Deceive, Detect, Disrupt: Autonomous Digital Twin Decoy Networks for Cyber Threat Engagement

Abstract

This proposal outlines the development of a realistic **AI-driven digital twin decoy network** leveraging **Microsoft Azure** to enhance cyber defense through deception. The system will simulate a secure, enterprise-grade network environment in the cloud, dynamically adapting its traffic patterns and system behaviors using advanced AI/ML techniques – including **Generative Adversarial Networks (GANs)** for synthetic yet lifelike traffic generation, **reinforcement learning (RL)** for adaptive response strategies, and **time-series modeling** for credible temporal activity patterns. The entire decoy environment is defined and deployed using **Infrastructure as Code (IaC)**, ensuring consistency, scalability, and minimal manual effort in managing Azure resources. By mirroring the structure and behavior of a real corporate network, this digital twin decoy is designed to **mislead even state-sponsored cyber adversaries**, confounding their reconnaissance and exploitation efforts. Simultaneously, the platform will **detect, distract, and track adversaries** by collecting high-fidelity threat intelligence from their interactions within the decoy environment and feeding those insights back to defenders in real time. This proactive defense strategy aims to stay ahead of attackers by safely learning from their tactics in a sandbox environment, ultimately hardening the real networks. We present the background and related work in digital twins and cyber deception, detail the proposed system architecture and AI/ML methodology, describe the Azure-based deployment and automation approach, discuss security considerations and use cases, and analyze the strategic impact of this approach for Air Force and broader cybersecurity stakeholders.

Introduction

Modern organizations (in both commercial and defense sectors) face an onslaught of increasingly **sophisticated cyber threats** that often outpace traditional security defenses. Attackers — including advanced persistent threats (APTs) from state-sponsored groups — employ stealthy techniques to quietly infiltrate networks, exploit unknown vulnerabilities, and exfiltrate sensitive data. In this evolving threat landscape, breaches frequently go undetected until it's too late, resulting in costly damage or loss of critical information. To combat these challenges, defenders are seeking proactive and innovative strategies that can **outmaneuver attackers** rather than just react to incidents. One such approach is **cyber deception**, which involves setting traps or decoy resources within an IT environment to lure attackers into revealing themselves and to study their methods in a controlled setting. A classic example is the honeypot: a decoy system intentionally designed to appear as a lucrative target, thereby attracting adversaries and allowing defenders to observe malicious activities. Honeypots and broader honeynet environments have been used for decades to gather threat intelligence and divert attacks away from real assets. However, traditional honeypots are often static and limited in scope, making them easier for advanced attackers to eventually fingerprint and avoid. What is needed is a more **dynamic, lifelike decoy environment** that can continuously adapt and convincingly emulate a real network over time.

In parallel, the concept of the **digital twin** has risen to prominence as a means to create virtual replicas of physical systems. A digital twin is a **comprehensive virtual model** of an object, system, or environment that is kept synchronized with its real-world counterpart. Originating in Industry 4.0 and IoT, digital twins enable real-time monitoring and simulation of physical processes by mirroring sensor data and system states. This technology allows for what-if analyses, predictive maintenance, and optimization without risking the actual system. Recently, this idea has begun to **influence cybersecurity**, with researchers

proposing the use of digital twins to model entire network environments and simulate cyber threats within these virtual replicas. For example, a digital twin of a company's network can be used to **safely run cyber-attack simulations** and observe effects without endangering production infrastructure.

Combining the principles of cyber deception with digital twin technology leads to the concept of a **"digital twin decoy network."** This is essentially a high-fidelity **virtual replica of an enterprise network** that serves as a decoy environment for adversaries. Unlike a standalone honeypot (which might be a single fake server or service), a digital twin decoy is an **entire network simulation** – complete with multiple hosts, services, user behaviors, data, and interconnecting traffic – all designed to be indistinguishable from a real environment. The decoy network is instrumented with extensive telemetry to monitor attacker actions closely. Because it is a flexible software-defined environment, it can **adapt on the fly** in response to attacker behaviors or evolving threat intelligence. Traditional honeypots are largely passive traps, whereas a digital twin decoy provides a dynamic and integrated approach: it not only attracts attackers but also **simulates an entire network's behavior** in real-time, turning the decoy into an active defense system rather than just a passive sensor. In practice, this means as an attacker explores the decoy, the environment can evolve (e.g. revealing new fake assets or altering responses) to keep the attacker engaged and unaware of the ruse.

In this proposal, we advocate building such an AI-driven digital twin decoy network on **Microsoft Azure**, leveraging the **Azure Digital Twins** platform as a core enabling technology. Azure Digital Twins (ADT) allows modeling of complex environments as a graph of digital entities ("twins") with properties and relationships defined in Azure's Digital Twins Definition Language (DTDL). Originally intended for IoT scenarios (smart buildings, factories, etc.), ADT provides an ideal framework to model an enterprise network's topology – including devices, servers, user workstations, applications, and their inter-relations – in a structured way. By using Azure Digital Twins to represent the decoy network's blueprint, we gain a high level of abstraction and programmability: the model defines what types of nodes (e.g., Windows 10 client, Linux server, network switch, user account) exist and how they connect (e.g., subnets, trust relationships, data flows). **Azure's cloud infrastructure** can then instantiate this model as live resources: virtual networks, virtual machines (VMs) or containers, and platform services, all deployed through code. In line with modern DevSecOps practices, we will employ **Infrastructure as Code** tools (such as Azure Resource Manager templates, Bicep, or Terraform) to automatically deploy and configure the entire decoy environment. This ensures the environment can be reliably recreated or scaled on-demand, and that any changes to the decoy network (for example, adding a new subnet or simulating a new application) are managed through version-controlled code for consistency and repeatability.

Crucially, simply cloning a network's topology is not enough – **sophisticated adversaries will quickly recognize a fake environment if the behavior of systems and users seems artificial or static**. Therefore, our system integrates AI/ML techniques to imbue the digital twin with lifelike behavior. We will use generative models to produce realistic network traffic and host activity, and adaptive algorithms to respond to the attacker in real time, maintaining the illusion of a real network. Techniques like **GANs** can create plausible network traffic flows and data artifacts; prior research has shown that replaying recorded traffic is insufficient for deception, as advanced attackers can detect repetitive pattern. Instead, generative models (e.g., sequence-based GANs or variational autoencoders) can produce traffic that is statistically similar to real traffic but with enough variation to avoid fingerprinting. Likewise, **reinforcement learning** will be leveraged so that the decoy environment can autonomously adjust its

responses and environment states based on attacker behavior, effectively giving the decoy a "brain" to orchestrate dynamic interactions. In summary, the proposed solution merges cloud-based digital twin technology with cutting-edge AI/ML to create a **dynamic, self-adapting decoy network** that addresses the limitations of static honeypots.

SBIR Topic Objectives and Innovation

This work directly addresses the key objectives of USAF SBIR topic **AF252-0005: AI/ML-Generated Decoy Networks**:

- **Maximum Automation:** The decoy network generation and management are highly automated. We use Azure Digital Twins as a programmatic model of the network and **Infrastructure-as-Code** deployment pipelines to instantiate decoy environments with minimal manual input. Automated orchestration handles environment setup, attacker engagement, and teardown, allowing defensive cyber operators to deploy realistic decoys on-demand with one-click effort.
- **Deception Realism:** The decoy is lifelike enough to deceive a sophisticated state-sponsored hacker. Advanced AI/ML drives realism: **GAN-based generators** create synthetic network traffic, user activity, and data that statistically mimic real operations, while time-series models reproduce realistic usage patterns (e.g. daily login routines, periodic backups). The decoy hosts are populated with believable system logs, files, and behaviors. By mirroring a real network's topology and habits (even incorporating sanitized patterns from live networks), the decoy appears authentic and evolves over time, making it extremely difficult to distinguish from a real network.
- **Adversary Tracking and Intelligence:** All adversary interactions in the decoy are monitored and recorded in detail without alerting the attacker. Every malicious action (network scans, exploits, lateral movements, data exfiltration attempts) is captured via instrumented logging on decoy hosts and networks. These events feed into a threat intelligence pipeline (e.g. streaming to an analysis platform like Azure Sentinel) that **tracks the attacker's tactics, techniques, and procedures (TTPs)** in real time. The system can identify Indicators of Compromise and even capture malware for analysis. This provides defenders with a real-time view of adversary behavior as well as forensic data to inform countermeasures in the real environment.
- **Real-Time Control (Adaptive Responses):** The platform allows dynamic, real-time control over the decoy environment's reactions to the adversary. A **reinforcement learning-based agent** continuously decides how the decoy should adapt (e.g. deploying new decoy systems, modifying responses, or seeding additional fake "breadcrumbs" for the attacker) to maximize engagement. Defensive operators can configure response modes to be fully automated or require human approval (semi-automated) for certain actions. Through a management interface, operators retain the ability to intervene or adjust deception activities on the fly. This ensures that decoy modifications or countermeasures can be executed in real time, either autonomously or under human supervision, as the situation demands.

By innovatively combining cloud automation and AI-driven behavioral modeling, our approach meets the SBIR's challenge of creating **AI/ML-generated, dynamic decoy networks** that are easy to deploy yet convincing enough to fool APT-level adversaries, while giving defenders powerful tools to observe and influence attackers in real time.

System Architecture

The proposed system architecture for the AI-driven digital twin decoy network is composed of multiple layers, integrating Azure cloud components with AI-driven modules. The design is organized into the following layers (illustrated conceptually in our system diagram):

1. Digital Twin Network Model (Design Layer): At the foundation is the **Azure Digital Twins** model of the enterprise network we aim to simulate. Using Azure Digital Twins, we create a graph of **digital twins** representing all relevant entities in a corporate network and their relationships. Each type of network component (e.g. workstation, server, router, user account, application) is defined as a model in Azure's DTDL (Digital Twins Definition Language). For example, we define a model for a "Windows 10 Client Machine" twin with properties like hostname, OS version, installed software, and relationships like *connectedTo* (linking to a switch or network segment twin) or *hasUser* (linking to a user account twin). Similarly, a "Corporate User" twin has properties like role, fake credentials, typical login times, and relationships to the devices they use. Higher-level constructs such as network segments (e.g., HR subnet, DMZ, R&D network) can be modeled as well. This digital twin graph encodes the **blueprint of the decoy environment** – its topology, asset inventory, and even behavioral parameters for different entities. The model can be kept in sync (in a sanitized way) with an actual production network's design (minus sensitive data), or it can represent a purely fictional network designed for deception. The key is that this twin model is **rich enough to generate a lifelike environment** and is accessible programmatically via Azure APIs.

2. Infrastructure Deployment & Cloud Integration (Deployment Layer): Next, the system uses the twin model as a blueprint to deploy a live instance of the decoy network in Azure. We employ **Infrastructure as Code** (IaC) (e.g. Terraform scripts or Azure ARM/Bicep templates) to instantiate corresponding Azure resources for everything described in the digital twin model. This includes creating **Azure Virtual Networks** and subnets to mirror the network segments defined in the twin graph, deploying **virtual machines** (or containers) for each decoy host, setting up appropriate network security groups or firewall rules to replicate the real network's access controls (so that the decoy's network behavior and segmentation look authentic), and configuring Azure platform services as needed to simulate enterprise services (for example, deploying a decoy SQL Database to mimic a corporate database server). The use of IaC ensures we can stand up or tear down the entire environment reliably and repeatedly. It also allows deploying multiple instances of the decoy network if needed – for example, to cover different divisions of an organization or to isolate multiple simultaneous adversaries. Each decoy host VM is instrumented with monitoring and logging agents. For instance, a decoy Windows server might run a logging agent that sends its event logs to an Azure Log Analytics workspace, and a heartbeat agent to report its status back to the Azure Digital Twins instance (closing the loop so the twin's state reflects the actual VM state). The Azure Digital Twins service can be configured to emit events (via Azure Event Grid or Functions) so that changes in the twin state (e.g., a flag indicating a decoy host was "compromised") can trigger automated responses in the cloud environment. All sensitive operations (deploying VMs, configuring networks, etc.) are governed through Azure Role-Based Access Control (RBAC) and principle of least privilege, ensuring the decoy infrastructure is managed securely via code with minimal manual intervention. By encapsulating the environment definition in code, we achieve **consistency** (every deployment is identical unless the model or code is changed), **scalability** (Azure can easily handle

increasing the number of hosts or segments if the model expands), and **maintainability** (updates to the model propagate through an automated pipeline to update the environment).

3. Simulation & Behavior Engine (AI/ML Layer): Once the infrastructure is deployed, the focus shifts to making the decoy network operate in a realistic, autonomous manner. This is handled by the Simulation & Behavior Engine, which comprises several AI-driven subcomponents:

- **Background Traffic Generator (GAN-based):** This component produces background network traffic and system activity so that the environment looks "lived-in" even before any attacker interacts. It uses **generative models** (such as GANs or Variational Autoencoders) trained on real network data from enterprise environments to generate synthetic but realistic traffic patterns. For example, using a dataset of normal corporate network flows (captured from a real network's NetFlow logs or PCAPs), we train a model to generate synthetic network traffic that statistically mirrors legitimate user behavior. Prior work (e.g., the NetShare system) has demonstrated using time-series GANs to capture flow inter-arrival times and packet sizes in order to produce high-fidelity network traces that emulate real traffic. Our traffic generator will create various types of benign traffic: web browsing from decoy user workstations (simulated via actual HTTP requests to innocuous sites or internal web servers), periodic file transfers between application servers and databases, authentication and directory service traffic (Kerberos, LDAP) corresponding to users logging in and scheduled tasks, etc. To an attacker observing the network (e.g., via a packet sniffer on a compromised host), this background traffic should appear indistinguishable from that of a real enterprise. Importantly, we avoid simply replaying the same recorded traffic loops (which could be recognized by an observant adversary). Instead, the generator uses ML to create **statistically similar but non-repeating** traffic, possibly mutating packet timings or payload contents within realistic bounds. The GAN's discriminator (trained during development) helps ensure that the synthetic traffic is hard to differentiate from real traffic. Over time, we can retrain or fine-tune this model with new traffic data (including, potentially, incorporating some attacker-generated traffic patterns to make the environment even more convincing by occasionally mimicking malicious traffic as decoy noise).

- **User & Host Behavior Emulator (Agent-Based + Time-Series):** Beyond network packets, the decoy must simulate higher-level user and system behaviors: employees logging in, typing commands, accessing files, and systems performing maintenance tasks. We achieve this with a combination of scripted agents on the decoy hosts and AI models governing the schedule of these actions. For scheduling, we apply **time-series models** that capture the "rhythm of life" in the enterprise. For instance, an LSTM-based model or statistical model might dictate that logon events for certain user personas occur at 8–9am on weekdays, that data backup jobs run on Friday nights, or that a server exhibits heavy CPU load at end-of-month reporting periods. We can train these schedules using real login and job schedule data (sanitized) from a similar network. We inject seasonality and randomness so that patterns are believable but not overly regular. Each decoy host will run a lightweight agent (e.g., a script or daemon) that executes actions according to these schedules: e.g., logging a decoy user into a workstation via RDP, opening a document or database, or initiating a connection to another service. This creates the **illusion of active users and processes** doing normal work in the network. Modern techniques like **TimeGAN** (a generative adversarial network for time-series) could be employed to generate sequences of

events that have realistic temporal correlations (such as bursts of activity followed by idle periods). The end result is that an attacker poking around hosts will find believable evidence of routine user activity (recent logins, files with recent timestamps, etc.).

- **Adaptive Attacker Engagement (Reinforcement Learning Agent):** Perhaps the most critical AI component activates once an attacker is in play. A **reinforcement learning (RL) agent** monitors the attacker's actions (via the decoy's telemetry) and decides how the environment should react in order to keep the adversary engaged without tipping them off. In a static honeypot, all system responses are predetermined and eventually become predictable; in our system, the RL agent adds a layer of dynamic, learned responses aimed at **prolonging the deception**. Drawing inspiration from recent research like HoneyIoT. we will define a set of possible actions the decoy can take in response to attacker behavior. For example, if the attacker probes a certain service on a decoy server, possible decoy actions might include: responding with a benign error, emulating a known vulnerability (e.g. sending back a fake crash or dummy credentials), or dynamically launching a new decoy asset (e.g. "discovering" another server in the network for the attacker to pivot to). The RL agent's policy will be trained (using simulated attackers or past attack data) to choose responses that maximize the engagement and intelligence-gathering. The state for the RL agent will include the attacker's observed behavior so far (e.g. what systems have been compromised, what tools they are using) and a memory of deceptive artifacts already presented (to avoid repetition or contradictions). The reward function is engineered to reflect our objectives: **maximize attacker dwell time and activity** (for more data collection) and **maximize data gathered** (unique TTPs observed), while minimizing the chance of the attacker getting suspicious or disengaging. We may employ deep Q-learning or policy-gradient methods to train this agent. Effectively, this gives the decoy network a "self-driving" capability for handling intrusions – it can autonomously decide how to react to an attacker in a way that keeps the scenario going. For example, the RL agent might learn to randomize certain system responses (to avoid pattern detection) or to occasionally "sacrifice" a decoy component (letting the attacker feel they've succeeded in stealing something) in order to increase the attacker's confidence.

- **Threat Intelligence Collector & Analyzer:** In parallel to attacker engagement, the system includes a subsystem for **data collection and analysis** of everything the adversary does. All decoy host logs, network traffic captures, and the sequence of attacker commands or malware are funneled to a central analysis platform – this could be a SIEM like Azure Sentinel or a dedicated analytics stack. Here we apply analytical models (some ML-based for anomaly detection and correlation) to interpret the attacker's behavior and extract useful intelligence. For instance, any malware that the attacker uploads can be automatically detonated in a sandbox environment (Azure provides cloud sandbox services or we integrate with tools like Azure Defender for Endpoint) to see its behavior and identify signatures. The system can use ML classifiers to recognize the malware family or threat actor toolkit. All of the attacker's moves are tagged and catalogued (e.g., "attacker used Mimikatz tool at 10:25PM to scrape credentials from decoy memory"). This rich dataset is reported to the defensive operators in real time and stored for post-mortem analysis. The insights feed back into both the security team's knowledge and the AI modules themselves – for example, new attacker techniques observed can become additional training data for the RL agent (so it can handle them in future engagements), and can inform

updates to the generative models (e.g., to generate even more convincing decoy artifacts that mimic the attacker's tools' outputs, further confusing them).

4. Orchestration & Control Layer: Overseeing the whole system is an orchestration layer that ties together the digital twin model, the deployed cloud resources, and the AI components. This layer is implemented through a combination of Azure services (Functions, Logic Apps) and custom control logic. For example, an Azure Function can listen to events from Azure Digital Twins (such as a twin property change indicating a decoy host has been marked "compromised" by the system) and then trigger workflows in response. These workflows might notify the RL agent to adjust strategy, or even automatically scale out part of the environment by deploying an additional decoy server to ensure the attacker has more targets to explore. Another orchestrator routine periodically checks the twin model against the actual deployed resources to ensure they remain in sync; if a discrepancy is found (say an attacker managed to crash a decoy VM), the orchestrator can redeploy that node and update the twin state accordingly to maintain the illusion of an intact network. The orchestration layer also manages **snapshot, reset, and restore** operations: because the decoy environment will likely be compromised and manipulated by attackers, we need a way to revert it to a clean state or create new instances quickly. Using virtualization and IaC, the orchestrator can snapshot the state (or simply record needed state in the twin model), and later automatically rebuild or reset portions of the environment (e.g., re-image a VM, rotate credentials on decoy accounts) to recover from an engagement or prepare for a new one. Importantly, the orchestration also interfaces with the defenders: we will provide a management console or dashboard where cyber operators can monitor the decoy environment's status and ongoing attacker engagements. Through this interface, operators can also manually trigger certain actions or overrides if desired. The system supports multiple modes of operation – from fully autonomous response to human-in-the-loop control – as a safety and flexibility feature. For example, an operator could switch the decoy's response policy to "manual" for a particular high-stakes adversary, requiring analyst confirmation before the decoy executes certain actions (like revealing a fake database). This ensures that **decoy modifications or actions in real time can be automated, semi-automated, or manual** according to operational preference, fulfilling the SBIR requirement for selectable levels of control.

5. Isolation and Safety Layer: Finally, to prevent any risk to real assets, the architecture includes strict isolation measures. The decoy Azure virtual network is completely **isolated** from any production networks – there is no peering, VPN, or shared connectivity between the decoy environment and the real enterprise network. The decoy operates in its own Azure subscription or resource group that is treated as untrusted relative to production. Any integration points for realism (such as ingesting sanitized real telemetry into the decoy) are one-way and brokered through services that do not allow the decoy to initiate connections back into secure networks. Credentials are not shared between real and decoy systems; all user accounts, passwords, and keys in the decoy are fictitious or dedicated decoy credentials. Thus, even if an attacker fully compromises the decoy, they gain no foothold in the real environment. We also control **outbound connectivity** from the decoy to prevent an attacker from using the decoy as a platform to attack external targets or discover the deception. Outbound internet access from decoy hosts is funneled through a controlled egress point (such as an Azure Firewall) where traffic can be filtered and monitored. If an attacker tries to launch an attack from the decoy (e.g., DDoS or spreading malware externally), the security controls will detect and block it, or redirect such traffic to a sinkhole for observation. This "managed interaction" approach means the attacker might believe they succeeded (for example, their malware connects to their command-and-control server, but in reality it's connecting to our

sinkhole logging service). Additionally, operational security steps are taken to avoid tipping off the attacker that they are in a decoy. We remove or spoof any obvious cloud identifiers on decoy systems (for instance, hiding Azure-specific service banners or metadata) so that the environment appears on-premises and consistent with its story. The decoy systems are maintained just like real ones (regular updates, normal user activity) with a careful balance: we intentionally leave certain vulnerabilities or misconfigurations as bait, but in a controlled manner that we can monitor and contain. All decoy data is fictitious (or synthesized from templates) to avoid any real sensitive information; any real data patterns used for realism are heavily sanitized. Data collected from the adversary (logs, malware, captured traffic) is stored securely and handled as sensitive information by our team. We also apply Azure's security best practices in managing the decoy environment itself: using least-privilege access for any management components, isolating the operations team's access, and monitoring the decoy's health for any anomalies that might indicate our simulation has been detected or is malfunctioning. In summary, this layer ensures the decoy is a one-way trap: **easy for attackers to get in, impossible for them to break out** to anywhere harmful.

In aggregate, the system architecture leverages **Azure Digital Twins as the "brain" for modeling**, Azure's cloud infrastructure as the **body** of the decoy network, and a suite of **AI engines as the "lifeblood"** that animates this body with realistic behavior and adaptive deception. This layered design ensures each aspect (modeling, deployment, behavior generation, orchestration, safety) can be developed and refined independently, while Azure provides the scale and integration needed to run the complex environment efficiently.

Detailed AI/ML Methodology

Achieving a truly convincing decoy network requires a multi-faceted AI/ML approach. Here we delve deeper into how each AI/ML technique will be applied and how they work together to create an intelligent deceptive environment.

Generative Models for Synthetic Traffic & Artifacts: *Network Traffic Generation:* We will train specialized generative models to produce network traffic that closely emulates real-world patterns. One candidate approach is to use time-series GANs such as **TimeGAN**, which are designed for sequential data, to capture the joint distribution of network events (e.g., inter-arrival times, protocol mixes, packet sizes). Another model we will experiment with is **DoppelGANger**, which has shown success in generating multi-variate time series data with high fidelity by capturing both temporal patterns and cross-feature correlations. Training data will come from benign network traffic captures (web browsing, DNS queries, database transactions, IoT telemetry, etc.) in enterprise contexts. By training on a mixture of normal traffic types, the generator can output synthetic traffic that retains key statistical properties of real traffic (such as throughput distributions, bursty vs. steady flows, periodic heartbeat signals) without exactly replicating any specific real sequence. During deployment, this generator will run continuously (or at intervals) to inject fake traffic into the decoy network. For efficiency, we may not generate every single packet via an ML model (which could be computationally intensive); instead, the model might periodically output parameters or patterns for traffic flows, which are then enacted by lightweight scripts or agents. For example, the GAN might output "start a 5-minute data transfer flow at 10 Mbps between a file server and a client, with packet sizes following X distribution," and an agent will then simulate that flow with appropriate tools. This hybrid approach ensures realism while staying performant.

Host Logs and File Artifacts: Beyond network flows, attackers who gain deeper access (e.g., a shell on a machine) will scrutinize system logs, running processes, file systems, and other artifacts for inconsistencies. We will use generative techniques to populate **fake log entries and file contents** that are statistically coherent with the system's role. For instance, we can fine-tune a language model (like GPT-style or other large language model) on a corpus of Windows Event Logs or Linux syslogs to generate realistic log sequences. If an attacker runs wevtutil on a decoy Windows server, they would see a believable history of login events, system errors, patches applied, etc., matching the supposed uptime and usage of that server. Similarly, for file content, we can use GPT or similar models to generate dummy documents (e.g., fake PDFs or Word files containing plausible but fictitious business content) relevant to the decoy scenario. These AI-generated texts will be vetted to ensure they contain no real sensitive data — they will be generic or entirely fictitious, but contextually appropriate (for example, if the decoy company is "Acme Defense Corp," documents might discuss fictional projects or internal memos that seem credible). The key is to avoid any telltale markers of automation or emptiness; content should have small imperfections or variations as real data does. We may also use GAN-like approaches for non-text artifacts (e.g., generating user credential hashes that *look* real if an attacker dumps a password database, using a model trained on known password leak datasets). Overall, generative AI will permeate the decoy, filling it with **synthetic but authentic-looking data** at all levels.

Adversarial Resilience: Because we are using AI-generated content in a security context, we must consider that skilled adversaries might attempt to use their own analytics to detect whether something is AI-generated (for example, finding subtle anomalies in traffic or logs). To counter this, our generative model training will incorporate adversarial resilience testing. We will simulate "attacker detectors" during training – e.g., train additional discriminator models aimed at identifying fake vs. real by specific metrics – and ensure our generators learn to fool not just a generic discriminator, but also these specialized detectors. This could involve multi-objective training or adversarial training where we iteratively refine the output until it passes a battery of realism checks (including tests in frequency domain characteristics of traffic, log consistency over time, etc.) This reduces the chances of an attacker's tool flagging the environment as fake.

Reinforcement Learning for Adaptive Engagement: The reinforcement learning component is essentially the decision-making engine that "plays" the role of the environment responding to an attacker. We formalize the interaction between the attacker and the decoy as a sequential decision process (a game). The state observed by the RL agent includes various aspects of the decoy environment and the attacker's progress: which decoy hosts are compromised, what techniques the attacker has used so far, what alerts have been triggered, etc. The action space for the RL agent consists of the adaptive deception actions we can take, as described in the architecture (deploy new decoy systems, modify responses, plant breadcrumbs, throttle or sandbox the attacker's actions, etc.) (network-decoy.md). We will design a reward function that reflects the goals of engagement: the agent gets positive reward when the attacker spends more time or reveals more capabilities, and negative reward for actions that might scare off the attacker or end the interaction prematurely (network-decoy.md). For example, if the attacker continues to run new commands (indicating they are still fooled), that's positive; if the attacker suddenly stops activity (potentially suspecting something), that's negative. Training the RL agent will likely require creating a simulated attacker to play against, or using prior recorded attack traces to evaluate agent decisions. We can start with a simplified simulation (possibly using rule-based attackers or replay of known attack patterns) to train a basic policy. As we gather real engagement data from the decoy (in Phase II trials or

red team exercises), we will incorporate that to further train or fine-tune the agent. An important aspect is safety and sanity-checking the RL outputs: we will not allow the agent to take actions outside a predefined safe set, and certain actions will require consistency checks. For instance, if the agent chooses to spawn a new decoy server, the orchestration layer ensures it is properly configured and doesn't contradict the scenario (e.g., it wouldn't spawn a second "CEO's laptop" out of nowhere because that would be odd in the narrative). The RL-driven adaptation, as demonstrated in studies like HoneyIoT, is expected to significantly increase attacker engagement time and prevent attacker detection of the honeypot by dynamically altering the environment in ways that mimic reality. Our implementation will extend this concept to enterprise IT and cloud networks, giving the decoy a continuously learning and improving defensive posture.

Time-Series Analytics and Anomaly Detection: Time-series modeling plays a dual role in our system. First, as noted, it underpins the generation of normal background behavior (predictable daily or weekly patterns for network and user activity). Second, we will use time-series analysis for internal **attack detection and prediction** within the decoy. While the decoy's purpose is to engage attackers (so we won't shut them down as one would in a real network), we still want to algorithmically detect when a real attacker is active in the decoy (as opposed to our own background simulation). Essentially, nearly any deviation from the baseline pattern in the decoy is by design malicious, so our detection can be very sensitive. We will use anomaly detection on system metrics and logs to flag the presence of an intruder – e.g., if a decoy process begins executing unusual commands or a surge of network connections occurs that wasn't part of the generated background. These detections won't alert the attacker, but will feed into the orchestrator and RL agent to inform them that "the game is on." Additionally, we can apply predictive modeling to anticipate the attacker's next moves to some extent. By analyzing sequences of attacker actions and comparing them to known attack kill-chains or tactics (perhaps using a Markov Model or sequence classification network), we can predict likely next steps (for example, after obtaining a user password, the next typical step might be to attempt lateral movement to a specific server). The system can then proactively prepare certain decoy elements – if we expect the attacker to move to a database server next, the RL agent could ensure that server is ready and seeded with enticing fake data, improving our readiness.

MLOps and Continuous Learning: Given the complexity of these AI components, we will follow **MLOps best practices** to manage their lifecycle. Data from each attack engagement will be logged and later used to retrain or improve the models. For instance, if an attacker found something in the environment suspicious (causing them to disengage early), that's a critical data point – we would adjust the models or content to avoid that in the future. We plan periodic retraining or fine-tuning of the generative models (GANs) and the RL policy using the latest data. Azure's ML platform services (Azure Machine Learning) can automate parts of this pipeline – triggering training jobs, evaluating model performance (e.g., measuring how "real" the traffic looks via statistical tests or red team assessments), and deploying updated models into the decoy environment. We will also incorporate feedback from expert "white hat" operators (e.g., cyber red teams or subject matter experts) by having them test the decoy and provide observations on what aspects felt inauthentic. Their insights will be used to further refine scenarios and model parameters, effectively embedding human expertise into the AI training loop. This continuous improvement cycle means the decoy network will get smarter and more realistic over time. As attacker techniques evolve, our AI components will evolve in response – an **online learning** approach to keep the deception consistently one step ahead of new detection methods.

Integration of AI Components: All these ML components will be integrated into the Azure-based architecture via well-defined interfaces. For example, the traffic generator and user behavior models might output "events" or scripts that the orchestration layer deploys to the VMs (like scheduling a login at a certain time, or sending packets according to a certain pattern). The RL agent will likely run as a service (possibly in a container or VM) that subscribes to a feed of security events (from the decoy's monitoring system) and issues high-level commands to the orchestrator (for instance, telling the orchestrator to launch a new decoy instance or change a twin property that causes a certain behavior). We will ensure these interactions are robust and timely – the decoy must react quickly to keep up with live attackers.

In summary, our AI/ML methodology focuses on two pillars: **fidelity** (making the decoy as realistic as possible through generative modeling of normal behavior) and **adaptation** (making the decoy responsive to threats through reinforcement learning and predictive analytics). By combining these, the decoy environment becomes an ever-learning, dynamic counterpart to the real network – one that can engage and fool adversaries while constantly improving its craft.

Infrastructure Deployment and Automation Strategy

A core requirement for this decoy solution is the ability to deploy, scale, and reconfigure it with minimal manual effort. To that end, we adopt a fully automated cloud deployment strategy on Azure:

- **Infrastructure-as-Code (IaC):** The entire decoy environment – network, VMs, services, and Azure Digital Twins instance – is defined in code (using Terraform scripts or Azure ARM/Bicep templates). This means an operator can deploy the decoy with a single command or pipeline run, and the resulting environment will exactly match the intended design. We are creating modular IaC templates for different components (network segments, host types, data stores, etc.), which take parameters from the digital twin model. For example, a "workstation" module template will create a VM with the necessary specifications (OS image, decoy user accounts, fake data, monitoring agent) and connect it to the appropriate subnet, guided by the twin model's definition. Using IaC ensures **consistency and repeatability** – every deployment uses the same templates, avoiding configuration drift. It also provides **version control** (we can track changes to the environment blueprint over time) and easy rollback if needed.
- **Continuous Integration/Continuous Deployment (CI/CD):** We will set up automated pipelines (using Azure DevOps or GitHub Actions) to manage the decoy environment lifecycle. Any time we update the twin model or deployment code (for instance, to introduce a new type of server or adjust a network policy), the pipeline can automatically apply these changes to the running decoy environment. This enables rapid iteration and also quick recovery – if the decoy needs to be reset, the pipeline can destroy and redeploy the entire environment fresh. In Phase II, this pipeline can also be triggered on a schedule or by an operator to routinely refresh the environment (for example, nightly rebuilds to clear any persistent compromise an attacker may have left).
- **Scalability & Elasticity:** Because the decoy runs in Azure, we can leverage cloud scalability. We can easily increase the size or complexity of the decoy network by adjusting parameters (e.g., deploying 100 decoy hosts instead of 50 across additional subnets) to simulate larger environments if needed. The design also supports spinning up **multiple decoy environments in parallel** if required. For example, if multiple separate threat actors are to be engaged simultaneously (or to isolate different training scenarios), we could deploy separate instances of

the decoy network (in separate Azure resource groups or even separate Azure subscriptions) using the same IaC templates. Azure's ability to rapidly provision resources means we can isolate each adversary in their own decoy "sandbox" without them ever knowing there are other decoys. We can also incorporate auto-scaling rules for certain decoy components; for instance, if the attacker launches a denial-of-service attack on a decoy web server, Azure could automatically spawn additional decoy web server instances to absorb the traffic, maintaining the illusion of a resilient infrastructure.

- **Monitoring & Maintenance Automation:** We treat the decoy environment as a production system from an SRE/DevOps perspective. Azure Monitor will track the health and performance of all decoy components (CPU usage on VMs, memory, disk, network I/O, etc.). If any component crashes or behaves unexpectedly (not due to attacker activity but due to a bug or resource exhaustion), automated scripts or functions can attempt to restart or redeploy that component. We will implement automated maintenance tasks via Azure Functions or Automation Runbooks – for example, periodically seeding new fake data (rotate decoy file content, change dummy user passwords on schedule) so that the environment never becomes stale. These tasks and checks are also defined as code, ensuring reliability. When we need to update the decoy software (say we improve the agent code or AI models), we can utilize a blue-green deployment approach: deploy a second instance of the decoy environment with the updated software, test it, then switch attacker traffic to it (e.g., by manipulating DNS if the attacker's point of entry is via a domain name) and decommission the old instance. This avoids disrupting an engagement while still allowing upgrades.

- **Security Automation:** Security guardrails are built into the deployment process itself. For example, the IaC templates will include Azure Policies or configuration that ensure no management ports (SSH/RDP) are left open except what's intentionally part of the deception, and that the decoy VMs only use the virtual network we set (to avoid any accidental connection to real subnets). The Azure Digital Twins instance is deployed with strict access control (using managed identities and role assignments) so that only our decoy orchestrator service can query or modify it, preventing even a compromised decoy VM from directly interfacing with the twin management plane. We also leverage Azure Key Vault for any sensitive secrets (though the decoy is mostly self-contained with fake credentials, any service keys for Azure APIs or certificates for decoy servers will be stored securely). In short, the same automation that makes deployment easy also enforces the security isolation by design, reducing the chance of misconfiguration that could expose the deception.

By using automation at every step – from environment provisioning to runtime operations – we meet the SBIR objective of **maximal efficiency with minimal manual input**. An operator could deploy a complex decoy that mirrors a real network's blueprint in minutes, and the system can automatically keep it running and adjust it as needed. This also makes the solution portable and repeatable: the decoy can be deployed at different locations or tweaked for different network scenarios by modifying the twin model and rerunning the IaC pipeline, without rebuilding everything from scratch.

Security Considerations

Designing a cyber deception environment that actively invites adversaries requires careful controls to manage risk. Key security considerations and our mitigation approaches include:

- **Isolation from Real Systems:** The decoy network is completely isolated from production networks and data. It operates in a separate Azure environment with no inbound or outbound trust to real systems. Any data imported for realism (e.g., statistical patterns from production logs) flows one-way into the decoy and never includes actual sensitive information. Credentials used in the decoy are unique to it; even if stolen by an attacker, they are useless outside the decoy.
- **Preventing Malicious Use of Decoy:** We prevent attackers from leveraging the decoy as a stepping stone to harm others. Outgoing traffic from the decoy to the internet is tightly controlled (via firewall rules and monitoring). We allow just enough outbound connectivity to make the decoy convincing (e.g., letting a decoy host perform Windows updates or an attacker download their tools) but we intercept anything clearly malicious. If an attacker tries to launch attacks from the decoy, those packets can be dropped or diverted to a sinkhole under our control. This way, our decoy doesn't become an unwitting accomplice in cybercrime.
- **Avoiding Attacker Detection of the Decoy:** Operational security (OPSEC) is maintained so that attackers remain unaware they are in a fake environment. We scrub obvious cloud fingerprints from the decoy hosts – for instance, removing Azure-specific agent identifiers, using internal hostnames and IP schemes that look like a typical enterprise (no ".cloudapp.net" addresses, etc.). The decoy systems generate normal system noise (CPU load, memory usage, patch levels) consistent with real machines. We will even introduce some intentional imperfections (like a slightly misconfigured server or an outdated software version) to avoid an overly pristine environment that might arouse suspicion. Those intentional weaknesses are carefully chosen and isolated so that when the attacker exploits them, they don't jeopardize control of the decoy. Essentially, we make the decoy "imperfect" in realistic ways to serve as believable bait.
- **Protecting Collected Data:** All information gathered from attacker interactions (e.g., keystrokes, files, tool output) is stored in a secure repository accessible only to authorized defenders. Though the decoy data is fake, the attacker might inadvertently provide real stolen data or sensitive tools during their operation, so we treat the collected data as sensitive. It will be encrypted at rest, and any sharing of the threat intelligence outside our team will be done carefully (sanitizing any references that could reveal the decoy's identity or techniques to the attacker).
- **Cloud Security & Hardening:** We apply strict security to the Azure infrastructure hosting the decoy. Only our decoy management services (or admins) have credentials to the Azure subscription running the decoy; regular IT admins of the real network have no access (preventing them from accidentally exposing the deception). Within Azure, we use least-privilege roles for each component; e.g., the RL agent service might only have permission to send commands to a queue and read certain logs, nothing more. We avoid direct manual administration of decoy VMs; all actions go through our orchestrated pipelines, leaving less room for human error that could tip off an attacker (for example, an admin login appearing in logs unexpectedly).
- **Handling Multiple or Persistent Threats:** If multiple attackers end up in the decoy at different times, we ensure one won't see traces of another. After an engagement, the decoy can be automatically reset or rebuilt (wiping any backdoors the attacker may have left). If simultaneous,

we isolate them in separate decoy instances. This prevents attackers from detecting "noise" or alterations caused by other intrusions.

- **Operational Monitoring of the Decoy:** We monitor the decoy environment itself for signs of trouble – not just attacker activity, but issues like an attacker possibly detecting the decoy. For example, if our normally active decoy processes stop (maybe the attacker managed to kill our simulation agent on a host), that's an anomaly we detect and respond to (perhaps by restarting the agent or spawning a replacement host). Our team maintains OPSEC when interacting with the decoy: analysts will not log into decoy machines in obvious ways or leave any artifacts that an attacker could find. Communication about the decoy is done out-of-band. We treat running the decoy like running an intelligence operation, where stealth on the defender's part is also important.

By addressing these considerations, we ensure that the decoy network yields maximum defensive value (engaging and observing adversaries) with minimal risk. The decoy is a controlled environment where attackers can be allowed to "play" freely, but that freedom ends at the decoy's boundaries.

Use Cases

To illustrate the practical utility of the AI-driven digital twin decoy network, we describe several representative use cases:

Use Case 1: APT Lateral Movement Trap (Enterprise Scenario). A large enterprise suspects it is targeted by an Advanced Persistent Threat (APT) group. The company deploys the decoy network to mirror a particularly sensitive segment of its real network – for example, the R&D subnet where valuable intellectual property is stored. The decoy is populated with enticing fake documents (e.g., project plans or schematics generated to look authentic). If the APT breaches the perimeter of the real network and begins lateral movement, the deception system quietly redirects the attacker into the decoy environment (using techniques like ARP spoofing or DNS manipulation on compromised hosts to point them to decoy servers instead of real ones). Once inside the decoy R&D network, the attacker encounters what appears to be the real environment: machines and file shares with expected naming conventions, user accounts, and data. The AI-driven behavior engine ensures that everything responds consistently and normally – the attacker can even perform actions like opening files or running network scans without raising suspicion. Suppose the attacker exfiltrates the fake documents, believing they hit the jackpot. Meanwhile, the defenders have silently recorded the entire kill chain: the tools used, exploits attempted (perhaps even a zero-day), and the exact data exfiltration path. All of this intelligence is invaluable – the defenders can now remediate those attack vectors in the real network (e.g., if a previously unknown exploit was used, they can apply virtual patches or enhanced monitoring in the real environment). In this scenario, the decoy both protected the real "crown jewels" by diverting the attacker and provided a detailed forensic record of the APT's tactics. This information can be shared within the industry or with government agencies to bolster collective defense.

Use Case 2: Threat Hunting & Intelligence Collection (Government/Defense Scenario). A government cyber defense team or defense contractor sets up the decoy network as a **threat intelligence honeynet**. They configure the decoy to resemble a small defense contractor's network, complete with faux project names and email accounts. They then deliberately "leak" the existence of this network in forums or via phishing campaigns (for instance, they allow certain phishing emails to successfully phish

credentials that only work on the decoy). This attracts state-sponsored hackers interested in defense secrets. When an attacker uses those stolen credentials to VPN into the decoy, they believe they have breached a real defense contractor. The decoy then observes the attacker's post-compromise behavior: reconnaissance, attempts to escalate privileges, and so on. The **RL agent** in our system notices, for example, the attacker looking for domain admin privileges, and dynamically "reveals" an enticing new target: a decoy domain controller containing references to classified projects (all fake). The attacker goes after it, thinking it's pivotal. Every action is logged, and when they attempt to dump credentials or exfiltrate data, silent alarms notify the defenders. Over a period of weeks, the threat actors continue to operate in the decoy, believing they have a foothold, while the defenders collect extensive intelligence on their tools (every malware sample, every command and IP address they use). The defenders can even feed misinformation back to the attackers (for example, fake documents that the adversary might incorporate into their reports). This use case demonstrates the decoy's value for **counter-intelligence**: it provides a controlled environment to monitor, study, and even deceive adversaries with minimal risk. Real-world analogues, such as Microsoft's use of fake Azure tenants to study phishing attackers, have proven the effectiveness of such techniques.

Use Case 3: Insider Threat Detection (Internal Security). Not all threats come from the outside; malicious insiders or compromised internal accounts pose serious risks. The decoy network can be deployed within an organization as a hidden sensor for insider activity. For example, the security team sets up a decoy fileshare or database that is made to look highly sensitive (e.g., labeled "Executive Salary Data" or "Acquisition Plans"), and makes it visible internally (perhaps mentioned on an internal wiki or directory). A normal employee would ignore it, but a malicious insider might attempt access out of curiosity or ill intent. When they do, they are interacting with the decoy environment. The system monitors their behavior: maybe they start running unusual queries or trying to siphon data. Because the decoy is isolated, even if the insider tries something destructive (like planting malware or creating new user accounts), it only affects the fake environment. Meanwhile, security is alerted in real time to this activity and can gather evidence of the insider's actions. The decoy can even adapt – for instance, once the insider is hooked, the RL agent might introduce a fake admin credential or additional secret data to see if the insider takes the bait, further incriminating themselves. This use case shows how deception technology can augment internal monitoring by catching suspicious behavior that slips past normal controls (an insider with legitimate access would not trigger an IDS by simply using their permissions, but they would reveal themselves by accessing a honeytoken resource). The advantage is that we catch the insider in the act within a contained environment, preventing real damage while gathering evidence of misconduct.

Use Case 4: Critical Infrastructure / ICS Protection. In industrial control system (ICS) or critical infrastructure settings (power grids, water treatment plants, etc.), running live experiments or allowing adversary interaction with real systems is obviously dangerous. A digital twin decoy network can mimic an ICS environment for such facilities. Using Azure Digital Twins, we can model not just IT components but also IoT sensors and controllers. For instance, a water treatment plant decoy might include virtual programmable logic controllers (PLCs), Human-Machine Interface (HMI) systems, and simulated sensor readings (tank levels, pressure, chemical rates) generated by a GAN trained on normal operational data. If a threat actor (like those behind Stuxnet or TRITON malware) targets the facility, they can be diverted into the decoy, which replicates the control network. The attacker might attempt destructive actions (e.g., changing a pump's operation or altering chemical injection levels). The decoy will respond as if those

commands are taking effect (the fake sensor data changes accordingly to show, say, chlorine levels rising), giving the attacker the impression of success. In reality, no physical process is impacted. The defenders, however, get to observe exactly what the attacker tried to do – what malware they deployed, what safety systems they tried to override – all without risk to the real plant. This provides invaluable insight into threats against critical infrastructure. After the engagement, the facility can improve its actual security (patching the exploited vulnerabilities, strengthening network segmentation, etc.) based on the decoy incident. This use case demonstrates **proactive defense for critical systems**: we create a full replica of an ICS environment as a battleground where we can safely engage nation-state actors and learn from them, protecting the real-world processes from harm.

These use cases highlight the versatility of the proposed decoy network. It can be tailored to enterprise IT, government networks, insider scenarios, or even industrial systems – any context where understanding and thwarting sophisticated threats is a priority. In each case, the decoy provides a safe environment to **detect, engage, and study adversaries** while shielding real assets from damage.

Phase I Work Plan and Phase II Goals

Phase I (Feasibility Study & Prototype Design): In Phase I, we will demonstrate the feasibility of the concept and develop a proof-of-concept decoy environment with key AI components. The main objectives include:

1. **Network Digital Twin Modeling:** Define a representative enterprise network segment (in collaboration with stakeholders) and build an initial Azure Digital Twins model capturing its assets and relationships as the decoy blueprint.

2. **Prototype Decoy Deployment:** Use IaC to deploy a small-scale decoy network on Azure (a handful of VMs/services). Implement preliminary AI components – e.g., train a basic GAN on sample traffic to generate network noise, and set up a rudimentary RL agent framework – to validate that AI-driven behavior can be injected and managed in the cloud environment.

3. **Initial Attack Simulation:** Simulate an attacker scenario (via a red team exercise or automated penetration test) against the prototype. Demonstrate that the decoy generates realistic background activity and that the attacker can be lured and observed. Evaluate the prototype's performance (realism of deception, data captured) to identify any glaring gaps.

4. **AI/ML Methodology Validation:** Conduct experiments to validate the chosen AI approaches. For example, train the GAN on available enterprise traffic data and assess realism, and run the RL agent in a simulated environment to ensure the formulation (state, actions, rewards) leads to reasonable strategies. Refine the design based on results.

5. **Reporting & Phase II Planning:** Deliver a feasibility report documenting the Phase I prototype, results of the attack simulation, and lessons learned. This report will include the refined system architecture and AI/ML design for the full system, and a detailed Phase II execution plan (tasks, timeline, risk mitigation).

By the end of Phase I, we will have a working prototype and detailed design that de-risks the key innovations (Azure Digital Twins integration and AI-driven deception), setting the stage for building the full system in Phase II.

Phase II (Full System Development & Demonstration): Phase II will build the complete decoy network platform and demonstrate it in a realistic environment (targeting TRL 6).

1. **Complete System Implementation:** Expand the digital twin model and IaC deployment to a full enterprise-scale decoy network (e.g., dozens of decoy hosts across multiple subnets). Fully implement all AI components: train robust GAN models on large, relevant datasets to cover diverse network behaviors, deploy the RL agent with a refined policy (incorporating Phase I findings), and integrate the user behavior simulation and threat intelligence pipeline.

2. **Integration and System Testing:** Integrate the AI/ML modules with the cloud environment and orchestration layer. Rigorously test end-to-end functionality – ensuring the decoy can deploy with one command, that background activities run correctly, that the RL agent can ingest telemetry and issue adaptive responses, and that the operator dashboard effectively displays events and allows control. Iteratively improve stability and performance through testing.

3. **Realism Refinement and Security Hardening:** Perform extensive red-team testing to uncover any signatures or patterns that might reveal the decoy. Tune the generative content and system configurations to eliminate those telltales. Apply security hardening in the Azure environment (lock down accounts, network paths, and ensure compliance with Air Force cybersecurity policies). Prepare documentation/artifacts needed for Authority to Operate (ATO) as applicable.

4. **Sandbox Demonstration (TRL-6):** Deploy the Phase II prototype in a realistic sandbox or test network provided by the Air Force. Conduct a live demonstration involving a sophisticated attack scenario. Show that defensive cyber operators can use the software to generate a decoy network on the fly and that the system successfully deceives and contains a simulated APT. We will demonstrate real-time tracking of the adversary and dynamic decoy adaptations, as well as the ability for operators to intervene (proving the human-in-the-loop control features). This validates the solution in an environment approximating operational conditions.

5. **Evaluation & Transition Preparation:** Evaluate the Phase II results by measuring key metrics (deployment time, attacker dwell time in decoy, number of TTPs captured, etc.) and collecting operator feedback on the tool's usability and effectiveness. Using these insights, refine the system as needed. Develop a transition roadmap for Phase III, outlining steps to deploy the technology in an operational Air Force network (including any integration with existing tools like SIEMs or SOAR platforms) and identifying potential early adopters in other Air Force units or DoD organizations. We will also outline commercialization plans for dual-use applications in the private sector (e.g., offering the decoy system as a managed security service for critical infrastructure companies).

By the end of Phase II, we will deliver a fully integrated **prototype software application** that meets the SBIR objectives: enabling operators to easily deploy realistic, dynamic decoy networks and effectively track and control adversary engagements in real time. This prototype will be demonstrated in a relevant environment, providing confidence in its operational value and paving the way for Phase III transition.

Strategic Impact and Future Applications

Adopting AI-driven digital twin decoy networks represents not just a new tool but a strategic shift in cyber defense posture for the Air Force and beyond. Some of the broader impacts include:

- **Proactive Defense and Deterrence:** This technology enables a move from reactive cybersecurity (responding to breaches after they're detected) to proactive engagement. By deploying decoy environments, defenders **take the initiative** – threat actors are not just detected, but actively drawn into a controlled battle space where the defenders hold the advantage. If state-sponsored hackers repeatedly waste time and expose secrets attacking elaborate decoys, it imposes a cost on them. Over time, widespread use of such deceptions can have a deterrent effect: adversaries become more cautious, slow down their operations, or even avoid targeting organizations known to employ advanced decoys. In military terms, it's like laying minefields and traps in cyberspace – attackers must advance carefully, reducing their momentum and increasing their chances of missteps.
- **Intelligence-Driven Security:** The threat intelligence gained from these decoy engagements is extremely valuable. We get direct insight into adversaries' tactics: their zero-day exploits, their command-and-control infrastructure, their target priorities. For Air Force cyber units, this means knowing the enemy's playbook in advance. Patterns observed in the decoy can inform the defense of real networks (e.g., if an attacker consistently goes after a certain type of system or uses a novel malware strain, the real network can be hardened accordingly). On a broader scale, this intel can be shared across the DoD and intelligence community. One organization's decoy "catch" can help protect others – akin to vaccine development by analyzing a virus in a lab environment. This collective benefit enhances national cyber defense. It also supports active counter-intelligence: by understanding who is attacking and what they're after, the DoD can coordinate responses or even feed disinformation back to adversaries to confuse their efforts.
- **Enhanced Resilience and Continuous Learning:** Deploying a digital twin decoy cultivates a culture of continuous security improvement. Every attack on the decoy is essentially a free training exercise for the defenders – it's an opportunity to learn and adapt without suffering damage. Organizations can use the decoy to test their own incident response processes (since they can practice on decoy incidents that feel real). The constant stream of threat data ensures defenses stay up-to-date against the latest tactics. Over the long term, this can significantly improve resilience, as the organization is always one step ahead, having seen tomorrow's attack techniques in their decoy yesterday.
- **Synergy with Zero Trust and Defense-in-Depth:** The decoy approach complements modern Zero Trust principles. Zero Trust operates under "assume breach" – and a decoy is a perfect component for the "assumed breach" scenario, giving an intruder a place to go that is not the real crown jewels. In a layered defense, the decoy can act as the last line (or an inner diversion) that catches attackers who somehow penetrated other layers. It adds an unpredictable element to the defense-in-depth strategy, something adversaries cannot easily account for in advance.
- **Psychological Impact on Adversaries:** Strategic use of deception introduces uncertainty for attackers. When high-end adversaries begin to suspect that some of their targets might actually be elaborate decoys, it forces them to question the reliability of the data they stole and the safety of

their footholds. This can slow their operations and sow doubt. Historically, deception has been a powerful tool in warfare (for example, the use of fake armies and misinformation in WWII); in cyber warfare, our solution provides that capability at network scale. There is also the opportunity for counter-offensive actions: if adversaries act on fake data from a decoy, it can be a means to expose their intentions or even lead them into traps (e.g., malware in fake documents that phone-home from the attacker's system, revealing their location).

- **Force Multiplication for Cyber Defenders:** Automation and AI in the decoy mean a small defensive team can handle many more incidents effectively. Instead of chasing ghosts in their real network, one analyst can monitor multiple adversary engagements in the decoy simultaneously, since the heavy lifting of response is largely automated. This scales the capacity of cyber defense teams and makes better use of skilled human operators. The return on investment could be substantial – preventing a single major breach can save millions, and our approach aims to prevent or mitigate multiple.
- **Dual-Use and Commercial Potential:** While the immediate application is Air Force networks facing nation-state threats, the technology has broad dual-use potential. Other DoD agencies, government networks, and even critical infrastructure industries (energy, finance, healthcare) are all targets of advanced cyber attacks and could benefit from autonomous decoy networks. In the commercial sector, large enterprises increasingly face APT attacks as well (e.g., espionage against tech companies). Our solution could be productized as a cybersecurity platform or cloud service for companies that need to protect high-value data. This aligns with Phase III goals of commercialization – the platform could be offered (with appropriate customization) to any organization looking to bolster its cyber defenses with active deception. By leading in this area, the Air Force can also shape best practices and standards for cyber deception operations, maintaining technological leadership in cybersecurity.

In summary, the proposed project not only promises a powerful tool for tactical cyber defense (catching attackers in the act), but also contributes to a strategic advantage. It changes the cost calculus for adversaries, yields continuous intelligence, and amplifies the effectiveness of cyber defenders. Successful implementation and deployment in the Air Force could serve as a model for wider adoption, ultimately raising the bar for cybersecurity across both the military and civilian domains.

Conclusion

We have proposed a Phase I/II effort to develop an **autonomous, AI/ML-driven digital twin decoy network** that aligns closely with the objectives of topic AF252-0005. By combining Azure-based digital twin modeling, cloud automation, and advanced AI techniques, our approach will enable the Air Force to automatically generate highly realistic decoy networks that can **deceive state-sponsored adversaries, monitor their activities in real time, and adaptively respond** to maintain the illusion. The system architecture and AI methodology have been detailed to show technical feasibility, and a clear plan is laid out to build and demonstrate the capability in a Phase II prototype.

In essence, this project will deliver a paradigm shift in cyber defense: instead of only hardening networks and waiting for alarms, we create an active **digital battleground** where we invite the adversary in, control the narrative, and turn their attack into a source of intelligence. The proposed decoy network will provide **maximum automation** in deployment and management, **unprecedented realism** through AI-driven behavior, thorough **adversary tracking** via integrated monitoring, and **real-time control** through autonomous agents and operator-in-the-loop options. This enables defensive cyber operators to engage threats on their terms.

By investing in this cutting-edge deception technology, the Air Force can significantly enhance its cyber resilience and gain insights into attacker tradecraft, all while safeguarding real operations. Success in this SBIR effort will pave the way for operational deployment (Phase III) and potential adoption across DoD and critical industries, marking an important step toward outpacing and outsmarting cyber adversaries.

Short bio for Jason Lind

Jason Lind is a seasoned software engineer and entrepreneur with over a decade of experience architecting and deploying large-scale, mission-critical systems for both commercial and defense customers. As the lead engineer for a U.S. Space Force application, he re-architected a legacy C# ASP.NET MVC system into a SignalR-enabled, PostgreSQL-backed solution that adheres to Department of Defense standards for real-time telemetry and data synchronization. His deep expertise in cloud-native architectures within the Microsoft Azure ecosystem has guided multiple projects—from Infrastructure-as-Code deployments using ARM/Bicep and Terraform to the design of complex Azure Digital Twins models for immersive, high-fidelity environments.

As co-founder of Groundbreaker Solutions LLC, Jason spearheads research into autonomous, AI-driven cyber-deception platforms. Under his direction, the team has developed a cloud-based digital twin decoy network that proactively engages advanced persistent threats, combining generative AI, reinforcement learning, and secure cloud orchestration. He also co-founded Courseware.coach, an AI-powered education platform built on ASP.NET Blazor, ReactiveUI, and Azure Cognitive Services, further demonstrating his ability to translate cutting-edge research into production-ready systems.

Known to colleagues as “Odyss3us,” Jason bridges strategic vision with hands-on development. His work in undersea vehicle communication research via fog computing and his leadership in StratML standardization—collaborating directly with DoD stakeholders—underscore a unique blend of technical acumen and strategic insight. With a proven track record of technical leadership, Jason is uniquely qualified to guide the design, deployment, and continuous improvement of high-fidelity digital twin decoy networks for cybersecurity operations.