ИССЛЕДОВАНИЯ

Worlds: движок для агентного пентестинга

11 ФЕВРАЛЯ 2026 ГОДА

ШЕЙН КОЛДУЭЛЛ И МАКС ХАРЛИ

СОДЕРЖАНИЕ

Мы доработали модель 8B, которая прошла путь от рассылки спама с несуществующими модулями Metasploit до полного компрометации домена GOAD. И мы сделали это, используя исключительно синтетические обучающие данные. Нам не пришлось выделять более крупные траектории модели, создавать 12 000 реальных хостов, агрегировать данные клиентов или использовать цифровые двойники. Благодаря синтезу сетевой динамики и инструментальных механизмов мы смогли недорого и эффективно масштабировать высококачественные обучающие данные для любой конфигурации сети.

Поделиться

моделирования, который генерирует реалистичные траектории тестирования на проникновение в произвольные сети Active Directory, полностью на центральном процессоре, за несколько секунд для каждой траектории. Эти результаты тонкой настройки доказывают, что наши клиенты смогут добиться многого с помощью этой системы, когда она появится на нашей платформе в ближайшие недели.

В этом посте мы расскажем, почему проблему с обучающими данными для систем безопасности невозможно решить с помощью реальной инфраструктуры, как Worlds устраняет разрыв между Sim2Real и что модель, обученная на его выходных данных, может сделать с сетью Active Directory.

Малые модели в сфере безопасности

Мы твердо верим в силу небольших специализированных моделей. Они идеально подходят для сферы безопасности: не нужно загружать в сеть 800 ГБ модельных весов, можно контролировать C2, использовать готовые модели и библиотеки логического вывода (LoLMIL). Показатели моделей с 8–32 миллиардами параметров не уступают показателям передовых аналогов.

Рассмотрим пример из области разработки программного обеспечения: в октябре 2024 года модель Claude 3.5 Sonnet показала новый рекорд на SWE-Bench Verified — 49 %. Год спустя модель KAT-Dev-32B достигла показателя в 62,4 % на том же бенчмарке. Модель Qwen3-Coder-30B-A3B достигла показателя в 50 % всего с 3 миллиардами активных параметров. Сегодня открытом доступе, и небольшие модели быстро совершенствуются в области кодирования. А как насчет наступательной безопасности?

Проблема с обучающими данными

Основным препятствием для создания небольших моделей для автоматизации операций по обеспечению безопасности является доступ к высококачественным средам и данным. Большая их часть хранится в частных компаниях или непригодна для использования из-за требований законодательства. У разработчиков программного обеспечения есть GitHub… а что можно сказать об аналоге в сфере безопасности? Его не существует, а существующие варианты не оправдывают ожиданий.

Использование созданных вручную лабораторных сред: Мы построили наше исследование PentestJudge на основе GOAD и использовали его в качестве оценки для обучения с подкреплением в рамках нашей презентации Offensive AI Con. Созданные вручную лабораторные среды, такие как GOAD, отлично подходят для людей и тестирования механизмов агентов, но каждая конфигурация статична, ее сложно настроить, отслеживать и сбрасывать. Учитывая производительность лучших моделей (мы видели, как они решали эту задачу целиком менее чем за 5 минут), можно предположить, что GOAD и его варианты уже стали эталоном, и создать на их основе что-то значимое будет непросто.

Создание инфраструктуры с нуля: для настройки сетей Windows требуются создание необходимых объемов данных. Сети Windows дорого размещать в облаке, а создание сетей, которые по количеству хостов соответствуют корпоративным сетям, и их параллельное размещение для RL обходится в сотни тысяч долларов. Вот почему большинство сетей, используемых для тестирования, CTF или онлайн-занятий, содержат от пяти до двадцати хостов. Мы просим модели обучаться на редких топологиях (от трех до пяти узлов) и экстраполировать полученные данные на подавляющее большинство топологий.

Leveraging large networks: Those that are in a position to aggregate large network data are unable to use it due to distribution and usage restrictions. You would have to be a services organization: an MSSP or pentesting firm in a position to aggregate sensitive data. If you manage it, you've got data that's impossible to share, dangerous to work with, and still only represents a small fraction of the distribution of all networks that a model would want.

No matter which way you try to work with real infrastructure, you simply can't generate thousands of variations at the speed and scale required for meaningful training data. So, borrowing inspiration from Google DeepMind's work on simulated environments, we gathered up our domain expertise and asked, "what if we just built the data we wanted directly without any network at all?"

Addressing the Sim2Real Gap

The concept of using programmatically simulated computer networks to train security agents isn't new. For years there were dozens of RL papers showing different models learning to perform penetration tests. Despite that research, there were

These research papers had an agent performing *in simulation*. Instead, a *model* of the environment was created. Not a language model that takes in tokens and spits out tokens. Rather, it's a simplified representation of an existing object. Just as Sims is a model of real life or Hearts of Iron is a model of military history.

Security researchers have historically attempted to simulate network operations using Partially Observable Markov Decision Processes (POMDPs). Agents take actions with partial information about their environments, taking observations of that environment as they take an action. It's a reasonable mathematical model of real ops, yet it rarely results in agents that can be deployed in production.

The key to this problem is the *size* of the gap between simulation and reality, the "Sim2Real gap". All models are wrong, but there are two requirements for making one that's useful:

1. **Actions and observations in a simulation must match reality.** Instead of `SCAN <host>`, a command that tells you whether a host exists, and what services and ports are open on it, you have access to `nmap` which has subtler requirements around calling the tool and a wider variety of responses to interpret. An agent learning to speak the simulation's "tool language" cannot be placed in a real-world harness, unless those languages match.
2. **Relevant and accurate state dynamics of the environment must be modeled.** A large part of network operations is abusing Active Directory. If Active Directory relationships aren't modeled (very common), the distribution shift between the state dynamics of simulation and reality would be so large that the actions the agent knew how to take wouldn't be effective in the real world.

A Realistic World Model for Network Operations and Automated Penetration Testing

Worlds is a simulation engine that generates complete penetration testing trajectories. It includes the full sequence of tool calls, observations, and reasoning an agent would produce when attacking an Active Directory network, without ever provisioning a real host.

The first question we asked when designing Worlds was, "What state dynamics are the most important to represent in modern network operations?" Active Directory, of course. It's common for networks with strong vulnerability management programs to be compromised by unintended results of their Active Directory relationships. Credential attacks, delegation abuse, ACL abuse, and AD CS abuse all stem from Active Directory misconfigurations, so a high-fidelity simulation of Active Directory dynamics were the biggest "bang for our buck" from a development perspective. The second was vulnerability scanning, as nmap and nuclei both publish known ports and services, and have CVEs associated with them.

The state dynamics themselves are straightforward to model given the graph nature of Active Directory, with strong candidates like ADSynth built on top of tools like adsimulator. We broke Worlds into two conceptual layers:

Manifests

Manifests are Worlds' single source of truth, defining the bounds of hosts, services, principals, and their relationships. This defines the diagram for our network. Manifests could be created from real networks using `ldapsearch`, or a collector, but to

This enables choosing a network and then having the vulnerabilities and specific qualities of the network generated probabilistically, making it easy to sample diverse networks. On top of this, we layer in the misconfigurations we want our models to exploit—kerberoastable service accounts, AS-REP roastable users, and excessive ACL permissions. Based on these configurations we ensure that the relationships are valid, and CVEs are accurately assigned to relevant hosts and services. Finally, we ensure there is at least one valid path from a specific starting point on the network to compromise of the domain.

This represents our 'relevant state dynamics'. Everything layered on top of this is done to increase the fidelity of the simulation's ability to be a useful learning signal for LLM agents. There are some generative elements that we use in the environment that get used when an agent issues certain tool calls —enumerating fileshares, for example. Relevant files can be generated and cached based on AD information.

Tool Layer

The manifest defines the state of our environment, but trajectories require interaction. For LLM agents, we're interested in actions represented by realistic tool calls and their responses. From the agent's perspective, this is all just text. It sends text formatted in a specific way to elicit a tool-call, and receives text back.

Worlds creates an endpoint that takes in strings, representing the contents of a tool call the way they might be written from a command line. This might be an `nmap` command, an `ldapsearch`, an `smbclient`, or even host-based utilities like `whoami` or `cat`.

can be filled in or causally generated by querying the manifest instance at runtime. The arguments are parsed and the manifest is queried to determine what the "realistic" output would be, according to the template.

In addition to tool output, events are emitted based on whether this created new information for the agent. The handler function that parses these tools emits events of any change to the underlying network state caused by a tool call, as well as keeping track of what an agent "knows" having run that specific tool. This allows us to have a deterministic understanding of which paths represent real solutions to the network.

Creating Trajectories Using Synthetic Data Generation Techniques

The Worlds simulation layer produces complete trajectories of agent tool calls. An initial prompt as a goal, followed by the inputs and outputs of every tool call on the way to successfully accomplishing the goal. This can be done to generate arbitrary amounts of diverse tool call sequences for arbitrary networks, without ever leaving the CPU. An accurate network simulation, and high-fidelity tool calls and their responses resulting in a complete objective is a good start, but there are a few things we need to generate to make the trajectories training quality.

We can now proceed to augment this dataset with strong open LLMs such as Kimi K2.5 using synthetic data generation techniques.

Synthetic Reasoning

For each step, we want the result of the last tool call to motivate the next tool call. Reasoning tokens before issuing the next one. Since we have the full trajectory that led up to success or failure, as well as the manifest, we can generate a chain of thought that mirrors what we would expect to see from a real agent actively exploring a real environment. Most models now prefer using `<think>` tokens at inference time, so it's best for that further finetuning data to reflect that style. It also greatly increases the entropy of the dataset, which we found to be useful during training.

To be precise about what's synthetic here: the *trajectories* come entirely from the Worlds simulation. The *reasoning traces* are generated by a language model given the full trajectory context. We are not distilling attack behavior from a frontier model. The domain knowledge lives in the simulation; the language model provides natural language scaffolding around it.

During our experiments, we found it was crucial that the model generating the reasoning trace had access to the entire trajectory. Without encouragement to reference prior tool call results, reasoning became fixated on an individual tool call. Models trained on that sort of synthetic reasoning were able to justify any tool call regardless of what had happened on the last turn. This led to loops of recon that never built on prior information, with reasoning that was plausible but ineffective. Models that used more grounded reasoning used more of the surrounding context and did not suffer from this problem.

Failure Recovery

Failures and missteps are a normal part of operations and environment exploration, and are absent in the initial tool call trajectories. The ability important function of a robust security agent. We make sure to add examples of erroneous commands, failed exploits, and their errored responses to ensure the resulting dataset does not give up after one fat-fingered command.

These techniques resemble what Pleias has published about Synthetic Playgrounds. It's deceptively difficult to get these techniques to work at scale, and to lead to a high-quality dataset. Many hours are spent inspecting the datasets. But once the recipe is functional, these techniques combined with domain knowledge stack to great effect.

Results: From Zero to Domain Admin on GOAD

For training, Worlds produced:

- **10,023 trajectories** across a 49-host network spanning 2 subnets
- **872 unique users**, 31 AD groups, realistic OU structures
- **2,100+ unique (host, principal) starting scenarios**
- 10 host types: Domain Controllers, Certificate Authorities, SQL servers, web servers, mail servers, file servers, CI servers, VPN endpoints, and workstations
- 5 distinct attack strategies: ADCS Certificate Abuse, Kerberoasting, AS-REP Roasting, ACL/Permission Abuse, and credential pivoting

Model Evaluation Setup

All models were evaluated with a single tool allowing direct calls to the command line in a Docker container. We found this one-tool “command line wrapper” representation of the tool layer was easiest to translate into an evaluation harness. While it would be more efficient to express the results of these tool calls through compact context and episodes to make up to 100 tool calls.

Scoring used a "compromise score" based on what an agent uncovered in a network, beyond the starting credentials. Points are collected for Users when passwords or NTLM hashes are displayed in tool outputs, and for Hosts when the local administrator password or hash appear. Higher-privileged accounts are worth more than regular users; Domain Controllers are worth more than servers. We employed a context management technique where, when the context window filled up, useful information was compacted and appended to the initial prompt.

Initial Model Performance: Nemotron Orchestrator 8B

Nemotron Orchestrator 8B was chosen as the initial model for testing, because of the kinship we felt given the interesting synthetic environment techniques during its training. We evaluated it with the credentials `hodor:hodor` (no points were awarded for this) with the goal of achieving Domain Admin.

Nemotron scored 0, making no progress solving GOAD. Without grounding in Active Directory pentesting techniques, the model had a very 2017 idea of pentesting: run `nmap`, see an SMB server, load up Metasploit and see if it's vulnerable to any modules, and call it a day. When that failed, it would load a password list and spray the domain before giving up. Even with a valid low-privileged credential, Nemotron didn't know how to leverage AD relationships to gain more information about the network. The knowledge simply wasn't available in the training data.

Training Results

measured performance against GOAD. When synthetic reasoning or failure recovery increased dataset size, we removed samples to keep token counts consistent across experiments.

Tool Trajectories Only

The loss of tool-call-only data quickly cratered, converging within 1000 samples, and the model learned to predict the connection between different tools early on in training. When deployed with this LoRA, Nemotron showed no performance improvement over the base evaluation. Compromise score: 0.

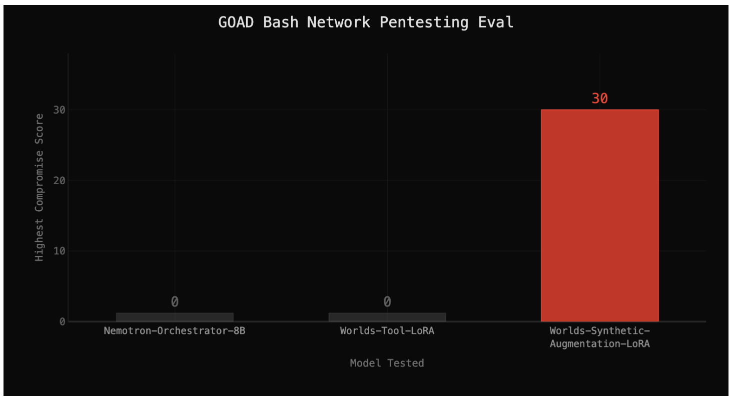
During inference, Nemotron always outputs `<think>` tokens before it makes tool calls. Without the reasoning traces in the fine-tuning data, Nemotron struggled to make use of that information, even if it was stored somewhere in the LoRA adapter.

Achieving Domain Admin Using Tool Trajectories With Synthetic Augmentation

Taking the same dataset and layering in reasoning traces and failure recovery changed the results dramatically. The loss curves for these runs converged more gradually, and continued to trickle down after several thousand samples. This is consistent with the model learning to predict both tool usage *and* reasoning.

The resulting model regularly leveraged the `hodor:hodor` credentials to AS-REP roast a higher-privilege account hash. Cracking that hash, it enumerated SYSVOL, found higher-privilege credentials, dumped LSASS with them, and ultimately achieved Domain Admin in the `NORTH`

The jump from 0 to domain compromise on a real network from just a LoRA trained on synthetic data validates our core thesis: you don't need real infrastructure to generate training data that transfers to real networks.



How Will Synthetic Training Data Impact Security?

Worlds is the first step towards truly scalable, high-quality training data for security. The goal of this research was to determine whether synthetic trajectories generated at low cost could yield measurable gains on real-world evaluations. Our results confirm it.

The implications of Worlds extend beyond training a single model:

For model trainers and AI labs: Generate Supervised Fine-Tuning (SFT) datasets at arbitrary scale for any network topology, attack path, or scenario without provisioning infrastructure or handling sensitive data. Data ships in standard chat-template format, ready for fine-tuning.

For red teams and offensive security: Train task-specific small models that run on-prem without exfiltrating sensitive context to an API. Control the model, the environment, and the data.

same data can power detection rule validation, purple team exercises, and training data for defensive models that need to recognize varied attack behavior patterns.

For security product teams: Build domain-specific security capabilities into your products without needing access to real customer networks. Let Worlds generate the distribution your model needs to see.

With a working recipe in hand, we're already running the next round of experiments: larger models, more diverse network configurations, and longer training runs. At the release of Worlds in the next few weeks, customers will be able to generate SFT datasets for their own models, on demand: configurable network sizes, topologies, attack strategies, and output formats.

We took an 8B model from zero to Domain Admin using synthetic data alone. What could you do with large-scale training data tailored to your network? Request early access to Worlds.

LINKEDIN

X

BLUESKY

GITHUB