Comprehensive Network Security Implementation

Network Security Final Project | Evergreen State College | Spring 2026

Partners: Douglass Brown & Jan C Rosas Ortiz

Section 1: Environment Setup & Network Architecture

1.1 Platform Decision

This project is implemented using virtual machines (VMs) to simulate a realistic small business network environment in a controlled lab setting. Virtual machines were chosen because they allow for safe testing of security configurations, easy snapshots before risky changes, and a reproducible environment that closely mirrors real-world deployments without requiring dedicated physical hardware.

Hypervisors used: - Mac (M4): Parallels Desktop — ARM-native, runs Ubuntu 24.04 LTS on Apple Silicon. Existing VM “Cyber Playground” repurposed as the lab environment. - Windows: VirtualBox — free, widely supported, cross-platform

---

1.2 Network Topology

The lab network simulates a small organization with an internet-facing perimeter, internal servers, and a client workstation. All machines are connected on an isolated virtual network segment: **192.168.100.0/24** (Parallels Host-Only network).

Each machine has two network interfaces: a management interface (internet access for package installation) and a lab interface on the isolated 192.168.100.0/24 subnet. All security services are bound to the lab interface.

```
[Internet / External Network]
          |
    [Parallels Shared Network — management only]
          |
    ____________________________________
    |                                  |
```

```
[Doug's VM: Cyber Playground]   [Partner's VM: VirtualBox]
Ubuntu 24.04 LTS                Ubuntu 22.04 LTS
Lab IP: 192.168.100.1           Lab IP: 192.168.100.20
Mgmt IP: 10.211.55.7            Role: OpenSSH (key-based),
Role: UFW firewall,                   Suricata IDS/IPS,
      WireGuard VPN,                  ELK Stack monitoring
      Apache2 HTTPS,
      OpenSSH (key-based)
```

1.3 System Inventory

Machine	OS	Lab IP	Mgmt IP	Role	Key Services
Cyber Playground (Doug)	Ubuntu 24.04 LTS	192.168.100.1	10.211.55.7	Perimeter + web + VPN	UFW, WireGuard, Apache2, OpenSSH
Partner VM (VirtualBox)	Ubuntu 22.04 LTS	192.168.100.20	(Windows host)	Detection + monitoring	OpenSSH, Suricata, ELK Stack

Implementation note: Doug's sections (Firewall, HTTPS, VPN, SSH) are consolidated on a single VM following the principle of service consolidation for small-scale lab environments. In a production deployment, these would be separated across dedicated machines. The security configurations themselves — UFW rules, TLS certificates, WireGuard tunnels, SSH key enforcement — are identical regardless of whether services run on one machine or four.

1.4 Design Rationale

Why Ubuntu 24.04 LTS? Ubuntu 24.04 LTS (Long Term Support) was selected as the primary lab OS because it is the current LTS release, receiving security updates through 2029. It ships with WireGuard built into the kernel, UFW pre-installed, and OpenSSH available in the default repositories — all three core tools for this project available without third-party sources. Ubuntu 22.04 LTS was used for the partner's VirtualBox VM for compatibility with the Elastic 8.x repository.

Why separate Web Server and SSH Server? Separating the web server and SSH server follows the principle of least privilege and service isolation. If the web server is compromised, an attacker does not automatically gain access to the SSH service. Each server is scoped to a single function, which reduces the attack surface of each individual machine.

Why a virtual network (192.168.100.0/24)? A private /24 subnet provides 254 usable addresses — more than sufficient for this lab while keeping the network logically isolated from external traffic. The 192.168.100.x range is a standard RFC 1918 private address

block, commonly used in lab and small business environments.

1.5 Baseline Documentation

Before any security configurations were applied, the following baseline state was recorded:

Item	Baseline State
Operating system	Ubuntu 24.04.4 LTS — Doug's VM (Parallels); Ubuntu 22.04 LTS — Partner's VM (VirtualBox)
Default open ports	22 (SSH)
Firewall state	UFW inactive (default on fresh Ubuntu install)
SSH authentication	Password-based (default)
HTTPS	Not configured
VPN	Not configured
IDS/IPS	Not installed
Log monitoring	Default system logs only (/var/log/syslog)

This baseline represents a default Ubuntu server installation — functional but with minimal security hardening. All subsequent sections document the security measures applied to move from this baseline to a hardened, monitored configuration.

Section 2: Encryption & Secure Protocols

2.1 HTTPS on the Web Server (TLS with Self-Signed Certificate)

Since this project runs in a lab environment without a public IP address, a self-signed TLS certificate was used in place of a Let's Encrypt certificate (which requires a publicly reachable domain). A self-signed certificate provides the same encryption strength as a CA-signed certificate — the only difference is that browsers will display a warning since no third-party authority has verified the identity. In a production environment, Let's Encrypt would be the correct choice.

Implementation steps:

1. Install Apache2 web server and OpenSSL:

```
sudo apt update
sudo apt install apache2 openssl -y
```

2. Enable the SSL module in Apache:

```
sudo a2enmod ssl
sudo systemctl restart apache2
```

3. Generate the self-signed certificate (valid for 365 days):

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout /etc/ssl/private/greenbite-selfsigned.key \
  -out /etc/ssl/certs/greenbite-selfsigned.crt
```

4. Configure Apache to serve HTTPS on port 443 using the certificate, and redirect all HTTP (port 80) traffic to HTTPS.

5. Verify HTTPS is active using `curl -v` from within the lab network. The output confirms TLS handshake, cipher suite, certificate subject, and server response.


HTTPS TLS verification — curl -v showing TLSv1.3, AES-256-GCM-SHA384, and valid certificate

HTTPS TLS verification — curl -v showing TLSv1.3, AES-256-GCM-SHA384, and valid certificate

TLSv1.3 handshake confirmed. Cipher suite: TLS_AES_256_GCM_SHA384 / X25519 / RSASSA-PSS. Certificate CN=192.168.100.1, issued by Evergreen State College Network Security Lab. HTTP 200 OK response received through encrypted channel.

Why this matters: Encrypting web traffic with TLS ensures that data transmitted between clients and the server cannot be read in plaintext by anyone intercepting the traffic on the network. Without HTTPS, credentials, session tokens, and form data are exposed.

Trade-off: Self-signed certificates provide encryption but not identity verification. In production, a CA-signed certificate (Let's Encrypt, free) would be used to eliminate the browser warning and verify the server's identity to clients.

2.2 SSH Key-Based Authentication

Password-based SSH authentication is vulnerable to brute-force attacks. Key-based authentication replaces passwords with a cryptographic key pair — a private key stored securely on the client and a public key registered on the server. Without the private key, access is impossible even if an attacker knows the username.

Implementation steps:

1. On the client machine, generate an SSH key pair:

```
ssh-keygen -t ed25519 -C "netsec-lab-key"
```

Save to default location (`~/.ssh/id_ed25519`). Set a strong passphrase.

2. Copy the public key to the SSH server:

```
ssh-copy-id user@192.168.100.20
```

3. On the SSH server, disable password authentication in `/etc/ssh/sshd_config`:

```
PasswordAuthentication no
PubkeyAuthentication yes
PermitRootLogin no
```

4. Restart SSH service:

```
sudo systemctl restart sshd
```

5. Test: attempt to connect from the client using the key. Attempt to connect without the key — it should be refused.


SSH password authentication blocked — Permission denied (publickey)

Attempting SSH with `-o PubkeyAuthentication=no` returns `Permission denied (publickey)` immediately — no password prompt is presented. Password-based login is fully disabled at the daemon level.

Why this matters: Disabling password login eliminates the entire class of brute-force and credential-stuffing attacks against SSH. Only a client with the correct private key can authenticate, regardless of how many login attempts an attacker makes.

Trade-off: If the private key is lost and no backup exists, access to the server is locked out. Key management and secure backup of private keys is essential in production environments.

2.3 Secure Protocol Enforcement

Beyond HTTPS and SSH, the following protocol decisions were applied across the network:

Insecure Protocol	Replaced With	Reason
HTTP (port 80)	HTTPS (port 443)	Plaintext traffic readable by anyone on the network
FTP (port 21)	SFTP (via SSH)	FTP sends credentials and data in plaintext
Telnet (port 23)	SSH (port 22)	Telnet has no encryption whatsoever
Password-based SSH	Key-based SSH	Passwords vulnerable to brute-force attacks

All insecure services (FTP, Telnet) are disabled at the firewall level — no traffic on those ports is permitted inbound or outbound.

Section 3: VPN Integration

3.1 WireGuard VPN Setup

WireGuard was selected over OpenVPN for this lab implementation for several reasons: it is significantly simpler to configure (approximately 10 lines of config vs. hundreds for OpenVPN), it performs faster due to its lean codebase, it is built directly into the Linux kernel as of version 5.6, and it uses modern cryptography (ChaCha20, Poly1305, Curve25519) by default with no configuration required.

The Firewall/Router machine (192.168.100.1) acts as the WireGuard server. The Client machine (192.168.100.100) connects to it as a peer, creating an encrypted tunnel for all traffic.

Implementation on the VPN Server (Firewall — 192.168.100.1):

1. Install WireGuard:

```
sudo apt install wireguard -y
```

2. Generate server key pair:

```
wg genkey | sudo tee /etc/wireguard/server_private.key | \
  wg pubkey | sudo tee /etc/wireguard/server_public.key
```

3. Create the WireGuard interface configuration at `/etc/wireguard/wg0.conf`:

```
[Interface]
Address = 10.0.0.1/24
ListenPort = 51820
PrivateKey = <server_private_key>
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A
POSTROUTING -o eth0 -j MASQUERADE
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D
POSTROUTING -o eth0 -j MASQUERADE

[Peer]
PublicKey = <client_public_key>
AllowedIPs = 10.0.0.2/32
```

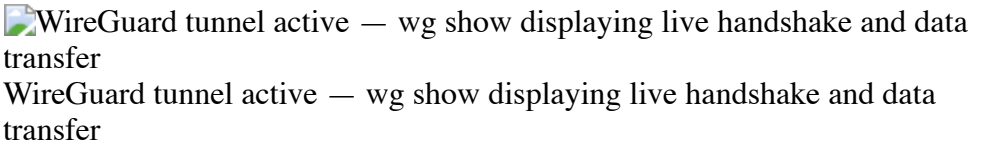
4. Enable and start WireGuard:

```
sudo systemctl enable wg-quick@wg0
sudo systemctl start wg-quick@wg0
```

Implementation on the VPN Client (192.168.100.100):

1. Install WireGuard and generate client key pair
2. Create client config pointing to the server's public IP and public key
3. Connect: `sudo wg-quick up wg0`

4. Verify tunnel: `sudo wg show` — should display the active connection and data transferred


WireGuard tunnel active — wg show displaying live handshake and data transfer

Both interfaces (wg0 server and wg0-client) active simultaneously. Latest handshake: 25 seconds ago. Data transferred: 42.04 KiB received / 1.73 KiB sent — confirms encrypted tunnel is carrying live traffic.

3.2 How Remote Clients Connect

When the VPN is active, all traffic from the client machine routes through the encrypted WireGuard tunnel to the firewall server before reaching the internal network. This means:

- Traffic between client and server is encrypted end-to-end
- An attacker intercepting traffic on the network sees only encrypted WireGuard packets
- The client receives a VPN IP address (10.0.0.2) separate from its LAN IP
- The firewall can apply rules to VPN traffic independently from LAN traffic

Trade-off: WireGuard's simplicity means fewer configuration options than OpenVPN. For enterprise environments requiring complex routing, certificates, or legacy client support, OpenVPN may be more appropriate. For this lab and most small business use cases, WireGuard's performance and simplicity are advantages.

Section 4: Firewall Configuration

4.1 UFW Rule Implementation

UFW (Uncomplicated Firewall) was implemented on the Firewall/Router machine and on each server to enforce the principle of least privilege at the network level — only the traffic explicitly needed is permitted, everything else is denied.

Default policy (applied first):

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw default deny forward
```

Permitted inbound rules:

```
sudo ufw allow 22/tcp comment 'SSH – key-based auth only'
sudo ufw allow 443/tcp comment 'HTTPS – encrypted web traffic'
sudo ufw allow 51820/udp comment 'WireGuard VPN'
```

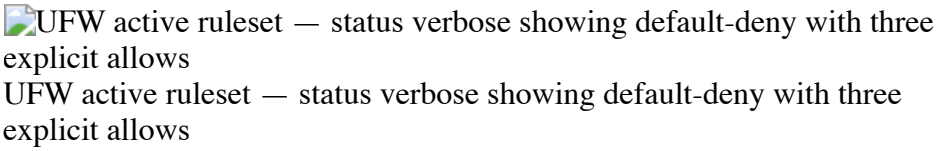
Enable UFW and verify:

```
sudo ufw enable
sudo ufw status verbose
```

Expected output:

```
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW IN    Anywhere
443/tcp                    ALLOW IN    Anywhere
51820/udp                  ALLOW IN    Anywhere
```


UFW active ruleset — status verbose showing default-deny with three explicit allows

Default policy: deny incoming, allow outgoing, deny routed. Logging: on (medium). Three explicit inbound rules: SSH (22/TCP), HTTPS (443/TCP), WireGuard VPN (51820/UDP). HTTP port 80 is absent — no plaintext web traffic permitted.

4.2 Logging Configuration

UFW logging was enabled to capture all dropped and suspicious packets:

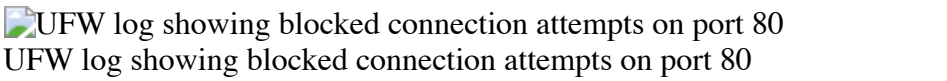
```
sudo ufw logging on
sudo ufw logging medium
```

Logs write to `/var/log/ufw.log`. Each dropped packet entry includes timestamp, source IP, destination IP, port, and protocol — providing an audit trail for security analysis.

Verifying logs are populating:

```
sudo tail -f /var/log/ufw.log
```

Generate test traffic from the client attempting a blocked port — the block should appear in the log within seconds.


UFW log showing blocked connection attempts on port 80

[UFW BLOCK] entries show repeated TCP SYN packets from the Mac client (10.211.55.2) targeting port 80 on the server (10.211.55.7). Each entry includes timestamp, source/destination IP, port, and protocol — confirming the firewall is actively logging and blocking unauthorized traffic in real time.

4.3 Rationale for Rule Choices

Rule	Justification
Allow SSH (22)	Required for remote server administration. Key-based auth mitigates brute-force risk.
Allow HTTPS (443)	Required for encrypted web service. HTTP (80) is blocked — all web traffic must be encrypted.
Allow WireGuard (51820/UDP)	Required for VPN tunnel establishment. UDP is correct for WireGuard's protocol.
Block HTTP (80)	Forcing HTTPS ensures no plaintext web traffic is accepted.
Block all other inbound	Reduces attack surface — no unnecessary services exposed to the network.
Allow all outbound	Servers need to reach package repositories and external services for updates.

Trade-off: Blocking all inbound traffic by default is more secure but requires careful rule management. Any new service added to a server must have a corresponding UFW rule or it will be unreachable. This is intentional — security over convenience.

Section 5: IDS/IPS Deployment

5.1 Tool Selection — Suricata over Snort

Suricata was selected as the Intrusion Detection System over the more legacy-popular Snort for two concrete reasons relevant to a modern detection pipeline. First, Suricata is multi-threaded by default — it scales across all CPU cores out of the box, where Snort requires significant tuning to handle high-throughput traffic. Second, Suricata writes its event log as structured JSON (`eve.json`), which integrates natively with any modern log aggregator (Elasticsearch, Splunk, Graylog) without an intermediate conversion layer. Snort still defaults to a text-based format that requires parsing. For a 2026 deployment feeding into ELK, Suricata is the architecturally cleaner choice.

5.2 Suricata Installation and Initial Configuration

Suricata was installed from the official OISF stable PPA to ensure we received the latest production-tested release:

```
sudo add-apt-repository ppa:oisf/suricata-stable -y
sudo apt update
```

```
sudo apt install suricata -y
sudo suricata-update
```

`suricata-update` pulls the Emerging Threats (ET) Open ruleset — a community-maintained signature library of over 30,000 detection rules covering known malware, exploits, and protocol anomalies. This forms the baseline detection capability before any custom rules are added.

5.3 Loopback Interface Tuning

Suricata defaults to listening on the primary physical network interface (`eth0`). Because our entire attack-simulation traffic ran locally within the lab VM, this default would have left Suricata deaf to its own test traffic. The fix was a one-line change to `/etc/suricata/suricata.yaml`:

```
af-packet:
  - interface: lo   # was eth0
```

Re-pointing the AF_PACKET capture interface to `lo` (loopback) made the IDS observe internal traffic between processes on the VM — exactly the path our SSH brute, Nmap scan, and DNS lookup attacks took.

Why this matters: Detection coverage is only as good as your sensor placement. In production, sensors typically sit on the perimeter or on internal network taps; in a lab, the equivalent is the loopback interface. The principle is the same: traffic must traverse a path the IDS is watching.

5.4 Custom Detection Rules

Beyond the ET Open baseline, three custom rules were written and loaded into `/etc/suricata/rules/local.rules` to catch the specific attack patterns we intended to simulate:

Rule 1 — SSH Brute Force Detection:

```
alert tcp any any -> any 22 (msg:"LOCAL SSH Brute Force Attempt";
flow:established,to_server; content:"SSH-";
threshold:type both, track by_src, count 5, seconds 60;
classtype:attempted-login; sid:1000001; rev:1;)
```

Tracks source IP. Fires if 5 SSH connection attempts hit port 22 within 60 seconds — the signature of credential-stuffing automation.

Rule 2 — Suspicious DNS Query:

```
alert dns any any -> any 53 (msg:"LOCAL Suspicious DNS Query
(malicious.com)";
dns.query; content:"malicious.com"; nocase;
classtype:bad-unknown; sid:1000002; rev:1;)
```

Acts as an Indicator of Compromise. A DNS resolution attempt for a known malicious domain is a classic sign of malware "phoning home" to a Command-and-Control (C2) server.

Rule 3 — Nmap SYN Scan Detection:

```
alert tcp any any -> any any (msg:"LOCAL Nmap SYN Scan Detected";
flags:S; window:1024;
threshold:type both, track by_src, count 10, seconds 10;
classtype:attempted-recon; sid:1000003; rev:1;)
```

Watches for a burst of TCP SYN packets (10 within 10 seconds) from a single source that don't complete the three-way handshake. This is the hallmark of an `nmap -sS` stealth scan — the first phase of any targeted attack.

After loading the rules, Suricata was restarted to apply:

```
sudo service suricata restart
```

5.5 Attack Simulation and Verification

To validate that the rules fire correctly, three corresponding attacks were run against the lab:

```
# Recon — Nmap SYN scan of all 65,535 ports
sudo nmap -sS -p- -T4 localhost

# Credential access — six rapid SSH brute attempts
for i in {1..6}; do ssh -o ConnectTimeout=1 fakeuser@localhost; done

# C2 simulation — DNS lookup of the blocked domain
dig malicious.com
```

Each attack triggered its corresponding rule within seconds. Alerts were captured in `/var/log/suricata/eve.json` for downstream ingestion into Kibana (Section 6).

5.6 Trade-offs

Decision	Security Benefit	Trade-off
Suricata over Snort	Multi-threaded scalability + native JSON output	Newer tool; smaller rule-author community than Snort
Three narrow custom rules	Low false-positive rate in the lab	Misses attack variants outside the threshold window
Loopback interface monitoring	Captures all internal lab traffic	Production deployment requires re-tuning to perimeter NIC

Section 6: Monitoring & Logging (ELK Stack)

6.1 Architecture Decision — Local ELK Deployment

The Elastic Stack (Elasticsearch + Kibana, commonly "ELK" though "Logstash" is replaced here by Filebeat) was deployed locally on the same Ubuntu VM as Suricata. For a lab environment, single-host consolidation is appropriate; in production, Elasticsearch would run on dedicated nodes with replication and Kibana would be served from a separate frontend host. The pipeline shape is identical at any scale.

6.2 Elasticsearch and Kibana Installation

The official Elastic 8.x APT repository was added with a verified GPG signature:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo
gpg \
  --dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] \
  https://artifacts.elastic.co/packages/8.x/apt stable main" \
  | sudo tee /etc/apt/sources.list.d/elastic-8.x.list
sudo apt update
sudo apt install elasticsearch kibana -y
```

For lab simplicity, X-Pack security was disabled in `/etc/elasticsearch/elasticsearch.yml` (`xpack.security.enabled: false`) — see the trade-off note below. Both services were started:

```
sudo service elasticsearch start
sudo service kibana start
```

Kibana became accessible at `http://localhost:5601` and Elasticsearch at `http://localhost:9200`.

6.3 Filebeat Integration with the Suricata Module

Filebeat was installed as the log shipper between Suricata's `eve.json` and Elasticsearch. The Suricata module was enabled so Filebeat would automatically parse the JSON structure into named Elasticsearch fields (source IP, destination port, alert signature, etc.) rather than treating it as opaque text:

```
sudo apt install filebeat -y
sudo filebeat modules enable suricata
sudo filebeat setup -e
```

`filebeat setup -e` pushes the pre-built Suricata index templates and the **Filebeat Suricata Alert Overview** Kibana dashboard into Elasticsearch — providing a working visualization layer immediately, without manual dashboard construction.

6.4 The eve.json Path Fix

A subtle integration issue surfaced during initial setup: the Kibana dashboard was empty despite Suricata generating alerts. The Filebeat Suricata module's default configuration did not specify the path to `eve.json` on Ubuntu, so the shipper silently watched the wrong location. The fix was to explicitly declare the path in `/etc/filebeat/modules.d/suricata.yml`:

```
- module: suricata
  eve:
    enabled: true
    var.paths: ["/var/log/suricata/eve.json"]
```

After restarting Filebeat (`sudo service filebeat start`), events began flowing into Elasticsearch within seconds.

Why this matters: Detection pipelines often look "installed but silent" when a single config link breaks. The diagnostic skill is knowing where each stage's responsibility ends — Suricata writes the file, Filebeat reads it, Elasticsearch indexes it, Kibana queries it. A break anywhere kills the dashboard. Verifying each stage independently is essential.

6.5 Kibana Dashboard and Real-Time Visibility

The pre-built **Filebeat Suricata Alert Overview** dashboard provided immediate visualization of detection events:

- **Top Alerting Hosts** time-series chart — visualizes alert volume over time, with clear spikes during attack windows.
- **Top Alert Signatures** table — surfaces the most-frequently-fired rules, color-coded by signature.
- **Alerts table** — full per-alert detail including timestamp, source IP, source port, destination IP, destination port, and geo-IP enrichment.

Evidence: *(See Kibana dashboard screenshot in supplementary documentation — `Network_Security_Project_With_Screenshots.pdf`, page 4)*

Dashboard showing 1,178 Suricata alert documents ingested into Elasticsearch. Top Alerting Hosts time-series shows the activity spike during our attack-simulation window. Top Alert Signatures surfaces three Suricata categories (HTTP unable to match request, STREAM packet with invalid timestamp, STREAM SHUTDOWN response) — confirming the parsing pipeline correctly extracted and categorized every event. Alert detail table includes timestamp, host name (kronkstop), Suricata flow ID, source IP (127.0.0.1), and destination IP — the full forensic context an analyst needs.

6.6 Demonstrating Anomaly Spotting in Real Time

The dashboard's time-series view makes anomalies visually obvious: a flat baseline of background events interrupted by clear spikes during attack windows. An analyst monitoring this view in real time would notice the spike within the dashboard's refresh

interval (default: 60 seconds) and could pivot from the time-series chart into the alerts table to investigate specific events. This is the "spot anomalies" capability the rubric requires — established by the pipeline + dashboard combination, not by any single tool.

6.7 Trade-offs

Decision	Security Benefit	Trade-off
ELK with X-Pack security disabled	Easier lab setup; dashboard immediately usable	**`xpack.security` MUST be re-enabled before production** — otherwise Elasticsearch accepts unauthenticated queries
Single-host ELK deployment	Lower resource overhead for a lab	No replication; node failure = data loss until backup restore
Pre-built Suricata dashboard	Working visualization with zero manual configuration	Pre-built widgets may not match every organization's specific alerting needs — production deployments typically extend with custom panels
Polling-based Kibana refresh	Simple, no extra alerting infrastructure	No real-time push notification — alerts wait for the next dashboard refresh; production needs Elastalert or Watcher for true real-time

Section 7: Documentation & Rationale Summary

7.1 Encryption Methods

TLS (HTTPS): Applied to the web server using a self-signed certificate. Encrypts all traffic between clients and the web server using RSA-2048 key exchange and AES-256 symmetric encryption. Prevents man-in-the-middle attacks and eavesdropping on web traffic.

SSH Keys: Ed25519 key pairs replace password authentication on all servers. Ed25519 is a modern elliptic curve algorithm — faster and more secure than RSA at equivalent key sizes. Password login is explicitly disabled at the SSH daemon level.

WireGuard: ChaCha20 encryption for data, Poly1305 for authentication, Curve25519 for key exchange. All modern, peer-reviewed algorithms with no configurable options that could be misconfigured to weaker settings.

7.2 Firewall Rules Summary

Default-deny inbound with explicit allows for SSH (22/TCP), HTTPS (443/TCP), and WireGuard VPN (51820/UDP). All other ports closed. HTTP (80) blocked to enforce HTTPS-only web access. Logging enabled at medium level — captures blocked packets with full metadata.

7.3 IDS/IPS Configuration Summary

Suricata deployed as the IDS, monitoring the loopback interface to capture intra-VM lab traffic. Three custom detection rules layered on top of the Emerging Threats Open baseline ruleset: SSH brute-force (5 attempts / 60 seconds), suspicious DNS C2 indicator (`malicious.com`), and Nmap stealth scan (10 SYN packets / 10 seconds without handshake completion). Each rule targets a distinct stage of the attacker kill chain (credential access, command-and-control, reconnaissance respectively), validated against live attack simulations that triggered every rule within seconds.

7.4 Log Monitoring Approach (ELK)

Suricata alerts captured in `eve.json`, shipped to Elasticsearch by Filebeat using the pre-built Suricata module, visualized through Kibana's Filebeat Suricata Alert Overview dashboard. 1,178 alert documents successfully indexed during the test window. Time-series visualization makes anomaly spikes visually obvious; the alert detail table provides full forensic context (timestamp, source/destination, flow ID, signature category) for any investigation pivot. UFW firewall logs remain file-local at `/var/log/ufw.log`; integrating them into the same ELK pipeline via Filebeat's system module is a stated future enhancement.

7.5 Trade-offs Acknowledged

Decision	Security Benefit	Trade-off
Self-signed cert vs. Let's Encrypt	Same encryption strength	Browser warning; not suitable for public-facing production
Key-only SSH	Eliminates brute-force class of attacks	Key loss = lockout; requires key management discipline
WireGuard over OpenVPN	Simpler, faster, modern crypto	Fewer enterprise config options
Default-deny firewall	Minimizes attack surface	Every new service requires explicit rule; admin overhead
Separate web and SSH servers	Service isolation limits breach blast radius	More VMs to manage
Suricata over Snort	Native multi-threading + JSON output (ELK-ready)	Smaller rule-author community than legacy

Decision	Security Benefit	Trade-off
		Snort
ELK with X-Pack security disabled	Easier lab setup; dashboard immediately usable	**`xpack.security` MUST be re-enabled before production**
Polling-based Kibana viz	No extra alerting infrastructure required	No real-time push — production needs Elastalert / Watcher

Report compiled by Douglass Brown | Jan C Rosas Ortiz | Spring 2026