"How to", Practical approach to Multi Agent Cyber Defense

Dylan Renier Pérez Narváez
Master in Computer Science
Advisor: Lisabel Rodríguez Espinosa, J.D.
Polytechnic University of Puerto Rico
Graduate Project EXPO, May 2026

Abstract *— This project is a guided "how to" approach that explains how multi agent defense is studied, how theory guides exploration, and how practical understanding develops. It includes two hands-on proof of concepts, one built in n8n and one in Python, that put these ideas into practice using real tools. The goal is to help connect research and application in order to simplify the study of the field. This will give the reader both conceptual understanding as well as a real application of the material.*

Keywords *— Autonomous Agents, Cyber Defense, Distributed Security, Multi Agent Systems*

INTRODUCTION

Cybersecurity no longer deals with isolated incidents. Attacks adapt, coordinate, and move across networks. Centralized monitoring struggles with response time and context. Static rules fail once an experienced attacker learns about a system and adjusts their attack to be perfect for a specific network. This shift forces defenders to rethink structure and control across security systems. Forum estimated that cybercrime cost the global economy $10.5 trillion in 2025 [1]. Most of the time a lot of these professionals end up having to change many things just to "switch it up" in the network. In my experience in the field most organizations have a team that takes information from attacks and from there keeps refining the network, building on top of the already made infrastructure. According to recent industry analyses, over 60% of enterprise breaches involve lateral movement following a credential compromise [2]. Defense becomes an ongoing adjustment process instead of a fixed setup, and that becomes a double-edged sword for the organization, because missing any new technology means falling behind.

Multi agent cyber defense responds to these challenges through distribution. Instead of one control point, several agents operate at the same time across the environment. Each agent focuses on a specific task such as monitoring traffic, analyzing behavior, or flagging suspicious activity. Agents share information so decisions reflect more than one perspective. If one part of the system fails or gets overwhelmed, other agents continue operating. Most research on this topic assumes that people already know what agents are and how to use them. My aim in this project is to explain and apply the knowledge on multi agents to defense. The work treats multi agent defense as something to learn by studying the behavior and interactions in a network. By the end of this paper the reader will have a step-by-step walkthrough of a working multi agent system they can replicate on their own. One note on terminology. The word "agent" gets used in different ways in research. In the stricter reinforcement learning definition, an agent has a policy, observes a state, takes actions, and learns from rewards. In the less restrictive definition common in recent LLM based systems, an agent is an autonomous component with a specialized role that makes independent decisions based on its input. This project uses the loose definition for the worked examples. The focus is on structure and communication between agents, not on trained policies or reward functions.

PURPOSE AND OBJECTIVES

The purpose of this project is to give a clear guide for understanding multi agent cyber defense. While there is extensive literature on autonomous agents, distributed decision making and learning based agents, these topics are often presented in isolation that makes learning these extremely difficult to understand how they all work together

in the real world. This project organizes those ideas into a coherent framework, examines the technologies that support them, and includes two proof of concepts that show the principles in a working scenario.

THEORETICAL FOUNDATIONS

Multi agent cyber defense is built on the idea that security should not depend on a single point of control. Each agent works with only the information given to it and stays hyper focused on its own task. The decision an agent makes on one part of the system does not interrupt the decisions made on another part. Coordination mechanisms help those agents work together when needed so the system behaves coherently as a whole even though each agent only sees a small part of the environment.

Core Principles and Architectures

The first important principle is autonomy. Each agent works independently and reacts when changes happen in its specific environment. They do not need a person constantly looking over them and can react faster than a person can. Think of this as an illness. You go to the doctor when you get symptoms, now imagine you get a daily blood test that tells you if you are sick even before you notice the symptoms. This is how these agents work, they focus on a part of the system and if anything is out of place they react. The second principle is cooperation. If there is no cooperation between agents, the system will fail because an attacker can overwhelm a single agent. With cooperation agents can confirm observations, avoid duplicating work, and give a unified response. This cooperation can be handled via predefined rules or through a learning mechanism. The third principle is trust and oversight. There must always be a balance between what we trust the agent to do and what we need to verify. A lot of people use these tools and see it as a "black box" where the issue gets fixed, but nobody knows how. There must be clear rules to explain why, how, and what it did. The agent must "present its case" so the professional can understand and manage it accordingly. There are multiple ways to run these agents. Fully decentralized has no global constraints. Fully centralized does not let the agent do anything without approval. Normally the most used is hybrid architecture where agents can act locally while following a set of global restraints. This is the architecture used in the proof of concepts later in this paper

TECHNOLOGIES

Multi agent cyber defense relies on a combination of technologies that let agents take in information, communicate, and act. Understanding these technologies is important because without that knowledge the agents become a black box. The system might be working but there is no way to know why it did what it did or how to fix it when something goes wrong.

Agent-based Modeling and Architecture

Agent-based modeling focuses on defining each individual agent, their states, and behavior. The first area to understand is the perception layer, which is how the agent gathers information from its environment. In cyber defense this might include capturing network packets, monitoring system calls, pulling log data, or integrating with SIEM systems and firewalls. Then you have the decision-making engine, which determines what actions the agent should take using either rule-based policies or utility-based reasoning where it weighs costs and benefits. When it comes to agent architecture, there are a few common types. Reactive agents follow basic conditions to action rules and are fast but limited. Deliberative agents maintain an idea of the environment and plan accordingly but are computationally expensive. BDI agents (Belief-Desire-Intention) model beliefs about the world, goals, and committed plans, which make them useful for goal-oriented security tasks. In practice most systems use hybrid agents that combine reactive components for fast response with deliberative layers for strategic planning.

Distributed Intrusion Detection Systems

Distributed Intrusion Detection Systems are one of the most mature applications of multi agent principles in cyber defense. The basic idea is to distribute detection logic across multiple nodes, with each node running an agent responsible for local monitoring [3]. They structure agents by tiers. At the bottom level there are data cleaning agents that process raw information from different sources and convert it into a common format. Above that you have detection agents that focus on specific tasks like watching over privileged programs or monitoring network traffic. At the top level there are higher level agents that take all the data gathered by the lower level agents and data mine the entire dataset. These top level agents can identify coordinated attacks because they view the system from a higher level than any individual agent. Imagine a combined NFS and rlogin attack on a UNIX system. A single agent looking at network traffic might not see the issue. But when one agent flags weird NFS activity, another sees a file modification, and a third notices an rlogin from an untrusted host, the high level agents can look at all these discrepancies together, identify that this is a coordinated attack, and figure out where it started. This tiered structure is exactly the architecture used in the proof of concepts built later in this paper.

Reinforcement Learning for Defense

Reinforcement Learning is basically telling the agent if they did their job right or messed up somewhere. An agent watching network traffic sees suspicious activity and has a set of actions: block the IP, ignore the issue, or call a human. If it blocks a malicious IP it gets positive feedback. If it ignores a real attack or blocks legitimate traffic, it gets negative feedback. Over time it learns which actions work best in different situations [3]. A good example comes from research on detecting attacks against privileged programs like send mail on UNIX systems. Normal programs follow predictable patterns in their system calls and when a program gets exploited those patterns change. Algorithms like RIPPER [4] learn what normal patterns look like and flag anything that deviates. The proof of concepts in this paper do not use reinforcement learning directly, but the Pattern Analysis Agent performs a similar function by matching behavioral patterns against known attack techniques.

Agent Communication

For multi agent systems to work agents need to communicate and share insight on attacks. Agents typically use several types of messages: information messages that report suspicious activity, request messages when an agent needs more data from another agent, proposal messages when an agent suggests a coordinated response, and alert messages for urgent situations that need immediate attention [5]. There is also the question of how much agents should communicate. If every little observation gets shared the channel floods and important messages get lost. Research on weight based communication scheduling addresses this by having each agent report a weight based on its current observation, and a scheduler prioritizes messages by importance [5]. There are two main communication layouts. Broadcasts have one agent relay every message to every other agent, which creates unnecessary traffic. Figure 1 shows this layout. The publish-subscribe approach groups agents by topic so they only receive relevant messages as shown in Figure 2.

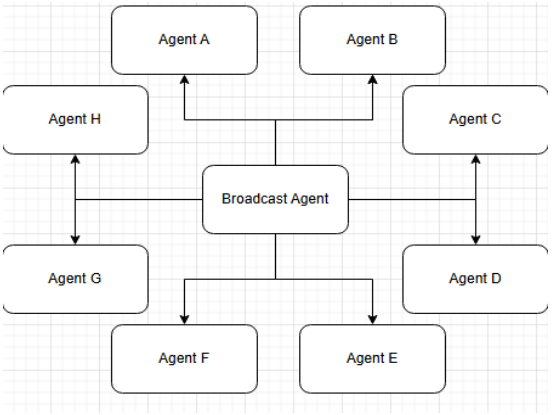

Figure 1
Broadcast Layout [3]

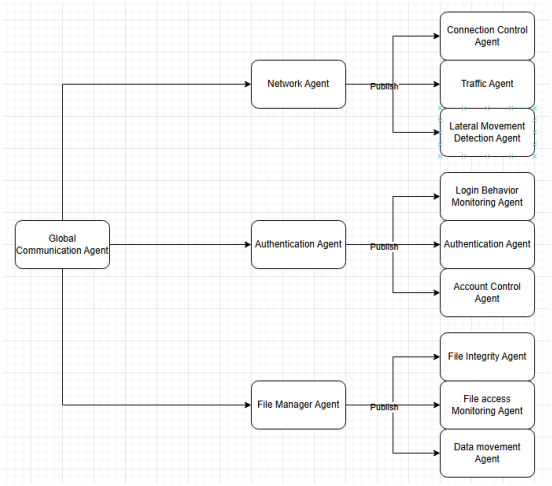

Figure 2
Publish Subscribe Layout [3]

IN PRACTICE

So far we have explained how these multi agent systems work and how they would look in cyber defense. The next step to understanding the topic is through examples. For this part two implementations were made. The first is a multi agent cyber defense workflow built in n8n (AI Workflow automation) that runs suspicious inputs through specialized agents in a fixed pipeline. The second is a Python script that takes this concept further by letting Claude decide which agents to call based on what the input actually contains. We will go through how to build them and give an explanation on how they work, what they do, and how they differ from each other.

CREATING THE WORKFLOW

This will be a walkthrough on how to build an agent cyber defense workflow. The system created takes in user input for suspicious behavior, runs it through three specialized autonomous agents, and produces a single threat assessment. This first example gives us a look as to what is going on in the "black box" in the second example.

Setup

For the first example the setup begins with getting an account for n8n cloud. N8n cloud is a visual AI workflow builder that allows users to set up AI workflow visually rather than working directly with code [6] [7]. This becomes very useful when paired with API keys that can call LLM's and can call websites to then bring you data to use in your workflow. The next step is to get an AI API key, for this example we used Anthropic's Claude API key. This can be obtained through console.anthropic.com. You need to create an account, add a payment method, and load credits. Once your account has credits go to the API Keys section and generate a new key. Copy it and save it somewhere safe because you will not be able to see it again after leaving that page. Last step was to determine which LLM model fit our use case. It was determined that Haiku was going to be used since it has very fast response times and is very cheap in per call cost [8].

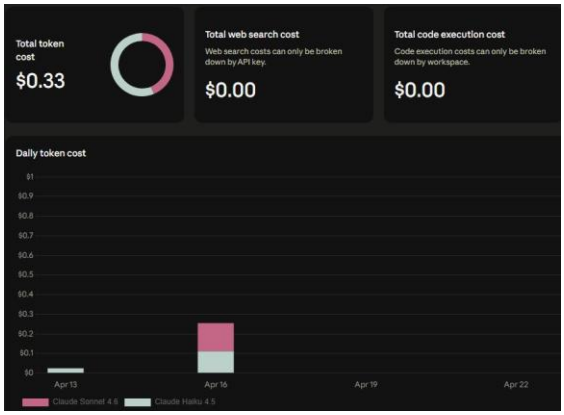

Figure 3
Anthropic API Cost [8]

To put this into perspective, Figure 3 shows the Anthropic API cost dashboard during development. On the same day both Haiku and Sonnet were used and the cost was roughly similar, but Haiku was run significantly more times than Sonnet and still ended up being less expensive. Figure 4 shows the token usage breakdown. Haiku used only about 7 thousand more tokens than Sonnet even though it was called many more times with 3 agent calls per run. This should be analyzed whenever looking at different alternatives for API key costs. If the application uses multiple Sonnet calls then the price starts ramping up very quickly and that's something important to keep in mind, these models are not

cheap to use and should fit the use case as close as possible.

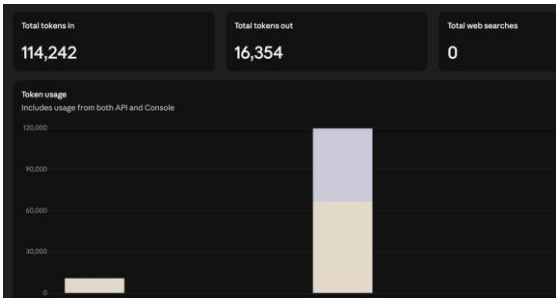

Figure 4
Anthropic API Token Usage [8]

Architecture Development

The workflow created has 4 different Claude instances, each set up with a specific task. These are the following: Input Analysis, Network Analysis, Pattern Analysis, and the Coordinator. Whenever a user inputs something into the chat interface, n8n captures that input and sends it as a variable to Input Analysis, then it keeps going through them until it reaches the Coordinator and outputs the threat analysis. This approach of embedding AI agents into workflow nodes to create multi-step automated processes is consistent with recent research on combining n8n with AI driven decision making [9]. Figure 5 shows the full workflow.

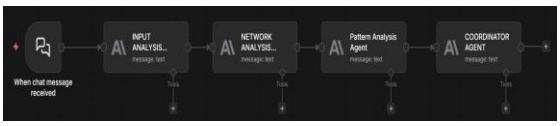

Figure 5
N8N Cyber Defense Workflow [9]

Every node in this workflow maps directly to a concept discussed earlier in this paper. The Input Analysis Agent is the data cleaning agent from the Distributed Intrusion Detection Systems section, the bottom tier that converts raw data into a common format before passing it up the chain. The Network Analysis Agent and Pattern Analysis Agent are the mid tier detection agents, each hyper focused on one domain. The Coordinator Agent is the high level agent that sees all findings and produces a unified assessment. This is the hybrid architecture where agents act autonomously but the Coordinator enforces global constraints.

Building it with N8N

The chat trigger that was presented before, works as the entry point for the workflow. It waits for user input, when the user enters a suspicious network situation, the chat trigger takes that input and sends it to the next agent. The first agent in the list would be the Input analysis agent. The job is to take whatever the user entered and clean in into a predetermined format. It identifies the input type, cleans the raw data, and pulls out key indicators that stand out. Every agent downstream works from this structured output instead of the raw user input. The Network Analysis Agent only analyzes network related indicators. It does not look at behavioral patterns because that is the next agent's job. It focuses on suspicious IP ranges, unusual ports, known malicious patterns, and traffic anomalies. It returns a threat level on a 0 to 5 scale, a short description of what it found, and a confidence score. The Pattern Analysis Agent focuses exclusively on behavioral attack patterns. It looks for lateral movement, credential attacks, privilege escalation, and anything matching a known attack technique. The important design decision here is that this agent reads directly from the Input Analysis Agent, not from the Network Analysis Agent. Each detection agent gets the same cleaned input independently, so their analyses are not biased by what the other agent found. The Coordinator Agent is the high level agent that brings everything together. It receives the outputs from all three previous agents and produces a unified threat assessment with an overall severity level, a summary combining both agents' findings, and a note about which agent flagged the highest threat. It also accounts for potential agent errors by increasing severity if either detection agent returns a high finding.

Testing

To make sure that the created example works correctly, we used four different test scenarios were

created. Each one is supposed to trigger a different combination of agents so that we can validate the constraints and if the agents themselves are working. The first test focused on the network agent. A SYN flood input with a suspicious external IP hitting multiple ports across hundreds of internal hosts was used to test the Network Agent. A SYN flood is a type of denial of service attack (DOS) in this case using SYN packets. This means that the system get flooded with these packets and tries to overwhelm the target in order to disrupt operations. When the information was sent to the agent the Network Agent flagged it at threat level 5 while the Pattern Agent returned 0 since there was no behavioral context. The Coordinator returned Very High driven by the Network Agent.

The second test was focused on patterns with no network data in order to test the pattern agent. For this test the user opens Word, Word spawns cmd.exe, PowerShell runs a base64 encoded command, and a new admin account gets created. The Network Agent returned 0 since there are no IPs or ports. The Pattern Agent flagged it at threat level 5 as a macro attack with privilege escalation. The Coordinator returned Very High based on the Pattern Agent alone.

Now that we have verified that both agents are working by themselves we need to verify that they work at the same time. The third test was a combined test that used a suspicious account authenticating from an external IP at 2:47am, enumerating domain accounts, then connecting outbound to port 4444. Both agents flagged this at threat level 4 to 5. Neither agent alone tells the full story but together they give a complete picture of the attack.

The last test was for no threat, meaning that both the network and the pattern agent would not find any threats. The test was a normal business hours login from an internal IP, routine HR portal access, a single file download, and a clean logout. Both agents returned threat level 0 with high confidence. The Coordinator returned None. This validates that the system does not over-flag normal activity.

Table 1
N8N Workflow Agent Output Comparison

Test	Severity	Network Threat	Pattern Threat	Agents Flagged
SYN Flood	5	5	0	Network Agent
Word Attack	5	0	5	Pattern Agent
Both Attack	5	4 - 5	4 - 5	Both Agents
Normal Activity	0	0	0	Neither

PYTHON AGENT

The n8n workflow is a pipeline. It runs every agent in a fixed order no matter what the input looks like. The Python agent takes a different approach. Instead of hardcoding the sequence, it gives Claude a list of available tools and lets Claude decide which ones to call based on what it sees in the input. This is what makes it a true agent rather than a pipeline.

What the Script Does

The script uses the Anthropic Python library [10] to communicate with Claude's API. It defines three tools: network_analysis, pattern_analysis, and coordination_agent. Each tool has a description written in plain English that Claude reads to decide when to use it. These descriptions are specific about what each tool does and when it should or should not be called. For example, the network analysis description says to call it only if the input contains network data such as IP addresses, port numbers, or traffic patterns, and not to call it if the input only describes user behavior with no network data present. Figure 6 shows the tool definitions in the script.

There is also a system prompt that acts as the orchestration rules. It tells Claude to analyze what type of data is in the input before deciding which agents to activate. If the input has only network data, skip the pattern agent. If the input has only behavioral data, skip the network agent. If both types are present, call both. The coordination agent always runs last. Figure 7 shows the system prompt. Each tool is backed by its own Python

function that builds a prompt and sends it to Claude as an independent API call.

```
"name": "network_analysis",
"description": """Analyzes network-level indicators
 only: IPs, ports, traffic patterns.
 Does not assess behavior.""",
"input_schema": {
    "type": "object",
    "properties": {
        "raw_input": {"type": "string"}
    },
    "required": ["raw_input"]
}
```

Figure 6
Python Script Tool Definition

```
SYSTEM_PROMPT = """You are the orchestrator of a
multi-agent cyber defense system.
Call network_analysis and pattern_analysis as
needed based on what the input contains.
Skip network_analysis if the input has no
network indicators. Skip pattern_analysis if
the input has no behavioral indicators.
After the analyses are done, call cordination_agent
to produce the final assessment."""
```

Figure 7
System Prompt for Agent Orchestrator

Agent Loop

The core of the script is a while True loop inside the run_agent function. This loop is what separates an agent from a pipeline. Every iteration Claude receives the full conversation history including all previous tool calls and their results. It uses that history to decide what to do next. When Claude wants to call a tool it returns a response with a stop reason of "tool_use" along with the tool name and arguments. The script catches this, runs the matching Python function, prints the result to the terminal, and sends the result back to Claude. The loop only breaks when Claude returns a stop reason of "end_turn" which means it is done analyzing. This observation, reasoning, and action cycle is what recent literature describes as the core of agentic AI systems [11]. Figure 8 shows the agent loop in the script.

The conversation history acts as the agent's memory within a single run. By the time Claude calls the Coordinator it has the full context of everything that happened. This is similar to the information messages described in the Agent Communication section where agents pass findings upward through the system. It's worth mentioning that the python script does not have a cleaning agent. This is because in N8N there was a need for input formatting in order to make sure that it went through every step correctly. This doesn't exist in the python script because there simply isn't any need for it. Claude's tool use capability handles raw input directly without needing a separate cleaning step.

```
while True:
    response = client.messages.create(
        model=MODEL,
        max_tokens=1000,
        system=SYSTEM_PROMPT,
        tools=TOOLS,
        messages=messages
    )
if response.stop_reason == "tool_use":
    messages.append({"role": "assistant",
                     "content": response.content})
    results = []
    for block in response.content:
        if block.type == "tool_use":
            display = block.name
            print(f"\n{display}")
            result = run_tool(block.name, block.input)
            print(result)
            results.append({
                "type": "tool_result",
                "tool_use_id": block.id,
                "content": result
            })
    messages.append({"role": "user", "content": results})
else:
    break
```

Figure 8
Python Agent Loop

Intelligent Tool Selection

The most important difference between the two builds is that the Python agent chooses which tools to use. A pure network input like "Incoming traffic from IP 185.220.101.34 on ports 22, 80, 443, 8080" will only trigger the Network Analysis and Coordinator agents. The LLM in this case, sees that there is nothing behavioral to analyze, and skips the Pattern Analysis tool entirely as shown in Figure 9. A pure behavioral input like the admin attack will only trigger Pattern Analysis and the Coordinator as shown in Figure 10.

```
> Incoming traffic from IP 185.220.101.34 on ports 22, 80, 443, 8080

network_analysis
NETWORK THREAT LEVEL: 2

NETWORK DESCRIPTION: Traffic from IP 185.220.101.34 is originating f
rom a known Tor exit node, which is a common anonymization service u
sed for both legitimate privacy purposes and malicious activities. T
he multi-port probing pattern (22, 80, 443, 8080) suggests reconnais
sance scanning across common service ports. This combination indicat
es potential reconnaissance activity, though the threat level remain
s low without evidence of successful exploitation or malicious paylo
ad delivery.

CONFIDENCE: 4

cordination_agent
OVERALL SEVERITY: Low

THREAT SUMMARY: Traffic from a known Tor exit node probing multiple
common service ports (22, 80, 443, 8080) indicates reconnaissance sc
anning activity. While the multi-port pattern is consistent with act
ive enumeration, there is no evidence of successful exploitation, ma
licious payloads, or confirmed attack behavior. The activity warrant
s monitoring but does not currently pose an immediate threat.

AGENTS THAT FLAGGED THIS: Network Agent
```

Figure 9

Terminal Output: Network Agent Only

```
> User jsmith logged in normally at 9am, opened Microsoft Word, then
 Word spawned cmd.exe which ran PowerShell with a base64 encoded com
mand and created a new admin account named "svc_update

pattern_analysis
PATTERN THREAT LEVEL: 5

PATTERN DESCRIPTION: Multiple indicators of malicious behavior detec
ted: legitimate application (Word) spawning command-line tools (cmd.
exe/PowerShell), use of base64 encoding to obfuscate commands, and u
nauthorized creation of a privileged admin account. This chain repre
sents a classic privilege escalation and persistence attack pattern
consistent with malware execution or compromised credentials.

CONFIDENCE: 5

cordination_agent
OVERALL SEVERITY: Very High

THREAT SUMMARY: Multiple indicators of active malicious compromise d
etected on jsmith's workstation. A legitimate application (Microsoft
 Word) was weaponized to execute obfuscated PowerShell commands, cul
minating in unauthorized creation of a privileged admin account ("sv
c_update"), which represents a classic privilege escalation and pers
istence mechanism. This attack pattern is consistent with malware ex
ecution or credential compromise and requires immediate containment
and investigation.

AGENTS THAT FLAGGED THIS: Pattern Agent
```

Figure 10

Terminal Output: Pattern Agent Only

```
> User account "admin_backup" authenticated from IP 91.108.4.15 at 2
:47am, immediately ran net user /domain to enumerate all accounts, t
hen established an outbound connection to 185.220.101.45 on port 444
4

network_analysis
NETWORK THREAT LEVEL: 5

NETWORK DESCRIPTION: Multiple indicators of compromise detected in s
equence: authentication from a suspicious IP at an unusual hour (2:4
7am), immediate execution of domain enumeration command suggesting r
econnaissance activity, and establishment of outbound connection to
a known Tor exit node (185.220.101.45) on non-standard port 4444. Th
is pattern is consistent with active lateral movement and data exfil
tration.

CONFIDENCE: 5
PATTERN DESCRIPTION: The sequence demonstrates classic post-compromi
se lateral movement behavior: authentication from an unusual IP at a
n atypical hour, immediate execution of domain enumeration commands,
 and rapid establishment of an outbound connection to an external IP
 on a non-standard port. This chained activity pattern is consistent
 with credential compromise, reconnaissance, and command-and-control
 communication.

CONFIDENCE: 5

cordination_agent
OVERALL SEVERITY: Very High

THREAT SUMMARY: The "admin_backup" account has been compromised and
is actively being exploited for lateral movement and potential data
exfiltration. The attacker authenticated from a suspicious IP at an
unusual hour, immediately performed domain enumeration reconnaissanc
e, and established communication with a known Tor exit node, indicat
ing command-and-control activity. This represents an active, multi-s
tage attack with clear indicators of credential compromise and malic
ious intent.

AGENTS THAT FLAGGED THIS: Both
```

Figure 11

Terminal Output: Both Agents Triggered

When the input contains both network and behavioral indicators, Claude calls both agents. Figure 11 shows a combined input where a suspicious account authenticates from an external IP at 2:47am, enumerates domain accounts, and connects outbound to port 4444. Both agents flag it at threat level 5 and the Coordinator returns Very High severity. Figure 12 shows the opposite case where normal business activity triggers neither agent and the Coordinator returns None. These four test cases together prove that the Python agent adapts its behavior based on what the input contains rather than running every agent every time.

```
> User maria.lopez logged in from 10.0.0.45 at 8:55am on a Tuesday,
opened the HR portal, downloaded the monthly payroll report PDF, the
n logged out at 9:10am

network_analysis
NETWORK THREAT LEVEL: 0

NETWORK DESCRIPTION: User maria.lopez accessed the network from inte
rnal IP 10.0.0.45 during business hours with a normal session durati
on of 17 minutes. Activity consisted of legitimate, authorized acces
s to the HR portal and standard document retrieval with proper logou
t procedure. No suspicious network indicators detected.

CONFIDENCE: 5

pattern_analysis
PATTERN THREAT LEVEL: 0

PATTERN DESCRIPTION: User accessed HR portal during standard busines
s hours on a weekday and downloaded a payroll report consistent with
 legitimate job responsibilities. Session duration was brief and pur
poseful with normal login/logout behavior.

CONFIDENCE: 5
cordination_agent
OVERALL SEVERITY: None

THREAT SUMMARY: This activity represents normal, authorized business
 operations with no indicators of compromise or policy violation. Al
l access parameters—timing, location, duration, application, and act
ions—align with legitimate employee responsibilities and standard se
curity practices.

AGENTS THAT FLAGGED THIS: Neither
```

Figure 12

Terminal Output: Both Agents Triggered

CHALLENGES AND LIMITATIONS

These builds demonstrate the structural and communication aspects of multi agent defense, not the learning based aspects. They show how agents can be organized into tiers, specialized by role, and how a coordinator produces a unified response from distributed findings. They do not show how agents learn from experience or how reward signals shape behavior. The communication in the n8n version is one directional. Data flows up through the tiers to the Coordinator. The Python version has slightly more flexibility since Claude can choose which agents to call, but it still cannot go back and ask a previous agent to re-analyze with new information.

Real multi agent systems have bidirectional communication where agents can request more information or challenge each other's findings. Even with these limitations both proof of concepts accomplish what they set out to do. They take the structural and communicative concepts from this paper and turn them into something you can interact with, test, and see how the pieces work together.

CONCLUSION

Multi agent cyber defense is a real approach to dealing with the fact that modern attacks are too fast, too distributed, and too adaptive for any single system to handle alone. This paper walked through the core principles of autonomy, cooperation, and trust, and the technologies that make these systems work including agent architectures, distributed intrusion detection, reinforcement learning, and communication patterns. The two proof of concepts took the theory concepts from this paper and turned them into something real that can be replicated. The n8n workflow showed the tiered agent structure as four actual nodes wired together in a visual pipeline. The Python agent showed what happens when the orchestration itself becomes intelligent and Claude decides which agents to activate based on the input. Both implementations used the same core agents and the test cases proved they work the way the theory says they should. I think this project shows that multi agent cyber defense is not as abstract or inaccessible as a lot of the research makes it seem. With the right tools and a clear understanding of the principles it is possible to go from reading about agents to building a working system in a matter of hours. The goal of this paper was to connect research and application in a way that is easy to understand for anyone trying to get into the field.

REFERENCES

[1] C. R. Landolt, C. Wursch, R. Meier, A. Mermoud, and J. Jang-Jaccard, *Multi-Agent Reinforcement Learning in Cybersecurity from Fundamentals to Applications*, NATO STO-MP-IST-209, 2025.

[2] M. Imani, T. Lan, and N. D. Bastian, *Multi-Agent Cyber Defense with Multi-Level Zero-Trust Actions*, IEEE MILCOM, 2025.

[3] G. G. Helmer, J. S. K. Wong, V. Honavar, and L. Miller, *Intelligent Agents for Intrusion Detection*, Iowa State University, 1998.

[4] Q. Beckman. (2025, April 8). *[WITH CODE] Model: RIPPER Rule-Based Algorithm* [Online]. Available: https://www.quantbeckman.com/p/the-ripper-reckoning-extracting-rules.

[5] Z. Da, *Research on Multi-Agent Communication and Collaborative Decision-Making Based on Deep Reinforcement Learning*, arXiv:2305.17141, 2023.

[6] U. R. Pol, "No-Code Intelligence: Building AI Agents with N8N," in *International Journal for Multidisciplinary Research (IJFMR)*, vol. 7, no. 6, Nov.-Dec. 2025.

[7] A. Javaid, Ready Tensor, and M. Abdelhamid, *n8n: Building No-Code Agentic Workflows* [Online]. Available: https://app.readytensor.ai/publications/n8n-building-no-code-agentic-workflows-gfrpsUkufCGL.

[8] Anthropic. (2025). *Claude Model Pricing* [Online]. Available: https://platform.claude.com/docs/en/about-claude/pricing

[9] H. Azmat, *Automating Business Workloads with n8n and AI Agents*, 2025.

[10] Anthropic. (2025). *Anthropic Python SDK* [Online]. Available: https://pypi.org/project/anthropic/.

[11] K. M. Joon, "No-Code AI Automation: Combining n8n with Agentic AI," *in Pohang University of Science and Technology (POSTECH)*, 2025.