Assignment 3 - CCI and Incident Response Supply Chain Compromise Response Plan CSE 4380

Group Thorin

April 11th, 2025

Members:	Obadah Al-Smadi
	Betim Hodza
	Elliot Mai
	Benjamin Niccum
	Nicholas Pratt
Instructor:	Trevor Bakker

Contents

1 Introduction **4**

2 Incident Response Plan **5**
2.1 Comprehensive Inventory Development 5
2.2 Supply Chain Component Tracing 7
2.3 Compromise Determination Criteria 8
2.4 Containment, Eradication, and Recovery Procedures 10

3 Future Prevention Strategy **12**
3.1 Continuous Monitoring System 12
3.2 Supplier Security Requirements 12
3.3 Verification and Validation Process 12
3.4 SBOM Maintenance Process 13
3.5 CI/CD Pipeline Integration 13
3.6 VEX Handling Process 14

4 Supplier Engagement Plan **15**
4.1 Cooperative Supplier Process 15
4.2 Non-Cooperative Supplier Strategy 15

List of Figures

List of Tables

1 Example Inventory Entry for GPS Module . . . 7
2 Language-Specific SBOM Strategies . . . 13

1 Introduction

We at AeroTech Industries have received news that a supply chain attack is suspected to affect a JSON software library that could have been used in one of our software stacks or by our third-party suppliers. We are aware of this issue now and our cybersecurity team is hard at work to determine if our technology was compromised. To do this, we are creating a plan to scan our components and identify which (if any) components are compromised. To be more thorough, we also will be looking at our software suppliers and seeing if any libraries they used were also compromised. Half of our suppliers are willing to help us scan the code base, but the other half say they cannot assist us, but we do have the binary for those software and will review as thoroughly as possible. Once we have found all that is compromised, we will contain and eradicate the compromise, and attempt to recover any lost data. Finally, we will give a thorough review on how this incident happened and how we can prevent this in the future. To do this, we will create a Software Bill Of Materials maintenance process, be more firm on our security requirements, have continuous monitoring of software dependencies, and integrate supply chain security controls with our security framework.

2 Incident Response Plan

2.1 Comprehensive Inventory Development

2.1.1 List of Instances JSONparser 3.2.1 is in Use

- The Intrusion Detection Systems (IDS) uses JSONparser 3.2.1 JSON has a prevalence in log formats for IDS's and the parser is good in embedded environments.
- The Electronic Speed Controller doesn't directly use it in use, but it's used in configuration.
- Mission Planning and Execution Software is using JSONparser 3.2.1 for execution logs.
- Sensor-Calculation Module is using the JSONparser 3.2.1 this sensor converts raw sensor data and prepares it for use in different modules (flight controller, telemetry module).
- The Telemetry module uses JSONparser 3.2.1 in some instances through different protocols in certain communication methods like LTE/5G, and RF.
- Authentication and Authorization mechanisms (web based) use JSONparser 3.2.1 this is because JSON structures the payload.
- The Command and Control Protocol in our communication software uses JSONparser 3.2.1, which makes it significantly easier to parse and process commands.
- Satellite Communication System uses JSONparser 3.2.1 mainly for data output (communications).
- The GPS module uses JSONparser 3.2.1 to set the GPS and contain position data.
- Ground Control Station software uses JSONparser 3.2.1 for parsing drone telemetry and for drone configuration.
- AeroTech's Mobile App also uses JSONparser 3.2.1 like the ground control station does.

2.1.2 Methodology for Identifying JSONparser 3.2.1 Usage

2.1.3 Static Analysis

Statically Analyze code base to find out if JSON parser 3.2.1 is in use

1. Dependency Audit:
   - Scan build files (e.g., CMake, Makefile, .env for python) and dependency manifests (e.g., package.json, requirements.txt) for references to JSONParser 3.2.1 or its source (e.g., GitHub, PyPI).
   - Check firmware/software images for linked libraries (e.g., ldd(or binstats) on binaries, strings for version tags like "JSONParser 3.2.1").
2. Code Base Inspection:
   - Utilize Static code base scanners like OWASP Dependency checker and ScanCode-Toolkit.
   - Not all Code base scanners might be supported for our languages (like ADA), we will get an intern to search code base for include and identify components using JSON libraries.

2.1.4 Dynamic Monitoring

Detecting anomalies during run time.

1. Network anomalies:
    - Sudden spikes in outbound traffic
    - Connections to unfamiliar IPs
    - exfiltration of data
2. System irregularities:
    - Unexpected Process Behavior: JSON related processes consuming extra CPU / Memory
    - Unexplained file changes in JSON Configurations.
    - Missing or altered logs, gaps in telemetry where it tries to hide activity.
    - Unexpected JSON input or data, either user side or drone messages back to Ground Control Station.
3. Behavioral red flags:
    - Abnormal Component Behavior.
    - Telemetry Discrepancies, where there's inconsistent or incorrect data from JSON outputs from components.
    - Front End Application UI Response or incorrect data issues on Ground Control Station.
4. Dynamic Tools
    - Using network sniffers (Wireshark) and system monitoring to pick up RF frequency over the air.
    - X9-Onboard Diagnostic tools to analyze live LTE/5G, and GNSS telemetry feeds.

2.1.5 Inventory Documentation Structure

The inventory documentation is structured as a hierarchical table, organized by system category to catalog all hardware, software, and third-party components of the X9 drone. This structure ensures traceability and supports supply chain risk assessment, particularly for identifying usage of libraries such as JSONParser 3.2.1.
Each component is documented with the following fields:

- **Component Name**: Official identifier (e.g., "Collins AeroSpace Navstrike").
- **Category**: Navigation & Control Hardware.
- **Description**: Functional purpose (e.g., "GPS-based positioning for navigation").
- **Supplier/Manufacturer**: Origin (e.g., "Collins Aerospace").
- **Version**: Hardware/software version (e.g., "v5 SAASM").
- **Dependencies**: Software/hardware dependencies (e.g., "JSONParser 3.2.1 for telemetry").
- **Integration Notes**: Connection to X9 (e.g., "Interfaces with VECTOR-600 via I2C").
- **Supply Chain Provenance**: Source details (e.g., "U.S.-sourced, certified").
- **Likelihood of JSONParser 3.2.1 Usage**: Risk rating (e.g., "Moderate").
- **Vulnerability Notes**: Supply chain risks (e.g., "Field-reprogrammable, verify updates").

Field	Details
Component Name	Collins AeroSpace Navstrike
Category	Navigation & Control Hardware
Description	Provides GPS-based positioning for navigation
Supplier/Manufacturer	Collins Aerospace
Version	v5 SAASM
Dependencies	JSONParser 3.2.1 (telemetry parsing)
Integration Notes	Interfaces with VECTOR-600 via I2C
Supply Chain Provenance	Sourced from U.S. distributor, certified
Likelihood of JSONParser 3.2.1 Usage	Moderate
Vulnerability Notes	Field-reprogrammable software, check for tampered updates

Table 1: Example Inventory Entry for GPS Module

2.2 Supply Chain Component Tracing

The objective of supply chain component tracking is to have a methodology to trace suppliers' components and map them to a compromised vulnerability (JSONparser 3.2.1 in this case). Afterwards we will approach our methodology of handling these vulnerabilities with cooperative, proprietary sensitive, and CTOS suppliers to help with compliance.

2.2.1 Component Origin Mapping

1. Supply Chain Mapping
    - Identify Suppliers: List all vendors and manufacturers for each component.
    - Trace Distribution Path: Document the steps that each of the components takes throughout assembly. (From the time they're purchased, shipped, to assembled.)
    - Record Metadata: Document details about the component such as country of origin and batch numbers. (This can also include anything digital in software making like GitHub commits on version of software.)
2. Integrity Verification
    - Hardware Checks: Verify hardware integrity through physical inspection of serial numbers, physical markings from manufacturers, or any signs of tampering.
    - Software Checks: Validate hashes of software libraries against official releases. Audit build pipelines for unauthorized updates if possible.
    - Third-Party Validation: Verify certificates of authenticity from suppliers, and cross check firmware versions with vendor-vendor provided baselines.
3. Compromise Detection
    - Static Analysis: Scan binaries for unexpected code behavior using disassembler, and SBOM dependency checkers.
    - Dynamic Testing: Monitor runtime behaviors of each component of the X9, and test components through controlled input.
    - Supply Chain Anomalies: Investigate supplier breaches and flag deviations in performance from suppliers.

2.2.2 Supplier Categorization and Handling

2.2.3 Cooperative Suppliers

3 suppliers willing to scan codebases but need detailed steps, tools, and outputs.

- Provide Detailed guide to scan codebase:
- Use SBOM software (e.g., OWASP Dependency-Check) to scan all code libraries used for dependencies.
- Check for any signs of JSONparser 3.2.1 from running SBOM scan results.
- Request all affected software files linked to JSONparser 3.2.1 (e.g. source files).
- Request suppliers update to a patched version (e.g., JSONparser 3.2.2, if available).
- Request ongoing use of SBOM tools to maintain dependency lists for future FOSS supply chain risks.

2.2.4 Proprietary Information

2 suppliers withholding library details, claiming proprietary information; They have provided binaries of their software only.

- Request binary hashes and version metadata if possible (e.g. file signatures) for verification of versions if supplier allows for it.
- Analyze binaries automatically with tools such as Binary Ninja and FOSSLight Binary Scanner.
- Analyze binaries with reverse-engineering tools if automated tools don't work. (e.g. Ghidra, Binwalk, IDA Pro) to detect JSONparser 3.2.1 library.
- Test Binaries in a sandbox w/ sample JSON inputs to monitor for anomalies known to occur with vulnerabilities with the parser.
- Request third-party audit reports or certifications under NDA, if possible with supplier.
- If JSONparser 3.2.1 detected, document evidence of library detection, request supplier for mitigation plans and provide a patched binary.

2.2.5 COTS Suppliers

1 Supplier claims that their Commercial Off-The-Shelf status makes them not obligated to assist in codebase scanning; they provide binaries only.

- Verify Binary authenticity by requesting suppliers to provide hashes or request the distributors of the software used.
- Analyze binaries automatically with tools (Binary Ninja and FOSSLight Binary Scanner).
- Analyze binaries manually with tools like ghidra.
- Test binaries in a sandbox environment and monitor for JSONparser 3.2.1 behaviors by inputting json exploits known with 3.2.1.
- If signs of JSONparser 3.2.1 detected, document findings and contact supplier and request for them to update and patch their codebase.

2.3 Compromise Determination Criteria

2.3.1 Technical Indicators of Compromise

- Integrity Failure: If thorough hardware and software testing for integrity fails, such as hash mismatches and or altered hardware, this is indicator for compromise.
- Behavioral Anomalies: From dynamic testing, if the component for the X9 has unexpected behavior (unexpected outputs, and or network anomalies), this is also a sign of compromise.

- Metadata Issues: If supplier metadata we obtain has a mismatch of component origin, batch number, or any digital hash verification on software libraries, this could be a sign of compromise. Probably best to contact the supplier to ensure that any changes are supposed to be valid or not.
- Threshold: A Component is deemed compromised if it meets one or more criteria across integrity, behavior, or metadata anomalies. These will be weighted by severity (e.g. hash mismatch == high, documentation mismatch = Low/Moderate).

2.3.2 Risk Assessment Framework

The Risk Assessment Framework evaluates the risk of compromised components in the AeroTech X9 supply chain, focusing on the JSONParser 3.2.1 supply chain attack.
It prioritizes response actions based on likelihood and impact, supporting the incident response plan.
Framework Steps:

1. **Component Identification**: Use the inventory (Section 2.1.2) to list components potentially using JSONParser 3.2.1, cross-referenced with supplier categories (Section 2.2.2).
2. **Risk Factor Assessment**: Assess supplier cooperation, JSONParser usage likelihood, verification gaps, and component criticality.
3. **Likelihood Scoring (1-5)**:
   - 1 (Low): Cooperative supplier, no JSONParser, verified hashes.
   - 3 (Moderate): Proprietary/COTS, possible JSONParser, partial verification.
   - 5 (High): Unverified binaries, confirmed JSONParser, no cooperation.
4. **Impact Scoring (1-5)**:
   - 1 (Low): Non-critical (e.g., payload mount), minimal impact.
   - 3 (Medium): Secondary (e.g., telemetry), moderate disruption.
   - 5 (High): Critical (e.g., navigation), mission failure or safety risk.
5. **Risk Calculation**: Risk Score = Likelihood $\times$ Impact (1-5).
   - Low (1): Monitor.
   - Moderate (3): Investigate, mitigate.
   - High (5): Urgent action.
6. **Action Prioritization**: High: Sandbox test, escalate to supplier, patch. Medium: Verify further. Low: Document, monitor.

Example Application: For the UAV Navigation VECTOR-600 Flight Controller (Proprietary Supplier):

- *Likelihood*: 4 (Proprietary, binary only, Moderate JSONParser likelihood).
- *Impact*: 5 (Critical for navigation).
- *Risk Score*: $4 \times 5 = 20$ (High).
- *Action*: Test in sandbox with JSON inputs, request hash, prioritize patching.

This framework integrates with the supplier categorization (Section 2.2.2) and component

2.4 Containment, Eradication, and Recovery Procedures

2.4.1 Containment Strategies

Containment strategies aim to limit the impact of the JSONParser 3.2.1 supply chain compromise on the AeroTech X9, preventing further exploitation while the scope is assessed.
Strategies:

- **Isolate Affected Components**: Quarantine X9 systems or subsystems with confirmed or suspected JSONParser 3.2.1 usage (e.g., flight controller, 5G receiver). Disconnect from external networks (e.g., disable 5G) and ground drones if operational risks are detected (Risk Score ≥ 16, per Section 2.2.2).
- **Restrict Network Access**: Block outbound traffic from components using JSONParser 3.2.1 (e.g., telemetry modules) via onboard firewalls to prevent data exfiltration. Limit inbound JSON commands to trusted ground control stations (GCS).
- **Disable Vulnerable Functions**: Temporarily disable JSON parsing in non-critical systems (e.g., camera telemetry on VEGA PTZ) by rerouting to binary protocols where possible, reducing attack surface.
- **Monitor Active Systems**: Deploy real-time monitoring (e.g., Wireshark on 5G/RF interfaces, and analyzing logs) to detect anomalous JSONParser behavior (e.g., unexpected IPs, malformed JSON), ensuring containment holds during investigation.
- **Supplier Coordination**: Notify Cooperative Suppliers to halt JSONParser-related updates, request Proprietary and COTS Suppliers to confirm binary status (e.g., hash matches), per Section 2.2.2 handling approaches.

Example: If the flight controller (High Risk, Score 20) uses JSONParser 3.2.1, AeroTech isolates it by disabling 5G comms, restricts GCS inputs to binary, and monitors logs for exploits, containing potential navigation hijacks.

2.4.2 Eradication Methods

Eradication methods focus on removing the compromised JSONParser 3.2.1 library and any associated malicious artifacts from the AeroTech X9 and supplier-provided components.
Methods:

- **Replace Compromised Software**: Update X9 software stacks (e.g., flight control, telemetry modules) to a patched JSONParser version (e.g., 3.2.2) or an alternative library (e.g., RapidJSON). For Cooperative Suppliers, request updated source/binaries; for Proprietary/COTS, demand patched binaries.
- **Reimage Affected Systems**: Wipe and reflash firmware on critical components with verified images from AeroTech's secure repository, removing any tampered JSONParser instances.
- **Remove Malicious Artifacts**: Scan and delete JSONParser-related files (e.g., "libjsonparser.so") or injected code identified via static analysis on X9 systems and supplier binaries, ensuring no potential attack surface is minimized.
- **Validate Supplier Components**: Require Cooperative Suppliers to resubmit SBOM scans post-update (per Section 2.2.2); analyze Proprietary/COTS binaries in a sandbox to confirm JSONParser removal or absence of exploits.
- **Verify Eradication**: Run dependency checks (e.g., OWASP Dependency-Check) and runtime tests (e.g., JSON input '"test": "data"") on all updated X9 components to ensure no JSONParser 3.2.1 or malicious behavior persists.

2.4.3 Recovery Procedures

Recovery procedures restore the AeroTech X9 to a secure operational state following the containment and eradication of the JSONParser 3.2.1 compromise, ensuring mission readiness.
Procedures:

- **Reintegrate Systems**: Reconnect isolated components (e.g., 5G receiver, flight controller) to the X9 network after installing verified software/firmware (e.g., patched JSONParser or alternatives), restoring full functionality.
- **Validate Functionality**: Conduct flight tests (e.g., hover, waypoint navigation) to confirm critical systems operate correctly post-update, using binary protocols where JSONParser was disabled.
- **Verify Security**: Test with safe JSON inputs (e.g., "{"mission": "survey"}") and monitor for anomalies (e.g., Wireshark, system logs) to ensure no residual exploits remain; re-run risk assessment (Section 2.2.2) targeting updated components.
- **Update Supplier Agreements**: Require Cooperative Suppliers to adopt patched libraries, request Proprietary/COTS Suppliers to certify updated binaries, aligning with future prevention.
- **Document Recovery**: Log all actions (e.g., "Flight Control Board reflashed April 10, 2025, tested clean") in the X9 incident report, updating the inventory (Section 2.1.2) with new versions and hashes for traceability.

3 Future Prevention Strategy

3.1 Continuous Monitoring System

3.1.1 Monitoring Tools and Infrastructure

To establish a robust continuous monitoring system:

- Deploy intrusion detection systems (IDS) and runtime application self-protection (RASP) to monitor for suspicious activity.
- Utilize dependency tracking tools like Dependency-Track or CycloneDX for real-time vulnerability alerts.
- Integrate monitoring dashboards with SIEM (Security Information and Event Management) platforms for centralized oversight.

3.1.2 Process Workflow

The workflow for continuous monitoring will include:

- Data Collection: Gather logs, telemetry, and dependency data from all systems.
- Analysis: Use automated tools to detect anomalies or vulnerabilities.
- Alerting: Notify relevant teams of potential threats or deviations.
- Remediation: Implement fixes or mitigation based on detected risks.

3.2 Supplier Security Requirements

3.2.1 Contractual Requirements

Supplier contracts will mandate:

- Adherence to NIST SP 800-161 security standards.
- Regular submission of SBOMs for transparency in software dependencies.
- Implementation of secure coding practices and vulnerability management programs.

3.2.2 Security Assessment Procedures

- Conducting periodic audits of supplier systems and processes.
- Requiring penetration testing reports to validate system resilience.
- Reviewing third-party certifications to ensure compliance with industry standards.

3.3 Verification and Validation Process

3.3.1 Pre-Integration Verification

Before integrating components into the system:

- Verify the authenticity of components using cryptographic signatures.
- Cross-check dependencies against known vulnerability databases.

3.3.2 Security Testing

- Performing static and dynamic analysis on components using tools like SonarQube or OWASP ZAP.
- Conducting fuzz testing to uncover hidden vulnerabilities in libraries.

3.3.3 Integration Validation

During integration:

- Validate system functionality under simulated attack scenarios.
- Ensure compatibility with existing security controls.

3.4 SBOM Maintenance Process

3.4.1 General Framework

The SBOM maintenance process

- Regular updates to reflect changes in software dependencies.
- Automation of SBOM generation during CI/CD pipeline builds using tools like SPDX or CycloneDX.

3.4.2 Language-Specific Strategies

Language	**SBOM Generation Tools**	**Special Considerations**	**Binary Analysis Approach**
C and C++	Dependency scanning tools	Address complex linking dependencies	Use disassembly tools for binaries
Python	Pip-audit, Safety	Monitor dynamic package installations	Analyze package metadata
Java	Maven Dependency Plugin	Handle transitive dependencies	Inspect JAR files for vulnerabilities
Ada	GNAT Dependency Analyzer	Ensure compliance with safety-critical standards	Perform manual inspections

Table 2: Language-Specific SBOM Strategies

3.4.3 Binary Analysis Methodology

To analyze binary files:

- Use reverse engineering tools like IDA Pro or Ghidra to inspect compiled code.
- Employ heuristic analysis to detect embedded vulnerabilities.

3.5 CI/CD Pipeline Integration

3.5.1 SBOM Generation in Build Process

Integrate SBOM generation into the CI/CD pipeline by:

- Automating dependency scans during each build cycle.
- Embedding SBOM validation checks as part of deployment gating criteria.

3.5.2 Supply Chain Attack Monitoring

Monitor for supply chain attacks by:

- Using behavioral analytics to detect unusual component activity during builds.
- Implementing anomaly detection algorithms in CI/CD workflows.

3.6 VEX Handling Process

3.6.1 Document Intake and Processing

The process for handling Vulnerability Exploitable Exchange (VEX) documents includes:

- Parsing VEX documents using standardized formats (e.g., JSON or XML).
- Storing processed data in a centralized repository for tracking exploitable status.

3.6.2 Integration with Security Systems

Integrate VEX handling with security systems by:

- Linking VEX data with SIEM platforms for automated threat correlation.
- Using VEX insights to prioritize patching efforts based on exploitable risks.

4 Supplier Engagement Plan

4.1 Cooperative Supplier Process

4.1.1 Required Scanning Tools

Our suppliers who are willing to cooperate with us will be required to use the following tools when scanning the codebase:

- **Syft and Grype**: For generating SBOMs and identifying vulnerabilities.
- **ClamAV**: To detect malware that could be within the codebase.
- **CodeQL**: For Static Application Security Testing.
- **Dependency Track**: To track vulnerabilities in third-party packages.

Our cooperative suppliers have to ensure that the versions of these scanning tools that they use are up to date with the latest stable version before applying them.

4.1.2 Step-by-Step Procedures

1. Conduct an internal SBOM generation with Syft.
2. Run Grype with the SBOM that was generated to find any known vulnerabilities.
3. Perform a Static Application Security Test analysis using CodeQL on the codebase.
4. Document any instance in which the JSONpasrer 3.2.1 library was used, or anything that may have had it as a dependency.
5. Share the detailed scan reports, SBOMs, and mitigation strategies that were generated through the process.

4.1.3 Expected Deliverables

Each supplier who chooses to cooperate must submit the following deliverables:

- SBOM in SPDX or CycloneDX format.
- Grype vulnerability scan output (JSON or PDF).
- SAST analysis report.
- Signed attestation confirming findings and actions taken.

4.2 Non-Cooperative Supplier Strategy

4.2.1 Proprietary Information Suppliers

For suppliers who refuse to cooperate or disclose because of proprietary claims, different measures including binary analysis will be taken:

- Perform binary analysis using acceptable tools such as Binwalk, Ghidra, or Radare2 to reverse-engineer library usage.
- Use strings and symbol extraction to detect the presence of JSONparser 3.2.1 or any dependencies.
- Analyze dynamic behavior in isolated VMs or containers (sandboxing) to safely find possible exploits.
- Request attestation that the library is not used, signed by a technical authority.

4.2.2 COTS Suppliers

For suppliers who refuse to cooperate or disclose because of a lack of contractual obligation or being a COTS software, the following measures will also be taken:

- Invoke SBOM analysis on the accessible binary using open-source binary SBOM tools like Ghidra.
- Compare binary signatures to known JSONparser 3.2.1 patterns to detect any matches.
- Record non-cooperation and escalate through procurement/legal for future contractual adjustments.