nível no método Stats configuração 4

Nível	Atk	AmpIn	AmpOut	Vic
1	43	46	1894	1894
2	430	460	19824	20226
3	4171	4508	190928	191849
4	39783	43295	521367	524066
5	382158	398318	1334901	1334785
6	2619998	2604621	9347530	9344831
7	18166055	18299223	58974183	58980100
8	20474768	22342209	68508737	68508853
9	19279222	21657674	70239001	70237678
10	20298356	22364951	67573505	62164946

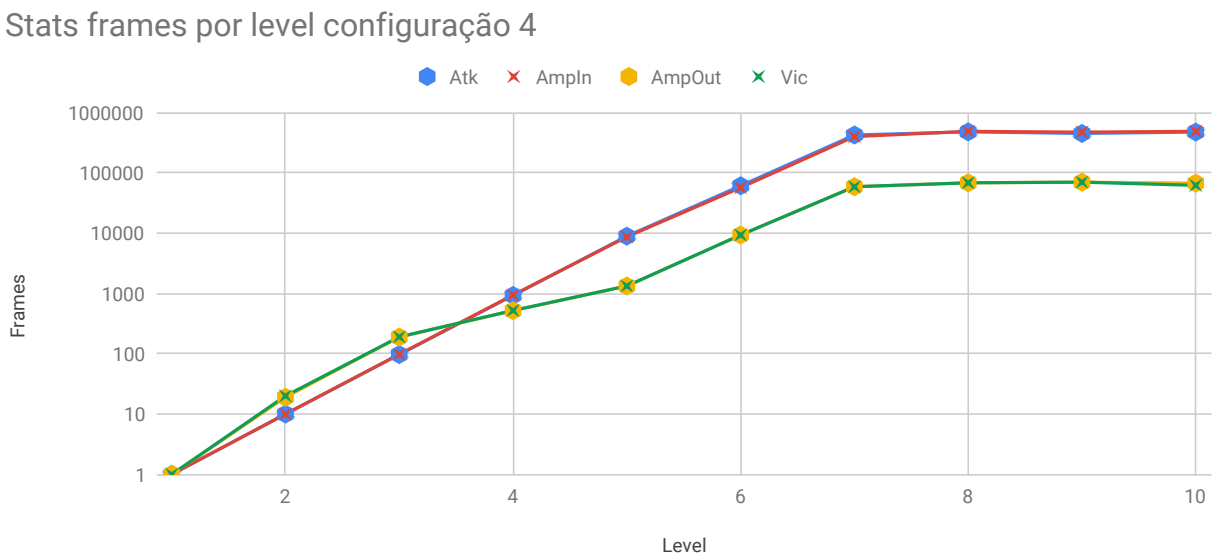

Figura 4.16: Stats frames por level configuração 4

Tabela 4.28: Quantidade de pacotes para cada nível no método Stats configuração 4

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	1	1
2	10	10	19	20
3	97	98	189	190
4	925	941	518	521
5	8887	8658	1327	1327
6	60930	56622	9296	9293
7	422466	397809	58651	58657
8	476157	485700	68134	68134
9	448354	470819	69854	69853
10	472054	486194	67203	61821

Tabela 4.29: Razão de bytes por pacotes para cada nível no método Stats configuração 4

Nível	Input	Output
1	46.0	1894.0
2	46.0	1043.37
3	46.0	1010.2
4	46.01	1006.5
5	46.01	1005.95
6	46.0	1005.54
7	46.0	1005.51
8	46.0	1005.5
9	46.0	1005.51
10	46.0	1005.51

Configuração 5

Essa configuração se caracteriza por ter o atacante e vítima mais fortes e a vítima mais fraca, similar à configuração 3.

No método GetSet tivemos um fluxo de 49 bytes/pacote chegando no amplificador e um máximo de 1426.8 bytes/pacote saindo do amplificador no nível 8 como apresentado na tabela 4.33, apesar de o atacante saturar apenas no nível 7, como mostram os gráficos 4.17 e 4.18.

Já para o método Stats, tivemos um fluxo de 46 bytes/pacote chegando no amplificador 5 e um máximo de 1004.18 bytes/pacote saindo do amplificador também no nível 10 como apresentado na tabela 4.36, com o atacante saturando no nível 7, como mostram os gráficos 4.19 e 4.20.

Tabela 4.30: Configuração dos ataques

Config	Sputnik	Spitfire	Orion
1	Atacante	Refletor	Vítima
2	Vítima	Atacante	Refletor
3	Refletor	Vítima	Atacante
4	Atacante	Vítima	Refletor
5	Refletor	Atacante	Vítima
6	Vítima	Refletor	Atacante

GetSet

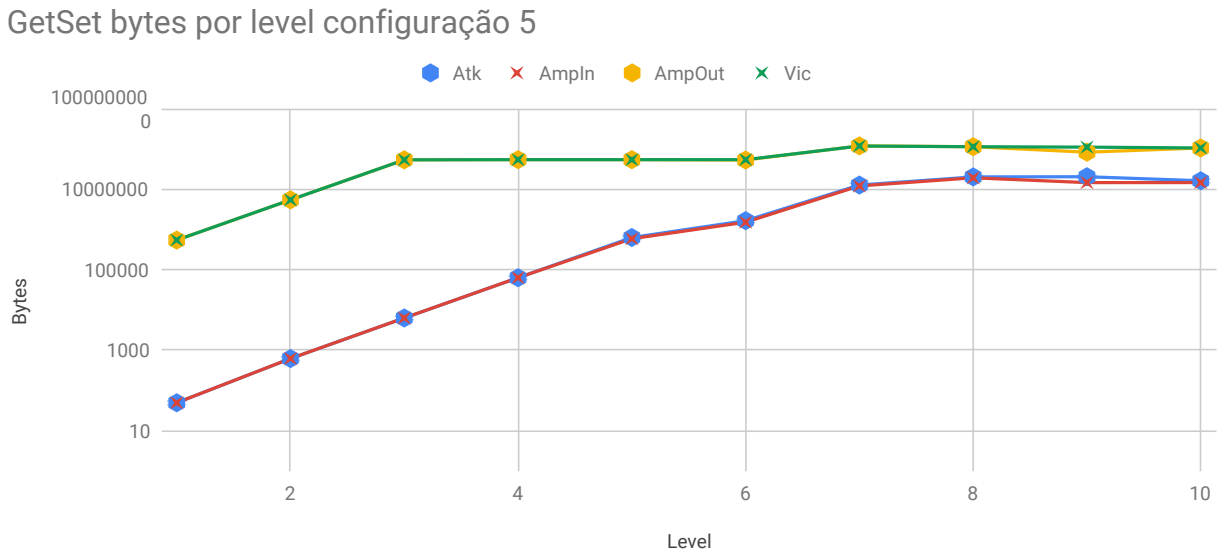

Figura 4.17: GetSet bytes por level configuração 5

Tabela 4.31: Quantidade de bytes para cada nível no método GetSet configuração 5

Nível	Atk	AmpIn	AmpOut	Vic
1	49	49	537898	537898
2	490	490	5378980	5378980
3	4900	4900	53251902	53251902
4	49000	49000	53789800	53789800
5	490000	457170	53789800	53789800
6	1284192	1176254	52665646	53789800
7	9876430	9399650	118337560	117494436
8	15731557	14932642	112868319	112826142
9	15899480	11350565	82399690	110166080
10	12621518	11387572	105087378	105125557

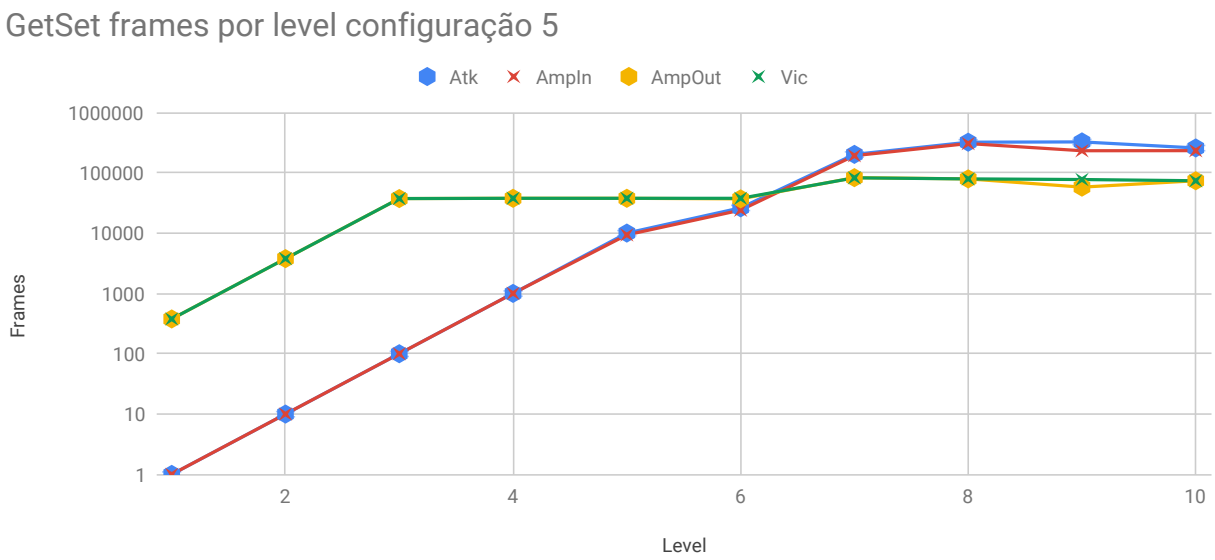

Figura 4.18: GetSet frames por level configuração 5

Tabela 4.32: Quantidade de pacotes para cada nível no método GetSet configuração 5

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	377	377
2	10	10	3770	3770
3	100	100	37323	37323
4	1000	1000	37700	37700
5	10000	9330	37700	37700
6	26208	24005	36912	37700
7	201559	191829	82940	82349
8	321052	304747	79106	79077
9	324479	231644	57752	77212
10	257582	232399	73653	73680

Tabela 4.33: Razão de bytes por pacotes para cada nível no método GetSet configuração 5

Nível	Input	Output
1	49.0	1426.79
2	49.0	1426.79
3	49.0	1426.79
4	49.0	1426.79
5	49.0	1426.79
6	49.0	1426.79
7	49.0	1426.79
8	49.0	1426.8
9	49.0	1426.79
10	49.0	1426.79

Stats

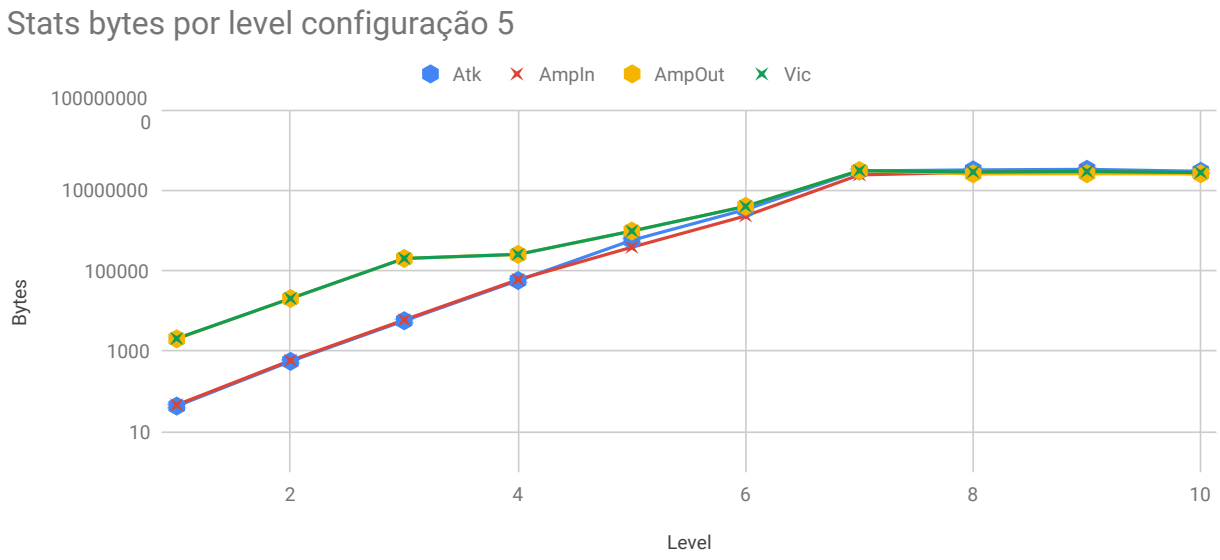

Figura 4.19: Stats bytes por level configuração 5

Tabela 4.34: Quantidade de bytes para cada nível no método Stats configuração 5

Nível	Atk	AmpIn	AmpOut	Vic
1	43	46	2007	2038
2	430	460	20070	20070
3	4300	4600	198693	198693
4	43000	46000	250875	250875
5	430000	293737	953325	953325
6	2496889	1764615	3925692	3925692
7	22729868	18461952	30323763	30323763
8	24154562	21341976	25159404	27966742
9	24947456	22998058	25348532	28745845
10	22294433	19560524	25242970	27224865

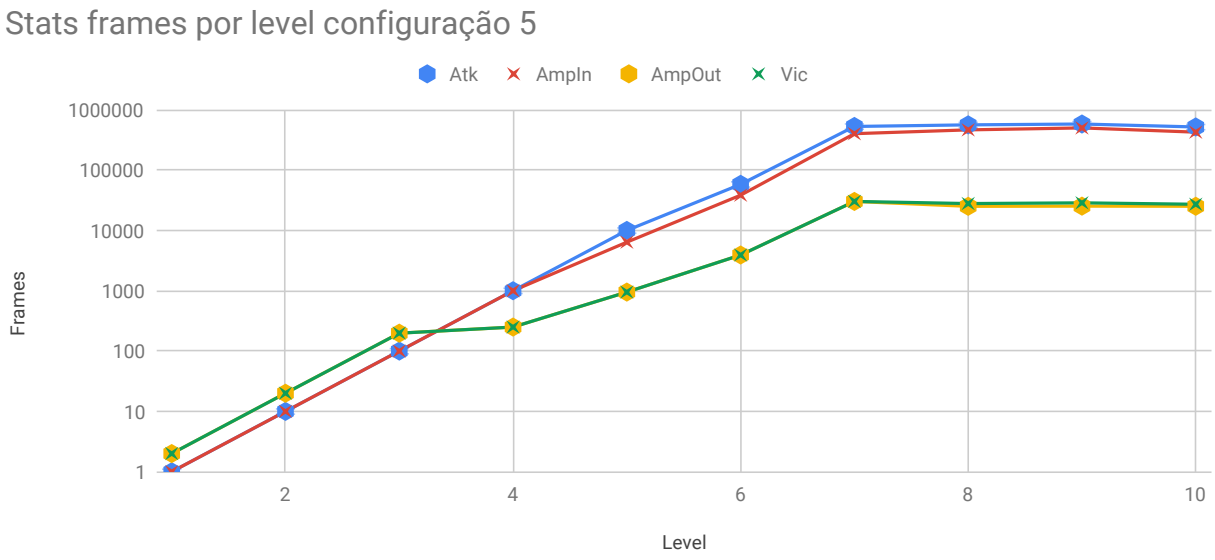

Figura 4.20: Stats frames por level configuração 5

Tabela 4.35: Quantidade de pacotes para cada nível no método Stats configuração 5

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	2	2
2	10	10	20	20
3	100	100	198	198
4	1000	1000	250	250
5	10000	6385	950	950
6	58067	38361	3912	3912
7	528601	401346	30218	30218
8	561734	463956	25071	27869
9	580173	499957	25256	28644
10	518475	425228	25138	27116

Tabela 4.36: Razão de bytes por pacotes para cada nível no método Stats configuração 5

Nível	Input	Output
1	46.0	1003.5
2	46.0	1003.5
3	46.0	1003.5
4	46.0	1003.5
5	46.0	1003.5
6	46.0	1003.5
7	46.0	1003.5
8	46.0	1003.53
9	46.0	1003.66
10	46.0	1004.18

Configuração 6

Essa configuração se caracteriza por ter o refletor e atacante mais fortes e a vítima mais fraca, semelhante à configuração 2.

No método GetSet, tivemos um fluxo de 49 bytes/pacotes chegando no amplificador e um máximo de 1426.81 bytes/pacote saindo do amplificador no nível 6 como apresentado na tabela 4.40, com o atacante saturando no nível 7, como mostram os gráficos 4.21 e 4.22.

Já para o método Stats, tivemos um máximo de 50 bytes/pacote chegando no amplificador no nível 2 e um máximo de 1032.05 bytes/pacote saindo do amplificador também

no nível 2 como apresentado na tabela 4.43, apesar de o atacante saturar no nível 7, como mostram os gráficos 4.23 e 4.24.

Tabela 4.37: Configuração dos ataques

Config	Sputnik	Spitfire	Orion
1	Atacante	Refletor	Vítima
2	Vítima	Atacante	Refletor
3	Refletor	Vítima	Atacante
4	Atacante	Vítima	Refletor
5	Refletor	Atacante	Vítima
6	Vítima	Refletor	Atacante

GetSet

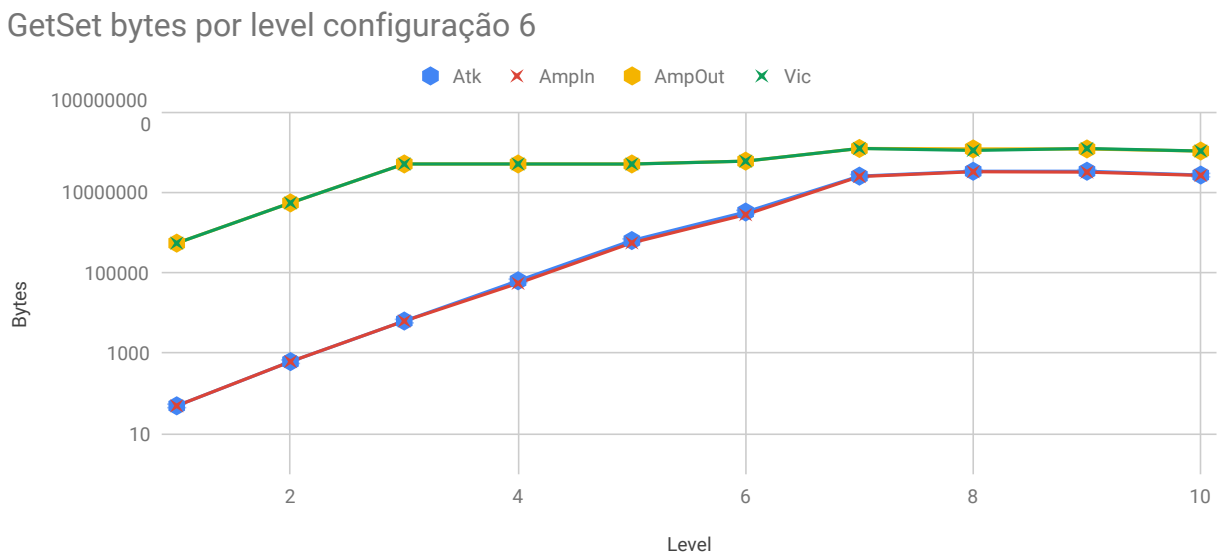

Figura 4.21: GetSet bytes por level configuração 6

Tabela 4.38: Quantidade de bytes para cada nível no método GetSet configuração 6

Nível	Atk	AmpIn	AmpOut	Vic
1	49	49	537898	537898
2	490	490	5378980	5378980
3	4900	4900	50024514	50024514
4	49000	42434	50024514	50024514
5	490000	429926	50024514	49782120
6	2509113	2159675	58846041	58846041
7	19587367	19118692	121625586	121621588
8	25906711	25214224	118493018	108738699
9	25929192	24644432	118131448	121273910
10	20861554	20429599	104956471	104939621

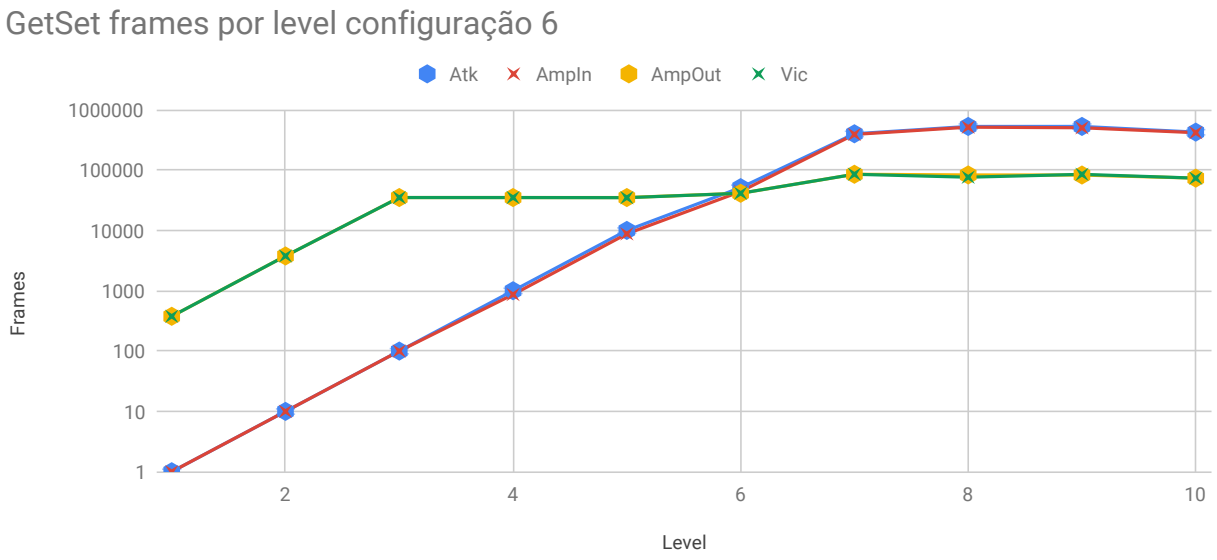

Figura 4.22: GetSet frames por level configuração 6

Tabela 4.39: Quantidade de pacotes para cada nível no método GetSet configuração 6

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	377	377
2	10	10	3770	3770
3	100	100	35061	35061
4	1000	866	35061	35061
5	10000	8774	35061	34891
6	51206	44075	41243	41243
7	399742	390177	85244	85241
8	528708	514576	83048	76212
9	529167	502947	82795	84998
10	425746	416930	73561	73549

Tabela 4.40: Razão de bytes por pacotes para cada nível no método GetSet configuração 6

Nível	Input	Output
1	49.0	1426.79
2	49.0	1426.79
3	49.0	1426.79
4	49.0	1426.79
5	49.0	1426.79
6	49.0	1426.81
7	49.0	1426.79
8	49.0	1426.8
9	49.0	1426.79
10	49.0	1426.8

Stats

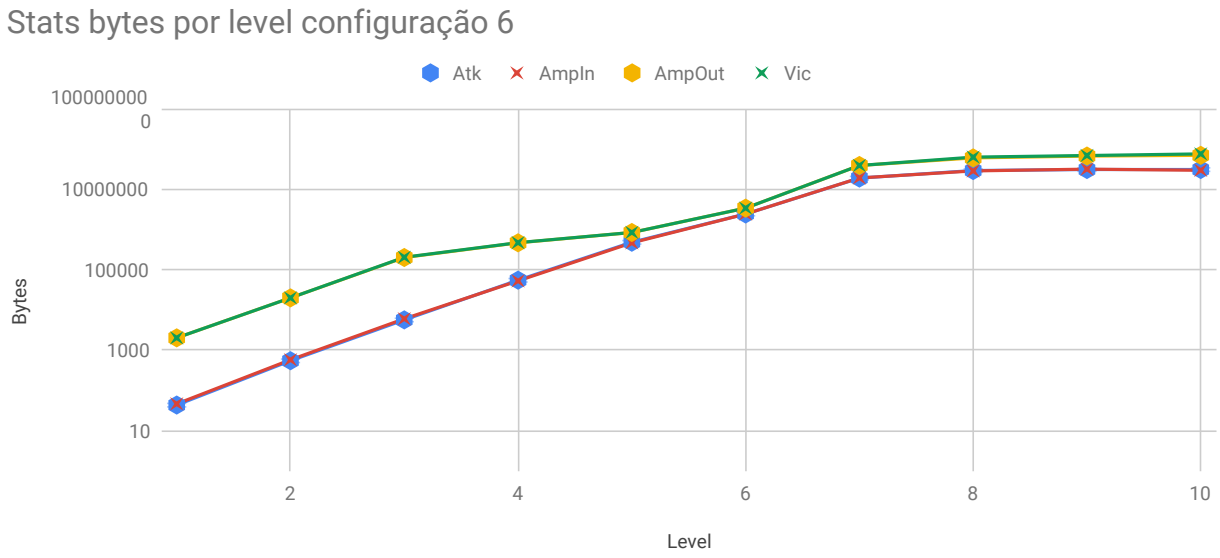

Figura 4.23: Stats bytes por level configuração 6

Tabela 4.41: Quantidade de bytes para cada nível no método Stats configuração 6

Nível	Atk	AmpIn	AmpOut	Vic
1	43	46	2001	2001
2	421	450	19609	19609
3	4265	4609	197698	200500
4	41323	40397	458629	462631
5	362765	355929	828414	829214
6	1827568	1853119	3339669	3344071
7	14284608	14602111	38358769	38346763
8	21969224	21884619	59247090	61939354
9	23305174	24174462	65852335	67844705
10	23309044	22672700	68650286	74495357

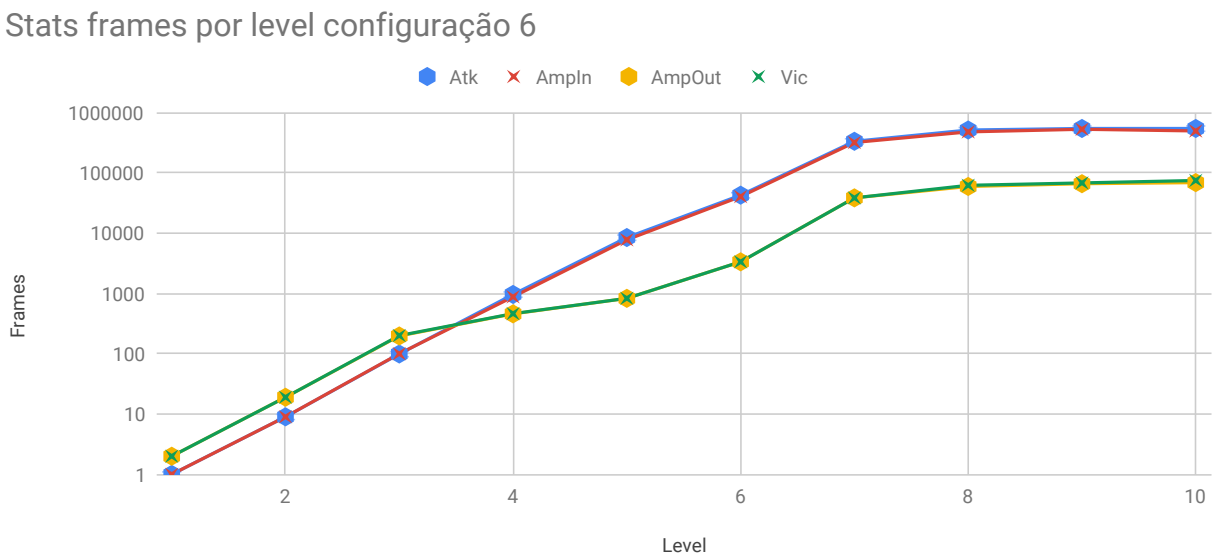

Figura 4.24: Stats frames por level configuração 6

Tabela 4.42: Quantidade de pacotes para cada nível no método Stats configuração 6

Nível	Atk	AmpIn	AmpOut	Vic
1	1	1	2	2
2	9	9	19	19
3	99	100	197	200
4	961	878	458	462
5	8436	7737	828	828
6	42501	40285	3338	3342
7	332200	317437	38339	38327
8	510912	475752	59216	61908
9	541980	525531	65818	67810
10	542070	492884	68584	74428

Tabela 4.43: Razão de bytes por pacotes para cada nível no método Stats configuração 6

Nível	Input	Output
1	46.0	1000.5
2	50.0	1032.05
3	46.09	1003.54
4	46.01	1001.37
5	46.0	1000.5
6	46.0	1000.5
7	46.0	1000.52
8	46.0	1000.53
9	46.0	1000.52
10	46.0	1000.97

4.1.1 Amplificação

Nessa seção iremos apresentar as tabelas de amplificação. O cálculo de amplificação apresentado nas tabelas 4.44 e 4.46 é similar à fórmula apresentada em 2.2. A diferença aqui é que optamos por fazer a divisão de todos os bytes recebidos pelo refletor por todos os bytes enviados pelo refletor em cada nível para facilitar a visualização de sua saturação. As tabelas 4.45 e 4.47 apresentam a amplificação dos pacotes de acordo com a quantidade de pacotes que chega no amplificador com a quantidade que é enviado para a vítima.

Tabela 4.44: Tabela de amplificação dos bytes por nível no método GetSet

	Configuração					
Nível	**1**	**2**	**3**	**4**	**5**	**6**
1	11735.22	10977.51	10977.51	13629.51	10977.51	10977.51
2	12176.51	10977.51	10977.51	11672.95	10977.51	10977.51
3	12596.76	10757.96	10799.41	11347.03	10867.74	10209.08
4	1329.95	1075.8	1097.75	1334.73	1097.75	1178.88
5	138.07	112.16	111.69	138.31	117.66	116.36
6	21.03	41.5	35.01	33.74	44.77	27.25
7	6.0	13.25	11.37	9.66	12.59	6.36
8	4.96	7.92	7.44	7.94	7.56	4.7
9	5.19	8.15	7.56	7.99	7.26	4.79
10	4.95	8.6	8.54	7.03	9.23	5.14

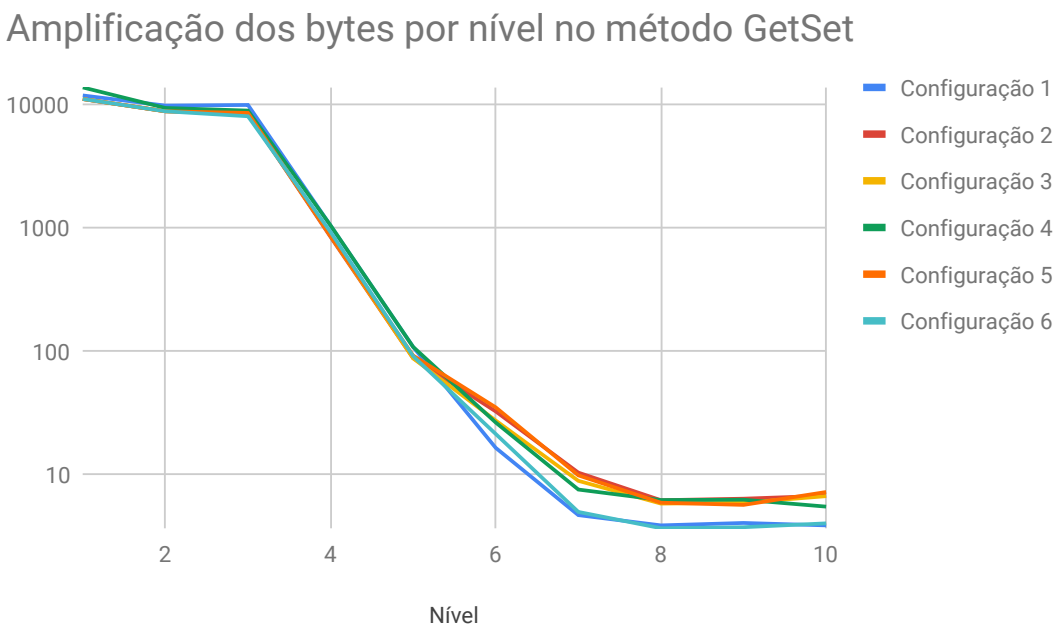

Figura 4.25: Gráfico amplificação dos bytes por nível no método GetSet

Tabela 4.45: Amplificação dos pacotes por nível no método GetSet

	Configuração					
Nível	**1**	**2**	**3**	**4**	**5**	**6**
1	403	377	377	468	377	377
2	418.11	377	377	418.11	377	377
3	432.6	369.46	373.23	392.14	373.23	350.61
4	45.68	36.95	37.7	45.84	37.7	40.49
5	4.74	3.85	3.84	4.75	4.04	4
6	0.72	1.43	1.2	1.16	1.54	0.94
7	0.21	0.46	0.39	0.33	0.43	0.22
8	0.17	0.27	0.26	0.27	0.26	0.16
9	0.18	0.28	0.26	0.27	0.25	0.16
10	0.17	0.3	0.29	0.24	0.32	0.18

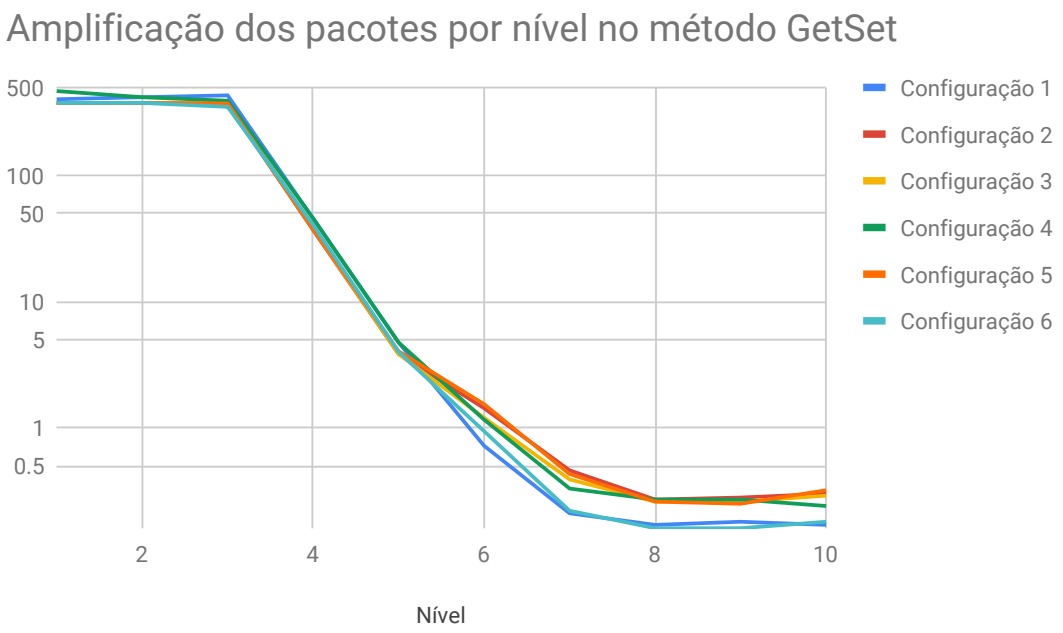

Figura 4.26: Gráfico amplificação dos pacotes por nível no método GetSet

Tabela 4.46: Amplificação dos bytes por nível no método Stats

	Configuração					
Nível	**1**	**2**	**3**	**4**	**5**	**6**
1	42.96	43.67	43.48	41.17	43.63	43.5
2	42.96	43.67	43.48	43.1	43.63	43.58
3	42.96	42.8	43.48	42.35	43.19	42.89
4	17.12	5.56	10.83	12.04	5.45	11.35
5	3.36	3.05	2.61	3.35	3.25	2.33
6	2.23	3.07	2.2	3.59	2.22	1.8
7	3.17	3.28	1.57	3.22	1.64	2.63
8	2.81	3.23	1.12	3.07	1.18	2.71
9	3.08	3.25	1.18	3.24	1.1	2.72
10	3.04	3.17	1.12	3.02	1.29	3.03

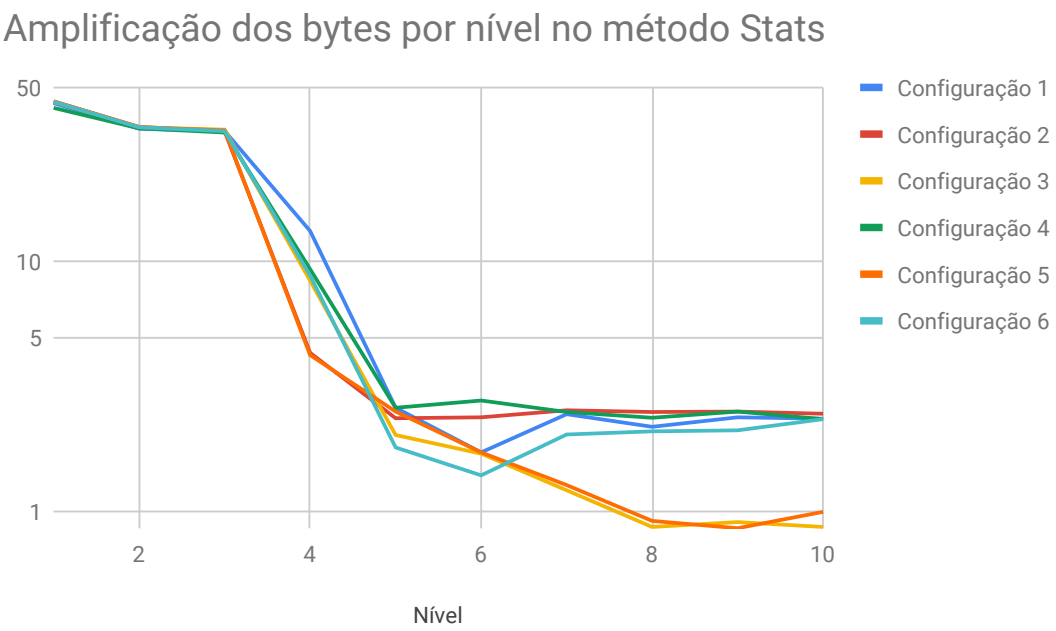

Figura 4.27: Gráfico amplificação dos bytes por nível no método Stats

Tabela 4.47: Amplificação dos pacotes por nível no método Stats

	Configuração					
Nível	**1**	**2**	**3**	**4**	**5**	**6**
1	2	2	2	1	2	2
2	2	2	2	1.9	2	2.11
3	2	1.96	2	1.93	1.98	1.97
4	0.8	0.25	0.5	0.55	0.25	0.52
5	0.16	0.14	0.12	0.15	0.15	0.11
6	0.1	0.14	0.1	0.16	0.1	0.08
7	0.15	0.15	0.07	0.15	0.08	0.12
8	0.13	0.15	0.05	0.14	0.05	0.12
9	0.14	0.15	0.05	0.15	0.05	0.13
10	0.14	0.15	0.05	0.14	0.06	0.14

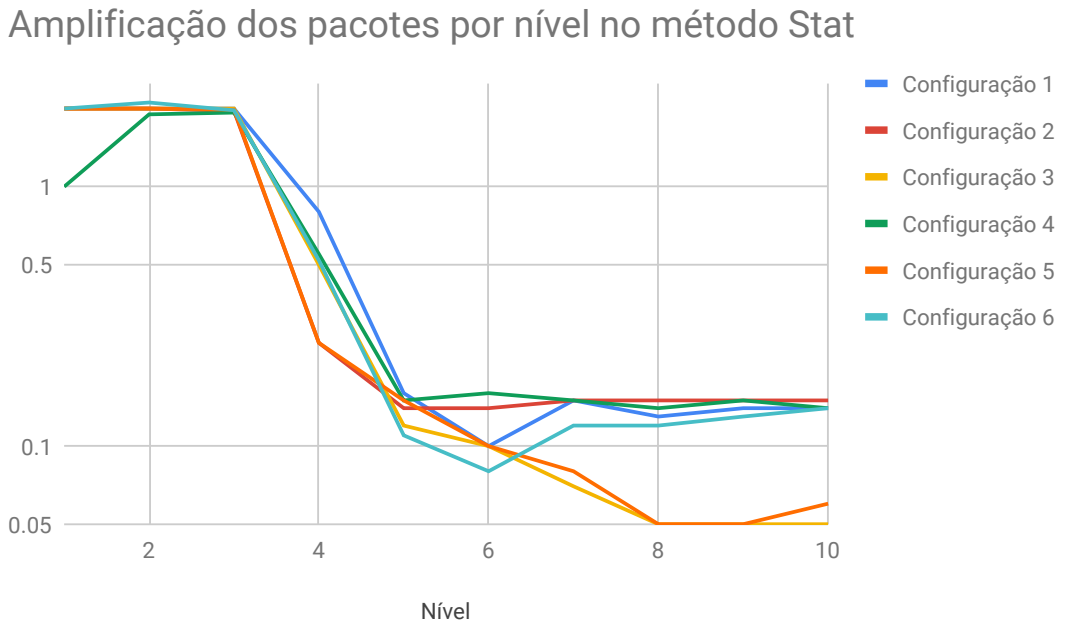

Figura 4.28: Gráfico amplificação dos pacotes por nível no método Stats

4.1.2 Análise da amplificação

Ao observarmos a tabela 4.44, vimos que as configurações 1 e 4 foram aquelas nas quais tivemos as maiores amplificações no método GetSet, com a configuração 4 fornecendo a maior amplificação. Essas duas configurações possuem o refletor forte, mas são refletores diferentes, com a configuração 1 utilizando o Spitfire e a 4 o Orion. O interessante foi que essa maior amplificação não se repetiu nas configurações 2 e 6, que apresentaram uma amplificação próxima à amplificação fornecida pelo Sputnik, que era a máquina mais fraca. A maior semelhança nessas configurações foi a utilização do Sputnik como atacante, mas como ele é responsável apenas por enviar as requisições para o refletor e utilizamos o Linderhof na execução do ataque em todas as configurações, não verificamos qualquer influência que o atacante poderia causar no refletor além de aumentar as requisições.

No método Stats, a tabela 4.46 mostrou que as configurações 1 e 4 geraram as menores amplificações e as configurações 2 e 6 geraram as maiores amplificações, sendo exatamente o oposto do que aconteceu no método GetSet. Apesar de ser outro comando esperávamos que as melhores configurações em ambos os métodos fossem iguais, isso nos da mais uma pista de que o atacante não teve uma influência no aumento da amplificação.

Essa maior amplificação observada foi devida a uma maior amplificação dos pacotes que saíram do refletor; entretanto não conseguimos descobrir o motivo de tal.

4.1.3 Análise de performance do atacante

O atacante não recebe nenhum pacote, ele apenas envia para o refletor os pacotes Get ou Stats Memcached. A preparação que deve ser feita antes de começar o ataque GetSet não foi considerada na análise, já que nessa etapa do ataque não existe nenhuma amplificação, com o pacote de Set Memcached sendo muito maior que a resposta do servidor, se existir.

Ao observarmos o comportamento do atacante, verificamos que ele se comportou da maneira esperada, i.e, sem saturar, até o nível 7, apesar de, a partir do nível 6, ele não conseguiu entregar a quantidade de pacotes desejada. Nos gráficos apresentados na seção 4.1, observamos que nos níveis de 8 ao 10 a geração de pacotes começa a ficar constante, indicando a saturação no atacante.

No log do Linderhof, observamos que a máquina Sputnik conseguiu gerar os pacotes esperados até o nível 6 e as máquinas Spitfire e Orion tiveram um comportamento melhor, indo até o nível 7. Esse comportamento já era esperado, uma vez que o Sputnik é máquina mais fraca utilizada nos teste.

4.1.4 Análise de performance do refletor

No ataque o refletor é responsável por receber as requisições do atacante, amplificá-las e enviá-las para a vítima. Como utilizamos a configuração padrão do Memcached, o maior valor que conseguimos armazenar no servidor é de 524288 bytes com cada requisição tendo 49 bytes já incluindo os cabeçalhos IP e UDP, já que não consideramos o cabeçalho ethernet. Isso nos dá uma valor teórico para a amplificação de:

$$Amp = \frac{20 + 8 + 524288}{49} \approx 10700 \tag{4.1}$$

Para o comando Stats, cada pacote de requisição possui 46 bytes com a reposta tendo 2004 bytes com ambos os valores já incluindo os cabeçalhos IP e UDP. Isso nós dá uma valor teórico para a amplificação de:

$$Amp = \frac{2004}{46} \approx 43 \tag{4.2}$$

Ao observarmos a tabela 4.44, verificamos que no método GetSet a amplificação no primeiro nível foi acima do esperado em todas as configurações. Isso ocorreu devido à amplificação de pacotes feita pelo servidor Memcached que observamos na tabela 4.45. Um pacote nesse método conseguiu gerar uma alta quantidade de pacotes em retorno para a vítima, por isso observamos uma amplificação um pouco maior que a teórica. Nas

configurações 1 e 4, tivemos uma maior amplificação exatamente devido a essa maior amplificação dos pacotes.

Do nível 1 ao 3 o refletor manteve a amplificação, e no nível 4 tivemos uma acentuada redução nos bytes enviados no refletor, onde temos o início da saturação. Nos níveis seguintes observamos apenas a redução da amplificação, com os níveis de 8 em diante estando dentro do ponto de saturação do atacante.

No método Stats, a amplificação no primeiro nível estava dentro do esperado, com exceção do nível 4, que enviou apenas um pacote. A saturação nesse método ocorreu a partir do nível 4, assim como o método Stats. Esse método também amplificou a quantidade de pacotes enviados para o cliente de 1 para 2 pacotes na maioria das configurações, com exceção da configuração 4. Essa amplificação, porém, não é tão relevante quanto no método GetSet.

4.1.5 Análise da vítima

A vítima apenas recebe os pacotes do refletor. Ao observarmos os gráficos que apresentam os bytes por nível, vimos que a maioria dos bytes que saem do refletor chegam na vítima.

4.2 Considerações finais

Com os testes feitos em laboratório, comprovamos que o Memcached possui características para funcionar como um refletor e um amplificador que consegue alcançar altos valores de amplificação. Entretanto, a amplificação não se sustentou com uma alta injeção de pacotes no servidor, sendo muito mais vantajoso para o atacante injetar até 100 pacotes por segundo.

Na versão 1.5.10 do Memcached, o valor padrão para um item armazenado foi reduzido de 1Mb para 524288 bytes. Apesar disso, o servidor pode estar configurado para armazenar um item com valores maiores que o padrão, sendo assim possível gerar uma amplificação ainda maior.

Como as máquinas tinham diferença de hardware optamos por fazer a rotação delas entre atacante, amplificador e vítima. Esperávamos observar diferentes pontos de saturação entre as configurações, entretanto não foi isso que aconteceu, elas se comportaram de maneira similar para ambos os métodos. Algumas configurações apresentaram uma maior ou menor amplificação dos bytes devido a uma diferença no número de pacotes que o servidor utilizou para gerar a resposta. Não descobrimos o motivo do servidor Memcached utilizar um diferente número de pacotes na geração da resposta dessas configurações.

Como o atacante satura apenas no nível 7 na maioria da configurações e não observamos nenhuma saturação no roteador, a saturação no nível 4 no refletor pode ser por uma saturação no Memcached ou no barramento das máquinas utilizadas.

Capítulo 5

Conclusão e trabalhos futuros

5.1 Conclusão

Essa monografia apresentou as características de um ataque de negação de serviço por reflexão amplificada explorando o Memcached, que é uma aplicação para caching em memória de chave-valor.

Primeiro foram apresentados os princípios de um ataque de negação de serviço, o conceito de uma ataque refletido amplificado e como explorar o Memcached para efetuar um ataque de reflexão amplificada.

Em seguida, apresentamos a ferramenta que foi implementada para gerar o ataque. Nela podemos adicionar qualquer tipo de ataque de negação de serviço, mas a ênfase é para ataque refletidos amplificados, sendo necessário apenas criar um mirror para a preparação e chamada do injetor para iniciar o ataque.

Com a ferramenta desenvolvida, fizemos a coleta de dados em laboratório para verificar as propriedades do ataque. Nessa coleta utilizamos três máquinas e um roteador para conectá-las. Como as máquinas não tinham o mesmo poder computacional, optamos por fazer o rodízio das mesmas entre atacante, amplificador e vítima para reduzir qualquer interferência que essa diferença de poder computacional pudesse deixar na análise dos dados, resultando em 6 configurações de teste. Executamos cada nível de ataque por 5 segundos e os pacotes enviados e recebidos por cada máquina em cada configuração foram apresentados em gráficos e tabelas.

Com a análise dos dados coletados, verificamos alguns aspectos interessantes. O atacante, a partir do nível 8 observamos que a quantidade de pacotes gerados fica constante, indicando uma saturação. Entretanto, essa saturação no atacante não interferiu na análise do refletor, já que ele saturou antes do atacante. A respeito do refletor a amplificação foi dentro do esperado nos níveis de um a três. A saturação do refletor ocorreu a partir do nível 4, que foi o primeiro ponto no qual observamos uma grande redução da amplificação.

Como o refletor saturou antes do atacante, essa saturação pode ser devido ao Memcached descartar alguns pacotes ou uma saturação no barramento das máquinas utilizadas.

Pelos resultados dos testes, vimos que o ataque de negação de serviço por reflexão e amplificação utilizando o Memcached é viável e entrega uma grande amplificação, mas deve injetar-se uma pequena quantidade de pacotes no refletor com um espaço de tempo considerável entre o envio de cada pacote para obtermos o melhor desempenho do refletor.

Devido a versão padrão do Memcached não possuir nenhum tipo de autenticação, ao contrario da sua versão raramente utilizada com autenticação via TLS, devesse evitar expô-lo para a internet. Caso necessário, os projetistas devem tomar algumas precauções para detectar, classificar, rastrear e mitigar o ataque. Esse ataque pode ser facilmente mitigado ao utilizar apenas o protocolo TCP. Caso seja necessário utilizar o protocolo UDP algumas precauções devem ser tomadas, como utilizar Ingress Filtering/BPC38 e lista de controle de acesso. [19]

5.2 Trabalhos futuros

Os seguintes trabalhos futuros são propostos:

- **Melhora do Linderhof:** Após análise do comportamento do refletor Memcached, seria uma boa ideia adicionar uma funcionalidade no Netuno para controlar a frequência de injeção dos pacotes. Outra melhoria seria a utilização de uma lista de refletores de ataque junto com uma distribuição de carga.
- **Implementação de novos mirrors**: A ferramenta Linderhof tem o potencial de possuir uma variedades de ataques DoS refletidos.
- **Executar teste em uma configuração mais próxima do real:** Para entender melhor como o ataque funciona, deveríamos fazer a coleta de dados em um ambiente mais próximo de real, para verificarmos os possíveis efeitos colaterais do ataque.

Referências

[1] Lipson, Howard F.: *Tracking and tracing cyberattacks : Technical challenges and global policy issues*. 2002. ix, 1, 2

[2] Gandhi, Robin, Anup Sharma, William Mahoney, William Sousan, Qiuming Zhu e Phillip Laplante: *Dimensions of Cyber-Attacks*. Ieee Technology and Society Magazine, páginas 28–38, 2011, ISSN 0278-0097. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5725605. 2

[3] skottler: *February 28th ddos incident report*. https://githubengineering.com/ddos-incident-report/. 3

[4] Tung, Liam: *New world record ddos attack hits 1.7tbps days after landmark github outage*. https://www.zdnet.com/article/new-world-record-ddos-attack-hits-1-7tbps-days-after-landmark-github-outage/. 3

[5] Cloudflare: *Famous ddos attacks | the largest ddos attacks of all time*. https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/. 3

[6] Peng, Tao, Christopher Leckie e Kotagiri Ramamohanarao: *Survey of network-based defense mechanisms countering the DoS and DDoS problems*. ACM Computing Surveys, 39(1):3–es, 2007, ISSN 03600300. http://portal.acm.org/citation.cfm?doid=1216370.1216373. 5

[7] Specht, Stephen M. e Ruby B. Lee: *Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures*. International Workshop on Security in Parallel and Distributed Systems, (9):543–550, 2004, ISSN 15384047. 5, 6

[8] Eddy, W.: *TCP SYN Flooding Attacks and Common Mitigations*. Network Working Group, página 3, 2000. 6

[9] Alomari, Esraa, Selvakumar Manickam, B. B. Gupta, Shankar Karuppayah e Rafeef Alfaris: *Botnet-based Distributed Denial of Service (DDoS) Attacks on Web Servers: Classification and Art*. International Journal of Computer Applications, 49(7):24–32, 2012, ISSN 09758887. http://research.ijcaonline.org/volume49/number7/pxc3880724.pdf. 7, 10

[10] insecure.org: *Windows nt/95/3.11 out of band (oob) data barf*. https://insecure.org/sploits/windows.OOB.DOS.html. 7

[11] insecure.org: *Ping of death*. https://insecure.org/sploits/ping-o-death.html. 7

[12] Chang, Author Rocky K C: *Defending against Distributed Denial-of- Service Attacks : A Tutorial*. (October):42–51, 2002. 7

[13] Radhakrishnan, Sivasankar, Yuchung Cheng, Jerry Chu, Arvind Jain e Barath Raghavan: *TCP fast open*. Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies on - CoNEXT '11, páginas 1–12, 2011, ISSN 2070-1721. http://dl.acm.org/citation.cfm?doid=2079296.2079317. 8

[14] Anagnostopoulos, Marios, Georgios Kambourakis, Panagiotis Kopanos, Georgios Louloudakis e Stefanos Gritzalis: *DNS amplification attack revisited*. Computers and Security, 39(PART B):475–485, 2013, ISSN 01674048. 8

[15] Gu, Qijun e Peng Liu: *Denial of Service Attacks*. Handbook of Computer Networks, 3:454–468, 2012, ISSN 21660735. 9

[16] Memcached: *Memcached text protocol*. https://github.com/memcached/memcached/blob/master/doc/protocol.txt. 13

[17] Memcached: *Memcached binary protocol*. https://github.com/memcached/memcached/wiki/BinaryProtocolRevamped. 15

[18] Gondim, João José Costa, Robson de Oliveira Albuquerque, Anderson Clayton Alves Nascimento, Luis Javier García Villalba e Tai Hoon Kim: *A methodological approach for assessing amplified reflection distributed denial of service on the internet of things*. Sensors (Switzerland), 16(11):1–31, 2016, ISSN 14248220. 26

[19] Dobbins, Roland: *Memcached reflection/amplification description and ddos attack mitigation recommendations*. https://www.netscout.com/blog/asert/memcached-reflectionamplification-description-and-ddos-attack. 69