Mycelium Product Brief

1. Executive Summary

Mycelium is a modular, scriptable RF cyber operations platform designed for resilience testing and security assessment of modern wireless systems. It replaces disparate, inflexible tools with a single, integrated platform, enabling operators to automate complex test scenarios, accelerate discovery, and improve the resilience of critical RF-dependent systems. Built on a powerful **Condition-Action Directive Engine**, Mycelium provides an unparalleled ability to script intelligent, autonomous behaviors for RF cyber operations.

2. The Mycelium Advantage

Mycelium's architecture is built on three core principles: Modularity, Automation, and Extensibility.

- **Modular Plugin Architecture:** Mycelium is built on a radically modular foundation where *every* component is a dynamically loaded plugin. This includes not only Protocols, Modulations, and SDRs, but also CLI Commands, and even the Conditions and Actions used by the Directive Engine. This allows for unprecedented extensibility and rapid integration of new capabilities.
- **The Directive Engine:** At the heart of Mycelium is its scriptable Condition-Action-AfterAction engine. Operators create a stack of "Directives," where each directive pairs a **Condition** (e.g., "Have we received a Mode-S message from a specific aircraft?") with an **Action** (e.g., "Log the message to a file") and an optional **AfterAction** (e.g., "Stop the tool"). This enables complex, stateful logic to be described in simple, readable scripts.
- **Universal Hardware Support:** By leveraging the SoapySDR abstraction layer, Mycelium supports a vast range of SDR hardware out-of-the-box, from low-cost devices like the RTL-SDR to high-performance platforms.
- **Flexible Interfaces:** Mycelium supports multiple interaction modes to fit any workflow. An interactive CLI is available for hands-on operation, a Script Interpreter enables automated and repeatable mission playbooks, and a built-in MCP (Model Context Protocol) server allows AI agents and LLMs to programmatically control Mycelium for fully autonomous RF operations.

3. Core Concepts in Action: Example Scripts

The best way to understand the power of Mycelium is to see its Directive Engine in action. The following scripts can be run from the Mycelium command line or loaded from a file (`load -f <filename>`).

Example 1: Simple Surveillance & Logging **Goal:** Monitor the primary ACARS frequency and log every received message to a text file.

Concept: This script demonstrates a basic "always do this" workflow. The condition is `Always`, meaning the action is performed every time the directive runs.

Note: Before running, find your SDR's ID with the `sdr show` command.

```
# SCRIPT: acars_logger.myce
# DESCRIPTION: Creates a tool to monitor and log all ACARS traffic.

# 1. Create a tool named 'acars_rx' using the ACARS protocol.
tool create -p ACARS -n acars_rx

# 2. Configure the tool to use a specific SDR, tune to the
#    primary ACARS frequency, and use the standard AM:MSK modulation.
#    Replace <SDR_ID> with the actual ID of your SDR (e.g., soapy:0).
tool set -n acars_rx -r <SDR_ID> -f 129.125e6 -m AM:MSK

# 3. Create a directive to continuously receive data.
#    Condition: Always (always true)
#    Action: Receive (pull data from the SDR into the processing buffer)
directive create -t acars_rx -n receive_data -c Always -a Receive

# 4. Create a second directive to log any received messages.
#    Condition: Any (true if the buffer contains a valid message)
#    Action: Log (write messages in ASCII text format to a file)
directive create -t acars_rx -n log_messages -c Any -a Log
directive set -t acars_rx -n log_messages -a Log -f ./acars_log.txt -a

# 5. Start the tool.
execute
```

Example 2: Automated Selective Targeting **Goal:** Monitor all Mode-S traffic, but only start logging ACARS messages from a specific aircraft of interest (ICAO address `A8C27B`).

Concept: This script showcases conditional logic and cross-tool directive control. Both tools run simultaneously from the start, but the logger's directives are initially disabled, so it runs silently. When the `spotter` detects the target aircraft, its `Enable_Directive` AfterAction activates the logger's directives on-the-fly turning the passive logger into an active one without any restart.

```
# SCRIPT: target_tracker.myce
# DESCRIPTION: Watches for a specific aircraft and activates a logger when found.

# 1. Create a tool to listen for our target aircraft.
#    Replace <SDR_ID_1> and <SDR_ID_2> with the actual IDs of your SDRs.
tool create -p ModeS -n spotter
```

```
tool set -n spotter -r <SDR_ID_1> -f 1090e6 -m PPM

# 2. Create the ACARS logging tool.
tool create -p ACARS -n logger
tool set -n logger -r <SDR_ID_2> -f 129.125e6 -m AM:MSK

# 3. Create receive and log directives for the 'logger' tool, then disable
#    them so the logger runs silently until the target is found.
directive create -t logger -n receive_acars -c Always -a Receive
directive create -t logger -n log_acars -c Any -a Log
directive set -t logger -n log_acars -a Log -f ./target_A8C27B.log -a
directive disable -t logger -n receive_acars
directive disable -t logger -n log_acars

# 4. Create the 'spotter' directive and configure each component.
#    Condition: Check if a Mode-S message has the ICAO address 'A8C27B'.
#    Action: Print the message to the console.
#    AfterAction: Enable the logger's directives by name to begin
#                 capturing ACARS traffic.
directive create -t spotter -n find_target -c ModeS_ICAO_Equals -a Print -A
 ↪  Enable_Directive
directive set -t spotter -n find_target -c ModeS_ICAO_Equals -i A8C27B
directive set -t spotter -n find_target -A Enable_Directive -N receive_acars -N
 ↪  log_acars -T logger

# 5. Start both tools. The 'logger' will run silently until the target is
 ↪  spotted.
execute
```

Example 3: Combined Airspace Operations Node **Goal:** Run a full ACARS ground station that is also a network gateway, while simultaneously monitoring Mode-S and performing IFF interrogations.

Concept: This script highlights Mycelium's advanced capabilities by creating a complex "operations node." The ACARS tool acts as a full POA by starting background squitter and auto-acknowledgment services, and then enters a primary loop to act as a network-to-RF gateway. This demonstrates initialization, background services, and remote control, all running alongside other independent RF tools.

```
# SCRIPT: airspace_node.myce
# DESCRIPTION: A multi-tool script for advanced airspace operations.
# This example assumes three USRPs are connected, discovered via 'sdr show'.
```

```
# --- Tool 1: ACARS POA & Network Gateway ---

# 1a. Create a full-duplex ACARS tool with a USRP transceiver.
tool create -p ACARS -n acars_node
tool set -n acars_node -t <SDR_ID_1> -r <SDR_ID_1> -f 131.55e6 -m AM:MSK

# 1b. Create one-shot setup directives to start background services.
#     The 'Disable_Directive' AfterAction targets itself by index to ensure it
 ↪  only runs once.
directive create -t acars_node -n start_squitter -c Always -a
 ↪  ACARS_Squitter_Start -A Disable_Directive
directive set -t acars_node -n start_squitter -A Disable_Directive -i 0
directive create -t acars_node -n start_autoack -c Always -a ACARS_Auto_Ack_Start
 ↪  -A Disable_Directive
directive set -t acars_node -n start_autoack -A Disable_Directive -i 1

# 1c. Create the main operational loop: listen on a socket, then transmit.
#     This allows injecting data from a remote source into the live RF
 ↪  environment.
directive create -t acars_node -n listen_tcp -c Always -a TCP_Receive
directive set -t acars_node -n listen_tcp -a TCP_Receive -P 9001
directive create -t acars_node -n forward_rf -c Any -a Transmit


# --- Tool 2: Mode-S Surveillance Logger ---

# 2a. Create the Mode-S logging tool using a second USRP.
tool create -p ModeS -n modes_logger
tool set -n modes_logger -r <SDR_ID_2> -f 1090e6 -m PPM

# 2b. Create directives to receive and log valid Mode-S messages.
directive create -t modes_logger -n rx -c Always -a Receive
directive create -t modes_logger -n log -c ModeS_Valid -a Log
directive set -t modes_logger -n log -a Log -f ./modes_log.json -a


# --- Tool 3: IFF Interrogator ---

# 3a. Create the IFF interrogation tool using a third USRP.
tool create -p IFF -n iff_interrogator
tool set -n iff_interrogator -t <SDR_ID_3> -f 1030e6 -m PPM
```

```
# 3b. Create directives to periodically transmit a Mode 3/A interrogation.
directive create -t iff_interrogator -n interrog -c Always -a IFF_Interrogate
directive set -t iff_interrogator -n interrog -a IFF_Interrogate -m 3A
directive create -t iff_interrogator -n wait -c Always -a Delay
directive set -t iff_interrogator -n wait -a Delay -s 5


# --- Execution ---

# 4. Start all three tools to run simultaneously.
execute
```

Example 4: AI-Driven Adaptive Monitoring Goal: Create a fully autonomous ACARS transceiver controlled by an AI agent. The AI receives live decoded traffic for real-time analysis and can inject its own ACARS messages for transmission closing the loop from passive observer to active RF participant.

Concept: When launched with the `-I MCP` flag, Mycelium exposes its full API to any AI agent or LLM. `MCP_Output` streams decoded traffic to the AI. `MCP_Input` gives the AI a channel to push data back into the tool for transmission. Together, they enable a fully autonomous RF engagement loop with no human in the middle. The AI can also use the MCP API to dynamically reconfigure the session changing frequencies, adding tools, or stopping operations without a restart.

```
# SCRIPT: ai_transceiver.myce
# DESCRIPTION: AI-controlled ACARS transceiver. The AI receives live decoded
# traffic and can transmit its own ACARS messages in response  autonomously.
# Run with: ./Myce -I MCP ai_transceiver.mycelium

# 1. Create a full-duplex ACARS tool.
#    Replace <TX_SDR_ID> and <RX_SDR_ID> with your SDR IDs (or use a single
#    full-duplex device like a USRP for both TX and RX).
tool create -p ACARS -n acars_node
tool set -n acars_node -t <TX_SDR_ID> -r <RX_SDR_ID> -f 129.125e6 -m AM:MSK

# 2. Continuously receive ACARS traffic.
directive create -t acars_node -n rx -c Always -a Receive

# 3. Stream each decoded message to the AI agent.
#    The AI identifies aircraft of interest, flags anomalies, and builds
#    a real-time operational picture from the live message feed.
directive create -t acars_node -n to_ai -c Any -a MCP_Output
directive set -t acars_node -n to_ai -a MCP_Output -m "ACARS traffic"
```

```
# 4. Accept ACARS messages from the AI for transmission (non-blocking).
#    The AI can push message bytes into the tool at any time, enabling
#    autonomous RF responses without human intervention.
directive create -t acars_node -n from_ai -c Always -a MCP_Input
directive create -t acars_node -n tx -c Any -a Transmit

# 5. Execute. The AI now has full read/write access to the RF environment.
execute
```

Example 5: Custom Signal Processing Pipeline **Goal:** Receive raw 433 MHz ISM sensor packets, route them through a custom decoder script, and log the human-readable output.

Concept: Mycelium does not need built-in support for every proprietary sensor format. The `Exec` action lets operators drop in any external script as a processing stage, while `Send_To_Tool` / `Receive_From_Tool` provide a thread-safe queue between tools decoupling each stage so they run at their own pace. A single tool can fan out to multiple destinations, and a single logger can receive from multiple sources, illustrating how pipeline stages are independently reusable. This pattern works with any language or tool that reads from stdin and writes to stdout.

```
# SCRIPT: sensor_pipeline.myce
# DESCRIPTION: A three-stage pipeline that receives 433 MHz ISM sensor packets,
# decodes them with a custom script, and routes the output to a dedicated logger.

# --- Stage 1: ISM Receiver ---
tool create -p ISM_Generic -n ism_rx
tool set -n ism_rx -r <SDR_ID> -f 433.92e6 -m AM:ASK

# Receive raw ISM packets, forward to the decoder, and also send a copy of
# the raw bytes directly to the logger for a complete packet record.
directive create -t ism_rx -n rx -c Always -a Receive
directive create -t ism_rx -n to_decoder -c Any -a Send_To_Tool
directive set -t ism_rx -n to_decoder -a Send_To_Tool -T decoder
directive create -t ism_rx -n to_logger -c Any -a Send_To_Tool
directive set -t ism_rx -n to_logger -a Send_To_Tool -T logger


# --- Stage 2: Custom Decoder ---
# No SDR required  this tool only processes data forwarded from the receiver.
tool create -p Raw -n decoder

# Wait for a packet from the ISM receiver (blocking avoids a busy-wait loop).
directive create -t decoder -n recv -c Always -a Receive_From_Tool
directive set -t decoder -n recv -a Receive_From_Tool -b
```

```
# Pipe the raw packet bytes through an external decoder script.
# The script reads from stdin and writes the decoded sensor reading to stdout,
# replacing the buffer with human-readable output for the next directive.
directive create -t decoder -n decode -c Any -a Exec
directive set -t decoder -n decode -a Exec -C "./decode_sensor.py" -d

# Forward the decoded output to the logger stage.
directive create -t decoder -n forward -c Any -a Send_To_Tool
directive set -t decoder -n forward -a Send_To_Tool -T logger


# --- Stage 3: Logger ---
tool create -p Raw -n logger

# Wait for data from any upstream tool (blocking avoids a busy-wait loop).
# The logger's inbox receives both raw packets from Stage 1 and decoded
# readings from Stage 2, giving a complete record in a single file.
directive create -t logger -n recv -c Always -a Receive_From_Tool
directive set -t logger -n recv -a Receive_From_Tool -b

# Log whatever arrives  raw bytes or decoded output.
directive create -t logger -n log -c Any -a Log
directive set -t logger -n log -a Log -f ./sensor_readings.log -a


# Start all three stages simultaneously.
execute
```

4. Technical Specifications

Mycelium's modular nature means its capabilities are constantly expanding.

- **Core Framework (Production-Ready):**
  - **Interfaces:** Interactive CLI, Script Interpreter, MCP Server (for AI control).
  - **Directive Engine:** A scriptable Condition-Action-AfterAction engine where all Conditions, Actions, and AfterActions are themselves extensible plugins. This allows for the creation of highly specialized logic and integrations.
  - **Supported SDRs:** Universal support via SoapySDR, including HackRF, RTL-SDR, LimeSDR, PlutoSDR, USRP, and more. Also includes a suite of Virtual SDRs for offline development.
  - **Platform:** Linux. Distributed as native packages for all major Linux distribution families.

- **Available Mission Packs:**
  - **Aviation Pack:** Includes production-ready protocols for **ACARS**, **Mode-S**, and **ADS-B**. *Roadmap: VDL-M2 and HFDL support planned for future releases.*
  - **Tactical Pack:** Includes production-ready support for legacy **IFF** modes (**1, 2, 3/A, C**). *Roadmap: Link 16 and Link 4 support planned for future releases pending program support.*
  - **IoT Pack:** Includes a generic protocol for analyzing common **433 MHz ISM** band devices. *Roadmap: Z-Wave, Zigbee, Bluetooth LE, and Wi-Fi support planned for future releases.*
- **Available Modulations:**
  - **Passband:** AM, FM
  - **Baseband:** ASK, BFSK, BPSK, CPFSK, D8PSK, DQPSK, MSK, PPM

5. Licensing and Availability

Mycelium is a proprietary, commercial software product developed and owned by The Cyber Grove. It is available for purchase under flexible licensing terms designed to meet the needs of government and corporate partners.

6. About The Cyber Grove

The Cyber Grove is a Maryland-based small business founded on the mission to deliver world-class cybersecurity tools into the hands of appropriate cyber security teams across the United States.

Contact us to schedule a demonstration or discuss licensing options tailored to your program requirements.

Contact: brent.wood@thecybergrove.com **Website:** https://www.thecybergrove.com