# Experience: ENTRAP – An IaC Enterprise Cyber Range for Reproducible Security Assessment and Dataset Generation

Rathan Appana, Aditya Kulkarni, and Bheemarjuna Reddy Tamma
*Center for Cryptography and Cybersecurity, Indian Institute of Technology Hyderabad, India*
{rathan.appana, aditya.kulkarni}@prjt.cse.iith.ac.in, tbr@iith.ac.in

***Abstract*—An enterprise breach causes most of its damage after the perimeter is crossed. Once an attacker gains a foothold, they can move laterally to other hosts, take over the trusted services of the network, and encrypt the backups needed for recovery. Defenders, on the other hand, need to study this post-compromise stage, but they cannot rehearse it on a production enterprise network. Building such an enterprise network manually is time-consuming, and resetting it to its clean state between runs is still harder. As a result, security experiments are difficult to perform, compare and reproduce.**

**We develop Enterprise Network Testbed for Reproducible Attack and Defense Practices (ENTRAP), an Infrastructure-as-Code (IaC) cyber range that defines a fully segmented enterprise as version-controlled automation. ENTRAP uses Ansible, KVM/Libvirt, and Cloud-init to provision the environment and reset it to a clean state whenever necessary. The enterprise network includes several entities such as a gateway, an internal Demilitarized Zone (DMZ), DNS, DHCP, a backup server, a web server, and clients. On this network, a user can learn to assemble the entities, launch attacks, and write rules or defensive measures that harden its security. We launch three attacks: a Ransomware-as-a-Service (RaaS)-backed ransomware campaign, a Layer-2 DHCP starvation and rogue DHCP attack, and perimeter web exploitation. Each attack runs from a clean state, and ENTRAP acts as a reproducible generator of labeled attack datasets. Using the collected dataset, we train a Random Forest classifier for ransomware detection and present its evaluation results. We publicly release the codebase, collected logs, and datasets to facilitate further research.**

***Index Terms*—Cyber Range, Infrastructure-as-Code, Intrusion Detection, Ransomware, Datasets.**

## I. Introduction

Enterprise networks form the backbone of modern organizations by connecting users, services, and information across different organizational segments, all governed by security policies. With a massive amount of sensitive data flowing through these networks, they become the primary target for cyberattacks. As organizations grow to include several more entities, they become increasingly dependent on these networks to process routine operations; thereby, the surface area for potential compromise grows proportionally. Despite years of investment and improvements in security tooling and awareness, attackers have kept pace by continuing the trajectory of the overall cost of data breaches. IBM's 2025 Cost of a Data Breach report [1] recorded a global average breach cost of USD 4.4 million, down by 9% from the previous year. This first decline in five years was driven largely by faster breach containment through AI-powered defenses. However, this improvement did not last longer; IBM's 2026 report [2] recorded a global average cost of USD 4.99 million, a 12% increase and an all-time high driven by a 56% rise in AI-powered attacks. Additionally, according to the Verizon 2025 report [3], ransomware remains a primary driver of cyberattacks, present in 44% of breaches.

### A. Cyber Ranges and Automated Enterprise Network Testbed

Studying the attack surface of an enterprise network requires a controlled experimental environment that reproduces its segments, services, and inter-entity trust relationships, such as a client trusting its DHCP for network configurations. In such an environment, researchers can launch multi-layer attacks (network and application), collect labeled datasets, experiment with defensive strategies, and train personnel, all without impacting the production network. A cyber range provides this by offering purpose-built network environments to safely launch cyberattacks, study their behavior across individual entities and the network as a whole, and easily reset between experiments. Such an environment allows analysis of the complete attack lifecycle in isolation. However, a cyber range requires an underlying testbed: an emulated enterprise network that reproduces the organization's topology, services, and security policies. Manually emulating such a network requires significant time and effort and, more importantly, it cannot be easily reset to a clean state between experiments. This motivates the need for an *automated* enterprise network testbed, one that can be described as code, instantiated on demand, torn down, and rebuilt identically any number of times. In this work, we present Enterprise Network Testbed for Reproducible Attack and Defense Practices (ENTRAP): an automated testbed that instantiates the cyber range for enterprise environments.

### B. Motivation and Research Gaps

Reproducible cyber ranges are not new, but existing ones leave certain gaps that we target in this work. Many published cyber ranges [4]–[6] are oriented toward the human-facing training part rather than toward a reproducible offense-and-defense experimentation part. The cyber ranges that automate environment creation tend to target either per-trainee content [5], [6], or specialized tactical networks [7] rather than the

segmented enterprise environments, which includes gateway, Demilitarized Zone (DMZ), infrastructure services, and client systems. Our work, ENTRAP, is designed to fill this gap and serves as a fully automated enterprise cyber range that allows cybersecurity researchers to launch multi-layer attacks against a segmented enterprise network, collect labeled datasets from each experiment, and study the effectiveness of various defense mechanisms in a controlled environment.

### C. Our Contributions

The key contributions of our work are as follows.

(i) We develop ENTRAP, a fully automated, Infrastructure-as-Code (IaC) enterprise testbed that instantiates a complete segmented enterprise network reproducibly from a clean state on demand.

(ii) On this testbed, we launch three attacks, (i) Ransomware-as-a-Service (RaaS)-backed ransomware campaign, (ii) a Layer-2 DHCP starvation combined with rogue DHCP server, and (iii) perimeter web exploitation, and we draft defensive mechanisms for each attack.

(iii) We train a ransomware detection model on the collected dataset and evaluate its robustness. We release the codebase of ENTRAP, collected logs, and datasets publicly for further research [8].

## II. Related Work

This section describes various cyber ranges, attacks performed on enterprise networks, and network defenses to mitigate such attacks.

### A. Reproducible Testbeds and Cyber Ranges

For years, network experimentation has faced a major hurdle: *reproducibility*. Table I compares ENTRAP with existing cyber ranges across these dimensions. Emulab [9] filled this gap by introducing an experimental network configuration that, once set, can be instantiated whenever required. It unified emulated, simulated, and wide-area resources under a shared `ns`-language script. When scaling these environments became the next challenge, subsequent work [10] multiplexed virtual hosts, routers, and links across fewer physical machines, which enabled them to run topologies an order of magnitude larger.

Cyber ranges take reproducible testbeds a step further by focusing specifically on security experiments. Yamin *et al.* [4] surveyed and evaluated these platforms across dimensions such as scenario generation, monitoring, teaming, scoring, and management, noting that many cyber range platforms heavily prioritize security training and human-in-the-loop exercises. To address the need of automation, CyRIS [5] launches cyber ranges on KVM from `YAML` descriptions. It populates these environments with vital security elements, including malware, attacks, captured traffic, and logs. Its generated ranges remain limited, usually confined to a single bus topology. CyTrONE [6] uses CyRIS as its backend engine to instantiate the environment and adds the training layer on top, including an instructor wizard that builds the exercise content and delivers it through an online learning platform, scaling smoothly to hundreds of trainees at the same time. With a different architectural approach, CyberVAN [7] provides a high-fidelity, GUI-based testbed that combines physical machines, virtual machines and network simulation to reproduce tactical and strategic military networks. However, its focus is high-fidelity military network scenarios operated as a closed-laboratory facility, rather than a segmented enterprise network expressed as reproducible, version-controlled infrastructure. DETERLab [11] is a shared, Emulab-based cybersecurity experimentation infrastructure that supports physical, virtual, and containerized resources for large-scale experiments.

TABLE I
Comparison of ENTRAP with existing cyber ranges

| Cyber range | Target environment | How the environment is defined? |
|---|---|---|
| Emulab [9] | Generic network topologies; not security-specific | `ns` script |
| CyRIS [5] | Single-host training ranges on a bus topology | YAML description; per-trainee instantiation |
| CyTrONE [6] | Per-trainee training exercise (using CyRIS) | YAML description |
| CyberVAN [7] | Tactical/Strategic military network scenarios | XML scenario file; GUI-based |
| DETERLab [11] | Shared infrastructure for large-scale cybersecurity experiments | Emulab-based testbed; experiment definitions and MAGI workflows |
| **ENTRAP** | Segmented enterprise network with DMZ and inline IDS/IPS | IaC (Ansible, Cloud-init); clean state rebuild per run |

### B. Enterprise Attack and Network Defenses

Normally, enterprise network security experiments focus on behavior after the initial compromise. MITRE ATT&CK [12] organizes this behavior into several stages including initial access, discovery, lateral movement, and impact. Kharraz *et al.* [13] show that the consequences of attacks such as ransomware strongly depend on how far an attacker can propagate within an internal network. These observations motivate test environments that not only represent the initial compromise but also the internal services, credentials, protocols, and trust relationships that enable movement between entities.

Evaluating defense against infrastructure-level threats requires replicating inherently vulnerable protocols in a controlled environment. The unauthenticated broadcast design of DHCP in RFC 2131 exposes enterprise segments to address pool starvation and rogue responses; although a cryptographic message authentication exists in RFC 3118, its deployment rate remains very low. Instead, DHCP snooping at the switch level blocks rogue server responses from untrusted ports and is the standard approach, but software bridge environments typically lack this feature. Validating enterprise resilience requires observing how inline IDS/IPS systems such as Snort and Suricata react to these infrastructure exploits.

Existing platforms target training exercises (CyRIS, CyTrONE), tactical networks (CyberVAN), or shared cybersecurity experimentation at scale (DETERLab). However, these platforms do not provide a locally deployable, IaC-based enterprise network testbed that combines segmented topology, vulnerable infrastructure services, and inline IDS/IPS within an environment reconstructible from a known clean state.

TABLE II
VIRTUAL MACHINES AND THEIR ROLES

| VM | IP address(es) | Role & key technology |
|---|---|---|
| gateway1 | 10.10.0.1 (LAN)<br>173.168.0.10 (WAN) | NAT edge router; Suricata IPS inline via nftables/NFQUEUE; DNATs :80 → web1 |
| coreswitch | 10.10.0.2 (uplink)<br>10.10.10.1 (infra)<br>10.10.20.1 (client) | Internal router / DMZ boundary; stateful iptables; isc-dhcp-relay |
| dns1 | 10.10.10.53 | BIND9; authoritative for enterprise.lab |
| dhcp1 | 10.10.10.2 | Kea DHCP4; pools .150 – .200 per segment |
| backup1 | 10.10.10.20 | Samba + rsync; nightly SSH-pull backup |
| web1 | 10.10.10.10 | Apache2/PHP; deliberately vulnerable |
| client1..N | 10.10.20.150 – .200 | Ubuntu workstations; DHCP-assigned |
| c2server | 173.168.0.50 | Flask RaaS C2; GonnaCry payload server |

## III. ENTRAP: PROPOSED TESTBED

ENTRAP is an enterprise cyber range expressed entirely as IaC, reproducing the segments of a real enterprise network. Figure 1 outlines its four layers (read bottom-up): (i) *infrastructure and automation layer*, which provisions all entities using KVM/QEMU, libvirt, and Ansible, (ii) *enterprise network layer*, which defines four isolated segments, (iii) *services and security layer*, which details the services and defensive controls built in each entity, and (iv) *attacks and data output layer*, which illustrates the attack launched on testbed and the logs recorded during each attack.

The goal of the design is reproducibility: every run starts from the same clean state, so an experiment (offensive) that damages or breaches the environment can be reset before the next run. A clean state refers to a freshly provisioned deployment in which all VMs, virtual networks, service configurations, and experiment-generated state from the previous run have been removed and recreated using the same ENTRAP automation. It also provides the feasibility of a working attack toolkit and an inline detection stack; as a result, both offensive and defensive mechanisms can be exercised repeatedly and reset between runs.

### A. Technology Stack and Software Used

We use KVM/QEMU with libvirt, which provides the virtualization layer on which every entity runs as a Virtual Machine (VM). Cloud-init provisions each VM at first boot by setting its hostname, network interfaces, users, and specified packages. Ansible automates the entire setup: it defines the entities, applies roles to them, and configures services in a fixed order. All entities share the Ubuntu 22.04 server cloud image as a common base. Network enforcement and services use nftables on the gateway, iptables on the internal router, Kea DHCP4 with an ISC DHCP relay on the router, BIND9 for the DNS server, Samba with rsync for the backup server, and Apache2 with PHP for the web server. Suricata [14] runs inline and uses a set of local and community rules to protect the enterprise network. None of these entities is novel; they are designed on established standards. The key contribution is that ENTRAP assembles these entities in a fixed order and re-instantiates them identically on demand: seven entities across four isolated segments.

### B. VMs and their Roles

Table II lists every entity in the testbed with IP addresses and key technologies used. The gateway (gateway1) is the only entity that performs Network Address Translation (NAT). The internal router (coreswitch) routes traffic between the infrastructure and client segments and enforces access control policy at the segment boundary; although web1 shares the infrastructure subnet with the other services, coreswitch ACLs restrict inter-segment traffic, creating a logical DMZ boundary without a physically separate network segment. Its name reflects the topological position as the core of the network, although it operates at Layer 3. The remaining internal entities serve as the DNS, DHCP, backup server, and web server, alongside client workstations. Finally, a dedicated attacker-controlled entity, c2server, sits on the simulated Wide Area Network (WAN), and it is referred to during the ransomware campaign described in Section IV-B.

### C. Virtual Networks

The enterprise network is divided into four isolated libvirt networks (see Figure 1): a simulated WAN (wan-net), a gateway-to-core transit link (link-net), an infrastructure segment (infra-net), and a client segment (client-net). Each network is completely separated into its own virtual switch (virbr1–virbr4) and does not allow cross bridge connectivity at Layer 2. The gateway is the only entity that masquerades outbound traffic to the simulated WAN, and the only inline path from WAN is a Destination NAT (DNAT) rule that exposes web1 on TCP:80. Ansible must run on the hypervisor because the internal networks have no outside access. Ansible instead uses an SSH ProxyJump via gateway1 to reach each of them.

### D. Network Topology

Figure 2 presents the network topology of our enterprise network testbed, which consists of several entities. The gateway separates the simulated WAN from the enterprise and runs the inline IDS/IPS. The internal router sits between the infrastructure and client segments and is the single point of DMZ access control policy. All deep packet inspection occurs at the gateway, while the internal router applies rules specific to address- and port-levels.

## IV. OFFENSIVE AND DEFENSIVE MEASURES

This section states the attacker model, launching three independent attacks on the enterprise network: a RaaS campaign, a Layer-2 DHCP starvation with a rogue DHCP server, and a perimeter web exploitation, and describes the defensive mechanisms to detect and mitigate each of these attacks. We also tabulate the testbed instantiation time from a clean state and summarize the network flows recorded at the gateway.

### A. Attacker Model

**Attacker Identity:** We consider a single attacker, *Trudy*, whose starting position and privileges vary per attack.

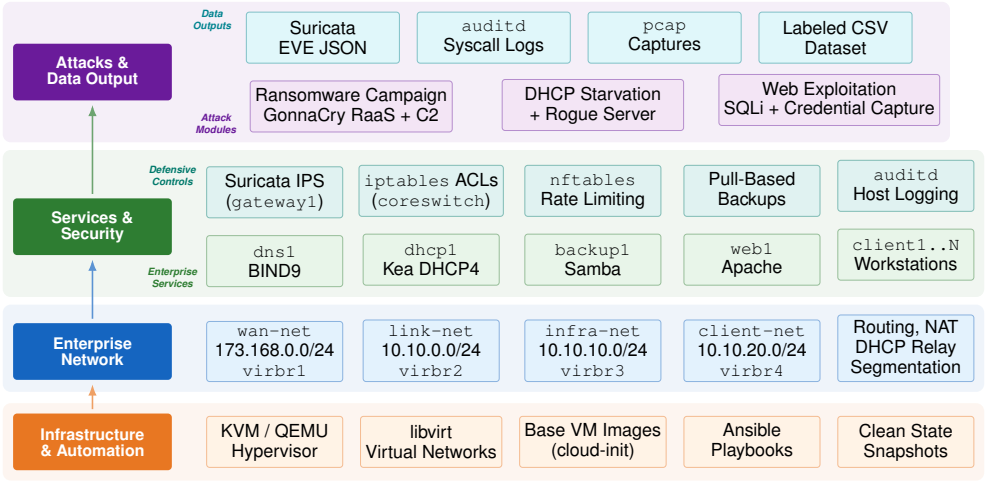



Fig. 1. ENTRAP architectural overview (read bottom-up). Infrastructure automation, enterprise network segments, services paired with defensive controls, and attack modules with their resulting outputs.

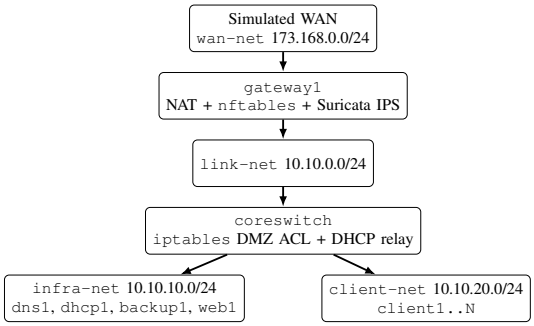



Fig. 2. Network topology of the ENTRAP cyber range.

- *Ransomware campaign:* Trudy is an *external* attacker who controls `c2server` (173.168.0.50) on the simulated WAN. Trudy has already performed an initial compromise phase (e.g., a phishing click or a leaked weak password) on a client. This is an assumed precondition, and not a demonstrated exploit.
- *DHCP starvation and rogue server:* Trudy is an *insider* on the client segment (`client-net`, 10.10.20.0/24), and operates from `client2`.
- *Perimeter web exploitation:* Trudy is an *external* attacker operating from the simulated WAN (`wan-net`, 173.168.0.0/24) without having an internal access.

**In scope:** Across all three attacks performed, Trudy can send and receive arbitrary traffic, craft Layer-2 frames, spoof MAC addresses, and run standard tools (`Scapy`, `dnsmasq`, `nmap`). Trudy can also reuse any keys recovered on one entity against every other entity. Trudy, with these three capabilities, can exercise the three attacks on the enterprise network: (i) delivery and execution of ransomware attack on a compromised client, lateral movement, and encryption, (ii) exhausting the IP address in the legitimate DHCP pool, and setting up a rogue server with configurations on gateway IP address and DNS resolver address, (iii) exploitation of a deliberately vulnerable web server.

**Out of scope:** Trudy does not control the hypervisor, physical bridges, or the gateway's firewall rules. She also cannot access the host machine, compromise the base VM image, or tamper with any of the Ansible playbooks that setup the very enterprise network.

**Role of the C2 server:** The `c2server` (173.168.0.50) is only used for the ransomware campaign. It hosts the GonnaCry RaaS, which is a web program (built with Flask) that does three things: it hands out the encryptor and decryptor binaries, tracks each victim with a unique machine ID, and manages the key-exchange protocol for decryption. This server has no role to play in the DHCP attack and the web exploitation attack.

Note that the communications between the compromised client and the C2 server uses plaintext HTTP protocol on port 80. ENTRAP does not include a PKI or certification authority, so a trusted TLS certificate cannot be issued within it. Also, this is a deliberate design choice we make so that the Suricata rules inline at the `gateway1` can perform deep packet inspection and check if the traffic matches with the rules. We acknowledge that even under HTTPS with TLS 1.3, the SNI field in the `ClientHello` message is in plaintext and Suricata can still identify if the packet is directed to the C2 server. However, we do not implement SNI-based detection in the current version of the ENTRAP testbed. Also, Encrypted Client Hello (ECH) conceals the SNI field as well, but we have left these two approaches as a part of our future work.

**Correlation of attacks:** The three attacks are run independently each starting from a clean state of the enterprise network. No attack depends on the outcome of the other.

### B. Offensive Measures

*1) Ransomware Campaign:* This attack demonstrates the ransomware campaign from payload delivery from an external `c2server`, client-side file encryption, lateral movement to the backup server, backup server's file encryption and key exfiltration back to the C2 server (as shown in Figure 3). The

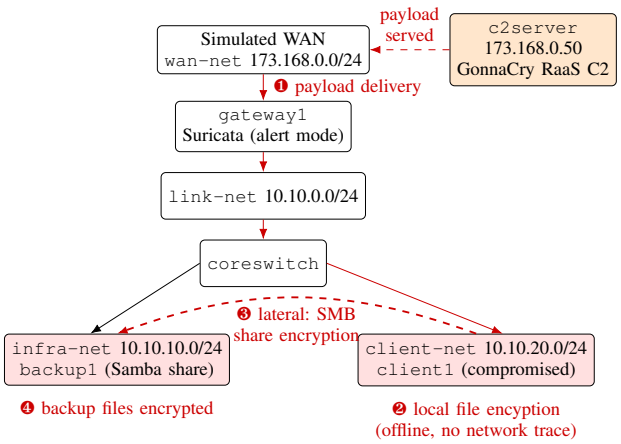



Fig. 3. Ransomware campaign.

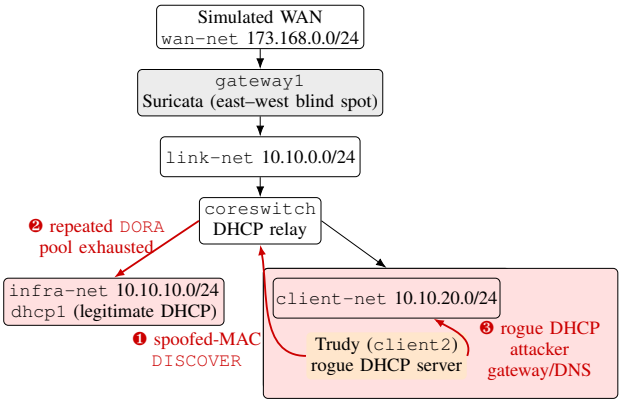



Fig. 4. DHCP starvation and rogue DHCP server.

north–south path (client↔WAN) and the east–west path (client segment↔infra segment), produce logs for the labeled dataset.

The `c2server` runs on port 80 and implements the RaaS backend. It serves two things: `/download-gonnacry/` delivers the encryptor binary, and `/decrypt/` accepts the victim's encrypted private key and returns the decrypted key after verifying payment. The server holds the RSA-2048 private key which is the root of the entire key hierarchy; without this, no victim can recover their files. The communications between the victim and the `c2server` are exchanged in plaintext on port 80. This is deliberately chosen for the testbed because the Suricata, deployed at the `gateway1` can easily perform deep packet inspection to match the rules on unencrypted packets, rather than dealing with additional complexity of analyzing encrypted payloads over HTTPS.

The ransomware attack is launched using GonnaCry [15], implemented as a Python 3 application, proceeds in the following stages.

(i) *Payload delivery:* The compromised client sends a HTTP `GET` request to the `c2server` to download the ransomware binary. This communication gets captured at the `gateway1`, where Suricata `alert` rules log the connection, but do not block packets.

(ii) *Local file encryption:* The downloaded binary generates a fresh RSA-2048 key pair unique to this victim and encrypts targeted files, such as, documents, databases, source codes, VM disk images. Also, the extensions like `.pem`, `.key`, and `.p12` files are encrypted using randomly generated AES-256-GCM key. Each AES key is encrypted with the victim's RSA public key. The victim's RSA private key is then encrypted with the `c2server`'s public key and is stored locally at `~/gonnacry/`. Encrypted files are then renamed with `.GNNCRY` extension, and a `RANSOM_NOTE.txt` containing machine ID and instructions for paying a ransom are written to the disk. This stage does not leave any trace at the gateway.

(iii) *Lateral movement and encryption:* GonnaCry also searches for backup services and mounts any SMB network shares that are reachable from the victim. Once identified, the ransomware encrypts backup files using the same AES-256-GCM/RSA-2048 scheme. This lateral encryption step destroys completely the backup archive as a recovery path, thereby forcing the victim to depend solely on the C2 server for decryption.

(iv) *Key exfiltration:* The victim sends the locally stored encrypted RSA private key along with the ransom to the `c2server`'s `/decrypt/` endpoint through HTTP `POST`. The server decrypts using its own RSA private key and return the victim's RSA private key to decrypt all AES keys and finally recover the files by the victim.

*2) Layer-2 DHCP Starvation and Rogue Server:* In this attack, Trudy is an insider operating from `client2` on the client segment who exhausts the legitimate DHCP IP address pool, thereby denying services to legitimate clients and coercing them to use a rogue DHCP server and its provided configurations. The attack, presented in Figure 4, runs in the following two coordinated phases from `client2`.

(i) *DHCP pool exhaustion:* Trudy targets the legitimate DHCP server (`dhcp1`, 10.10.10.2) by sending a sequence of unique, fabricated DHCP `DISCOVER` messages via the `coreswitch`. As discussed in RFC 2131, the DHCP protocol does not have a built-in mechanism to authenticate clients and verify the legitimacy of the `DISCOVER` message. `dhcp1` responds to all these spoofed requests and commits a lease through the full `DORA` (`DISCOVER`–`OFFER`–`REQUEST`–`ACKNOWLEDGE`) handshake, until its address pool exhausts.

(ii) *Rogue DHCP server:* Trudy then instantiates a rogue server on `client2` with a small and different address range than the legitimate DHCP server. Now that the legitimate address pool is drained, new clients broadcasted `DISCOVER` messages are responded to by the rogue server. These new clients complete the `DORA` handshake with the rogue server and receive the configuration parameters like the default gateway, and the DNS resolver address. If Trudy wishes, then the default gateway can be set to `client2`'s own IP address, which allows Trudy to observe all the outbound traffic, which enables a transparent `MITM` attack. Otherwise, the rogue server responds with the real gateway address, so the

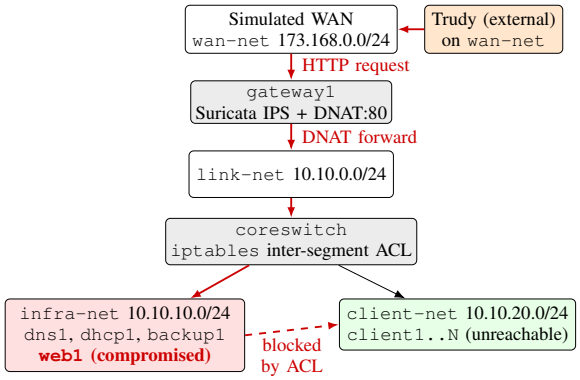



Fig. 5. Trudy reaches web1 from the simulated WAN through the gateway's DNAT rule (red solid path). The dashed line shows that the lateral movement from the infrastructure segment to the client segment is blocked by the coreswitch ACLs. Blue nodes indicate defensive controls, and the green node indicates the unreachable segment.

connectivity to the clients appears normal.

*3) Perimeter Web exploitation:* Figure 5 presents a web exploitation vector where Trudy exploits an Internet-facing web application vulnerability to gain access to the network, as an initial access step. This technique maps to the MITRE ATT&CK ID T1190 [16]. The scenario considered is that an enterprise network hosts an internal web application, meant to serve internal subnets on plaintext HTTP; however, it can accidentally become externally reachable through a misconfigured firewall rule. Once exposed, the combination of the plaintext protocol and the application that was never hardened for public access creates an entry point for Trudy. The DNAT rule on gateway1 simulates an external exposure scenario, where HTTP traffic over TCP port 80 is forwarded to web1, rendering it to be reachable from the simulated WAN. Trudy, operating from the WAN, identifies this exposed service via network reconnaissance and initiates interaction. Another condition assumed in this scenario is that web1 is deliberately misconfigured with two distinct vulnerabilities: a reflected SQL injection vulnerability and insecure login form that sends credentials in plaintext via HTTP POST requests. Using this, Trudy exploits the SQL injection vulnerability to exfiltrate sensitive data from the web1 backend database, and is also in a position to intercept sensitive credentials transmitted insecurely through the form in plaintext.

### C. Defensive Measures

This section provides defenses against each attack in Section IV-B.

*1) Shared Control: Suricata Inline IPS at the Gateway:* All routed traffic through gateway1 is forwarded to Suricata via NFQUEUE in inline prevention mode (dropping packets). The queue is configured as fail-open: if Suricata terminates or becomes saturated, traffic continues uninspected rather than causing a network outage. This design represents an explicit availability-over-security trade-off. The HOME_NET variable encompasses the infrastructure, client subnets, and the gateway's WAN interface. The rule set combines community and local rules that detect activity, including ransomware communication between a compromised client and an attacker-controlled C2 server.

*2) Mitigating the Ransomware Campaign:* The compromised client contacts the c2server twice: once to download the payload and another to POST the encrypted private key for decryption. In both connections, the traffic flows through gateway1, where Suricata inspects them against rules matching .onion DNS resolution and domain generation algorithm (DGA) beaconing. Shifting the nature of these rules from alert to drop causes the payload download to be blocked before it even reaches the compromised client, which entirely prevents the infection. The following is the set of rules that mitigate the campaign.

```
# Alert mode: logs but does not block
alert dns ... msg:"Tor .onion query"; sid:1000070;
# Drop mode: blocks payload
drop dns ...msg:"Tor .onion query"; sid:1000070;
```

We note that this detection relies on plaintext payload inspection, but the choice of using HTTP over HTTPS for C2 communication is discussed under the attacker model (refer Section IV-A). In the current configuration, Suricata operates in alert mode, which allows to log full ransomware lifecycle and collect the dataset. Even in the worst-case scenario where the payload gets downloaded on the client, the *pull-based* backups keep the archive completely isolated. Because the backup server pulls the data rather than the client pushing it, the client has no way to reach or encrypt the archives during the attack. As an additional guideline, organizations can protect backup servers by storing backup encryption keys inside a Hardware Security Module (HSM) [17].

*3) Mitigating the DHCP Layer-2 Attack:* The DHCP starvation attack entirely operates within the client segment's broadcast domain, which Suricata at the gateway cannot record. To mitigate IP pool exhaustion, we require a control to be placed at the bridge level, therefore we apply an aggregate rate limit at the client bridge, which restricts the number of DHCP DISCOVER messages forwarded per second.

```
# Rate-limit DHCP DISCOVER on bridge
nft add rule bridge filter forward \
   udp dport 67 limit rate over 10/second \
   drop comment "dhcp-flood-guard"
```

This slows the exhaustion attack to the point where it becomes detectable before the IP pool is fully drained.

*4) Mitigating the Perimeter Web Exploitation:* The external entry point in ENTRAP is web1, which is reachable from the simulated WAN through the gateway's DNAT rule. To mitigate SQL injection and directory traversal attacks, Suricata rule set, deployed at gateway, matches these requests and drops them before they even reach the web server. In case, Trudy does compromise web1, network segmentation isolates it inside the DMZ. However, security controls at coreswitch1 enforces strict inter-segment ACLs, which restrict web1 to initiate connection to the client segment or any other services (backup, DNS, DHCP) present within the infrastructure segment itself. To prevent lateral movement via credential reuse, all service-to-service communications mandate SSH key-based authentication.

TABLE III
ENTRAP BUILD TIME FROM A CLEAN STATE

| Phase | Duration (s) |
|---|---|
| Network Creation (4 `libvirt` bridges) | 6 |
| VM provisioning (7 entities + 2 clients, Cloud-init) | 227 |
| Service Configuration (Ansible roles) | 719 |
| Logging configuration | 103 |
| **Total** | **1,055** |

### D. Setup and Experiment Cost

Table III reports the time taken to build the entire enterprise system from a clean state on a single hypervisor host (Intel Xeon W-2255, CPU 10cores, 20 threads, 3.70GHz 32GB RAM, running Arch Linux). The setup proceeds in four stages: (1) *network creation*, which defines and starts the four isolated `libvirt` bridges; (2) *VM provisioning*, which creates each entity's disk image and Cloud-init ISOs, and boots all nine VMs; (3) *service configuration*, which uses Ansible over SSH to install and configure every service in order of dependency; and (4) *logging configuration*, which turns on protocol-split forensic logging across the environment. The biggest bottleneck is VM provisioning, mostly because of Cloud-init's first-boot initialization and package installation. A complete build from a clean state takes under 18 minutes. Because we use the same playbooks to reset the enterprise system, this time is also the absolute maximum time one has to wait to get to a clean state. Even if an experiment completely tears down the enterprise setup, it will be fully restored to the clean state within a single provisioning cycle.

Table IV shows the traffic recorded at `gateway1` between the compromised client (10.10.20.151) and the C2 server (173.168.0.50). In Transfer 1 and Transfer 2, the client requests the ransomware payload from the `c2server` and receives it to encrypt the files. The encryption runs locally on compromised client and hence leaves no trace at the gateway. Once the files are encrypted, the client submits the encrypted private key to the `c2server` and receives the decryption key to recover files.

Table V shows a combined DHCP starvation and rogue server attack. The rogue server starts first, serving a range within the same subnet but distinct from the legitimate server's IP address pool. The legitimate DHCP server is then flooded with 49 consecutive forged requests, exhausting the address pool. The victim's renewal is refused by a rogue `DHCPNAK`, and its released address is seized, so its next `DORA` exchange can complete only with the rogue server. This gives Trudy control over the newly configured gateway and DNS resolver addresses of the affected client.

TABLE IV
PAYLOAD DELIVERY FLOWS OBSERVED AT `GATEWAY1`

| Phase | Direction | Size | Port |
|---|---|---|---|
| `client`: 10.10.20.151 `C2`: 173.168.0.50:80 | | | |
| Download 1 – request | `client` → `C2` | ∼580 KB | :44406 |
| Download 1 – response | `C2` → `client` | ∼31.5 MB | |
| Download 2 – request | `client` → `C2` | ∼586 KB | :56552 |
| Download 2 – response | `C2` → `client` | ∼31.9 MB | |
| *File encryption at compromised client – no network trace* | | | |
| Encrypted RSA key | `client` → `C2` | ∼5 KB | :56554 |
| Server`ACK` | `C2` → `client` | ∼2 KB | |

TABLE V
DHCP STARVATION AND ROGUE SERVER ATTACK

| Step | Observation and Effect |
|---|---|
| *Rogue DHCP server deployed* | |
| Start daemon | `dnsmasq` starts on `client2` with IP pool .210–.220, and a 12-hour lease time. |
| *Legitimate DHCP pool drained* | |
| Lease flood | 49 forged MACs request IPs .152–.200 at ∼10 leases/s. Each lease lasts 7200 s, completely draining the real DHCP server's IP pool. |
| Forgery Proof | `hostname` and `client_id` fields are empty on all 50 attack leases, whereas legitimate clients always include them. |
| *Legitimate lease invalidation* | |
| Renewal denied | Victim sends a `REQUEST` for .150, the rogue server replies with `NAK`, causing the victim to drop its current lease. |
| Address seized | Released .150 is instantly grabbed by a fake MAC address, which prevents reclamation. |
| *Victim bound to rogue server* | |
| `DORA` on rogue | Full `DORA` runs through the rogue server. `client1` binds to .216, giving the attacker control over its gateway and DNS resolver. |

## V. USE CASES

ENTRAP's clean state reset enables many use cases beyond the attacks and defenses mentioned above.

1) *Dataset Collection:* We collect logs specific to each attack using a controller script that timestamps each run, executes the attack, and invokes an Ansible playbook to pull logs from every entity in parallel into a timestamped directory per run. These logs consist of network IDS events from the gateway, host telemetry (such as `auditd`, `syslog`, and `auth.log`) from the clients, and service logs (Kea DHCP, BIND9, Samba and Apache) from the infrastructure segment. Each run creates an attack-specific label directory to distinguish between different attacks.
   For the DHCP attack, the dataset contains Kea's lease table and the per-request DHCP `DORA` log, presenting the pool starvation under spoofed-MAC floods, alongside gateway traffic flow and anomaly events. For the ransomware campaign, the dataset captures compromised client-side `syslog`, gateway Suricata logs of C2 communication, and flow records, which provide the initial-access and encryption phase footprints. As a result of collecting known attack labels and precise timestamps, the dataset can be used to train supervised classifiers that distinguish between attack or benign traffic. And further samples from multiple clean state runs can serve as a validation dataset to evaluate the robustness of the model. We release our codebase, collected logs (Suricata JSON files, `auditd` traces, Kea DHCP logs, and `pcap` files) and labeled datasets publicly on our repository [8].
2) *Replicating & Hardening Custom Enterprise Environment:* Because the testbed is defined in Ansible playbooks, cybersecurity researchers can modify configurations to reflect their own enterprise system. Updating subnets, adding services, or changing policies requires

only editing the relevant playbook. Once deployed, users can launch attacks, inspect each entity, and evaluate defensive measures. If an entity breaks during an attack, then it can be destroyed and rebuilt to a clean state. This cycle of configuring, attacking, evaluating, and resetting provides a safe way to evaluate defenses before applying them in production enterprise environments.

3) *Cryptographic Analysis of Ransomware Encryption Schemes:* The ransomware campaign uses AES-256-GCM, implemented as a module that can be substituted with a different algorithm. Because the testbed can be easily reset to a clean state, a cryptanalyst can substitute a different algorithm on each run to collect the resulting plaintext and ciphertext pairs. Across multiple runs, researchers can study their susceptibility to chosen-plaintext attacks.
4) *Learning Enterprise Network Assembly:* ENTRAP provides hands-on experience in building a realistic enterprise network in the correct order. The Ansible playbooks map out entity roles and their dependencies explicitly, so students and practitioners can read the code and understand exactly why one entity must be configured before the next. This ordered assembly replicates what an administrator faces in a real enterprise, but in a safe environment where mistakes carry no consequences.

## VI. Ransomware Detection Experiment

We explore the *dataset collection* use case described in Section V. We train a Random Forest (RF) classifier on our testbed logs and evaluate its robustness on another run on the same testbed. We executed three GonnaCry ransomware runs and one baseline run on the testbed, each starting from a clean state snapshot, producing 9,068 labeled events (5,282 benign and 3,786 ransomware). Events falling within the recorded attack window are labeled `gonnacry`; and those outside (normal system activity on hosts, and the idle attacker VM) as `benign`. These samples combine Suricata flow records from `gateway1` with `auditd` syscall traces from the client VMs. Rather than using raw numeric fields, we have 24 behavioral features from the logs: seven network-level features from Suricata (`bytes_toserver`, `bytes_toclient`, `pkts_toserver`, `pkts_toclient`, `flow_duration`, `dest_port`, `src_port`), binary flags from the ransomware executable (`exe_is_gonnacry`, `exe_from_tmp`), one-hot encoded syscall types (`rename`, `execve`, `connect`, `openat`, `unlink`), audit key categories (`file_rename`, `net_connect`, `tmp_write`, `exec`), encryption path markers (`.GNNCRY`, `gonnacry/`), and destination flags for the C2 and backup servers. The training samples were split 80:20 with stratified sampling (7,254:1,814). To evaluate robustness, we re-ran a single ransomware attack on a clean state enterprise network setup and collected 3,573 new samples (1,685 benign, and 1,888 ransomware) that the model had never encountered. Table VI reports the results.

TABLE VI
Ransomware detection

| **Evaluation** | *Accuracy* | *Precision* | *Recall* | *F1-score* |
|---|---|---|---|---|
| Trained (80:20) | 0.92 | 1.00 | 0.80 | 0.89 |
| Inferred | 0.89 | 1.00 | 0.79 | 0.88 |

On the trained samples, the classifier achieves 92% accuracy with a precision of 100% at a recall of 80%. On the evaluation dataset, accuracy drops marginally to 89% at the same precision of 100% (zero false positives), at a recall of 79%. The minimal drop in recall confirms that the behavioral features generalize across clean-state rebuilds.

## VII. Conclusions and Future work

We developed ENTRAP, an IaC enterprise cyber range that defines a complete segmented enterprise network as version-controlled automation code and rebuilds the environment identically from a clean state on demand. We launched a ransomware campaign, DHCP starvation and a rogue DHCP server, and perimeter web exploitation attacks and deployed corresponding defensive measures. As each attack and log stream is defined as code and each experiment starts from the clean state, ENTRAP can be used to generate reproducible, labeled datasets for these attacks. Using the collected dataset, we trained a Random Forest classifier for ransomware detection and demonstrated that the behavioral features generalize across independent clean state runs, achieving 89% accuracy and zero false positives. We release the ENTRAP codebase, logs, and labeled datasets in this GitHub repository [8].

**Lessons Learned and Research Implications:** We observed several insights working with ENTRAP. First, the pull-based backup approach survived the ransomware campaign because it pulls files from the clients instead of letting the clients push files to the backup server. This simple choice eliminates lateral movement and the possibility of encrypting backup files from a compromised client. Second, we noticed that the DHCP starvation attack runs entirely within the client segment's broadcast domain, and Suricata IPS at the gateway cannot observe it. This shows that the enterprise network needs east–west monitoring at the bridge level, not just north-south inspection at the gateway. Third, even if a web server is compromised, the coreswitch ACLs prevents lateral movement to the client segment, showing that dividing the network into separate segments limits the blast radius of a perimeter breach.

As a part of future work, (i) we plan to integrate Ansible AWX as a web-based interface, enabling authenticated users to provision, reset, and interact with the testbed through a browser with a role-based access control for multi-user learning, (ii) we plan to include a VPN gateway to model a perimeter-less enterprise network, and extend Suricata's detection to identify HTTPS-based C2 communication using SNI-based rules and TTLS fingerprinting, and (iii) we wish to build robust detection models using the datasets collected from our testbed and evaluate them against zero-day attacks.

## References


[1] IBM Security, "Cost of a Data Breach Report 2025," IBM, Tech. Rep., 2025. [Online]. Available: https://www.ibm.com/think/x-force/2025-cost-of-a-data-breach-navigating-ai

[2] IBM Security, "Cost of a Data Breach Report 2026," IBM, Tech. Rep., 2026. [Online]. Available: https://www.ibm.com/reports/data-breach

[3] Verizon, "2025 data breach investigations report," Verizon Business, Tech. Rep., 2025. [Online]. Available: https://www.verizon.com/business/resources/reports/2025-dbir-data-breach-investigations-report.pdf

[4] M. M. Yamin, B. Katt, and V. Gkioulos, "Cyber ranges and security testbeds: Scenarios, functions, tools and architecture," *Computers & Security*, vol. 88, p. 101636, 2020.

[5] C. Pham, D. Tang, K.-i. Chinen, and R. Beuran, "CyRIS: A Cyber Range Instantiation System for Facilitating Security Training," in *Proceedings of the 7th Symposium on Information and Communication Technology*, 2016.

[6] B. Razvan, P. Cuong, T. Dat, C. Kenichi, T. Yasuo, and S. Yoichi, "CyTrONE: An Integrated Cybersecurity Training Framework," 2017.

[7] R. Chadha, T. Bowen, C.-Y. J. Chiang, Y. M. Gottlieb, A. Poylisher, A. Sapello, C. Serban, S. Sugrim, G. Walther, L. M. Marvel *et al.*, "CyberVAN: A Cyber Security Virtual Assured Network Testbed," in *Milcom Military Communications Conference*. IEEE, 2016.

[8] R. Appana, A. Kulkarni, and B. R. Tamma, "ENTRAP: Infrastructure-as-Code (IaC) for Enterprise Network," https://github.com/LetsBeSecure/ENTRAP-IaC-Testbed, 2026, accessed: 2026-11-12.

[9] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 255–270, 2002.

[10] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale Virtualization in the Emulab Network Testbed," in *USENIX Annual Technical Conference (USENIX ATC)*, 2008.

[11] D. Project, "The DETER Project," https://deter-project.org/about_deterlab.html, 2026, accessed: 2026-11-14.

[12] The MITRE Corporation, "MITRE ATT&CK: Enterprise techniques," https://attack.mitre.org/, 2024, accessed: 2026-08-17.

[13] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2015.

[14] Open Information Security Foundation, "Suricata: Open source ids / ips / nsm engine," https://suricata.io/, 2024, accessed: 2026-08-21.

[15] Tarcisio, Marinho and Vipul, Vaibhaw and Martin, Ligabue, "GonnaCry," https://github.com/tarcisio-marinho/GonnaCry, 2017, accessed: 2026-08-18.

[16] L. Don and W. Stifel, Yossi, "Exploit Public-Facing Application," https://attack.mitre.org/techniques/T1190/, 2026, accessed: 2026-09-12.

[17] R. Ron and P. Victoria, "Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations," https://nvlpubs.nist.gov/nistpubs/SpecialPublications/800-171r3/NIST.SP.800-171r3.html, 2024, accessed: 2026-08-22.