Relatório de Arquitetura do Projeto de HoneyPot em sistema distribuído

Matheus Pamplona Oliveira - 201900600
Samuel Santos Machado - 201705643

Introdução

Este projeto tem como objetivo desenvolver um sistema honeypot distribuído voltado para monitorar e registrar atividades maliciosas, utilizando conceitos fundamentais de sistemas distribuídos. O sistema implementa funções-chave como simulação de serviços vulneráveis, coleta e análise de dados de intrusão, e comunicação eficiente entre os nodos distribuídos.

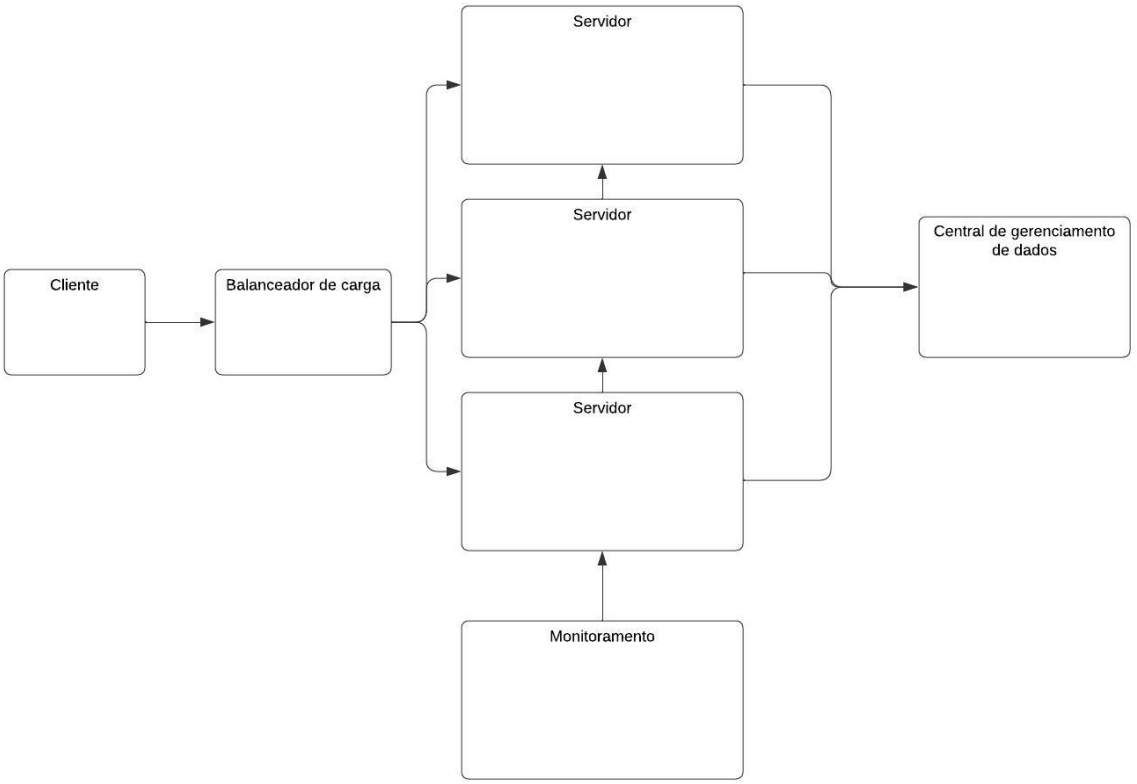
O projeto aborda soluções para desafios clássicos em sistemas distribuídos, como:

- **Coleta de Dados em Tempo Real**: Garantir que os dados das interações dos atacantes sejam capturados e processados com rapidez e precisão.
- **Escalabilidade**: Permitir a integração de múltiplos honeypots em diferentes locais, suportando um grande número de conexões simultâneas.
- **Resiliência**: Assegurar que o sistema permaneça funcional e confiável mesmo diante de ataques ou falhas em nodos específicos.

Utilizando ferramentas como NGINX para o balanceamento das cargas, técnica ddos para o ataque, o failover para transferência da carga e socket.IO para criar uma comunicação em tempo real entre honeypots e o servidor de monitoramento..

Arquitetura

O projeto segue uma arquitetura composta por diferentes camadas que trabalham em conjunto para entregar escalabilidade, resiliência e segurança para os servidores. Abaixo, explicamos as principais partes do diagrama:



Cliente: O cliente no nosso caso seria o nosso atacante que vai utilizar a técnica DDos para acessar um dos servidores de honeypot.

Balanceador de carga: Utilizamos o NGINX que atua como balanceador externo, distribuindo as requisições dos honeypots para o servidor de monitoramento ou failover, garantindo eficiência e escalabilidade.

Servidores em paralelo: São os nossos honeypot que capturam os dados das conexões e enviam informações, filtrando e bloqueando IPs se necessário.

Central de gerenciamento de dados: Centraliza os dados recebidos dos honeypots, exibe no painel em tempo real e retransmite as informações para os servidores de failover.

Monitoramento: Interface visual que apresenta as conexões ativas, logs e métricas em tempo real, permitindo gerenciamento do sistema.

Conceitos de Sistemas Distribuídos Utilizados

Nesse projeto de HoneyPot utilizamos diversos conceitos fundamentais de sistemas distribuídos, tais como:

1. **Comunicação por Sockets e comunicação direta:** Utilizamos a biblioteca net para criar um servidor que aceita conexões TCP. Sendo um exemplo de comunicação direta entre processos em sistemas distribuídos.

2. **Arquitetura Cliente-Servidor:** O honeypot atua como um servidor que aceita conexões de clientes maliciosos. Além disso, ele se comunica com um servidor de monitoramento através de sockets usando o protocolo Socket.IO.

3. **Middleware de Monitoramento:** O honeypot envia dados para um servidor central de monitoramento, que atua como um middleware para coletar, processar e exibir informações sobre as conexões. Esse é um exemplo de integração e centralização de logs em sistemas distribuídos.

4. **Gerenciamento de Recursos Distribuídos:** No honeypot implementamos um mecanismo básico para gerenciar recursos de rede, bloqueando IPs específicos. Este é um exemplo de política de segurança em sistemas distribuídos, onde recursos críticos são protegidos contra abuso.

5. **Escalabilidade e Modularidade:** A arquitetura do honeypot permite escalar horizontalmente, adicionando mais instâncias que reportam ao mesmo servidor de monitoramento.

6. **Failover e Alta Disponibilidade:** O sistema utiliza uma lista de servidores de failover para assegurar continuidade operacional em caso de falha no servidor principal. Dados são enviados aos servidores de failover, garantindo resiliência.
7. **CORS (Cross-Origin Resource Sharing):** Configuração do CORS no monitor_server que permite que o painel de monitoramento seja acessado por diferentes origens. Isso é essencial para sistemas distribuídos com interfaces web acessíveis a partir de domínios diferentes.

8. **Monitoramento em Tempo Real com Atualizações Periódicas:** A função setInterval envia atualizações periódicas para os clientes conectados ao painel, garantindo que o estado do sistema seja refletido em tempo real. Isso reflete conceitos de sistemas de monitoramento distribuídos em tempo real.

Linha do Tempo: Fluxo do Processo

1. **Início do Servidor:** No início do honeypot está ativo, sendo monitorado e registrando o estado normal do tráfego.

2. **Início do Ataque :** Um atacante tenta explorar o honeypot, enviando várias requisições ao mesmo tempo e acessando por várias portas

3. **Identificação de Atividade Suspeita:** O honeypot detecta um volume excessivo quantidade de pacotes vindo do nosso caso de um mesmo IP em um curto período

4. **Log de Atividades:** O honeypot começa a bloquear esses acessos, grava e retorna pro atacante, que no nosso caso vai passar pelo o nosso monitoramento que registra em tempo real o que acontece no honeypot.

5. **Pós-Ataque DDoS:** O balanceador detecta o aumento de tráfego e distribui as requisições entre múltiplos servidores honeypot. O balanceador redireciona o tráfego para outros servidores ativos, evitando que o ataque sobrecarregue ou derrube o sistema.

Perguntas e Respostas: Métricas de Sucesso

1. Qual é o principal indicador de sucesso do sistema honeypot?

O principal indicador de sucesso é a capacidade do honeypot de atrair e registrar interações maliciosas de forma detalhada e precisa, incluindo logs de atividades, padrões de ataque e dados dos atacantes (como IPs e métodos utilizados).

2. Como medir a eficácia na coleta de dados?

A eficácia na coleta de dados pode ser medida pelo número de eventos maliciosos registrados, pela completude das informações coletadas e pela taxa de sucesso em identificar padrões úteis para análise posterior.

3. O sistema é escalável para suportar múltiplos nodos distribuídos?

Sim, a escalabilidade é avaliada pelo número de nodos que podem ser adicionados ao sistema sem degradação perceptível de desempenho, mantendo tempos aceitáveis de comunicação e registro de eventos entre nodos.

4. Como garantir a disponibilidade do honeypot mesmo durante ataques?

A resiliência é medida pela capacidade do sistema de continuar operando mesmo sob carga adversa, utilizando monitoramento ativo e estratégias de failover para redirecionar tráfego entre nodos quando necessário.

Abaixo, algumas imagens de códigos e estrutura do sistema bem como capturas de tela do projeto em funcionamento:

Trecho do honeypot em javascript, com algumas informações de configuração e que serão transmitidas (Conexões ativas, requisições e Ip’s bloqueados).

```
const failoverServers = ["http://localhost:3013", "http://localhost:3014"];
let activeConnections = 0;
let totalRequests = 0;
const attackLogs = [];  // Para registrar os IPs das conexões detectadas

app.get("/", (req, res) => {
  res.sendFile(__dirname + "/dashboard.html");
});

app.get("/metrics", (req, res) => {
  res.json({
    activeConnections,
    totalRequests,
    attackLogs
  });
});

io.on("connection", (socket) => {
  console.log("A honeypot connected.");
  activeConnections++;
  updateDashboard();

  socket.on("honeypotData", (data) => {
    console.log("Data received from honeypot:", data);
    totalRequests++;

    // Armazenar logs das conexões detectadas
    attackLogs.push({
      ip: data.clientAddress,
      port: data.attackerPort,
      timestamp: new Date().toISOString(),
      message: data.message
    });

    updateDashboard();

    failoverServers.forEach((failoverUrl) => {
      sendToFailover(failoverUrl, data);
    });
  });

  socket.on("disconnect", () => {
    console.log("A honeypot disconnected.");
    activeConnections--;
    updateDashboard();
  });

  function updateDashboard() {
    io.emit("monitorUpdate", {
      message: `Connections: ${activeConnections}, Total Requests: ${totalRequests}`,
      activeConnections,
      totalRequests,
      attackLogs
    });
```

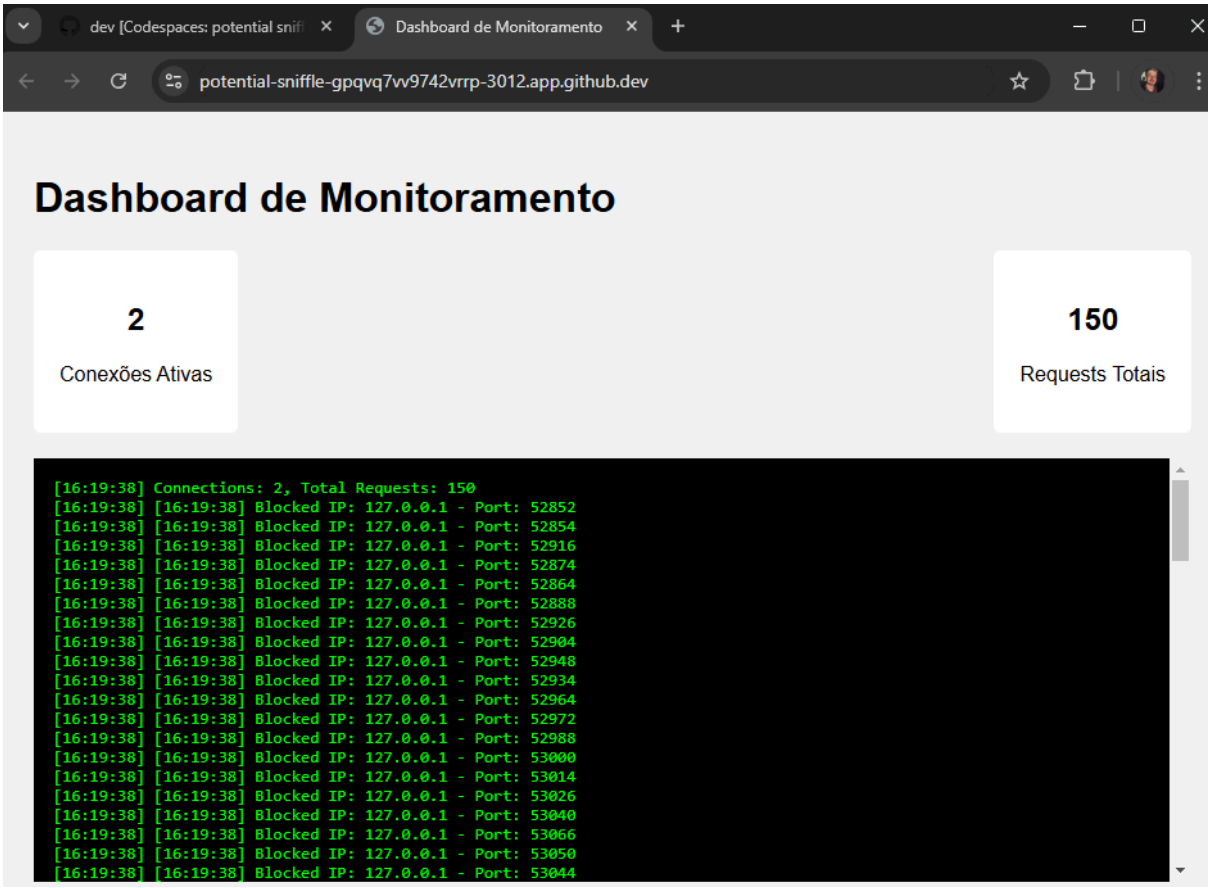
Trecho do monitor em javascript que vai capturar as informações do honey e atualizar em tempo real e exibir caso o servidor se desconecte.

```html
<body>
  <h1>Dashboard de Monitoramento</h1>

  <div class="metrics">
    <div class="metric">
      <h2 id="active-connections">0</h2>
      <p>Conexões Ativas</p>
    </div>
    <div class="metric">
      <h2 id="total-requests">0</h2>
      <p>Requests Totais</p>
    </div>
  </div>

  <div id="logs"></div>

  <script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>
  <script>
    const logsDiv = document.getElementById("logs");
    const socket = io("https://potential-sniffle-gpqvq7vv9742vrrp-3012.app.github.dev/");

    socket.on("monitorUpdate", (data) => {
      const { activeConnections, totalRequests, message, attackLogs } = data;
      document.getElementById("active-connections").textContent = activeConnections;
      document.getElementById("total-requests").textContent = totalRequests;

      // Exibe a mensagem principal
      addLog(message);

      // Processa os logs de ataques
      attackLogs.forEach((log) => {
        if (log.message.includes("Connection closed:")) {
          const messageParts = log.message.split('::ffff:')[1];  // Extrai IP e porta
          const [ip, port] = messageParts.split(':'); // Divide IP e Porta

          if (ip && port) {
            const timestamp = `[${new Date().toLocaleTimeString()}] `;
            addLog(`${timestamp}Blocked IP: ${ip} - Port: ${port}`);
          }
        }
      });
    });

    function addLog(message) {
      const timestamp = `[${new Date().toLocaleTimeString()}] `;
      logsDiv.innerHTML += `${timestamp}${message}\n`;

      // Verifica se a barra de rolagem estava no fundo antes de atualizar
      const shouldScrollToBottom = logsDiv.scrollHeight - logsDiv.scrollTop === logsDiv.clientHeight;

      if (logsDiv.childNodes.length > 100) {
        logsDiv.removeChild(logsDiv.firstChild);
      }

      // Se estava no fundo, mantém a rolagem lá
      if (shouldScrollToBottom) {
        logsDiv.scrollTop = logsDiv.scrollHeight;
      }
    }
  </script>
</body>
</html>
```

Trecho do nosso dashboard em html que exibirá no navegador todas as informações de conexões ativas, total de requisições, tentativas de conexões e os bloqueios do ataque informando o ip e porta que foram utilizadas.

```python
import socket
import threading
import time

def dos_attack(target_ip, target_port, message, delay=0):
    """Realiza um único ataque enviando mensagens para o servidor de destino."""
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:
            client.connect((target_ip, target_port))
            client.sendall(message.encode('utf-8'))
            print(f"Packet sent to {target_ip}:{target_port}")
            time.sleep(delay)  # Intervalo entre os envios, se configurado
    except Exception as e:
        print(f"Failed to connect to {target_ip}:{target_port} - {e}")


def start_attack(target_ip, target_port, message, num_threads=100, delay=0):
    """Inicia múltiplos ataques simultâneos usando threads."""
    threads = []
    for i in range(num_threads):
        t = threading.Thread(target=dos_attack, args=(target_ip, target_port, message, delay))
        t.start()
        threads.append(t)

    for t in threads:
        t.join()


if __name__ == "__main__":
    # Configurações do ataque
    target_ip = "127.0.0.1"  # IP do honeypot
    target_port = 8081        # Porta do honeypot
    message = "GET / HTTP/1.1\r\nHost: localhost\r\n\r\n"  # Payload básico HTTP
    num_threads = 50          # Número de threads simulando conexões simultâneas
    delay = 0.1               # Atraso entre os envios de mensagens em segundos

    print(f"Iniciando ataque DDoS contra {target_ip}:{target_port}...")
    start_attack(target_ip, target_port, message, num_threads, delay)
    print("Ataque concluído.")

```

Esse é o atacante que utilizamos que faz uma alta requisições massivamente para acessar o servidor.



Aqui está uma tela dele funcionando, com as conexões do honeypot e monitor e com o ataque já sendo executado e as informações retornando.

Todos os códigos do sistema estão disponíveis no GitHub do grupo SegInf.

Referências:

[1] https://lyceumonline.usf.edu.br/salavirtual/documentos/1661.pdf
[2]https://ransomware.org/how-to-prevent-ransomware/threat-hunting/honeypots-and-honeyfiles/
[3] https://www.honeyd.org/
[4] https://techblog.zrp.com.br/afinal-sistemas-distribuidos-o-que-sao/
[5] https://www.facom.ufu.br/~faina/BCC_Crs/GSI028-2014-1S/DL/DS-Ch01.pdf
[6] Honeypot Wood, Robert Stanleyhall, 2015
[7] Honeypots E Honeynets - Aprenda A Detectar E Enganar Invasores, Marcos Flávio Araújo Assunção, 2009