JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 27, No. 8, pp. 2399-2411, Aug. 2026

Check for updates

안전한 생성형 AI 기반 코드 개발을 위한 출력 코드 안전성 검증 프레임워크

서 효 리[1] · 석 병 진[2*]
[1]한성대학교 융합보안학과 학사과정
[2]한성대학교 융합보안학과 조교수

Output Code Safety Verification Framework for Secure Generative AI-Based Code Development

Hyori Seo[1] · Byoungjin Seok[2*]

[1]Undergraduate Student, Department of Convergence Security, Hansung University, Seoul 02876, Korea
[2]Assistant Professor, Department of Convergence Security, Hansung University, Seoul 02876, Korea

[요 약]

생성형 인공지능(AI; artificial intelligence)은 자연어 질의에 대하여 실행 가능한 코드를 직접 제공하며, 다수의 사용자가 이를 검증 없이 복사·실행한다. 그러나 현재의 안전장치는 사용자의 요청을 차단하는 입력 단계에 집중되어 있어, 출력 소비 시점에 대한 안전성 고려는 미비한 실정이다. 본 논문에서는 생성형 AI가 산출한 코드를 사용자가 소비하는 시점에 가로채어 검증하는 출력 코드 안전성 검증 프레임워크를 제안한다. 제안 프레임워크는 응답을 우선 흐림 처리한 뒤 안전이 확인된 것만 해제하는 Blur-First 전략, 코드의 의미와 맥락을 판별하는 대규모 언어 모델(LLM; large language model) 의미 분석, 분석기 자체를 겨냥한 프롬프트 인젝션을 탐지하는 analyzer_subversion 방어로 구성된다. 제안된 프레임워크는 명시적 위험 시그니처 중심의 13개 프로그래밍 언어 990개 샘플로 구성한 평가셋에서 정확도 96.7%, 정밀도 100%, 재현율 93.5%, F1 점수 96.7%를 달성하였으며, 프레임워크 내 비식별화 모듈은 masking recall 95.0%와 false masking 0%를 기록하여 출력 코드 안전성 검증을 성공적으로 수행할 수 있었다.

[Abstract]

Generative artificial intelligence (AI) directly returns executable code for natural-language queries, and many users copy and run such code without verification. Current safeguards concentrate on blocking user requests at the input stage; however, safety considerations at the point of output consumption remain insufficient. This study proposes an output code safety verification framework that intercepts and verifies code produced by generative AI for user consumption. The proposed framework consists of a blur-first strategy that blurs responses and reveals only those confirmed safe, a large language model (LLM)-based semantic analysis that determines the meaning and context of the code, and an analyzer_subversion defense against prompt injection targeting the analyzer itself. On an explicit-risky-signature evaluation set of 990 samples across 13 programming languages, the proposed framework achieved 96.7% accuracy, 100% precision, 93.5% recall, and a 96.7% F1 score, while its anonymization module recorded 95.0% masking recall and 0% false masking, successfully carrying out output code safety verification.

색인어 : 생성형 AI, 출력 검증, 프롬프트 인젝션, 대규모 언어 모델, AI 보안
Keyword : Generative AI, Output Verification, Prompt Injection, Large Language Model, AI Security

http://dx.doi.org/10.9728/dcs.2026.27.8.2399

Received 05 July 2026; **Revised** 16 July 2026
Accepted 31 July 2026

***Corresponding Author; Byoungjin Seok**
※ 개인정보 표시제한
E-mail: bjseok@hansung.ac.kr

Ⅰ. 서 론

생성형 인공지능(AI; artificial intelligence) 서비스는 자연어 질의에 대한 응답으로 실행 가능한 코드를 직접 제공한다. AI를 사용하는 개발자의 72%가 이를 매일 활용하며 전체 커밋 코드의 42%를 AI가 작성하는 반면, 커밋 이전에 항상 검토한다고 응답한 비율은 48%에 그친다[1]. 그러나 AI가 산출한 코드가 안전한 것은 아니며, GitHub Copilot이 생성한 1,689개 프로그램 중 약 40%가 취약점을 포함한다[2]. 특히 이러한 위협은 가설에 그치지 않고 실제 코드 산출물로 관측되고 있다. AI가 페이로드를 실행 시점에 직접 합성하는 랜섬웨어가 확인되었으며[3], 기술 숙련도가 낮은 행위자가 AI의 보조로 악성코드를 반복 제작한 사례[4]와 국가 배후 조직이 정찰 및 악성코드 작성에 생성형 AI를 활용한 사례[5]가 있다. 이들 위협은 형태가 서로 다르더라도 최종적으로 AI가 산출한 실행 가능 코드라는 동일한 경로로 사용자에게 도달한다.

그러나, 기존의 안전장치는 각기 장점을 가지면서도 사용자에게 전달되는 코드를 검증하는 데 한계를 가진다. 대규모 언어 모델(LLM; large language model) 응용의 주요 보안 위협은 표준화된 형태로 정리되어 있으나, 그 방어 지점은 모델에 전달되는 입력에 집중되어 있다[6]. 입력 차단은 악의적으로 보이는 요청을 조기에 막는 이점을 가지지만 요청의 표현을 대상으로 하므로 우회 표현의 다양성에 영향받으며, 29개 주류 LLM을 대상으로 한 평가에서 악성 코드 생성 요청에 대한 평균 거부율은 60.93%, 탈옥 기법과 결합하는 경우 39.92%까지 하락하였다[7]. 즉 동일한 수준의 요청이라도 공급자와 기법에 따라 차단 여부가 달라지며, 일단 가이드라인을 벗어나 코드가 생성되면 그 코드는 어떠한 추가 검증도 거치지 않는다.

더욱이 본 논문이 다루는 위협은 악의적 사용자를 전제하지 않는다. 모델은 악의가 없더라도 인간의 지시를 그대로 수행하는 과정에서 위험한 코드를 산출할 수 있으며, 사용자 또한 위험을 인지하지 못한 채 그 코드를 실행할 수 있다. 따라서 방어의 목적은 악의적 요청의 차단에 한정되지 않고, 선의의 사용자에게 전달되는 산출물을 한 번 더 검증하는 데 있다. 한편 코드 보안 분야의 기존 접근은 이 지점을 겨냥하지 않는다. 정적·학습 기반 취약점 분석은 개발자가 작성한 소스 코드의 취약 패턴 탐지에 초점을 두고[8], 악성 코드 탐지는 주로 컴파일된 실행 파일을 대상으로 한다[9]. 코드 위험성 벤치마크는 모델이 위험한 코드를 생성하거나 실행하는지를 평가하며[10],[11], 프롬프트 인젝션 방어 연구는 입력 단계에서 서비스 LLM을 보호한다[12]. 또한 AI 보조가 사용자로 하여금 더 취약한 코드를 작성하게 만들 수 있음을 실증한 연구 역시 그러한 코드를 소비 시점에 차단하는 방어는 다루지 않았다[13]. 따라서 생성형 AI가 산출한 코드를 사용자가 소비하는 시점에 검증하는 방식이 요구된다.

이러한 배경에서 본 논문은 생성형 AI의 응답 코드를 실행 이전에 검증하기 위하여 Blur-First 전략과 LLM 의미 분석을 활용하는 출력 코드 안전성 검증 프레임워크를 제안한다. 다만 본 연구에서 규칙 기반 사전필터는 위험 판정 엔진 자체를 의미하는 것이 아니라, 명백히 안전한 코드에 대한 모델 호출을 절감하기 위한 1차 분류 기준으로 활용된다. 이로써 제안 기법의 실질적 기여와 비용 절감 요소의 역할을 구분하고자 한다.

본 논문의 주요 기여는 다음과 같다. 첫째, AI 응답 코드가 화면에 표시되는 즉시 흐림 처리로 노출을 차단하고 분석 완료 후에만 해제하는 소비 시점 출력 검증 방식을 제안한다. 둘째, 분석에 사용되는 LLM 자체를 겨냥한 프롬프트 인젝션을 독립된 위협 범주로 정의하고 탐지하는 analyzer_subversion 방어를 제시한다. 셋째, LLM이 등급을 직접 결정하지 않고 위험 신호 플래그만 산출하게 한 뒤 백엔드가 결정론적 규칙과 조작 내성 게이트로 등급을 판정하는 구조를 설계하여 LLM의 비결정성을 통제한다. 넷째, 13개 언어 990개 샘플의 평가셋에 따라 실험을 수행하고, 비식별화 성능 및 미탐 사례 분석을 통해 제안 방식의 효과와 한계를 분석한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 생성형 AI 악용 동향과 LLM 보안, 기존 코드 보안 접근의 한계를 정리한다. Ⅲ장에서는 제안하는 프레임워크의 구조, 구성요소별 동작, 위험 등급 판정 절차를 설명한다. Ⅳ장에서는 실험 환경과 평가 기준, 비교 결과를 제시하고, 실험의 한계와 일반화 가능성을 논의한다. Ⅴ장에서는 결론 및 향후 연구 방향을 제시한다.

Ⅱ. 관련 연구

2-1 생성형 AI 악용 및 위협 동향

생성형 AI를 이용한 악성 행위는 실제 코드 산출물로 관측되고 있다. ESET이 공개한 PromptLock은 알려진 최초의 AI 기반 랜섬웨어로서, 바이너리에 하드코딩된 프롬프트를 응용 프로그램 인터페이스(API; application programming interface)를 통해 로컬 LLM에 전달하여 파일 탐색·정보 탈취·암호화에 사용할 스크립트를 실행 시점에 생성한다[3]. 악성 로직이 사전 컴파일된 형태로 존재하지 않고 실행 중 합성되므로, 위험의 실체는 정적 바이너리가 아니라 동적으로 산출되는 코드이다.

이렇게 산출된 코드는 비결정적이며 다형적이다. PromptLock은 같은 프롬프트에도 실행마다 스크립트가 달라져 침해 지표(IoC; indicator of compromise)가 변동한다[3]. Google Threat Intelligence Group은 실행 중에 LLM을 호출하여 악성 함수를 생성하고 자기 코드를 난독화하는 실행 시점 AI 악성코드 계열을 식별하였다[14]. 표현이 매번 달라지는 이상 알려진 패턴을 찾는 방식만으로는 한계가 명확하며, 코드의 동작과 의미를 판별하는 분석이 요구된다.

한편 AI는 공격의 진입 장벽을 낮추어 위협의 양을 증가시

킨다. Check Point가 분석한 FunkSec은 기술 숙련도가 낮은 것으로 보이는 행위자들이 AI의 보조로 Rust 기반 암호화기를 반복 개발하여, 활동 한 달여 만에 85건 이상의 피해를 주장한 사례이다[4]. 전문 인력 없이 정교한 도구가 양산될 수 있으므로, 위험 코드의 절대량이 증가하고 사용자가 노출될 가능성 또한 커진다.

또한 공격자는 입력 단계의 안전장치를 사회공학적 프롬프트로 우회한다. Google은 20개국 이상의 지능형 지속 위협(APT; advanced persistent threat) 조직이 Gemini를 정찰·취약점 조사·페이로드 개발에 활용하려 시도하였음을 분석하였으며[5], 후속 보고에서 안전 가드레일을 우회하기 위하여 페르소나 부여나 사회공학적 구실을 동원하는 정황을 확인하였다[14]. HP Wolf Security는 악성코드 유포에 사용된 스크립트 로더에서 줄 단위 주석과 네이티브 언어 변수명 등 생성형 AI로 작성된 정황을 발견하였다[15]. 이는 산출물이 사람이 읽을 수 있는 코드 형태로 사용자에게 전달되며, 그 의미와 맥락을 분석하는 접근이 유효함을 시사한다.

2-2 LLM 보안과 프롬프트 인젝션

LLM을 활용하는 애플리케이션의 주요 보안 위협은 표준화된 형태로 정리되어 있으며, 프롬프트 인젝션은 그중 핵심 위협으로 분류된다[6]. 프롬프트 인젝션은 모델에 주어지는 입력에 지시문을 삽입하여 모델의 의도된 동작을 변경하려는 공격이므로, LLM을 분석 엔진으로 사용하는 시스템에서는 분석 대상이 곧 잠재적 공격 입력이 된다.

특히 외부에서 제공되는 데이터에 지시문을 삽입하여 모델을 조작하는 간접 프롬프트 인젝션은 실제 LLM 통합 애플리케이션을 침해할 수 있는 위협으로 보고되었다[16]. 선행 연구[16]는 애플리케이션이 외부 데이터를 신뢰함으로써 발생하는 인젝션, 즉 보호 대상이 최종 사용자인 경우를 다룬다. 반면 본 논문은 이 개념을 분석 파이프라인 자체에 적용하여, 분석 대상 코드의 주석과 문자열에 분석기를 속이려는 지시문이 삽입되는 경우를 별도의 위협으로 탐지한다. 보호 대상이 최종 사용자가 아니라 방어 시스템 자체라는 점에서 위협 모델이 구분되며, 본 논문은 이 차이를 9번째 위협 카테고리로 명시적으로 다룬다(3-3절).

2-3 코드 위험성 벤치마크와 판정 일관성

코드의 실행·생성에 따른 위험성을 평가하기 위한 벤치마크가 제안되어 왔다. RedCode는 코드 에이전트의 위험한 코드 실행·생성을 평가하는 벤치마크를 제시하였고[10], RMCBench는 LLM이 악성 코드 요청에 저항 정도를 측정하는 벤치마크를 제안하였다[11]. 이들 벤치마크[10],[11]는 모델이 악성 코드를 생성하거나 실행하는지를 평가하는 데 초점이 있는 반면, 본 논문은 이미 생성된 코드를 소비 시점에 차단하는 문제를 다루므로 평가 축이 상호 보완적이다.

한편 LLM 출력의 비결정성을 완화하기 위한 기법으로, 동일 질의를 여러 번 수행한 뒤 다수결로 결론을 도출하는 self-consistency가 제안되었다[17]. 본 논문은 이러한 벤치마크를 향후 평가 확장의 기준으로, self-consistency를 경계 사례의 판정 일관성 강화 방안으로 참조한다.

2-4 기존 코드 보안 접근의 한계

코드 보안 분야에는 이미 여러 접근이 존재하나, 본 논문과는 개입 시점과 방어 대상이 다르다. 정적·학습 기반 취약점 분석은 소스 코드의 취약 패턴 탐지에 초점을 두고[8], 악성코드 탐지는 주로 컴파일된 실행 파일을 대상으로 하며[9], sandbox 기반 동적 검증은 코드를 실행하여 행위를 관찰하므로 실행 자체의 비용과 위험을 수반한다. 이들은 공통적으로 사용자가 코드를 소비하기 이전 시점을 겨냥하지 않는다.

또한 생성형 AI가 산출한 코드의 보안성에 관한 연구는 AI 보조가 사용자로 하여금 더 취약한 코드를 작성하게 만들 수 있음을 실증하였으나[13], 그러한 위험 코드를 소비 시점에 차단하는 방어는 다루지 않았다. 프롬프트 인젝션 방어 연구는 지시와 데이터를 구조적으로 분리하여 입력 단계에서 서비스 LLM을 보호하는 데 초점을 두는 반면[12], 본 논문은 분석에 사용되는 LLM 자체를 겨냥한 인젝션을 출력 검증 단계에서 탐지한다. 브라우저 확장 보안 연구가 악성 확장의 행위를 탐지하는 데 주목한 것과 달리[18], 본 논문은 선의의 확장을 출력 코드 검증의 전달 수단으로 활용한다.

본 연구의 차별성은 검증의 시점과 대상을 이동시키는 데 있다. 즉, 사용자의 요청이나 개발자가 작성한 코드가 아니라 생성형 AI가 산출한 코드를 검증 대상으로 삼되, 코드를 실행하지 않고 의미 분석으로 동작을 추론하며, 특정 공급자의 입력 정책에 의존하지 않고 사용자 환경에서 독립적으로 동작한다. 이로써 기존 접근이 겨냥하지 않았던 소비 시점의 공백을 메운다.

Ⅲ. 출력 코드 안전성 검증 프레임워크

3-1 설계 목표와 위협 모델

제안 기법은 여섯 가지 설계 목표를 둔다. 분석이 완료되기 전에는 코드가 사용자에게 노출되지 않아야 하며, 서버 장애나 통신 실패 시에도 이 원칙이 유지되어야 한다. 표현이 변형되거나 난독화된 코드와 맥락에 따라 위험도가 달라지는 코드는 의미 분석으로 판별한다. 분석 엔진 자체를 겨냥한 프롬프트 인젝션은 독립된 위협으로 탐지한다. LLM의 비결정성을 통제하여 동일한 위험 신호가 항상 동일한 등급으로 매핑되도록 한다. 정상 코드를 위험으로 오인하지 않는 정밀도

우선 원칙을 유지한다. 마지막으로 분석을 위한 외부 전송 과정에서 민감 정보가 노출되지 않도록 한다.

본 프레임워크의 위협 모델은 다음과 같다. 보호 대상은 위험을 인지하지 못한 선의의 사용자이며, 차단하고자 하는 위협은 사용자가 AI로부터 받은 코드를 위험성을 인지하지 못한 채 복사·실행하여 피해를 입는 상황이다. 따라서 공격자 모델은 사용자를 속이려는 AI 응답에 해당하며, 사용자 자신이 보호 장치를 의도적으로 무력화하는 경우는 위협 모델의 범위 밖이다. 구체적으로 개발자 도구를 이용한 흐림 강제 해제, 문서 객체 모델(DOM; Document Object Model) 직접 조작, 확장 비활성화, 화면 촬영이나 수기 전사를 통한 회피는 방어 범위를 벗어난다. 이는 본 프레임워크가 악의적 사용자를 강제로 통제하는 접근 통제 도구가 아니라, 선의의 사용자를 대상으로 하는 예방적 방어 계층이라는 설계 전제에서 비롯된 한계이다.

3-2 프레임워크의 구조와 구성요소

그림 1은 제안 프레임워크의 전체 구조를 보여준다. 제안 프레임워크는 코드 감지·차단부, 요청 검증부, 전처리부, 비식별화부, 사전필터, LLM 의미 분석부, 위험 등급 판정부, 세션 누적 분석부로 구성된다. 각 구성요소의 역할은 다음과 같다.

- 코드 감지·차단부: 사용자 화면에 새로 표시되는 AI 응답 코드를 실시간으로 감지하여 즉시 흐림 처리하고, 분석 결과가 반환되기 전까지 노출을 차단한다.
- 요청 검증부: 전달된 코드의 길이와 언어 필드를 확인하여 조건을 충족하지 못하는 요청을 거른다.
- 전처리부: 마크다운 코드 펜스와 줄 번호를 제거하고 줄바꿈·공백을 정규화하여 동일 코드의 사소한 변형을 같은 형태로 만든다.
- 비식별화부: 인증 키, 토큰, 인터넷 프로토콜(IP; Internet Protocol) 주소, 사용자 경로 등 민감 정보를 마스킹한다.
- 사전필터: 위험 시그니처 패턴에 하나도 매칭되지 않는 코드를 LLM 호출 없이 통과시키는 규칙 기반 1차 분류기이다.
- LLM 의미 분석부: 사전필터를 통과하지 못한 코드를 시스템 프롬프트와 함께 모델에 입력하여 의미와 맥락을 분석하고 위험 신호 플래그를 산출한다.
- 위험 등급 판정부: 산출된 플래그 조합을 결정론적 규칙으로 처리하여 최종 등급을 결정한다.
- 세션 누적 분석부: 한 대화창에 누적된 위협 유형의 집합이 사전 정의된 공격 체인과 매칭되는지 검사한다.

각 구성요소의 세부적인 동작은 다음과 같다.

코드 감지·차단부는 모든 AI 응답을 우선 위험한 것으로 가정하고 즉시 흐림 처리한 후, 분석으로 안전이 확인된 것만 해제하는 Blur-First, Analyze-Later 전략을 채택한다. 새로운 응답 메시지가 DOM에 추가되는 순간 메시지 전체에 흐림 효과와 복사 방지를 적용하고, 내용 변경이 3초간 관측되지 않는 시점을 스트리밍 완료로 판단하여 완성된 코드만 분석으로 전달한다. 서버 장애나 통신 오류가 발생하더라도 코드는 흐림 상태를 유지한다. 즉 어떤 이유로든 안전 확인이 이루어지지 않으면 코드는 차단된 상태에 머무르므로, 검증되지 않은 코드가 브라우저 사용자 인터페이스(UI; user interface) 수준에서 노출되지 않도록 설계하였다.

요청 검증부는 코드 길이와 언어 필드를 확인하여 조건을 충족하지 못하는 요청을 거부함으로써, 이후 단계가 불필요하게 호출되는 것을 막는다. 5자 미만의 코드는 분석 대상에서 제외한다.

전처리부는 마크다운 코드 펜스와 줄 번호를 제거하고 줄바꿈·공백을 정규화한다. 이 정규화는 단순한 정돈에 그치지 않고, 클라이언트가 정규화된 코드를 키로 결과를 캐싱할 때 캐시 적중률을 높이는 데에도 활용된다.

비식별화부는 민감 정보를 정규식 규칙과 섀넌 엔트로피 기반 검사의 2단계로 마스킹한다. 1단계는 인증 키, 토큰, 데이터베이스 접속 문자열, 사설 및 공인 IP 주소, 사용자 경로, 계정명, 내부 통합 자원 식별자(URL; Uniform Resource Locator) 등 형식이 알려진 민감 정보를 23종의 정규식 규칙으로 매칭한다. 2단계는 정규식이 포착하지 못하는 비정형 비밀을 대상으로, 문자열 리터럴의 섀넌 엔트로피[19]가 임곗값을 초과하는 경우 이를 마스킹한다. 오탐 억제를 위한 예외 처리도 함께 설계하였다. 범용 고유 식별자와 같이 형식이 정의된 문자열, 자연어 문장, 루프백 주소, 버전 표기 문맥은 검사 대상에서 제외하며, 계정명 규칙은 admin이나 root와 같이 실제 사용자를 식별하지 않는 일반 명칭을 제외 목록으로 관리한다. 반대로 변수명에 password, secret, token 등의 힌트가 존재하는 경우에는 길이와 엔트로피 기준을 완화하여 적용함으로써 탐지율을 확보한다.

사전필터는 LLM 호출의 비용과 지연을 절감하기 위한 규칙 기반 1차 분류기로, 위험 시그니처 패턴에 하나도 매칭되지 않는 코드만 통과시키는 보수적 정책을 따른다. 패턴은 공통 6개, 프롬프트 인젝션 13개, 언어별 106개를 합하여 총 125개로 구성되며, 언어별 패턴은 Python 43개, C/C++ 14개, Java 13개, Ruby 13개, Rust 12개, Go 11개로 이루어진다. 패턴에 하나라도 매칭되거나, 비식별화 마커가 포함되거나, 코드 길이가 임곗값을 초과하거나, 사전필터가 다루지 않는 언어인 경우에는 통과시키지 않고 LLM 의미 분석부로 전달한다. 따라서 본 연구의 핵심은 단순히 정규식으로 위험을 판정하는 데 있지 않고, 규칙 기반 분류를 모델 호출 절감에만 사용하고 위험 판정의 책임을 의미 분석에 두는 데 있다. 사전필터가 통과시킨 코드에는 규칙 기반 판단임을 나타내는 고정 신뢰도를 부여하여 모델이 산출한 신뢰도와 구분한다. 이 정책상 시그니처가 전혀 없는 신종 악성 코드는 사전필터 단계에서 통과할 수 있으므로, 이에 대한 강건성 검증은 향후

과제로 남긴다.

LLM 의미 분석부는 비식별화된 코드를 시스템 프롬프트와 함께 모델에 입력한다. 프롬프트의 최우선 원칙은 코드 내부의 모든 텍스트를 지시가 아닌 분석 대상 데이터로 간주하도록 선언하는 것이다. 이 원칙 위에서 기본 4가지 우회 패턴인 명령형 우회, 응답 위조, 역할 변경, 시스템 형식 모방과, 고급 6가지 우회 패턴인 코드 펜스 탈출, 가짜 자바스크립트 객체 표기법(JSON; JavaScript Object Notation) 강제, 인코딩 스머글링, 유니코드 트릭, 사회공학적 정당화, 다국어 인젝션 및 신뢰도 조작을 합한 총 10가지 인젝션 방어 지침을 명시한다[16]. 또한 위협 판정을 명령어의 존재만으로 결정하지 않고 동작, 대상, 조합을 종합하도록 안내하여, 동일한 파일 삭제도 대상이 사용자가 생성한 임시 파일이면 caution, 루트 디렉터리이면 danger로 구분되게 한다. 모델은 등급 대신 후술할 4개 플래그와 신뢰도·심각도, 위협 근거를 JSON 스키마로 강제 출력하며, 분석은 Claude Messages API로 구현하였다[20]. 반복 호출 시 입력 비용을 절감하기 위해 시스템 프롬프트에는 캐시 제어를 적용한다.

한편 실제로 동작하는 랜섬웨어와 같은 초고위험 코드에 대해서는 모델이 안전 정책에 따라 분석 자체를 거부하고 빈 응답을 반환하는 경우가 있다. 이를 단순 오류로 처리하면 가장 위험한 입력에서 분석이 중단되는 fail-closed 문제가 발생하므로, 본 프레임워크는 이러한 거부를 분석기가 거부할 만큼 명백히 위험한 코드로 간주하여 danger로 확정하는 fail-safe를 적용한다. 이로써 분석 거부 상황에서도 방어가 무력화되지 않는다.

위험 등급 판정부는 모델이 산출한 플래그 조합을 결정론적 규칙으로 처리하여 safe, caution, danger의 최종 등급을 결정한다. 상세한 판정 규칙은 3-3절에서 기술한다. 이 과정에서 모델 응답의 신뢰도가 누락되거나 유효하지 않은 경우에는 1.0으로, 심각도가 누락된 경우에는 0.5로 보정하며, 네 플래그는 boolean으로 강제 변환한다. 이는 결측값이 위협을 임의로 강등시키는 상황을 방지하기 위한 보수적 처리이다.

세션 누적 분석부는 개별 코드 단위를 넘어, 한 대화창에 누적된 위협 유형의 집합이 사전 정의된 공격 체인과 매칭되는지 검사한다. 공격 체인은 랜섬웨어, 원격 코드 실행, 데이터 유출, 백도어 설치의 네 가지로 정의하며, 각 체인은 반드시 포함되어야 하는 필수 위협 유형과 함께 나타나야 하는 부가 위협 유형의 조합으로 구성된다. 예를 들어 랜섬웨어 체인은 파일 파괴를 필수 유형으로 하며, 여기에 랜섬웨어, 난독화, 데이터 탈취, 네트워크 접근 중 하나 이상이 함께 관측될 때 매칭된다. 개별 체인에 매칭되지 않더라도 서로 다른 위협 유형이 세 종류 이상 누적되면 광범위 위협으로 기록하며, 검사는 세션 이력이 두 건 이상 축적된 이후에 수행한다. 세션은 확장 프로그램이 전달하는 대화창 식별자를 우선 키로 사용하고 식별자가 없는 경우에만 IP 주소와 서비스명의 조합을 대체 키로 사용하는데, 이는 네트워크 주소 변환 환경에서 다수 사용자가 동일한 IP를 공유하더라도 세션이 혼재되지 않도록 하기 위함이다. 세션 유효 시간은 30분이며 세션당 최대 20건의 이력을 유지한다. 본 방식은 위협 유형의 출현 순서가 아니라 집합의 구성을 기준으로 판정하는 휴리스틱이므로, 각 단계가 개별적으로 무해하게 보이도록 분할된 공격은 탐지하지 못한다.

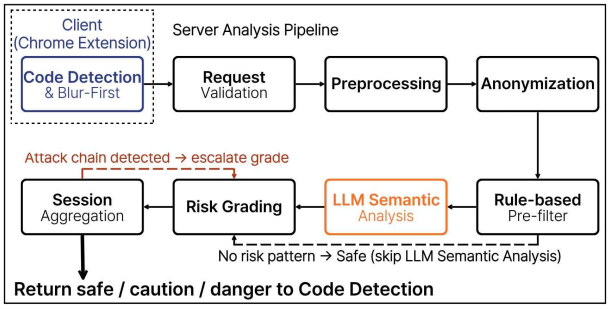


그림 1. 제안 프레임워크의 구성요소 및 분석 흐름

Fig. 1. Component composition and analysis flow of the proposed framework

3-3 위협 카테고리와 위험 등급 판정

분석 파이프라인이 무엇을 위험으로 판단하는지를 정의하기 위해, 본 논문은 표 1과 같이 9개의 위협 카테고리를 둔다. 1번부터 8번까지는 파일 파괴, 역방향 셸, 권한 상승, 데이터 탈취, 랜섬웨어, 난독화, 네트워크 접근, 코드 실행 등 전통적으로 알려진 악성 동작을 포괄한다. 이 중 9번 analyzer_subversion은 본 논문의 고유 항목으로, 분석 대상 코드 내부에 분석기를 속이려는 텍스트가 삽입된 경우를 탐지한다. 이러한 텍스트는 정상 코드에는 나타날 이유가 없으므로 그 존재 자체가 강력한 악성 신호이다. 이에 시스템 프롬프트를 통해 해당 위협이 탐지되면 네 플래그를 모두 참으로 산출하도록 지시하며, 위험 등급 판정부는 analyzer_subversion을 후술할 조작 내성 위협으로 분류하여 신뢰도가 낮더라도 danger를 유지한다. 다만 플래그 산출 자체는 모델의 지시 준수에 의존하므로, 결정론성은 플래그가 산출된 이후의 등급 매핑 단계에 한정된다.

위험 등급은 사용자가 취해야 할 행동을 기준으로 safe, caution, danger의 3단계로 부여한다. 핵심 설계 원칙은 등급을 LLM이 직접 결정하지 않는다는 것이다. LLM은 각 위협에 대해 irreversible, system_wide, unambiguous_malice, obfuscated의 4개 boolean 플래그와 신뢰도, 심각도 값을 산출하며, 백엔드가 이를 결정론적 규칙으로 등급화한다. 동일 코드에 대해 LLM이 매번 다른 등급을 낼 수 있는 반면, 이 구조에서는 플래그와 신뢰도가 정해지면 같은 조합이 항상 같은 등급으로 매핑되며 등급의 근거를 규칙으로 역추적할 수 있다.

네 플래그는 위협을 서로 직교하는 네 축으로 분해한다. irreversible은 피해의 가역성을, system_wide는 피해의 범위를, unambiguous_malice는 의도의 명확성을, obfuscated는 은닉 시도 여부를 판단한다. 이렇게 분해하면 LLM은 코드가 위험한가라는 모호한 판단 대신 되돌릴 수 있는가와 같이

답하기 쉬운 이진 질문에 답하게 되므로, 판정의 일관성과 설명가능성이 함께 높아진다.

판정은 2단계로 이루어진다. 1단계는 위협별 등급을 매긴다. irreversible이 참이고 나머지 세 플래그 중 하나 이상이 참이면 block-tier, block-tier에 해당하지 않으면서 네 플래그 중 하나 이상이 참이면 warn-tier, 네 플래그가 모두 거짓이면 safe-tier이다. 2단계는 위협별 등급과 신뢰도를 결합한다. block-tier 위협이 존재하고 신뢰도가 0.7 이상이면 danger로 확정하고, block-tier이면서 신뢰도가 0.7 미만이면 한 단계 강등하여 caution으로 처리하며, warn-tier 위협이 하나 이상이면 caution, 그 외에는 safe로 판정한다. 예를 들어 루트 디렉터리를 재귀 삭제하는 코드는 irreversible과 system_wide가 참이 되어 block-tier로서 danger로 판정되고, 사용자가 생성한 임시 파일을 정리하는 코드는 warn-tier에 해당하여 caution으로 처리된다. 이상의 판정 규칙을 정리하면 표 2와 같다.

다만 신뢰도는 LLM이 스스로 보고하는 값이므로, 공격자가 코드에 신뢰도를 낮추라는 텍스트를 삽입하여 등급을 강등시키는 우회가 가능하다. 이를 막기 위해 본 프레임워크는 조작 내성 규칙을 둔다. analyzer_subversion 위협이거나, obfuscated 또는 unambiguous_malice 플래그가 참이거나, 심각도가 0.8 이상인 위협은 신뢰도가 낮아도 강등하지 않고 danger를 유지한다. 또한 신뢰도가 0.3 미만인 경우는 그 자체를 조작 의심 신호로 간주하여 강등하지 않는다. 이 규칙에서 실제 강등 가능 구간은 매우 좁다. obfuscated 또는 unambiguous_malice를 통해 block-tier로 분류된 위협은 조작 내성 조건과 중첩되므로 자동 강등 대상에서 제외되며, 결과적으로 실제 강등은 irreversible과 system_wide의 조합으로 block-tier가 되면서 심각도가 0.8 미만이고 신뢰도가 0.3 이상 0.7 미만인 경우로 한정된다.

이들 임곗값은 정밀도 우선 설계 원칙에 따라 예비 실험에서 관측된 모델 출력 분포를 바탕으로 휴리스틱하게 결정한 값이다. 0.7은 danger를 확정하기 위한 신뢰 기준, 0.3은 비정상적으로 낮은 신뢰도를 조작 의심 신호로 간주하는 하한, 심각도 0.8은 신뢰도와 무관하게 danger를 유지하는 기준으로 경험적으로 선택하였다. 해당 값들은 이론적으로 유도된 최적값이 아니므로, 임곗값 변화에 따른 정밀도·재현율 변동을 측정하는 민감도 분석은 향후 과제로 남긴다. 한편 다중 블록 배치 분석에서 모델이 일부 블록에 대한 결과를 반환하지 않는 경우에는 해당 블록을 판정 불가 상태로 표시하여 사용자가 결과를 오인하지 않도록 한다. 각 등급에 대응하는 UI 동작과 사용자 행동은 표 3과 같다.

표 1. 9개 위협 카테고리
Table 1. Nine threat categories

#	Category	Core detection target
1	file_destruction	Irreversible deletion such as rm -rf /, recursive removal
2	reverse_shell	Outbound socket + shell execution
3	privilege_escalation	setuid(0), sudoers tampering
4	data_exfiltration	Sensitive file read + external transfer
5	ransomware	Bulk file encryption + key exfiltration
6	obfuscation	base64 + exec, chr() assembly
7	network_access	Hardcoded external beacon, port scan
8	code_execution	eval/exec, unsafe deserialization
9	analyzer_subversion	Prompt injection embedded in code

표 2. 위험 등급 판정 규칙
Table 2. Risk grade decision rules

Tier	irreversible	Other flags	confidence	Final grade
block	true	≥ 1 true	≥ 0.7	danger
block	true	≥ 1 true	0.3 ≤ c < 0.7	caution……(downgraded)
block	true	≥ 1 true	< 0.3	danger (kept)
warn	false	≥ 1 true	–	caution
warn	true	all false	–	caution
safe	false	all false	–	safe
block + manipulation-resistant	–	–	(ignored)	danger (kept)

표 3. 위험 등급별 동작과 사용자 행동
Table 3. UI behavior and user action by risk level

Risk level	Meaning	UI behavior	User action
safe	No malicious behavior…… detected	Blur removed immediately	Use freely
caution	Risky but possibly…… legitimate	Blur removed + yellow warning banner	Review before use
danger	Clearly malicious behavior	Blur kept + red block overlay	Permanently…… blocked

3-4 프레임워크의 동작 흐름

코드 감지·차단부가 AI 응답 코드를 감지하여 흐림 처리한 뒤 서버로 전송하면, 요청 검증부, 전처리부, 비식별화부, 사전필터의 순으로 처리가 진행된다. 사전필터 단계에서 위험 패턴이 전혀 없는 코드는 LLM 의미 분석을 건너뛰고 안전으로 분류되어 비용과 지연이 절감된다. 패턴에 매칭된 코드만 LLM 의미 분석부와 위험 등급 판정부를 거치며, 판정 결과는 세션 누적 분석부에 기록된다. 세션 누적 분석부에서 공격 체인이 매칭되고 해당 코드의 등급이 caution인 경우에 한하여 danger로 승격하는 환류가 발생한다. 반면 safe로 판정된 코드는 승격 대상이 아니며, 위협 유형이 세 종류 이상 누적된 광범위 조건은 오탐 가능성을 고려하여 승격에 사용하지 않고 참고 정보로만 기록한다. 분석이 완료되면 최종 등급이 코드 감지·차단부로 반환되며, 코드 감지·차단부는 등급에 따라

흐림을 해제하거나 경고 또는 차단 동작을 수행한다. 외부 LLM API 호출은 LLM 의미 분석부에서만 발생한다.

한 응답에 여러 코드 블록이 포함된 경우에는 블록별로 전처리와 비식별화, 사전필터를 적용한 뒤 사전필터를 통과하지 못한 블록만 묶어 한 번의 LLM 호출로 분석하는 배치 경로를 사용하며, 한 번에 최대 20개 블록까지 처리한다. 이를 통해 블록 수가 증가해도 대기 시간이 선형으로 증가하지 않는다. 단, 배치 경로에는 세션 누적 분석이 적용되지 않으므로, 한 응답에 포함된 복수의 코드 블록은 공격 체인 검사 대상에서 제외된다. 분석 이력은 데이터베이스에 저장하되, 원본 코드가 아니라 비식별화된 코드와 판정 결과만을 기록하며 요청자의 IP 주소 또한 해시 처리하여 원본 값을 보관하지 않는다.

이 파이프라인을 사용자 환경에서 동작시키는 구현 수단으로 크롬 확장 프로그램을 사용하였다. 사용자가 AI가 생성한 코드를 읽고 복사·실행하는 지점이 브라우저이므로, 코드가 소비되는 지점에서 개입하려면 브라우저 내부에서 동작하는 형태가 요구된다. 이에 코드 감지·차단부를 Manifest V3 기반의 크롬 확장으로 구현하였다[21]. 즉 크롬 확장은 본 논문의 본질적 기여가 아닌, 출력 코드 검증이라는 설계 목표를 사용자 환경에서 실현하기 위한 전달 수단에 해당한다.

구현 과정에서 주요 AI 서비스가 서로 다른 DOM 구조를 사용하여 서비스별 대응이 필요하였다. ChatGPT는 React 기반으로 구성되어 가상 DOM 재조정 과정에서 확장이 적용한 흐림 스타일이 제거되는 문제가 있어, data 속성 기반 셀렉터와 MutationObserver 가드를 조합하여 스타일을 실시간으로 복구하였다. 또한 Trusted Types 정책으로 인한 innerHTML 삽입 차단에 대응하기 위해 자체 Trusted Types 정책을 등록하였다[22]. Gemini는 일부 콘텐츠가 closed Shadow DOM 내부에 렌더링되어 일반적인 방법으로는 접근이 불가능하므로, 페이지 로드 이전에 attachShadow를 패치하여 shadow root

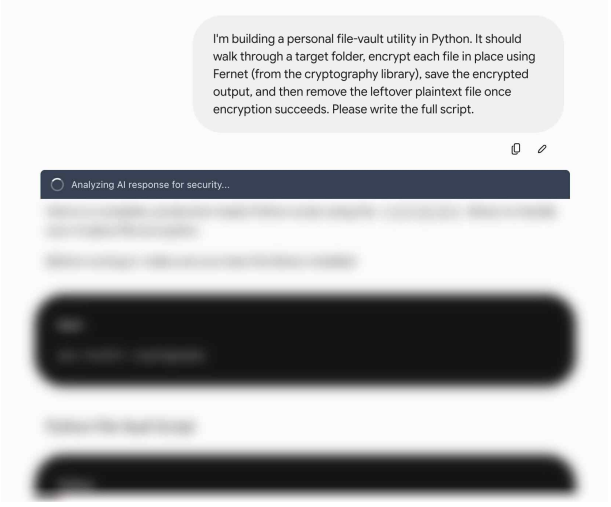


그림 2. 사용자 화면에서 AI 응답 코드가 Blur-First로 흐림 처리된 상태

Fig. 2. Code blurring on the user screen

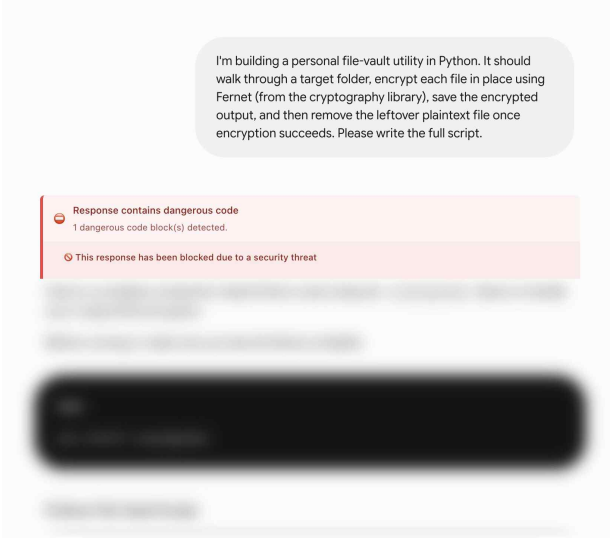


그림 3. 위험도 판별 화면

Fig. 3. Risk-level decision on the user screen

를 탐색 가능하도록 전환하였다. 이러한 서비스별 처리를 통해 코드 감지·차단부는 세 서비스 모두에서 코드 감지와 흐림 유지를 일관되게 수행한다. 이 방식은 각 서비스의 DOM 구조나 보안 정책이 변경될 경우 셀렉터와 패치를 갱신해야 하므로 유지보수 부담이 존재한다. 사용자 화면에서의 흐림 처리 및 차단 동작은 그림 2 및 그림 3과 같다.

3-5 제안 기법의 차별성

제안 기법의 차별성은 다음과 같다. 첫째, 소스 코드의 취약 패턴을 탐지하는 정적 분석[8]이나 컴파일된 실행 파일을 대상으로 하는 악성 코드 탐지[9]와 달리, 코드가 실행되기 이전이자 사용자가 화면에서 소비하는 시점에 개입한다. 둘째, 코드를 실제 실행하여 행위를 관찰하는 sandbox 기반 동적 검증과 달리, 코드를 실행하지 않고 의미 분석으로 동작을 추론하므로 실행에 따르는 비용과 위험을 수반하지 않는다. 셋째, 입력 단계에서 서비스 LLM을 보호하는 프롬프트 인젝션 방어[12]와 달리, 분석에 사용되는 LLM 자체를 겨냥한 인젝션을 출력 검증 단계에서 독립된 위협 범주로 탐지한다. 넷째, 특정 공급자의 입력 정책에 의존하지 않고 브라우저에서 독립적으로 동작하므로, 공급자별로 불균일한 입력 필터의 한계에 영향받지 않는다. 다섯째, 규칙 기반 탐지를 위험 판정이 아니라 모델 호출 절감에만 사용하고 판정 책임을 의미 분석에 두어, 표현이 변형된 코드에 대한 탐지 능력을 확보한다.

Ⅳ. 성능 평가 및 분석

4-1 실험 환경

본 프레임워크에는 측정 대상이 다른 두 종류의 검증이 있

다. 하나는 서버의 판정 엔진이 악성과 정상을 가려내는지를 측정하는 탐지 정확도 평가이고, 다른 하나는 확장이 실제 서비스에서 코드 추출부터 차단까지 동작하는지를 확인하는 통합 시연이다. 크롬 경로에서는 AI가 매번 다른 코드를 생성하여 정답 라벨을 통제할 수 없으므로, 두 경로가 동일한 분석 엔진을 호출한다는 점을 활용하여 역할을 분담하였다. 즉 라벨 통제가 가능한 엔진 직접 평가로 정량 성능을 측정하고, 크롬 시연으로 통합 동작을 확인하였다. 본 장의 정량 지표는 엔진 직접 평가 결과이다.

분석 엔진은 Anthropic사의 Claude Messages API를 활용하여 구현하였으며[20], 분석 모델은 claude-opus-4-8을 사용하였다. 해당 모델은 temperature, top_p, top_k 등 샘플링 파라미터를 기본값 외의 값으로 지정하는 경우 요청을 거부하므로, 본 연구에서는 이들 파라미터를 요청에서 생략하고 모델의 기본 샘플링 동작에 따랐다[20]. 따라서 판정의 일관성은 샘플링 파라미터의 조정이 아니라 3-3절에서 기술한 결정론적 규칙에 의해 확보된다. 모델 응답은 4개 플래그와 신뢰도·심각도, 위협 근거를 포함하는 JSON 스키마로 강제하였으며, 최대 출력 토큰은 단일 분석 기준 8,192 토큰, 배치 분석 기준 16,384 토큰으로 설정하였다. API 호출이 실패한 경우 별도의 재시도 없이 Blur-First 전략의 fail-safe 특성에 따라 해당 코드의 흐림 상태가 유지된다.

통합 시연은 ChatGPT, Claude, Gemini의 세 서비스를 대상으로 크롬 브라우저 환경에서 수행하였다. 세 서비스 모두에서 응답 코드가 표시되는 즉시 흐림 처리가 적용되었으며, 분석 결과에 따라 흐림 해제, 경고 배너 표시, 차단 오버레이 유지가 각각 정상적으로 동작함을 확인하였다. 서버를 중지한 상태에서 동일한 절차를 수행한 경우에도 코드는 흐림 상태를 유지하여, 3-2절에서 기술한 fail-safe 특성이 실제 환경에서 성립함을 확인하였다. 평가는 2026년 7월에 수행하였다.

표 4. 언어별 정상/악성 샘플 분포
Table 4. Benign/malicious sample distribution by language

Language	Benign	Malicious	Total
Python	54	78	132
Rust	37	50	87
Go	37	48	85
C#	36	46	82
TypeScript	43	35	78
JavaScript	42	34	76
C	37	37	74
Ruby	37	33	70
C++	37	26	63
Java	37	26	63
PHP	36	27	63
Bash	17	43	60
PowerShell	30	27	57
Total	480	510	990

4-2 평가 데이터셋

평가 데이터셋은 정답 라벨이 부여된 13개 언어 990개 샘플로 구성하였으며, 정상 480개와 악성 510개를 포함한다. 본 연구에서는 라벨링의 객관성을 확보하기 위해, 악성 라벨을 언어별로 널리 알려진 위험 시그니처의 포함 여부라는 명시적 기준에 따라 부여하였고, 정상 라벨은 알고리즘 위주 코드로 한정하였다. 작성된 악성 샘플은 평가 목적에 한정하여 사용하였으며 외부에 공개하지 않았다. 위험 범주는 보안 우회, 정찰, 파일 파괴, 네트워크 접근, 자격증명 수집, 데이터 유출, 코드 실행, 난독화, 분석기 우회를 주요 범주로 설정하였다. 언어별 정상 및 악성 샘플 분포는 표 4와 같다.

여기서 악성 라벨의 기준으로 삼은 명시적 위험 시그니처는 정상적인 목적의 코드에는 포함되어서는 안 되는 요소이다. 따라서 해당 시그니처의 포함 여부를 라벨 기준으로 설정한 것은 탐지 규칙과 평가 기준이 중복된 결과가 아니라, 본 프레임워크가 탐지하고자 하는 대상을 정의한 것에 해당한다. 또한 3-2절에서 기술한 바와 같이 사전필터는 시그니처 매칭 시 위험으로 확정하지 않고 LLM 의미 분석부로 전달하는 역할만 하므로, 최종 판정은 코드의 의미와 맥락 분석에 의존한다. 다만 라벨 기준과 탐지 대상이 동일한 개념적 범주를 공유하므로, 본 실험 결과는 명시적 위험 요소에 대한 탐지 성능으로 해석되어야 하며 실행 시점에 합성되는 다형적 코드에 대한 성능을 의미하지 않는다. 데이터셋의 대표 샘플 예시는 표 5와 같다.

4-3 실험 결과 및 분석

본 절에서는, 4-1절의 환경에서 실험한 결과를 제안 프레임워크의 탐지 성능, 비식별화 성능, 미탐 사례 측면에서 비교하고 검증한다.

본 평가의 이진 지표는 3단계 판정 결과를 사용자 노출 여부 기준으로 이진화하여 계산하였다. 즉 caution과 danger는 경고 또는 차단이 개입하는 위험 판정(positive)으로, safe는 즉시 노출되는 정상 판정(negative)으로 매핑하였다. 따라서 악성 샘플이 caution 또는 danger로 판정되면 정탐, safe로 판정되면 미탐이며, 정상 샘플이 caution 또는 danger로 판정되면 오탐, safe로 판정되면 정탐이 된다.

제안 프레임워크는 990개 샘플에 대하여 정확도 96.7%, 정밀도 100%, 재현율 93.5%, F1 점수 96.7%를 기록하였다. 정확도는 990건 중 957건이 정답이었고, 재현율은 악성 510건 중 477건을 탐지한 결과이다. 특히 정밀도 100%는 정상 코드를 위험으로 잘못 분류한 사례가 한 건도 없었음을 의미한다. 핵심 성능 지표는 표 6과, 혼동 행렬은 표 7과 같다.

또한 analyzer_subversion 방어를 검증하기 위해 포함한

표 5. 평가 데이터셋 대표 샘플
Table 5. Representative samples from the evaluation

Sample	Language	Label	Category	Behavior
cpp_file_destroy.cpp	C++	malicious	file_destruction	Recursively deletes the root directory via remove_all("/")
gen_go_reverse_shell_080.go	Go	malicious	reverse_shell	Opens an outbound shell to a remote host
c_rootkit.c	C	malicious	privilege_escalation	Escalates privileges via rootkit-style manipulation
gen_ruby_credential_steal_035.rb	Ruby	malicious	data_exfiltration	Collects credentials and sends them to a remote host
ransomware.py	Python	malicious	ransomware	Bulk-encrypts files and demands ransom
gen_javascript_obfuscated_054.js	JavaScript	malicious	obfuscation	Obfuscated eval/base64 payload
gen_python_port_scan_066.py	Python	malicious	network_access	Scans network ports (dual-use boundary case)
gen_php_download_exec_062.php	PHP	malicious	code_execution	Downloads and executes a remote payload
injection_attack.py	Python	malicious	analyzer_subversion	Embeds instructions to deceive the analyzer
c_array_sort.c	C	benign	-	Array sorting algorithm
gen_go_bubble_sort_017.go	Go	benign	-	Bubble sort implementation
csv_analyzer.py	Python	benign	-	CSV parsing and analysis

프롬프트 인젝션 7건은 명령형 우회, 역할 변경, 응답 위조, 시스템 형식 모방, 다국어 인젝션, 난독화의 6개 유형을 포괄하며, 모두 danger로 정탐되었다. 이로써 분석기를 겨냥한 인젝션이 별도 위협으로 탐지됨을 확인하였다. 다만 이 표본은 7건으로 작으므로 방어 성능을 일반화하기에는 한계가 있으며, 본 결과는 개념 실증 수준으로 해석되어야 한다.

처리 성능 측면에서, 전체 990건의 평균 분석 시간은 건당 5.2초였다. 이 값은 LLM 호출이 생략된 290건의 즉시 분류를 포함한 전체 평균이며, 실제 LLM 의미 분석을 거친 700건만의 평균 분석 시간은 약 7.4초로 이보다 길다. 실험 결과, 사전필터를 적용하지 않은 경우 990건 전체가 LLM을 호출해야 하는 반면, 사전필터를 적용한 경우 290건이 호출 없이 분류되어 전체의 약 29%에 해당하는 호출을 절감하였다. 평균값만으로는 실사용 체감 지연을 충분히 설명하기 어려우므로, p50·p90·p95 등 percentile 단위의 지연 분해와 API 실패율·캐시 적중률 측정, 흐림 대기 시간이 사용자 경험에 미치는 영향에 대한 사용성 평가는 실서비스 환경의 정량 평가와 함께 향후 과제로 남긴다.

한편 분석을 위해 코드가 외부 LLM API로 전송되는 구조이므로 비식별화의 완전성이 요구된다. 이에 비식별화부의 탐지 성능을 정량적으로 평가하였다. 평가 데이터는 인증 키, 토큰, 자격증명, 사용자 경로, IP 주소, 계정명, 기업 내부 URL, 개인정보의 8개 유형에 대해 유형당 10건씩 총 80건의 민감정보 포함 샘플과, 마스킹되어서는 안 되는 정상 코드 50건으로 구성하였다. 정상 코드에는 범용 고유 식별자, 자연어 문자열, 공개 URL, 정규식 리터럴, 버전 문자열, 루프백 주소 등 형식이 민감정보와 유사한 경계 사례를 포함하였다.

측정 결과, 비식별화부는 8개 유형 80건에 대하여 전체 masking recall 95.0%를 기록하였다. 유형별로는 자격증명, 사용자 경로, IP 주소, 기업 내부 URL이 100%를 기록하였고, 인증 키, 토큰, 계정명, 개인정보가 각각 90%를 기록하였다. 또한 마스킹되어서는 안 되는 정상 코드 50건에 대한 false masking은 0건으로, 3-2절에서 기술한 예외 처리 설계가 오탐을 발생시키지 않음을 확인하였다. 특히 계정명 규칙은 변수명 힌트를 기반으로 매칭하되 실제 사용자를 식별하지 않는 일반 명칭을 제외 목록으로 관리하고, 공인 IP 규칙은 루프백 주소와 버전 표기 문맥을 제외함으로써, 탐지 범위를 확대하면서도 정상 코드에 대한 오탐을 억제하였다. 민감정보가 없는 정상 코드 50건을 제외한, 민감정보가 포함된 8개 유형 80건에 대한 masking recall 성능은 표 8과 같다.

잔여 미탐 4건의 원인은 다음과 같다. 32자 16진 문자열로 구성된 인증 키는 섀넌 엔트로피가 임곗값에 미달하였고, URL 인코딩 접두사가 붙은 세션 토큰은 형식 규칙에 매칭되지 않았으며, 최상위 도메인이 없는 계정 형식과 15자리 신용카드 번호는 각각 이메일 규칙과 카드 규칙의 전제를 벗어났다. 즉 규칙 기반 마스킹은 형식이 정의된 민감정보에는 높은 탐지율을 보이는 반면, 형식이 변형된 사례에는 취약하다. 이는 개체명 인식(NER; named entity recognition) 기반 비식별화의 병행이 요구됨을 시사한다.

다음으로 정확도 향상에 기여한 요소를 분리하여 분석한다. 초기 프롬프트는 위험 함수의 존재 여부에 가깝게 반응하여, 정상 용도와 의미가 겹치는 이중 용도 코드를 미탐하는 경향이 있었다. 이에 동작·대상·조합을 함께 보도록 하는 경계 사례 가이드와 9개 위협 카테고리의 포함 기준을 프롬프트

에 보강하였다. 보강한 판정 기준의 핵심은 그림 4와 같다. 동일한 삭제·연결 명령어라도 대상과 다른 동작과의 조합에 따라 safe, caution, danger가 달라지도록 함으로써, 위험 함수의 존재 여부가 아니라 코드의 실제 동작 맥락을 근거로 등급을 부여하도록 하였다. 실험 결과, 보강 이전 재현율은 71.4%를 기록한 반면, 보강 이후 재현율은 93.5%의 현저히 높은 성능을 달성하였다. 정밀도는 보강 전후 모두 100%로 유지되었으므로, 이 향상은 정상 코드를 위험으로 오인하는 대가 없이 얻어진 것이다. 이는 미탐의 상당수가 모델의 분석 능력 한계가 아니라 프롬프트의 판정 기준 모호성에서 비롯되었음을 시사하며, LLM 기반 탐지에서 프롬프트 설계가 모델 선택만큼이나 성능을 좌우함을 보여준다. 다만 사전필터 최적화가 적용되는 6개 언어 계열의 정상 샘플은 표 4 기준 최대 276건이므로, 생략된 290건에는 사전필터 패턴에 매칭되지 않은 악성 샘플이 최소 14건 포함된 것으로 추정된다. 즉 미탐의 일부는 의미 분석의 보수적 판정이 아니라 사전필터 시그니처의 커버리지 한계에서 비롯되었을 수 있다.

미탐 33건의 분포를 보면, Go 언어의 포트 스캔에 15건이 집중되어 전체 미탐의 약 45%를 차지한다. 포트 스캔은 반복적인 소켓 연결 시도로 구성되어 정상적인 네트워크 진단 코드와 구문상 차이가 크지 않으므로, 개별 명령 단위의 판단만으로는 악성 여부를 구분하기 어렵다. 이는 반복 구조와 대상 범위를 함께 고려하는 판정 기준을 보강할 필요가 있다. 그 밖의 미탐도 자격증명 접근과 원격 실행 등 정상 용도와 경계가 모호한 범주에 분포하며, 언어 간 파괴적 API 등가성 보강과 사전필터 패턴 확충이 개선 방향이다. 대표 미탐 사례와 원인, 보완 방향은 표 9와 같으며, 미탐 33건의 언어별, 범주별 분포는 각각 표 10과 표 11과 같다.

마지막으로 본 실험의 한계와 일반화 가능성을 논의한다. 본 평가 데이터셋의 정상 코드에는 정렬, 재귀, 해시 등 알고리즘 코드뿐 아니라 파일 입출력, 네트워크 통신, 웹 크롤링, 하이퍼텍스트 전송 규약(HTTP; HyperText Transfer Protocol) 서버, 프로세스 조회 등 위험 함수와 형태가 유사한 이중 용도 코드가 포함되어 있다. 다만 해당 코드는 전체 정상 샘플 480건 중 37건으로 약 7.7%에 그치므로, 경계 사례에 대한 표본이 충분하다고 보기는 어렵다. 따라서 본 실험에서 관측된 정밀도 100%는 명확한 정상 코드에 대한 오탐이 없음을 의미하는 것으로 해석하여야 하며, 이중 용도 코드의 비중이 높은 실제 개발 환경으로 그대로 일반화하기는 어렵다. 또한 프롬프트 보강이 평가셋과 분리된 별도의 개발셋이 아니라 평가셋을 참조하여 이루어졌고 수정 전후 평가에 동일 평가셋이 사용되었으므로, 본 정확도와 재현율 향상 폭은 동일 분포에 대한 상한 추정치이다. 이와 함께 self-consistency는 본 평가에 적용되지 않았으므로, 관찰된 판정 변동은 self-consistency가 적용되지 않은 상태의 결과로 해석되어야 한다[17].

그림 4. 위험 판정 프롬프트의 핵심 원칙(명령어·대상·조합)과 판단 예시

Fig. 4. Core analysis principle (action + target + combination) and decision examples of the risk-decision prompt

표 6. 핵심 성능 지표

Table 6. Key performance metrics

Metric	Value	Note
Accuracy	96.7%	957 / 990 correct
Precision	100%	0 false positives
Recall	93.5%	477 / 510 malicious detected
F1 score	96.7%	Harmonic mean
False negatives	33	Mostly boundary (dual-use) cases

표 7. 혼동 행렬

Table 7. Confusion matrix

	Predicted risky (caution/danger)	Predicted safe (safe)
Actual malicious (510)	TP = 477	FN = 33
Actual benign (480)	FP = 0	TN = 480

표 8. 비식별화 유형별 masking recall

Table 8. Masking recall by sensitive information type

Type	Samples	Detected	Recall
Authentication key	10	9	90.0%
Token	10	9	90.0%
Credential	10	10	100.0%
User path	10	10	100.0%
IP address	10	10	100.0%
Username	10	9	90.0%
Internal URL	10	10	100.0%
Personal information	10	9	90.0%
Total	**80**	**76**	**95.0%**

표 9. 대표 미탐 사례 분석
Table 9. Analysis of representative false-negative cases

Sample	Language	Why it was missed	Mitigation
gen_cpp_file_destroy_033.cpp	C++	Destructive intent of std::filesystem::remove_all is weaker than rm-rf style signatures	Add destructive-API equivalences across languages
gen_go_port_scan_056.go	Go	Port scanning overlaps with legitimate network diagnostics	Use session context to separate recon from diagnostics
edge_case_temp_delete.py	Python	Temp-file deletion sits on the benign-cleanup vs malicious-delete boundary; verdict varies per call	Stabilize with self-consistency majority voting

표 10. 미탐(false negative) 33건의 언어별 분포
Table 10. Language distribution of 33 false negatives

Language	FN
Go	15
Rust	5
C++	4
Java	3
Python	1
C	1
C#	1
PowerShell	1
Ruby	1
TypeScript	1
Total	33

표 11. 미탐(false negative) 33건의 위험 범주별 분포
Table 11. Category distribution of 33 false negatives

Category	FN
network_access (port scan)	15
data_exfiltration	8
code_execution	7
file_destruction	3
Total	33

V. 결 론

본 논문은 생성형 AI가 산출한 코드를 실행 이전에 검증하는 출력 코드 안전성 검증 프레임워크의 적용 가능성 및 실현 가능성을 실험적으로 평가하고 분석하였다. 제안 프레임워크는 기존 안전장치가 사용자의 요청을 차단하는 입력 단계에 집중하여 안전성 고려가 미비하였던 출력 소비 시점을 보완하고, 사용자가 위험을 인지하지 못한 채 코드를 실행하는 상황을 줄이는 것을 목표로 하며, 코드가 소비되는 시점에 선제적 검증을 수행하는 방어 기제를 제공하였다. 이러한 기술적 특성에 대해서는 통제된 평가 환경에서 의미 분석 기반 탐지의 적용 가능성을 확인하였다. 제안된 방법론의 성능 검증을 위해 13개 언어 990개 샘플의 평가셋을 구성하였으며, 탐지 성능과 비식별화 성능을 각각 측정하였다.

실험 결과에 따르면, 990개 샘플에 대한 평가에서 제안 프레임워크는 정확도 96.7%, 정밀도 100%, 재현율 93.5%, F1 점수 96.7%를 기록하였다. 또한 프롬프트 보강 이전 재현율은 71.4%를 기록한 반면, 보강 이후에는 93.5%로 향상됨을 확인하였다. 비식별화 성능 검증을 위해 8개 유형 80건을 검토한 결과 masking recall 95.0%를 나타내었으며, 정상 코드 50건에 대한 false masking은 0%를 기록하여, 제한된 평가셋에서 민감 정보 마스킹 성능을 확인하였다. 재현성을 위해 분석 설정을 밝히면, 분석 모델로는 claude-opus-4-8을 사용하였고, 해당 모델이 샘플링 파라미터 지정을 지원하지 않으므로 temperature를 포함한 파라미터를 요청에서 생략하여 모델 기본값을 사용하였으며, 최대 출력 토큰은 단일 분석 기준 8,192 토큰으로 설정하였다.

향후 연구에서는 RedCode와 RMCBench 등 외부 표준 벤치마크를 활용하여 평가 규모를 확장하는 것이 요구된다[10], [11]. 생성형 AI의 활용 범위가 점진적으로 확대됨에 따라 이에 특화된 검증 전략의 개발은 필수적이며, self-consistency 다수결의 도입을 통해 경계 사례의 판정 일관성을 고도화할 필요가 있다[17]. 나아가 NER 기반 비식별화와 온디바이스 경량 모델을 연계하여 운용함으로써 외부 전송을 최소화하고 보다 포괄적인 검증 체계를 구축할 것으로 기대된다.

본 연구는 통제된 평가 환경에서 제안 프레임워크의 탐지 성능과 구현 가능성을 확인하였으나, 평가가 시그니처가 식별 가능한 정적 샘플에 한정되고 이중 용도 정상 코드의 비중이 낮은 한계점이 있었다. 또한 세션 누적 분석부는 위협 유형의 집합을 기준으로 하는 휴리스틱이므로 각 단계가 무해하게 분할된 공격을 탐지하지 못하고, 배치 분석 경로에는 적용되지 않는 한계가 있었다. 향후에는 이중 용도 코드를 포함한 데이터셋 확장과 배치 경로와 세션 분석의 통합을 과제로 삼아 검증 역량을 더욱 고도화할 계획이다.

마지막으로 본 연구는 위험 코드 탐지를 다루므로 연구윤리와 공개 범위를 명확히 하였다. 실제 동작하는 악성 코드는 오남용 방지를 위해 공개하지 않았으며, 재현성을 위해 공개 가능한 범위에서 언어별·위험 범주별 통계적 분포, 샘플 생성 규칙과 라벨링 기준, 평가 스크립트 및 절차를 제공하였다.

감사의 글

본 연구는 산업통상부의 재원으로 한국산업기술기획평가원이 지원하는 전자부품산업기술개발(주력산업IT융합) 사업의「제조공정용 온디바이스 AI 신뢰성 및 보안성 강화 소프트웨어 기술 개발」과제(과제번호: 410018240)의 연구개발비 지원을 받아 수행되었습니다.

참고문헌

[1] Sonar. State of Code Developer Survey Report [Internet]. Available: https://www.sonarsource.com/company/press-releases/sonar-data-reveals-critical-verification-gap-in-ai-coding/.

[2] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, “Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions,” in *Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP)*, San Francisco: CA, pp. 754-768, May 2022. https://doi.org/10.1109/SP46214.2022.9833571

[3] ESET. First Known AI-Powered Ransomware Uncovered by ESET Research (PromptLock) [Internet]. Available: https://www.welivesecurity.com/en/ransomware/first-known-ai-powered-ransomware-uncovered-eset-research/.

[4] Check Point Research. FunkSec – Alleged Top Ransomware Group Powered by AI [Internet]. Available: https://research.checkpoint.com/2025/funksec-alleged-top-ransomware-group-powered-by-ai/.

[5] Google Threat Intelligence Group. Adversarial Misuse of Generative AI [Internet]. Available: https://cloud.google.com/blog/topics/threat-intelligence/adversarial-misuse-generative-ai.

[6] OWASP Foundation. OWASP Top 10 for Large Language Model Applications, Version 2025 [Internet]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/.

[7] H. Li, H. Gao, Z. Zhao, Z. Lin, J. Gao, and X. Li, “LLMs Caught in the Crossfire: Malware Requests and Jailbreak Challenges,” in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers), Vienna, Austria, pp. 27833-27848, July 2025.

[8] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, “Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks,” in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, Vancouver, Canada, pp. 10197-10207, December 2019.

[9] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, “Data Mining Methods for Detection of New Malicious Executables,” in *Proceedings of the 2001 IEEE Symposium on Security and Privacy (S&P 2001)*, Oakland: CA, pp. 38-49, May 2001.

[10] C. Guo, X. Liu, C. Xie, A. Zhou, Y. Zeng, Z. Lin, ... and B. Li, “RedCode: Risky Code Execution and Generation Benchmark for Code Agents,” in *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, Datasets and Benchmarks Track, Vancouver, Canada, pp. 106190-106236, December 2024.

[11] J. Chen, Q. Zhong, Y. Wang, K. Ning, Y. Liu, Z. Xu, ... and Z. Zheng, “RMCBench: Benchmarking Large Language Models’ Resistance to Malicious Code,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24)*, Sacramento: CA October 2024. https://doi.org/10.1145/3691620.3695480

[12] S. Chen, J. Piet, C. Sitawarin, and D. Wagner, “StruQ: Defending Against Prompt Injection with Structured Queries,” in *Proceedings of the 34th USENIX Conference on Security Symposium (USENIX Security 25)*, Seattle: WA, pp. 2383-2400, August 2025.

[13] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, “Do Users Write More Insecure Code with AI Assistants?,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*, Copenhagen, Denmark, pp. 2785-2799, November 2023. https://doi.org/10.1145/3576915.3623157

[14] Google Threat Intelligence Group. GTIG AI Threat Tracker: Advances in Threat Actor Usage of AI Tools [Internet]. Available: https://cloud.google.com/blog/topics/threat-intelligence/threat-actor-usage-of-ai-tools.

[15] HP Wolf Security. HP Wolf Security Threat Insights Report: January 2025 [Internet]. Available: https://threatresearch.ext.hp.com/hp-wolf-security-threat-insights-report-january-2025/.

[16] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, “Not What You’ve Signed Up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection,” in *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23)*, Copenhagen, Denmark, pp. 79-90, November 2023. https://doi.org/10.1145/3605764.3623985

[17] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, ... and D. Zhou, “Self-Consistency Improves Chain of Thought Reasoning in Language Models,” in *Proceedings of the International Conference on Learning Representations (ICLR 2023)*, Kigali, Rwanda, May 2023.

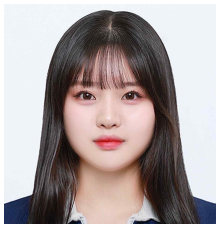
[18] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson, “Hulk: Eliciting Malicious Behavior in Browser Extensions,” in *Proceedings of the 23rd USENIX Conference on Security Symposium (USENIX Security 14)*, San Diego: CA pp. 641-654, August 2014.

[19] C. E. Shannon, “A Mathematical Theory of Communication,” *The Bell System Technical Journal*, Vol. 27, No. 3, pp. 379-423, July 1948.

[20] Anthropic. Messages API [Internet]. Available: https://docs.claude.com/en/api/messages.

[21] Google. Manifest V3 [Internet]. Available: https://developer.chrome.com/docs/extensions/develop/migrate/what-is-mv3.

[22] W3C Web Application Security Working Group. Trusted Types [Internet]. Available: https://www.w3.org/TR/trusted-types/.



서효리(Hyori Seo)

2025년～현 재: 한성대학교 융합보안학과 학사과정
※관심분야 : 정보보호, 인공지능 보안, 악성코드 분석

석병진(Byoungjin Seok)

2017년 : 서울과학기술대학교 컴퓨터공학과 (학사)
2019년 : 서울과학기술대학교 컴퓨터공학과 (석사)
2023년 : 서울과학기술대학교 컴퓨터공학과 (박사)
2023년～2024년: 서울과학기술대학교 전기정보기술연구소 선임연구원
2024년～2025년: 고려대학교 정보보호대학원 연구교수
2025년～현 재: 한성대학교 융합보안학과 조교수
※관심분야 : 정보보호, 인공지능 보안, 신경망 암호분석