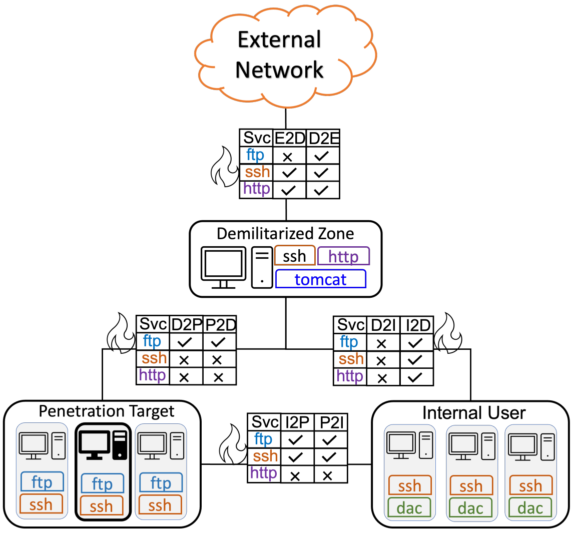
APPENDIX A: Penetration Testing Environment

We demonstrate a simplified example of constrained policy optimization on a multi-objective RL framework to introduce our penetration testing environment, which aims to maximize benefits while being cost-aware.

Environment Components

In this paper, the RL agent employs DQN techniques, enhanced with experience replay and ε-greedy exploration. Constrained agents extend this approach into a multi-objective setting by jointly optimizing task performance and operational cost, demonstrating the feasibility of RL in domains where side effects (e.g., detection risk or time cost) must be explicitly managed.



Svc	E2D	D2E
ftp	×	✓
ssh	✓	✓
http	✓	✓

Svc	D2P	P2D
ftp	✓	✓
ssh	×	×
http	×	×

Svc	D2I	I2D
ftp	×	✓
ssh	×	✓
http	×	✓

Svc	I2P	P2I
ftp	✓	✓
ssh	✓	✓
http	×	×

Figure 1: Network layout of the simulated penetration testing environment. The Demilitarized Zone (DMZ) is externally accessible and hosts multiple vulnerable services. The host highlighted in **bold** in the Penetration Targets zone is the final target and will be highly rewarded once compromised.

As illustrated in Figure 1, we set up firewall-allowed permissions and blocked communication rules to simulate the penetration testing. In our penetration testing environment, five public-facing services are available, including: `http` (web server, port 80), `ssh` (secure shell, port 22), `ftp` (file transfer protocol, port 21), `tomcat` (Apache servlet container), and `dac` (discretionary access control list service, a Windows-specific privilege escalation vector). While present on user hosts, the `dac` service is not a network-facing protocol. It is exploited locally for privilege escalation simulation.

Meanwhile, a segmented network architecture comprising four major zones is adopted: the External Internet (untrusted), the Demilitarized Zone (DMZ), the Internal Network (trusted), and the Penetration Target zone. The DMZ acts as a buffer and hosts public-facing services (e.g., web server, SSH, Tomcat) while isolating internal resources from direct exposure to external exposure. In our simulation, the DMZ contains a Linux-based host with public access. For the final target of our simulation, we set a target host with sensitive information in the Penetration Target zone.

Moreover, we set up firewall-allowed permissions and service-blocked rules to limit unnecessary exposure across zones and simulate penetration testing. As shown in Figure 1, **Svc** denotes a service permitted by firewall rules. Permissions in paths are introduced in Table 1.

These configurations simulate a layered security architecture requiring multi-step privilege escalation and lateral movement to reach sensitive targets.

Penetration Testing in RL: State, Action, Reward, and Cost

In the simulated environment, each interaction is recorded as a sequence of (*state, action, reward, cost*), with each episode forming an *penetration path*. Over time, the agent learns which sequences of actions produce the maximum cumulative rewards while minimizing costs.

Action: The agent operates in discrete action spaces encoded as integers. In our simulated environment, there are five types of actions, including:

- **Scan:** Probes a host to enumerate available services or discover reachability.
- **Exploit:** Attempts to compromise a specific service with a known vulnerability.
- **Privilege Escalation:** Gains higher-level access on an already compromised host.
- **Pivot:** Moves laterally from one host to another, leveraging existing access.
- **No-op:** A passive action is taken when no meaningful or safe operations are available.

State: The state represents the basic configuration of the network and determines how actions affect the environment and what agents will subsequently observe.

The simulation environment's internal state includes host connections, services, privileges, and compromise status. It will be updated after the agent takes action. Based on the new state and the outcome of the actions (rewards and costs), an observation will

Table 1: Network zone communication rules with firewall permission.

Code	Direction	Description
E2D	External → DMZ	Only `http` and `ssh` services are permitted into the DMZ, exposing limited public-facing functionality.
D2E	DMZ → External	All services are allowed in the internal network.
D2I	DMZ → Internal User	Direct communication is blocked to prevent lateral movement without escalation.
I2D	Internal User → DMZ	Internal users can access the DMZ via `ssh` for secure administrative tasks.
I2P	Internal User → Penetration Target	Users connect using `ssh` and `ftp`; `ftp` is unrestricted to support simulated data movement.
P2I	Penetration Target → Internal User	Permissions mirror those of **I2P**.
D2P	DMZ → Penetration Target	`ssh` and `http` are blocked; `ftp` is allowed for limited file transfer and simulated exfiltration.
P2D	Penetration Target → DMZ	Permissions mirror those of **D2P**.

be generated and returned to the agent as input to the agent's decision-making policy. It will contain indicators of discovered services, compromised hosts, current privilege levels, and auxiliary flags (e.g., success, permission denied, or connection errors).

In our penetration testing environment, the agent does not have direct access to the environment's full internal state. Instead, it receives the observation after each action. The agent learns observations to infer hidden aspects of the environment, prioritizes strategic actions, and adapts its policy to maximize reward while minimizing operational costs. This separation of state and observation allows us to simulate both fully observable and partially observable settings by limiting or filtering information passed from the true state to the agent.

Reward and Cost: Agent actions are selected by maximizing a scalarized Q-value that incorporates expected reward and cost. The environment computes both rewards and costs dynamically based on the agent's current state and chosen action. A non-zero reward is assigned for meaningful transitions (e.g., successful exploitation of a vulnerability, privilege escalation, or compromise of the final target host).

Each action incurs a fixed cost of 1, which accumulates over the episode. The cost-aware agent learns the cost value with TD updates. This cost signal is incorporated into the constraint policy to balance reward maximization and adherence to a specified cost constraint.

This sparse reward setup mimics real-world penetration testing, where only strategically correct sequences of actions lead to measurable success. The agent accumulates reward signals over time and learns the optimal penetration path through TD updates in DQN training.

APPENDIX B: Random- and Rule-Based Method

We show the implementation details of the random-based method and the rule-based method for penetration testing here. Based on the same environments as the standard DQN and our method, these two methods are executed as baselines against our method to compare penetration testing performance in reward and cost awareness.

Random-Based Method

We implement the random method as a naive baseline that selects actions uniformly and randomly from the action space, without using observations or learning. As shown in Algorithm 1, it interacts with the environment for a fixed number of training steps per episode, logging cumulative reward, cost, and goal success. This setup provides a memoryless baseline for benchmarking our Constrained-DQN.

Algorithm 1: Random-Based Method for Penetration Testing

```
Input  : Environment $\mathcal{E}$, total training steps $T$
Output: Episode trajectories with action logs and
        goal status
Initialize step counter $t \leftarrow 0$, episode index $k \leftarrow 1$;
while $t < T$ do
    Reset environment, get initial observation $o_0$;
    Initialize episode return $R_k \leftarrow 0$, cost $C_k \leftarrow 0$;
    while episode not terminated and $t < T$ do
        Sample action $a_t \sim \mathcal{U}(\mathcal{A})$ ;   // Random
          selection over action space
        Execute $a_t$ in $\mathcal{E}$, observe reward $r_t$, next
          obs $o_{t+1}$;
        $R_k \leftarrow R_k + r_t$, $C_k \leftarrow C_k + 1$, $t \leftarrow t + 1$;
        $o_t \leftarrow o_{t+1}$;
    Record action sequence, $R_k$, $C_k$, and goal
      completion flag;
    $k \leftarrow k + 1$;
```

Rule-Based Method

We implement the rule-based penetration testing method that utilizes the decision strategy logic inspired by practical penetration testing procedures, shown in Algorithm 2. It reflects staged techniques commonly used in tools such as the Metasploit Framework in conjunction with popular penetration testing tools (e.g., Nmap, Nessus, OpenVAS, Armitage, Wireshark, and Netcat), which start with network and service scan and then proceed to exploitation and privilege escalation.

The rule-based method tracks the host state by parsing observations and prioritizes scans (e.g., process, OS, service, and subnet scans). After that, attempting exploits or privilege escalation will be executed. We define failed exploits as failed penetration after a retry limit, which is set to a maximum of 2 to prevent repeated attempts on ineffective targets. Meanwhile, if one action cannot satisfy the strategy logic (i.e., no exploits and no need for privilege escalation), we return it to a *No-op*. This heuristic strategy provides a deterministic and interpretable baseline for evaluating our approach.

Algorithm 2: Rule-Based Method for Penetration Testing

```
Input  : Environment $\mathcal{E}$, total training steps $T$
Output: Episode logs with action traces and goal
        status
Initialize step counter $t \leftarrow 0$, episode index $k \leftarrow 1$;
while $t < T$ do
    Reset environment and internal agent state;
    Receive initial observation $o_0$;
    Initialize episode return $R_k \leftarrow 0$, cost $C_k \leftarrow 0$;
    while episode not terminated and $t < T$ do
        Parse state vector to extract host-level
          information (e.g., reachability, access,
          services);
        // Hierarchical rule-based action
          selection
        if process or OS scan is available then
            Select action $a_t \leftarrow$ process/OS scan
        else if service scan is available then
            Select action $a_t \leftarrow$ service scan
        else if subnet scan is available then
            Select action $a_t \leftarrow$ subnet scan
        else if exploitable service is known and
          retry budget not exceeded then
            Select action $a_t \leftarrow$ exploit
        else if privilege escalation opportunity is
          found then
            Select action $a_t \leftarrow$ privilege
              escalation
        else if secondary process/OS or service or
          subnet scan is still available then
            Retry scans in the same priority order
        else
            Select fallback scan or No-op action
              $a_t$
        Execute $a_t$ in $\mathcal{E}$, observe $r_t$, next
          observation $o_{t+1}$, and info;
        Update internal state (e.g., access levels,
          exploit attempts, scan coverage);
        $R_k \leftarrow R_k + r_t$, $C_k \leftarrow C_k + 1$, $t \leftarrow t + 1$;
        $o_t \leftarrow o_{t+1}$;
    Log action sequence, text trace, and goal
      outcome for episode $k$;
    $k \leftarrow k + 1$;
```