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.”

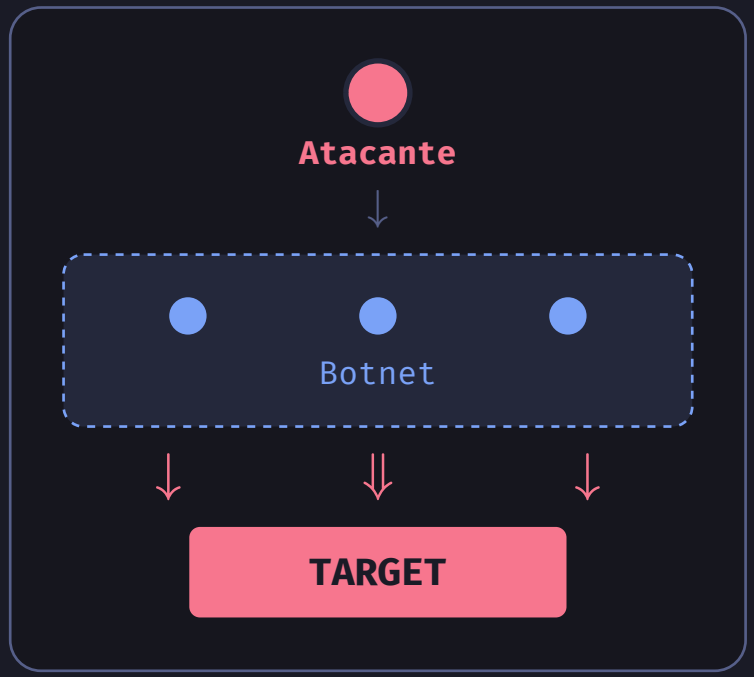
El Concepto: DoS vs DDoS

DoS (Denial of Service)

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

DDoS (Distributed DoS)

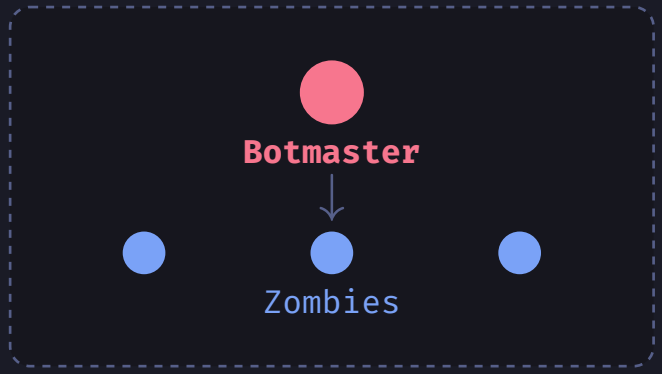
- N Atacantes (Botnet) vs 1 Objetivo.
- Múltiples orígenes coordinados.
- Tráfico indistinguible del real.
- Difícil de mitigar en el origen.



Botnets: Infraestructura Distribuida

Definición

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



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

El Principio de Asimetría

Cliente (Tú)

Generar Texto
(HTTP Request)

Coste: Insignificante

Servidor (Víctima)

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

L7	Aplicación	HTTP, DNS
L6	Presentación	SSL/TLS
L5	Sesión	Sockets
L4	Transporte	TCP, UDP
L3	Red	IP, ICMP
L2	Enlace	MAC, ARP
L1	Física	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: Coste(Atacante) $\ll$ 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)
```

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.

A Romper Cosas

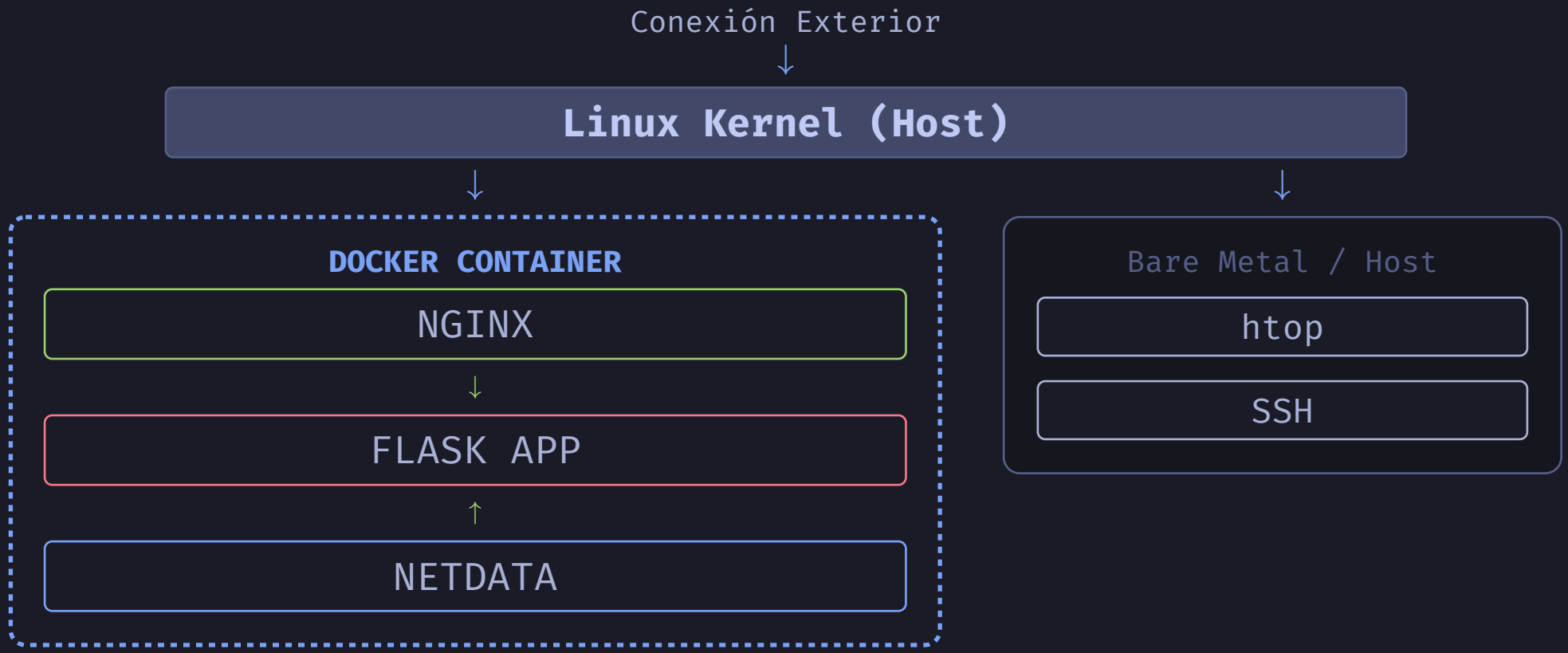
```
user@botnet: $ python3 bot.py

⚠️  Connecting to C&C Master…
✓ Connection Established.

user@botnet: $ _
```

Abrid vuestras terminales.

Arquitectura del Objetivo



Direcciones IP

OBJETIVO (Víctima)

192.168.X.X

BOT (Zombie)

192.168.Y.Y

Descargad los recursos en **http://192.168.X.X/static**

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 π.

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
}
```

Mitigación: Arquitectura Defensiva

Edge

- **CDN / Proxy:** Oculta tu IP Real para evitar ataques directos.
- **WAF:** Reglas automáticas contra bots.
- **Challenge:** Captchas/JS si detecta anomalías.

Gateway

- **Rate Limit:** Frena IPs agresivas (`limit_req`).
- **Caching:** Servir desde RAM evita tocar la CPU.
- **Timeouts:** Cortar conexiones lentas (Slowloris).

Application

- **Validación:** Rechazar números absurdos.
- **Async:** Tareas pesadas a colas (Redis), nunca bloquear.
- **Cuotas:** Límites duros de RAM por proceso.

<EOF />

Gracias por asistir.

SEIF | Jesús Blázquez

```
user@bot: $ shred -u history
> Traces wiped.

user@bot: $ ./questions.sh
? Waiting for input…

user@bot: $ exit
Session closed.
```