

# Taller DoS

Protocolos y agotamiento de recursos

SEIF, Jesús Blázquez | UAM

# Seguridad de la Información: CIA

Confidencialidad

Integridad

**Disponibilidad**



## Denial of Service (DoS)

“Cualquier evento que disminuya o elimine la capacidad de un sistema para realizar la función para la que fue diseñado.”

# El Concepto: DoS vs DDoS

## DoS (Denial of Service)

- 1 Atacante vs 1 Objetivo.
- Origen único.
- Fácil de mitigar (Bloqueo de IP / Firewall).

## DDoS (Distributed DoS)

- N Atacantes (Botnet) vs 1 Objetivo.
- Múltiples orígenes coordinados.
- Tráfico indistinguible del real.
- Difícil de mitigar en el origen.

INSERTAR IMAGEN  
(Esquema Botnet)

# El Principio de Asimetría


**Cliente (Tú)**

Generar Texto  
(HTTP Request)



Coste: Insignificante

**Servidor (Víctima)**

- 
- A white double arrow icon pointing to the right, indicating the flow of data or process from the client to the server.
1. Parsear TCP/HTTP
  2. Validar JSON
  3. Reservar Memoria (RAM)
  4. Ciclos de CPU (Lógica)
  5. I/O Wait (Disco/Red)

Coste: Exponencial

# El Modelo OSI

<b>L7</b>	<b>Aplicación</b>	HTTP, DNS
<b>L6</b>	<b>Presentación</b>	SSL/TLS
<b>L5</b>	<b>Sesión</b>	Sockets
<b>L4</b>	<b>Transporte</b>	TCP, UDP
<b>L3</b>	<b>Red</b>	IP, ICMP
<b>L2</b>	<b>Enlace</b>	MAC, ARP
<b>L1</b>	<b>Física</b>	Cable, WiFi

# Vectores de Ataque

## Volumétrico (L3/L4)

- **Objetivo:** Saturar el enlace.
- **Recurso:** Ancho de Banda.
- **Método:** UDP/SYN/ICMP Flood.

Descartado: Colapsa WiFi

## Agotamiento (L7)

- **Objetivo:** Procesamiento.
- **Recurso:** CPU, RAM, Workers.
- **Método:** Peticiones Complejas.

OBJETIVO DEL TALLER

**Asimetría:**  $\text{Coste(Atacante)} \ll \text{Coste(Objetivo)}$

# HTTP: Protocolo de Texto

Protocolo de comunicación sin estado basado en el intercambio de peticiones y respuestas entre cliente y servidor.

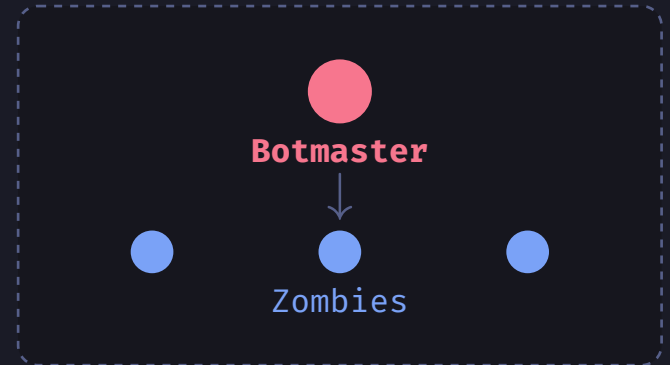
```
POST /api/v1/submit HTTP/1.1
Host: victim-server
Content-Type: application/json

{
  "query": "search_term",
  "limit": 100
} <- Body / Datos (Opcional)
```

# Botnets: Infraestructura Distribuida

## Definición

Red de dispositivos infectados (“Zombies”) operados por un actor central (“Botmaster”) sin el conocimiento de sus propietarios.



- **Amplificación:** 1 Orden  $\rightarrow$  N Ataques.
- **Anonimato:** El tráfico no viene del atacante.
- **Resiliencia:** Si un nodo cae, quedan miles.



# A Romper Cosas

```
user@botnet: $ python3 bot.py
```

```
⚠ Connecting to C&C Master...
```

```
✓ Connection Established.
```

```
user@botnet: $ _
```

Abrid vuestras terminales.

# Arquitectura del Objetivo

Conexión Exterior



**Linux Kernel (Host)**



**DOCKER CONTAINER**

NGINX



FLASK APP

Bare Metal / Host

htop

SSH

# Direcciones IP

**OBJETIVO (Víctima)**

**192.168.X.X**

**BOT (Zombie)**

**192.168.Y.Y**

Configurad estas IPs en bot.py

# Reglas de Compromiso

 **Por motivos educativos** 

1. Solo atacar la IP del laboratorio.
2. Prohibido atacar infraestructura de la Universidad.
3. No ataques volumétricos (UDP Flood) → Tiraréis el AP WiFi.

**Objetivo:** Entender la fragilidad para aprender a defenderla.

# Objetivo 1: Workers (Bloqueo)

## Vulnerabilidad: SSRF Síncrono

El endpoint `/monitor` usa `requests.get()` de forma síncrona.

Si le pedimos que conecte a una **API externa lenta**, el Worker de Gunicorn se queda “congelado” esperando la respuesta hasta el timeout.

**Estrategia:** Usar servicios públicos de delay para bloquear el worker sin gastar nuestra CPU.

### Payload:

```
GET /monitor?target=...  
HTTP/1.1
```

Target URL:

<https://httpbin.org/delay/5>

## Objetivo 2: CPU (Cálculo)

### Vulnerabilidad: Complejidad Algorítmica

El endpoint /pi utiliza el método de Monte Carlo para estimar  $\pi$ .

**Efecto:** El **Load Average** del servidor se dispara. Si supera el número de cores, todo el sistema se congela.

#### Payload:

**GET** /pi?iterations=...  
HTTP/1.1

#### Iterations:

**5000000**

(Cuanto más alto, más daño)

## Objetivo 3: RAM (Crash)

### Vulnerabilidad: Memory Leak Controlado

El endpoint `/allocations` reserva espacio en memoria con caducidad. Podemos “apilar” peticiones (Stacking) hasta llenar la memoria física.

**Efecto: OOM Killer.** Linux detecta el peligro y mata el proceso del servidor.

#### Payload:

**POST** `/allocations`

HTTP/1.1

Content-Type:  
application/json

```
{  
    "mb": 500  
}
```