

CVE-RAG: AI 코드 생성기의 취약의 존성 자동 차단을 위한 보안 템플릿 기반 프레임워크

A Security Template-Based Framework for Automatic Blocking of
Vulnerable Dependencies in AI Code Generators

목차



- 1 연구 배경 및 문제 정의
- 2 핵심 개념 및 용어 설명
- 3 기존 연구 및 한계
- 4 CVE-RAG 프레임워크 아키텍처
- 5 시스템 시연
- 6 실험 결과 및 분석
- 7 결론 및 향후 연구

연구 배경

AI 코드 생성 도구의 부상

- GitHub Copilot, Cursor 등 AI 코드 생성 도구 확산
- 자연어 프롬프트로 코드 자동 생성 → 개발 생산성 향상
- 대규모 데이터셋으로 사전 학습된 모델 활용

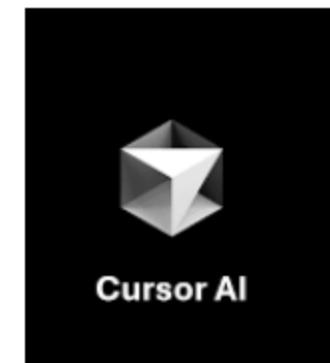
핵심 문제

- 학습 데이터 시점 제한으로 최신 보안 취약점 인식 불가
- 취약한 라이브러리 버전 추천 문제 발생
- 단순 "최신 버전 사용" 지시의 한계

LLM (Large Language Model)

정의 및 특징

- 대규모 텍스트 데이터로 학습된 인공지능 모델
- 자연어 이해 및 생성 능력 보유
- 코드 생성에 활용 (GitHub Copilot, Cursor 등)



Github Copilot

한계점

- 학습 데이터 시점 이후 정보 부재
- 취약 라이브러리 버전 추천 위험

A message from Cursor AI: Pillow와 Django를 사용해서 파이썬 프로그램을 만드려고 해. requirements.txt를 작성해줘.

네, Django와 Pillow를 사용한 파이썬 프로젝트를 위한 requirements.txt 파일을 작성해 드리겠습니다.

```
Django==5.0.4
Pillow==10.2.0
asgiref==3.7.2
sqlparse==0.4.4
tzdata==2024.1
```

Pillow buffer overflow vulnerability
High severity GitHub Reviewed Published on Apr 3, 2024 to the GitHub Advisory Database • Updated on Nov 19, 2024

Vulnerability details Dependabot alerts 1

Package	Affected versions	Patched versions
pillow (pip)	< 10.3.0	10.3.0

이 파일을 프로젝트에 추가하거나 배포하기 전에 업데이트를 권장합니다.

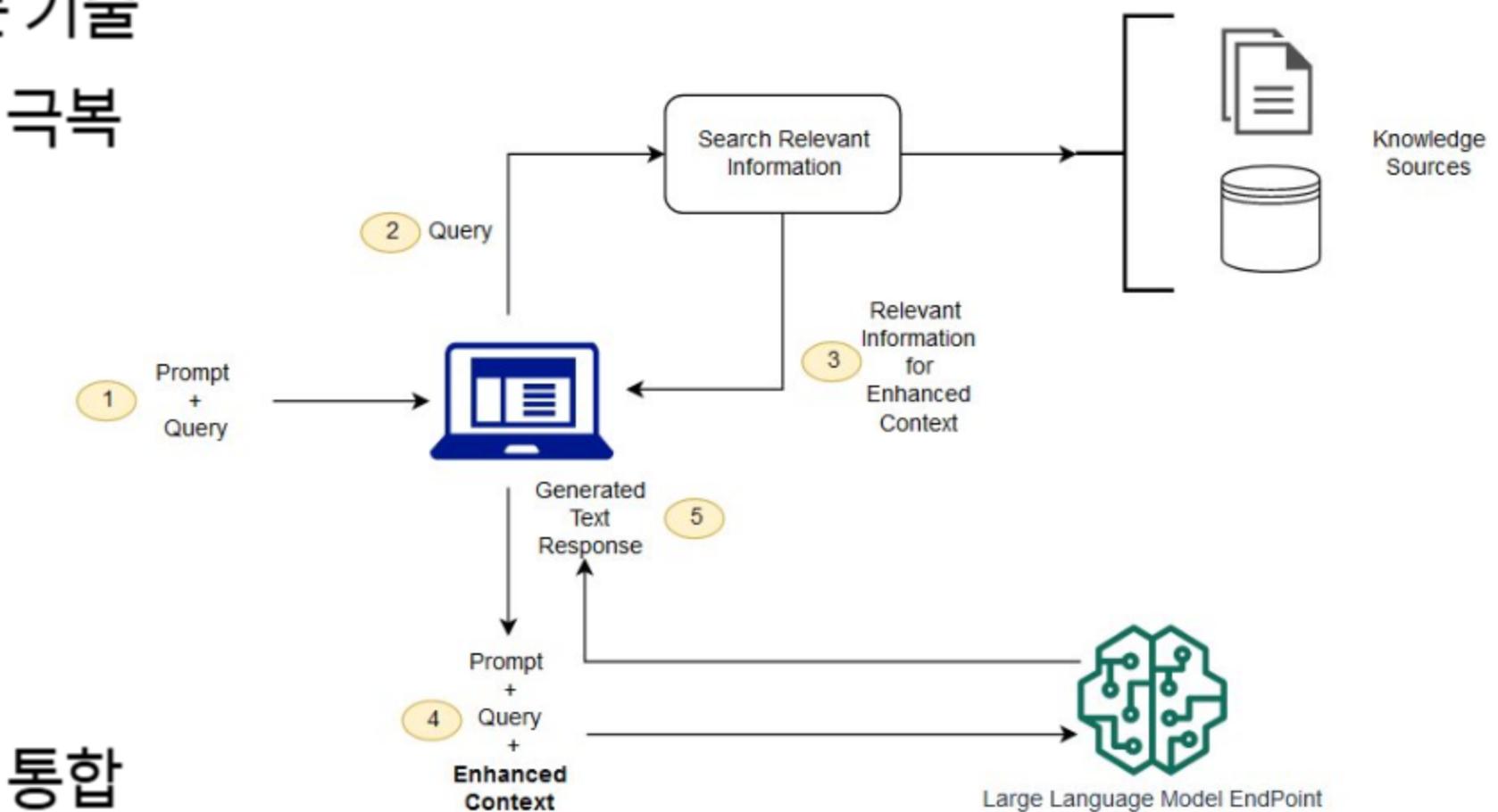
RAG (Retrieval-Augmented Generation)

개념

- 외부 지식을 검색하여 LLM의 응답을 보강하는 기술
- 최신 정보 활용 가능 → 학습 데이터 시점 한계 극복

작동 원리

- 쿼리 생성: 사용자 입력 기반 검색 쿼리 형성
- 문서 검색: 관련 문서/데이터 검색
- 관련 문서 추출: 유사도 기반 관련 정보 선별
- LLM 컨텍스트 추가: 검색 결과를 LLM 입력에 통합
- 향상된 응답 생성: 외부 지식 기반 정확한 응답 생성



구현 방식

- TF-IDF/코사인 유사도 검색 활용

CVE (Common Vulnerabilities and Exposures)

정의

- 공개된 사이버 보안 취약점을 식별하는 표준 시스템
- 고유 식별자(CVE ID) 부여 → CVE-YYYY-NNNNN 형식

CVE 데이터 구조

- 취약점 식별자 (CVE-ID)
- 영향받는 소프트웨어/버전
- 취약점 설명 및 심각도
- 패치된 버전 정보

중요성

- 보안 취약점 식별 및 추적 표준화
- 패치 관리 및 보안 업데이트 기준

The screenshot shows the CVE website interface. At the top, there's a navigation bar with links for 'CVE List', 'CNAs', 'About', 'WGs', 'News & Blog', and 'Board'. To the right, there's a logo for 'NVD' with links for 'CVSS Scores' and 'CPE Info'. Below the navigation is a search bar with options for 'Search CVE List', 'Downloads', 'Data Feeds', 'Update a CVE Record', and 'Request'. A message indicates 'TOTAL CVE Records: 188256'. A notice states: 'NOTICE: Transition to the all-new CVE website at [WWW.CVE.ORG](#) is underway and will last up to one year. ([details](#))'. Another notice says: 'NOTICE: Changes coming to [CVE Record Format JSON](#) and [CVE List Content Downloads](#) in 2022.' The main content area shows the details for CVE-2022-24329, including the CVE-ID, a link to the National Vulnerability Database (NVD), and a description: 'In JetBrains Kotlin before 1.6.0, it was not possible to lock dependencies for Multiplatform Gradle Projects.'

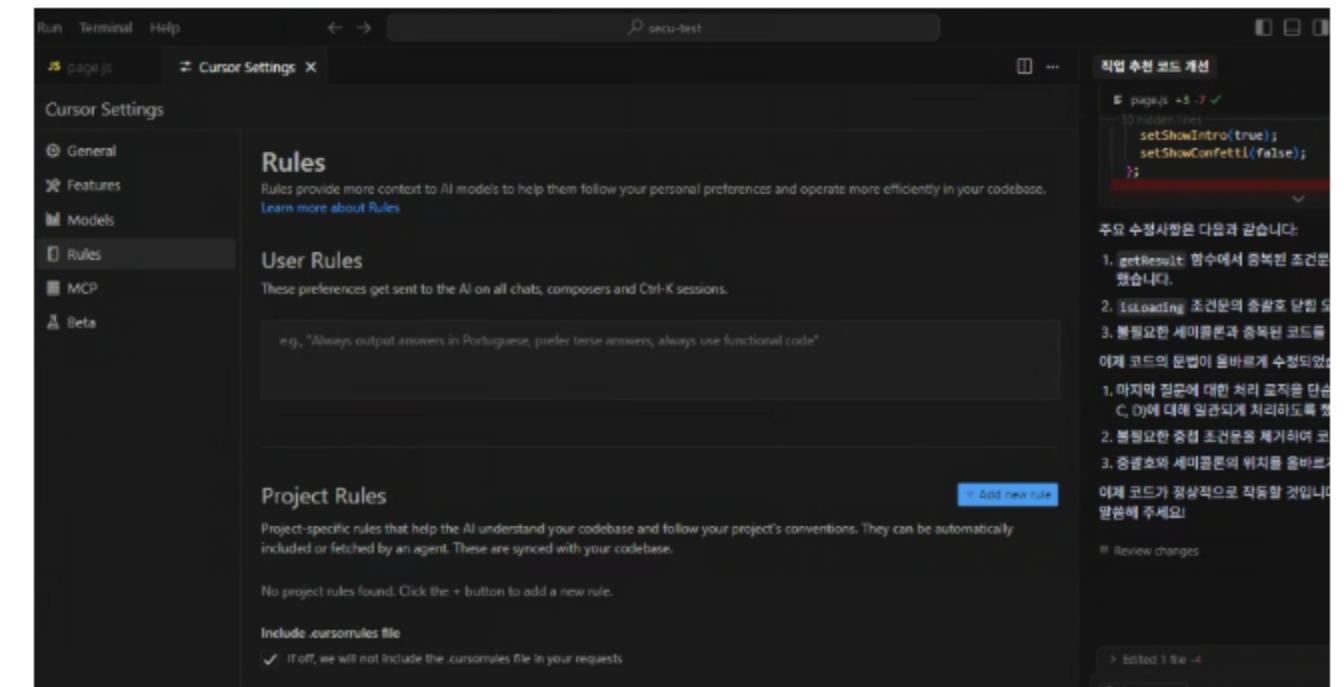
AI 코드 생성 도구

주요 도구

- GitHub Copilot: 코드 자동 완성, 함수 생성
- Cursor: AI 기반 코드 에디터, 코드 생성/편집 보조
- 기타: Amazon CodeWhisperer, Tabnine 등

Cursor 도구의 특징

- 자연어 프롬프트로 코드 생성
- .cursorrules 템플릿으로 코드 생성 가이드라인 제공
- 보안 규칙 적용 가능



문제 정의

AI 코드 생성기의 취약 의존성 문제

AI 코드 생성기의 취약점 현황

- 인기 있는 5개 AI 모델 생성 코드의 48%가 취약점 포함
- 상업용 모델: 생성 코드의 5% 이상이 존재하지 않는 패키지 사용
- 오픈소스 모델: 생성 코드의 22%가 존재하지 않는 패키지 사용



파급 효과

- 안전하지 않은 코드 배포로 인한 잠재적 보안 위협
- 보안 취약점 노출로 인한 피해 가능성
- 개발자의 추가 보안 검토 시간 소요

기존 연구 및 한계

연구의 차별점

기존 연구

Vul-RAG: CVE 데이터로 생성된 코드의 취약점 탐지
SOSSecure: StackOverflow 데이터 기반 보안성 평가
Codexity: 정적 분석 도구로 보안 취약점 수정

한계점

코드 생성 이후 취약점 탐지 (사후 대응적)
코드 수정 부담 발생
개발 프로세스 지연



본 연구

코드 생성 전 보안 템플릿 제공
(선제적 예방)
취약점 발생 자체를 원천 차단

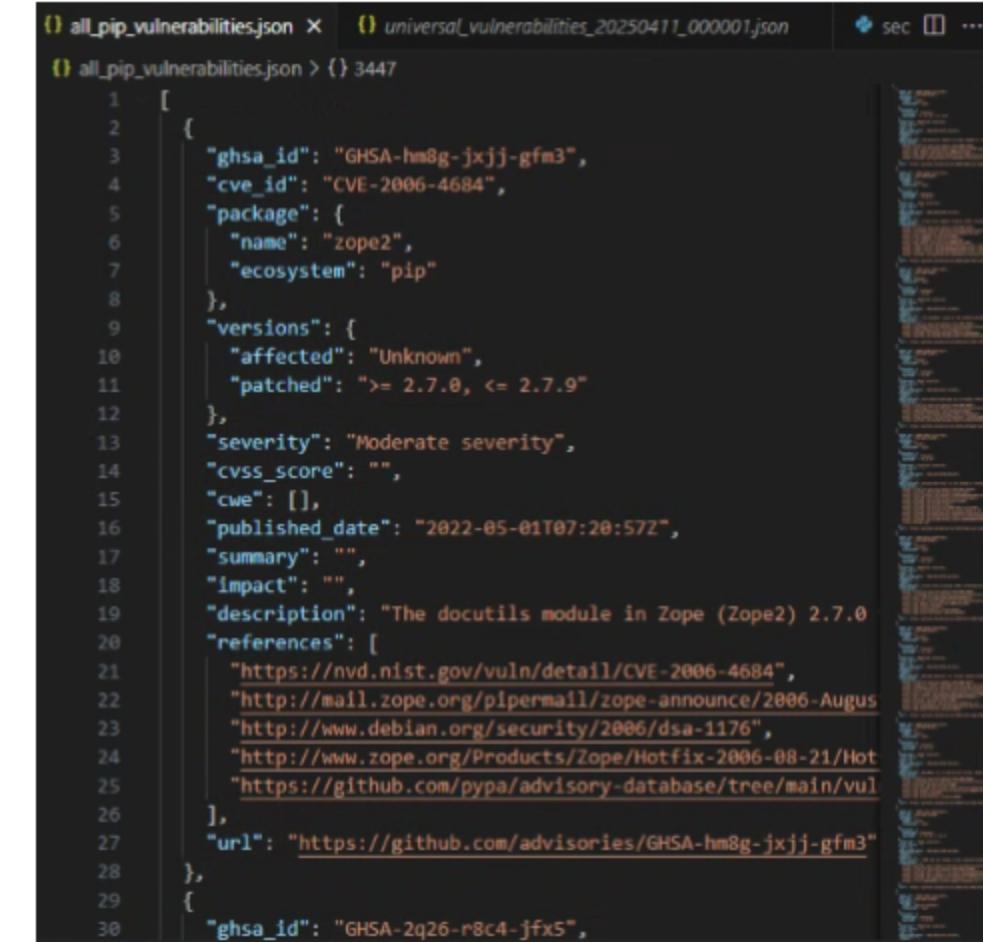
CVE-RAG 프레임워크 개요

핵심 아이디어

- CVE 데이터베이스 + RAG 기술 결합
- 코드 생성 전 보안 템플릿 자동 생성
- AI 코드 생성기가 안전한 의존성 사용하도록 유도

주요 특징

- 맞춤형 보안 가이드라인 제공
- 단순 "최신 버전" 추천이 아닌 구체적 버전 제약 제시
- 취약한 코드 패턴 자동 탐지 및 경고



```
1 [  
2 {  
3   "ghsa_id": "GHSA-hm8g-jxjj-gfm3",  
4   "cve_id": "CVE-2006-4684",  
5   "package": {  
6     "name": "zope2",  
7     "ecosystem": "pip"  
8   },  
9   "versions": {  
10    "affected": "Unknown",  
11    "patched": ">= 2.7.0, <= 2.7.9"  
12  },  
13  "severity": "Moderate severity",  
14  "cvss_score": "",  
15  "cwe": [],  
16  "published_date": "2022-05-01T07:20:57Z",  
17  "summary": "",  
18  "impact": "",  
19  "description": "The docutils module in Zope (Zope2) 2.7.0  
20  "references": [  
21    "https://nvd.nist.gov/vuln/detail/CVE-2006-4684",  
22    "http://mail.zope.org/pipermail/zope-announce/2006-Augus",  
23    "http://www.debian.org/security/2006/dsa-1176",  
24    "http://www.zope.org/Products/Zope/Hotfix-2006-08-21/Hot",  
25    "https://github.com/pypa/advisory-database/tree/main/vul",  
26  ],  
27  "url": "https://github.com/advisories/GHSA-hm8g-jxjj-gfm3"  
28 },  
29 {  
30   "ghsa_id": "GHSA-2q26-r8c4-jfx5",  
31 }]
```

[GitHub Advisory 크롤링을 통한 벡터화 대상 취약점 DB파일]



[프레임워크 개념도]

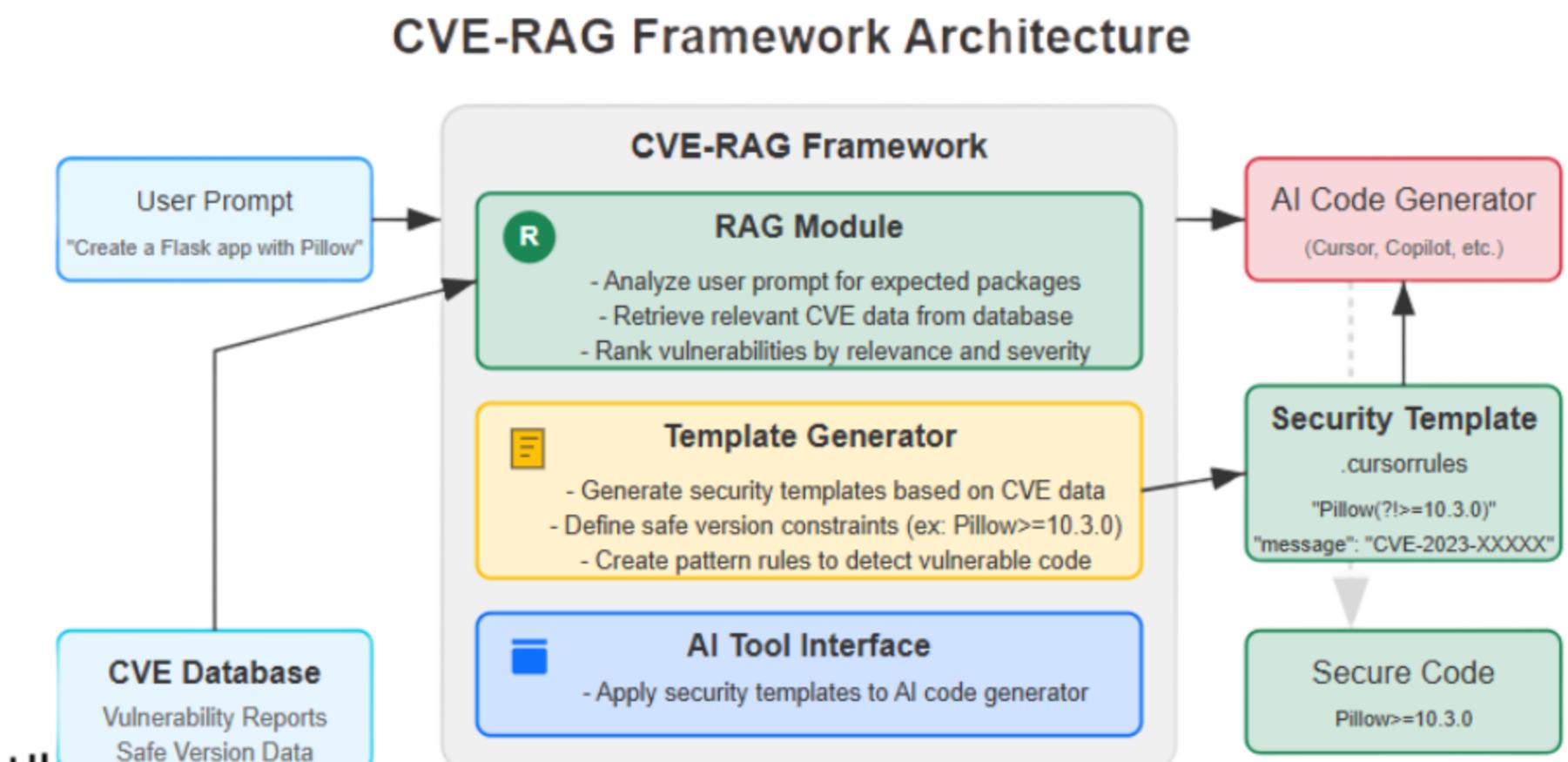
시스템 아키텍처

전체 구성

- RAG 모듈
- 템플릿 생성기
- AI 도구 인터페이스

데이터 흐름

- 사용자 입력 → 프롬프트 분석
- 예상 패키지 식별 → CVE 데이터베이스 검색
- 취약점 정보 추출 → 보안 템플릿 생성
- 템플릿 적용 → 안전한 코드 생성



RAG 모듈 상세

기능

- 사용자 프롬프트 분석
- 관련 취약점 정보 검색
- TF-IDF/코사인 유사도 기반 검색

구현 요소

- 벡터화: TF-IDF 벡터라이저
- 유사도 계산: 코사인 유사도
- 관련성 순위화: 상위 K개 결과 선별

```
● ● ●  
def find_relevant_vulnerabilities(self, query: str, top_k: int = 10):  
    # 쿼리 벡터화  
    query_vector = self.vectorizer.transform([query])  
    # 유사도 계산  
    similarities = cosine_similarity(query_vector, self.vectors).flatten()  
    # 상위 인덱스 추출  
    top_indices = similarities.argsort()[-top_k:][::-1]
```

템플릿 생성기 상세

```
{ } .cursorrules > ...  
1  [  
2    {  
3      "version": 1,  
4      "rules": [  
5        {  
6          "name": "Use safe version of Django",  
7          "pattern": "Django(?!=>-5.1.5)",  
8          "message": "Security: Use Django>=5.1.5 to avoid known vulnerabilities",  
9          "priority": "high"  
10        },  
11        {  
12          "name": "Use safe version of Pillow",  
13          "pattern": "Pillow(?!=>=10.3.0)",  
14          "message": "Security: Use Pillow>=10.3.0 to avoid known vulnerabilities",  
15          "priority": "high"  
16        }  
17      ]  
]
```

```
Security Advisory: Enhanced Code Security Guidelines  
Package Version Requirements  
known security vulnerabilities, use these version constraints:  
  - ``**: '>=2.3.2'  
  - ``**': '>=10.3.0'  
Inheritable Code Patterns to Avoid  
Security Recommendations  
  - validate and sanitize user input before using it in any operation.  
  - use parameterized queries or ORMs to prevent SQL injection vulnerabilities if you're using a database.  
  - implement proper authentication and authorization mechanisms to restrict access to sensitive resources.  
  - keep your dependencies up to date to patch known vulnerabilities. Use 'pip install --upgrade' regularly.  
  - use static analysis tool like Bandit to identify potential security flaws in your code.  
  - implement logging and monitoring to detect and respond to suspicious activity.  
  - enable Content Security Policy (CSP) to prevent cross-site scripting (XSS) attacks.  
  - configure your web server (e.g., Nginx, Apache) with security best practices, such as disabling directory listing and setting proper HTTP headers.  
  - manually review your code for security vulnerabilities.  
  - implement file type validation by checking the file's magic number instead of relying solely on the file extension.  
  - limit file upload sizes to prevent denial-of-service attacks.  
  - do not upload files outside the web server's document root to prevent direct access.  
  - use a web application firewall (WAF) to protect against common web attacks.
```

기능

템플릿 구성 요소

- 취약점 정보 분석
- .cursorrules 형식 템플릿 생성
- 안전한 버전 및 패턴 규칙 생성
- 안전한 라이브러리 버전 규칙
- 취약한 코드 패턴 탐지 규칙

템플릿 생성기 상세

기능

- 취약점 정보 분석
- .cursorrules 형식 템플릿 생성
- 안전한 버전 및 패턴 규칙 생성

```
{ } .cursorrules > ...  
1 [ "version": 1,  
2   "rules": [  
3     {  
4       "name": "Use safe version of Django",  
5       "pattern": "Django(?!=>=5.1.5)",  
6       "message": "Security: Use Django>=5.1.5 to avoid known vulnerabilities",  
7       "priority": "high"  
8     },  
9     {  
10       "name": "Use safe version of Pillow",  
11       "pattern": "Pillow(?!=>=10.3.0)",  
12       "message": "Security: Use Pillow>=10.3.0 to avoid known vulnerabilities",  
13       "priority": "high"  
14     }  
15   ]  
16 ]  
17 ]
```

[템플릿 생성기로 생성된 .cursorrules]

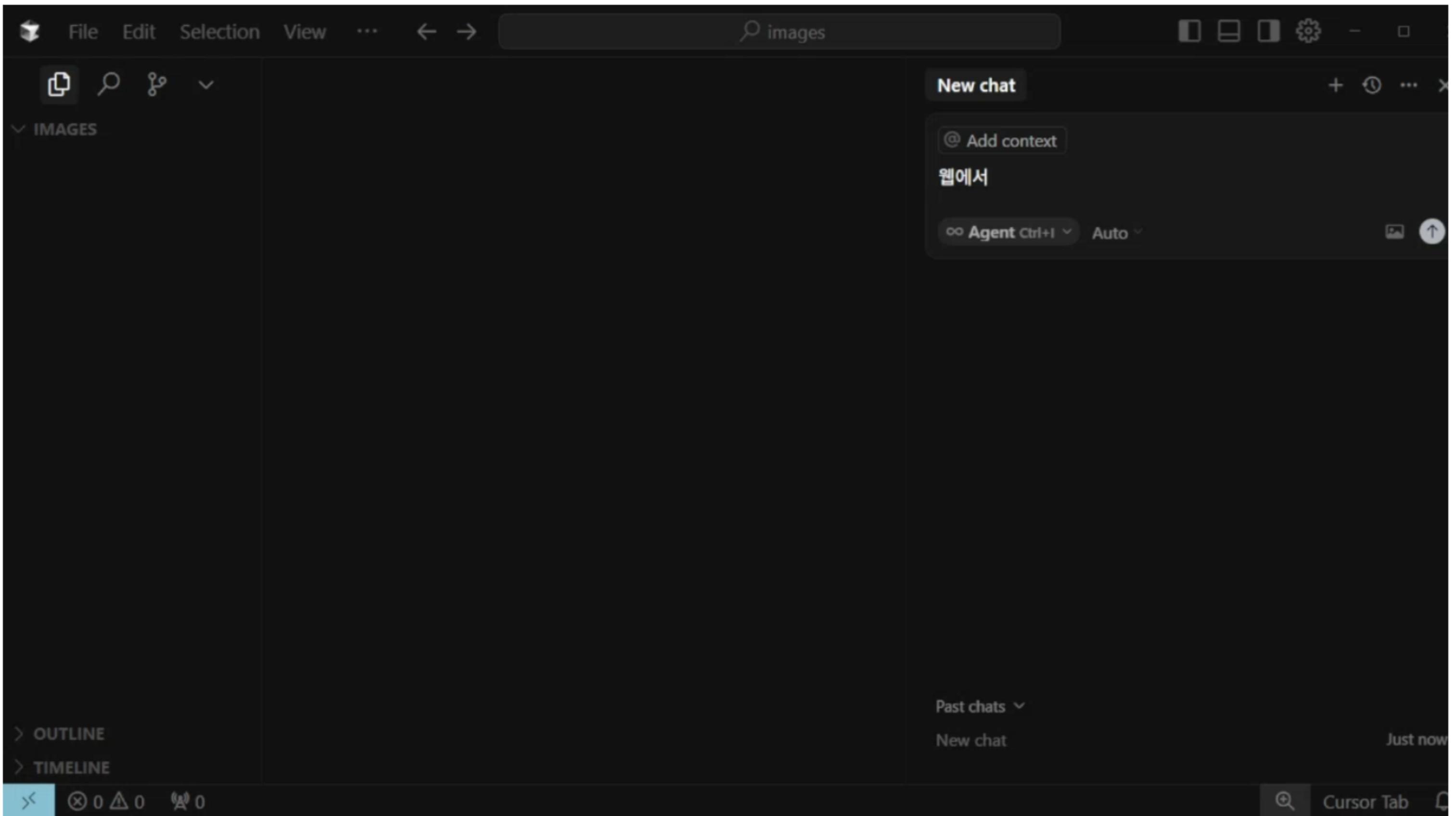
템플릿 구성 요소

- 안전한 라이브러리 버전 규칙
- 취약한 코드 패턴 탐지 규칙

```
# Security Advisory: Enhanced Code Security Guidelines  
## 🔍 Package Version Requirements  
Based on known security vulnerabilities, use these version constraints:  
- **Flask**: '>=2.3.2'  
- **Pillow**: '>=10.3.0'  
## 🚫 Vulnerable Code Patterns to Avoid  
## 🔒 Security Recommendations  
1. Always validate and sanitize user input before using it in any operation.  
2. Use parameterized queries or ORMs to prevent SQL injection vulnerabilities if you're using a database.  
3. Implement proper authentication and authorization mechanisms to restrict access to sensitive resources.  
4. Keep your dependencies up to date to patch known vulnerabilities. Use 'pip install --upgrade' regularly.  
5. Use a static analysis tool like Bandit to identify potential security flaws in your code.  
6. Implement logging and monitoring to detect and respond to suspicious activity.  
7. Use a Content Security Policy (CSP) to prevent cross-site scripting (XSS) attacks.  
8. Configure your web server (e.g., Nginx, Apache) with security best practices, such as disabling directory listing and setting proper HTTP headers.  
9. Regularly review your code for security vulnerabilities.  
10. Implement file type validation by checking the file's magic number instead of relying solely on the file extension.  
11. Limit file upload sizes to prevent denial-of-service attacks.  
12. Store uploaded files outside the web server's document root to prevent direct access.  
13. Use a web application firewall (WAF) to protect against common web attacks.
```

[템플릿 생성기로 생성된 코드 및 라이브러리 규칙]

시스템 시연



실험 설계

실험 목표

- CVE-RAG 템플릿 적용 유무에 따른 보안성 비교

실험 환경

- Cursor AI 도구 사용
- A그룹: CVE-RAG 템플릿 적용
- B그룹: 템플릿 미적용

실험 시나리오

- 이미지 처리: Flask + Pillow를 사용한 이미지 업로드 및 리사이징 웹 애플리케이션
- API 통신: Requests를 사용한 외부 API 데이터 가져오기 및 CSV 작성
- 데이터베이스 연동: Django를 사용한 사용자 조회 관리 명령

측정 지표

- pip-audit: 의존성 취약점 분석
- Bandit: 코드 패턴 취약점 분석

실험 결과 - 취약점 수 비교

의존성 취약점 분석 결과

- CVE-RAG 템플릿 적용 유무에 따른 보안성 비교

주요 발견점

- CVE-RAG 템플릿 적용 시 의존성 취약점 100% 감소
 - (평균 7개 → 0개)
- 특히 이미지 처리 프롬프트에서 가장 많은 취약점(10개) 발견
- 주요 취약 라이브러리
 - Pillow 10.0.0: 이미지 처리 관련 취약점
 - Werkzeug 2.3.7: Flask 관련 취약점
 - Requests 2.31.0: API 통신 취약점
 - Cryptography 41.0.7: 암호화 관련 취약점

프롬프트 시나리오	템플릿 미적용(B)	템플릿 적용(A)	주요 취약 라이브러리 버전
이미지 처리	10개	0개	Pillow 10.0.0, Werkzeug 2.3.7
API 통신	7개	0개	Requests 2.31.0
데이터베이스	4개	0개	Cryptography 41.0.7
평균	7개	0개	

템플릿 분석

생성된 규칙 분석

- 총 규칙 수: 평균 12개/템플릿
- 버전 제약 규칙: 평균 5개/템플릿
- 코드 패턴 규칙: 평균 7개/템플릿

규칙 예시 (Pillow 관련)



```
{  
  "name": "Use safe version of Pillow",  
  "pattern": "Pillow(?!=>=10.3.0)",  
  "message": "Security: Use Pillow>=10.3.0 to avoid CVE-2024-28219 (buffer overflow exists)",  
  "priority": "high"  
}
```

템플릿 생성 중 CVE 여러 개 존재 시, Known Vulnerability라고 언급하며, 모든 CVE 취약점에서 안전
할 수 있는 버전을 추천

결론

주요 성과

- AI 코드 생성의 의존성 취약점 100% 감소
- 안전한 라이브러리 버전 사용률 100% 달성
- 선제적 보안 접근법의 효과성 입증

차별화 가치

- "최신 버전 사용" 대비 구체적 취약점 기반 가이드
- 코드 생성 전 보안 정보 활용한 선제적 접근
- 맞춤형 보안 템플릿 자동 생성

확장 방향

- 다양한 AI 코드 생성 도구 지원 (GitHub Copilot, Tabnine 등)
- 범용 템플릿 형식 개발
- 코드 패턴 취약점 감지 능력 향상

기술적 개선

- 맥락 인식 취약점 탐지 개선
- 자동 취약점 패치 제안 메커니즘

감사합니다

참고문헌

- X. Du, G. Zheng, K. Wang et al., "Vul-RAG: Enhancing LLM-based Vulnerability Detection via Knowledge-level RAG," arXiv, 2024.
- M. Mukherjee and V. J. Hellendoorn, "SOSecure: Safer Code Generation with RAG and StackOverflow Discussions," arXiv, 2025.
- S. Y. Kim, Z. Fan, Y. Noller, and A. Roychoudhury, "Codexity: Secure AI-assisted Code Generation," arXiv, 2024.