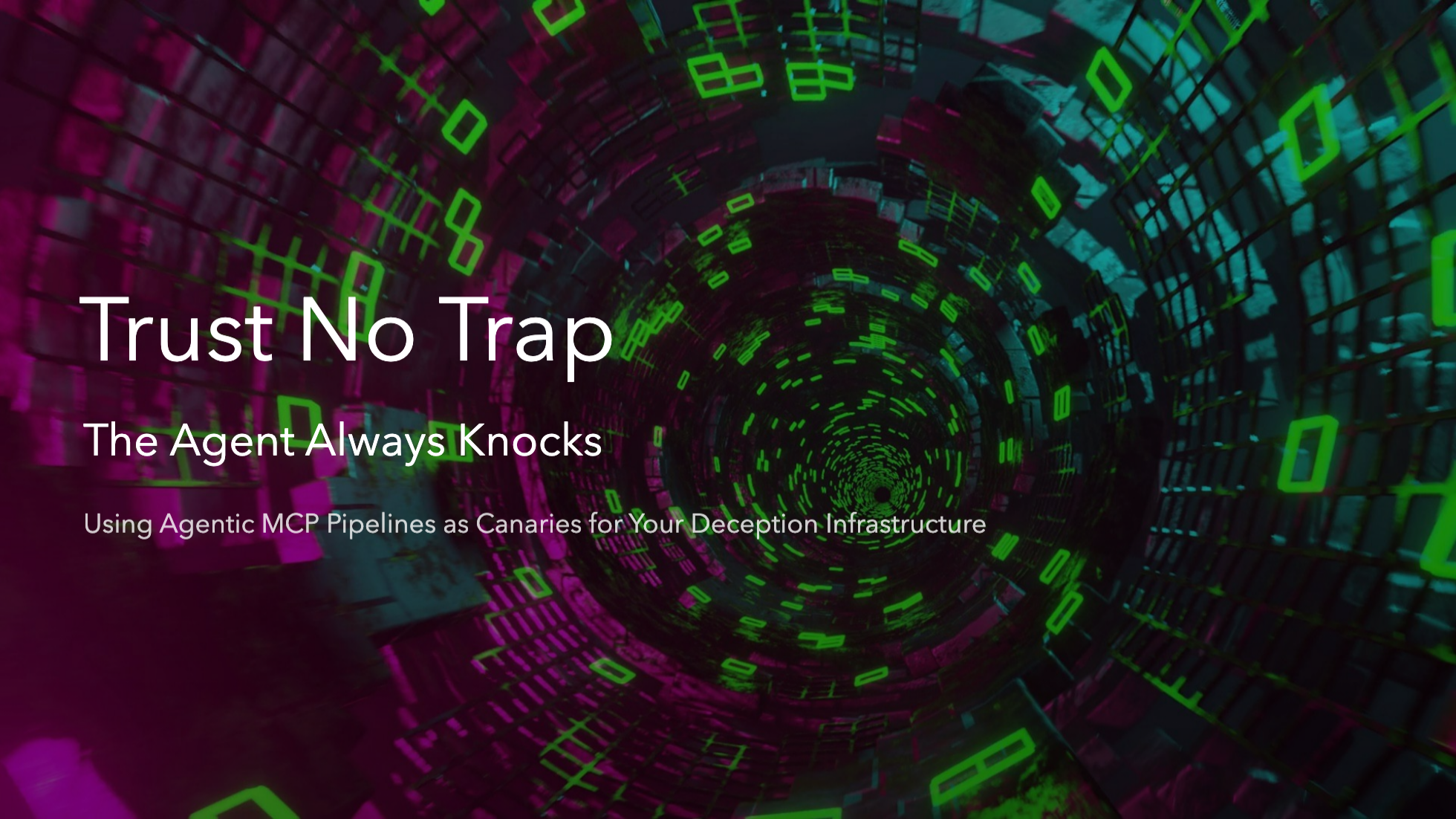

Trust No Trap
The Agent Always Knocks
Using Agentic MCP Pipelines as Canaries for Your Deception Infrastructure

Agenda

1. **The Deception Problem** — Your honeypots might be lying to you
2. **The Validation Gap** — How do you know traps are trapping?
3. **Agentic Canaries** — The concept behind this talk
4. **The Toolchain** — Atomic Red Team MCP + Microsoft Sentinel MCP
5. **Pipeline Architecture** — How the pieces connect
6. **Scenarios** — Live validation walkthroughs
7. **What We Found** — Common misconfiguration patterns
8. **Getting Started** — Run it yourself

The Deception Problem

Your traps might not be trapping

"A honeypot that doesn’t alert is just a free pivot point."

The silent failure mode nobody talks about

How Deception Tech Fails Silently

Infrastructure Failures

- SIEM integration breaks after an update
- Firewall rule blocks honeypot telemetry
- Log agent crashes and stops forwarding
- Authentication token for alert webhook expires
- TLS certificate on honeypot management plane expires

Configuration Drift

- Canary token not seeded in the right places
- Honeynet VLAN rules change during network reconfig
- Alert suppression rule accidentally catches honeypot events
- Honeypot credentials rotated without updating bait files
- Alert routing logic breaks during SIEM migration

None of these make noise. They just silently stop working.

The Validation Gap

How Most Teams "Validate"

1. Deploy honeypot ✅
2. Test it once during setup ✅
3. Assume it keeps working ❌
4. Check it again… never

Average time to discover a broken honeypot:

Nobody measures this. That's the problem.

What We Actually Need

- Regularly probe deception assets like an attacker would
- Verify that alerts fire in the detection platform
- Detect configuration drift before attackers do
- Generate evidence that deception controls are working

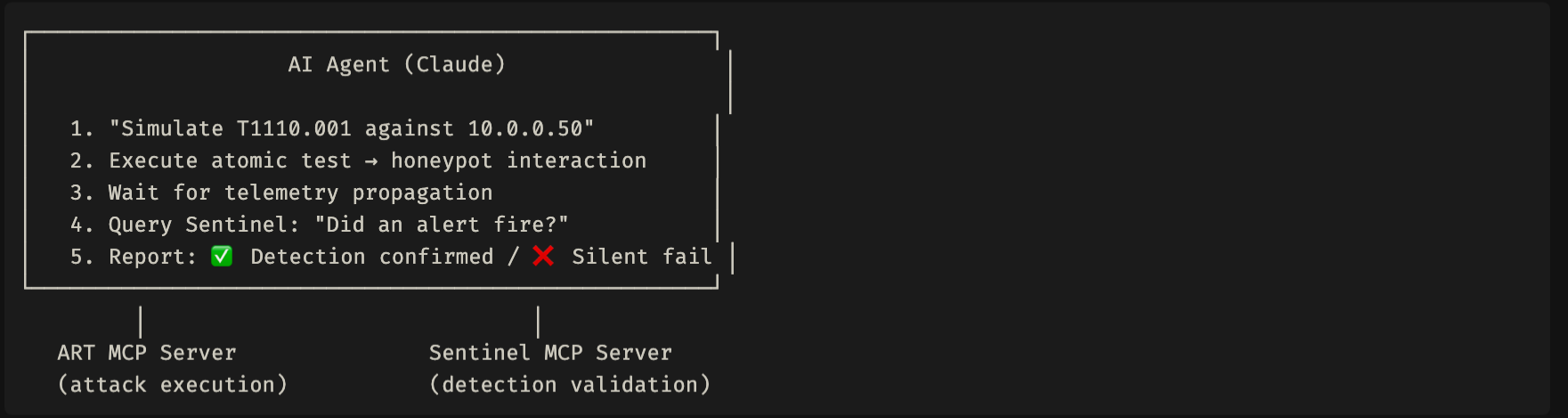
The solution: automate the knock. Make an agent be the canary.

The Agentic Canary

Using AI agents to validate your deception layer

The Core Concept

An AI agent that acts like an attacker, probes your honeypots, then checks if your SIEM noticed.



The agent is the canary for your canaries.

Why Agents? Why MCP?

Agents vs. Scripts

- Scripts are **static** – agents reason about results
- Scripts fail **silently** – agents surface anomalies
- Scripts need **hardcoded logic** – agents adapt to context
- Scripts require **maintenance** – agents interpret intent

The Reasoning Loop

An agent can:

1. Run a test
2. Observe partial telemetry
3. Hypothesize why an alert didn't fire
4. Run a different test to confirm
5. Conclude with evidence

Why MCP Specifically?

- **Composability** – swap attack tools without changing the agent
- **Auditability** – every tool call is logged
- **Isolation** – attack server and detection server stay separate
- **Portability** – works with Claude, Cursor, Gemini, any MCP client
- **Multi-server** – one conversation, multiple platforms

MCP turns the agent into a **composable security operator** – not a chatbot.

The Toolchain

Atomic Red Team MCP + Microsoft Sentinel MCP

Atomic Red Team MCP — The Attack Engine

1,500+ atomic tests mapped to MITRE ATT&CK, directly callable by an AI agent.

- `query_atomics` — Find tests by TTP, platform, keyword
- `execute_atomic` — Run a test against a target
- `validate_atomic` — Check YAML schema compliance
- `server_info` — Platform and execution status

```
# Deploy to your lab target
uvx atomic-red-team-mcp \
  --transport streamable-http \
  --port 8000
```

In the Canary Pipeline Context

The ART MCP server runs **on or near** your deception infrastructure. It performs the actual interaction that a honeypot should detect.

```
Honeypot Network:
   ├── SSH Honeypot (10.0.1.10)
   ├── RDP Honeypot (10.0.1.11)
   ├── SMB Share Honeypot (10.0.1.12)
   └── ART MCP Server (10.0.1.99)
        ↑ executes atomics that
          interact with the above
```

Microsoft Sentinel MCP — The Detection Engine

Microsoft's Sentinel MCP server exposes your SOC's detection data to AI agents.

Core Tools

- `run_kql_query` — Execute KQL against Log Analytics workspace
- `list_incidents` — Fetch incidents with filters
- `get_incident` — Full details on a specific incident
- `list_alert_rules` — Enumerate active analytics rules
- `get_alert_rule` — Check rule configuration

Setup

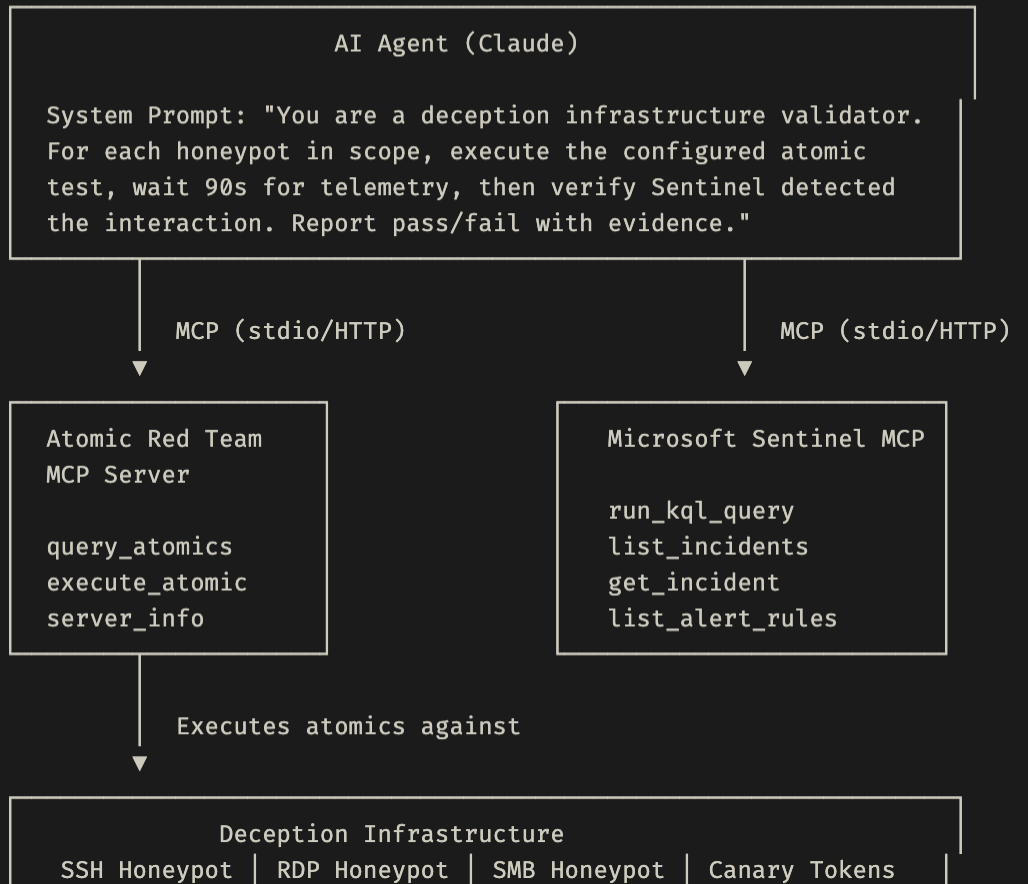
```
{
  "mcpServers": {
    "sentinel": {
      "url": "https://sentinel.microsoft.com/mcp/data-exp
    }
  }
}
```

Sentinel MCP gives the agent **read access to your detection results** — closing the feedback loop.

Pipeline Architecture

How the pieces connect

Full Architecture



Scenarios

Validating deception infrastructure in practice

Scenario 1: SSH Honeypot Validation

Goal: Confirm brute-force attempts against the SSH honeypot generate Sentinel incidents.

Prompt: *"Validate SSH honeypot at 10.0.1.10. Run T1110.001 and confirm Sentinel detected it."*

```
Agent → ART MCP:
  query_atomics("T1110.001", platform="linux")
  → "SSH Brute Force with Hydra" (abc-123)

Agent → ART MCP:
  execute_atomic("abc-123", {
    target_host: "10.0.1.10",
    username_list: "test,admin,root",
    password_list: "password,admin,123456"
  })
  → Executed. 47 attempts, 0 successful.

[Agent waits 90 seconds]
```

```
Agent → Sentinel MCP:
  run_kql_query("""
    SecurityEvent
    | where TimeGenerated > ago(5m)
    | where IpAddress == "10.0.1.10"
    | where EventID == 4625
    | summarize count() by IpAddress
  """)
  → 47 rows  ✅ Telemetry confirmed

Agent → Sentinel MCP:
  list_incidents(
    filter="SSH Brute Force", last_hours=1)
  → 1 incident created  ✅ Alert fired
```

Scenario 2: Canary Token Validation

Goal: Verify that a canary token embedded in a fake AWS credentials file fires when accessed.

Prompt: *"Validate our canary token in /shares/finance/aws-backup-creds.txt — simulate credential access and verify the Canarytokens webhook fires and Sentinel ingests the alert."*

```
Agent → ART MCP: query_atomics("T1552.001", platform="windows")
         Found: "Credentials In Files" tests

Agent → ART MCP: execute_atomic("T1552.001-3", {
           path: "\\\\fileserver\\finance\\aws-backup-creds.txt"
         })
         Status: File opened, contents read.

[Agent waits 120 seconds — canary token DNS resolution + webhook + SIEM ingestion]

Agent → Sentinel MCP: run_kql_query("""
           CanaryTokenAlerts_CL
           | where TimeGenerated > ago(10m)
           | where token_path_s contains "aws-backup-creds"
         """)
         Result: 0 rows → ❌ No telemetry

Agent → Sentinel MCP: list_alert_rules(filter="canary")
         Result: Rule "Canarytokens Alert Ingestion" — Status: DISABLED ⚠️
```

What We Found

Common misconfiguration patterns

Failure Mode Taxonomy

Running this pipeline against real deception deployments, we found these patterns:

Silent Ingestion Failures (Most Common)

- Data connector disabled post-migration
- Log agent stopped forwarding after OS update
- Custom log table schema changed, queries broken
- Webhook authentication expired

Alert Rule Issues

- Alert rule disabled for "investigation" and never re-enabled
- Suppression window set too wide, catching honeypot activity
- KQL query referencing renamed table column
- Threshold set too high for honeypot traffic volume

The Diagnosis Depth Advantage

When a validation fails, the agent doesn't just say "broken." It investigates.

```
❌ FAIL: ssh-honeypot-prod — No incident created

Investigating ...

1. Checking raw telemetry:
   run_kql_query("Syslog
     | where Computer == '10.0.1.10'
     | take 5")
   → 0 results ← No logs arriving

2. Checking data connector:
   run_kql_query("_LogOperation
     | where Detail contains '10.0.1.10'")
   → Last ingestion: 18 days ago

3. Checking alert rule:
   get_alert_rule("SSH Brute Force Detected")
   → Status: Enabled, Last triggered: 19d ago
```

```
Root cause hypothesis:
  The syslog data connector for 10.0.1.10
  stopped sending data 18 days ago.
  Alert rule is healthy — no telemetry
  is arriving.
  Likely cause: syslog forwarder on host.

Recommended action:
  SSH to 10.0.1.10 and check:
  systemctl status rsyslog
```

Considerations

What to keep in mind

Safety Rails and Responsible Use

✅ Required Controls

- ART MCP server **scope-limited** to honeypot network segment
- Execution requires explicit `ART_EXECUTION_ENABLED=true`
- Sentinel MCP is **read-only** – no write operations
- All tool calls logged – full auditability
- Pipeline runs under a **dedicated service principal**
- Change control: canary scope config in version control

❌ Do Not

- Run ART MCP on production segments
- Point the pipeline at real systems "to see what happens"
- Use the same credentials as your production red team tooling
- Skip the telemetry wait – false negatives from timing
- Ignore cleanup commands in atomics – honeypots accumulate state

The pipeline is testing your detection layer, not attacking it. Scope it accordingly. The honeypot network is the attack surface. Keep it that way.

Takeaways

The Three Laws of Deception Validation

1. Every honeypot is assumed broken until proven otherwise. Configuration drift is inevitable. Validate continuously, not once at deploy time.

2. Attack simulation and detection validation belong in the same workflow. Separating them leaves a gap. The agent closes it by owning both sides of the loop.

3. A failure without a root cause is just noise. The agent's job isn't just to find failures. It's to diagnose them so they get fixed.

Get Started

Attack Engine

```
# Install ART MCP
claude mcp add art \
  uvx atomic-red-team-mcp

# With execution enabled
# (lab environments only)
claude mcp add art \
  -e ART_EXECUTION_ENABLED=true \
  uvx atomic-red-team-mcp
```

Detection Engine

```
# Install Sentinel MCP
claude mcp add sentinel \
  --url https://sentinel.microsoft.c

# Verify connectivity
"List my active alert rules
 in Sentinel"
```

Run the Pipeline

```
"You are a deception validator.
 For each honeypot in scope,
 execute its atomic test,
 wait 90s, then verify
 Sentinel detected it.
 Report pass/fail."
```

ART MCP: `github.com/cyberbuff/atomic-red-team-mcp` · `uvx atomic-red-team-mcp`

Blog: `cyberbuff.substack.com`

Trust No Trap

The agent always knocks. Make sure your traps are listening.

```
Your honeypots deserve zero trust.
Validate them like an attacker would.
Automate the knock.
```

Questions?