Cyberwheel Research: Foundational Step-by-Step Explanation

Complete Beginner's Guide to Understanding the Research

August 12, 2025

Contents

1 What Are We Actually Trying to Do? (The Big Picture)

Foundation Concept

The Core Problem: How can we train computer programs (AI agents) to automatically defend computer networks against cyber attacks, where both the attackers and defenders are learning and adapting to each other?
Think of this like teaching two chess players simultaneously - one trying to attack and win, the other trying to defend and prevent the attack - except instead of chess, it's cybersecurity.

Intuitive Explanation

Real-World Analogy: Imagine you're training two security guards:

- **Red Team (Attacker)**: Tries to break into a building using various methods
- **Blue Team (Defender)**: Tries to detect and stop the break-ins using cameras, alarms, and decoy rooms

Both teams get better over time by learning from their successes and failures. The defender learns where to place cameras and decoy rooms to catch attackers, while the attacker learns to avoid detection and find new ways in.

2 What is Reinforcement Learning? (Starting from Zero)

Foundation Concept

Reinforcement Learning (RL) is a way to train computer programs by having them:

1. Try different actions in an environment
2. Get rewards (positive) or penalties (negative) based on their actions
3. Learn which actions lead to better rewards over time

This is exactly how humans and animals learn - through trial and error with feedback.

Concrete Example

Simple Example: Teaching a computer to play Pac-Man

- **Environment**: The Pac-Man game maze
- **Actions**: Move up, down, left, right
- **Rewards**: +10 for eating a dot, +50 for eating a ghost, -100 for getting caught
- **Learning**: Over many games, the computer learns strategies that maximize its total score

2.1 Key RL Concepts We Need to Understand

Mathematical Details

The Mathematical Framework:

- **State (S)**: What the agent can observe about the environment
- **Action (A)**: What the agent can do
- **Reward (R)**: Feedback the agent receives
- **Policy (π)**: The agent's strategy (which action to take in each state)
- **Value Function (V)**: How good it is to be in a particular state

The Goal: Find a policy π that maximizes the expected total reward:

$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} R_t\right]$$

Where:

- T = total time steps
- γ = discount factor (0.95 in our research) - values future rewards less than immediate ones
- R_t = reward at time t

3 What Makes This "Adversarial"?

Foundation Concept

Adversarial Learning means we have two (or more) agents learning simultaneously, where one agent's success often means the other's failure. This is different from single-agent RL where there's only one learner.

Intuitive Explanation

Think of it like: Two players learning to play chess against each other

- Player 1 gets better at attacking
- Player 2 gets better at defending
- As Player 1 improves, Player 2 must adapt to the new strategies
- As Player 2 improves, Player 1 must find new ways to attack
- This creates an "arms race" of improvement

3.1 Why is Adversarial Learning Hard?

Mathematical Details

The Mathematical Challenge:
In single-agent RL, we optimize:

$$\max_{\pi} J(\pi)$$

In adversarial RL, we have a two-player game:

$$\max_{\pi^{(b)}} \min_{\pi^{(r)}} J^{(b)}(\pi^{(b)}, \pi^{(r)})$$

Where:

- $\pi^{(b)}$ = blue (defender) policy
- $\pi^{(r)}$ = red (attacker) policy
- Blue wants to maximize their reward
- Red wants to minimize blue's reward (maximize their own)

This is much harder because:

- The environment is no longer stationary (it changes as the opponent learns)
- Training can become unstable if one agent learns much faster than the other
- Finding equilibrium solutions is computationally challenging

4 The Cybersecurity Environment (Step by Step)

4.1 What is the "Environment"?

Foundation Concept

The Environment is a simulated computer network with:

- Multiple computers (hosts) - from 15 to 10,000 in our experiments
- Network connections between computers
- Some computers have vulnerabilities (security weaknesses)
- Some computers can be "decoys" (fake computers designed to trap attackers)

Concrete Example

Concrete Network Example:

- 15 computers in a small office network
- 3 of them are servers (valuable targets)
- 2 of them are decoy computers (look real but are traps)
- 10 of them are regular workstations
- Computers are connected in subnets (like floors in a building)

4.2 What Can the Red Agent (Attacker) Do?

Intuitive Explanation

Red Agent Actions mirror real-world cyber attacks:

1. **Discovery**: Scan the network to find computers and services
2. **Reconnaissance**: Probe computers to find vulnerabilities
3. **Privilege Escalation**: Exploit vulnerabilities to gain access
4. **Impact**: Steal data or disrupt services on compromised computers

Mathematical Details

Red Agent State Space: $S^{(r)} \in \mathbb{R}^{d_r}$ where $d_r = 2|H| + |S| + 299$
This means the red agent observes:

- Current position (which computer they've compromised)
- Knowledge of network topology (what they've discovered)
- Current attack phase (discovery, reconnaissance, etc.)
- Available attack techniques (295 from MITRE ATT&CK framework)

Action Space: $|\mathcal{A}^{(r)}| = 12 \times |H|$ actions

- For each host H, there are 12 possible attack actions
- Total actions scale with network size

4.3 What Can the Blue Agent (Defender) Do?

Intuitive Explanation

Blue Agent Actions mirror real-world cyber defense:

1. **Deploy Decoys**: Place fake computers to mislead attackers
2. **Remove Decoys**: Take down decoys that aren't working
3. **Isolate Hosts**: Disconnect compromised computers from the network
4. **Do Nothing**: Sometimes the best action is to wait and observe

Mathematical Details

Blue Agent State Space: $S^{(b)} \in \mathbb{R}^{d_b}$ where $d_b = 3|H| + 2$
The blue agent observes:

- Current alerts (immediate warnings about suspicious activity)
- Alert history (memory of past attacks)
- Decoy deployments (where fake computers are placed)
- Metadata (constant values and counts)

Action Space: $|\mathcal{A}^{(b)}| = 2|S||\mathcal{D}| + |H| + 1$

- Deploy or remove decoys on subnets S with decoy types $\mathcal{D}$
- Isolate any of the $|H|$ hosts
- Plus one "do nothing" action

5 The Reward System (How Agents Learn What's Good/Bad)

Foundation Concept

Rewards tell the agents whether their actions were good or bad. This is how they learn over time.

5.1 Red Agent Rewards

Intuitive Explanation

Red agent gets rewards for:

- **Successful attacks** on real computers (+points)
- **Advancing through attack phases** (+points)
- **Getting detected** (-points) - penalty for being caught

Mathematical Details

Red Reward Formula:

$$R_{t,h}^{(r)} = \sum_i \alpha_i \cdot \mathbf{1}[\text{technique}_i \text{ successful}] + \beta \cdot |\text{assets compromised}| - \lambda \cdot \mathbf{1}[\text{detected}]$$

Where:

- $\alpha_i > 0$ = reward for successful attack technique
- $\beta > 0$ = bonus for compromising valuable assets
- $\lambda > 0$ = penalty for getting caught
- $\mathbf{1}[\cdot]$ = indicator function (1 if true, 0 if false)

5.2 Blue Agent Rewards

Intuitive Explanation

Blue agent gets rewards for:

- **Tricking attackers into decoys** (+BIG points)
- **Protecting real computers** (+points)
- **Using too many resources** (-points) - cost of maintaining decoys

Mathematical Details

Blue Reward Formula:

$$R_{t,h}^{(b)} = R_{\text{deception}} + R_{\text{protection}} + R_{\text{cost}}$$

Where:

$$R_{\text{deception}} = \begin{cases} 10 \cdot |R_{\text{red}}^{\text{base}}| & \text{if red attacks decoy} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$R_{\text{protection}} = \begin{cases} -|R_{\text{red}}^{\text{base}}| & \text{if red attacks real host} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$R_{\text{cost}} = -c_{\text{deploy}} \cdot N_{\text{new decoys}} - c_{\text{maintain}} \cdot \sum_i \text{decoy}_i \tag{3}$$

Key Insight: The "10×" multiplier for deception means tricking an attacker into a decoy gives 10 times more reward than preventing an attack on a real computer. This strongly encourages the use of deception.

6 The PPO Algorithm (How the Agents Actually Learn)

Foundation Concept

PPO (Proximal Policy Optimization) is the specific machine learning algorithm we use to train our agents. It's a state-of-the-art method for reinforcement learning.

Intuitive Explanation

Think of PPO like a cautious student:

- The student tries new strategies, but not too different from what worked before
- If a new strategy works well, they adjust their approach slightly in that direction
- If a new strategy fails, they adjust away from it
- They never make huge changes all at once (this prevents "forgetting" good strategies)

6.1 The PPO Mathematical Formula

Mathematical Details

PPO Objective Function:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$

Let's break this down piece by piece:
What is $r_t(\theta)$?

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

This is the ratio of:

- New policy probability of taking action a_t in state s_t
- Old policy probability of taking the same action

What is $\hat{A}_t$? This is the "advantage" - how much better than average this action was:

- If $\hat{A}_t > 0$: This action was better than expected
- If $\hat{A}_t < 0$: This action was worse than expected

What does "clip" do? It prevents the ratio $r_t(\theta)$ from getting too big or too small:

- If $r_t(\theta) < 1-\epsilon$: Set it to $1-\epsilon$ (typically 0.8)
- If $r_t(\theta) > 1+\epsilon$: Set it to $1+\epsilon$ (typically 1.2)
- Otherwise: Keep the original value

Why clip? This prevents the policy from changing too drastically in one update, which could destabilize learning.

7 SULI: Our Novel Training Method

Foundation Concept

SULI (Self-play with Uniform Learning Initialization) is our contribution to solving training instability in adversarial RL.

Intuitive Explanation

The Problem with Normal Adversarial Training:

- Sometimes one agent learns much faster than the other
- The fast learner dominates and the slow learner stops improving
- Training becomes unstable or gets stuck in poor solutions

SULI Solution:

- Start both agents with the same "uniform" strategy (all actions equally likely)
- Let them learn together gradually
- Regularly reset if one gets too dominant
- This creates more balanced, stable learning

Mathematical Details

SULI Mathematical Framework:
Uniform Initialization:

$$\pi_0^{(b)}(a|s) = \pi_0^{(r)}(a|s) = \frac{1}{|\mathcal{A}|} \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

Both agents start with equal probability for all actions.
Co-evolution Update:

$$\theta_{k+1}^{(b)} = \theta_k^{(b)} + \alpha \nabla_\theta L^{\text{PPO}}(\theta_k^{(b)}, \pi_k^{(r)}) \tag{4}$$
$$\theta_{k+1}^{(r)} = \theta_k^{(r)} + \alpha \nabla_\theta L^{(r)}(\theta_k^{(r)}, \pi_k^{(b)}) \tag{5}$$

Both agents update simultaneously based on their interaction.
Balance Constraint:

$$|J^{(b)}(\pi_k^{(b)}, \pi_k^{(r)}) - J^{(r)}(\pi_k^{(b)}, \pi_k^{(r)})| \leq \beta$$

If the performance difference gets too large, we intervene to rebalance.

8 The Seven-Phase Experimental Methodology

Foundation Concept

Our research follows a systematic 7-phase approach to thoroughly validate our methods, from basic functionality to large-scale deployment.

8.1 Phase 1: System Validation

Intuitive Explanation

Goal: Make sure everything works correctly

- Test with small 15-computer network
- Verify all software components function
- Ensure logging and monitoring work
- Like testing a car before a long road trip

8.2 Phase 2: Blue Agent Training

Intuitive Explanation

Goal: Train 8 different defensive strategies

- Small: Basic defense with limited resources
- Medium: Balanced detection and deception
- HighDecoy: Maximum use of decoy computers
- PerfectDetection: Theoretical best-case scenario
- (Plus 4 more specialized variants)

8.3 Phase 3: Red Agent Training

Intuitive Explanation

Goal: Train 5 different attack strategies

- RL: Learning-based adaptive attacker
- ART: Adversarial robustness testing
- Campaign: Persistent, stealthy attacks
- (Plus 2 more attack variants)

8.4 Phase 4: Cross-Evaluation Matrix

Intuitive Explanation

Goal: Test all combinations of attackers vs defenders

- 8 defenders $\times$ 5 attackers = 40 combinations
- Like a tournament where every team plays every other team
- Identifies which defensive strategies work best against which attacks

8.5 Phase 5: SULI Co-Evolution

Intuitive Explanation

Goal: Test our novel SULI training method

- Train both agents simultaneously using SULI
- Compare against traditional training methods
- Demonstrate improved stability and performance

8.6 Phase 6: Scalability Testing

Intuitive Explanation

Goal: Prove the method works on large networks

- Test on 1,000, 5,000, and 10,000 computer networks
- Measure computational requirements and performance
- Ensure real-world applicability

8.7 Phase 7: Statistical Analysis

Intuitive Explanation

Goal: Ensure results are scientifically valid

- Run multiple experiments with different random seeds
- Calculate confidence intervals and significance tests
- Prepare results for academic publication

9 Key Evaluation Metrics (How We Measure Success)

Foundation Concept

We need concrete ways to measure whether our methods are working. Here are the main metrics we use:

9.1 Deception Effectiveness

Mathematical Details

$$\text{Deception Rate} = \frac{\text{Number of attacks on decoys}}{\text{Total number of attacks}}$$

What this means: What percentage of attacker actions are wasted on fake computers?

- 0.0 = Attacker never falls for decoys (bad for defense)
- 1.0 = Attacker only attacks decoys (perfect defense)

9.2 Asset Protection Rate

Mathematical Details

$$\text{Protection Rate} = \frac{\text{Number of uncompromised real computers}}{\text{Total number of real computers}}$$

What this means: What percentage of real computers remain safe?

- 0.0 = All real computers compromised (complete failure)
- 1.0 = All real computers protected (perfect success)

9.3 Mean Time to Compromise (MTTC)

Mathematical Details

$$\text{MTTC} = \mathbb{E}[\text{Time until first successful attack on critical asset}]$$

What this means: On average, how long does it take an attacker to successfully breach something important?

- Lower values = Defense fails quickly
- Higher values = Defense delays attacks successfully

10 Research Contributions and Impact

Foundation Concept

What We Discovered:

1. SULI training reduces training failures by 90%
2. Deception-based defense strategies outperform detection-only approaches
3. The framework scales successfully to enterprise-size networks (10,000+ computers)
4. Systematic evaluation reveals optimal defensive strategies for different threat scenarios

Intuitive Explanation

Why This Matters:

- **For Cybersecurity**: Provides concrete guidance on when and how to use deception in network defense
- **For AI Research**: Demonstrates how to train stable adversarial agents in complex environments
- **For Practice**: Offers scalable methods that could be deployed in real enterprise networks
- **For Future Work**: Establishes benchmark methods and metrics for evaluating cybersecurity AI

11 Conclusion: The Complete Picture

Foundation Concept

This research combines several cutting-edge areas:

- **Reinforcement Learning**: Training agents through trial and error
- **Adversarial Training**: Training competing agents simultaneously
- **Cybersecurity**: Applying AI to real-world security problems
- **Large-Scale Experimentation**: Rigorous scientific validation

The result is a comprehensive framework for automatically training cybersecurity defense systems that can adapt to new and evolving threats.