Supplier-Independent Capability Verification of Software and Firmware with AI-Driven ATT&CK Mapping

Mitchell Petingola
Research Assistant
School of Computing
Queen's University
Kingston, ON, Canada
mitchell.petingola@queensu.ca

Philippe Charland
Defence Scientist
Mission Critical Cyber Security Section
Defence Research and Development Canada
Quebec, QC, Canada
philippe.charland@drdc-rddc.gc.ca

Steven H. H. Ding
Assistant Professor
School of Information Studies
McGill University
Montreal, QC, Canada
steven.h.ding@mcgill.ca

Benjamin C. M. Fung
Professor
School of Information Studies
McGill University
Montreal, QC, Canada
ben.fung@mcgill.ca

Abstract: In defense and other security-critical domains, software, and firmware must perform only their mission-required functions. Hidden or dormant capabilities, such as unauthorized data transmission, cryptographic routines, or file manipulation, expand the attack surface because they can later be activated by attackers after deployment. The 2020 SolarWinds supply chain compromise demonstrated how trusted updates can be weaponized, with embedded code that remains inactive at first, but later enables movement through secure networks, while still being undetected. Relying solely on supplier attestations is therefore insufficient. Even reputable vendors may unknowingly deliver unnecessary or vulnerable code. To mitigate this, we advocate independent reverse engineering and capability verification to identify and limit what software and firmware are technically able to do. This verification gives commanders and certifiers confidence that the deployed systems remain secure, predictable, and trustworthy.

To systematically classify and communicate verified capabilities, the MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) framework provides a suitable taxonomy that is widely adopted across defense and intelligence communities as a common reference for adversary techniques. However, its existing behavior-scanning methods are limited: Dynamic analysis (sandboxing) only observes the code paths that happen to execute, missing latent or environment-dependent functions, while static analysis typically recognizes a limited number of ATT&CK techniques, leaving large blind spots. We address this gap by combining language models with Capa-style rule generation to produce fast, binary-level matching rules. This extends Capa, originally built for malware triage, into a broader verification tool that produces detailed, auditable reports. In experiments on 878 malware samples, our method expanded ATT&CK coverage from 148 to 262 techniques, including high-impact behaviors such as credential dumping and persistence via scheduled tasks. This broader mapping enables more robust independent capability verification, reduces compliance risks, and strengthens assurance for mission-critical defense and security-critical systems.

Keywords: *capability verification, large language models, mission assurance*, *MITRE ATT&CK, supply chain security*

1. Introduction

Modern defense, government, and other security-critical systems increasingly depend on complex software and firmware supplied by third-party vendors [1], [2]. These components are often deeply integrated into mission-critical platforms, ranging from command-and-control systems to embedded controllers and network devices. In such environments, security assurance is not limited to correctness or the absence of known vulnerabilities; it also requires confidence that the deployed software performs only its intended, mission-required functions. Any additional, unnecessary, or hidden capability (whether benign, legacy, or malicious) expands the attack surface and introduces unnecessary risk.

Recent supply chain compromises have demonstrated the severity of this problem. The SolarWinds incident, for example, showed how adversaries can embed dormant capabilities into trusted software updates that initially appear benign, pass standard validation, and remain inactive for extended periods [3]. Once deployed at scale, these capabilities can later be activated to enable espionage, lateral movement, or persistence within highly secured networks. Such behavior may not be observable during pre-deployment testing or short-term sandboxing, yet the underlying capability to perform it already exists in the binary.

Traditional software assurance relies on supplier attestations, certifications, and vulnerability disclosures. While necessary, these measures are insufficient on their own. Vendors may unintentionally ship unused libraries, debug logic, or overly permissive subsystems that introduce capabilities beyond mission requirements. In high-assurance environments, trust must be reinforced through independent technical verification. Reverse engineering and capability analysis can help bound what software can do, independent of vendor intent or stated functionality.

A major challenge in capability verification is how to systematically classify, communicate, and reason about discovered capabilities in a way that is understandable to analysts, engineers, and decision-makers. Low-level reverse-engineered evidence, such as application programming interface (API) calls, cryptographic primitives, or file system operations, is precise but difficult to aggregate into higher-level security implications. A standardized taxonomy is therefore required to bridge the gap between binary-level evidence and operationally meaningful conclusions.

The MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) framework offers a taxonomy for such high-level capabilities and has become a de facto standard across defense, intelligence, and industry [4]. However, ATT&CK is most commonly applied in incident response and threat hunting contexts, where it is used to label observed adversary behavior. Its application to capability verification, which determines what software *could* do rather than what it *has* done, remains limited. Existing dynamic analysis methods often miss latent behaviors, while static analysis methods leave large portions of the ATT&CK matrix unmapped.

To address this gap, we propose an AI-driven approach that combines large language models (LLMs) with Capa-style rule generation to automatically extend binary-level detection of ATT&CK techniques [5]. By generating fast, interpretable rules that operate directly on compiled software and firmware, our method extends ATT&CK mapping to enable independent assessment of software capabilities.

A. MITRE ATT&CK Framework

MITRE ATT&CK is a globally adopted knowledge base that categorizes adversary behavior [4]. Organized as a matrix, ATT&CK defines high-level tactics such as initial access, persistence, credential access, and command and control, and maps them to specific techniques and sub-techniques that describe how those objectives are achieved—for example, access via local security authority subsystem service (LSASS) memory. Each technique is supported by descriptions, examples, detection ideas, and references to known threat groups and malware (e.g., Mimikatz for credential dumping). Abstracting low-level technical actions into standardized techniques allows analysts, defenders, and decision-makers to reason about threats at an operational level.

Despite its widespread use, ATT&CK was not originally designed as a static capability taxonomy. Most mappings are derived from dynamic observations: sandbox executions, endpoint telemetry, network logs, or incident response artifacts. As a result, ATT&CK coverage is limited by the behavior triggered during execution. Techniques that depend on specific environmental conditions, user interaction, command and control activation, or dormant logic may never be observed, even though the software contains the necessary functionality.

Static analysis can reveal these latent capabilities, but existing ATT&CK-aligned static tools recognize only a subset of techniques, often focused on well-known indicators. These typically include easily identifiable artifacts such as suspicious API imports or hard-coded URLs and strings. Many ATT&CK techniques, particularly those involving nuanced logic or higher-level intent, lack reliable static signatures. This creates blind spots that are unacceptable in high-assurance settings, where the absence of evidence cannot be equated with evidence of absence.

In this work, ATT&CK is repurposed as a capability classification framework. Rather than asking whether a technique has been observed from executing a sample, we ask whether it contains the technical building blocks required to perform that technique. This repurposing aligns ATT&CK with mission assurance goals and drives the need for improved static, binary-level coverage.

B. Capa Analysis Tool

Capa is an open-source binary analysis tool developed by Mandiant to identify high-level capabilities in executable files [5]. Unlike traditional signature-based malware detection,

Capa uses a rule-based approach to infer semantic behaviors from low-level features such as API calls, strings, control-flow patterns, and constants. Originally, Capa was developed for malware triage and reverse engineering support. Given a suspicious binary, Capa can rapidly identify capabilities such as file manipulation, network communication, process injection, or encryption usage. These findings help analysts prioritize samples and focus manual analysis where it is most needed. Capa's rule engine is efficient, allowing it to scale to large datasets without the overhead of dynamic execution. Figure 1 shows a Capa rule for System Shutdown (T1529).

FIGURE 1: CAPA RULE FOR SYSTEM SHUTDOWN (T1529)

```yaml
rule:
  meta:
    name: shutdown system
    namespace: host-interaction/os
    authors:
      - michael.hunhoff@mandiant.com
    scopes:
      static: function
      dynamic: call
    att&ck:
      - Impact::System Shutdown/Reboot [T1529]
    examples:
      - 39C05B15E9834AC93F206BC114D0A00C357C888DB567BA8F5345DA0529CBED41:0x10008D60
  features:
    - or:
      - api: user32.ExitWindowsEx
      - api: user32.ExitWindows
      - api: advapi32.InitiateSystemShutdownEx
      - api: advapi32.InitiateSystemShutdown
      - api: advapi32.InitiateShutdown
```

However, Capa's coverage is limited by the availability and quality of its rules. While it includes mappings to some ATT&CK techniques, this mapping is incomplete and skewed toward common malware behaviors. Writing high-quality Capa rules is labor-intensive and requires deep reverse engineering expertise, which has constrained the growth of its ruleset. As a result, many ATT&CK techniques remain unsupported.

Despite these limitations, Capa provides an ideal foundation for capability verification. Its static, binary-level operation aligns with the need to detect latent functionality, and its rule-based design supports transparency and auditability. By augmenting Capa with AI-assisted rule generation, it becomes possible to systematically expand ATT&CK coverage while preserving interpretability and performance.

This paper builds on Capa's strengths and addresses its limitations by using LLMs to generate new, ATT&CK-aligned rules informed by real malware evidence and publicly available knowledge. The result is a scalable approach to supplier-independent capability verification

that bridges the gap between low-level binary analysis and high-level mission assurance requirements.

2. Related Work

While Capa's symbolic approach provides transparency and analyst trust, it relies almost entirely on manually authored rules. The process of rule creation and maintenance is a major bottleneck, limiting coverage and adaptability as malware evolves. This motivates approaches that can automate or assist rule generation, while preserving Capa's explainable and transparent rule-based design.

Existing work on generating rules using LLMs for Capa and similar analysis tools has shown promise. While focused on vulnerability detection and asset management rather than capability verification, ERS0 integrates the Capa engine into an AI-driven analysis pipeline and explores the use of LLMs to generate Capa rules aligned with MITRE ATT&CK techniques [6]. Rule generation in ERS0 is driven by a retrieval-augmented generation (RAG) pipeline that uses Atomic Red Team as its main knowledge source [7], incorporating technique descriptions and example attack implementations to guide the selection of APIs, system calls, and constants. While this approach enables scalable and systematic rule bootstrapping, Atomic Red Team examples are intentionally minimal and technique-centric, therefore, they may not fully capture the diversity and implementation-specific characteristics of real-world software. This observation motivates complementary approaches that expand the ATT&CK technique coverage of Capa rules based on sandbox indicators from real-world malware.

A related research area explores the automatic generation of Yet Another Recursive Acronym (YARA) and similar signature-based rules beyond Capa. Early pre-LLM approaches such as AutoYara used statistical feature mining and clustering over strings or byte patterns to derive candidate YARA rules [8]. These methods demonstrated partial automation but were limited by shallow semantics and poor generalization. More recent work leverages LLMs to generate YARA and intrusion detection system (IDS) rules from malware samples, threat reports, or cyber threat intelligence [9], [10], [11]. Related research has also applied automated rule generation to vulnerability patch verification, using synthesized detection logic to reason about whether security fixes have been correctly applied [12]. While these efforts establish the feasibility of automated and LLM-driven rule generation, they do not target semantic rule systems such as Capa or explicit ATT&CK technique definitions.

Beyond automated rule generation, prior work has explored deep-learning approaches for classifying ATT&CK techniques directly from software [13], [14], [15]. These methods generally achieve better performance by learning complex features from large datasets. However, they typically operate as a black box, offering limited insight into the specific program behaviors or features that contribute to a prediction. This lack of interpretability

poses a significant limitation in mission-critical and high-assurance settings, where analysts must be able to inspect, justify, and audit detections.

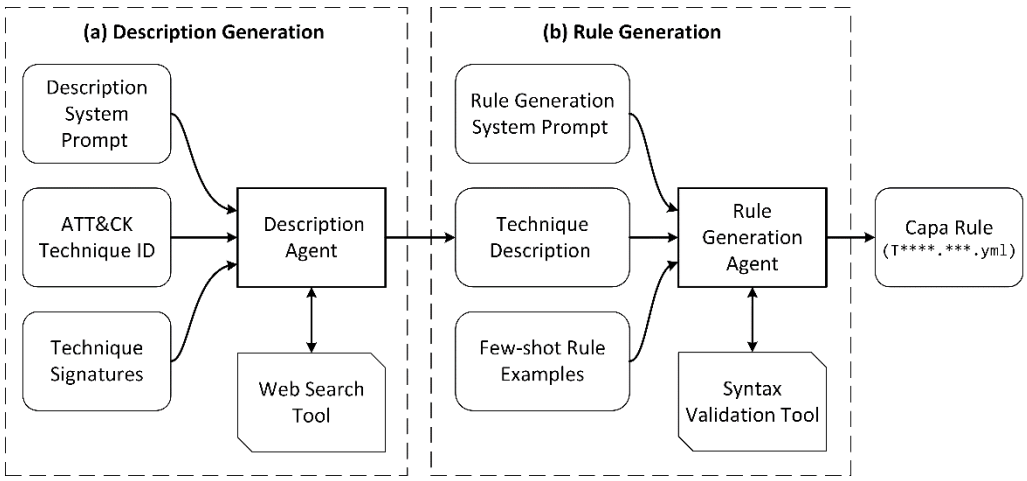
3. Methodology

The methodology for this system is built on a multi-step, agentic pipeline. The architecture uses two distinct LLM agents operating within a reasoning and acting (ReAct) framework [16]. This setup enables the agents to reason about potential technique indicators and to iteratively refine their outputs using external tools, specifically web search and a syntax validation tool, for verification and correction. The high-level goal is to take raw evidence from malware executions and transform it into a set of precise, semantically meaningful rules that can identify those same behaviors statically in any unknown piece of software.

As shown in Figure 2, the workflow begins by feeding the first agent with a system prompt, a specific MITRE ATT&CK technique ID, and a set of ATT&CK indicators for that technique, which are obtained from Falcon Sandbox analysis reports via the Hybrid Analysis API [17]. This initial stage is focused on knowledge synthesis, gathering all the indicators that point toward a specific software capability from both sandbox reports and documentation obtained from a web search tool. The second stage then takes the gathered intelligence and hands it off to a second agent, which acts as a specialized coder. This agent's job is to translate the technical findings in natural language into the specific YAML Ain't Markup Language (YAML) syntax and format used by Capa.

We explicitly separate the natural-language evidence synthesis stage from the formal rule generation stage, treating them as two distinct steps in the pipeline. This separation allows each stage to be guided by independent system prompts and model parameters (e.g., temperature) and provides transparency into the intermediate reasoning and evidence used to justify each detection.

FIGURE 2: PIPELINE FOR AUTOMATED RULE GENERATION



A. Description Generation

The primary knowledge base for our approach is derived from existing ATT&CK technique matches obtained through dynamic analysis, specifically the Falcon Sandbox [18]. Using the Hybrid Analysis API, we queried malware samples associated with individual ATT&CK techniques and collected technique-specific signatures. This process yielded a corpus of 298 unique ATT&CK techniques and sub-techniques, with the relevant information extracted into the JavaScript Object Notation (JSON) format. An example of a signature for technique T1003 is presented in Figure 3.

FIGURE 3: SIGNATURE EXAMPLE FOR TECHNIQUE T1003


```
{
  "name": "Attempts to invoke APIs commonly associated with credential theft and data exfiltration functionality",
  "description": "\"ARAD.exe\" called \"NtQueryInformationToken\" with parameters (UID: 00000000-00005256)",
  "attck_id": "T1003",
  "attck_id_wiki": "https://attack.mitre.org/techniques/T1003"
}
```


A major challenge is that many sandbox-reported indicators reflect purely dynamic behavior and environment-dependent execution, whereas our analysis framework, Capa, operates only on static binaries. Consequently, some sandbox indicators cannot be used directly.

For each ATT&CK technique, the agent is provided with the technique identifier and a minimum of eight signature examples derived from the sandbox execution of multiple malware samples. These indicators ground each technique in concrete, real-world implementations, while maintaining a bounded and consistent context for rule generation. The agent then synthesizes this information along with external sources such as official MITRE ATT&CK documentation, Atomic Red Team implementations [6], and API reference material, to infer static implementation patterns that are consistent with the observed behavior.

The agent is guided by a system prompt that explicitly constrains its output to statically observable evidence, including API usage patterns, characteristic strings, registry interactions, and other artifacts that can be extracted from binaries without execution. The resulting output is a structured natural-language report organized into four sections: (1) a high-level summary of the ATT&CK technique; (2) contextualization of indicators observed in sandbox reports; (3) detailed static indicators and implementation patterns suitable for rule construction; and (4) final notes to guide rule authorship. This description synthesis process is formalized in Algorithm 1. The DescriptionAgent starts by initializing an empty memory (lines 1–2) and enters a continuous reasoning loop (line 3). At each iteration, it selects the next action using a ReAct-style step that considers the system prompt, technique ID, signatures, and accumulated memory (line 4). If the chosen action is a web search, the agent queries the search tool and stores the result in memory (lines 5–6). If the action signals completion, the agent outputs the final technique description and terminates (lines 7–8), while a failure action immediately returns $\perp$ (lines 9–10). Any other intermediate action is

simply added to memory (lines 11–12), and the loop repeats until a finalization or failure condition is reached (line 14).

These descriptions function as an intermediate representation between dynamic behavioral observations and static detection logic. They are used to guide the generation of Capa rules, ensuring that rule generation is grounded in both real malware behavior and statically verifiable evidence, while maintaining interpretability and auditability throughout the process.

Algorithm 1 DESCRIPTIONAGENT

Require: WebSearchTool
Require: S_t (signatures), t (technique id), *SystemPrompt*
Ensure: Technique description D_t or $\perp$

```
1: procedure DescriptionAgent(S_t, t, SystemPrompt)
2: mem ← ∅
3: loop
4:     action ← ReActStep(SystemPrompt, t, S_t, mem)
5:     if action.type = Search then
6:         mem ← mem ∪ {WebSearchTool(action.query)}
7:     else if action.type = Finalize then
8:         return action.output
9:     else if action.type = Fail then
10:        return ⊥
11:    else
12:        mem ← mem ∪ {action}
13:    end if
14: end loop
```

B. Rule Generation

In the final stage, the structured natural-language technique descriptions are used to generate properly formatted Capa rules. A second system prompt positions the language model as an expert malware analyst and Capa rule author, with the explicit objective of translating natural-language descriptions into valid, ATT&CK-aligned YAML rules. The prompt constrains the model to produce rules that adhere strictly to Capa's syntax, schema, and naming conventions, ensuring they can be immediately deployed.

The rule generation workflow proceeds iteratively. The model reviews the technique description, selects appropriate static indicators, and combines them using meaningful logical constructs, including nested AND and OR conditions, to balance coverage and specificity. Few-shot examples drawn from existing high-quality Capa rules are included in the system prompt to anchor the generation process in established best practices for rule scope, feature selection, and ATT&CK annotation.

A key component of this stage is automated validation. The model has access to a Capa validation tool that executes syntax and compatibility checks against the generated YAML. Validation output is returned to the model, which corrects formatting, schema, or legacy

issues before retesting. This loop continues until the rule passes validation without errors. Only the final, validated YAML rule is emitted and written to the ruleset under a namespace derived from the ATT&CK tactic and parent technique. This rule generation process is formalized in Algorithm 2.

Algorithm 2 RuleAgent

Require: CapaValidationTool
Require: D_t (description), t (technique id), *SystemPrompt*
Ensure: Validated rule R_t or $\perp$

1: **procedure** RuleAgent($D_t, t, SystemPrompt$)
2: $mem \leftarrow \emptyset$
3: **loop**
4: $action \leftarrow$ ReActStep($D_t, t, SystemPrompt, mem$)
5: **if** $action$.type = ProposeRule **then**
6: $r \leftarrow action$.output
7: $(ok, \delta) \leftarrow$ CapaValidationTool(r)
8: **if** ok **then**
9: **return** r
10: **else**
11: $mem \leftarrow mem \cup \{\delta\}$
12: **end if**
13: **else if** $action$.type = Fail **then**
14: **return** $\perp$
15: **else**
16: $mem \leftarrow mem \cup \{action\}$
17: **end if**
18: **end loop**

The RuleAgent begins by initializing an empty memory (lines 1–2) and enters a continuous loop (line 3). At each iteration, it selects an action using a ReAct-style step based on the technique description, technique ID, system prompt, and current memory (line 4). When the action is to propose a rule, the agent validates the proposed rule using the capability validation tool (lines 5–7). If the validation succeeds, it immediately returns the rule (lines 8–9). Otherwise, it stores the validation feedback in memory to refine future proposals (lines 10–12). If the agent instead produces a failure action, it returns $\perp$ (lines 13–14). All other intermediate actions are appended to memory (lines 15–16), and the loop continues until a valid rule is generated or failure occurs (line 18).

Algorithm 3 defines the end-to-end pipeline of the rule generation process across all ATT&CK techniques. It iterates over techniques, invokes the description and rule agents, and aggregates successfully validated rules into the final ruleset.

Algorithm 3 GENERATERULESET

Require: HybridAnalysisAPI
Require: $\mathcal{T}$ (technique list), MaxSignatures, DescriptionAgentPrompt, RuleAgentPrompt
Ensure: Ruleset $\mathcal{R}$

1: **procedure** GENERATERULESET($\mathcal{T}$)
2: $\mathcal{R} \leftarrow \emptyset$
3: **for all** $t \in \mathcal{T}$ **do**
4: $S_t \leftarrow \text{HYBRIDANALYSISAPI}(t)$
5: $S_t \leftarrow \text{SELECTATMOST}(S_t, MaxSignatures)$
6: **if** $S_t = \emptyset$ **then**
7: **continue**
8: **end if**
9: $D_t \leftarrow \text{DESCRIPTIONAGENT}(S_t, t, DescriptionAgentPrompt)$
10: **if** $D_t = \perp$ **then**
11: **continue**
12: **end if**
13: $R_t \leftarrow \text{RULEAGENT}(D_t, t, RuleAgentPrompt)$
14: **if** $R_t \neq \perp$ **then**
15: $\mathcal{R} \leftarrow \mathcal{R} \cup \{R_t\}$
16: **end if**
17: **end for**
18: **return** $\mathcal{R}$

The GenerateRuleSet procedure initializes an empty ruleset $\mathcal{R}$ (lines 1–2) and iterates over each technique t in the input list (line 3). For each technique, it extracts behavioral signatures using the Hybrid Analysis API and limits them to a maximum number (lines 4–5). If no signatures are returned, the technique is skipped (lines 6–8). It then invokes the DescriptionAgent to produce a technique description and skips the technique if this step fails (lines 9–12). Next, the RuleAgent is called to generate a validated detection rule from the description (line 13), and any successfully validated rule is added to the global ruleset (lines 14–16). After all techniques are processed, the complete ruleset is returned (lines 17–18).

4. Experimental Evaluation

A. Model Selection

We apply the methodology using four foundational language models: GPT 5 Mini, GPT 5, Gemini 3 Flash Preview, and Gemini 2.5 Pro. These models were selected to compare across providers (OpenAI and Google) and model sizes. Each model is used within the same agentic pipeline described in Section 3, including identical prompts, tools, and validation procedures. This ensures that observed differences in rule quality can be attributed to the capabilities of the underlying foundation models.

For all experiments, the temperature for both the description generation agent and the rule generation agent is fixed at 0.5. To bound computational cost and prevent unbounded agent behavior, we cap the maximum number of agent iterations at 30 and enforce a maximum

execution time of 3 minutes per agent invocation. Executions that fail to produce a valid, syntactically correct Capa rule are rejected.

B. Dataset

The evaluation dataset was constructed by aggregating malware samples from two complementary sources: Hybrid Analysis and the official Capa test dataset.

First, malware samples were identified using Hybrid Analysis by querying for each ATT&CK technique for which at least one Capa rule exists. Using the VirusShare repository [19], up to 50 samples were retrieved for each technique, resulting in approximately 7,000 unique binaries. This initial collection contained a substantial number of droppers and unpackers. To control analysis cost and reduce noise, several filtering criteria were applied. Samples were discarded if their file size was smaller than 100 KB or larger than 10 MB, if they contained more than ten imported functions (a common characteristic of droppers and loaders), or if they were not valid Portable Executable (PE) or Executable and Linkable Format (ELF) binaries. After filtering, 493 binaries remained. For each of these samples, the corresponding Hybrid Analysis reports were obtained and used to derive ATT&CK technique labels at the sub-technique level based on observed behavioral indicators.

To complement this dataset and ensure coverage of well-studied binaries with established rule behavior, the existing Capa test dataset was included. This corpus contains 385 binaries spanning both malicious and benign software and is commonly used to test manually authored Capa rules.

Combining both sources resulted in a final dataset of 878 binaries. This mixed dataset balances real-world malware observed in the wild with curated test samples, enabling evaluation of rule agreement across diverse software.

C. Rule Agreement

To evaluate the quality of LLM-generated Capa rules, we assess their agreement with existing label sources using macro-averaged precision, recall, F1 score, and mean Jaccard Index across the dataset. Macro-averaged precision, recall, and F1 score are computed independently for each technique i, and then averaged to ensure each technique is weighted equally:

$$P_i = \frac{TP_i}{TP_i + FP_i} \quad \bar{P} = \frac{1}{N}\sum_{i=1}^{N} P_i \quad R_i = \frac{TP_i}{TP_i + FN_i} \quad \bar{R} = \frac{1}{N}\sum_{i=1}^{N} R_i$$

$$F1_i = \frac{2 \times P_i \times R_i}{P_i + R_i} \quad \overline{F1} = \frac{1}{N}\sum_{i=1}^{N} F1_i$$

where N is the number of techniques being evaluated.

The Jaccard Index measures the overlap between the predicted techniques in the sets A_j and B_j for each sample j, and is then averaged across the dataset of size M:

$$J(A_j, B_j) = \frac{|A_j \cap B_j|}{|A_j \cup B_j|} \quad \bar{J} = \frac{1}{M}\sum_{j=1}^{M} J(A_j, B_j)$$

These metrics summarize agreement by averaging over techniques (precision, recall, and F1 score) and over samples (Jaccard Index).

To simplify our evaluation, we consider all sub-techniques equal to their parent technique. However, these sub-technique rules will still provide more specific capability information in real-world workflows.

We focus on two main label sources for our evaluation. First, we compare our rules against the existing handwritten Capa ruleset, evaluating whether LLM-generated rules identify the same ATT&CK techniques as expert-authored rules when applied to the same binaries. One limitation of this comparison is that we are restricted to techniques already supported by Capa, since no static reference labels exist for unsupported techniques.

We also evaluate static–dynamic agreement by comparing our rules against techniques inferred from Falcon Sandbox behavioral reports. Unlike the static reference, sandbox analysis provides broader technique coverage, but identifies only behaviors observed during execution, which may be incomplete. For this reason, recall is emphasized for sandbox agreement, while precision and F1 score are reported for completeness.

To summarize the overall agreement, the observed techniques from both sources (existing Capa rules and Falcon Sandbox) will be merged and compared against our LLM-generated rules. This will provide a summary of the agreement across both label sources. Due to the limited coverage of static label sources and the incompleteness of dynamic label sources, the reported metrics should only be interpreted as measures of agreement, rather than direct indicators of rule correctness.

While the validation tool described in Section 3 provides the agent with the ability to iteratively improve and fix rule syntax, some rules are not successfully generated before the agent's execution times out. To provide fair agreement measurements, we only consider techniques that exist in both the label source and the specific model’s ruleset. Additionally, some techniques only occur a limited number of times within our dataset. To ensure we are evaluating rules on samples with enough variation, we only evaluate rules where the corresponding technique is implemented in five or more samples. The number of rules evaluated for each model and evaluation configuration is reported for transparency.

D. Per-Technique Evaluation

Beyond evaluating the overall ruleset produced by each model, we assess the quality of individual technique detections in isolation. This enables an analysis of the difficulty level of generating rules for each technique type. Consistent with the agreement-based evaluation, only techniques occurring more than five times in the dataset are considered. This analysis is restricted to the top-performing model using the merged labels from existing Capa rules and Falcon Sandbox, as the focus is on cross-technique performance rather than differences between LLMs.

5. Results

A. Syntactic Correctness

Table I summarizes the size of the rulesets produced by each model. Both OpenAI models (GPT 5 Mini and GPT 5) generated 262 syntactically valid Capa rules, demonstrating more consistent rule generation and validation success. Google's models produced slightly smaller rule sets, with Gemini 3 Flash Preview generating 248 rules and Gemini 2.5 Pro generating 233 rules. As previously mentioned, the existing handwritten Capa ruleset contains 148 rules.

Overall, these results indicate that the OpenAI models were more reliable at producing syntactically correct and executable rules within the agent's time budget, whereas Google's models showed a slight drop in successful rule generation.

TABLE I: RULESET SIZE BY MODEL

Provider	**Model**	**Techniques**	**Sub-Techniques (Rules)**
OpenAI	GPT 5 Mini	119	262
	GPT 5	120	262
Google	Gemini 3 Flash Preview	115	248
	Gemini 2.5 Pro	114	233
Mandiant	Existing Capa Rules	81	148

B. Rule Agreement

Table II reports rule agreement metrics across the three evaluation configurations. Agreement with the existing Capa ruleset (a) varies across models. Gemini 2.5 Pro achieves the highest macro-averaged precision (0.5834), recall (0.5692), F1 score (0.5094), and mean Jaccard Index (0.4195), followed by Gemini 3 Flash Preview. The OpenAI models achieve lower macro-averaged recall and F1 scores, with GPT 5 Mini demonstrating slightly higher

precision and recall than GPT 5, despite being a smaller model. The Falcon Sandbox baseline shows substantially lower recall and overlap under the static agreement evaluation.

Macro-averaged recall against Falcon Sandbox behavioral labels (b) ranges from 0.1511 to 0.1861 across LLMs. Mean Jaccard indices are uniformly low. The baseline Capa rules show similarly low agreement in this setting.

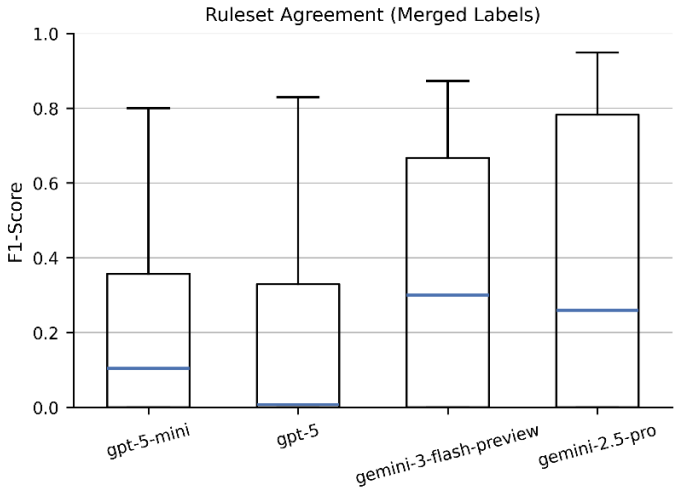
Under the merged-label evaluation (c), macro-averaged recall ranges from 0.2646 to 0.4168, with corresponding F1 scores between 0.2484 and 0.3642. Gemini 2.5 Pro and Gemini 3 Flash Preview again achieve the highest recall, F1 score, and Jaccard Index, while the OpenAI models report lower but comparable values. Figure 4 presents per-technique F1 scores for the merged-label evaluation, summarizing the distribution across each ruleset.

TABLE II: RULE AGREEMENT ACROSS STATIC, DYNAMIC, AND MERGED-LABEL SOURCES

Model	**Technical Evaluation**	**Precision**	**Recall**	**F1 Score**	**Jaccard**
(a) Existing Capa Rules (Static Agreement)					
GPT 5 Mini	40	0.5050	0.3625	0.3228	0.1720
GPT 5	42	0.4505	0.3281	0.3056	0.2440
Gemini 3 Flash Preview	43	0.5543	**0.5770**	0.4949	0.3204
Gemini 2.5 Pro	44	**0.5834**	0.5692	**0.5094**	**0.4195**
Falcon Sandbox (baseline)	38	0.2961	0.0109	0.0179	0.0080
(b) Falcon Sandbox (Dynamic Agreement)					
GPT 5 Mini	27	0.0639	0.1628	0.0627	**0.0281**
GPT 5	27	**0.0728**	0.1652	**0.0782**	0.0249
Gemini 3 Flash Preview	37	0.0705	0.1511	0.0447	0.0102
Gemini 2.5 Pro	37	0.0326	**0.1861**	0.0386	0.0136
Capa Rules (baseline)	23	0.0570	0.1268	0.0331	0.0080
(c) Existing Capa Rules + Falcon Sandbox (Merged Labels)					
GPT 5 Mini	52	0.3812	0.2742	0.2484	0.1594
GPT 5	55	0.3694	0.2646	0.2512	0.2125

Gemini 3 Flash Preview	56	0.4200	0.3883	0.3489	0.2800
Gemini 2.5 Pro	57	**0.4307**	**0.4168**	**0.3642**	**0.3695**

FIGURE 4: F1 SCORE DISTRIBUTION UNDER THE MERGED-LABEL EVALUATION



C. Per-Technique Evaluation

Table III reports the ten ATT&CK techniques for which Gemini 2.5 Pro achieved the highest per-technique performance under the merged-label evaluation. Several techniques demonstrate perfect scores across all three metrics, including Query Registry (T1012) and System Shutdown/Reboot (T1529), each achieving precision, recall, and an F1 score of 1.0 over 139 and 98 occurrences, respectively. Other frequently occurring techniques, such as Shared Modules (T1129) and Obfuscated Files or Information (T1027), also show strong performance, with F1 scores above 0.86 despite substantially higher support. Across the top ten techniques, coverage spans 7 of the 14 ATT&CK tactics, including discovery, execution, collection, impact, defense evasion, privilege escalation, and credential access.

Focusing on the rules that demonstrate the poorest performance, 19 techniques had zero true positives across the evaluation dataset. Among these, techniques associated with the defense evasion tactic were the most prevalent, including techniques such as reflective code loading (T1620) and masquerading (T1036). It is worth noting that these techniques have substantially lower support than the top-performing techniques in Table III, indicating that they are infrequently observed both in sandbox execution and by existing Capa rules.

TABLE III: TOP 10 ATT&CK TECHNIQUES DETECTED BY GEMINI 2.5 PRO, ORDERED BY F1 SCORE

#	ID	Technique	Tactic(s)	Occur.	Prec.	Recall	F1
1	T1012	Query registry	Discovery	139	1.00	1.00	1.00

2	T1529	System shutdown/reboot	Impact	98	1.00	1.00	1.00
3	T1115	Clipboard data	Collection	60	1.00	0.97	0.98
4	T1129	Shared modules	Execution	248	1.00	0.90	0.95
5	T1033	System owner/user discovery	Discovery	44	0.98	0.91	0.94
6	T1134	Access token manipulation	Defense evasion, privilege escalation	115	0.96	0.91	0.94
7	T1056	Input capture	Collection, credential access	35	1.00	0.83	0.91
8	T1569	System services	Execution	22	0.95	0.82	0.88
9	T1222	File and directory permissions modification	Defense evasion, privilege escalation	107	0.85	0.91	0.87
10	T1027	Obfuscated files or information	Defense evasion	213	0.77	1.00	0.87

6. Backdoor Case Study

To evaluate zero-day backdoor detection in a trusted supply chain context, we conducted a case study in which the SUNBURST backdoor, having the characteristics listed in Table IV, was embedded into a legitimate automotive software package. The resulting binary preserved all normal application functionality, mimicking a realistic vendor compromise scenario, in which a signed update channel distributes a covert implant. The modified package was analyzed using five capability rulesets from Table II. All detections were performed statically with identical extraction settings. No malware family labels or indicators of compromise (IoC) were provided to the rule generators in the previous steps, modeling a true zero-day condition. These results are reported in Table V.

TABLE IV: EXPECTED BACKDOOR CHARACTERISTICS

Category	**Description**
Covert C2	DNS and/or HTTPS-based command and control masquerading as telemetry

Data encoding	Base64 or custom encoding for exfiltration and staging
Anti-analysis	Virtual machine/sandbox detection, delayed execution
Discovery	Process, system, and geolocation enumeration
Obfuscation	Dynamic API resolution, XOR decode loops, packed payloads
Privilege handling	Token manipulation, access rights modification
Masquerading	Signed binary abuse, deceptive file extensions
Persistence prep	Service or system process modification

As expected, the implant delays execution to evade sandbox-based dynamic analysis, then establishes covert DNS/HTTPS command and control, encodes host profiling data in Base64, performs environment discovery and anti-VM checks, applies obfuscation techniques such as dynamic API resolution, and manipulates privileges, all while masquerading inside trusted binaries to persist undetected within the automotive software supply chain.

The human-authored Capa rules primarily detected benign dual-use behaviors, such as registry access, geolocation queries, and virtualization checks. While technically accurate, these rules reflect an implicit trust model: The host application is assumed legitimate, so potentially dangerous primitives such as encoding, command and control–style web usage, token manipulation, and loader behavior are intentionally ignored to avoid false positives. In a supply chain compromise scenario, this assumption collapses, causing the backdoor to blend seamlessly into the baseline functionality of the trusted application.

In contrast, the AI-generated rules consistently identified intent-driven primitives rather than application context. Across all models, independent detection of covert C2 logic, encoding behaviors, obfuscation techniques, privilege handling, and masquerading artifacts emerged without prior knowledge of SUNBURST. Although no single model achieved full coverage, the collective AI rulesets reconstructed the complete operational profile of the backdoor. This demonstrates that behavior-first, ATT&CK-quantified detection fundamentally outperforms traditional trust-anchored capability rules for zero-day supply chain threats.

TABLE V: DETECTION RESULTS COMPARISON

Capability Class	**Human Capa**	**Gemini 2.5 Pro**	**Gemini 3 Flash**	**GPT 5 Mini**	**GPT 5**
Covert Web C2	No	Yes	No	No	No
DNS / data encoding	No	Yes	Yes	No	No

Anti-VM / sandbox	Yes	No	No	No	No
Process discovery	No	No	Yes	No	No
Dynamic API resolution	No	No	Yes	No	No
XOR / custom decode loops	No	No	No	Yes	No
Privilege / token manipulation	No	Yes	No	No	Yes
Masquerading / extension abuse	No	No	No	Yes	Yes
Signed Binary Proxy Execution	No	No	No	No	Yes
Create / Modify System Process	No	No	No	Yes	No

7. Discussion

Across all evaluations, LLM-generated rules demonstrate measurable agreement with existing static and dynamic reference labels. While absolute agreement levels vary by model and label source, several patterns emerge.

First, agreement is highest when evaluating generated rules against existing Capa rules, reflecting the shared static analysis basis of both approaches. Lower agreement against Falcon Sandbox labels is expected, as sandbox-derived techniques depend on executed behaviors and environmental conditions that may not be triggered during dynamic analysis. In this context, disagreement should not be interpreted as incorrect detection, but rather as evidence of differing observability between static and dynamic analysis.

Second, although agreement values are limited, the LLM-generated rules consistently achieve higher agreement scores than the corresponding baselines in both static–static and static–dynamic comparisons. This indicates that the generated rules capture additional capability-relevant indicators beyond those present in the existing static rulesets or dynamic observations alone.

Third, model-specific behavior reveals a clear trade-off between rule generation reliability and agreement quality. Both GPT 5 Mini and GPT 5 produce syntactically valid rules more consistently within the agentic pipeline, reducing validation failures and manual intervention. In contrast, Gemini 3 Flash Preview and Gemini 2.5 Pro generate rules that, when valid, exhibit stronger agreement with existing labels. This suggests that rule quality and rule generation robustness are partially decoupled, and it may be worth investigating using different models at different stages of the agentic pipeline.

Finally, per-technique analysis shows that high agreement is concentrated among techniques with clearer static indicators and higher support, while techniques with low support frequently yield zero detections. This pattern reflects limitations in the available evidence rather than a systematic failure of the generation approach.

A. Operational Impact and Considerations

From an operational perspective, the results support the use of AI-assisted rule generation as a scalable approach for expanding ATT&CK coverage in static analysis workflows. Even where agreement with existing labels is imperfect, the generated rules provide interpretable artifacts that can be incorporated into existing tooling.

Existing Capa rules are intentionally narrow, targeting specific low-level behaviors or code patterns that are subsequently annotated with ATT&CK techniques. This design promotes precision and interpretability but often requires multiple rules to collectively represent a single technique and may result in incomplete coverage. In contrast, the proposed approach generates full rules explicitly at the ATT&CK technique and sub-technique levels, combining multiple static indicators into a single rule. This enables more direct reasoning about technique presence and simplifies downstream ATT&CK-aligned capability assessment. However, technique-level rules operate at a higher level of abstraction and may be more sensitive to variation across implementations. As a result, while this approach can improve coverage and expressiveness, it may trade fine-grained specificity for broader, high-level verification.

Importantly, rules that demonstrate lower agreement should not be dismissed as unusable. In many cases, these rules capture partial or weak indicators that may not be sufficient for reliable automated detection but still offer valuable starting points for expert refinement. These rules can accelerate rule development by finding candidate features and narrowing selection during manual analysis.

The disagreement between static and dynamic detections further highlights the value of static capability assessment in mission-critical environments. Dynamic analysis provides high-confidence evidence of executed behavior but offers limited guarantees about dormant or conditionally executed capabilities. Static, ATT&CK-aligned rules complement this by identifying what software could do, supporting more conservative and defensible assurance decisions when execution-based evidence is incomplete or unavailable.

B. Limitations and Future Work

Several limitations constrain the interpretation of our results. First, all evaluations rely on imperfect reference labels. Existing Capa rules represent expert knowledge, but do not cover the full ATT&CK matrix, while sandbox-derived labels reflect only behaviors exercised during execution. Agreement metrics only quantify overlap with these sources, rather than their correctness with respect to true software capability.

Second, techniques with very low occurrence counts are excluded from evaluation to ensure statistical stability. This underrepresents rare techniques, some of which may be operationally significant, despite limited prevalence in available data.

Additionally, integrating human-in-the-loop workflows for reviewing and refining generated rules could improve trust and adoption in high-assurance environments. Rather than replacing expert analysis, AI-assisted rule generation should be viewed as an approach to accelerate capability verification, while preserving interpretability and accountability.

8. Conclusion

This work investigated the feasibility of supplier-independent capability verification through AI-driven generation of ATT&CK-aligned static detection rules. By generating technique-level Capa rules using LLMs and evaluating them against established static and dynamic reference labels, we assessed whether automated rule generation can meaningfully expand the coverage of security-relevant software capabilities.

The results show that LLM-generated rules achieve measurable agreement with existing expert-authored rules and execution-based labels, while consistently outperforming baseline comparisons in both static–static and static–dynamic evaluations. Although agreement with execution-based labels was limited, this likely indicates differences in observability between static and dynamic analysis, rather than limitations in the generated rules, as supported by similar agreement metrics on expert-authored Capa rules.

Beyond quantitative metrics, the generated rules provide interpretable artifacts that can be directly integrated into existing static analysis workflows. Even where agreement is low, these rules offer valuable starting points for expert refinement, accelerating the development of high-confidence detections. Overall, this work demonstrates that AI-driven ATT&CK mapping can enhance capability-based software assurance by supporting defensible verification of software and firmware in security-critical environments.

Acknowledgments

This research is supported by the Vector Scholarship in Artificial Intelligence, provided by the Vector Institute.

REFERENCES

[1] J. Boyens et al., “Cybersecurity supply chain risk management for systems and organizations,” National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-161r1, May 2022. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/161/r1/upd1/final

[2] H. Ghanbari, K. Koskinen, and Y. Wei, “From SolarWinds to Kaseya: The rise of supply chain attacks in a digital world,” *Journal of Information Technology Teaching Cases*, Nov. 2024.

[3] E. D. Wolff et al., “Navigating the SolarWinds supply chain attack,” *The Procurement Lawyer*, vol. 56, no. 2, 2021.

[4] MITRE Corporation, “MITRE ATT&CK.” Accessed: Jan. 11, 2026. [Online]. Available: https://attack.mitre.org/

[5] Mandiant, “GitHub—mandiant/capa: The FLARE team’s open-source tool to identify capabilities in executable files.” Accessed: Jan. 11, 2026. [Online]. Available: https://github.com/mandiant/capa

[6] M. Beninger et al., “ERS0: Enhancing military cybersecurity with AI-driven SBOM for firmware vulnerability detection and asset management,” in *Proceedings of the 16th International Conference on Cyber Conflict: Over the Horizon (CyCon)*, Tallinn, Estonia, May 2024, pp. 141–160.

[7] Red Canary, “Atomic Red Team.” Accessed: Jan. 11, 2026. [Online]. Available: https://atomicredteam.io/

[8] E. Raff et al., “Automatic YARA rule generation using biclustering,” in *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, Virtual Event, USA, Nov. 2020, pp. 71–82.

[9] X. Zhang et al., “Automatically generating rules of malicious software packages via large language model,” 2025, *arXiv:2504.17198*.

[10] S. Mitra et al., “FALCON: Autonomous cyber threat intelligence mining with LLMs for IDS rule generation,” 2025, *arXiv:2508.18684*.

[11] X. Wang et al., “LLMDYara: LLMs-driven automated YARA rules generation with explainable file features and DNAHash.” Accessed: Jan. 11, 2026. [Online]. Available: https://i.blackhat.com/BH-USA-25/Presentations/USA-25-Wang-LLMDYara-LLMs-Driven-Automated-YARA.pdf

[12] S. S. Antar et al., “Vulnerability patch verification for military software systems through AI-driven code-level rule generation,” in *Proceedings of the 17th International Conference on Cyber Conflict: The Next Step (CyCon)*, Tallinn, Estonia, May 2025, pp. 189–208.

[13] J. Fairbanks et al., “Identifying ATT&CK tactics in Android malware control flow graph through graph representation learning and interpretability,” in *Proceedings of the IEEE International Conference on Big Data*, Orlando, FL, Dec. 2021, pp. 5602–5608.

[14] H. Sun et al., “Malware2ATT&CK: A sophisticated model for mapping malware to ATT&CK techniques,” *Computers & Security*, vol. 140, May 2024.

[15] S. Vasan et al., “DEEPCAPA: Identifying malicious capabilities in Windows malware,” in *Proceedings of the 2024 Annual Computer Security Applications Conference*, Honolulu, HI, Dec. 2024, pp. 826–842.

[16] S. Yao et al., “ReAct: Synergizing reasoning and acting in language models,” 2023, *arXiv:2210.03629*.

[17] CrowdStrike, “Free automated malware analysis service—powered by Falcon Sandbox.” Accessed: Jan. 11, 2026. [Online]. Available: https://www.hybrid-analysis.com

[18] CrowdStrike, “Malware analysis | CrowdStrike Falcon® threat intelligence” Accessed: Jan. 11, 2026. [Online]. Available: https://www.crowdstrike.com/en-us/platform/threat-intelligence/malware-analysis/

[19] VirusShare, “VirusShare.com.” [Online]. Available: https://virusshare.com