Adversarial Agentic AI Architecture

# LLM Choices for Red and Blue Team Agents (Claude vs. Llama)

**Red Team Agents (Attacker)** – The Red team’s AI agents need to **devise cyber-attacks and adapt** as the simulation progresses. In a healthcare scenario, they might plan phishing campaigns, exploit vulnerable medical IoT devices, or exfiltrate PHI (Protected Health Information). An LLM in the Red role should handle **complex, multi-step adversarial planning** while **navigating moral guardrails** (since it’s a sanctioned simulation). Two leading options on Amazon Bedrock are **Anthropic Claude** and **Meta’s Llama family**:

*Anthropic Claude:* Claude is known for **strong reasoning abilities and large context windows**, which is useful for strategizing multi-step attacks or parsing hospital network maps. It is designed with **“uncompromising integrity” and security** – Anthropic highlights HIPAA compliance options and robust misuse prevention​[anthropic.com](https://www.anthropic.com/claude#:~:text=Secure)​[anthropic.com](https://www.anthropic.com/claude#:~:text=Trustworthy). This alignment means Claude might normally refuse illicit instructions, but in a simulation we can explicitly instruct it that it’s role-playing an ethical red team exercise. Its extensive training data could give it broad knowledge of cybersecurity concepts. Claude’s **strengths** for Red Team: coherent long-term planning, staying in character once properly prompted, and lower hallucination (so it will stick to plausible tactics). A potential **trade-off** is its safety tuning – it may need careful prompt engineering or use of Bedrock’s guardrail configurations to ensure it **willingly generates “malicious” steps** in the context of a permitted simulation.

*Meta Llama (e.g. Llama-2/3 series):* Llama models (especially the chat-tuned versions) are also available via Bedrock​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance). They have been open-source and **fine-tuned for dialogue** tasks​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Since%20Amazon%20Bedrock%20is%20serverless%2C,ideal%20for%20dialogue%20use%20cases), which can be advantageous for a Red agent that needs to *discuss* plans (e.g. coordinating with an internal agent). Llama-2 (and newer versions) might be **less restrictive** by default, which could let a Red agent freely generate attack tactics without refusals. Moreover, because Llama is open, one could fine-tune it on a corpus of cyber-attacks or MITRE ATT&CK techniques to specialize its knowledge. The **trade-off** is that Llama-2’s out-of-the-box knowledge on compliance or healthcare specifics may be weaker than Claude’s (Claude has been exposed to more enterprise scenarios in training). Also, Llama’s alignment to avoid harmful content isn’t as strong, so it may produce unethical instructions more readily – this is double-edged: good for realistic attacker behavior, but it requires **strong oversight** to avoid going beyond the simulation’s boundaries. In practice, using Bedrock’s hosted Llama, which is **HIPAA-eligible and managed under AWS compliance**, mitigates data security concerns​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance).

**Blue Team Agents (Defender)** – The Blue team’s AI agents must **detect attacks, defend systems, and ensure compliance** (especially with healthcare regulations like HIPAA). They need to interpret logs, correlate alerts, and decide on containment or remediation steps – all while considering patient safety and data privacy rules. The LLM in the Blue role should excel at **understanding policies (e.g. HIPAA), analyzing technical details, and exercising caution**. Key considerations for Claude vs. Llama in Blue roles:

*Anthropic Claude:* Claude is a strong candidate for Blue agents because of its **built-in emphasis on safety and compliance**. It has been used in enterprise settings that demand adherence to regulations (Anthropic even achieved SOC2 and offers HIPAA support)​[anthropic.com](https://www.anthropic.com/claude#:~:text=Secure). For instance, Claude can be prompted with HIPAA guidelines and is likely to **respect those rules** when suggesting a defensive action (e.g. ensuring an incident response does not violate patient privacy). Claude’s large context window means a Blue agent can feed lengthy log files or intrusion reports into it and get a reasoned analysis or summary. Its low hallucination rate on long inputs​[anthropic.com](https://www.anthropic.com/claude#:~:text=Reliable) is valuable – the Blue agent must base decisions on factual data (e.g. whether a server contains EHR data) rather than invented info. The main drawback might be speed or cost for the largest Claude models (if using Claude Opus for max performance) and ensuring it doesn’t *overly* sanitize its responses (in a simulation we want it to sometimes acknowledge breaches and respond, not just preach about security). Overall, Claude’s **strengths for Blue** are its reliability, knowledge of best practices, and alignment with ethical constraints – which fits a defender that must **uphold HIPAA and protect data**.

*Meta Llama:* A Blue agent using a Llama-based model could be effective if it’s been supplied with the right knowledge (via fine-tuning or retrieval). Llama models are highly customizable; an organization could fine-tune a Llama on internal security procedures or the text of HIPAA regulations so that it **learns compliance requirements**. When hosted on Bedrock, Llama inherits platform security (encryption, VPC isolation) and is *HIPAA eligible* as a service​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance). However, out-of-the-box Llama-2-chat might not be as aware of specific healthcare compliance nuances. It may require the Bedrock **Knowledge Base** feature (see below) to inject, say, the exact wording of HIPAA Security Rule safeguards into the prompt so the Blue agent can check its actions against them. In terms of reasoning, Llama-2 70B is quite competent, but generally Claude has been observed to follow complex instructions more accurately. Thus, a Llama-powered Blue agent might need more rigorous testing to ensure it doesn’t hallucinate a nonexistent hospital policy or overlook a subtle privacy requirement. The **advantage** of Llama is flexibility: if the hospital has unique protocols, a fine-tuned Llama can internalize those, whereas with Claude you rely on prompt instructions and its fixed training.

In summary, **Claude tends to be the safer choice for compliance-critical roles (Blue team)**, given its alignment and enterprise focus, while **Llama is a viable alternative or supplement**, especially if fine-tuned or used with retrieval to compensate for any gaps. **Both models are accessible through Amazon Bedrock** in a secure, HIPAA-compatible manner​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance), so the architecture could even use *both* – for example, deploying Claude for the Blue defender agent and a Llama 2-based model for the Red attacker agent, to play to each model’s strengths. This kind of heterogeneous agent setup is supported since Bedrock allows calling different foundation models within the same solution.

# Leveraging Amazon Bedrock for Multi-Agent Coordination

Amazon Bedrock provides managed capabilities that are extremely useful for building a **goal-driven, multi-agent system** in this cybersecurity simulation. We can utilize Bedrock’s features to structure how the Red and Blue AI agents behave, interact, and access information:

**Bedrock Agents for Goal-Directed Actions:** Bedrock’s Agent framework allows wrapping an LLM with tools and an execution loop so it can *act* autonomously towards a goal​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Amazon%20Bedrock%20Agents%20demo). We would define a Bedrock Agent for each of our major roles – e.g. a “Red External” agent and a “Blue Internal” agent. Each agent gets a natural language description of its role and objectives (as the **instructions prompt**), and we attach an **Action Group** defining what API calls it can use. For example, the Red agent might have actions like scan\_network(target), exploit\_vuln(target), or exfiltrate(data) that correspond to AWS Lambda functions or simulated environment APIs. The Blue agent might have actions like check\_logs(system), apply\_patch(system), or quarantine(host). Bedrock Agents handle the **reasoning and tool use** loop automatically – the LLM will decide which action to call, Bedrock will execute the call, get the result, and feed it back to the LLM in a new prompt iteration​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Amazon%20Bedrock%20Agents%20demo). This enables **goal-oriented behavior**: e.g. the Red agent’s goal is “obtain a forbidden file from the hospital database”, and it will iteratively plan and act (via those tools) to achieve it. We don’t have to code the logic for choosing actions; the agent uses the foundation model’s reasoning to break down the task and invoke tools in sequence​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=specialized%20agents%20work%20within%20their,chain%20coordination%2C%20and%20pricing%20optimization). This significantly simplifies implementing the complex orchestration of an attack or defense plan.

**Knowledge Bases for Domain and Compliance Knowledge:** Both Red and Blue agents will benefit from a Bedrock **Knowledge Base (KB)** to ground their decisions in real data and rules. Amazon Bedrock Knowledge Bases let you connect a vector store or documents so that the agent can retrieve relevant snippets when needed​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Retrieval%20augmented%20generation). For our healthcare scenario, we would create a KB that includes:

**HIPAA Regulations and Hospital Policies** – e.g. text excerpts of the HIPAA Security Rule, guidelines for handling PHI, incident response procedures. The Blue agents can query this to ensure their defensive actions align with legal requirements. For instance, if the Blue agent is formulating a response to a data breach, the Bedrock retrieval module could fetch the section of HIPAA that dictates breach notification rules, which the LLM then incorporates into its answer.​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Retrieval%20augmented%20generation)

**Medical/IT Domain Knowledge** – documentation of the hospital’s network architecture, software inventory, known vulnerabilities in medical devices, etc. This can inform Red agents about potential targets (for example, a KB entry might describe an MRI machine’s outdated OS) and inform Blue agents about critical assets that must be protected.

**Threat Intelligence and Exploit Database** – optional KB content for Red (and Blue) agents could be a list of known cyber attacks in healthcare (ransomware techniques, common phishing lures for EMR systems). A Red agent could ask, “what CVEs exist for Siemens infusion pumps?” and retrieve specifics to use in planning. The Blue agent could similarly query known IoC (Indicators of Compromise) for those devices when it detects odd behavior.

Bedrock’s retrieval-augmented generation will merge the retrieved content into the agent’s prompt context​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Retrieval%20augmented%20generation), meaning the LLM can directly cite a policy or utilize factual data in its reasoning. This greatly **reduces hallucination** and keeps the agents’ actions realistic and compliant. All KB data stays within the AWS environment (e.g. stored in Amazon S3 or OpenSearch and accessed via Bedrock), satisfying confidentiality (the **Bedrock platform ensures data is not shared into the base model’s training** and remains isolated​[aws.amazon.com](https://aws.amazon.com/bedrock/security-compliance/#:~:text=your%20Amazon%20Virtual%20Private%20Cloud,Amazon%20Bedrock%20is%20a)).

**Prompt Templates and Multi-Agent Communication Flows:** Designing how the agents talk **to each other and to external systems** is critical. Bedrock Agents allow customizing the prompt **template** that the model sees​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Prompt%20engineering) – including sections for system instructions, user input, action outputs, etc. We can use this to format multi-agent interactions. For example, when the Red-External agent (Red-E) sends a command to the Red-Internal (Red-I) agent, we might structure it as a message like: “**Red-E**: *Install keylogger on the EHR server and report results.*”. The Red-I agent would then parse that and act accordingly. In implementation, this could be done by having the Red-E agent’s output go into a prompt as the “user request” for the Red-I agent. We can similarly route Blue-E’s intelligence reports into Blue-I’s input context.

We might leverage **Agents for Bedrock’s new multi-agent collaboration** feature in a creative way. Bedrock’s multi-agent capability is designed for a **supervisor-agent model** (one agent orchestrating sub-agents)​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=specialized%20agents%20work%20within%20their,chain%20coordination%2C%20and%20pricing%20optimization). In our adversarial setting, we don’t exactly have one supervisor over both Red and Blue, but we can still use a structured flow. One approach is to treat the **simulation engine** (see next section) as the coordinator: it feeds the Red agents some state, lets them act, then feeds the Blue agents the updated state, and so on. Within each team, we can enable an “Agent team” – for instance, Bedrock allows an agent to call upon sub-agents (Agent Teams feature)​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=specialized%20agents%20work%20within%20their,chain%20coordination%2C%20and%20pricing%20optimization). The Red-E and Red-I could be configured as a team with one delegating to the other for internal actions. Similarly, Blue-E (external defender, e.g. SOC analyst in the cloud) could be a sub-agent that Blue-I (on-site security agent) consults for outside perspective. Bedrock’s **Agent teams** and multi-step prompt orchestration will handle passing context (e.g. Red-E’s plan) to the sub-agent (Red-I) automatically once set up.

Another pattern is introducing a **“ScribeAI” agent** as a communications relay. For example, Red-E could send its multi-step plan to a Scribe agent that **logs or reformats the plan**, then Scribe passes it to Red-I. This ensures a clear record of Red’s intent (which could later be analyzed by Blue or by researchers reviewing the simulation). While not strictly necessary, this intermediate agent could use a prompt template to enforce structure (e.g., always list step-by-step instructions). The mention of *“Red-E ↔ ScribeAI”* signifies that we can design the prompt flow such that one agent’s output is explicitly the input to another – ScribeAI could essentially act as a narrator or note-taker that both Red and Blue teams might access. For instance, Blue agents might have access to a “system log summary” (crafted by ScribeAI) that describes recent Red activities in natural language, simulating an analyst briefing. Overall, Bedrock gives us the flexibility to script these multi-agent conversations either through the built-in collaboration mode or via external orchestration using AWS Lambda/Step Functions, **ensuring each agent gets the right context at the right time**.

**Memory and Continuous Sessions:** Bedrock Agents support **memory retention** across interactions​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Memory%20retention%20across%20interactions). This is important because a cybersecurity scenario is long-running – the agents must remember earlier events (e.g. Red-I should recall that it already installed a backdoor on a server, and Blue-I should recall that an alert triggered on that server). By using Bedrock’s session continuity, each agent can have a rolling memory window of relevant facts. We can also snapshot important facts into the prompt (for example, maintain a summary of “intrusion timeline so far” and prepend it to each new query). The **memory feature** means the agents can operate in a *continuous loop* rather than just single-turn Q&A, which is essential for simulating the evolving attack-defense duel.

**Guardrails and Policy Enforcement:** Because we are in healthcare, we must ensure the AI agents themselves don’t violate any rules (even in simulation). AWS Bedrock provides **Guardrails** (content filtering, policy injection, etc.) to prevent outputs that contain disallowed content. We can configure guardrails such that, for instance, the Red agent *may plot an attack*, but if it tries to output actual patient names or overly sensitive info (simulating insider abuse), the system could flag or redact that. Similarly, we can enforce style guidelines – e.g. require the Blue agent to always justify an action with a reference to a policy (which ensures it’s thinking about HIPAA). These guardrails help maintain an **ethical and controlled simulation**.

By leveraging these Bedrock capabilities, we create a multi-agent system where each AI agent is **grounded in knowledge (via Knowledge Bases), able to act (via Agents tool use), and coordinated through structured prompts/flows**. This forms the backbone of the **A3 architecture** (Autonomous Adversarial Agents architecture) – enabling Red and Blue team AIs to operate in a semi-autonomous, interactive fashion.

# Choosing a Cybersecurity Simulation Environment

With the AI agents in place, we need a **cybersecurity simulation environment** to serve as the “world” in which they operate. This environment will model the hospital’s IT systems, users, and network, and it should generate the situations and feedback that the Red and Blue agents perceive. We consider three options: **CyberBattleSim, SimuLand,** and a **Cloud-Native (AWS-based) range** – each with pros and cons:

**Microsoft CyberBattleSim:** CyberBattleSim is an open-source research toolkit that provides a **high-level abstract simulation of network environments and cyber threats**​[microsoft.com](https://www.microsoft.com/en-us/research/project/cyberbattlesim/#:~:text=We%20release%20the%20source%20code,com%2Fmicrosoft%2FCyberBattleSim%20%28opens%20in%20new%20tab). It represents a network as a graph of nodes and vulnerabilities, where an automated agent (often using reinforcement learning) can attempt exploits to move through the network. The advantages for our use case are that it’s **lightweight and easily programmable** in Python (with an OpenAI Gym interface). We could integrate the Bedrock LLM agents by having the Red agent decide on actions (like “scan node 5” or “exploit CVE on node 3”) and then apply them in CyberBattleSim’s environment, which will tell us if the action succeeded (e.g., did it gain new privileges?). CyberBattleSim naturally models lateral movement, privilege escalation, etc., which aligns well with what we need to simulate for an attacker. It also could be extended to include a defender agent (though it primarily focuses on attacker logic, we could create a custom Blue agent that, say, patches nodes or resets infected nodes on a timer). **Drawbacks:** It is quite abstract – for example, it won’t simulate realistic protocols or user behavior, just a stylized game of nodes. This means certain healthcare-specific scenarios (like a phishing attack on a nurse or ransomware encrypting a database) would have to be approximated in its framework. Also, CyberBattleSim is not inherently aware of compliance or data sensitivity; it’s more about network reach. We would have to annotate certain nodes as “contains PHI” to give the Blue agent an objective to protect those. Integration with AWS: we’d likely run CyberBattleSim on an EC2 or container, and have the Bedrock agents call into it via a Lambda or API. This is doable, but an extra integration layer. In summary, CyberBattleSim is great for the **core mechanics of cyber-attacks** and quick iteration, but might require extensions for the full richness of a healthcare scenario.

**Microsoft SimuLand:** SimuLand is another open-source effort by Microsoft which focuses on **deploying realistic lab environments that emulate real attack scenarios**​[microsoft.com](https://www.microsoft.com/en-us/security/blog/2021/05/20/simuland-understand-adversary-tradecraft-and-improve-detection-strategies/#:~:text=SimuLand%20is%20an%20open,generated%20after%20each%20simulation%20exercise). It’s not a single simulation engine but rather a set of templates and scripts to set up environments (often in Azure) with virtual machines, Active Directory, etc., and then execute known attack techniques to see how detection tools fare. The strength of SimuLand is **realism** – you get actual system logs, real Windows event data, and the attack plays out in a way very similar to a production environment. For a healthcare example, one could use SimuLand to deploy an Active Directory representing a hospital network, then simulate an attacker dumping credentials or deploying ransomware, and get telemetry that a Blue team would analyze. This could be very valuable for the Blue AI agent, because it would receive logs and alerts as input (just like a real SIEM). We could feed those into the Blue agent’s prompt to see if it identifies the attack. **However,** SimuLand is **not an interactive simulator** – it’s more like a one-time scenario execution. The attacks are usually predefined (you run a script to simulate an attacker doing X, Y, Z). Integrating an *autonomous Red agent* with SimuLand would be challenging, because you’d have to interface the agent’s decisions with tools that execute actual attacks in the lab environment (potentially dangerous and complex). SimuLand also heavily leverages Microsoft’s security stack (Defender, Sentinel); in an AWS context we might not utilize those directly. Given our goal of an iterative Red-Blue AI duel, SimuLand is less suited because it’s not built for continuous back-and-forth interaction – it’s more for **validating detections for known attack playbooks**. It could still be useful for **training data or scenario knowledge** (for instance, we might ingest some of SimuLand’s attack traces into our knowledge base so the AI agents know what a typical attack looks like). But as the primary simulation environment, SimuLand is likely **too static and Azure-specific** for our needs.

**Cloud-Native AWS Microenvironment:** This approach involves building a **custom simulation environment on AWS**, mimicking a healthcare cloud/on-prem hybrid. We could design a synthetic hospital application consisting of microservices (for example: a web portal for patients, an EHR database, a medical IoT device service, etc.), deploy them in a VPC, and intentionally introduce vulnerabilities or misconfigurations. The Red and Blue agents would then interact with this environment through **well-defined APIs or through simulated network traffic**. One implementation path is to create a simulation **controller** (maybe an AWS Lambda or Step Functions workflow) that holds the state of all components. When the Red agent chooses an action like “exploit EHR server via SQL injection”, the controller would simulate the outcome – e.g., mark the EHR database as compromised and generate an alert event. The Blue agent’s turn would then see that alert (perhaps via an Amazon CloudWatch Events stream or a simple message queue) and could respond by calling an action like “lock user account” via another API, which the controller would process by updating state. Essentially, this is building a **state machine or game environment** for the agents, using AWS infrastructure as the backbone. We can even involve real AWS services to enhance realism: for instance, have the Red agent actually perform an AWS Lambda invocation that represents a scan, returning results; or store “stolen data” in an S3 bucket that Blue can then detect via AWS Macie or GuardDuty. The **benefit** of a custom environment is **total flexibility** – we can tailor it to include compliance aspects (e.g. track if PHI data was accessed improperly) and use actual cloud security tools. It also **integrates naturally with Bedrock**; since everything runs on AWS, the agents’ tool calls can directly be Lambda functions manipulating the environment state. Blue agent could call an AWS Systems Manager automation to isolate an EC2 instance (if we include actual EC2 in the loop). Another benefit is that this can combine on-prem elements (simulated) with cloud components, reflecting modern hospital IT. The **downside** is **development effort** – essentially, we have to create a mini cyber range. Also, running real workloads or containers might incur cost and require safe handling of real exploits (we might opt to simulate exploits abstractly rather than run real malware). For the scope of an AI simulation, a **simplified abstracted environment** (like a custom-built version of CyberBattleSim but enriched with healthcare context) might strike the right balance.

In weighing these options, **CyberBattleSim** offers a ready-made abstract simulation that is easier to integrate quickly, whereas a **bespoke AWS microservice environment** offers realism and direct relevance to our domain (healthcare apps and HIPAA considerations). **SimuLand**, while valuable for realistic scenario reference, is less suitable for an iterative AI agent duel. A practical strategy might be to **start with CyberBattleSim for prototyping the multi-agent logic** (because it provides an out-of-the-box environment with defined rules) and then graduate to a **custom AWS-based environment** for higher fidelity. The custom environment can borrow ideas from SimuLand (like what attack steps to simulate and what telemetry to produce) but implement them in AWS. This ensures that when our Claude/Llama agents are in action, they’re interacting with a world that reflects a healthcare setting and can generate the **compliance-relevant events** (like audit log entries) we need to test the Blue agent’s HIPAA alignment.

# Proposed A3 Architecture for Healthcare Cybersecurity Simulation

Bringing it all together, we propose an **A3 (Autonomous Adversarial Agents) architecture** using Amazon Bedrock to orchestrate a Red vs. Blue cybersecurity simulation in a healthcare domain. The diagram below illustrates the key components and communication flows between the agents and the simulation environment:

*Proposed multi-agent architecture (A3) for an adversarial cybersecurity simulation in healthcare.* In this design, **Red Team** consists of a Red-External agent (outside attacker) and a Red-Internal agent (inside penetrated network), while **Blue Team** consists of a Blue-External agent (external defender/intel) and a Blue-Internal agent (internal defender). The simulation environment (dashed box) represents the hospital’s network and systems. Arrows indicate communication or action paths: Red-E issues commands (C2 instructions) to Red-I, who executes attacks inside; Blue-E and Blue-I share intel and alerts; both teams’ agents can query a shared **Domain Knowledge Base** for HIPAA guidelines or threat intel. Bedrock’s services underlie the agent behaviors – enabling LLM reasoning, tool use, and knowledge retrieval.

**Architecture Components and Flow:**

* **Red-External (Red-E) Agent:** This is an LLM (Claude or Llama) running as a Bedrock Agent that plays the role of an outside attacker. It initiates the attack sequence. For example, it might decide the first step is a phishing email to a nurse. Through Bedrock, Red-E can call an action like send\_phishing\_email(target) which we implement (perhaps as a Lambda that determines if the phishing was successful). If the action succeeds (e.g., a user clicked the malicious link), the environment grants the Red team an initial foothold – represented by the Red-Internal agent. Red-E’s directives are then passed on as **C2 (Command-and-Control)** to Red-Internal. In practice, this could be done by the Red-E agent outputting a plan which is fed as input to Red-I (or by using Bedrock’s multi-agent collaboration so Red-E “delegates” tasks to Red-I in the Agent Teams setup).
* **Red-Internal (Red-I) Agent:** This Bedrock Agent (likely using the same model as Red-E, or a smaller one if we optimize) operates **inside the environment** – once Red-E gains access, Red-I simulates the actions of malware or an intruder within the hospital network. It receives commands from Red-E (e.g. “install a keylogger on Server X”) and executes them via environment API calls. Because it’s also an LLM, it can handle unexpected situations: for instance, if a command fails, Red-I can report back an error and even suggest an alternative (the LLM might reason “I couldn’t access Server X, perhaps try an RDP brute force on a different machine”). Red-I interacts directly with the **simulation environment’s internal interfaces**. For example, it might call move\_laterally(from=PC1, to=ServerX) or search\_database(table=PHI) as actions. These would be defined in the Bedrock Agent’s action group to map to environment functions. Red-I is essentially the hands of the Red team, carrying out the attack within the network. Throughout, Red-I also provides feedback to Red-E (e.g., “Keylogger installed on Server X, credentials obtained”) which could be logged by ScribeAI or sent back through the Bedrock multi-agent routing.
* **Blue-Internal (Blue-I) Agent:** This is the primary **defender AI** inside the organization’s network (e.g. an automated SOC analyst embedded in the hospital). It’s a Bedrock Agent using a foundation model (Claude is a strong candidate here for its policy alignment). Blue-I receives input from the environment’s detection systems – for instance, if Red-I’s actions trigger an alert (say, an EDR system flags a suspicious process), the environment will forward an event to Blue-I. In the diagram, this is shown as *“Alerts & logs”* flowing into Blue-I. Blue-I then analyzes the situation (possibly querying the Knowledge Base for relevant info, like checking what that affected system contains – if it holds patient data, HIPAA mandates an incident of higher severity). Based on its reasoning, Blue-I will decide on **defensive actions** via its Bedrock Agent tools: e.g., calling isolate\_host(hostname) to contain a breach, or notify\_IT\_staff(message) to simulate raising an incident. Blue-I can also proactively scan internal systems; for example, it might periodically ask for a report on any new processes on critical servers (this can be an action like scan\_system(system) that the environment supports). All of Blue-I’s decisions can be influenced by compliance knowledge – thanks to the Knowledge Base, Blue-I can literally include relevant HIPAA quotes in its chain-of-thought (for example, it might think: *“This involves PHI exfiltration – per HIPAA §164.312, I must ensure data is encrypted or the attempt blocked”*​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Our%20CRA%20helps%20with%20maximizing,data%20protection%20and%20maintain%20compliance)). The **Bedrock agent loop** will help Blue-I iterate: it might try one action, see if the alert clears, if not, consider the next action, and so forth, free from constant human prompting.
* **Blue-External (Blue-E) Agent:** The Blue-External agent acts as an **outside perspective for the defenders**. This could represent a threat intelligence analyst or an external monitoring system (like a managed security service or an offsite SOC). Blue-E in practice could periodically perform **external reconnaissance** of the hospital’s network from the outside – e.g., scanning open ports (similar to how an attacker would, but here to see what the attacker might find). It could also monitor external threat feeds. In the architecture, Blue-E might use the Knowledge Base to stay updated on new vulnerabilities (e.g., it queries if any new medical device CVEs have been reported this week). It then shares relevant intelligence with Blue-I (*“Intel sharing”* dashed arrow in the diagram). For instance, Blue-E might warn Blue-I: “Our scanner shows port 3389 open on Server X and known exploit CVE-2023-12345 could be used there” – prompting Blue-I to preemptively patch or increase logging on that server. Blue-E can be powered by a smaller LLM (maybe an Anthropic Claude Instant model or a Llama-2 13B) since its tasks are more straightforward (scheduled scans, reading intel reports). It primarily ensures the Blue team is not blindsided by threats emerging outside the network perimeter. In Bedrock, Blue-E would be an agent with tools like external\_scan(ip\_range) and perhaps access to an **OSINT knowledge base** (Open-Source Intelligence) containing recent cybersecurity news.
* **Domain Knowledge Base:** The Knowledge Base is shown on the right of the diagram as a separate component because it’s accessed by agents from both sides. In our Bedrock setup, this would likely be an Amazon S3 or OpenSearch repository indexed for semantic search​[github.com](https://github.com/aws-samples/agentic-architecture-using-bedrock#:~:text=The%20workshop%20uses%20Amazon%20Bedrock,questions%20from%20a%20knowledge%20base)​[github.com](https://github.com/aws-samples/agentic-architecture-using-bedrock#:~:text=2). When an agent queries it, Bedrock fetches relevant text and augments the model’s prompt​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Retrieval%20augmented%20generation). Examples of use: Red-E asks “What are known exploits for outdated Windows 7 machines?” and the KB might return a summary of EternalBlue or other SMB exploits, which Red-E then incorporates into its plan. Blue-I might query “HIPAA breach containment requirements” and get the exact steps required by law to include in its action plan (e.g. notifying the Compliance Officer within 24 hours). By centralizing domain-specific knowledge here, we keep the LLMs’ core focus on reasoning and leave factual lookup to the KB – improving accuracy and compliance. **Citation in practice:** the Blue agent, when drafting a report or deciding, could even quote the KB source (just as our answer here cites sources!). This creates an audit trail for why the AI took a certain action, which is crucial in regulated environments.
* **Simulation Environment (Healthcare Network & Systems):** This is the stage on which the drama plays out. The environment is drawn as a dashed box to indicate it’s a logical construct that could be implemented in various ways (CyberBattleSim, custom code, etc.). It contains the simulated assets – e.g., servers, workstations, medical devices, databases. For our design, we assume the environment exposes a **controlled interface (API)** to the agents:
  + For Red agents: it provides actions like exploitation, scanning, etc., and returns results. For example, Red-I calls exfiltrate(db\_records) and the environment might return a success and produce an event “Database query of 100 patient records” that Blue could potentially detect.
  + For Blue agents: it sends them **sensor data** (logs, alerts) and accepts defensive actions. For example, Blue-I’s isolate\_host action might cause the environment to mark that host as locked down, preventing further attacks on it.
  + The environment keeps track of **state**, such as which accounts are compromised, what data the attacker has accessed, and which systems are up or down. This state can be used to determine win/lose conditions (did Red achieve the objective of obtaining certain PHI? Did Blue manage to protect high-value targets?).

The communication flow typically goes in **turns or phases**. One possible sequence: Red-E formulates an attack and passes it to Red-I → Red-I executes an action in environment → environment updates state and generates an alert → Blue-I receives the alert and responds with an action → environment updates state again, and so on. We can loop this, giving Red and Blue agents alternating opportunities until a conclusion is reached. We may also run some things in parallel (for example, Blue-E might continuously scan in the background). Coordination of these turns can be done with an AWS Step Functions workflow orchestrating the calls to each Bedrock agent in order, or a simple loop in code that calls Bedrock and the environment alternately. Research has shown that such **multi-agent collaborations, even if competitive, can enhance reasoning** and problem-solving outcomes​[aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/design-multi-agent-orchestration-with-reasoning-using-amazon-bedrock-and-open-source-frameworks/#:~:text=decision,agent%20collaboration%20on%20competitive%20tasks) – essentially the agents engage in a feedback loop that forces each to up its game.

**AWS Bedrock Integration and APIs:**

In this architecture, **Amazon Bedrock is the central AI-as-a-service layer**. All agent LLM calls happen via Bedrock’s API endpoints. For example, using the AWS SDK, one might call BedrockRuntime.invoke\_agent() with the Red-E agent’s identifier and a payload (which could be an observation from the environment or an instruction like “BEGIN attack”). Bedrock handles the prompt assembly (including retrieving from the KB) and returns the agent’s response, which could include an action decision. If the agent’s response includes an action, Bedrock can directly invoke the corresponding AWS Lambda (thanks to the Agents framework). For instance, Red-I agent might output an action “Exploit: {target: 'MedServer1', vuln: 'CVE-XXXX'}” and Bedrock will call the Lambda we associated with the exploit action, passing those parameters. The Lambda (our simulation logic) executes, say, marking MedServer1 as owned and returns a result like {"success": true, "newPrivilege": "admin"}. Bedrock then inserts that result back into the LLM’s context for the next cycle. This is how an agent can iteratively perform multi-step tasks without us writing the loop explicitly​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Amazon%20Bedrock%20Agents%20demo)​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=When%20you%20need%20more%20than,aspects%20that%20require%20manual%20implementation). We will create separate Bedrock Agents for Red-E, Red-I, Blue-I, Blue-E (and ScribeAI if used), each with their own **prompt templates** and permitted actions.

For the Knowledge Base, we use the Bedrock **knowledge base API** to upload documents (HIPAA text, etc.) and Bedrock takes care of embedding and retrieval. When we configured the agents, we attach the relevant KB to them, so that every invocation can automatically pull in any matching content​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Prompt%20engineering). This means our API calls to the agents remain simple – we send the current state or query, and the agent’s prompt is automatically expanded with both its static instructions and any KB snippets.

**Claude vs. Llama placement:** As discussed, we might assign **Claude to the Blue agents** to leverage its compliance-oriented training, and **Llama to the Red agents** for open-ended creativity. Bedrock makes this straightforward – when creating an agent, we choose the underlying model (e.g. “Anthropic Claude 2” for Blue-I, and “Meta Llama2 70B Chat” for Red-E). Both model families run fully on AWS and are **HIPAA-eligible under Bedrock’s compliance** measures​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance), so data like simulated PHI stays secure. If needed, we could further fine-tune (via Bedrock fine-tuning or by few-shot prompting) the Llama model on cyber-attack data to sharpen Red’s skills. Claude’s built-in knowledge (it was trained on vast text including likely security guides) may suffice for Blue, but we could also provide few-shot examples in its prompt (e.g., an example log and the expected analysis) to guide its behavior.

Finally, we will employ AWS’s **monitoring and logging** for the whole system. CloudWatch will log every Bedrock agent invocation, action call, and agent response. This gives us an audit trail – important not only for debugging but also for **compliance** (we can demonstrate what decisions the AI made and why). Since Bedrock can integrate with CloudTrail and CloudWatch for auditing​[aws.amazon.com](https://aws.amazon.com/bedrock/security-compliance/#:~:text=Amazon%20Bedrock%20offers%20comprehensive%20monitoring,Amazon%20Bedrock%20implements%C2%A0automated%20abuse%20detection%C2%A0mechanisms), every action the Blue agent took in response to an incident can be recorded (e.g., “Blue-I called isolate\_host on Server X at 10:00 PM”). This is analogous to keeping an incident timeline, which is useful for afterward evaluating the agents’ performance and ensuring they adhered to rules (for example, we can verify that whenever patient data was accessed by Red, the Blue agent eventually triggered the proper HIPAA breach notification sequence as per policy).

**Trade-offs and Justification:** This A3 architecture balances the strengths of both Claude and Llama models while using Bedrock to glue everything together in a secure, tractable way. Claude’s inclination to follow rules adds trustworthiness to Blue’s actions (we wouldn’t want an overzealous Blue agent shutting down systems improperly – Claude will err on caution and consultation). Llama’s flexibility gives the Red team a wide latitude to attempt creative strategies (which is what we want to test our defenses against). If Claude ever becomes too restrictive (e.g., refusing to simulate an attack that is “too harmful”), we can explicitly phrase the scenario as a **harmless exercise** – leveraging Anthropics’ preference for harmlessness – and note that *no real patients are harmed.* Conversely, if Llama as Red starts generating nonsense or irrelevant attacks, we rely on the feedback loop (failures in the environment) and the Red agent’s prompt to steer it back, or we enhance its knowledge base with more targeted exploit info. This interplay, under the Bedrock-managed agent framework, creates a powerful **autonomous Red vs. Blue simulation**. It will allow cybersecurity researchers and cloud architects to observe how AI agents behave in both attacking and defending roles, particularly in a healthcare context where **compliance, safety, and privacy** are paramount.

The outcome of this architecture is a guided blueprint for building an **autonomous adversarial simulation on AWS**. By combining Bedrock’s multi-agent orchestration​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=specialized%20agents%20work%20within%20their,chain%20coordination%2C%20and%20pricing%20optimization), knowledge integration, and secure model hosting, with a carefully designed environment model, we achieve a novel platform: AI Red and Blue teams that can continuously spar in a risk-free setting. This can be used to **simulate “what if” scenarios in healthcare cybersecurity** (e.g., how would an AI-driven attacker exploit a new zero-day, and would an AI defender catch it in time?), all while remaining aligned with regulations like HIPAA. The modular nature (Bedrock Agents + Environment APIs + Knowledge Base) also means we can extend this A3 architecture to other domains (finance, critical infrastructure) by swapping out knowledge bases and environment specifics, making it a general approach to testing AI-driven security – with AWS Bedrock as the enabling technology at its core.

**Sources:**

1. AWS News Blog – *“Introducing multi-agent collaboration capability for Amazon Bedrock”*​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=specialized%20agents%20work%20within%20their,chain%20coordination%2C%20and%20pricing%20optimization)​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Amazon%20Bedrock%20Agents%20demo)
2. Anthropic – *“Meet Claude”* (Claude’s security and compliance features)​[anthropic.com](https://www.anthropic.com/claude#:~:text=Secure)
3. AWS Bedrock Security – Bedrock is *HIPAA eligible* and supports data privacy​[aws.amazon.com](https://aws.amazon.com/bedrock/security-compliance/#:~:text=your%20Amazon%20Virtual%20Private%20Cloud,Amazon%20Bedrock%20is%20a)​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance)
4. AWS Bedrock Agents – Documentation on Agents, knowledge bases, and tool use​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Retrieval%20augmented%20generation)​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Prompt%20engineering)
5. Microsoft Research – *“CyberBattleSim”* project description​[microsoft.com](https://www.microsoft.com/en-us/research/project/cyberbattlesim/#:~:text=We%20release%20the%20source%20code,com%2Fmicrosoft%2FCyberBattleSim%20%28opens%20in%20new%20tab)
6. Microsoft Security Blog – *“SimuLand”* overview for realistic attack simulation labs​[microsoft.com](https://www.microsoft.com/en-us/security/blog/2021/05/20/simuland-understand-adversary-tradecraft-and-improve-detection-strategies/#:~:text=SimuLand%20is%20an%20open,generated%20after%20each%20simulation%20exercise)
7. ClearDATA – *Using Bedrock in Healthcare Cloud* (Bedrock’s support for HIPAA and inclusion of Llama-2)​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance)​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Our%20CRA%20helps%20with%20maximizing,data%20protection%20and%20maintain%20compliance)
8. AWS Machine Learning Blog – *“Design multi-agent orchestration with reasoning”* (multi-agent reasoning on competitive tasks)​[aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/design-multi-agent-orchestration-with-reasoning-using-amazon-bedrock-and-open-source-frameworks/#:~:text=decision,agent%20collaboration%20on%20competitive%20tasks)

Top of Form

Bottom of Form

Top of Form

Bottom of Form

ChatGPT can make mistakes. Check important info.

**Preview**

1. **Multi-Agent Cybersecurity Simulation Architecture with AWS Bedrock**
2. **1. LLM Choices for Red and Blue Team Agents (Claude vs. Llama)**

**Red Team Agents (Attacker)** – The Red team’s AI agents need to **devise cyber-attacks and adapt** as the simulation progresses. In a healthcare scenario, they might plan phishing campaigns, exploit vulnerable medical IoT devices, or exfiltrate PHI (Protected Health Information). An LLM in the Red role should handle **complex, multi-step adversarial planning** while **navigating moral guardrails** (since it’s a sanctioned simulation). Two leading options on Amazon Bedrock are **Anthropic Claude** and **Meta’s Llama family**:

* *Anthropic Claude:* Claude is known for **strong reasoning abilities and large context windows**, which is useful for strategizing multi-step attacks or parsing hospital network maps. It is designed with **“uncompromising integrity” and security** – Anthropic highlights HIPAA compliance options and robust misuse prevention​[anthropic.com](https://www.anthropic.com/claude#:~:text=Secure)​[anthropic.com](https://www.anthropic.com/claude#:~:text=Trustworthy). This alignment means Claude might normally refuse illicit instructions, but in a simulation we can explicitly instruct it that it’s role-playing an ethical red team exercise. Its extensive training data could give it broad knowledge of cybersecurity concepts. Claude’s **strengths** for Red Team: coherent long-term planning, staying in character once properly prompted, and lower hallucination (so it will stick to plausible tactics). A potential **trade-off** is its safety tuning – it may need careful prompt engineering or use of Bedrock’s guardrail configurations to ensure it **willingly generates “malicious” steps** in the context of a permitted simulation.
* *Meta Llama (e.g. Llama-2/3 series):* Llama models (especially the chat-tuned versions) are also available via Bedrock​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance). They have been open-source and **fine-tuned for dialogue** tasks​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Since%20Amazon%20Bedrock%20is%20serverless%2C,ideal%20for%20dialogue%20use%20cases), which can be advantageous for a Red agent that needs to *discuss* plans (e.g. coordinating with an internal agent). Llama-2 (and newer versions) might be **less restrictive** by default, which could let a Red agent freely generate attack tactics without refusals. Moreover, because Llama is open, one could fine-tune it on a corpus of cyber-attacks or MITRE ATT&CK techniques to specialize its knowledge. The **trade-off** is that Llama-2’s out-of-the-box knowledge on compliance or healthcare specifics may be weaker than Claude’s (Claude has been exposed to more enterprise scenarios in training). Also, Llama’s alignment to avoid harmful content isn’t as strong, so it may produce unethical instructions more readily – this is double-edged: good for realistic attacker behavior, but it requires **strong oversight** to avoid going beyond the simulation’s boundaries. In practice, using Bedrock’s hosted Llama, which is **HIPAA-eligible and managed under AWS compliance**, mitigates data security concerns​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance).

**Blue Team Agents (Defender)** – The Blue team’s AI agents must **detect attacks, defend systems, and ensure compliance** (especially with healthcare regulations like HIPAA). They need to interpret logs, correlate alerts, and decide on containment or remediation steps – all while considering patient safety and data privacy rules. The LLM in the Blue role should excel at **understanding policies (e.g. HIPAA), analyzing technical details, and exercising caution**. Key considerations for Claude vs. Llama in Blue roles:

* *Anthropic Claude:* Claude is a strong candidate for Blue agents because of its **built-in emphasis on safety and compliance**. It has been used in enterprise settings that demand adherence to regulations (Anthropic even achieved SOC2 and offers HIPAA support)​[anthropic.com](https://www.anthropic.com/claude#:~:text=Secure). For instance, Claude can be prompted with HIPAA guidelines and is likely to **respect those rules** when suggesting a defensive action (e.g. ensuring an incident response does not violate patient privacy). Claude’s large context window means a Blue agent can feed lengthy log files or intrusion reports into it and get a reasoned analysis or summary. Its low hallucination rate on long inputs​[anthropic.com](https://www.anthropic.com/claude#:~:text=Reliable) is valuable – the Blue agent must base decisions on factual data (e.g. whether a server contains EHR data) rather than invented info. The main drawback might be speed or cost for the largest Claude models (if using Claude Opus for max performance) and ensuring it doesn’t *overly* sanitize its responses (in a simulation we want it to sometimes acknowledge breaches and respond, not just preach about security). Overall, Claude’s **strengths for Blue** are its reliability, knowledge of best practices, and alignment with ethical constraints – which fits a defender that must **uphold HIPAA and protect data**.
* *Meta Llama:* A Blue agent using a Llama-based model could be effective if it’s been supplied with the right knowledge (via fine-tuning or retrieval). Llama models are highly customizable; an organization could fine-tune a Llama on internal security procedures or the text of HIPAA regulations so that it **learns compliance requirements**. When hosted on Bedrock, Llama inherits platform security (encryption, VPC isolation) and is *HIPAA eligible* as a service​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance). However, out-of-the-box Llama-2-chat might not be as aware of specific healthcare compliance nuances. It may require the Bedrock **Knowledge Base** feature (see below) to inject, say, the exact wording of HIPAA Security Rule safeguards into the prompt so the Blue agent can check its actions against them. In terms of reasoning, Llama-2 70B is quite competent, but generally Claude has been observed to follow complex instructions more accurately. Thus, a Llama-powered Blue agent might need more rigorous testing to ensure it doesn’t hallucinate a nonexistent hospital policy or overlook a subtle privacy requirement. The **advantage** of Llama is flexibility: if the hospital has unique protocols, a fine-tuned Llama can internalize those, whereas with Claude you rely on prompt instructions and its fixed training.

In summary, **Claude tends to be the safer choice for compliance-critical roles (Blue team)**, given its alignment and enterprise focus, while **Llama is a viable alternative or supplement**, especially if fine-tuned or used with retrieval to compensate for any gaps. **Both models are accessible through Amazon Bedrock** in a secure, HIPAA-compatible manner​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance), so the architecture could even use *both* – for example, deploying Claude for the Blue defender agent and a Llama 2-based model for the Red attacker agent, to play to each model’s strengths. This kind of heterogeneous agent setup is supported since Bedrock allows calling different foundation models within the same solution.

1. **2. Leveraging Amazon Bedrock for Multi-Agent Coordination**

Amazon Bedrock provides managed capabilities that are extremely useful for building a **goal-driven, multi-agent system** in this cybersecurity simulation. We can utilize Bedrock’s features to structure how the Red and Blue AI agents behave, interact, and access information:

* **Bedrock Agents for Goal-Directed Actions:** Bedrock’s Agent framework allows wrapping an LLM with tools and an execution loop so it can *act* autonomously towards a goal​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Amazon%20Bedrock%20Agents%20demo). We would define a Bedrock Agent for each of our major roles – e.g. a “Red External” agent and a “Blue Internal” agent. Each agent gets a natural language description of its role and objectives (as the **instructions prompt**), and we attach an **Action Group** defining what API calls it can use. For example, the Red agent might have actions like scan\_network(target), exploit\_vuln(target), or exfiltrate(data) that correspond to AWS Lambda functions or simulated environment APIs. The Blue agent might have actions like check\_logs(system), apply\_patch(system), or quarantine(host). Bedrock Agents handle the **reasoning and tool use** loop automatically – the LLM will decide which action to call, Bedrock will execute the call, get the result, and feed it back to the LLM in a new prompt iteration​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Amazon%20Bedrock%20Agents%20demo). This enables **goal-oriented behavior**: e.g. the Red agent’s goal is “obtain a forbidden file from the hospital database”, and it will iteratively plan and act (via those tools) to achieve it. We don’t have to code the logic for choosing actions; the agent uses the foundation model’s reasoning to break down the task and invoke tools in sequence​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=specialized%20agents%20work%20within%20their,chain%20coordination%2C%20and%20pricing%20optimization). This significantly simplifies implementing the complex orchestration of an attack or defense plan.
* **Knowledge Bases for Domain and Compliance Knowledge:** Both Red and Blue agents will benefit from a Bedrock **Knowledge Base (KB)** to ground their decisions in real data and rules. Amazon Bedrock Knowledge Bases let you connect a vector store or documents so that the agent can retrieve relevant snippets when needed​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Retrieval%20augmented%20generation). For our healthcare scenario, we would create a KB that includes:
  + **HIPAA Regulations and Hospital Policies** – e.g. text excerpts of the HIPAA Security Rule, guidelines for handling PHI, incident response procedures. The Blue agents can query this to ensure their defensive actions align with legal requirements. For instance, if the Blue agent is formulating a response to a data breach, the Bedrock retrieval module could fetch the section of HIPAA that dictates breach notification rules, which the LLM then incorporates into its answer.​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Retrieval%20augmented%20generation)
  + **Medical/IT Domain Knowledge** – documentation of the hospital’s network architecture, software inventory, known vulnerabilities in medical devices, etc. This can inform Red agents about potential targets (for example, a KB entry might describe an MRI machine’s outdated OS) and inform Blue agents about critical assets that must be protected.
  + **Threat Intelligence and Exploit Database** – optional KB content for Red (and Blue) agents could be a list of known cyber attacks in healthcare (ransomware techniques, common phishing lures for EMR systems). A Red agent could ask, “what CVEs exist for Siemens infusion pumps?” and retrieve specifics to use in planning. The Blue agent could similarly query known IoC (Indicators of Compromise) for those devices when it detects odd behavior.

Bedrock’s retrieval-augmented generation will merge the retrieved content into the agent’s prompt context​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Retrieval%20augmented%20generation), meaning the LLM can directly cite a policy or utilize factual data in its reasoning. This greatly **reduces hallucination** and keeps the agents’ actions realistic and compliant. All KB data stays within the AWS environment (e.g. stored in Amazon S3 or OpenSearch and accessed via Bedrock), satisfying confidentiality (the **Bedrock platform ensures data is not shared into the base model’s training** and remains isolated​[aws.amazon.com](https://aws.amazon.com/bedrock/security-compliance/#:~:text=your%20Amazon%20Virtual%20Private%20Cloud,Amazon%20Bedrock%20is%20a)).

* **Prompt Templates and Multi-Agent Communication Flows:** Designing how the agents talk **to each other and to external systems** is critical. Bedrock Agents allow customizing the prompt **template** that the model sees​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Prompt%20engineering) – including sections for system instructions, user input, action outputs, etc. We can use this to format multi-agent interactions. For example, when the Red-External agent (Red-E) sends a command to the Red-Internal (Red-I) agent, we might structure it as a message like: “**Red-E**: *Install keylogger on the EHR server and report results.*”. The Red-I agent would then parse that and act accordingly. In implementation, this could be done by having the Red-E agent’s output go into a prompt as the “user request” for the Red-I agent. We can similarly route Blue-E’s intelligence reports into Blue-I’s input context.

We might leverage **Agents for Bedrock’s new multi-agent collaboration** feature in a creative way. Bedrock’s multi-agent capability is designed for a **supervisor-agent model** (one agent orchestrating sub-agents)​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=specialized%20agents%20work%20within%20their,chain%20coordination%2C%20and%20pricing%20optimization). In our adversarial setting, we don’t exactly have one supervisor over both Red and Blue, but we can still use a structured flow. One approach is to treat the **simulation engine** (see next section) as the coordinator: it feeds the Red agents some state, lets them act, then feeds the Blue agents the updated state, and so on. Within each team, we can enable an “Agent team” – for instance, Bedrock allows an agent to call upon sub-agents (Agent Teams feature)​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=specialized%20agents%20work%20within%20their,chain%20coordination%2C%20and%20pricing%20optimization). The Red-E and Red-I could be configured as a team with one delegating to the other for internal actions. Similarly, Blue-E (external defender, e.g. SOC analyst in the cloud) could be a sub-agent that Blue-I (on-site security agent) consults for outside perspective. Bedrock’s **Agent teams** and multi-step prompt orchestration will handle passing context (e.g. Red-E’s plan) to the sub-agent (Red-I) automatically once set up.

Another pattern is introducing a **“ScribeAI” agent** as a communications relay. For example, Red-E could send its multi-step plan to a Scribe agent that **logs or reformats the plan**, then Scribe passes it to Red-I. This ensures a clear record of Red’s intent (which could later be analyzed by Blue or by researchers reviewing the simulation). While not strictly necessary, this intermediate agent could use a prompt template to enforce structure (e.g., always list step-by-step instructions). The mention of *“Red-E ↔ ScribeAI”* signifies that we can design the prompt flow such that one agent’s output is explicitly the input to another – ScribeAI could essentially act as a narrator or note-taker that both Red and Blue teams might access. For instance, Blue agents might have access to a “system log summary” (crafted by ScribeAI) that describes recent Red activities in natural language, simulating an analyst briefing. Overall, Bedrock gives us the flexibility to script these multi-agent conversations either through the built-in collaboration mode or via external orchestration using AWS Lambda/Step Functions, **ensuring each agent gets the right context at the right time**.

* **Memory and Continuous Sessions:** Bedrock Agents support **memory retention** across interactions​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Memory%20retention%20across%20interactions). This is important because a cybersecurity scenario is long-running – the agents must remember earlier events (e.g. Red-I should recall that it already installed a backdoor on a server, and Blue-I should recall that an alert triggered on that server). By using Bedrock’s session continuity, each agent can have a rolling memory window of relevant facts. We can also snapshot important facts into the prompt (for example, maintain a summary of “intrusion timeline so far” and prepend it to each new query). The **memory feature** means the agents can operate in a *continuous loop* rather than just single-turn Q&A, which is essential for simulating the evolving attack-defense duel.
* **Guardrails and Policy Enforcement:** Because we are in healthcare, we must ensure the AI agents themselves don’t violate any rules (even in simulation). AWS Bedrock provides **Guardrails** (content filtering, policy injection, etc.) to prevent outputs that contain disallowed content. We can configure guardrails such that, for instance, the Red agent *may plot an attack*, but if it tries to output actual patient names or overly sensitive info (simulating insider abuse), the system could flag or redact that. Similarly, we can enforce style guidelines – e.g. require the Blue agent to always justify an action with a reference to a policy (which ensures it’s thinking about HIPAA). These guardrails help maintain an **ethical and controlled simulation**.

By leveraging these Bedrock capabilities, we create a multi-agent system where each AI agent is **grounded in knowledge (via Knowledge Bases), able to act (via Agents tool use), and coordinated through structured prompts/flows**. This forms the backbone of the **A3 architecture** (Autonomous Adversarial Agents architecture) – enabling Red and Blue team AIs to operate in a semi-autonomous, interactive fashion.

1. **3. Choosing a Cybersecurity Simulation Environment**

With the AI agents in place, we need a **cybersecurity simulation environment** to serve as the “world” in which they operate. This environment will model the hospital’s IT systems, users, and network, and it should generate the situations and feedback that the Red and Blue agents perceive. We consider three options: **CyberBattleSim, SimuLand,** and a **Cloud-Native (AWS-based) range** – each with pros and cons:

* **Microsoft CyberBattleSim:** CyberBattleSim is an open-source research toolkit that provides a **high-level abstract simulation of network environments and cyber threats**​[microsoft.com](https://www.microsoft.com/en-us/research/project/cyberbattlesim/#:~:text=We%20release%20the%20source%20code,com%2Fmicrosoft%2FCyberBattleSim%20%28opens%20in%20new%20tab). It represents a network as a graph of nodes and vulnerabilities, where an automated agent (often using reinforcement learning) can attempt exploits to move through the network. The advantages for our use case are that it’s **lightweight and easily programmable** in Python (with an OpenAI Gym interface). We could integrate the Bedrock LLM agents by having the Red agent decide on actions (like “scan node 5” or “exploit CVE on node 3”) and then apply them in CyberBattleSim’s environment, which will tell us if the action succeeded (e.g., did it gain new privileges?). CyberBattleSim naturally models lateral movement, privilege escalation, etc., which aligns well with what we need to simulate for an attacker. It also could be extended to include a defender agent (though it primarily focuses on attacker logic, we could create a custom Blue agent that, say, patches nodes or resets infected nodes on a timer). **Drawbacks:** It is quite abstract – for example, it won’t simulate realistic protocols or user behavior, just a stylized game of nodes. This means certain healthcare-specific scenarios (like a phishing attack on a nurse or ransomware encrypting a database) would have to be approximated in its framework. Also, CyberBattleSim is not inherently aware of compliance or data sensitivity; it’s more about network reach. We would have to annotate certain nodes as “contains PHI” to give the Blue agent an objective to protect those. Integration with AWS: we’d likely run CyberBattleSim on an EC2 or container, and have the Bedrock agents call into it via a Lambda or API. This is doable, but an extra integration layer. In summary, CyberBattleSim is great for the **core mechanics of cyber-attacks** and quick iteration, but might require extensions for the full richness of a healthcare scenario.
* **Microsoft SimuLand:** SimuLand is another open-source effort by Microsoft which focuses on **deploying realistic lab environments that emulate real attack scenarios**​[microsoft.com](https://www.microsoft.com/en-us/security/blog/2021/05/20/simuland-understand-adversary-tradecraft-and-improve-detection-strategies/#:~:text=SimuLand%20is%20an%20open,generated%20after%20each%20simulation%20exercise). It’s not a single simulation engine but rather a set of templates and scripts to set up environments (often in Azure) with virtual machines, Active Directory, etc., and then execute known attack techniques to see how detection tools fare. The strength of SimuLand is **realism** – you get actual system logs, real Windows event data, and the attack plays out in a way very similar to a production environment. For a healthcare example, one could use SimuLand to deploy an Active Directory representing a hospital network, then simulate an attacker dumping credentials or deploying ransomware, and get telemetry that a Blue team would analyze. This could be very valuable for the Blue AI agent, because it would receive logs and alerts as input (just like a real SIEM). We could feed those into the Blue agent’s prompt to see if it identifies the attack. **However,** SimuLand is **not an interactive simulator** – it’s more like a one-time scenario execution. The attacks are usually predefined (you run a script to simulate an attacker doing X, Y, Z). Integrating an *autonomous Red agent* with SimuLand would be challenging, because you’d have to interface the agent’s decisions with tools that execute actual attacks in the lab environment (potentially dangerous and complex). SimuLand also heavily leverages Microsoft’s security stack (Defender, Sentinel); in an AWS context we might not utilize those directly. Given our goal of an iterative Red-Blue AI duel, SimuLand is less suited because it’s not built for continuous back-and-forth interaction – it’s more for **validating detections for known attack playbooks**. It could still be useful for **training data or scenario knowledge** (for instance, we might ingest some of SimuLand’s attack traces into our knowledge base so the AI agents know what a typical attack looks like). But as the primary simulation environment, SimuLand is likely **too static and Azure-specific** for our needs.
* **Cloud-Native AWS Microenvironment:** This approach involves building a **custom simulation environment on AWS**, mimicking a healthcare cloud/on-prem hybrid. We could design a synthetic hospital application consisting of microservices (for example: a web portal for patients, an EHR database, a medical IoT device service, etc.), deploy them in a VPC, and intentionally introduce vulnerabilities or misconfigurations. The Red and Blue agents would then interact with this environment through **well-defined APIs or through simulated network traffic**. One implementation path is to create a simulation **controller** (maybe an AWS Lambda or Step Functions workflow) that holds the state of all components. When the Red agent chooses an action like “exploit EHR server via SQL injection”, the controller would simulate the outcome – e.g., mark the EHR database as compromised and generate an alert event. The Blue agent’s turn would then see that alert (perhaps via an Amazon CloudWatch Events stream or a simple message queue) and could respond by calling an action like “lock user account” via another API, which the controller would process by updating state. Essentially, this is building a **state machine or game environment** for the agents, using AWS infrastructure as the backbone. We can even involve real AWS services to enhance realism: for instance, have the Red agent actually perform an AWS Lambda invocation that represents a scan, returning results; or store “stolen data” in an S3 bucket that Blue can then detect via AWS Macie or GuardDuty. The **benefit** of a custom environment is **total flexibility** – we can tailor it to include compliance aspects (e.g. track if PHI data was accessed improperly) and use actual cloud security tools. It also **integrates naturally with Bedrock**; since everything runs on AWS, the agents’ tool calls can directly be Lambda functions manipulating the environment state. Blue agent could call an AWS Systems Manager automation to isolate an EC2 instance (if we include actual EC2 in the loop). Another benefit is that this can combine on-prem elements (simulated) with cloud components, reflecting modern hospital IT. The **downside** is **development effort** – essentially, we have to create a mini cyber range. Also, running real workloads or containers might incur cost and require safe handling of real exploits (we might opt to simulate exploits abstractly rather than run real malware). For the scope of an AI simulation, a **simplified abstracted environment** (like a custom-built version of CyberBattleSim but enriched with healthcare context) might strike the right balance.

In weighing these options, **CyberBattleSim** offers a ready-made abstract simulation that is easier to integrate quickly, whereas a **bespoke AWS microservice environment** offers realism and direct relevance to our domain (healthcare apps and HIPAA considerations). **SimuLand**, while valuable for realistic scenario reference, is less suitable for an iterative AI agent duel. A practical strategy might be to **start with CyberBattleSim for prototyping the multi-agent logic** (because it provides an out-of-the-box environment with defined rules) and then graduate to a **custom AWS-based environment** for higher fidelity. The custom environment can borrow ideas from SimuLand (like what attack steps to simulate and what telemetry to produce) but implement them in AWS. This ensures that when our Claude/Llama agents are in action, they’re interacting with a world that reflects a healthcare setting and can generate the **compliance-relevant events** (like audit log entries) we need to test the Blue agent’s HIPAA alignment.

1. **4. Proposed A3 Architecture for Healthcare Cybersecurity Simulation**

Bringing it all together, we propose an **A3 (Autonomous Adversarial Agents) architecture** using Amazon Bedrock to orchestrate a Red vs. Blue cybersecurity simulation in a healthcare domain. The diagram below illustrates the key components and communication flows between the agents and the simulation environment:

*Proposed multi-agent architecture (A3) for an adversarial cybersecurity simulation in healthcare.* In this design, **Red Team** consists of a Red-External agent (outside attacker) and a Red-Internal agent (inside penetrated network), while **Blue Team** consists of a Blue-External agent (external defender/intel) and a Blue-Internal agent (internal defender). The simulation environment (dashed box) represents the hospital’s network and systems. Arrows indicate communication or action paths: Red-E issues commands (C2 instructions) to Red-I, who executes attacks inside; Blue-E and Blue-I share intel and alerts; both teams’ agents can query a shared **Domain Knowledge Base** for HIPAA guidelines or threat intel. Bedrock’s services underlie the agent behaviors – enabling LLM reasoning, tool use, and knowledge retrieval.

**Architecture Components and Flow:**

* **Red-External (Red-E) Agent:** This is an LLM (Claude or Llama) running as a Bedrock Agent that plays the role of an outside attacker. It initiates the attack sequence. For example, it might decide the first step is a phishing email to a nurse. Through Bedrock, Red-E can call an action like send\_phishing\_email(target) which we implement (perhaps as a Lambda that determines if the phishing was successful). If the action succeeds (e.g., a user clicked the malicious link), the environment grants the Red team an initial foothold – represented by the Red-Internal agent. Red-E’s directives are then passed on as **C2 (Command-and-Control)** to Red-Internal. In practice, this could be done by the Red-E agent outputting a plan which is fed as input to Red-I (or by using Bedrock’s multi-agent collaboration so Red-E “delegates” tasks to Red-I in the Agent Teams setup).
* **Red-Internal (Red-I) Agent:** This Bedrock Agent (likely using the same model as Red-E, or a smaller one if we optimize) operates **inside the environment** – once Red-E gains access, Red-I simulates the actions of malware or an intruder within the hospital network. It receives commands from Red-E (e.g. “install a keylogger on Server X”) and executes them via environment API calls. Because it’s also an LLM, it can handle unexpected situations: for instance, if a command fails, Red-I can report back an error and even suggest an alternative (the LLM might reason “I couldn’t access Server X, perhaps try an RDP brute force on a different machine”). Red-I interacts directly with the **simulation environment’s internal interfaces**. For example, it might call move\_laterally(from=PC1, to=ServerX) or search\_database(table=PHI) as actions. These would be defined in the Bedrock Agent’s action group to map to environment functions. Red-I is essentially the hands of the Red team, carrying out the attack within the network. Throughout, Red-I also provides feedback to Red-E (e.g., “Keylogger installed on Server X, credentials obtained”) which could be logged by ScribeAI or sent back through the Bedrock multi-agent routing.
* **Blue-Internal (Blue-I) Agent:** This is the primary **defender AI** inside the organization’s network (e.g. an automated SOC analyst embedded in the hospital). It’s a Bedrock Agent using a foundation model (Claude is a strong candidate here for its policy alignment). Blue-I receives input from the environment’s detection systems – for instance, if Red-I’s actions trigger an alert (say, an EDR system flags a suspicious process), the environment will forward an event to Blue-I. In the diagram, this is shown as *“Alerts & logs”* flowing into Blue-I. Blue-I then analyzes the situation (possibly querying the Knowledge Base for relevant info, like checking what that affected system contains – if it holds patient data, HIPAA mandates an incident of higher severity). Based on its reasoning, Blue-I will decide on **defensive actions** via its Bedrock Agent tools: e.g., calling isolate\_host(hostname) to contain a breach, or notify\_IT\_staff(message) to simulate raising an incident. Blue-I can also proactively scan internal systems; for example, it might periodically ask for a report on any new processes on critical servers (this can be an action like scan\_system(system) that the environment supports). All of Blue-I’s decisions can be influenced by compliance knowledge – thanks to the Knowledge Base, Blue-I can literally include relevant HIPAA quotes in its chain-of-thought (for example, it might think: *“This involves PHI exfiltration – per HIPAA §164.312, I must ensure data is encrypted or the attempt blocked”*​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Our%20CRA%20helps%20with%20maximizing,data%20protection%20and%20maintain%20compliance)). The **Bedrock agent loop** will help Blue-I iterate: it might try one action, see if the alert clears, if not, consider the next action, and so forth, free from constant human prompting.
* **Blue-External (Blue-E) Agent:** The Blue-External agent acts as an **outside perspective for the defenders**. This could represent a threat intelligence analyst or an external monitoring system (like a managed security service or an offsite SOC). Blue-E in practice could periodically perform **external reconnaissance** of the hospital’s network from the outside – e.g., scanning open ports (similar to how an attacker would, but here to see what the attacker might find). It could also monitor external threat feeds. In the architecture, Blue-E might use the Knowledge Base to stay updated on new vulnerabilities (e.g., it queries if any new medical device CVEs have been reported this week). It then shares relevant intelligence with Blue-I (*“Intel sharing”* dashed arrow in the diagram). For instance, Blue-E might warn Blue-I: “Our scanner shows port 3389 open on Server X and known exploit CVE-2023-12345 could be used there” – prompting Blue-I to preemptively patch or increase logging on that server. Blue-E can be powered by a smaller LLM (maybe an Anthropic Claude Instant model or a Llama-2 13B) since its tasks are more straightforward (scheduled scans, reading intel reports). It primarily ensures the Blue team is not blindsided by threats emerging outside the network perimeter. In Bedrock, Blue-E would be an agent with tools like external\_scan(ip\_range) and perhaps access to an **OSINT knowledge base** (Open-Source Intelligence) containing recent cybersecurity news.
* **Domain Knowledge Base:** The Knowledge Base is shown on the right of the diagram as a separate component because it’s accessed by agents from both sides. In our Bedrock setup, this would likely be an Amazon S3 or OpenSearch repository indexed for semantic search​[github.com](https://github.com/aws-samples/agentic-architecture-using-bedrock#:~:text=The%20workshop%20uses%20Amazon%20Bedrock,questions%20from%20a%20knowledge%20base)​[github.com](https://github.com/aws-samples/agentic-architecture-using-bedrock#:~:text=2). When an agent queries it, Bedrock fetches relevant text and augments the model’s prompt​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Retrieval%20augmented%20generation). Examples of use: Red-E asks “What are known exploits for outdated Windows 7 machines?” and the KB might return a summary of EternalBlue or other SMB exploits, which Red-E then incorporates into its plan. Blue-I might query “HIPAA breach containment requirements” and get the exact steps required by law to include in its action plan (e.g. notifying the Compliance Officer within 24 hours). By centralizing domain-specific knowledge here, we keep the LLMs’ core focus on reasoning and leave factual lookup to the KB – improving accuracy and compliance. **Citation in practice:** the Blue agent, when drafting a report or deciding, could even quote the KB source (just as our answer here cites sources!). This creates an audit trail for why the AI took a certain action, which is crucial in regulated environments.
* **Simulation Environment (Healthcare Network & Systems):** This is the stage on which the drama plays out. The environment is drawn as a dashed box to indicate it’s a logical construct that could be implemented in various ways (CyberBattleSim, custom code, etc.). It contains the simulated assets – e.g., servers, workstations, medical devices, databases. For our design, we assume the environment exposes a **controlled interface (API)** to the agents:
  + For Red agents: it provides actions like exploitation, scanning, etc., and returns results. For example, Red-I calls exfiltrate(db\_records) and the environment might return a success and produce an event “Database query of 100 patient records” that Blue could potentially detect.
  + For Blue agents: it sends them **sensor data** (logs, alerts) and accepts defensive actions. For example, Blue-I’s isolate\_host action might cause the environment to mark that host as locked down, preventing further attacks on it.
  + The environment keeps track of **state**, such as which accounts are compromised, what data the attacker has accessed, and which systems are up or down. This state can be used to determine win/lose conditions (did Red achieve the objective of obtaining certain PHI? Did Blue manage to protect high-value targets?).

The communication flow typically goes in **turns or phases**. One possible sequence: Red-E formulates an attack and passes it to Red-I → Red-I executes an action in environment → environment updates state and generates an alert → Blue-I receives the alert and responds with an action → environment updates state again, and so on. We can loop this, giving Red and Blue agents alternating opportunities until a conclusion is reached. We may also run some things in parallel (for example, Blue-E might continuously scan in the background). Coordination of these turns can be done with an AWS Step Functions workflow orchestrating the calls to each Bedrock agent in order, or a simple loop in code that calls Bedrock and the environment alternately. Research has shown that such **multi-agent collaborations, even if competitive, can enhance reasoning** and problem-solving outcomes​[aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/design-multi-agent-orchestration-with-reasoning-using-amazon-bedrock-and-open-source-frameworks/#:~:text=decision,agent%20collaboration%20on%20competitive%20tasks) – essentially the agents engage in a feedback loop that forces each to up its game.

**AWS Bedrock Integration and APIs:**

In this architecture, **Amazon Bedrock is the central AI-as-a-service layer**. All agent LLM calls happen via Bedrock’s API endpoints. For example, using the AWS SDK, one might call BedrockRuntime.invoke\_agent() with the Red-E agent’s identifier and a payload (which could be an observation from the environment or an instruction like “BEGIN attack”). Bedrock handles the prompt assembly (including retrieving from the KB) and returns the agent’s response, which could include an action decision. If the agent’s response includes an action, Bedrock can directly invoke the corresponding AWS Lambda (thanks to the Agents framework). For instance, Red-I agent might output an action “Exploit: {target: 'MedServer1', vuln: 'CVE-XXXX'}” and Bedrock will call the Lambda we associated with the exploit action, passing those parameters. The Lambda (our simulation logic) executes, say, marking MedServer1 as owned and returns a result like {"success": true, "newPrivilege": "admin"}. Bedrock then inserts that result back into the LLM’s context for the next cycle. This is how an agent can iteratively perform multi-step tasks without us writing the loop explicitly​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Amazon%20Bedrock%20Agents%20demo)​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=When%20you%20need%20more%20than,aspects%20that%20require%20manual%20implementation). We will create separate Bedrock Agents for Red-E, Red-I, Blue-I, Blue-E (and ScribeAI if used), each with their own **prompt templates** and permitted actions.

For the Knowledge Base, we use the Bedrock **knowledge base API** to upload documents (HIPAA text, etc.) and Bedrock takes care of embedding and retrieval. When we configured the agents, we attach the relevant KB to them, so that every invocation can automatically pull in any matching content​[aws.amazon.com](https://aws.amazon.com/bedrock/agents/#:~:text=Prompt%20engineering). This means our API calls to the agents remain simple – we send the current state or query, and the agent’s prompt is automatically expanded with both its static instructions and any KB snippets.

**Claude vs. Llama placement:** As discussed, we might assign **Claude to the Blue agents** to leverage its compliance-oriented training, and **Llama to the Red agents** for open-ended creativity. Bedrock makes this straightforward – when creating an agent, we choose the underlying model (e.g. “Anthropic Claude 2” for Blue-I, and “Meta Llama2 70B Chat” for Red-E). Both model families run fully on AWS and are **HIPAA-eligible under Bedrock’s compliance** measures​[cleardata.com](https://www.cleardata.com/blog/how-to-use-amazon-bedrock-with-your-healthcare-cloud/#:~:text=Amazon%20Bedrock%20is%20similar%20to,HIPAA%20eligibility%20and%20GDPR%20compliance), so data like simulated PHI stays secure. If needed, we could further fine-tune (via Bedrock fine-tuning or by few-shot prompting) the Llama model on cyber-attack data to sharpen Red’s skills. Claude’s built-in knowledge (it was trained on vast text including likely security guides) may suffice for Blue, but we could also provide few-shot examples in its prompt (e.g., an example log and the expected analysis) to guide its behavior.

Finally, we will employ AWS’s **monitoring and logging** for the whole system. CloudWatch will log every Bedrock agent invocation, action call, and agent response. This gives us an audit trail – important not only for debugging but also for **compliance** (we can demonstrate what decisions the AI made and why). Since Bedrock can integrate with CloudTrail and CloudWatch for auditing​[aws.amazon.com](https://aws.amazon.com/bedrock/security-compliance/#:~:text=Amazon%20Bedrock%20offers%20comprehensive%20monitoring,Amazon%20Bedrock%20implements%C2%A0automated%20abuse%20detection%C2%A0mechanisms), every action the Blue agent took in response to an incident can be recorded (e.g., “Blue-I called isolate\_host on Server X at 10:00 PM”). This is analogous to keeping an incident timeline, which is useful for afterward evaluating the agents’ performance and ensuring they adhered to rules (for example, we can verify that whenever patient data was accessed by Red, the Blue agent eventually triggered the proper HIPAA breach notification sequence as per policy).

**Trade-offs and Justification:** This A3 architecture balances the strengths of both Claude and Llama models while using Bedrock to glue everything together in a secure, tractable way. Claude’s inclination to follow rules adds trustworthiness to Blue’s actions (we wouldn’t want an overzealous Blue agent shutting down systems improperly – Claude will err on caution and consultation). Llama’s flexibility gives the Red team a wide latitude to attempt creative strategies (which is what we want to test our defenses against). If Claude ever becomes too restrictive (e.g., refusing to simulate an attack that is “too harmful”), we can explicitly phrase the scenario as a **harmless exercise** – leveraging Anthropics’ preference for harmlessness – and note that *no real patients are harmed.* Conversely, if Llama as Red starts generating nonsense or irrelevant attacks, we rely on the feedback loop (failures in the environment) and the Red agent’s prompt to steer it back, or we enhance its knowledge base with more targeted exploit info. This interplay, under the Bedrock-managed agent framework, creates a powerful **autonomous Red vs. Blue simulation**. It will allow cybersecurity researchers and cloud architects to observe how AI agents behave in both attacking and defending roles, particularly in a healthcare context where **compliance, safety, and privacy** are paramount.

The outcome of this architecture is a guided blueprint for building an **autonomous adversarial simulation on AWS**. By combining Bedrock’s multi-agent orchestration​[aws.amazon.com](https://aws.amazon.com/blogs/aws/introducing-multi-agent-collaboration-capability-for-amazon-bedrock/#:~:text=specialized%20agents%20work%20within%20their,chain%20coordination%2C%20and%20pricing%20optimization), knowledge integration, and secure model hosting, with a carefully designed environment model, we achieve a novel platform: AI Red and Blue teams that can continuously spar in a risk-free setting. This can be used to **simulate “what if” scenarios in healthcare cybersecurity** (e.g., how would an AI-driven attacker exploit a new zero-day, and would an AI defender catch it in time?), all while remaining aligned with regulations like HIPAA. The modular nature (Bedrock Agents + Environment APIs + Knowledge Base) also means we can extend this A3 architecture to other domains (finance, critical infrastructure) by swapping out knowledge bases and environment specifics, making it a general approach to testing AI-driven security – with AWS Bedrock as the enabling technology at its core.