悪性パッケージ検出を目的としたSupervisor型LLMエージェントによるPythonパッケージの自律的な段階的解析

戸田 宇亮[1,2,a)]　若井 琢朗[1]　森 達哉[1,2,3]

概要：PyPI 等のパッケージレジストリでは，多段階の攻撃チェーンを持つ悪性パッケージが問題となっている．既存の自動検出手法は特定の悪性パターンを捉えることはできるものの，攻撃の全体像や最終意図といった高次のコンテキスト解明には至っていない．本稿では，この課題に対し，人間の専門家の分析を模倣した Supervisor-Worker 型 LLM マルチエージェントを提案する．本手法は，LLM の弱点を責務を分離した専門 Worker と動的な Plan-and-Execute ワークフローで構造的に補い，信頼性の高い段階的解析を実現する．300 件のパッケージを用いた評価では再現率 100%，適合率 99.3%を達成した．さらに，専門家によるユーザスタディでは，悪性パッケージ分析における実用的な品質を確認すると同時に，良性パッケージへの過剰警告の抑制や，専門家の役割に応じた別種の情報提供の必要性といった，人間と LLM ベースのセキュリティ解析エージェントが協働するための課題も明らかにした．

Autonomous Stepwise Analysis of Python Packages Using Supervisor-Worker LLM Agents for Malicious Package Detection

Takaaki Toda[1,2,a)]　Takuro Wakai[1]　Tatsuya Mori[1,2,3]

Abstract: Package registries like PyPI are increasingly threatened by malicious packages with multi-stage attack chains that existing automated methods fail to fully contextualize. To address this gap, we propose a Supervisor-Worker LLM multi-agent system that mimics human expert analysis. Our architecture ensures reliable, stepwise analysis by structurally compensating for LLM weaknesses (e.g., hallucination) through specialized Worker agents and a dynamic Plan-and-Execute workflow. A quantitative evaluation on 300 packages demonstrated high detection performance (100% recall, 99.3% precision). Furthermore, a user study with three security experts confirmed the reports' practical quality for threat analysis. It also revealed key challenges for human-AI collaboration in security, such as the need to suppress excessive warnings for benign packages and to provide distinct types of information tailored to specific expert roles.

1. はじめに

Python 言語の PyPI に代表されるソフトウェアパッケージレジストリは，他人のソースコードをパッケージとして再利用することが常識となった現代のソフトウェア開発にとって欠かせないインフラである．しかし，その利便性の裏でサプライチェーン攻撃の温床ともなっており，難読化やリモートペイロードの動的取得といった多段階の攻撃チェーンを駆使する悪性パッケージが深刻な問題となっている．この脅威に対し，既存研究では静的・動的解析や LLM を補助的に用いた検出手法が提案されてきた．これらの手法は特定の悪性パターンを高精度に検出できるものの，その説明性については不審なコード片の指摘に留まり，難読化を能動的に解除して複数ステップにまたがる攻撃の全体像や攻撃者の最終的な意図までを自動的に解明するアプローチは未だ確立されていない．

この課題を解決するため，本稿では人間の専門家が持つマルウェア解析プロセスを模倣した Supervisor-Worker 型マルチエージェントを提案する．これは，LLM が持つハル

[1] 早稲田大学 / Waseda University
[2] 理化学研究所 革新知能統合研究センター / RIKEN AIP
[3] 情報通信研究機構 / NICT
[a)] todatakaaki@nsl.cs.waseda.ac.jp

シネーションや文脈の混同といった内在的な弱点を，責務を分離した専門 Worker エージェントと，動的な Plan-and-Execute ワークフローによって構造的に補うよう設計されている．これにより，単一の LLM では達成困難な安定した段階的解析を実現する．本稿の貢献は次の通りである．

- Python パッケージの段階的解析に特化した，Supervisor-Worker 型マルチエージェントアーキテクチャを設計・実装した．
- 300 件のパッケージを用いた評価により，再現率 100%，適合率 99.3%という高い検出性能を実証した．さらに，実際の解析トレースを通じて，エージェントが動的に計画を修正しながら多段階の脅威を解明するプロセスを確認した．
- 3 名のセキュリティ専門家を対象としたアンケートを実施し，本エージェントが生成する自動解析レポートが悪性パッケージの脅威分析において実用的な品質を持つことを示すと同時に，良性パッケージに対する過剰な警告や，専門家の役割に応じた情報提供の必要性といった，次なる研究課題を特定した．

2. 背景と関連研究

2.1 PyPI のエコシステムと悪性パッケージの脅威

Python が機械学習や Web 開発に多用されるようになるにつれて，その主要なパッケージレジストリである PyPI 上のパッケージ数は増加の一途をたどり，現在では 60 万件近いパッケージが配布されている．PyPI は主に `pip` 等のユーティリティから使用されるが，パッケージのアップロードについても誰でも容易に行うことができ，今日の迅速なソフトウェア開発に欠かせない基盤となっている．

しかし，PyPI の利用の容易さを悪用して，ユーザに危害をもたらす悪意のあるパッケージがアップロードされる例があとを絶たない．2024 年 3 月には PyPI に対して有名なパッケージ名との TypoSquatting を狙った情報摂取マルウェアが短時間で大量にアップロードされ [1]，PyPI が新規パッケージの受付を一時停止する事態に追い込まれた．さらに，同年 11 月には AWS の認証情報を窃取するパッケージが発見され [2]，後にこれが 2021 年から PyPI 上に存在し続けており実に 37,100 回もダウンロードされていたことがわかった．また，より複雑な事例として，当初は良性であったパッケージがアップデートにより悪性化した例 [3] があり，現在は良性パッケージであってもそれは恒久的に良性であることを意味しない．

このような現状を受けて PyPI は Project Quarantine [4] などを通してマルウェアの報告受理から隔離までのプロセスを洗練させているが，検出の自動化が重要な課題として残っていることが PyPI 自身からも述べられている．また，実際にパッケージレジストリの運用者にアンケートを取った先行研究 [5] は，既存の自動検出手法の適用が難しい理由として偽陽性率が高すぎることを挙げている．

2.2 ケーススタディ：悪性 Python パッケージの実例

本節では，PyPI 上で過去に実際に配布されていた悪性パッケージから抜粋したコードを紹介する．プログラム 1 は ethereim というパッケージのバージョン 1.0.0 の `setup.py` ファイルの一部である（表示の都合上，長い文字列については途中を `(OMIT)` で置き換えている）．

```
class ynSmToSfOIO(OMIT)cgJswETrR(install):
  def run(self):
    import os
    if os.name == "nt":
      import requests
      from fernet import Fernet
      exec(
        Fernet(b'e2_3wb(OMIT)8Adk_rY=')
        .decrypt(b'gAAAA(OMIT)7OZwvw=')
      )
    install.run(self)
```

プログラム 1: 悪性パッケージの例（ethereim-1.0.0）インストール時には `run` 関数が自動的に呼び出される．

ユーザが本パッケージをインストールする際に `run` 関数が動作し，関数内では Windows 環境上でのみ難読化コードが復号され実行される．復号後の実際のコードは `exec(requests.get("https://funcaptcha.ru/paste2?package=ethereim").text)` であり，ダウンロードした文字列をコードとして実行している．この文字列こそ悪性パッケージの配布者が真に実行させたいコードであり，`run` 関数のコードはダウンローダであることがわかる．以上のように，PyPI 上で配布される悪性パッケージには複数段階を経て最終的なコードを実行するものが多く，パッケージ内のコードをそのまま読んでも最終的な意図はわからないことがある．パッケージの自動的な解析にあたり，このような特性を考慮する必要がある．

2.3 Python 悪性パッケージの自動検出に関する既存研究

Python 等のインタプリタ言語で作成された悪性パッケージを検出する方法論の学術研究は盛んに行われており，それらは静的解析のみに基づくもの・動的解析を取り入れたもの・そして LLM を用いるものに分類できる．

純粋な静的解析に基づく方法． Sun ら [6] はソースコードから生成したコールグラフとパッケージのメタデータを組み合わせた特徴を用いて，Zhang ら [7] はコールグラフを静的なルールで自然言語の説明に変換したデータを用いて訓練した BERT による検出手法を提案している．

静的解析は，一般に動作速度や要求リソースの点で効率的であるが動的に構築される攻撃コードには対応しにくい．また，昨今は LLM を用いることで正常に見えるメタデー

タを容易に作れるため，Sun ら [6] のようにメタデータを特徴として用いる方法の有効性が限定的になりつつある．

動的解析を取り入れた方法． 動的解析を取り入れた手法は比較的少ない．Zheng ら [8] は不審なパッケージのインストール，インポート，関数実行を行ってシステムコール等のトレースを取得し，ルールベースでパッケージの悪性を判定している．Huang ら [9] は静的解析と動的解析を併用しており，まずは静的解析を行って不審な場合には動的解析を行うことで検出精度と検査速度を両立している．

動的解析は難読化などの表面的な隠蔽に影響されない点で優れるが，リソースの制約と巧妙な回避技術（時間差で発火するものなど）により，すべてのプラットフォーム依存・環境依存のコードを網羅的に検出できるわけではない．

LLM を取り入れた方法とその課題． 近年 LLM を悪性パッケージ検出に適用する研究が出現しているが，その主な応用はコールグラフからの高リスク API の特定 [10] やコードからの静的特徴抽出 [11] といった，特徴量生成の補助に留まっている．これらの手法は従来研究より高い説明性を実現するものの，その説明性は不審なコード片の特定のような判定根拠の説明に限定されており，難読化の解除やリモートペイロードの取得といった能動的な解析を行い，攻撃者の最終的な意図や完全な攻撃チェーン，そして IoC (Indicator of Compromise) までを自動的に解明するアプローチは未だ確立されていない．これらの課題を解決する上で，単なる LLM とは異なり，自律的な計画立案とツール使用能力を持つ「LLM エージェント」が有望である．LLM エージェントは，ツールを用いて Web 検索やコード実行を動的に行えるため，既存研究では困難であったコードに対する能動的な解析を実現できる可能性がある．このような特性は先述のようなアプローチの実現に不可欠であり，もし実現されれば事後的な検出だけでなく防御側の予防的対策にも大きく貢献すると考えられる．

3. 設計と実装

3.1 全体的な設計

本研究で提案する LLM エージェントの全体像を図 1 に示す．このエージェントの設計は，人間のマルウェア解析者の認知プロセスに基づいている．先行研究 [12], [13] によれば，プロの解析者は，コード全体を俯瞰し仮説を立てるマクロな視点 (exploratory activities) と，難読化解除のような特定の作業に集中するミクロな視点 (exploitative activities) を動的に切り替えながら分析を進めることが知られている．本研究では，この認知プロセスを再現するために 2 つの主要な設計パターンを導入した．第一に，視点の切り替えを模倣するため，Supervisor-Worker 型マルチエージェントアーキテクチャを採用した．これは，解析全体の計画立案を担う単一の Supervisor と，特定ドメインの具体的なタスクを担う複数の Worker で構成される．第二

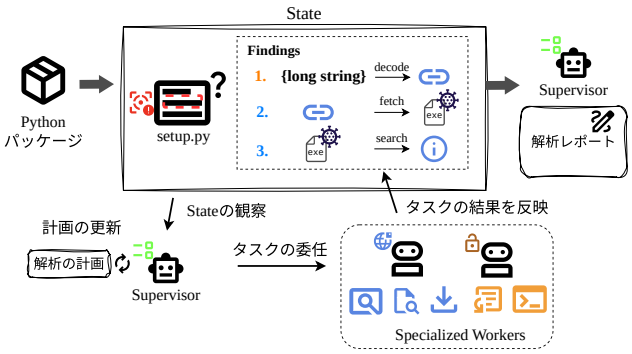

図 1: Supervisor-Worker マルチエージェントと Plan-and-Execute ワークフローを用いる本研究のエージェントのアーキテクチャ．

に，マルウェア解析が本質的に持つ不確実性に対応するため，Plan-and-Execute [14] ワークフローを採用した．難読化解除やリモートペイロード取得といったタスクは，実行するまで結果が予測できない．そのため，各ステップの結果に基づき Supervisor が動的に計画を再立案することで，解析プロセス全体の頑健性を確保する．

3.2 マルチエージェント内の各エージェント

Supervisor． 本研究の LLM エージェントにおいて，Supervisor はそれまでの解析で得られた情報をもとに Worker が遂行すべきタスクの計画を作成して順番に委任する．そして，これを何度か繰り返したのち最終的に複数の Worker からの応答を統合してレポートを生成する．重要な点として，Supervisor はタスクの目的や概要を計画として提示するが，具体的な実行手順までは必ずしも指定しない．これにより，Supervisor は Worker のツールの内部仕様や実装詳細に依存せずに計画を作成できる．

Workers． Worker は特定のドメインにおけるタスクを実際に遂行する役割を担う．各々の Worker はシステムプロンプトを通して明確な役割を与えられており，自らの責任範囲を理解した上で課された作業のみを行う．Worker の分離方法はソフトウェア工学における関心の分離の原則に従う．本研究では，Worker として次の 2 つを定義する．

- Deobfuscator: コード中の難読化箇所の復号を行う
- Web Researcher: Web アクセスおよび関連サービスによる情報収集を行う

3.3 LLM の制約への対処．

本研究で提案する設計は，LLM エージェントが持つ内在的な弱点を緩和し，信頼性を向上させる上でも重要である．

まず，Supervisor-Worker 型マルチエージェントアーキテクチャは，各 Worker のコンテキストを分離してハルシネーションの影響を局所化する．ある Worker でハルシネーションが起きても，その影響は当該 Worker の 1 セッション内に留まる．また，Supervisor が各 Worker の出力

をレビューすることで，誤りを検出する機会が生まれる．これは，単一コンテキストを持つエージェントでハルシネーションが起きるとそれ以降の全ての文脈が汚染される Context Poisoning [15] という現象を防ぐ上で有効である．

さらに，各エージェントの責務を単純に保つことで，LLM の性能を安定化する．最近の研究では，LLM に必要最小限のツールを与えた場合には解ける問題でもツールを多く与えすぎると解けなくなる現象 [16] や，入力が長くなると商用 LLM でさえモデルの性能が著しく低下する Context Rot [15] と呼ばれる現象が報告されている．本アーキテクチャでは，各 Worker が必要最小限のツールを持つシンプルな構成を維持するため，特にリソース制約のあるローカル LLM を用いる場合でも安定した性能を達成しやすい．

最後に，Plan-and-Execute ワークフローは，ツールの実行と評価を繰り返す単純なワークフローが陥りやすい局所的失敗の連鎖や，マルチエージェントアーキテクチャが直面しやすい無限ループを防止する．解析の開始点から最終的なレポートの生成までの長期的な計画を最初に作成し，状況に応じて適切に更新することで，システムレベルの一貫性が向上し，安定した結果を得ることが可能となる．

3.4 実装の詳細

ロールごとのモデル選択による効率性の向上． LLM エージェントは対話型 LLM に比べて非常に多くのトークンを消費し，Anthropic のレポート [17] によるとマルチエージェントの消費トークン数は 15 倍にもなる．一方で，同レポートではエージェントの性能と消費トークン数との強い相関も指摘されている．そこで，本研究では経済的な現実性と性能を両立する観点から Ollama によって駆動されるローカル LLM を活用する．また，エージェントごとに異なる LLM を使用してさらなる効率化を図る．Supervisor は解析の履歴全体を把握する必要があり高度なプランニング能力が求められるため比較的大規模な推論モデルである Qwen3:32B を用い，Worker にその小規模版でより高速な Qwen3:8B を用いる．

解析に不可欠な最小限のツールの実装． 前述の通り，LLM に過剰な数のツールを与えるとツール使用の精度が悪化しやすいため，有用でありうるツールをすべて実装するのではなく，解析に不可欠なツールのみを実装すべきである．なお，実装するツールを検討する際には PyPI 上の悪性パッケージ内で用いられている悪性動作展開手法を大規模に調査した既存研究 [18] を参照した．

まず，Deobfuscator に対しては悪性コード内に典型的に見られる Fernet と base64 難読化を解除するためのツールを実装した．これらのツールは実行失敗時に詳細なエラーメッセージを出力し，LLM の次の判断を補助する．また，JJEncode [19] のような算術演算と記号操作を用いる高度な難読化技術に対応するためコード片を安全に実行して動的解析を行えるサンドボックスツールを用意した．

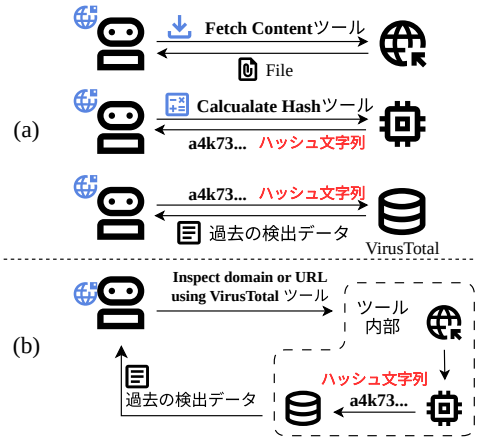

図 2: (a) LLM が複数のツールを正しい順番で使い，その中で正しくハッシュ値を扱うことに依存する信頼性の低い実装 (b) ツールの単一化によってそれらを解消した実装

さらに，Web Researcher に対してはコード内に現れるオブジェクトに関する背景情報を取得するためのツールとして Web 検索ツールと URL からのファイルダウンロードツールを実装した．また，既存研究 [18] では悪性パッケージ内の URL の多くが短時間で無効になることが判明しているため，直接アクセス以外の情報取得手段として VirusTotal 上で URL やドメインを検索するツールを実装した．なお，URL がファイルを指している場合はそのハッシュ値を計算して VirusTotal を検索するが，その際には図 2 (a) のように LLM に高エントロピー文字列であるハッシュ値を直接入出力させる実装を避け，図 2 (b) のように 1 つの大きなツール内で全てのステップを完結させることでシステムレベルの信頼性を向上させた．

4. 評価

4.1 実験設定

性能評価用データセットの作成． 性能評価用データセットは悪性パッケージ 150 件，良性パッケージ 150 件からなる 300 件のパッケージにより構成した．なお，悪性パッケージは PyPI 上の悪性パッケージに関する調査研究 [20] で作成された pypi_malregistry から選んだが，このデータセットは GitHub にて `https://github.com/lxyeternal/pypi_malregistry` で公開されており，未だに更新され続けている．ここでは Commit Hash が 153aba56 から始まる commit からランダムに 150 件を選んだ．良性パッケージは 2025/07/22 に PyPI 上に新規に公開されたものから 150 件を選んだ．なお，データセット内のすべてのパッケージが `setup.py` と `__init__.py` を含むことを確認した．

解析範囲の設定や動作環境の構成． 本実験では，パッケージ内の解析範囲を `setup.py` と `__init__.py` に限定した．これは，悪性パッケージの 87.6%がインストール時発火型

(setup.py が悪性ペイロードを含む) かインポート時発火型 (__init__.py が悪性ペイロードを含む) である [20] ためと，現行の実装では計算量上すべてのファイルを解析することが難しいためである．なお，実験には NVIDIA H100 NVL を 1 つ用いた．また，エージェントが特定のパッケージの解析に非常に長い時間をかけてしまい実験全体が停止することがないよう，20 分間のタイムアウトを設定した．

4.2 エージェントの解析プロセスの事例

まず，エージェントによる実際の解析事例を紹介する．プログラム 2 はインストール時発火型の悪性パッケージである libstrreplacecpu-7.3 のコードで，この内部では符号化されたコマンドを PowerShell の -EncodedCommand によって実行して悪性ファイルをダウンロード・起動する．

```
if not os.path.exists('tahg'):
  # www.esquelesquad.rip
  subprocess.Popen(
    'powershell -WindowStyle Hidden -
    EncodedCommand cABv...IAIgA=',
    shell=False, creationflags=subprocess.
    CREATE_NO_WINDOW
  )
setup(
  name = 'libstrreplacecpu',
  description = 'A library for creating a
    terminal user interface',
  author = 'EsqueleSquad',
  author_email = 'tahgoficial@proton.me',
)
```

プログラム 2: libstrreplacecpu-7.3 のコードの一部．エージェントの解析で参照されなかった部分は省略した．

そして，図 3 は本研究のエージェントがこのパッケージを解析したときのトレースを示している．
(1) Supervisor は最初の計画を作成する (New Plan)．
(2) Supervisor は Deobfuscator に難読化されたコマンドを復号するよう要求する．
(3) Deobfuscator は復号結果を Supervisor に報告する．
(4) Supervisor は (3) に基づいて計画を更新する．最初の計画内の “Decoded URL” は “Dropbox URL” に更新され，コマンド自体の解析は完了したと判断されたため PowerShell コマンドの追加解析は計画から削除される (Revised Plan)．
(5) Supervisor は Web Researcher に新たに出現した URL を調べるよう依頼し，これが過去に VirusTotal 上で 7 つのベンダから悪性と判断されていることを知る．
(6) Supervisor は Web Researcher にパッケージ作成者のメールアドレスとコード中の不審な URL を調べるように要求し，調査結果を得る．
(7) Supervisor は十分な情報が揃ったと判断し，解析レポートを生成する．

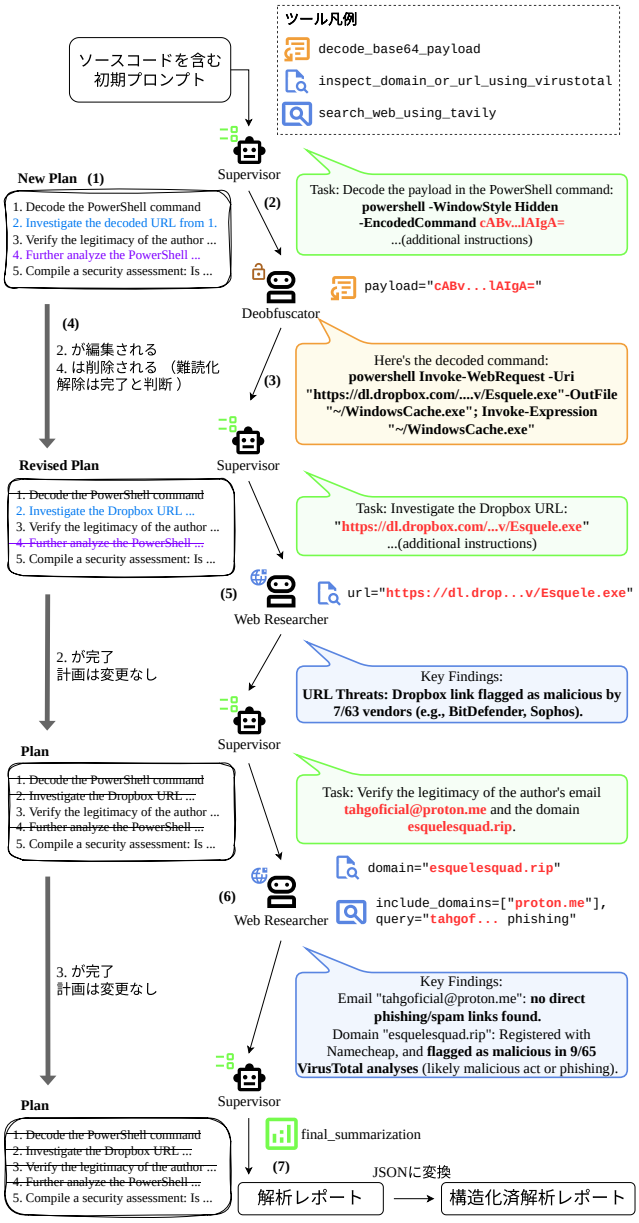

図 3: libstrreplacecpu-7.3 を本研究のエージェントが解析した際のトレース．

実際に生成された解析レポートの例は付録 A.1 に記載した．

4.3 検出性能の検証と解析レポートの品質検証

エージェントの性能と説明性をより詳細に評価するため，悪性パッケージの検出精度の定量的評価とプロの解析者に対するアンケートによる解析レポートの品質評価を行った．**悪性パッケージの検出性能の定量的評価．**作成した性能評価用データセットを用いて本研究のエージェントの悪性パッケージ検出性能を評価したところ，再現率が 1.000，適合率が 0.993 となり，1 件の偽陽性と 10 件のタイムアウトがあった．また，1 パッケージあたりの解析時間の中央値は 377 秒であった．以上の高い検出性能から，本研究のエージェントの設計の有効性が実証された．また，約 6 分という解析時間は本システムが実現する詳細な分析能力と

表 1: 評価者ごとの各観点内の平均スコア（良性）

評価者/観点	正確性	網羅性	明瞭性	実用性	信頼性
P1	3.7	3.0	3.7	4.3	3.3
P2	3.3	2.3	3.7	2.0	2.0
P3	3.7	3.3	5.0	3.0	3.7
平均	3.6	2.9	4.1	3.1	3.0

表 2: 評価者ごとの各観点内の平均スコア（悪性 1）

評価者/観点	正確性	網羅性	明瞭性	実用性	信頼性
P1	3.3	4.0	4.3	4.3	3.7
P2	3.0	3.0	2.3	3.0	3.3
P3	5.0	5.0	5.0	4.7	5.0
平均	3.8	4.0	3.9	4.0	4.0

の現実的なトレードオフであり，今後の並列化による高速化を考慮すれば実運用上十分に許容可能な数値である．

専門家評価によるレポート品質の検証. 本エージェントが生成する解析レポートの実用的な品質を評価するため，サイバーセキュリティの実務経験を持つ 3 名の専門家（P1: 社内セキュリティ戦略策定者，P2: レッドチーム，P3: 脅威分析エンジニア）に評価者として協力を依頼してアンケートを実施した．評価者の要件はマルウェア解析と Python プログラミングの経験があることとした．評価者は，良性パッケージ 1 件と悪性パッケージ 2 件（計 3 件）について，実際のソースコードと本エージェントが生成した解析レポートを対比し，次の 5 つの観点ごとに 3 つの質問を含む計 15 問に回答して品質を評価した．

- 正確性: 提示される情報が事実として正しく，コードと矛盾がないか．
- 網羅性: パッケージの挙動に関する重要な側面を全てカバーしているか．
- 明瞭性: 専門家にとって理解しやすく，論理的に一貫しているか．
- 実用性: 具体的で実行可能な提言を含んでいるか．
- 信頼性: 専門家がそれに基づき，どの程度自信を持って意思決定を行えるか．

回答選択肢には 5 段階のリッカート尺度を用い，例えば「レポートに記述されている技術的な分析内容は，提供されたソースコードに照らして正確である」という質問に対して 1（全く同意しない）~5（強く同意する）の間で採点した．また，パッケージごとにコメントの自由記述欄を設けた．評価のバイアスを低減するため，3 つのパッケージを提示する順番はランダム化し，評価項目に上記のカテゴリ見出しは記載しない設計とした．なお，本アンケートにおいては悪性パッケージの実サンプルを用いるため評価者に対する倫理的配慮が必要となる．詳細は第 5 章で議論する．

表 1~3 に評価結果を示す．これと評価者のコメントを合わせると，本エージェントは特に脅威の全体像の網羅性や

表 3: 評価者ごとの各観点内の平均スコア（悪性 2）

評価者/観点	正確性	網羅性	明瞭性	実用性	信頼性
P1	4.0	4.0	4.3	3.3	4.0
P2	2.7	4.0	3.7	3.0	3.3
P3	4.3	4.7	5.0	5.0	4.7
平均	3.7	4.2	4.3	3.8	4.0

レポート内の用語の明瞭性の観点から専門家に高く評価された一方で，良性パッケージに対してはリスクを過大評価する傾向があり，実運用に即さない過剰な警告を生成するバイアスがあるという課題も判明した．注目すべき観察として，評価は専門家の役割に依存し，P3（脅威分析エンジニア）からは高く評価された一方 P2（レッドチーム）からは技術的な厳密性や用語の正確性に対する指摘がなされた．特にレポート内の推奨事項については「誰に向けたものかわかりにくい」との意見も見られた．これは，本エージェントが脅威の発見と要約には優れるものの，良性コードの検証や特定の専門家が要求する詳細情報の提供では改善が必要であることを示しており，LLM 特有の考えられるリスクを全て列挙するような挙動や過度に一般化された論調が有用性を低下させている可能性を示唆している．

4.4 失敗した事例の分析

失敗事例は，エージェントが「悪性コード検出」という主タスクと「脆弱性検査」を混同したことに起因した．具体的には，1 件の偽陽性では現在危険で非推奨とされる `exec` によるバージョン変数の設定を脆弱性ではなく悪性と誤認していた．また，良性パッケージで発生したタイムアウトでは，多数の依存パッケージに対して網羅的な脆弱性検査を試みた結果，処理時間が超過していた．

4.5 マルチエージェントアーキテクチャの有効性の検証

最後に，本研究におけるマルチエージェント構成の有効性を定性評価するため，全プロンプトとツールを統合した単一のエージェントをベースラインとして定義した．表 4 の通り，ベースラインは単一で巨大なプロンプトと大量のツールが原因で正確な段階的実行ができず，計画外の行動を起こして頻繁に失敗した．対照的に，本研究のアーキテクチャは各 Worker に専門的なプロンプトと役割固有の最小限のツールセットを割り当てる．これにより，Worker は責任範囲外のタスクを Supervisor に委ねざるを得ないため，安定した段階的実行が強制され，失敗が抑制される．

5. 議論

制約事項. 本研究のエージェントには 3 つの主要な制約が存在する．第一に，分析範囲を悪性コードの主要なエントリポイントである `setup.py` と `__init__.py` に限定しており，他のファイル内の脅威を見逃しうる．第二に，評価用

表 4: ベースラインエージェントの失敗パターン．**アイコンの凡例:** 難読化, リモートペイロードの取得, 動的コード実行, 同じ行動の繰り返し, コード内の要素についてのハルシネーション, 使用ツールの誤り, 計画作成失敗, タイムアウト, 出力欠損, 解析成功.

パッケージ名	悪性行動	失敗パターン	最終結果
libstrreplacecpu version 7.3			
proggressbar2 version 0.1			
mariabd version 0.1			
ethereim version 1.0.0			

パッケージ数が 300 と小規模であり，定量的な有効性検証と失敗事例の調査が限定的である．第三に，現在の静的な Worker の構成は一般的な悪性コードパターンには有効なものの，この Worker の組み合わせで将来も含めた全ての巧妙な回避技術が解析できるわけではない可能性が高い．

今後の展望. 本研究のアーキテクチャは多方面への拡張が可能である．まず，大規模パッケージへのスケーラビリティ向上のため，コールグラフやテイント解析によるプログラムスライシングを導入し，パッケージ全体を扱いつつも解析に必要なコード片のみを選択的に抽出できる機構を実装する．また，過剰な警告を減少させた上で専門家の役割に応じてレポートの内容や焦点を動的に調整して情報の的確性を高め，より実運用に即した分析の実現を目指す．さらに，ツールの自動生成手法を応用して，静的なツールセットでは対応困難な未知の脅威への適応性を高める．最後に，LLM の推論並列化によりスループットを向上させ，最終的には PyPI でのリアルタイムスクリーニングを通じて実環境での有効性を検証する．

倫理的配慮と評価者等の安全確保. 悪性パッケージを扱う本研究の評価プロセスでは，評価者および実験環境の安全確保が最優先課題となる．本研究ではこの課題に対し，多層的な安全措置を講じた．まず，エージェントの検出性能評価では，実験用サーバを他のローカルネットワークから完全に隔離した．さらに，実験中に発生した全てのプロセスとネットワーク通信を詳細に記録し，実験後に検査することで，意図しない外部への影響がないことを確認した．次に，より高いリスクを伴う専門家による評価では，評価者のローカル環境を一切危険に晒すことなく実践的な解析環境を提供することが不可欠であった．このため，我々はオープンソースの Web IDE である Coder をホストし，評価者にはブラウザでアクセスできる完全に隔離されたコンテナベースのワークスペース内で評価作業を行うよう依頼した．この環境では，ワークスペースからのファイルダウンロードを技術的に完全に禁止しており，評価者が誤って悪性コードを自身の PC に持ち込むリスクを原理的に排除した．これにより，評価者が使い慣れた IDE やターミナルに近い環境で，かつ安全性を完全に担保された状態でレポートの品質評価に集中することを可能にした．

6. まとめ

本稿では，PyPI 上のパッケージを自律的に解析して悪性パッケージを検出するための拡張性の高い Supervisor-Worker 型マルチエージェントを設計し実装した．また，実際のパッケージサンプルを用いた実験でその高い検出性能を示し，解析中のトレースの目視確認とセキュリティ分野の専門家による解析レポートのアンケート評価を通して改善点を特定した．今後はより大規模なパッケージと多様な脅威への対応ならびに専門家の役割に応じた解析レポートの焦点の調整を可能にする．これらを通じ，有用で信頼性の高い自動解析技術とすることを目指す．

謝辞 この成果の一部は，NEDO (国立研究開発法人新エネルギー・産業技術総合開発機構) の委託業務 (JPNP24003) の結果得られたものです．

参考文献

[1] Checkmarx Security: PyPI halts new projects, users for 10 hours due to infostealer influx (2024).
[2] Ravie Lakshmanan: Malicious PyPI Package 'Fabrice' Found Stealing AWS Keys from Thousands of Developers (2024).
[3] Socket Threat Research Team: Monkey-Patched PyPI Packages Use Transitive Dependencies to Steal Solana Private Keys (2025).
[4] Mike Fiedler: Project Quarantine (2024).
[5] Vu, D.-L., Newman, Z. and Meyers, J. S.: Bad Snakes: Understanding and Improving Python Package Index Malware Scanning, *ICSE'23* (2023).
[6] Sun, X., Gao, X., Cao, S., Bo, L., Wu, X. and Huang, K.: $1 + 1 > 2$: Integrating Deep Code Behaviors with Metadata Features for Malicious PyPI Package Detection, *ASE'24* (2024).
[7] : Killing Two Birds with One Stone: Malicious Package Detection in NPM and PyPI using a Single Model of Malicious Behavior Sequence, *ACM TOSEM* (2024).
[8] Zheng, X., Wei, C., Wang, S., Zhao, Y., Gao, P., Zhang, Y., Wang, K. and Wang, H.: Towards Robust Detection of Open Source Software Supply Chain Poisoning Attacks in Industry Environments, *ASE'24* (2024).
[9] Huang, C., Wang, N., Wang, Z., Sun, S., Li, L., Chen, J., Zhao, Q., Han, J., Yang, Z. and Shi, L.: DONAPI: malicious NPM packages detector using behavior sequence knowledge mapping, USENIX Security'24 (2024).
[10] Gao, X., Sun, X., Cao, S., Huang, K., Wu, D., Liu, X., Lin, X. and Xiang, Y.: MalGuard: Towards Real-Time, Accurate, and Actionable Detection of Malicious Packages in PyPI Ecosystem, *arXiv:2506.14466* (2025).
[11] Wang, J., Li, Z., Qu, J., Zou, D., Xu, S., Xu, Z., Wang, Z. and Jin, H.: MalPacDetector: An LLM-Based Malicious NPM Package Detector, *IEEE TIFS* (2025).
[12] Lebiere, C., Bennati, S., Thomson, R., Shakarian, P. and Nunes, E.: Functional cognitive models of malware identification, *ICCM'15* (2015).

[13] Thomson, R., Lebiere, C., Bennati, S., Shakarian, P. and Nunes, E.: Malware identification using cognitively-inspired inference, *BRiMS'15* (2015).
[14] Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K.-W. and Lim, E.-P.: Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models, *ACL'23* (2023).
[15] Hong, K., Troynikov, A. and Huber, J.: Context Rot: How Increasing Input Tokens Impacts LLM Performance, Technical report, Chroma (2025).
[16] Paramanayakam, V., Karatzas, A., Anagnostopoulos, I. and Stamoulis, D.: Less is More: Optimizing Function Calling for LLM Execution on Edge Devices, *DATE'25* (2025).
[17] Anthropic Engineering Team: How We Built Our Multi-Agent Research System (2025).
[18] 戸田宇亮，若井琢朗，森　達哉：LLM エージェントを活用した PyPI 上の悪性パッケージ大規模調査，技術報告 ICSS2024-80 (2025). 信学技報.
[19] Ferrie, P.: Malware Analysis (2011).
[20] Guo, W., Xu, Z., Liu, C., Huang, C., Fang, Y. and Liu, Y.: An Empirical Study of Malicious Code In PyPI Ecosystem, *ASE'23* (2023).

付　　録

A.1　生成された解析レポートの例（抜粋）

```
## Final Verdict
**Malicious logic detected**

The code contains explicit malicious logic that proactively causes harm by downloading and executing an external executable (‘Esquele.exe‘) without user consent. This behavior directly contradicts the claimed purpose as a "terminal UI library," confirming it as a **malicious payload distribution mechanism**.
---

## Stepwise Execution of Malicious Activity
1. **Initial Trigger**: {Trigger of malicious behavior}

2. **Decoded PowerShell Command**:
   ‘‘‘powershell
   powershell (OMIT); Invoke-Expression "~/WindowsCache.exe"
   ‘‘‘
   - **Step 1**: Downloads ‘Esquele.exe‘ to ...
   - **Step 2**: Executes the downloaded file using ...

3. **Persistence & Execution**:
   - The executable (‘WindowsCache.exe‘) is stored in a user directory (‘~‘), ensuring persistence.
---

## Ultimate Goal of the Attacker
The attacker aims to **distribute ... While the file itself (‘Esquele.exe‘) showed no direct malicious detections, the Dropbox URL and domain (‘esquelesquad.rip‘) are strongly associated with malware. This suggests:
{Targeted Distribution, Package Name Social Engineering}
---

## Additional Threat Intelligence
 1. **Suspicious Infrastructure**:
   - **Domain ‘esquelesquad.rip‘**: Flagged as malicious in 9/65 VirusTotal analyses, likely used for phishing or C2 (command-and-control) operations.
   - **Dropbox URL**: Despite ‘Esquele.exe‘ being undetected, the URL is flagged as malicious by 7/63 vendors (BitDefender, Sophos).

 2. **Indicators of Compromise (IOCs):**
   - **URL:** ‘https://dl.dropbox.com/xxxxxxxxxxxxx/Esquele.exe?dl=0‘ (403 Forbidden, likely malicious).
   - **File:** ‘WindowsCache.exe‘ (disguised name for ‘Esquele.exe‘).
   - **Email:** ‘tahgoficial@proton.me‘ (ProtonMail is often used for anonymity in malicious campaigns).
{Attribution Considerations}
---

## Risk Rating & Recommendations
{"High Risk" Rating, Evidences}

### Mitigation Steps
{Avoid Installation, Monitor ‘Esquele.exe‘}
```

プログラム 3: 図 3 に示した解析の結果として自動生成された libstrreplacecpu-7.3 の解析レポート．オレンジと青のテキストはそれぞれ Deobfuscator と Web Researcher のタスクの結果を反映している．中括弧 {} で囲んだ箇所は紙面の都合上省略した部分についてそこに書かれていたことの概要を記した．