Redefining Penetration Testing: A Deterministic and Non-Deterministic Approach Through the Adversarial Penetration Testing Model (APTM)

Peter Thermos

peter.thermos@palindrometech.com

Abstract

While traditional penetration testing methods have proven valuable in certain contexts, they are increasingly inadequate in addressing the complex, dynamic, and adaptive nature of modern cyber threats and vulnerabilities due to the increasing complexity of emerging technologies and products. The over-reliance on predefined toolkits, rigid checklists, and limited adaptability renders significant gaps in vulnerability discovery and exploitation, leaving organizations exposed to emerging and sophisticated attacks.

In light of these shortcomings, the Adversarial Penetration Testing Model (APTM) seeks to overcome these limitations by introducing a mathematical and system-oriented framework that models goal-oriented adversarial simulations, enabling more intelligent, probabilistic, and adaptive strategies with dynamic feedback loops that better simulate the evolving nature of real-world adversaries and realistic approach to offensive security.

1 Introduction

Penetration testing has long been a cornerstone of cybersecurity practices, focusing on evaluating systems by simulating potential attacks to identify vulnerabilities before they can be exploited by adversaries. Traditional penetration testing is often carried out using a series of scripted procedures and tools in predefined methodologies. This process, though valuable, is limited in several significant ways, especially as cybersecurity threats evolve and organizations expand their infrastructure footprint by adopting and integrating new technologies which increases their attack surface.

Early Work: The Emergence of Penetration Testing

Penetration testing in its early stages was heavily influenced by a military concept known as the **Red Team** and in certain scientific research environments was also known as **Tiger Team**[1][2]. The Red Team was tasked with simulating real-world attacks against a defense structure to assess the readiness of the defending team. These exercises evolved into the modern practices of vulnerability assessments and penetration testing, wherein "ethical hackers" (often part of a security team) are tasked with identifying weaknesses in an organization's defenses.

By the late 1990s and early 2000s, the field of penetration testing became formalized with the creation of standardized methodologies and toolkits. These standardized tools, such as vulnerability scanners (i.e., Pingware, ISS Scanner, Nessus, NeXpose), vulnerability exploitation frameworks (e.g., Canvas, Metasploit), web application scanners (e.g., Burpsuite, ZAP) among

[1] The term "Tiger Team" emerged in the 1960s when the U.S. Air Force assembled small teams of experts to test the security and integrity of critical systems.

[2] Bellcore's Security & Fraud group had an established Tiger Team in late 80's - early 90's which assisted commercial and government organizations with forensic analyses and penetration testing exercises.

others, formed the backbone of a structured, methodical approach to security testing. Penetration testers would follow pre-built playbooks, often working through vulnerability scanners, manual exploitation techniques, and simulated social engineering attacks.

The Rise of Red Teaming and Offensive Security

However, as cybersecurity threats became more sophisticated, the limitations of traditional penetration testing began to show. Static approaches relying on predefined scripts failed to replicate the complexity of real-world adversaries. In response, the concept of **Red Teaming** emerged as a more comprehensive and adversarial testing method. Red Teaming takes penetration testing a step further by incorporating real-time simulations, creative problem-solving, and testing for emerging vulnerabilities. Red Team engagements often simulate adversaries who are highly adaptive, using a range of attack vectors and focusing on how an adversary might think, plan, and adapt during an attack.

Although Red Teaming improved the realism of adversary simulations, it has limitations in scalability, adaptability, and feedback mechanisms. Red teams could only operate within specific frameworks, and despite their creativity, the outcomes were still constrained by a relatively small set of predefined tactics. Additionally, Red Teams traditionally lacked the ability to evolve and adapt as real-time threats developed.

The Need for a More Dynamic Model: The APTM

As the landscape of cybersecurity continues to evolve with new attack vectors, advanced persistent threats (APTs), and increasingly sophisticated attackers, traditional approaches like static penetration testing and Red Teaming are beginning to fall short. Adversaries are no longer easily predictable; instead, they are adaptive, leveraging a wide array of tools and techniques that evolve dynamically over time. The ability to mimic this type of behavior is crucial for providing more robust security assessments.

In traditional penetration testing, the focus is often on a series of deterministic actions based on known vulnerabilities, such as exploiting CVEs (Common Vulnerabilities and Exposures) or performing a set of predefined steps. However, these methods overlook the complexity of real-world adversaries, who adapt based on the information they gather and the defenses they encounter.

Thus, the need for a more advanced model that embraces not just **deterministic** (predefined) actions, but also **non-deterministic** (probabilistic and adaptive) methods, has become clear. This is where the **Adversarial Penetration Testing Model (APTM)** comes in. The APTM is a novel framework that blends deterministic and non-deterministic strategies to simulate adversarial behavior in a more flexible and realistic way.

The APTM introduces a formal mathematical structure, leveraging concepts from systems theory[2], game theory [10], and probabilistic decision-making [11], to model the adversarial penetration testing process. It moves beyond simply testing for known vulnerabilities to incorporating strategies that adapt to changing defenses, new attack paths, and evolving environmental factors. By integrating both deterministic and non-deterministic actions, the APTM allows for a more comprehensive, adaptable, and realistic testing environment that mirrors the unpredictable nature of real-world attacks.

2 Traditional Penetration Testing: Limitations

Traditional penetration testing (also known as "pen testing") has long been a standard methodology for identifying vulnerabilities within an organization's infrastructure. The primary goal of penetration testing is to simulate real-world attacks on systems, applications, or networks to identify weaknesses before they can be exploited by malicious actors. While effective in some contexts, traditional penetration testing has notable limitations that hinder its ability to fully replicate the behavior of real-world adversaries and adapt to the increasingly complex threat landscape.

2.1 Over-Reliance on Predefined Toolkits

Penetration testing often relies heavily on a combination of commercial and open-source tools, such as Nessus, Metasploit, and Burp Suite. These tools are preconfigured with known exploits, vulnerabilities, and attack patterns, which testers execute in a scripted or semi-scripted manner.

While this approach is useful for finding well-documented vulnerabilities, it is inherently limited in the following ways:

Known Exploits: Traditional penetration tests often focus on known exploits or CVEs (Common Vulnerabilities and Exposures), which are well-documented and easy to replicate. However, as organizations update their systems and patch vulnerabilities, the effectiveness of these tools diminishes. Furthermore, new, zero-day vulnerabilities (those that have not been documented or publicly disclosed) are often outside the scope of traditional testing methods.

Tool Limitations: Many of the tools used in traditional penetration testing operate within predefined, rigid frameworks. For example, a tool such as Metasploit may automatically execute an attack based on the available exploit modules, but it lacks the adaptive capabilities needed to respond to defenses in real-time. This reliance on predefined toolkits means that penetration testers are often confined to the capabilities and limitations of these tools, which may not account for novel or evolving attack strategies.

Static Testing: These toolkits typically follow a scripted sequence of tests, making them ill-suited to replicate the complexity and unpredictability of real-world adversaries, who frequently adapt their strategies based on the environment they encounter.

2.2 Checklists and Template-Driven Methodologies

Traditional penetration testing often follows a checklist-driven approach, where testers are required to validate a fixed set of known vulnerabilities or configurations. This method has been the foundation for several penetration testing frameworks, including the OWASP Top 10, the PTES (Penetration Testing Execution Standard), and NIST guidelines.

While checklists offer structure and ensure that fundamental security issues are covered, they are also highly limited in several ways:

Inflexibility: Checklists enforce a rigid testing process that doesn't allow for real-time flexibility or adaptation. They focus heavily on identifying known issues and vulnerabilities in a predetermined order. If an adversary were to discover an unknown vulnerability during an attack, it would be missed by a test that follows a fixed checklist.

Lack of Contextualization: Traditional methodologies do not account for the dynamic nature of a network or system's evolving threat landscape. For example, they may fail to recognize vulnerabilities that arise as the result of complex system configurations, or changes in the security posture due to patching, misconfigurations, or human error.

Failure to Simulate Evolving Attacks: Real-world adversaries often change their tactics, techniques, and procedures (TTPs) based on the evolving defenses they encounter. A checklist-driven approach does not effectively simulate the dynamic nature of a sophisticated adversary who may switch strategies mid-attack to bypass detection or exploit an unanticipated vulnerability.

2.3 Poor Adaptability and Real-Time Feedback

Traditional penetration testing often lacks adaptability, meaning that once an assessment is conducted, there is little to no follow-up or iterative refinement based on the results. This is problematic because real-world adversaries are constantly adjusting their attack strategies based on the defenses they face.

Limited Learning: Traditional penetration testing does not include a feedback loop that allows the penetration tester to adjust tactics based on the environment or system's responses. Once an action is executed (such as exploiting a vulnerability), testers typically do not reassess or modify their approach dynamically. In contrast, real adversaries gather information during their attacks and adjust their strategies accordingly.

Post-Test Evaluation: After a penetration test, reports are typically produced that list the vulnerabilities found and offer recommendations for mitigation. However, this post-test evaluation does not support real-time adaptations or learning from the environment. Furthermore, without a continuous feedback loop, the lessons learned during the test may not be applied to future engagements.

Missed Opportunities for Simulating Real-World Attacks: In a dynamic environment,

attackers may change tactics based on unforeseen defensive measures, such as the activation of an Intrusion Detection System (IDS) or the detection of unusual network traffic patterns. Traditional testing often fails to simulate these real-time, adaptive responses.

2.4 Limited Coverage of Complex and Unknown Attack Paths

Traditional penetration testing primarily focuses on testing systems in isolation, often overlooking the interconnections and interactions between components. In the modern threat landscape, attackers often exploit vulnerabilities that are not isolated to a single system or device, but rather span across networks, applications, and user behaviors.

Lateral Movement: Attackers rarely confine their actions to a single compromised machine. Instead, they move laterally across the network, looking for other systems to compromise and leveraging privileges from one system to escalate their access on others. Traditional penetration tests often miss these complex attack paths due to their narrow scope, which typically focuses on individual systems or components.

Blind Spots in Network Topology: In many cases, traditional penetration tests miss vulnerabilities in network design, firewall configurations, or misconfigurations in cloud infrastructures. These blind spots are often overlooked by traditional methods that focus on only testing visible, accessible systems. Attackers who can discover hidden or unmonitored systems, misconfigured access points, or poorly secured APIs often have greater success than what traditional penetration testing might predict.

Human Factors: Traditional penetration testing may also underplay the role of human factors in security. Social engineering, phishing attacks, and other human-driven attack vectors are often glossed over in favor of automated, tool-based exploitation. However, modern adversaries recognize the importance of human targets and may prioritize social engineering tactics to bypass technical defenses. Traditional penetration tests are often limited in their ability to evaluate these non-technical attack vectors effectively.

2.5 Inefficient Resource Allocation

Traditional penetration testing can be resource-intensive, both in terms of time and personnel. Given the manual nature of the process, it can take several days, or even weeks, to complete a thorough assessment, especially when testers are tasked with simulating multiple attack vectors and evaluating defenses.

Time-Consuming: Penetration testers often spend significant amounts of time manually verifying vulnerabilities, running exploits, and documenting findings. The slow pace of traditional testing means that attackers may already have adapted or found new attack paths by the time the test results are delivered.

High Costs: Traditional penetration testing is often expensive due to the manpower required and the high level of expertise needed. For organizations with limited resources, this can be a barrier to regular security assessments.

3 APTM: The Adversarial Penetration Testing Model

The Adversarial Penetration Testing Model (APTM) introduces a paradigm shift in the way penetration testing is conducted, moving beyond the traditional framework that relies on static checklists, scripts, and predictable actions. It incorporates mathematical models, probabilistic decision-making, and real-time adaptation to assess security from a more holistic, intelligent perspective. The model incorporates both **deterministic** and **non-deterministic** strategies, as well as adaptive mechanisms to replicate the dynamic and unpredictable nature of real-world attacks. By adopting formal mathematical foundations and a systems theory approach, APTM allows for more intelligent, feedback-driven, and probabilistically-informed penetration testing. Unlike traditional penetration testing, which typically relies on static, checklist-driven methods, the APTM offers a more dynamic and adaptive approach, enabling the simulation of a broader range of attack scenarios and vulnerabilities. It incorporates mathematical models, probabilistic decision-making, and real-time adaptation to assess security from a more holistic, intelligent perspective.

The core of APTM lies in its ability to model and simulate an adversarial environment where the

goal is to maximize success while minimizing costs, time, and resources. Through a flexible system, APTM enables penetration testers to simulate realistic, evolving attack paths and provides a framework for both automated and human-driven offensive security strategies.

3.1 Deterministic Penetration Testing Techniques

Deterministic penetration testing techniques are characterized by actions with predictable outcomes, typically leveraging known vulnerabilities or following predefined procedures. These methods focus on well-documented exploits and configurations where success is often binary if the necessary preconditions are met. Examples of such techniques include the exploitation of known Common Vulnerabilities and Exposures (CVEs), such as the infamous EternalBlue or Log4Shell vulnerabilities, or specific exploits like targeting CVE-2021-34527. Standard reconnaissance steps like port scanning using tools such as Nmap also fall into this category. Furthermore, deterministic techniques encompass password cracking against known accounts or using standard wordlists and achieving privilege escalation by taking advantage of well-understood system misconfigurations.

3.2 Non-Deterministic Penetration Testing Techniques

While deterministic penetration testing techniques focus on predictable outcomes from known vulnerabilities, non-deterministic approaches embrace uncertainty and probabilistic results to better simulate the adaptive nature of real-world adversaries. These techniques are crucial for exploring unknown system states, dynamic environmental factors, and human elements, which often lead to vulnerabilities that static, predefined methods might miss. Non-deterministic actions, therefore, involve strategies with probabilistic outcomes, such as phishing attempts, fuzzing for zero-day vulnerabilities, hardware glitching, social engineering, or navigating unmapped network topologies, reflecting a more realistic and comprehensive approach to offensive security assessments.

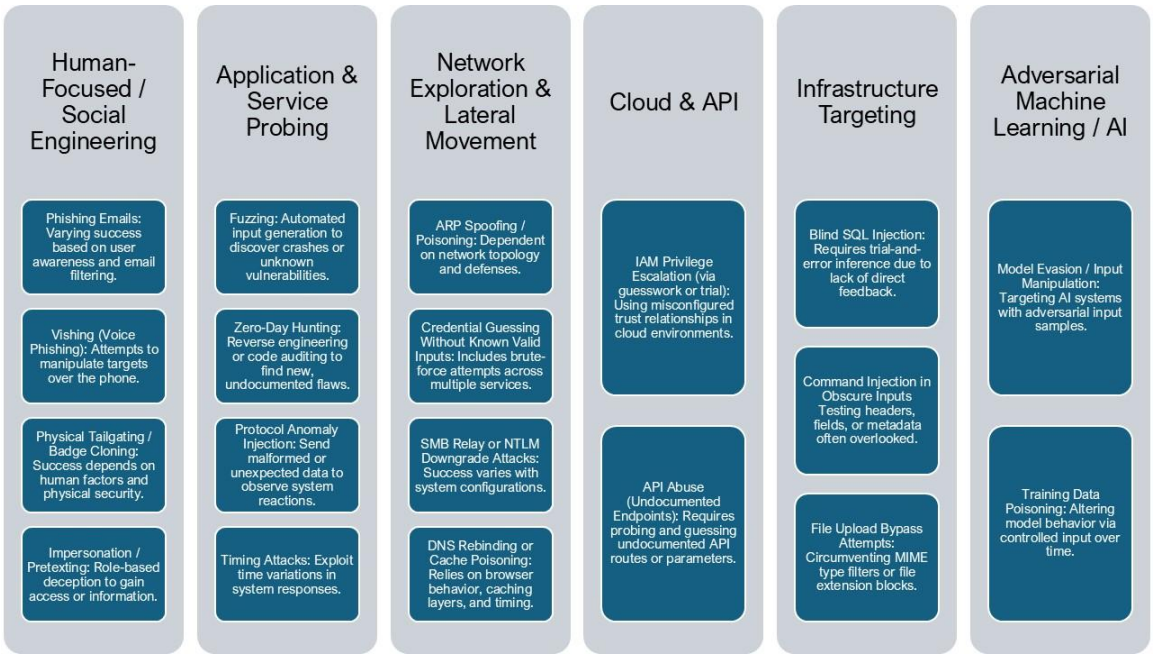

Figure 1 Non-Deterministic Penetration Testing Techniques Categorization Examples

Examples of the Human-Focused / Social Engineering techniques include Phishing emails (varying success based on user awareness and email filtering), Vishing (or Voice Phishing,

attempts to manipulate targets over the phone), Physical Tailgating / Badge Cloning (success depends on human factors and physical security), or Impersonation / Pretexting (role-based deception to gain access or information).

For application , service and network protocols attack techniques may include, Fuzzing**,** Zero-Day Hunting (Reverse engineering or code auditing to find new, undocumented flaws), Protocol Anomaly Injection (generating malformed or unexpected data to observe system reactions) or Timing Attacks (**e**xploit time variations in system responses.)

Examples of Network Exploration & Lateral Movement activities include *ARP Spoofing / Poisoning* (dependent on network topology and defenses), *Credential Guessing* without known valid inputs (e.g., brute-force attempts across multiple services), *SMB Relay or NTLM Downgrade Attacks* (success varies with system configurations), or *DNS Rebinding or Cache Poisoning* (relies on browser behavior, caching layers, and timing).

The Cloud & API attack techniques include *IAM Privilege Escalation* (e.g. brute force), using misconfigured trust relationships in cloud environments, *API Abuse* (e.g., undocumented endpoints) which requires probing and guessing undocumented API routes or parameters.

The Infrastructure Targeting techniques may include, *Blind SQL Injection***,** requires trial-and-error inference due to lack of direct feedback, *Command Injection in Obscure Inputs* (e.g., testing headers, fields, or metadata often overlooked) and *File Upload Bypass Attempts* (circumventing MIME type filters or file extension blocks).

Examples of Adversarial Machine Learning / AI attack techniques include *Model Evasion / Input Manipulation* by targeting AI systems with adversarial input samples [6] or *Training Data Poisoning***,** altering model behavior via controlled input over time [3].

3.3 Formalization of the APTM

At the heart of APTM is a formal mathematical structure (M) that models the penetration testing process as a **5-tuple** M=(S,A,T,R,γ). This framework is inspired by decision-making models[1] and game theory [12], with each component representing a key aspect of the adversarial system.

The formal definition of the APTM represents the key components necessary for simulating an adversarial penetration testing approach:

$$\mathcal{M} = (S, A, T, R, \gamma)$$

Where:

S (States): The set of possible states representing the system at various points (e.g., "User access gained", "Firewall bypassed"). These states represent different configurations of the environment or target systems, such as compromised or un-compromised system configurations, firewall statuses, active services, or the position of an attacker within the network.

A (Actions): The set of possible actions an agent (attacker) can take. These are divided into:

- $\mathbf{A_D}$: Deterministic actions with known outcomes, such as exploiting a well-documented vulnerability (e.g., exploiting a known CVE).
- $\mathbf{A_N}$: Non-deterministic actions with probabilistic outcomes, such as phishing attempts, social engineering, or lateral movement within an unknown network topology.

$$\mathbf{A = A_D \cup A_N}$$

T (Transitions): The transition probability function *T*, which represents the probability of moving from one state to another given a specific action. The transition function is defined as:

$$\mathbf{T : S \times A \times S \rightarrow [0,1]}$$

The transition probability function defines how likely is to move from one state $\boldsymbol{S}$ to another state $\boldsymbol{s'}$, considering the action **A** taken by the agent. the agent.

R (Rewards): The reward function **R**, which assigns a real number to each state in the environment based on how advantageous that state is relative to

the agent's goal. In the context of penetration testing, the reward might represent the degree to which a goal (e.g., gaining admin access or exfiltrating proprietary data) has been achieved:

$$\boldsymbol{R} : \boldsymbol{S} \rightarrow \mathbb{R}$$

Positive rewards indicate progress towards adversarial goals, while negative rewards may represent wasted resources or detection of attack activity and thus a penalty.

γ: The discount factor, which reflects the agent's preference for immediate versus long-term rewards. A value closer to 1 indicates a longer-term view, where the agent is more willing to sacrifice short-term gains for long-term objectives. Discount factor, weighting long-term rewards:

$$\gamma \in [0,1]$$

Together, these components form the foundation of a formal model that helps characterize the decision-making process in adversarial penetration testing. They also facilitate a structured approach to evaluating potential attack paths, prioritizing actions, and iterating strategies. This mirrors a **Markov Decision Process (MDP)** [13] for deterministic actions but incorporates **Partially Observable MDP (POMDP)**[9] when uncertainty is introduced (non-deterministic attacks or unknown system state).

3.4 Environment (E)

The **Environment (E)** represents the dynamic, multi-layered digital ecosystem in which the adversarial agent (the red team) operates. The environment serves as both the *target of the evaluation* and the *contextual space* where all interactions, transitions, and strategic evaluations occur. Unlike simplistic threat models, APTM defines the environment with fine-grained granularity and system-level awareness. It encapsulates not just technical infrastructure, but it can also incorporate human, behavioral, and policy-driven variables that influence the outcomes of both deterministic and non-deterministic actions. This includes the target systems, the defenses in place, and the various points of interaction available to the agent.

The following illustration provides an example of a typical target Environment but more complex environments can be represented such as an industrial controls network or communications network with signaling plane, partner interconnections and roaming interfaces or a product comprised by hardware, software components, network interfaces, API's etc.

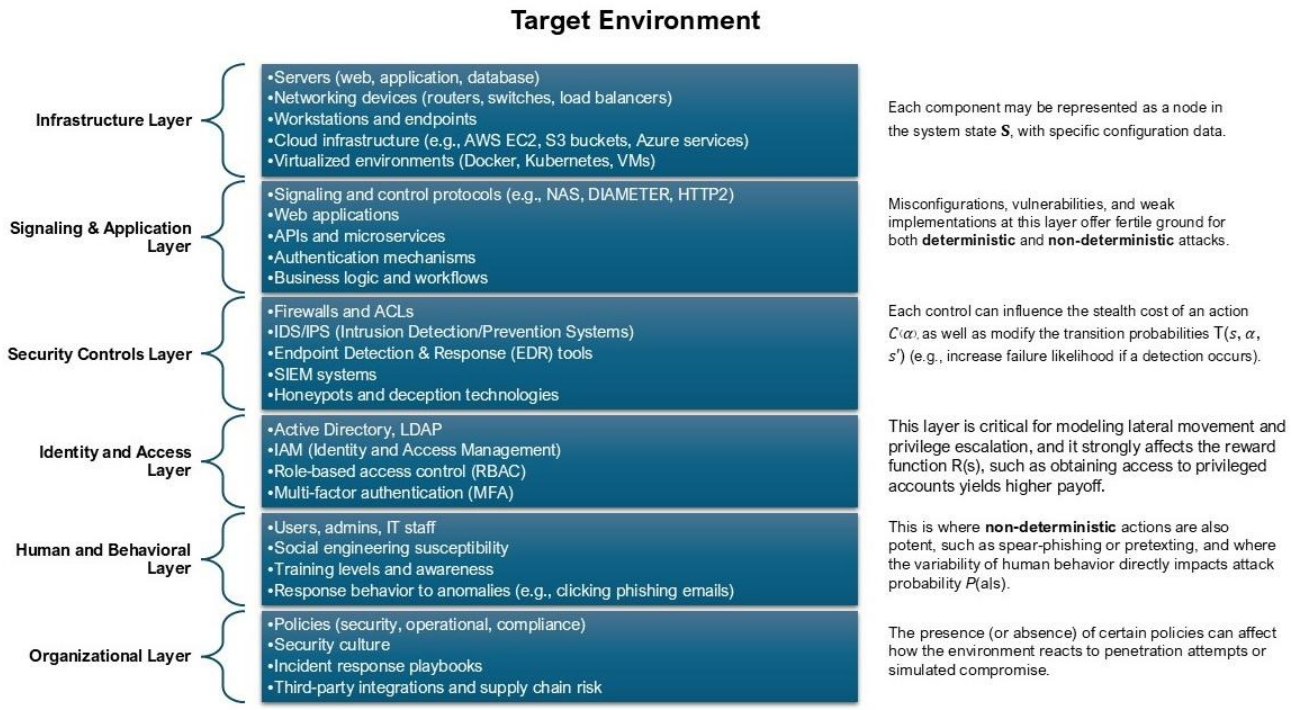

Figure 2 Target Environment Layers

- **Infrastructure:** Represents the physical and virtual resources of the system being evaluated, such as servers, network devices, workstations, cloud resources, and databases. The complexity and configuration of these resources will influence the agent's strategies. Note that *system* can also entail a product, comprised by various components and APTM is applied to verify and measure its security posture.

- **Defensive Measures:** This includes the security mechanisms designed to thwart attacks, such as firewalls, intrusion detection/prevention systems (IDS/IPS), endpoint detection and response (EDR) tools, network segmentation, and encryption. Furthermore, security mechanisms of a *system* representing a product may include, file-permissions, Mandatory Access Control (MAC), SMEP (Supervisor Mode Access Protection) and SMAP (Supervisor Mode Address Protection) to prevent attackers from executing user-space code. These defensive measures provide resistance to the agent's actions and may dynamically adapt over time, which is modeled in APTM through the feedback loop.

- **Access Points:** These are the potential vectors through which the agent can attempt to infiltrate the system. Access points could include exposed services, open ports, misconfigurations, APIs, or unpatched vulnerabilities that the agent can exploit to gain a foothold.

- **Human Layer:** This refers to the social engineering aspect of the environment, such as employees and user behavior. Social engineering tactics like phishing, pretexting, and baiting can be used to manipulate individuals into unwittingly assisting the agent in gaining access to the system. This layer adds an unpredictable and dynamic element to the environment.

The environment (E) is formalized as a context-sensitive structure:

$$E = \{I, A, S, H, P\}$$

Where *I* is the infrastructure configuration, *A* the active applications and exposed services, *S* represents the security controls in place, *H* the human actors and behavior profiles and *P* the organizational policies. Each of these can change during the course of a penetration test (e.g., detected events might activate IDS, generate logs and alerts, enhance firewall filtering and/or alert SOC teams and consequently altering *S* in mid-operation). Thus, in the APTM model, the environment is not considered static since it can react to agent actions through various reactive controls including, alerts and Logging (e.g., system violations or device tamper resistant sensor activation), adaptive defense (e.g., Supervisor Mode Access Protection, auto-scaling WAF rules), lockouts or account suspensions and trigger incident response (e.g., endpoint isolation).

This introduces real-time feedback, enabling the agent to learn from failed or successful actions via the feedback loop **Λ**, creating a **partially observable** and **adaptive** environment. The richness of the environment determines how closely APTM mimics real-world adversaries. A robust model includes a probabilistic modeling of human reactions., temporal elements (e.g., time of day affects access or user behavior), environmental noise (e.g., legitimate traffic, benign alerts) along with **Unknown unknowns**; elements the agent has no prior knowledge of, necessitating exploration.

The environment directly influences, the **choice of actions** (what's possible or worth attempting), the **success of actions** (what defenses are encountered), the **strategy** (e.g., stealthy evasion vs. brute-force escalation) and **learning** (how agent policies adapt based on environmental feedback).

3.5 Agent (A)

The **Agent (A)** represents the adversary navigating the environment to achieve specific goals, such as data exfiltration, privilege escalation, or persistent access. The agent is the intelligent, goal-directed entity executing both deterministic and non-deterministic actions based on a strategic policy and feedback from the environment. The agent could be a human (red team operator), an automated tool (scripted system or tool), or an AI/ML-driven agent that learns and adapts its strategies over time.

This agent can be instantiated as a human red team operator, an automated script-based attack system, or more powerfully, an AI/ML-driven autonomous attacker An agent in APTM is defined by three core traits:

i. **Knowledge Base (K):** Represents the information the agent has about the environment, such as system configurations, previously discovered vulnerabilities, and known defenses. In the case of AI agents, this knowledge base is continually updated based on feedback and previous actions. This represents the information the agent has about (a) the environment state S, (b) known vulnerabilities (c) toolsets and capabilities and (d) historical outcomes (prior successes/failures). The knowledge base evolves over time as the agent performs reconnaissance or receives feedback from executed actions.

ii. **Strategy (Σ):** The strategy or plan that the agent follows, which governs which actions it selects based on the current state of the system. A strategy might involve deterministic actions (e.g., exploiting known vulnerabilities) or non-deterministic actions (e.g., protocol fuzzing to identify 0-day vulnerabilities, trying a new social engineering tactic). This is the high-level decision framework used to choose actions based on (a) rule-based heuristics, (b) cost-reward balancing, (c) risk tolerance thresholds, (d) goal prioritization. In deterministic scenarios, the strategy is simple and rule-based Σ^{D} (deterministic planning tree, knowledge-driven). In non-deterministic cases, the strategy $\boldsymbol{\Sigma^{N}}$ incorporates probability and learning, requiring adaptive planning (e.g., probabilistic sampling, fuzzing, ML, intuition, creativity).

iii. **Adaptation Loop (Λ):** This loop represents the agent's ability to adapt its strategy based on feedback from the environment. After each action, the agent assesses the outcome and updates its knowledge base accordingly. This continuous learning process allows the

agent to improve its chances of success over time. The feedback mechanism that modifies behavior based on (a) action outcomes (success/failure), (b) changes in environment, and (c) updates in knowledge base. The adaptation loop enables learning-based evolution, where the agent refines its strategy over time to optimize future decisions.

The agent A is formally represented as:

$$A = (K, \Sigma, \Lambda)$$

Where, K is the Evolving knowledge base, Σ is the strategy function determining action a ∈ A given current state s ∈ S and Λ (lambda) learning and adaptation operator, mapping observations to policy updates.

Types of Agents include, but not limited to:

- **Human Pen Tester / Red Teamer**
  - Intuition-based decision-making
  - Guided by experience and real-time interpretation
  - Can simulate irrational or unpredictable behavior
  - Learning is tacit and slow but creative
- **Scripted/Automated Agents**
  - Predefined action chains
  - Linear or tree-like decision paths
  - Quick execution but limited adaptability
  - Can replicate deterministic behavior with precision
- **AI/ML Agents**
  - Reinforcement learning, Q-learning, or Bayesian inference
  - Probabilistic decision-making under uncertainty
  - Real-time policy optimization
  - Capable of modeling stealth, deception, and complex reward optimization

These agents represent varying levels of autonomy, risk modeling, and adaptability, aligning with different adversary profiles. Given a state s ∈ S, the agent queries the knowledge base K for known attributes, (b) applies its strategy Σ to compute optimal action $\alpha^* \in A$, (c) executes a^*, transitioning to state s′ with probability T(s,a,s′) and (d) observes feedback and updates K and Σ using Λ. Furthermore, Agents may be designed with different cognitive and behavioral profiles including *Aggressive* (i.e., prioritize high-reward actions, even if risky, *Cautious* (i.e., Favor stealth and low-risk actions, even if slower) or *Opportunistic* (i.e., shift dynamically based on environment state and learned cues). This enables simulation of realistic threat actors with distinct behavioral patterns, enhancing the training value for blue teams and automated defense systems. And the agent can be designed to operate in a continuous loop using action categories such as *Sense* (observe the current state), *Plan* (select action), *Act* (execute the action) and *Learn* (evaluate outcome and refine policy). This loop mirrors the traditional "OODA" (Observe, Orient, Decide, Act) cycle [5], applied in a formal mathematical context.

3.6 Deterministic vs Non-Deterministic Actions

In APTM, every action $\alpha \in A$ taken from a state $s \in \boldsymbol{S}$ has a certain probability of success which is defined as

$$P_{success}(\alpha, s) = \Pr[s' | s, a]$$

These actions fall into two categories based on predictability and outcome certainty, Deterministic and Non-Deterministic actions. This is a key feature distinguishing **deterministic** from **non-deterministic** penetration testing strategies. *Actions* (α) represent the discrete set of operations that an agent may execute within a given state $s \in S$ in order to transition to a new state $s' \in S$. Each action is selected based on strategic considerations, environmental conditions, and probabilistic outcomes.

$$P(\alpha \mid s) = \begin{cases} 1, & \text{success if } a \in A_D \text{ and preconditions are met} \\ 0 < p < 1, & \text{success if } a \in A_N \\ 0, & \text{failure if action is infeasible fromm state } s \end{cases}$$

- A_D : Set of deterministic actions.

- A_N : Set of non-deterministic actions.
- p: Probability derived from environmental uncertainty, agent confidence, past performance, etc.

Deterministic actions (α^D) yield predictable outcomes when preconditions are satisfied (known inputs yield predictable outputs). Their success is binary (either successful or not) and typically based on known information. If an attacker attempts to exploit a well-documented vulnerability in a system (such as an outdated version of a web server with a known exploit), the outcome is typically deterministic, as long as the preconditions (e.g., vulnerable version) are met. Examples of deterministic attacks include, exploitation of known CVEs (e.g., EternalBlue, Log4Shell), port scanning, password cracking, or privilege escalation via misconfigurations. The probability of success for deterministic actions is characterized as $P(\alpha^D \mid s) = 1$, given state s assuming conditions are fully met (or with low variability in outcome), they are often repeatable across environments and are frequently targeted early in attack paths.

Non-deterministic actions (α^N) have uncertain or probabilistic outcomes due to dynamic environmental factors, human elements, or unknowns in the system state (i.e., buffer overflow/underflow, race conditions). Examples activities, include Phishing (human variability in response), Fuzzing/Zero-Day discovery, brute force against unknowns, Social Engineering (e.g., vishing, pretexting) or propagation in unmapped networks (e.g., lateral movement through undocumented infrastructure).

$$P(\alpha^N|s) = f(K, \Sigma, s, E)$$

The variability of success of non-deterministic actions is $0 < P(\alpha^N|s) < 1$, $(\alpha \in A_N)$ where the outcome depends on real-time feedback and environmental dynamics, the risk-reward tradeoff is often higher and requires adaptive learning and strategic tuning. The result is dependent on the agent's Knowledge base (K), its strategy function (Σ) and the environment (E) state. Examples of such actions include:

Protocol Fuzzing: malformed protocol messages can be interpreted incorrectly by the receiving service and introduce unexpected behavior resulting in service or system disruption or memory overflow and system compromise. Thus, the unexpected behavior makes the results probabilistic.

Phishing Attacks: The success of a phishing attempt is not guaranteed and depends on various factors such as the context, the skills of the attacker, and the vigilance of the target.

Social Engineering: An attempt to manipulate a system administrator into divulging sensitive information may succeed or fail based on the social environment.

Table 1 Action Example Comparison

Action	Type	Predictability	Cost	Reward	Adaptability
Exploit CVE-2021-34527	Deterministic	High	Low	Medium	Low
Phishing Campaign	Non-Deterministic	Medium	Medium	High	High
Nmap Scan	Deterministic	High	Low	Low	Low
Zero-Day Discovery	Non-Deterministic	Low	High	Very High	Medium

3.7 Feedback loop and adaptive learning

In traditional penetration testing, once a test is completed, the feedback is typically provided as a report, which may contain vulnerabilities and recommendations for remediation. However, this process is static and does not allow for the continuous adaptation that occurs in real-world adversaries. The feedback loop in the APTM is an essential component of its dynamic nature. After each action, the agent assesses the environment's response, whether it was successful or whether the system has detected or mitigated the corresponding attack. This feedback is used to adjust the agent's knowledge base and strategy and is formally represented as:

$$\Lambda_t = Update\ (P_{success}\ , R, C, s_t, \alpha_t)$$

The feedback operator Λ can be implemented using various learning mechanisms such as Reinforcement Learning (RL) or Bayesian learning to maximize the expected cumulative reward over time [8]. The update function applied at a time step t modifies the agent's understanding of action outcomes and incorporates the observed results of an action and adjusts future behavior (e.g., improves probability estimates, adjusts cost evaluations, updates strategies) to facilitate the selection policy π and optimize the agent's learning based on which actions are most effective and under which conditions. The action success probability $P_{success}$ is the agent's estimate of how likely action α is to succeed in state s and helps determine whether an action is worth attempting based on historical data and current state. The $P_{success}$ function can be initialized statically or learned dynamically through Bayesian inference, reinforcement learning, or other probabilistic methods.

The learning loop enables agents to identify high-value low-cost actions, reduce ineffective or risky behaviors and shift toward strategies with higher observed success. The update adjusts the action probabilities, accounting for the outcome of the action taken. If a certain attack path was detected, the agent may alter its strategy and choose less detectable actions. If a defense mechanism was bypassed, the agent may refine its approach to increase the chances of future success. For Reinforcement Learning update (Q-Learning) [14] we associate each state-action pair with a Q-Value:

$$Q(s,\alpha) \leftarrow Q(s,\alpha) + \lambda\,[R(s') + \gamma \max_{\alpha'} Q\,(s',\alpha') - Q(s,\alpha)]$$

Where λ is the learning rate, γ is the discount factor, $R(s')$ is the reward received after transitioning to state s'. This allows the agent to learn the expected reward for actions and optimize overtime. To select the most optimal action α that achieves the best score for a given state α we use:

$$\pi^*(s) = arg\ \max_a Q(s,\alpha)$$

For example, in the context of penetration testing we may have three possible actions, $\alpha_1, \alpha_2, \alpha_3$ with expected outcomes:

$$\mathbb{E}[R(\alpha_1)] = 5$$

$$\mathbb{E}[R(\alpha_2)] = 12$$

$$\mathbb{E}[R(\alpha_3)] = 9$$

Then

$$arg \max_{a\,\in(\alpha_1,\alpha_2,\alpha_3)} E[R(\alpha)] = \alpha_2$$

Because α_2 has the highest expected reward. In the APTM agents select the optimal policy π^* that maximizes expected cumulative reward while minimizing cost and *argmax* finds the best such policy:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t\ (R(s_t) - C(\alpha_t))\right]$$

Where $R(s_t)$ is the reward for reaching a high-value state, and $s_t$is a snapshot of the environment at the current step (e.g., port 445 is accessible, credentials for admin obtained or have local root access). The $C(\alpha_t)$ function represents the cost of taking action α_t (e.g., run CVE-2021-34527 exploit or attempt credential reuse on a different system) at time *t*, which helps the agent optimize its actions based on the observed outcomes, and γ represents the discount factor balancing short-term vs. long-term goals. This models intelligent agents that weigh potential gain against action costs and risks. Every action incurs a cost **C** measured in terms of, time **t** (duration to execute), resources **r** (e.g., CPU usage, bandwidth, system calls, external services), and stealth risk $\boldsymbol{\delta}$ (likelihood of detection), defined as the cost function:

$$C(\alpha) = t\ (\alpha) + r(\alpha) + \ \delta(a)$$

Where $t\ (\alpha)$ is the time cost, $r(\alpha)$ the resource cost and $\delta(a)$ the stealth or detection penalty.

Over time, the agent develops a robust strategy π^*which dynamically selects the best action based on both deterministic logic and learned probabilities, allowing for efficient and realistic adversarial simulation. This feedback-driven approach enables red teams and autonomous agents to emulate real attackers who adapt, learn, and re-plan based on environmental resistance and behavioral signals.

4 Example: The Silent Slice, Penetrating a Private 5G Network

In this example scenario we apply the APTM to intercept sensitive sensor data in from a private 5G network by simulating a sophisticated, adaptive adversary.

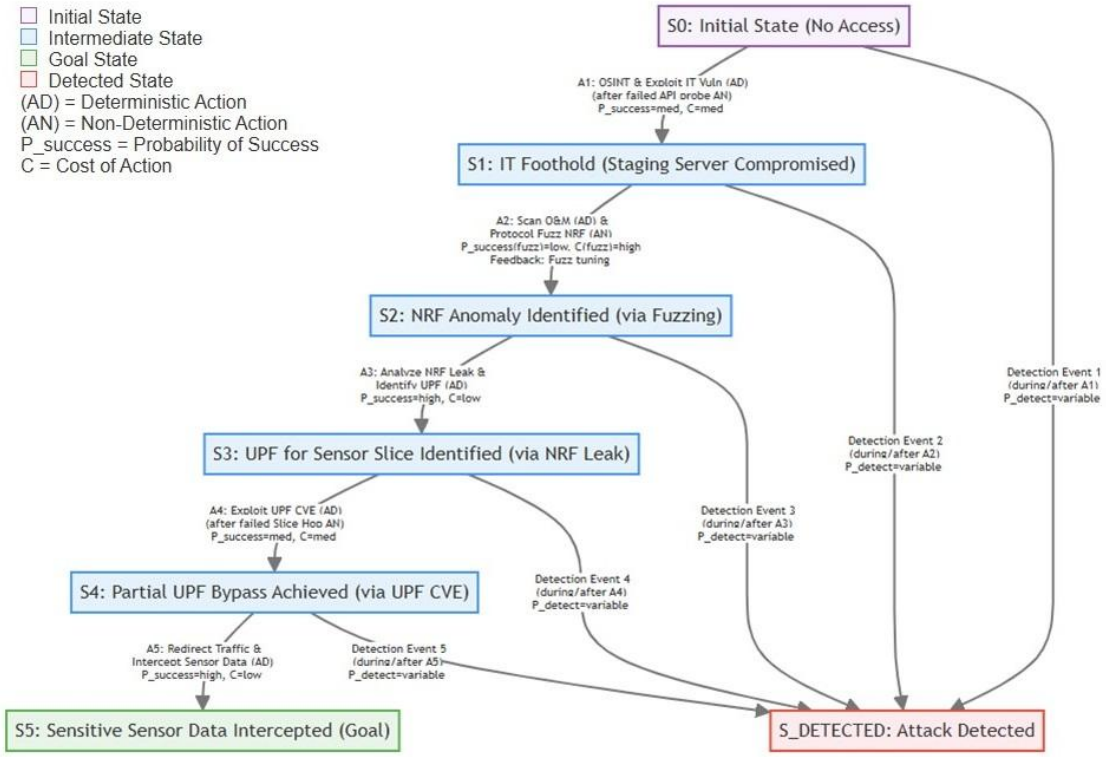

Figure 3 APTM applied during Private 5G Penetration Testing scenario

Phase 1: Breaching the Perimeter (S0 → S1)

Starting from an external position with no access (State S0), the agent initiates its campaign. An initial non-deterministic action (A_N) involves identifying exposed APIs related to the 5G network's edge components; this yields no immediate entry but provides valuable environmental feedback. Adapting its strategy, the agent then executes a deterministic action (A_D), leveraging Open Source Intelligence (OSINT) to identify and exploit a known vulnerability in an internet-facing staging server within the target infrastructure. This grants the first crucial foothold, transitioning the agent to State S1: IT Foothold (Staging Server Compromised). The cost (C) for this initial breach is moderate, with a medium probability of success (P_success).

Phase 2: From IT to the 5G Core's Edge (S1 → S2)

Now inside the IT network, the agent performs deterministic scanning (A_D) of the Operations & Maintenance (O&M) segment, to which the compromised server has unintended access. This reveals the presence of key 5G Network Functions (NFs). The focus then shifts to a critical non-deterministic action (A_N), protocol fuzzing against the Network Repository Function (NRF). The NRF is vital for service discovery within the 5G core. This fuzzing campaign is resource-intensive (high

Cost) with a low initial probability of success. However, the APTM's feedback loop is key; the agent meticulously tunes its fuzzing parameters based on the NRF's responses, eventually triggering an unexpected behavior, an anomaly that doesn't crash the NRF but indicates a subtle flaw. This marks the transition to State S2: NRF Anomaly Identified.

Phase 3: Leveraging Leaks and Targeting the User Plane (S2 → S3 → S4)

The NRF anomaly, upon closer analysis (a deterministic action, A_D), reveals a minor information leak: internal configuration details, including identifiers for User Plane Functions (UPFs) associated with specific network slices, particularly the one handling sensor data. This valuable intelligence propels the agent to State S3: UPF for Sensor Slice Identified. The cost for this analysis is low, and success is high given the prior discovery.

The agent initially attempts a non-deterministic action (A_N): trying to manipulate network slice information or hop between slices based on the NRF leak. This proves unsuccessful but provides further environmental feedback. Adapting, the agent now focuses on the identified UPF and executes a deterministic action (A_D): exploiting a known, unpatched CVE specific to that UPF model. This CVE allows a bypass of certain filtering rules when initiated from a trusted internal source (which the agent now emulates from the compromised IT segment). This targeted exploit leads to State S4: Partial UPF Bypass Achieved, with a moderate cost and probability of success.

Phase 4: Achieving the Objective (S4 → S5)

With the UPF partially bypassed, the agent executes its final set of deterministic actions (A_D), redirecting a portion of the traffic flowing through the compromised UPF. This allows the interception of telemetry data from the targeted sensor network slice. The cost is low, and success is high, culminating in State S5: Sensitive Sensor Data Intercepted. The primary objective of the penetration test is achieved.

Throughout this simulated attack, each action carried a variable probability of detection (P_detect), which, if triggered, would have shifted the agent to State S_DETECTED. The APTM framework allowed for this dynamic interplay of deterministic exploitation of knowns and non-deterministic probing of unknowns, guided by continuous feedback and adaptation, providing a far deeper understanding of the private 5G network's vulnerabilities than a traditional test could offer.

5 Conclusion

Penetration testing, as traditionally practiced, relies on static methods, predefined checklists, and predictable tools. These approaches, while effective in some cases, often fail to account for the dynamic nature of modern cybersecurity threats. Traditional methods typically rely on well-documented exploits and predictable outcomes, limiting the adaptability and efficiency required to simulate intelligent adversaries. This static nature of penetration testing leaves security teams vulnerable to evolving threats and new attack vectors. The Adversarial Penetration Testing Model (APTM) offers a transformative shift in this paradigm by incorporating both deterministic and non-deterministic strategies into the penetration testing process. By integrating mathematical modeling, feedback loops, and adaptive learning, APTM redefines how penetration tests are executed and evaluated. The model allows for more realistic and effective simulations of adversaries by combining predictable, scripted exploits with probabilistic, adaptive strategies. This hybrid approach better mimics real-world adversarial behavior and provides security teams with more comprehensive and accurate assessments of system vulnerabilities. As cyber threats continue to evolve and become more sophisticated, the need for a testing framework that can dynamically adapt to new attack techniques and defensive countermeasures becomes increasingly important. The APTM provides the flexibility and intelligence required to simulate realistic adversarial behavior, enabling defenders to test not only the technical resilience of their systems but also the adaptability of their detection and response strategies. By modeling the attacker as an agent capable of learning, adapting, and balancing risks, APTM supports the development of more robust security postures and better prepares organizations for advanced, persistent threats in an ever-evolving threat landscape.

6 References

1. Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.
2. Bertalanffy, L. von. (1968). *General System Theory: Foundations, Development, Applications*. George Braziller.
3. Biggio, B., Nelson, B., & Laskov, P. (2012). *Poisoning Attacks against Support Vector Machines*. Proceedings of the 29th International Conference on Machine Learning (ICML-12), 1467-1474.
4. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
5. Boyd, J. R. (1986). *Patterns of Conflict*. (Unpublished briefing).
6. Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014), *Explaining and Harnessing Adversarial Examples*. *arXiv preprint arXiv:1412.6572*.
7. Jakobsson, M., & Myers, S. (Eds.), (2006) *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. John Wiley & Sons.
8. Richard S. Sutton and Andrew G. Barto, (2018) *Reinforcement Learning: An Introduction*
9. Leslie P. Kaelbling, Michael L. Littman, Anthony R. Cassandra, 1998 *Planning and acting in partially observable stochastic domains,* Elsevier, Artificial Intelligence
10. V. Neumann, J., & Morgenstern, O. (1944). *Theory of games and economic behavior.* Princeton University Press.
11. H. Raiffa, 1968 *Decision Analysis. Introductory Lectures on Choices under Uncertainty*. Reading, Massachussets, Addison-Wesley, 1968
12. Myerson, R. B. (1991). *Game Theory: Analysis of Conflict*. Harvard University Press. https://doi.org/10.2307/j.ctvjsf522
13. Martin L. Puterman (2014). *Markov Decision Processes: Discrete Stochastic Dynamic Programming,* Wiley
14. Watkins, C.J.C.H., Dayan, (1992) P. *Q*-learning. *Mach Learning*, Springer