Network

DoS Attack (SYN Flood using Hping3) — Incident Response Report (SANS Framework)

Step #1: Preparation

In this phase, a controlled laboratory environment was prepared to simulate and analyze a Denial of Service (DoS) attack.

Environment Setup:

A web application was developed using C#, ASP.NET, HTML, and CSS, and hosted locally on a Windows machine using SQL Server 2022.

The website was assigned the hostname victim.lab.

Wireshark was installed on the Windows host to capture and inspect inbound traffic.

A Kali Linux virtual machine was used as the attacking system.

A Network Intrusion Detection System (NIDS), specifically Suricata, was deployed to monitor network activity and detect abnormal traffic behavior.

Purpose:

To evaluate the availability resilience of the web server and observe how SYN Flood DoS attacks are detected and handled in real time.

Step #2: Identification

The type of the incident is classified as:

Incident Type: Denial of Service (DoS) — SYN Flood Attack

Attacker Tool: hping3 generating high-rate TCP SYN packets

Detection Source: Suricata NIDS (via /var/log/suricata/fast.log)

Suricata Detection Logs (Proof):

```
11/10/2025-20:20:52.045728 [1:2210027:3] SURICATA STREAM ESTABLISHED SYN resend with different seq

{TCP} 192.168.56.3:40148 → 192.168.56.1:80

11/10/2025-20:20:52.045929 [1:2210044:2] SURICATA STREAM Packet with invalid timestamp

{TCP} 192.168.56.1:80 → 192.168.56.3:40148
```

These alerts indicate abnormal TCP behavior consistent with SYN Flooding (i.e., repeated SYN packets attempting to exhaust server resources and connection queues).

Step #3: Containment

During the experiment, the target website remained responsive, demonstrating that the Windows network stack and SYN cookie mechanism effectively mitigated the attack.

However, in a real production environment, containment actions should include:

Blocking the attacker's IP address at the firewall:

```
sudo iptables -A INPUT -s 192.168.56.3 -j DROP
```

Rate-limiting inbound TCP SYN requests at the server or router.

Logging and tracking timestamps for forensic and legal requirements.

Step #4: Eradication

Since this incident was strictly a network traffic-based DoS attack, no malware or persistent threat was introduced.

No files or system integrity were compromised.

The attacker’s IP address can be permanently blocked on the firewall to prevent repeated attacks.

Thus, eradication consisted of:

Permanent IP ban applied.

System integrity check completed (no modifications detected).

Step #5: Recovery

To ensure service stability after the attack:

The web server (IIS Express) was restarted to clear any stalled or half-open connections.

The website functionality and responsiveness were verified.

Network traffic returned to normal baseline levels.

Service Status: Fully Recovered and Operational.

Step #6: Lessons Learned

SYN Cookies are a critical defense mechanism and should remain enabled on all servers to protect against SYN Flood attacks.

Firewall / Router Filtering Policies should be configured to enforce:

Maximum 100 SYN packets per second per source IP.

Automatic temporary block for abnormal traffic rates.

DOS: Slowloris

Step #1: Preparation

In this experiment, a controlled lab environment was configured to analyze the impact of a Slow HTTP Headers (Slowloris) Denial of Service attack.

Environment Setup:

Web application hosted on Windows under the hostname victim.lab.

Database running on SQL Server 2022.

Kali Linux VM used as the attacking machine.

Suricata NIDS was used to monitor and detect abnormal HTTP header behavior.

Wireshark was used to capture packet traces for post-incident analysis (SlowLoris-after.pcapng).

Attack Tools Used:

```
sudo slowhttptest -c 1000 -H -g -u http://192.168.56.1
```

```
sudo slowloris 192.168.56.1 -p 80 -s 100000 --sleeptime 1 -v
```

Purpose: To evaluate the server’s ability to handle long-lasting half-open HTTP connections.

Step #2: Identification

Incident Type:

Application-Layer Denial of Service (Slow HTTP Headers / Slowloris)

Detection Source:

Suricata (/var/log/suricata/fast.log)

Suricata Alert Evidence (from screenshot):

```
11/10/2025-20:29:24.959430 [**] [1:2221035:1] SURICATA HTTP Request excessive header repetition [**]
```

```
[Classification: Generic Protocol Command Decode] [Priority: 3]
```

```
{TCP} 192.168.56.3:49862 → 192.168.56.1:80
```

Interpretation:

The attacker repeatedly opened HTTP sessions and sent headers slowly, causing header repetition alerts, which is a direct signature of Slowloris attacks.

Timeline:

Suricata triggered alert HTTP Request excessive header repetition.(20:29:24)

Web server slowed and struggled to respond as connections remained half-open.(20:29:30)

Containment actions applied.(20:29:50)

Step #3: Containment

Immediate Action Taken:

The attack was stopped by terminating the Slowloris processes:

```
sudo pkill -f slowloris
```

```
sudo pkill -f slowhttptest
```

Real-world recommended containment:

```
sudo iptables -A INPUT -s 192.168.56.3 -j DROP
```

This prevents the attacker from re-establishing new slow connections.

Step #4: Eradication

No malware or backdoor was introduced since the attack was volumetric at the application layer.

Actions taken:

Confirmed no malicious changes to web server files.

Attacker’s IP address was blocked to prevent reconnection.

System integrity verified.

Step #5: Recovery

To restore normal performance, the web service was restarted:

```
iisreset /restart
```

Result: Service responsiveness returned to normal and system resources stabilized.

Step #6: Lessons Learned

Slowloris targets connection handling logic, not bandwidth.

Mitigation recommendations:

Reduce HTTP Keep-Alive timeout.

Configure maximum concurrent connections per IP.

Deploy Reverse Proxy / WAF in front of the web application.

Enable Suricata rule to automatically block repeated slow header sessions:

```
drop http any any -> $HOME_NET 80 (msg:"Slowloris Detected"; http.header_names;
threshold:type both, track by_src, count 100, seconds 60; sid:1009002; rev:1;)
```

Incident Report — Man-in-the-Middle (ARP / DNS Spoofing)

SANS Incident Response Framework (Preparation → Identification → Containment → Eradication → Recovery → Lessons Learned)

Step 1 — Preparation:

A controlled laboratory network was prepared to evaluate the impact of local network deception and traffic interception. The lab topology consisted of multiple hosts on a single broadcast domain so that ARP poisoning was possible. The components and configuration were:

Victim host: Windows machine running a web browser (hostname victim.lab, IP 192.168.0.100 in the example).

Gateway / Router: LAN gateway IP 192.168.0.1.

Attacker host: Kali Linux virtual machine on the same network segment with packet-forwarding enabled as required for interception.

Monitoring host / capture: The attacking interface (eth0) had tcpdump running to capture packets. Suricata was running on the attacking/monitoring host to generate IDS alerts while ettercap was available for active interception and logging.

Tools used in the exercise (commands run during the test):

Enable IP forwarding on attacker:

```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

ARP poisoning (attacker → victim):

```
sudo arpspoof -i eth0 -t 192.168.0.100 192.168.0.1
```

ARP poisoning (attacker → gateway):

```
sudo arpspoof -i eth0 -t 192.168.0.1 192.168.0.100
```

DNS spoofing (optional):

```
sudo dnsspoof -i eth0 -f hosts.fake
```

Passive/active sniffing / manipulation:

```
sudo tcpdump -i eth0 -nn -A port 80
```

```
sudo ettercap -Tq -i eth0 -M arp:remote /192.168.0.1// /192.168.0.100//
```

Monitoring/IDS: Suricata running and logging to /var/log/suricata/ and packet captures saved to MTM-after.pcapng.

Objective: verify whether ARP table poisoning allows interception of victim↔gateway traffic, confirm Suricata detection of ARP spoofing, capture traffic for forensics and validate recommended mitigations.

Step 2 — Identification:

Incident type: Man-in-the-Middle (MITM) using ARP table poisoning, with optional DNS response manipulation.

Detection sources:

Suricata alerts (fast.log / eve.json).

tcpdump / MTM-after.pcapng capture showing ARP replies and forwarded streams.

ettercap session output showing ARP poisoning and sniffed flows.

Key evidence:

tcpdump / pcap excerpt (this is a direct example of the kind of lines you will extract from MTM-after.pcapng with tcpdump/tshark — keep them in your report as evidence):

```
21:16:50.034249 ARP, Reply 192.168.0.1 is-at 00:0f:3b:a1:d3:60 (oui Unknown), length 46
```

```
21:16:50.043269 IP 192.168.0.100.52568 > 146.190.62.39.80: Flags [P.], seq 595:1190, ack
766, win 253, length 595: HTTP: GET / HTTP/1.1
```

The first line shows an ARP reply advertising the gateway IP 192.168.0.1 at the attacker's MAC address (evidence of poisoning).

The second line shows an HTTP stream forwarded through the attacker (evidence of interception).

Ettercap / arpspoof output (example lines to include):

```
ARP poisoning victims:

  GROUP 1 : 192.168.0.1    B4:0F:3B:A1:D3:60

  GROUP 2 : 192.168.0.100 C4:75:AB:23:F1:95

Started Unified sniffing...
```

(These lines demonstrate the attacker advertising falsified ARP bindings; keep the exact lines/screenshots from your terminal.)

Indicators of compromise (IOCs):

Attacker MAC (as seen in ARP reply): 00:0f:3b:a1:d3:60 (example from your screenshot).

Attacker IP: 192.168.0.100 (example group member from your screenshot — use the real attacker IP recorded).

Gateway IP: 192.168.0.1 (spoofed by attacker).

Evidence files: MTM-after.pcapng, tcpdump_eth0.log, ettercap_session.log, Suricata eve.json extracts.

Interpretation: The presence of ARP reply packets that map the gateway IP to the attacker MAC together with forwarded HTTP traffic in the capture demonstrates a successful ARP table poisoning and an on-path interception of connections.

Step 3 — Containment

Containment goal: Stop further ARP poisoning and prevent the attacker from continuing to intercept traffic while preserving forensic evidence.

Immediate containment actions performed in the lab (and to document in the report):

Disconnect the attacker host from the network.

Action recorded with timestamp: power off / unplug attacker VM network adapter (document exact time).

Clear the victim ARP cache and reset its network interface so the victim issues fresh ARP requests to the gateway:

On Windows victim: arp -d * and then disable/enable the network adapter (document times).

Validate restoration of correct mappings by running arp -a on the victim and confirming the gateway maps to the gateway MAC.

Preserve evidence: do not delete capture files; copy MTM-after.pcapng, tcpdump outputs, Suricata logs, and ettercap logs to a secure evidence directory and calculate checksums (SHA-256). Example:

```
mkdir -p /evidence/mtm/2025-11-11

cp MTM-after.pcapng /evidence/mtm/2025-11-11/

sha256sum /evidence/mtm/2025-11-11/MTM-after.pcapng > /evidence/mtm/2025-11-11/MTM-after.pcapng.sha256
```

What to record in the report: exact timestamps for (a) when the attack was detected, (b) when the attacker host was disconnected, (c) when the victim ARP cache was cleared, and (d) file names/hashes of preserved evidence.

Step 4 — Eradication

Eradication goal: Remove the attacker's ability to repeat the attack and ensure no persistent artifacts remain.

Actions taken and to describe in the report:

Revert attacker configuration: If the attacker had made persistent changes (for example, scripts scheduled in cron), remove them. In this lab the attacker was temporary and was powered off; nonetheless record that the attacker VM was examined and any malicious scripts or persistent services were removed.

Remove forged DNS entries: if dnsspoof was used with a hosts file, ensure the DNS spoof file is removed from the attacker and the network.

Switch / router hardening (if simulated): apply static ARP entries for critical nodes (gateway, servers) where possible, or apply port security to prevent the same MAC address appearing on unexpected ports.

Network service checks: scan victim and gateway for unexpected processes or files; document ps, netstat/ss outputs confirming no unauthorized services remain.

Evidence of eradication: show arp -a output and tshark/tcpdump excerpts after remediation that demonstrate no further ARP reply packets advertising the attacker MAC.

Step 5 — Recovery

Recovery goal: Restore normal network operation and verify integrity and functionality.

Recovery steps taken and recorded:

Network verification:

Run two-way ping tests between victim and gateway.

Run arp -a on victim and gateway to confirm correct IP→MAC mappings.

Run traceroute to confirm traffic flows correctly to the gateway.

Service verification:

Confirm all application services on the victim are reachable and functioning.

Monitor network performance and logs for a period (example: 30 minutes) to ensure no recurring poisoning attempts.

Monitoring check:

Ensure Suricata is running and that the alert thresholds are tuned. Include a short period of post-recovery capture showing no further malicious ARP replies.

Document for the report: the verification commands, timestamps, and sample outputs (for example, ping, arp -a, and a brief excerpt from tshark showing normal traffic).

Step 6 — Lessons Learned and Recommendations

Summary lesson: Layer-2 network segments are susceptible to ARP spoofing when no link-layer protections are in place; detection and rapid containment depend on monitoring and switch-level controls.

Technical recommendations (actionable, prioritized):

Enable Dynamic ARP Inspection (DAI) on managed switches to block invalid ARP replies. DAI validates ARP packets against DHCP snooping binding tables and prevents unauthorized ARP entries from propagating.

Enable Port Security on switches to restrict the number of MAC addresses per port and to lock down ports to specific MAC addresses where practical.

Enforce Static ARP Entries for Critical Hosts (gateways, DNS servers, controllers) where possible — particularly for lab and production servers.

Segment the Network and Use Private VLANs to reduce the broadcast domain size and limit the effectiveness of ARP poisoning.

Prioritize and Tune NIDS Rules: ensure Suricata/IDS rules for ARP poisoning and suspicious DNS responses are active and set escalation actions (e.g., notify SOC, generate automatic isolations for confirmed spoofing).

Use End-to-End Encryption: ensure application traffic uses TLS so that even if traffic is intercepted, the content remains protected.

Implement Detection→Response Playbook: a short, tested runbook for ARP poisoning that states who disconnects the attacker, who clears ARP caches, how evidence is preserved, and how to verify recovery.

brute Force Authentication Attack and DOS using Burp-Suite and ZAPROXY

SANS Incident Response Framework

Step #1: Preparation

A controlled test environment was prepared to evaluate the security robustness of a C#/ASP.NET web application hosted on a Windows machine running IIS Express and SQL Server 2022 (victim.lab). The attacker machine was a Kali Linux virtual machine located within the same network.

Attack Tools Used:

Burp Suite Intruder — to conduct a structured brute force attack against the login interface.

OWASP ZAP (Fuzzer) — to generate high-volume requests that caused application resource exhaustion, leading to a Denial of Service condition.

Defense and Monitoring Tools:

Suricata NIDS — configured to detect authentication request spikes and abnormal HTTP request patterns.

Wireshark — used to analyze HTTP request and response behavior during the attack.

Burp Suite Brute Force Procedure Performed:

Open Firefox and configure a manual proxy:

HTTP Proxy: 127.0.0.1, Port: 8080

Enable “Use this proxy for HTTPS”.

Open the target login page and submit a known invalid password (e.g., fakepass).

In Burp Suite → Proxy → Intercept, right-click the captured request and select Send to Intruder.

Open Intruder → Clear positions → Select only the password field → Click ADD.

Open Payloads tab → Set Payload type to Simple List.

Load a password wordlist into Payload Configuration.

Open Grep - Match → Clear → Add the failure message string displayed by the application.

Start the attack and analyze differing responses to detect the correct password.

Purpose:

To determine whether authentication controls lacked brute-force protection and to assess application resilience under high request load.

Step #2: Identification (Detection)

Incident Type:

Credential Brute Force Attack followed by Application Layer Denial of Service.

Detection Sources:

Suricata Alerts: Signatures matched high-frequency login attempts consistent with automated password guessing behavior.

Burp Suite Output: A password (1111) was identified when the application response content differed from failure responses.

OWASP ZAP Observation: High thread-count fuzzing caused multiple 500 Internal Server Error responses, indicating application instability and connection pool exhaustion.

Evidence:

Suricata log entries indicating repeated POST requests from the attacker IP.

Login success occurring without account lockout.

Server rendered unresponsive under ZAP load.

Step #3: Containment

Containment actions were taken to stop both credential abuse and service degradation.

The attacker’s IP address (192.168.56.x) was blocked using the Windows firewall to prevent continued requests.

IIS Express and SQL Server were restarted to restore service availability and clear resource locks caused by the application crash.

Step #4: Eradication

The compromised password (1111) was changed to a secure alternative to invalidate attacker access.

Application input handling and error conditions were analyzed to identify the root cause of the crash.

The login handling function must be revised with:

Server-side validation

Exception handling

Rate throttling logic

No malware or persistence mechanisms were identified, as the attack was non-intrusive and resource-focused.

Step #5: Recovery

After service restart, the application returned to normal operational state.

Authentication was validated to confirm only correct credentials were accepted.

Normal response times were verified.

Recovery Status: Application stable and operating normally.

Step #6: Lessons Learned

This incident revealed several weaknesses in the application-layer security posture that allowed the attacker to both brute-force credentials and cause an application-level denial of service. First, there is no effective brute-force detection or account lockout mechanism in place. To mitigate this, implement a configurable account lockout policy that temporarily locks an account after a defined number of failed authentication attempts (for example, lock for 15 minutes after 5 failed attempts). Log both lockout events and the reasons for failed logins so they can be audited.

Second, the application allows unlimited login attempts from the same source IP, which enables automated credential-guessing tools to run at high speed. Enforce per-IP rate limiting at the application layer (for example, maximum 5 login attempts per minute per IP) and complement this with upstream controls (WAF or reverse proxy rate limiting) to reduce abusive traffic before it reaches the application.

Third, the server crashed when presented with certain malformed or unexpected input, demonstrating insufficient server-side input validation and error handling. Remedy this by implementing strict server-side validation for all input fields, applying whitelisting where feasible, and adding robust exception handling that fails safely. Ensure that unhandled exceptions do not return stack traces or internal error details to clients, and include defensive coding and fuzz testing in the development lifecycle to discover crash-causing inputs before production deployment.

Fourth, although the NIDS (Suricata) detected suspicious behavior, there was no automated or semi-automated response to stop the attack in real time. Tune the NIDS rules to reduce false positives and configure an automated or operator-assisted response (for example, integration with a firewall or orchestration tool to apply temporary blocks) for high-confidence detections. In addition, improve alerting so that high-severity events trigger immediate operational workflows.

Finally, reinforce these technical controls with operational measures: enable comprehensive authentication logging and centralized log collection, enforce strong password policies and multi-factor authentication for user accounts, and run regular security tests (rate-limit testing, brute-force simulations, fuzzing) against staging environments. Document a clear incident playbook that defines thresholds, responsibilities, and the containment steps to be followed when similar behavior is detected.