

REFN: A Reinforcement-Learning-From-Network Framework against 1-day/n-day Exploitations

Tianlong Yu¹, Lihong Liu¹, Ziyi Zhou¹, Fudu Xing³, Kailong Wang², Yang Yang¹

¹School of Artificial Intelligence, Hubei University, Wuhan, China

²Huazhong University of Science and Technology, Wuhan, China

³University of Southern California, Los Angeles, USA

tommyyu21@163.com, llh3401939433@outlook.com, 202421121013087@stu.hubu.edu.cn,

fuduxing@usc.edu, wangkl@hust.edu.cn, yangyang@hubu.edu.cn

Abstract—The exploitation of 1-day/n-day vulnerabilities poses severe threats to networked devices due to massive deployment scales and delayed patching (average Mean-Time-To-Patch exceeds 60 days). Existing defenses, including host-based patching and network-based filtering, are inadequate due to limited scalability across diverse devices, compatibility issues especially with embedded/legacy systems, and error-prone deployment process (e.g., manual patch validation). To address these issues, we introduce REFN (Reinforcement-Learning-From-Network), a novel framework that trains Large Language Models (LLMs) to autonomously generate network filters to prevent 1-day/n-day exploitations. REFN ensures scalability by uniquely employs Reinforcement Learning (RL) driven by online network rewards instead of traditional Human Feedback (RLHF). REFN guarantees compatibility via unified deployment on edge security gateways (e.g., Amazon Eero). REFN provides robustness via online validation using real network traffic. Crucially, REFN addresses three core challenges in training LLMs for exploit prevention: 1) expanding current LLMs’ limited vulnerability-fixing expertise via Agentic-RAG-based Knowledge Distillation; 2) bridging current LLMs’ language-to-network gaps through an RL-From-VNF Pipeline that translates language context (e.g., vulnerability description) into network enforcement; 3) addressing the LLM hallucination & non-determinism via the Online Agentic Validation that penalizes erroneous outputs. Evaluated across 22 families of 1-day/n-day exploits, REFN demonstrates effectiveness (21.1% higher accuracy than alternatives), efficiency (Mean-Time-To-Patch of 3.65 hours) and scalability (easily scale to 10K devices). REFN serves as an initial step toward training LLMs to rapidly prevent massive-scale 1-day/n-day exploitations.

I. INTRODUCTION

The real-world conflicts are rapidly expanding into cyberspace, driving massive-scale exploitations of 1-day/n-day vulnerabilities across networked devices [38], [8]. Landmark incidents, such as the 2021 Log4j vulnerability which affecting hundreds of millions of devices [8], demonstrate the severity of this threat. Offensive capabilities are further amplified by LLM-powered tools (e.g., WormGPT [29], HackerGPT [31]), while defenses are slow to respond and falling behind, e.g., critical patches face dangerous delays, with Mean-Time-To-Patch (MTTP) averaging 60 to 150 days [19].

Current vulnerability fixing strategies primarily utilize two approaches: host-based patching [17], [12], [10], [20], [15], [27], [14], [21], [25], [64], [63], [45], [66], [50] and network-based filtering [57], [26], [48], [62], [11], [18], [61]. Host-

based patching operates by updating software or hardware on vulnerable devices through patch generation, installation, and validation. Despite automation efforts including patch management tools [17], [12], [10], [20], [15], [27], [14], [21], [25] (largely restricted to standard computers) and more recent generic ML-based approaches [64], [63], [45], [66], [50], host-based patching solutions remain challenged by source code availability and the prohibitive cost of upgrading embedded/legacy systems. Network-based filtering includes manual rule-based filtering [26] (inherently error-prone and not scalable), generic ML-based network filtering [57], [48] (relying on statistical anomaly detection which causes disruptive false positives and fails against low-frequency attacks like APTs). It is also worth noting that generic LLMs can be used to generate patches or filtering rules via manual prompts. However, the generic LLMs [11], [18], [61] are plagued by hallucination and non-determinism, generating seemingly correct but functionally flawed patches and filtering rules.

Current vulnerability fixing mechanisms are ill-equipped to address large-scale 1-day/n-day exploitation targeting diverse networked infrastructure. Consider the compromise of millions of smart meters within an Advanced Metering Infrastructure (AMI) vulnerable to Smart Grid attacks like load oscillation [38]. Fixing such a vulnerability via host-based patching demands expert analysis and expensive deployment/verification across millions of devices. Network-based filtering alternatives, which rely on manual rules or anomaly detection, similarly fail to scale effectively to millions of AMI endpoints while introducing unacceptable risks of disrupting critical grid operations through false positives. We argue that existing approaches fail to address large-scale 1-day/n-day exploits effectively due to three fundamental limitations:

- **Scalability:** Patch and filtering rule generation relies critically on domain experts to manually analyze code and craft fixes—a process that is inherently unscalable. This is acutely demonstrated by vulnerabilities like Log4j, which impacted millions of heterogeneous devices from Apache servers to Smart City cameras. Manually generating tailored fixes for such diversity is prohibitively slow and costly.
- **Compatibility:** Host-based patching suffers severe compat-

ibility issues, particularly for embedded or legacy systems. A Log4j patch designed for Windows servers, for instance, will typically fail or malfunction on resource-constrained smart cameras due to platform and dependency mismatches.

- **Error-susceptibility:** Both patching and filtering approaches are intrinsically susceptible to errors, as validation remains exceptionally challenging across diverse device functionalities. For example, it is hard to verify that an AMI smart meter’s firmware patch installs correctly and will not disrupting critical grid operations.

To overcome these limitations, we introduce REFN (Reinforcement-Learning-From-Network): a novel framework that trains Large Language Models (LLMs) to generate network filters that can be deployed on edge security gateways to prevent 1-day/n-day exploits. REFN’s core innovation include leveraging Reinforcement Learning (RL) driven by real-time network rewards – not human feedback (RLHF), enabling autonomous adaptation to evolving 1-day/n-day threats. REFN ensures scalability as its RL-trained LLMs automatically generate tailored filters, eliminating dependency on manual patch/-fix generation. REFN guarantees compatibility via unified deployment on edge security gateways (e.g., Amazon Eero) abstracts away device-specific complexities, ensuring broad coverage including embedded/legacy systems. REFN is error-unsusceptible, as it can continuously perform online validation against real network traffic, and can correct erroneous filters by adjusting the online reward generated from network dataplane.

In developing REFN, we address three core challenges in *training LLMs for exploit prevention*: 1) **Limited vulnerability-fixing expertise:** LLMs trained on general-purpose data lack internalized vulnerability-fixing inference capabilities (e.g., unaware of day-1 vulnerability); 2) **Language-to-network gaps:** LLMs optimized for natural language (e.g., via RLHF) struggle to translate textual descriptions into executable network enforcements due to structural mismatches between linguistics and network; 3) **LLM hallucination and non-determinism:** LLMs generate inconsistent or erroneous outputs across iterations, producing unreliable filtering rules that disrupts normal functions.

To address these challenges, REFN introduces three novel designs: 1) **Agentic-RAG-based Knowledge Distillation:** dynamically retrieves and internalizes vulnerability intelligence from security databases, CVE reports, and historical fixes, enabling contextual reasoning for precise filter generation. 2) **RL-from-VNF Pipeline:** employs reinforcement learning (RL) driven by real-time rewards from Virtualized Network Function (VNF) to translate textual vulnerability descriptions into protocol-aware network enforcements. 3) **Online Agentic Validation:** enforces output reliability through real-time network feedback loops that penalize errors and refine filters during deployment.

The main contributions of this paper are as follows:

- **Proof-of-Concept:** We demonstrate the viability of training LLMs for preventing massive-scale 1-day/n-day exploits.
- **REFN Framework:** We introduce a novel framework that

leverages Reinforcement Learning (RL) driven by real-time network rewards – instead of human feedback (RLHF), to generate vulnerability-fixing filters. REFN is scalable, compatible and error-unsusceptible. Code available: <https://github.com/REFN2025/REFN2025>.

- **Security-Specialized LLM Model:** We provide the RL-trained vulnerability-fixing LLM specifically effective against 22 families of 1-day/n-day exploits. Model available: <https://huggingface.co/llhview/HuggingFace/tree/main>.
- **RL Dataset for Exploit Prevention:** We present the first dataset enabling Reinforcement Learning of LLMs to prevent 1-day/n-day exploits, covering 22 families of exploits and 65 types of devices. Dataset available: <https://github.com/REFN2025dataset/REFN2025/tree/master>.
- **Rigorous Evaluation:** Using our RL dataset, we comprehensively evaluate REFN and demonstrate its *effectiveness* (21.1% higher accuracy than alternatives), its *efficiency* (Mean-Time-To-Patch of 3.65 hours) and its *scalability* (easily scale to 10K devices).

II. MOTIVATIONS AND RELATED WORKS

A. Current vulnerability fixing approaches

Current vulnerability fixing mechanisms, including host-based patching and host-based patching, can be further specified as several types of approaches:

Manual patching: This approach involves human-driven updates to software/hardware systems through three core phases: patch generation, installation, and validation. Security administrators must manually execute each step, requiring substantial effort and expertise while offering limited scalability.

Patch management software: Developed to automate patching workflows [17], [12], [10], [20], [15], [27], [14], [21], [25], these tools primarily streamline delivery and installation for standard computing environments. However, they are ineffective for embedded or legacy systems due to platform-specific dependencies and lack of validation.

Generic ML-based patching: InferFix [50] employs ML-assisted repair; GraphSPD [63] and PAVUDI [45] utilize GNNs for patch analysis; PatchRNN [64] models patches via RNNs; SPI [66] identifies patches through commits. ML solutions [64], [63], [45], [66], [50] are constrained by source code availability and face prohibitive deployment costs for legacy/embedded devices.

Manual network filtering: This approach requires the security admins to craft rule-based filters or access control rules manually [47], [49], inherently limited by human error and impractical for large-scale deployments. The lack of automation also impedes rapid response to emerging threats.

Generic ML-based network filtering: These methods detect threats through statistical traffic anomalies [57], [48], [40], [39], [44], [42], [54]: Kitsune [57] uses autoencoder reconstruction deviations; Bartos et al. [40] and FlowLen [39] apply flow-sampling classifiers; Whisper [44] analyzes periodicity via spectral decomposition; DeepLog [42] models system

logs. Despite sophistication, they remain vulnerable to low-frequency APT evasion due to reliance on pattern deviations. **LLM based patching/filtering:** By prompting generic LLMs [11], [18], [61], this approach generates patches or filtering rules. However, it suffers from critical hallucination and non-determinism issues, producing seemingly valid but functionally flawed outputs that compromise security efficacy.

B. Issues with current approaches

Scalability issue: Current vulnerability remediation mechanisms face significant scalability challenges. The sheer diversity of vulnerable devices and systems—encompassing complex infrastructures, numerous software applications, and millions of endpoints—makes broad protection difficult. A prime example is the Log4j vulnerability, which impacted devices ranging from Apache servers to consumer appliances like Siemens refrigerators. Manually generating and validating patches or filtering rules for each unique vulnerability-device combination demands immense domain expertise and is impractical at scale. Furthermore, patch deployment itself consumes substantial time, manpower, and technical resources, a burden particularly acute for organizations with limited budgets. This scalability gap is exacerbated by the emergence of LLM-empowered exploitation tools (e.g., HackerGPT [31], WormGPT [29]), which dramatically enhance attackers’ ability to launch large-scale exploits.

Compatibility issue: Existing approaches also struggle with compatibility across diverse device ecosystems. End-of-life devices often lack manufacturer support and receive no security updates, leaving them persistently vulnerable. Many older or resource-constrained devices lack the hardware or software capability to accept firmware updates at all. Proprietary, closed-source systems present another barrier, as they cannot be audited by the community or have third-party patches developed. Furthermore, patching complex hardware like medical equipment or industrial control systems frequently requires specialized expertise that may be unavailable.

Error-susceptibility issue: Current patching mechanisms are inherently susceptible to errors. Fixes often require modifications across multiple system components, increasing the risk of implementation mistakes that can cause instability or functional loss. Conflicts with existing software or hardware versions are common, and managing dependencies adds further complexity. Crucially, the process relies heavily on manual intervention for both deployment and validation, introducing significant opportunities for human error. It is inherently challenging to anticipate all potential issues—especially in complex systems or embedded devices—meaning even validated patches can inadvertently introduce new problems or unexpected behavior. For example, the ML-based network filtering approaches may flag the benign accesses on a newly joined devices as malicious as it is rarely seen deviation.

C. New vantage point to prevent 1-day/n-day vulnerabilities

To tackle the *compatibility* and *scalability* issue, security vendors are shifting the vulnerability fixing function from

host-side to *Edge Security Gateways (ESG)*, including Amazon eero [2], Cisco Meraki [3], Netgear Orbi [6] and Linksys Velop [5]. In such network-fix paradigm, the vulnerability fixing is enforced as network filtering on the edge security gateways. The remote cloud services is responsible for generating the filtering rules and installing them on the edge security gateways. For example, the Cisco Talos Intelligence cloud service can generate a network filtering for Log4j, and deploy it on Meraki MX edge routers to detect and block Log4j exploits [4], [7]. The gateways hosting the vulnerability fixes are unified platforms such as Cisco IOS [13], Raspberry PI [24] or OpenWRT [22]. The network-based patches only need to adapt to several unified edge platforms instead of heterogeneous vulnerable devices. Unlike current host-based patching mechanisms - which are hard to upgrade and slow to respond to emerging vulnerabilities, the network-fix update [52] can be performed in seconds.

III. OVERVIEW

In this section, we present the overview of REFN- a scalable, compatible and error-unsusceptible framework that trains Large Language Models (LLMs) to autonomously generate and deploy network filters on Edge Security Gateways (ESG), and prevent 1-day/n-day exploitations across heterogeneous networked devices.

Threat model: Before detailing REFN’s design, we establish the following assumptions regarding adversarial capabilities and system trust boundaries:

Adversarial Capabilities: 1) The adversary can exploit all 1-day/n-day vulnerabilities in exposed devices; 2) The adversary can utilize LLM-based automation tools (e.g., HackerGPT [31], WormGPT [29]) to generate and scale exploits.

System Trust Assumptions: 1) The Edge Security Gateway (ESG) is a secure, trusted component and cannot be compromised by the attacker; 2) Cloud servers responsible for training the LLMs, generating and deploying network filters onto the edge security gateways are trusted and secure; 3) The edge security gateways have the capability to enforce network filters that inspect and block malicious traffic directed at vulnerable devices; 4) For encrypted traffic, we assume the edge security gateway can either decrypt the traffic (which is common for business ESGs such as Cisco Meraki [3]) using protocols like mTLS [58], or can infer malicious intent (which is also common for ESG Intelligence such as the Cisco Talos [4]) through analysis of metadata or unencrypted portions (e.g., packet headers, certificates).

Baseline approach: To address the core challenges of scalability, robustness, and compatibility in exploit prevention, we can design a baseline framework integrating the strategic vantage point of Edge Security Gateways (ESGs) identified in Section II, the generative capabilities of Large Language Models (LLMs), and the validation efficiency of Virtualized Network Function (VNF) to mitigate LLM hallucination. This streamlined pipeline operates as follows: upon disclosure of a 1-day/n-day vulnerability, an LLM-based generator dynamically crafts tailored filter rules leveraging Retrieval Augmented

USER: What is the key pattern for Sambacry attack?
 ASSISTANT: "Sambacry" is not a word or term that is commonly used or recognized. It is not a term that is commonly associated with any specific field of study or industry. It is possible that this word or term was created or invented by the person who is using it in this chat. Without more information or context, it is difficult to provide a more comprehensive answer to your question.

Figure 1: Gemma-3-4B (May-2025) lacks expertise.

Generation (RAG), addressing scalability by eliminating manual rule creation and enabling rapid response within the first day. These rules then undergo rigorous validation through VNF testing against synthetic exploit traffic and benign traffic samples, which identifies and filters erroneous outputs before deployment. Finally, validated rules are propagated to ESGs, achieving compatibility by providing a unified enforcement layer that protects diverse connected devices across heterogeneous network environments.

Challenges: While the baseline approach shows promise, several practical challenges hinder its implementation:

- **Limited Vulnerability-Fixing Expertise:** Existing LLMs [18], [11], [61], [36] are trained on general-purpose datasets and lack specialized knowledge in niche domains like vulnerability remediation (Figure 1). While techniques like Retrieval Augmented Generation (RAG) can incorporate vulnerability-related context into prompts, they fail to internalize domain-specific expertise. This limitation prevents LLMs from reliably inferring accurate filter fixes for novel 1-day/n-day vulnerabilities.
- **Gaps Between Language and Network:** Current LLMs are designed and optimized for natural language interactions, as exemplified by training paradigms like Reinforcement Learning from Human Feedback (RLHF). This creates a significant semantic and structural disconnect when applied to network security. Their effectiveness in processing raw network-layer data (e.g., packets) or generating precise protocol-specific rules remains unproven, limiting their practical utility for real-world network enforcement tasks.
- **LLM Hallucination and Non-Determinism:** LLM outputs are inherently non-deterministic, often producing inconsistent or contradictory rules across repeated iterations. This instability introduces reliability risks, as identical inputs may generate divergent outputs—some of which could be erroneous or unsafe. Such unpredictability undermines trust in automated systems requiring consistent, repeatable results for critical tasks like vulnerability-fixing filtering.

REFN’s ideas: To address the above challenges, we propose REFN on top of the baseline approach, as shown in Figure 2. REFN is built on a simple yet powerful premise: training LLMs into dynamic, network-aware vulnerability fixing engines that can deliver fixes on day one. To achieve this, the system directly tackles the three core challenges through strategic design. The first idea is to *close the expertise gap via knowledge distillation*. Instead of relying on generic LLM knowledge, REFN distills historical vulnerability fixes (e.g., past CVEs, patches) into the model during training. This equips the LLM with an implicit playbook of remediation strategies, enabling it to infer fixes for new 1-day

Challenge 1:
Limited
Vulnerability
fixing expertise

Challenge 2:
Gap between
language and
network

Challenge 3: LLM
Hallucination &
Non-determinism

Vulnerable
 Devices

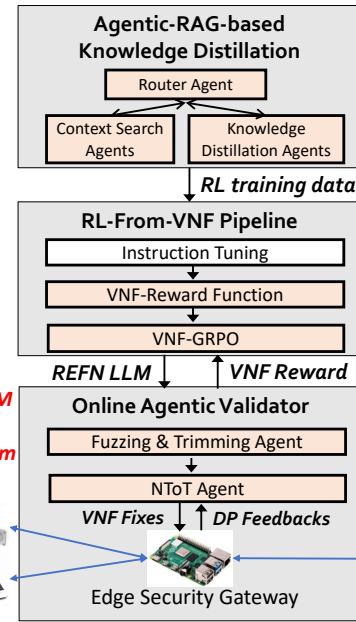


Figure 2: REFN’s workflow.

vulnerabilities by recognizing patterns from past fixes—even if the new vulnerability differs superficially. The second idea is to *translate language to network actions via RL-from-VNF pipeline*. REFN treats network filtering as a “language” the LLM must learn. Through reinforcement fine-tuning (ReFT), the model is trained to enforce vulnerability-fixing filters on raw network packets (e.g., dropping malicious payloads) rather than generating text. Rewards are tied to real-world outcomes - blocking malicious packets while preserving legitimate packets. The third strategy is to *punish hallucination via dataplane validation*. Every LLM generated filter is validated as Virtualized Network Function (VNF) with both benign and malicious traffic. Filters that fail to block attacks or disrupt legitimate flows are punished via a VNF-based online reward function, creating a feedback loop that force out the LLM hallucination in the training stage.

REFN’s workflow: As illustrated in Figure 2, REFN transforms raw 1-day/n-day vulnerability context into reliable security enforcements on Edge Security Gateways (ESGs) through three core components:

Agentic-RAG-Based Knowledge Distillation: This pipeline integrates agent-based systems, Retrieval-Augmented Generation (RAG), and knowledge distillation to transfer vulnerability-fixing expertise from powerful-but-expensive LLMs to specialized models (efficient for training fixes for new vulnerabilities). This architecture features three autonomous agents: a Router Agent directing queries, Context Search Agents retrieving vulnerability context, and Knowledge Distillation Agents extracting structured inference examples (e.g., filter rules). This automated pipeline ensures RL-optimized knowledge transfer while eliminating manual expertise bottlenecks (detailed in Section IV).

RL-From-VNF Pipeline: Distilled knowledge is processed

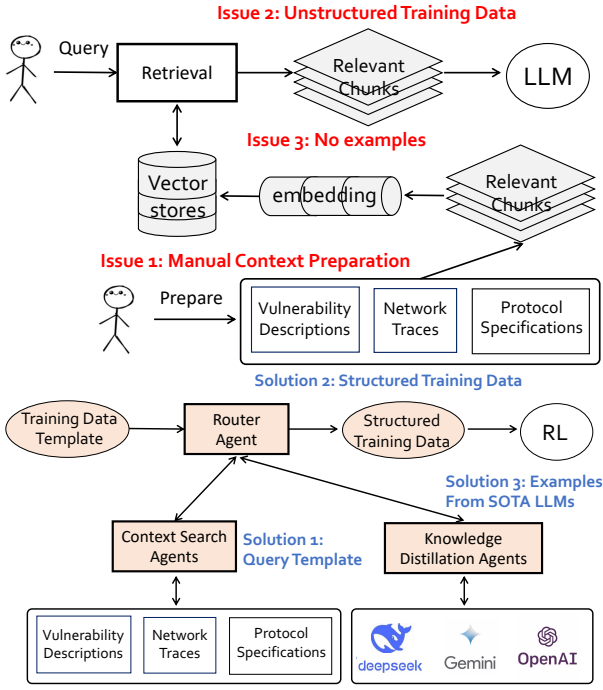


Figure 3: Common RAG (top) vs REFN’s Agentic-RAG-Based Knowledge Distillation (bottom).

by the RL-From-VNF LLM training pipeline, which replaces human feedback with Virtualized Network Function (VNF) validation, which automatically generate rewards/penalties based on security enforcement outcomes. To improve training inefficiency, REFN presents the *VNF-GRPO* algorithm that integrates GRPO optimization, LoRA adapters and VNF. The pipeline’s *VNF-Based Reward Function* evaluates each LLM-generated filter on both benign and malicious traffic samples, training a lightweight network-aware LLM capable of translating textual vulnerability contexts into practical vulnerability-fixing enforcements on the ESGs (detailed in Section V).

Online Agentic Validator: To address the LLM hallucination and non-determinism challenge, REFN’s Online Agentic Validator will validate the filters and generate quantitative rewards for the RL-From-VNF pipeline. Unlike coarse-grained network validation (e.g., binary block/no-block decisions), this validator employs fine-grained VNF-specific logic and can fuzz on LLM-generated filters to reward “near-correct” outputs (key for RL training convergence). This approach provides granular feedback for Reinforcement Fine-Tuning (ReFT), balancing precision with training stability while reducing hallucinations. The design enables effective RL convergence and reliable exploit prevention (detailed in Section VI).

IV. AGENTIC-RAG-BASED KNOWLEDGE DISTILLATION

When a 1-day/n-day vulnerability emerges, the Reinforcement Learning (RL) process requires curated training data, which currently is prepared manually with the common Retrieval-Augmented Generation (RAG) (Figure 3 top). This conventional approach suffers from three critical limitations: labor-intensive preparation of vulnerability contexts (descrip-

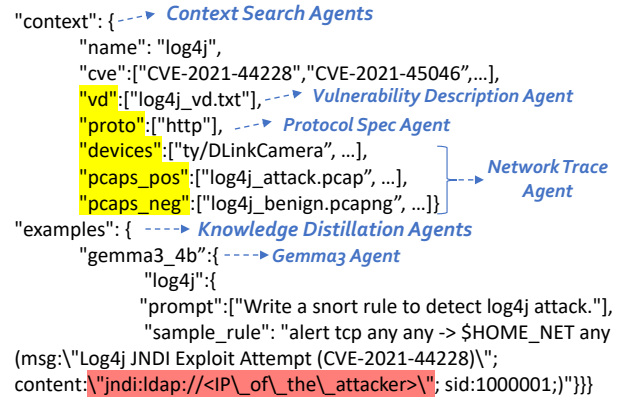


Figure 4: Training data template.

Log4j vulnerability was reported to apache by Chen Zhaojun of the Alibaba cloud security team on 24th November 2021 and published in a tweet on 9th December 2021. Apache software foundation assigned a maximum severity score of 10/10. The vulnerability allows attackers to remote code execution and the payload string looks like `\"$jndi:ldap://attacker.com/aj\"`. Lots of organization-affected services include Cloudflare, apple iCloud, Minecraft: java edition, stream, Tencent QQ, and Twitter.

Figure 5: Segment of Log4j [28] vulnerability description.

tions, network traces, protocol specifications), unstructured blended data that compromises RL effectiveness, and inability to distill knowledge from SOTA LLMs (DeepSeek-R1, Gemini-2.5, GPT-4o). To overcome these constraints, we introduce the Agentic-RAG Knowledge Distillation pipeline (Figure 3 bottom), which integrates agent-based systems, RAG, and knowledge distillation to transfer capabilities from expensive LLMs (e.g., DeepSeek-R1-671B) to efficient-to-train specialized models. The pipeline employs three autonomous components: a *Router Agent* that dynamically directs queries, *Context Search Agents* that retrieve vulnerability intelligence, and *Knowledge Distillation Agents* that extract structured inference samples from SOTA LLMs—automating knowledge transfer while ensuring RL-optimized, structured outputs.

A. Router Agent

When a 1-day/n-day vulnerability first emerges, REFN initiates training data preparation by sending a structured prompt to the Router Agent (RA). In this prompt, REFN presents a well-designed RL-training data template (Figure 4) that captures key RL-training structures for exploit prevention. Upon receiving the training-data-preparation prompt, the agent decomposes data preparation tasks based on the template and delegates them to specialized context search agents and knowledge distillation agents. Three context search agents operate in parallel: (1) The vulnerability description agent extracts vulnerability details from CVE databases and security advisories [9], such as the Log4j vulnerability description in Figure 5; (2) The protocol specification agent identifies relevant network protocols and their specifications for trace parsing; (3) The network trace agent retrieves packet captures (from public repositories [33], [46], [1]), populating positive/negative traffic examples (“pcap_pos/pcap_neg”) and device context. Concurrently, the knowledge distilla-

tion agent queries SOTA LLMs for vulnerability-fixing filter examples. While these examples may contain inaccuracies (e.g., Snort rules content “jndi:ldap://IP_of_the_attacker” with correct prefixes like “jndi:ldap” but erroneous continuations like “IP_of_the_attacker”), they provide valuable structural patterns. This enriched data significantly enhances Supervised Fine-Tuning (SFT), ultimately improving output quality throughout the RL pipeline.

B. Context Search Agents

The context search agent prepares critical vulnerability context, including descriptions (vd), network traces (pcaps), and protocol specifications (proto). A naive non-agent approach relies on conventional Retrieval-Augmented Generation (RAG) with in-context learning (ICL): first crawls vulnerability descriptions from sources like NVD [9], network traces from repositories [33], [46], [1], and protocol specifications from IETF [32], then chunks this heterogeneous data indiscriminately for ICL prompting. However, the unstructured blending of data types significantly compromises retrieval efficacy [56].

To address this limitation, REFN introduces a *vulnerability-to-trace pair-ranking* mechanism that explicitly correlates vulnerability descriptions vc with network traces nc , enhancing context preparation efficacy. Our approach segments the vulnerability description context vc into discrete sentence-level contexts x^{vc} and the network trace context nc into packet-level contexts x^{nc} . It then generates sentence-packet context pairs (x^{vc}, x^{nc}) and algorithmically selects the subset of pairs exhibiting the highest semantic correlation $\{(x^{vc}, x^{nc})\}$. Based on previous research [56], suppose x is the testcase, y is the ground truth for the testcase, C is the space of all ground truth, and there are k labeled examples $(x_1, \hat{y}_1), \dots, (x_k, \hat{y}_k)$, then the LLM output is: $\argmax_{y \in C} P(y|x_1, \hat{y}_1, \dots, x_k, \hat{y}_k, x)$. Consider each sentence-packet context pair (x^{vc}, x^{nc}) as a demonstration, the LLM output can be approximated by:

$$\argmax_{y \in C} P(y|[(x_1^{vc}, x_1^{nc}), \dots, (x_k^{vc}, x_k^{nc})]_{top_k}, x), \quad (1)$$

where $[(x_1^{vc}, x_1^{nc}), \dots, (x_k^{vc}, x_k^{nc})]_{top_k}$ are the top k most correlated sentence-packet context pairs. Such approximation is feasible because $nc \in C$, i.e., the network context nc augmented from the vulnerable device’s traces shares the same ground truth space C with the desired output y . In other words, the LLM output y will be close to vulnerability descriptions vc ’s real network traffic pattern.

To implement the sentence-packet context pair ranking efficiently, REFN provides a *join labeling* operation:

$$[(x_1^{vc}, x_1^{nc}), \dots, (x_k^{vc}, x_k^{nc})]_{top_k} = vc \bowtie_{con} nc, \quad (2)$$

where con is $d(x^{vc}, x^{nc}) < top_k(\{d(x^{vc}, x^{nc})\})$ ¹. More specifically, $d(x^{vc}, x^{nc})$ is the distance metrics between sentence context x^{vc} and the packet context x^{nc} . We leverages FastText [51] metrics for efficient distance calculation.

¹ $k = 10$ unless specified otherwise.

C. Knowledge Distillation Agents

The Knowledge Distillation (KD) Agents extract and structure actionable insights from SOTA LLMs to prepare data for the Supervised Fine-Tuning (SFT) stage of RL training. Leveraging transfer learning principles, these agents adapt general knowledge from expensive Pre-trained Language Models (PLMs) to the specific downstream task of vulnerability remediation through three key mechanisms:

Knowledge Inheritance Mechanism: PLMs acquire intrinsic linguistic patterns (syntax, semantics, knowledge representation) through self-supervised pretraining on large corpora. The distillation framework inherits this knowledge as initialization, avoiding inefficient learning from scratch. More specifically, given a list of known vulnerabilities $\{v\}$, a PLM M and a filter generation prompt p , the KD agents will generate a list of filters $\{f\} = M(\{v\}, p)$ and prepare the knowledge as $(\{v\}, p, \{f\})$ for the SFT process.

Essence of Task Adaptation: By minimizing the vulnerability-fixing-specific loss function on the structured training data, the model parameters in the SFT process are adjusted to adapt to the filter generation domain:

$$\theta_{SFT} = \arg \min_{\theta} \sum_{(x,y) \in \mathcal{D}} \mathcal{L}(f(x; \theta), y) \quad (3)$$

where θ represents the model parameters, \mathcal{D} denotes the labeled dataset, \mathcal{L} is the loss function (e.g., cross-entropy), and $f(\cdot)$ is the model’s predictive output.

Catastrophic Forgetting Control: In SFT, a low learning rate strategy (typically 1-10% of pretraining rates) combined with early stopping preserves foundational inference knowledge (e.g., generating filter rule) while accommodating vulnerability-fixing requirements (e.g., correct rule format).

Integrated into REFN’s *Instruction Tuning* stage (Section V), this distillation framework enables structured extraction of filtering rule patterns from SOTA LLMs, ensuring effective knowledge transfer to specialized security models.

V. RL-FROM-VNF PIPELINE

This section introduces the RL-From-VNF (RFV) pipeline, a novel framework that eliminates human feedback from the reinforcement learning loop. We first contrast RFV against state-of-the-art approaches (RLHF, DPO) to establish its unique network-driven paradigm. Next, we detail the pipeline’s operational workflow and present its VNF-GRPO optimization algorithm. Finally, we analyze the core VNF reward function, demonstrating how network-generated feedback replaces human evaluations to autonomously drive RL training.

A. Comparison with RLHF and DPO

As shown in Figure 6, we compare the RL-From-VNF pipeline with two SOTA RL methodologies: Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO).

RLHF: This approach includes three key stages. First, supervised fine-tuning (SFT) leverages human-generated instruction data to align a pre-trained model’s initial outputs

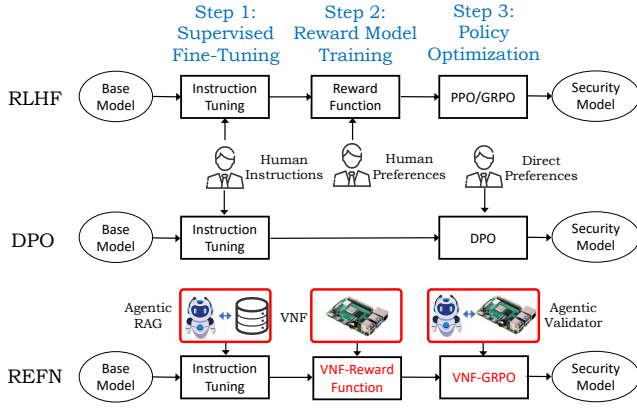


Figure 6: RL-From-VNF Pipeline.

with security objectives, such as filter rule generation. Second, reward modeling directly incorporates human feedback, where annotators rank or rate model responses to train a preference predictor that quantifies qualitative judgments into reward signals. Finally, policy optimization employs algorithms like PPO [59] or GRPO [60] to iteratively refine the model against this reward predictor, embedding human preferences throughout the optimization cycle.

DPO: This approach streamlines preference integration while maintaining human dependency. Similar to RLHF, DPO begins with SFT using human-crafted instruction-response pairs for initial task alignment. However, its policy optimization phase diverges significantly: annotators rank output pairs, and the DPO algorithm directly optimizes for preferred responses using a contrastive loss function that compares favorable against dispreferred outputs. This approach eliminates the explicit reward modeling step required in RLHF, instead embedding human feedback directly within the optimization process while reducing implementation complexity.

Reliance on human feedback: The reliance on human feedback in both RLHF and DPO workflows makes them ill-suited for fixing 1-day/n-day vulnerabilities, which demand real-time, automated responses. First, human feedback introduces latency: RLHF requires iterative human annotation to train reward models, while DPO depends on pre-collected human preference datasets. These steps create delays incompatible with the time-sensitive nature of 1-day/n-day exploits, where threats must be neutralized within hours or minutes of discovery. Second, human feedback loops struggle to scale with the sheer volume and diversity of network traffic—annotators cannot feasibly label every potential intrusion pattern in dynamic network environments. Third, human biases or inconsistencies in labeling could inadvertently prioritize non-critical alerts or miss adversarial evasion tactics, undermining precision.

RL-From-VNF idea: We propose the RL-From-VNF (RFV) pipeline—a paradigm where Virtualized Network Functions (VNFs) replace human feedback in the reinforcement learning loop. Unlike traditional RL that relies on handcrafted reward functions (e.g., “maximize throughput”), RFV leverages security VNFs deployed on edge gateways to autonomously validate vulnerability-fixing filters. These VNFs generate real-time

Algorithm 1 VNF-GRPO Algorithm

- 1: **Input:** Initial policy π_θ with parameters θ , batch of trajectories $\mathcal{D} = \{(x, e, p, y_\theta)\}$, VNF reward function \mathcal{R} , learning rate β , regularization factor λ , clipping factor ϵ , number of group members N
- 2: **Output:** Updated policy π'_θ with parameters θ'
- 3: // Initialize policy parameters
 $\theta \leftarrow \theta^{(0)}$
- 4: **for** each member i from 1 to N **do**
- 5: // Collect trajectory \mathcal{D}_i from member i
- 6: $VNF_i \leftarrow \text{VNF_GENERATOR}(x, e, i)$
- 7: $\mathcal{D}_i \leftarrow \text{VNF_VALIDATOR}(p \rightarrow VNF_i)$
- 8: // Compute VNF reward r_i
- 9: $r_i \leftarrow \mathcal{R}(\mathcal{D}_i)$
- 10: // Compute group-relative reward R_i
- 11: $R_i = \frac{r_i}{\sum_{j=1}^N r_j}$
- 12: // Compute the advantage function A_i
- 13: $A_i \leftarrow \text{VNF_ADVANTAGE}(R_i)$
- 14: **end for**
- 15: **for** each member i from 1 to N **do**
- 16: // Compute the surrogate objective function:
 $\mathcal{L}_i(\theta) = \text{SURROGATE_OBJ}(\pi_\theta, A, \epsilon)$
- 17: // Compute Gradient Ascent and Update the policy:
 $\theta_i = \theta_i + \beta \cdot \nabla_\theta \mathcal{L}_i(\theta)$
- 18: **end for**
- 19: **for** each agent i from 1 to N **do**
- 20: Apply regularization to stabilize the policy update:
 $\mathcal{L}_{\text{regularized}}(\theta) = \mathcal{L}(\theta) - \lambda \cdot \text{penalty}(\theta)$
- 21: Update the policy parameters θ_i with regularization:
 $\theta_i = \theta_i - \lambda \cdot \nabla_\theta \text{penalty}(\theta)$
- 22: **end for**
- 23: **Return:** Updated policy parameters θ'

reward/penalty signals based on concrete security outcomes, such as successful malicious traffic blocking without disrupting benign flows. This approach eliminates human subjectivity while harnessing domain-specific validation logic to guide RL optimization. By integrating VNF validation logic (e.g., firewall checks) as the reward generator, RFV eliminates manual reward engineering—a significant departure from conventional RL systems. This architecture directly enables zero-touch networking principles by establishing closed-loop automation: security policies self-optimize through continuous feedback from live network enforcement, requiring minimal human intervention while maintaining context-aware precision.

B. VNF-GRPO Algorithm

Algorithm 1 presents the VNF-GRPO algorithm and its differences (in blue) with the basic GRPO algorithm [60].

Input: The standard GRPO inputs include initial policy π_θ with parameters θ , learning rate β , regularization factor λ , clipping factor ϵ , number of group members N . The two distinguished inputs include batch of trajectories \mathcal{D} and VNF reward function \mathcal{R} . The basic GRPO batch of trajectories $\mathcal{D} = \{(x, e, y)\}$ is tuples of question x , CoT e and answer

y . However, since the dependency of Human Feedback is removed, REFN's batch of trajectories' answer y_0 is an empty set. The VNF feedbacks relies on the processing result on the online packets p . Therefore, REFN's batch of trajectories can be denoted as $\mathcal{D} = \{(x, e, p, y_0)\}$.

Update trajectories, rewards and advantages from VNF:

After the policy parameters are initiated (Line 3), the trajectories \mathcal{D}_i will be updated for each member i (Line 5-7). The VNF_GENERATOR will generate a virtualized function VNF_i for each member i (Line 6). The VNF_GENERATOR will update the trajectories based on the packets processing result of the VNF $p \rightarrow VNF_i$ (Line 7). The VNF reward function \mathcal{R} will compute the reward r_i based on the trajectories (Line 9), which is converted into the group-relative reward R_i (Line 11). The VNF_ADVANTAGE function will compute the advantage function A_i for member i based on the the group-relative reward R_i , which estimates the difference between the expected reward for an action and the average reward. Traditionally, the human feedbacks decide which actions are preferable given a certain state. In REFN, it is replaced by the VNF feedbacks via the VNF_ADVANTAGE function.

Update Policy with Group Relative Consideration: This part includes the step to Compute Surrogate Objective and the Gradient Ascent on Surrogate, which follows the basic GRPO. The $SURROGATE_OBJ(\pi_\theta, A, \epsilon)$ function is:

$$\mathbb{E} \left[\min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A(s, a), \text{CLIP}(\pi_\theta, s, a, \epsilon) A(s, a) \right) \right], \quad (4)$$

where π_θ is the policy with parameter θ , A is the advantage function, a is the action, s is the state, ϵ is the clipping factor. The CLIP function is:

$$\text{CLIP}(\pi_\theta, s, a, \epsilon) = \text{clip} \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}, 1 - \epsilon, 1 + \epsilon \right), \quad (5)$$

After that, the Gradient Ascent on Surrogate is computed and the policy is updated (Line 17).

Apply Regularization for Stability: To ensure stability, a regularization term is applied based on the relative group performance (Line 20-21). This is a penalty term that discourages large changes in the policy parameters.

C. VNF Reward Function

A naive reward function implementation would assign binary rewards per benign/malicious sample (e.g., per pcap file). However, such coarse-grained signals prove insufficient for effective reinforcement learning guidance [60], necessitating finer-grained alternatives: per network flow, per Application Data Unit (ADU), or per packet. The network packets are dynamic under various network configurations (e.g., switch MTU) and is not suitable as the basic unit for rewarding. The network flow are hard to assemble and hard to compare, which is key for assigning reward. Consequently, we choose the ADU, which is a consecutive sequence of packets from one device to another, leveraging its established adoption in

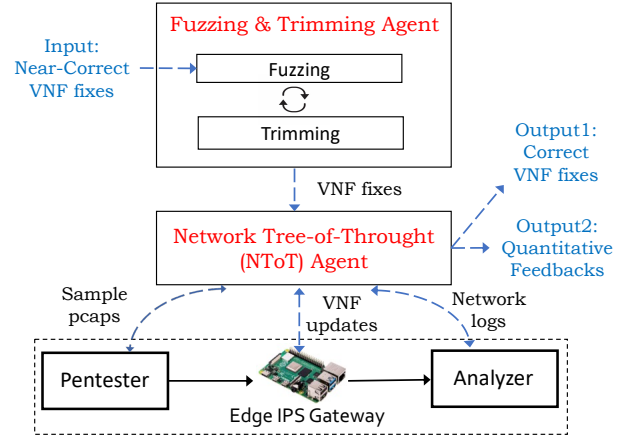


Figure 7: REFN's validation workflow.

network penetration testing methodologies [43] and inherent stability across network conditions.

Consider a benign sample (pcap) as a sequence of ADUs B and a malicious sample (pcap) as a sequence of ADUs M . A simple way to assign reward is to find out how much ADUs are blocked in B (FP) and how much ADUs are allowed in M (FN). However, this approach fundamentally misrepresents malicious sample M , where most ADUs constitute benign background traffic and only a small portion of ADUs in M is related to attack payloads. Penalizing these benign ADUs within M introduces contradictory signals that destabilize reinforcement learning. To resolve this distortion, we apply a differentiation operation to isolating and excluding benign ADUs in M before reward calculation:

$$M - B = \{m : m \in M \& \& m \notin \text{PairRank}(M, B)\} \quad (6)$$

where $\text{PairRank}(M, B)$ pairs the ADUs in M and B , rank them by their similarity and output the same ADUs in sequence M and sequence B .

Based on these observations, we define the reward function:

$$\mathcal{R} = 2pc/(p + c) \quad (7)$$

which is basically the F1-Score reward that address the unbalanced data (malicious data is much fewer than benign data), where the precision reward $p = \frac{TP(M-B)}{TP(M-B) + FP(B)}$, and the recall reward $c = \frac{TP(M-B)}{TP(M-B) + FN(M-B)}$. Note that in the above reward calculation, instead of tracking all the TP/TN/FP/FN in benign ADU sequence B and malicious ADU sequence M , we optimized to only track TP/FN in $M - B$ and FP in B . Given that M is much less than B , such optimization would greatly reduce the validation cost while providing effective reward to guide the RL process for the unbalanced security data.

VI. ONLINE AGENTIC VALIDATOR

To address LLM hallucination issues, REFN introduces an Online Agentic Validator, as shown in Figure 7. This solution targets two critical RL failure modes: 1) *Near-correct outputs*: generated VNF fixes that are near-correct but contain subtle discrepancies; 2) *Feedback granularity gap*:

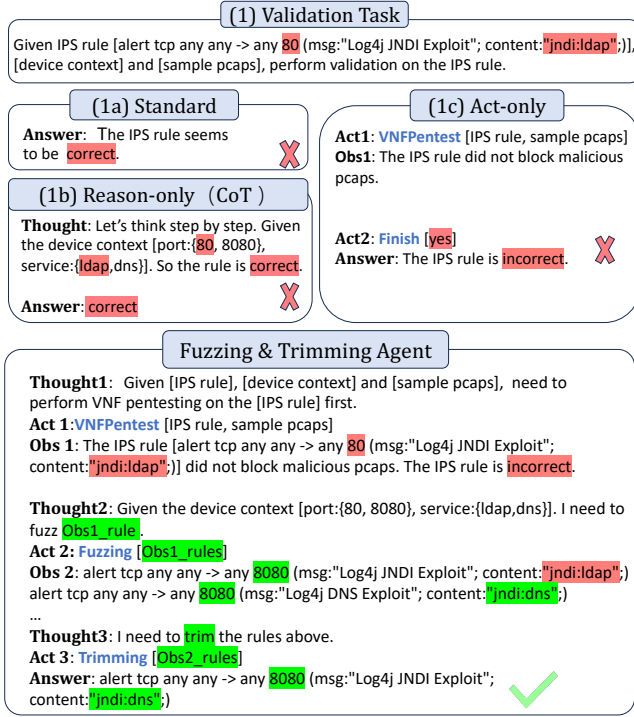


Figure 8: REFN’s fuzzing & trimming agent.

require quantitative diagnostic feedback to drive reinforcement learning beyond binary validation. REFN’s validator resolves these challenges through two integrated components: 1) A **Fuzzing & Trimming Agent** that bridges the correctness gap by iteratively refining near-correct VNF fixes into fully valid fixes via constraint-guided mutation; 2) A **Network Tree-of-Thought (N-ToT) Agent** that analyzes live penetration testing results to generate fine-grained metrics, providing the structured quantitative feedback required for RL optimization. This dual approach not only rectifies hallucination-induced inaccuracies but also establishes a closed-loop online system for continuous RL to prevent emerging 1-day/n-day exploits.

A. Fuzzing & Trimming Agent

The first challenge is handling *near-correct outputs*. Current LLM methodologies—including standard LLMs, Reason-only Agents, and Act-only Agents—are ill-suited for this purpose. Consider the validation task in Figure 8: given a near-correct filter rule (with a mismatched port 80 and partially incorrect content “jndi:ldap”), device context, and sample pcaps, validate the rule’s effectiveness. A standard LLM merely validates the rule’s format, falsely concluding it is correct. A Reason-only Agent performs limited context matching (e.g., checking ports 80/8080 and services ldap/dns) but also fails to detect the flaw. Only an Act-only Agent, by performing a *VNFPentest* (enforcing the rule on benign and malicious pcaps), identifies that malicious traffic bypasses the rule. However, the Act-only Agent only provides a binary pass/fail result that lacks the ability to refine the near-correct rule itself.

To address this limitation, REFN developed a fuzzing & trimming agent by extending the ReAct framework [65],

Algorithm 2 NToT-BFS

Require: dataplane ADUs ADU_D , validation agent AG_v , network protocol spec $netsp$, middlebox spec $boxspec$
Ensure: middlebox action $C_m \in (BLOCK, ALLOW, ALERT)$

- ▷ Create decision tree DT
- 1: $DT \leftarrow AG_v(netsp, boxspec)$
- ▷ Create thought generator $G()$
- 2: $G() \leftarrow AG_v(DT, netsp, boxspec)$
- ▷ Create states evaluator $V()$
- 3: $V() \leftarrow AG_v(DT, netsp, boxspec)$
- ▷ Perform NToT-BFS
- 4: $S_0 \leftarrow Init(DT)$
- 5: **for** $adu \in ADU_D$ **do**
- 6: $g = G(DT, S_t)$
- 7: $S_{t+1} = V(adu, DT, g)$
- 8: $t \leftarrow t + 1$
- 9: $C_m \leftarrow S_{t+1}$
- 10: **end for**
- 11: **return** C_m

specifically designed to refine near-correct VNF fixes into fully valid fixes. The agent employs a two-phase approach: first performing fuzzing to explore solution space through controlled randomness (“random walk”), followed by trimming to eliminate defective branches (e.g., malformed rules) and optimize search efficiency. As Figure 8 demonstrates, the agent begins by executing a *VNFPentest* (Act 1) and observe that the filter rule cannot block the malicious pcaps. However, it did not stop but continue to perform Fuzzing (Act 2) on the *near-correct* rule, generate variants via random walk towards the correct solution. During this process, invalid branches are systematically pruned. Ultimately, the agent concluded with the right filter rule (with port 8080 and content “jndi:dns”).

B. Network Tree-of-Thought (NToT) Agent

The second challenge is the *feedback granularity gap*, as quantitative feedbacks beyond binary result is required for RL process. REFN addresses this through its Network Tree-of-Thought (N-ToT) Agent, which analyzes live penetration testing results to generate structured quantitative feedback for RL optimization. As illustrated in Figure 7, the workflow begins when the NToT agent receives VNF fixes from the Fuzzing & Trimming agent and deploys them to the Edge IPS Gateway via VNF updates. The agent then coordinates penetration testing by: (1) configuring benign and malicious test cases for the Pentester, and (2) deploying expected dataplane actions to the Analyzer. During execution, the Pentester conducts tests while the Analyzer compares observed traffic against expected actions, returning network logs to the NToT agent. This enables the agent to validate VNF fixes and derive quantitative RL feedback—a complex task requiring inference of middlebox enforcement from dynamic dataplane traffic. For example, during Log4j vulnerability testing, the agent must precisely calculate the blocking rate for malicious payloads (e.g., “jndi:dns”) while measuring false positive rates on benign traffic. While LLM agents typically employ Chain-of-Thought (CoT) for multi-stage inference [16], this method proves inadequate for middlebox analysis due to two fundamental limitations: locally, CoT fails to explore alternative

Exploit Family	Vulnerability Examples
Log4j	CVE-2021-44228, CVE-2021-45046
SambaCry	CVE-2017-7494
Mirai	CVE-2020-5902
Modbus Injection	CVE-2022-1068
Spooky SSL	CVE-2022-3602, CVE-2022-3786
Eternal Blue	CVE-2017-0143, CVE-2017-0148
CryptoWall	CVE-2015-5560, CVE-2015-5122
AlexaEavesDropping	CVE-2023-33248
CiscoRouterCmp	CVE-2017-3881
CiscoRouterHttp	CVE-2024-20393
HuelightBlackout	CVE-2020-6007
PlexDVRExp	CVE-2020-5741
ToriiBot	CVE-2023-1389
Virut	CVE-2014-4114
Yakesmalware	CVE-2014-0224
Conficker	CVE-2008-4250
Gh0st RAT	CVE-2018-8174, CVE-2012-0507
Locky Ransomware	CVE-2012-0507, CVE-2015-5122
Pony	CVE-2017-11882
Bladabindi	CVE-2017-8759
WannaCry	CVE-2017-0144, CVE-2017-0145
Trojan.Valyria	CVE-2017-11882

Table I: The 22 families of 1-day/n-day exploits.

reasoning paths within a thought process; globally, it lacks state planning mechanisms to evaluate different continuation options across the inference trajectory.

To address these issues, REFN provide the NToT (Network-Tree-of-Thought) inference mechanism, as shown in Algorithm 2. The key idea of NToT is to build a decision-tree thought structure DT based on network and middlebox specifications (Line 1). Then, the agent will map the dataplane ADUs to the states of decision-tree DT and infer the middlebox actions (Line 4-11) based on the following components:

Thought generator: NToT provide a thought generator $G()$ to generate thought for each state S_t in decision-tree DT (Line 6). The thought generator is created based on the decision tree DT , the network protocol specifications $netspec$ and the middlebox specifications $boxspec$ (Line 2).

State evaluator: NToT provide a state evaluator $V()$ to evaluation multiple continuations at each state S_t based on current dataplane ADU adu , decision-tree DT and current thought g (Line 7). The state evaluator $V()$ is created based on the decision tree DT , the network protocol specifications $netspec$ and the middlebox specifications $boxspec$ (Line 3).

REFN’s NToT stores the decision tree structure and allows for the addition, deletion, or modification of nodes and edges. It also supports the inclusion of new protocols. REFN enables the easy analyze the live penetration testing results to generate fine-grained metrics and provide structured quantitative feedback (based on the VNF-Reward Function in Section V) required for RL optimization.

VII. DATASET AND IMPLEMENTATION

Dataset: We present the first dataset that enables RL-training of LLMs to prevent 1-day/n-day exploits. The dataset is generated by using REFN’s Agentic-RAG-Based Knowledge Distillation module to gather four key parts of data: 1) *vulnerability descriptions*: details from CVE databases and security advisories [9], such as the Log4j example in Figure 5; (2) *protocol specifications*: network protocols and their specifications [32] for trace parsing; (3) *network traces*:

packet captures (pcaps) from online repositories including NETRESEC [33], IoT-23 [46] and IoT Sentinel [55], populating positive/negative traffic examples (“pcap pos/neg”) and device context. The 22 families of 1-day/nday exploits (Table I) and benign samples from 65 types of devices. As shown in Figure 4, for each family of exploit (e.g., Log4j), there is a list of vulnerabilities (“cve” field), a vulnerability description text (“vd” field), corresponding devices (“devices” field), malicious pcaps (pcaps_pos) and benign pcaps (“pcaps_neg” field).

Implementation: We implemented REFN with 6K LoC on two desktop servers (one for RL training and one for testbed) and a edge security gateway. Each server is equipped with an NVIDIA RTX 4090 GPU (24GB VRAM), Intel Platinum 8352 CPU (36 cores), 32GB RAM, and 16TB HDD. The edge security gateway is implemented on a Raspberry Pi 4B running Snort 2.9.8.0. REFN’s base LLM model for RL training and agents is Gemma 3-4B. The agents integrates ReAct framework and is deployed using Ollama [34]. The RAG is implemented using LangChain [16] (chunk size = 500 and chunk overlap = 10). The vector store is using FAISS [30].

VIII. EVALUATION

In this part, we evaluate REFN and show that:

- REFN is *effective*, with $\geq 21.1\%$ accuracy improvement and $\geq 225.9\%$ F1-Score improvement than alternatives.
- REFN is *efficient* - Mean-Time-To-Patch (MTTP) is 3.65h (95.4% improvement); the fix installation delay (iDelay) is at second-scale (10X reduction) compared with alternatives.
- REFN is *scalable* - the Batched Training Time (BTT) for 22 1-day/n-day vulnerabilities is less than 0.5 day; can easily be applied to 10000 vulnerable devices (around 300 offices) with 1.5 hours of Accumulative Downtime (ADT) in total.

A. Experiment Setting

Basic setting: We established a penetration testbed (Figure 13 in Appendix A) to evaluate REFN and alternative approaches (Table II). This testbed incorporates attack launchpads and user interfaces connected to target devices via the edge security gateway. The testbed integrates both physical devices and virtualized devices (using QEMU [23]) to host the 22 families of 1-day/n-day exploits (Table I in Section VII). The testbed leverages virtualization to enables scalable hosting of expensive/hard-to-replicate embedded devices (e.g., smart grid transformers). A dedicated server hosting the REFN framework will generate the VNF fixes and deploy them on the edge security gateway for enforcement.

Alternative approaches: As shown in Table II, we evaluated seven categories of alternative vulnerability fixing approaches for comparison: 1) manual patching: we conducted an IRB-approved study with 10 security admins patching devices using official documentation; 2) patch management software: we tested 9 business-grade solutions (detailed in Table V in Appendix C) [17], [12], [10], [20], [15], [27], [14], [21], [25]; 3) generic ML-based patching: we evaluated typical

Table II: Overall effectiveness comparison of REFN and alternative approaches.

Approaches	FPR	FNR	Accuracy	F1-Score
manual patching	0.068	0.819	0.556	0.290
patch management software	0.034	0.910	0.528	0.161
generic ML-based patching	0.017	0.881	0.551	0.209
generic LLM-based patching	0.033	0.904	0.531	0.170
manual network filtering	0.032	0.893	0.537	0.188
generic ML-based network filtering	0.144	0.630	0.819	0.238
generic LLM-based network filtering	0	1	0.925	0
REFN	0.003	0.071	0.992	0.945
REFN's improvement	-	$\geq 88.7\%$	$\geq 21.1\%$	$\geq 225.9\%$

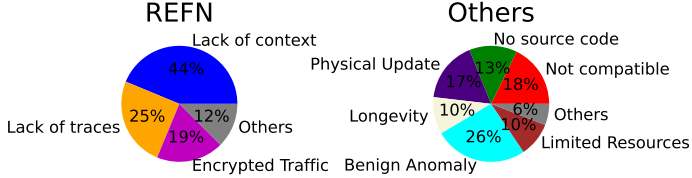


Figure 9: Pie graph analysis of invalid fixes.

ML-based patching approaches including GraphSPD [63], RNNPatch [64], PAVUDI [45] and SPI [66]; 4) generic LLM-based patching: we evaluated the patch generation capability of ChatGPT-4o [11], DeepSeek-R1 [35] and Gemma3-12B [36] (2025-July version); 5) manual network filtering: we conducted an IRB-approved study with 10 security admins creating Snort 2.9.8.0 [26] rules; 6) generic ML-based network filtering: we evaluated ML-based network filtering approaches including Kitsune [57] and ODDS [48]; 7) generic LLM-based network filtering: we evaluated the network filtering rule generation capability of ChatGPT-4o [11], DeepSeek-R1 [35] and Gemma3-12B [36] (2025-July version). More details about the IRB-approved study are presented in Section XI. To have fair comparison, for the seven categories of alternative approaches, we synthesized the best results across methods to represent the category's performance.

B. Effectiveness

Overall effectiveness: We evaluate the effectiveness of REFN and alternative approaches over 22 families of 1-day/n-day exploits and benign samples from 65 types of devices (detailed in REFN's dataset Section VII) on four metrics: 1) *FPR* (False Positive Rate), defined as $\frac{FP}{FP+TN}$; 2) *FNR* (False Negative Rate), defined as $\frac{FN}{FN+TP}$; 3) *Accuracy*, defined as $\frac{TP+TN}{P+N}$; 4) *F1-Score*, defined as $\frac{2*Precision*Recall}{Precision+Recall}$. We choose these four metrics because the 1-day/n-day exploit scenario is highly biased (benign samples > malicious samples) [41]. Therefore, relying solely on Accuracy is dangerously misleading, as a model can achieve high scores by always predicting benign. FPR reveals how often legitimate activities are wrongly flagged (causing operational disruption), while FNR measures the failure to detect actual attacks (leading to undetected breaches). The F1-score provides a crucial balanced metric over both FPs and FNs, offering a more complete picture of model effectiveness.

Table II presents the performance of REFN and alternative approaches using FPR, FNR, Accuracy, and F1-Score. While

Table III: Time-To-Patch (TTP) for REFN.

Time-To-Patch	Min	Max	P90	Mean
manual patching	3d 19.34h	7d 20.85h	5d 17.58h	5d 9.62h
patch software	3d 5.92h	7d 3.47h	5d 11.19h	4d 22.29h
manual network filtering	1d 21.23h	5d 4.79h	4d 18.76h	3d 6.61h
REFN	2.42h	5.29h	4.57h	3.65h
REFN improvement	94.7%	95.8%	96.0%	95.4%

the generic LLM-based network filtering approach achieves the lowest FPR (0) and second-highest Accuracy (0.819), its critical flaw is revealed by its FNR of 1.0 – it misclassifies all malicious samples as benign. Our analysis attributes this failure to the severe LLM hallucination during rule generation, producing seemingly correct but ultimately flawed filtering rules (exemplified in Figure 4). REFN effectively addresses this limitation; leveraging RL training with negative rewards for detected false positives, it achieves the second-lowest FPR (0.003). Crucially, REFN achieves the lowest FNR (0.071), representing an 88.7% reduction compared to the next best (generic ML-based at 0.627). Although the generic ML-based network filtering approach detects some 1-day/n-day exploits, its reliance on network traffic anomalies results in the highest FPR (0.144), causing significant disruption to benign traffic. For F1-Score, manual patching achieves the second-highest (0.290), but its manual nature hinders scalability. Overall, REFN demonstrates superior effectiveness, delivering $\geq 21.1\%$ higher Accuracy and $\geq 225.9\%$ higher F1-Score than the best alternative approaches.

Root Cause Analysis: We investigated the reasons behind ineffective fixes for both REFN and alternative approaches, as depicted in Figure 9. For alternative approaches, the key factors are: 1) benign anomaly (26%), key factor for high FPs in ML-based filtering; 2) not compatible (18%), key factor for FNs in patching; 3) need physical update such as serial cable (17%); 4) no source code available (13%); 5) limited resources (10%); 6) vendor's longevity issues (10%). For REFN, the key factors are: 1) lack of contexts (44%); 2) lack of traces (25%); 3) encrypted traffic (19%). Different from other approaches, REFN is not impacted by the compatibility issue. Notably, the inference mechanism can alleviate the impact of encrypted traffic (only 19%). The lack of contexts issue in REFN is caused by the accessibility of the training data including vulnerability descriptions and traces, and can be alleviated by crowd sourcing in the future.

C. Efficiency

Evaluating the efficiency of fixing 1-day/n-day vulnerabilities requires addressing two critical questions:

- *Can vulnerability fixing outpace exploitation?* Specifically, given the **Time-To-Patch (TTP)** – defined as the duration from vulnerability exposure to the deployment of a protective fix – does the system achieve a TTP consistently less than one day? This threshold represents the minimum exploitation window defined for 1-day/n-day vulnerabilities.
- *Does the fix installation minimize operational disruption?* Specifically, given the **Installation-Delay (iDelay)** – defined as the operational downtime imposed on a normal device during the fix deployment process – the iDelay should be low and ensuring minimal disruption to benign devices.

We evaluate the efficiency of REFN in terms of Time-To-Patch (TTP) and Installation-Delay (iDelay), and compare it with manual patching, patch management software, manual network filtering (in the IRB study, 10 admins manually create patches or network filters). We are unable to measure the TTP and iDelay for common ML and LLM patching/filtering approaches, because these approaches only provide trained models and their training time and deployment time are not disclosed (e.g., DeepSeek-R1 or the auto-encoder training in Kitsune [57]). However, the TTP for common ML and LLM patching/filtering approaches is estimated to be *a few days to several week* based on online reports [37], exceeding the critical 1-day threshold for outpacing the 1-day/n-day exploit.

Time-To-Patch (TTP): Table III presents the Min, Max, P90 (90th percentile), and mean Time-To-Patch (TTP) metrics for REFN. As shown in Table III, the Mean-Time-To-Patch (MTTP) for manual patching and patch management software is at least 4 days 22 hours. This high duration stems from the inherent complexity of host-based patching. For instance, remediating a Log4j vulnerability on an Apache server requires crafting JVM patches, testing compatibility and effectiveness, and redeploying both the JVM and Apache server – a costly procedure. Similarly, manual network filtering also incurs a significant MTTP of at least 3 days and 6 hours due to the intricate, error-prone process of crafting rules from vulnerability descriptions, parsing exploit packets, and thoroughly testing and adjusting the filters against both malicious and benign traffic. In contrast, REFN achieves a dramatically lower MTTP of just 3.65 hours. This represents a 95.4% improvement over the next fastest method (manual network filtering at 3 days 6.61 hours). Crucially, REFN’s fixing speed is significantly faster than the critical 1-day threshold associated with 1-day/n-day exploits. This efficiency is attributed to REFN’s key components: its Genetic-RAG-Based Knowledge Distillation for rapid data collection, the RL-from-NFV pipeline for efficient model training, and the Online Agentic Validator for swift validation and adjustment.

Installation-Delay (iDelay): Figure 10 presents the iDelay of manual patching, patch management software, manual network filtering, and REFN. iDelay measures the operational downtime imposed on a normal device during vulnerabil-

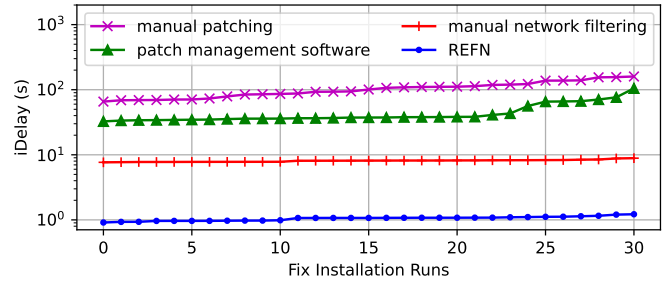


Figure 10: Installation-Delay (iDelay).

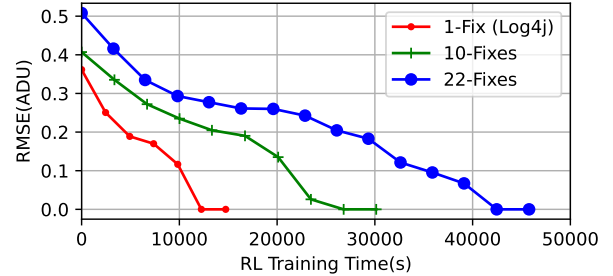


Figure 11: Batched Training Time (BTT).

ity fix deployment. To ensure a comprehensive evaluation, we conducted multiple runs (31) per approach over fixable vulnerabilities for all approaches, and sorted the results in ascending order. REFN achieves a significantly lower iDelay than alternatives, consistently requiring only around 1 second per deployment. In contrast, manual patching and patch management software incur delays of several minutes, primarily due to mandatory device/software restarts in host-based patching. Manual network filtering exhibits an iDelay of around 10 seconds, attributed to the need for administrator intervention and IPS hot restarts. This translates to REFN delivering at least a 10x reduction in iDelay compared to the next fastest feasible approach (manual network filtering) and a remarkable around 80x reduction compared to patching methods. This efficiency stems from REFN’s RL-supported and validated Virtual Network Function (VNF) deployment, leveraging agile middlebox update techniques [53] for near-instantaneous updates and minimum disruption.

D. Scalability

There are two key questions that needs to be answered in evaluating the scalability of fixing 1-day/n-day vulnerabilities:

- *How do the training time increases when the number of vulnerabilities scales up? Can the training be batched?* We define **Batched Training Time (BTT)** as REFN’s time cost for training multiple vulnerabilities at once.
- *What is the overall operation disruption for fix installation when the number of devices scales up?* We define **Accumulative Downtime (ADT)** as the total time of all devices’ normal function being disrupted by the fixing process.

Batched Training Time (BTT): As shown in We measured how REFN’s batched training time (BTT) shift from training 1-fix (Log4j) to training 10-fixes, to training 22-fixes. The x

Table IV: Accumulative Downtime (ADT).

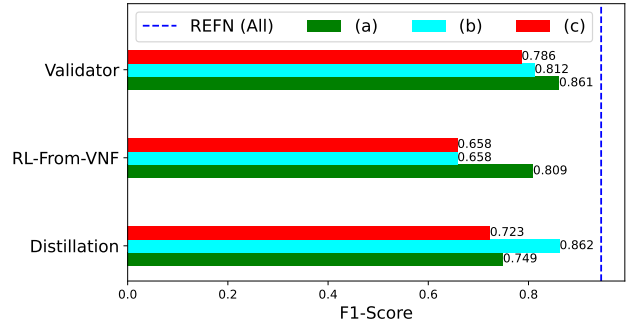
Devices	10	100	1000	10000
manual patching	1136s	12182s	-	-
patch software	689s	6608s	65521s	-
manual network filtering	82s	1038s	-	-
REFN	6s	56s	575s	5486s

axis is the overall training time at each iteration. An iteration is the process between the training start to the time the result is validated. The y axis is the RMSE (Root Mean Square Error) of the result when validated in each iteration. The RMSE is calculated based on correctly matched Application Data Units (ADUs) during the validation process. In Figure 11, the 1-Fix’s (red line, Log4j) training process took 5 iterations to complete. The total training time was 3.40 hours (12250s). The batched 10-fixes (green line line) took 8 iterations to complete. The total training time was 7.45 hours (26807s). The batched 22-fixes (green line line) took 14 iterations to complete. The total training time was 11.80 hours (42463s). From the result we can see that, REFN scales well when used for generating batches fixes for multiple vulnerabilities emerged in the same day. Comparing with 1-Fix, batched 10-fixes only take 2.2X of training time instead of 10X, batched 22-fixes only take 3.5X of training time instead of 22X. Also, REFN’s training time for all 22 families of vulnerabilities (each with one fix) is less than 0.5 day, which is much lower than the critical time threshold of 1 day for 1-day/n-day vulnerabilities.

Accumulative Downtime (ADT): We systematically increase the number of vulnerable devices (each with one vulnerability) from 1 to 10000 and measure the accumulative downtime across all devices. Table IV shows the ADT for manual patching, patch management software, manual network filtering and REFN. For manual patching and manual network filtering, their manual process can hardly scale beyond 100 devices given the 10 security admin (which is already high in manual cost) constrains in our IRB-study. patch management software can automated by scripts but its ADT is already 18.2 hours (65521s) when device number scale to 1000, and cannot scale to 10000 given resource constrains in our experiment. In comparison, REFN can reduce the ADT by at least 10X comparing with other approaches, and REFN can scale to 10000 vulnerable devices with 1.5h ADT (5486s), easily supporting a branch office with thousands of employees (suppose each employee correspond to less than 10 devices). Note that the ADT is the sum of downtime of all devices, not every device.

E. Ablation Study

As shown in Figure 12, we perform the ablation study and evaluate the benefit introduced by each of REFN’s component with the following settings: Distillation(a)-without agents; Distillation(b)-without RAG; Distillation(c)-remove all (manual text only); RL-From-VNF(a)-without VNF-reward function; RL-From-VNF(b)-without VNF-GRPO; RL-From-VNF(c)-remove all (SFT only); Validator(a)-without fuzzing & trimming; Validator(b)-without NToT; Validator(c)-remove all (no validation, random reward). Figure 12 shows the F1-Score for each setting. Without the Agentic-RAG-based Knowledge

**Figure 12: Ablation study on REFN components.**

Distillation (Distillation(c)), the F1-Score (0.723) is lower than REFN (blue line) by 23.5%. This is because the training data and knowledge is the key for the RL process. More specifically, the effectiveness impact of removing the Agent (Distillation(a), F1-Score 0.749, 20.7% reduction) is greater than removing the RAG (Distillation(b)). When replacing the RL-From-VNF Pipeline (RL-From-VNF(c)) with common SFT (Supervised Finetuning), the F1-Score (0.658) is lower than REFN by 30.4%. More specifically, the VNF-GRPO algorithm is absolutely necessary and without it (RL-From-VNF(b)), the F1-Score (0.658) is equally bad as removing the whole part. Without the Online Agentic Validator (Validator(c), random rewards), the F1-Score (0.786) is lower than REFN by 16.8%. Even though random rewards still demonstrates some effectiveness as it facilitates multiple iterations of training, the validation process as well as the fuzzing & trimming (Validator(a)) and the NToT (Validator(b)) is still important as it further improves the F1-Score. This is because the Online Agentic Validator is key to ensure the fix is correct and reduce the error-susceptibility.

IX. DISCUSSION

Relation with traditional host-based patching: REFN is not designed to replace traditional host-based patching, but rather to complement it by providing rapid edge protection. The framework generates and deploys vulnerability-fixing filters rapidly to prevent large-scale exploitation during the critical window before host-based patches can be applied. This approach is particularly valuable for protecting legacy/embedded devices where patching is prohibitively difficult or costly.

Handling encrypted traffic: Currently, REFN relies on the edge security gateway decryption (common in business/employee networks, e.g., Cisco Meraki scenarios) or context-inference mechanism to handle the exploitation via encrypted traffic. In the future, we will explore enhanced methods including enterprise proxy integration and zero-trust authentication solutions to strengthen encrypted threat prevention.

Handling LLM-based exploit tools: Theoretically, REFN can counter LLM-powered attack tools (e.g., HackerGPT [31], WormGPT [29]) through knowledge distillation that extracts exploit patterns from these adversarial systems. By analyzing outputs from tools like WormGPT, REFN’s distillation pipeline could preemptively identify and block novel attack vectors generated by malicious LLMs. Practical validation of this

capability remains future work, with planned testing in real-world attack scenarios.

X. CONCLUSION

The 1-day/n-day vulnerabilities pose severe threats to diverse networked devices at massive scales. To combat this challenge, we introduce REFN, a novel framework that provide network-driven Reinforcement Learning to train LLMs and automatically generate and deploy vulnerability-fixing filters at the edge. REFN effectively addresses large-scale exploitation across heterogeneous environments, demonstrating exceptional efficiency and scalability. Looking forward, REFN serves as an initial step toward rapidly preventing massive-scale exploitations at the edge.

XI. ETHICS CONSIDERATIONS

In this research, we conduct an IRB-approved study and invited ten security personal with vulnerability mining competition experiences to perform three tasks. The first task is a manual patch experiment, which requires the security admins to manually patch the vulnerable devices, using any available official documents and websites of the devices as the patching guide. The second task is writing common network filter rules, which requires the security admins to manually generate the vulnerability fixing rules on top of basic prevention rules in Snort 2.9.8.0. The third task is to use common LLMs to generate and deploy patches and network filter rules, which requires the security admins to manually craft the LLM prompts and perform filter deployments. Our Institutional Review Board (IRB) have censored the above evaluation and concluded that *human subjects are not evolved* (because for any data used in this study, all the sensitive information including the personal's identity have been removed) and the highest ethical standards are met. The experiment of this research is conducted in securely contained environment that satisfies the highest ethic standards.

REFERENCES

- [1] Hisilicon dvr hack. <https://github.com/tothi/pwn-hisilicon-dvr>, 2018.
- [2] Amazon eero. <https://eero.com/>, 2021.
- [3] Cisco meraki. <https://meraki.cisco.com/>, 2021.
- [4] How cisco meraki mx with advanced security can detect and block log4j exploits? <https://community.meraki.com/t5/Security-SD-WAN/Log4J-detection/m-p/135718>, 2021.
- [5] Linksys velop. <https://www.linksys.com/us/velop/>, 2021.
- [6] Netgear orbi. <https://www.netgear.com/home/wifi/mesh/>, 2021.
- [7] Threat advisory: Critical apache log4j vulnerability being exploited in the wild. <https://blog.talosintelligence.com/apache-log4j-rce-vulnerability/>, 2021.
- [8] Us warns log4j flaw puts hundreds of millions of devices at risk. <https://www.zdnet.com/article/log4j-flaw>, 2021.
- [9] National vulnerability database. <https://nvd.nist.gov/>, 2022.
- [10] Avira antivirus. <https://www.avira.com/zh-cn>, 2023.
- [11] Chatgpt. <https://openai.com/chatgpt>, 2023.
- [12] Chocolatey-the package manager for windows. <https://chocolatey.org/>, 2023.
- [13] Cisco ios. <https://www.cisco.com/c/en/us/products/ios-nx-os-software/index.html>, 2023.
- [14] Free software update for windows.download heimdal free. <https://heimdalsecurity.com/products/free-software-updater>, 2023.
- [15] Home updater: Overview and download—patch my pc. <https://patchmypc.com/home-updater>, 2023.
- [16] Langchain. <https://github.com/langchain-ai/langchain>, 2023.
- [17] Manageengine patch manager plus. <https://www.manageengine.cn/patch-management/>, 2023.
- [18] Microsoft copilot. <https://www.microsoft.com/en-us/microsoft-copilot>, 2023.
- [19] mttp. <https://feldeffect.com/blog/1-day-0-day-vulnerabilities-explained>, 2023.
- [20] Ninite-install or update multiple apps at once. <https://ninite.com/>, 2023.
- [21] Npackd. <https://npackd.org/>, 2023.
- [22] Openwrt. <https://openwrt.org/>, 2023.
- [23] Qemu. <https://www.qemu.org/>, 2023.
- [24] Raspberrypi. <https://www.raspberrypi.com/>, 2023.
- [25] Ruckzuck software package manager for windows. <https://ruckzuck.tools/>, 2023.
- [26] Snort. <https://www.snort.org/>, 2023.
- [27] Sumo documentation. <https://sumo.dlr.de/docs/index.html>, 2023.
- [28] What is the log4j vulnerability? <https://www.ibm.com/topics/log4j>, 2023.

- [29] Wormgpt: An ai tool for hackers. <https://www.popularmechanics.com/technology/security/a45533297/what-is-wormgpt/>, 2023.
- [30] Faiss. <https://faiss.ai/>, 2024.
- [31] Hackergpt. <https://github.com/Hacker-GPT/HackerGPT>, 2024.
- [32] Hypertext transfer protocol (http) specifications. <https://www.rfc-editor.org/rfc/rfc2616>, 2024.
- [33] Netresec: Publicly available pcap files. <https://www.netresec.com/?page=PcapFiles>, 2024.
- [34] Ollama. <https://ollama.com/>, 2024.
- [35] Deepseek-r1. <https://www.deepseek.com/en>, 2025.
- [36] Gemma3-12b. <https://ollama.com/library/gemma3:12b>, 2025.
- [37] What is the training duration for deepseek's r1 model? <https://zilliz.com/ai-faq/what-is-the-training-duration-for-deepseeks-r1-model>, 2025.
- [38] F. Alanazi, J. Kim, and E. Cotilla-Sánchez. Load oscillating attacks of smart grids: Vulnerability analysis. *IEEE Access*, 11:36538–36549, 2023.
- [39] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira. Flowlens: Enabling efficient flow classification for ml-based network security applications. In *NDSS*, 2021.
- [40] K. Bartos, M. Sofka, and V. Franc. Optimized invariant representation of network traffic for detecting unseen malware variants. In *USENIX security symposium*, pages 807–822, 2016.
- [41] S. Bhatt, P. K. Manadhata, and L. Zomlot. The operational role of security information and event management systems. *IEEE security & Privacy*, 12(5):35–41, 2014.
- [42] M. Du, F. Li, G. Zheng, and V. Srikanth. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1285–1298, 2017.
- [43] S. K. Fayaz, T. Yu, Y. Tobioaka, S. Chaki, and V. Sekar. BUZZ: Testing Context-Dependent policies in stateful networks. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, 2016.
- [44] C. Fu, Q. Li, M. Shen, and K. Xu. Realtime robust malicious traffic detection via frequency domain analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3431–3446, 2021.
- [45] T. Ganz, E. Imgrund, M. Härterich, and K. Rieck. Pavudi: Patch-based vulnerability discovery using machine learning. In *Proceedings of the 39th Annual Computer Security Applications Conference*, pages 704–717, 2023.
- [46] S. Garcia, A. Parmisano, and M. J. Erquiaga. Iot-23: A labeled dataset with malicious and benign iot network traffic (version 1.0.0). 2020.
- [47] W. He, M. Golla, R. Padhi, J. Ofek, M. Dürmuth, E. Fernandes, and B. Ur. Rethinking access control and authentication for the home internet of things (iot). In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 255–272, 2018.
- [48] S. T. Jan, Q. Hao, T. Hu, J. Pu, S. Oswal, G. Wang, and B. Viswanath. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *2020 IEEE symposium on security and privacy (SP)*, pages 1190–1206. IEEE, 2020.
- [49] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unversity. Contextiot: Towards providing contextual integrity to appified iot platforms. In *Proceedings of The Network and Distributed System Security Symposium*, volume 2017, 2017.
- [50] M. Jin, S. Shahriar, M. Tufano, X. Shi, S. Lu, N. Sundaresan, and A. Svyatkovskiy. Inferfix: End-to-end program repair with llms. *arXiv preprint arXiv:2303.07263*, 2023.
- [51] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.
- [52] J. Khalid and A. Akella. Correctness and performance for stateful chained network functions. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 501–516, 2019.
- [53] J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella. Paving the way for {NFV}: Simplifying middlebox modifications using {StateAlyzr}. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 239–253, 2016.
- [54] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches. In *USENIX Security Symposium*, 2021.
- [55] M. Miettinen, S. Marchal, I. Hafeez, T. Frassetto, N. Asokan, A.-R. Sadeghi, and S. Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In *ICDCS*, 2017.
- [56] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.
- [57] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. 2018.
- [58] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. Multi-context tls (mctls) enabling secure in-network functionality in tls. *ACM SIGCOMM Computer Communication Review*, 45(4):199–212, 2015.
- [59] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [60] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- [61] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [62] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier. Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 193–204, 2004.
- [63] S. Wang, X. Wang, K. Sun, S. Jajodia, H. Wang, and Q. Li. Graphspd: Graph-based security patch detection with enriched code semantics. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2409–2426. IEEE, 2023.
- [64] X. Wang, S. Wang, P. Feng, K. Sun, S. Jajodia, S. Benchaaboun, and F. Geck. Patchrnn: A deep learning-based system for security patch identification. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*, pages 595–600. IEEE, 2021.
- [65] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [66] Y. Zhou, J. K. Siow, C. Wang, S. Liu, and Y. Liu. Spi: Automated identification of security patches via commits. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(1):1–27, 2021.

A. Evaluation Testbed

[illegible]

Figure 14: OTNW ICS network topology.

Name	Version
ManageEngine Patch Manager Plus	10.1.2220.20
Chocolatey	2.2.0
Avira	1.1.92.6
Ninite	d0021
Patch My PC Home Updater	4.5.0.3
SUMo	5.17.9.541
Heimdal Free	3.6.4
Npackd	1.26.9.0
RuckZuck	1.7.3.1

Table V: Patch management software.