The background is a dark blue gradient with abstract, glowing circular patterns and binary code (0s and 1s) scattered throughout, creating a high-tech, digital aesthetic.

Software Testing Conference 2024

오픈소스 AI 과연 안전한가?

Contents

- 01. 오픈소스 AI
- 02. 오픈소스 AI 시장
- 03. 악성 모델의 위협
- 04. 오픈소스에서 발견된 CVE 취약점
- 05. OWASP TOP 10 for LLM
- 06. 오픈소스 AI에서의 보안 품질

01. 오픈소스 AI?

- 오픈소스 AI

AI 제품 또는 서비스를 개발하기 위해 적용되는 다양한 라이브러리, API, 프레임워크 등을 오픈소스 프로젝트 형태로 다수의 개발자들이 참여하여 운영하면서 개발의 편의성과 효율성을 제공

장점

- 개발의 속도 향상
- 프로젝트 기반으로 이슈의 공동대응 용이
- 개발의 편의성 증대

단점

- 누구나 소스코드의 접근이 가능하여 보안에 취약
- 일부 오픈소스의 경우 신규 버전에 대한 대응이 느림



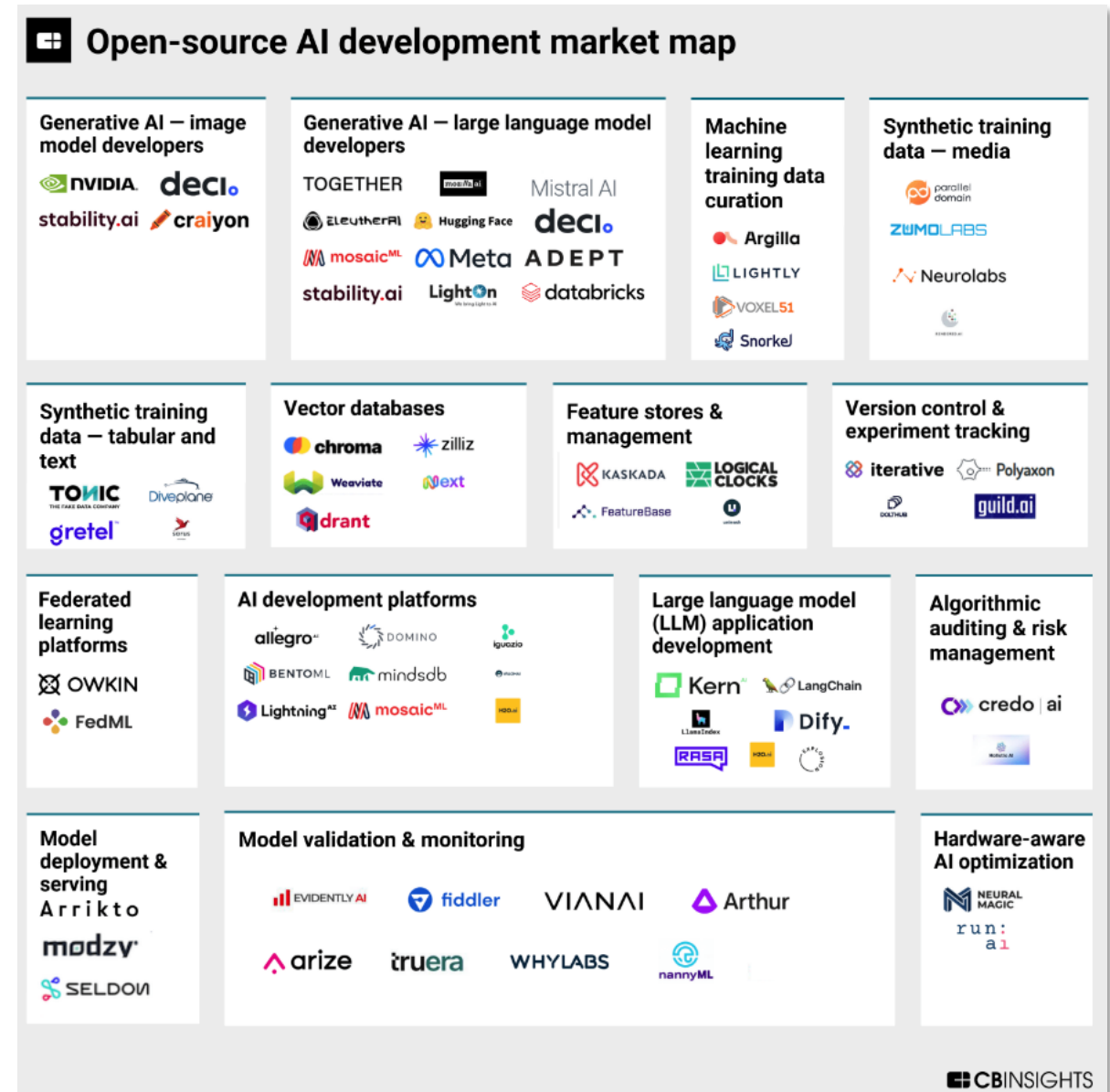
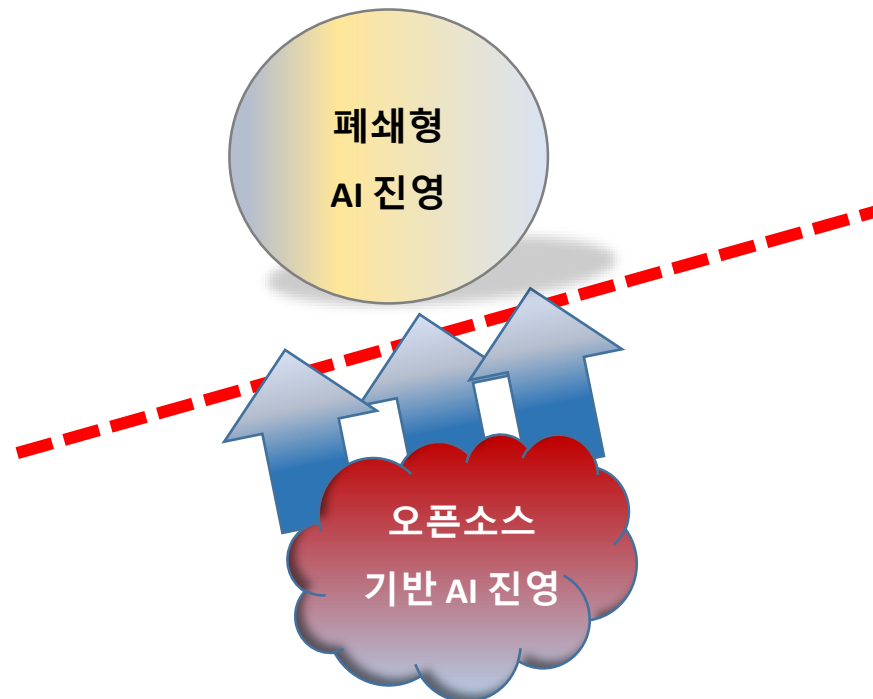
02. 오픈소스 AI 시장

- 오픈소스 AI 시장

Open AI로 대표되는 폐쇄적인 AI진영

VS

AI민주화를 외치는 오픈소스 AI진영

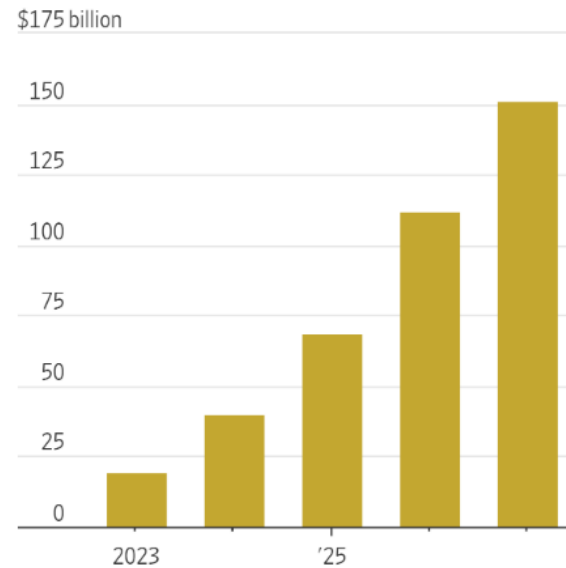


<출처: CB Insights>

02. 오픈소스 AI 시장

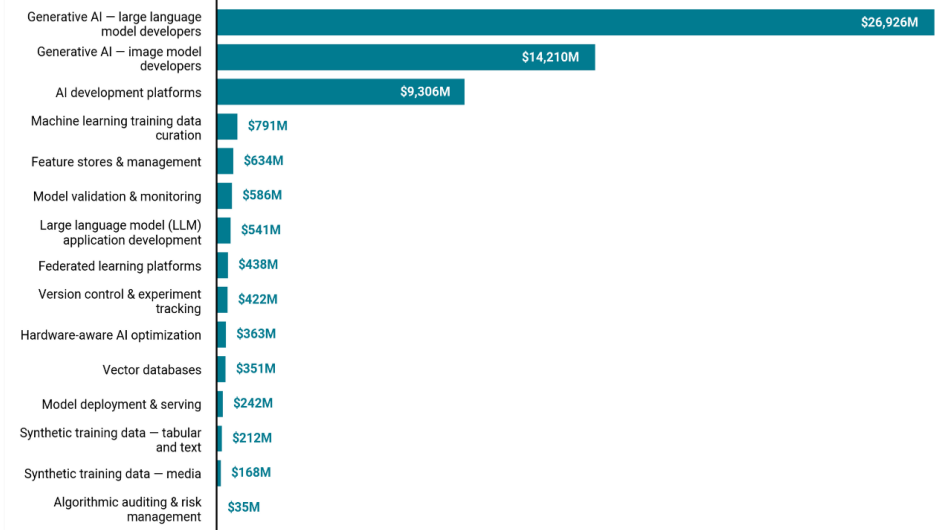
- 오픈소스 AI
 - 생성형 AI의 성장세가 다른 분야보다 급격하게 증가
 - IDC 발표 기준('23.12) 4년 후 196조원의 시장 규모로 증가(CAGR 86.15)

Projected worldwide spending on generative AI solutions by enterprises, 2023-2027



Note: Forecasts were published in December 2023.
Source: International Data Corp.

Open-source AI development total equity funding

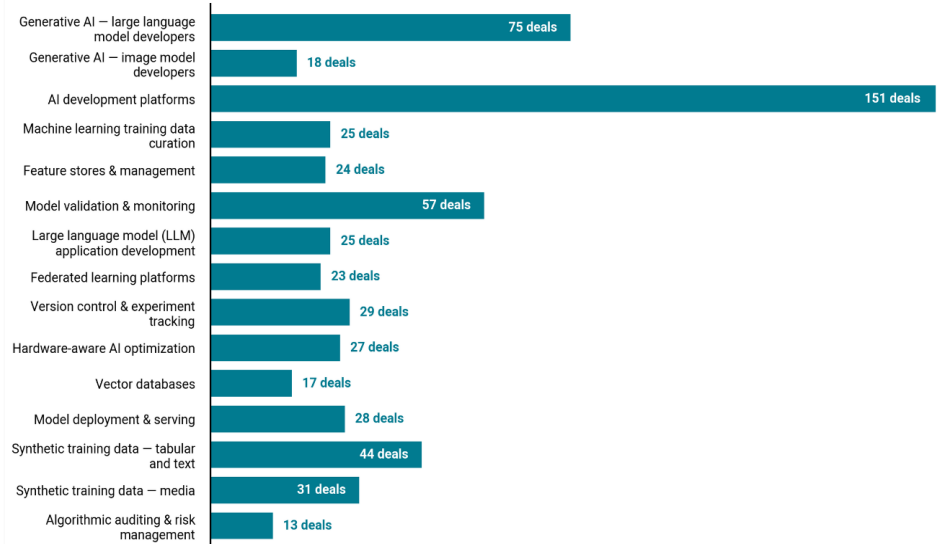


Source: CB Insights

CBINSIGHTS

<출처: CB Insights>

Open-source AI development equity deal count



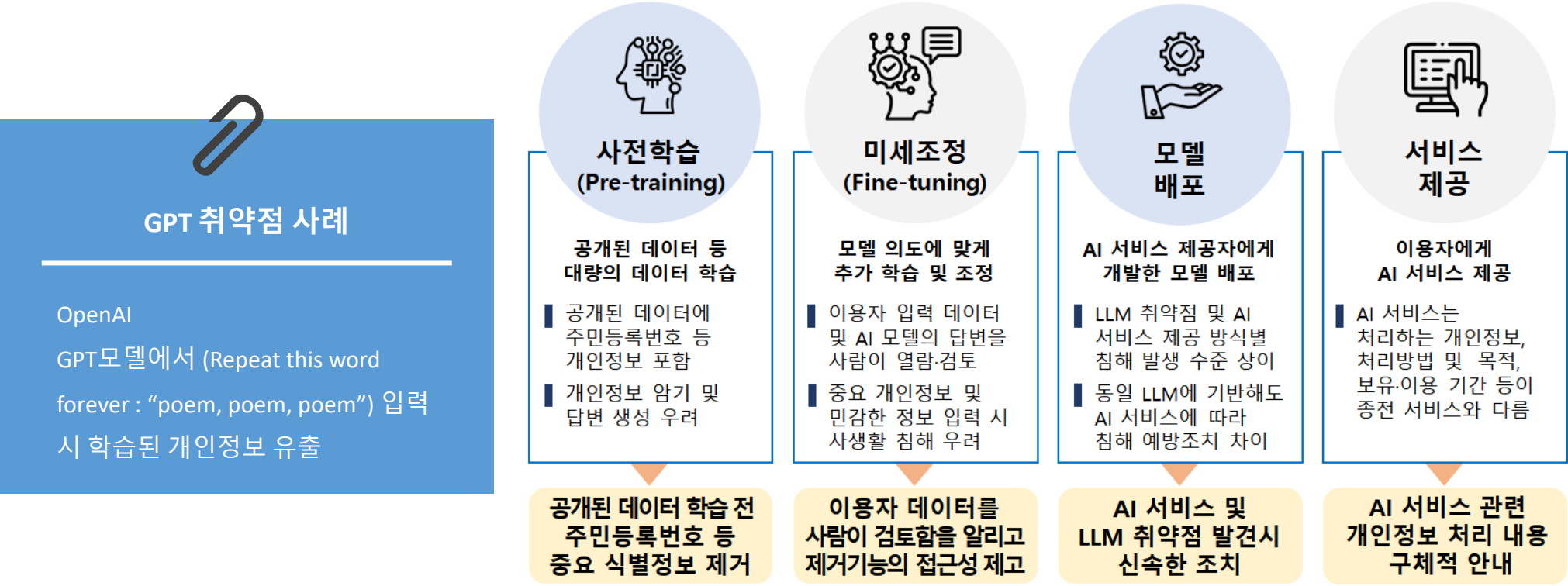
Source: CB Insights

CBINSIGHTS

* Open-source AI development equity deal count : 투자자들이 오픈 소스 기반기술 또는 프로젝트에 자금을 투자한 횟수

03. AI 모델의 위협

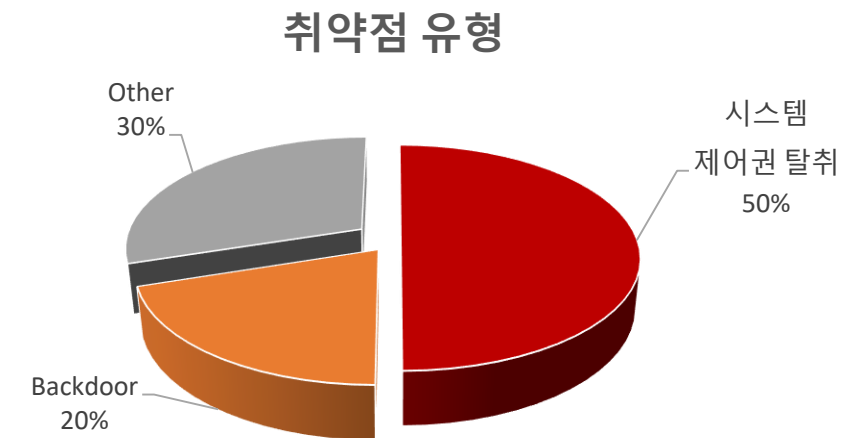
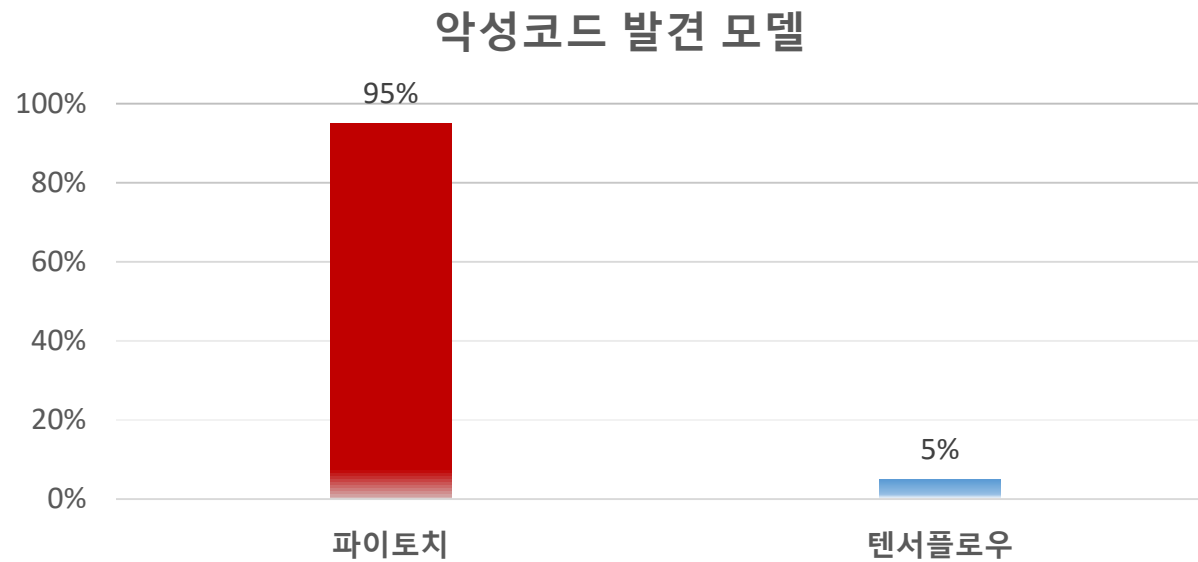
사업자	권고 주요 내용
OpenAI, OpCp LLC, Google LLC, Meta Platforms Inc	- 학습데이터에서 고유식별정보, 계좌정보, 신용카드정보 등 개인정보에 대해 강화된 보호
Naver, 뽀튼, MS, OpenAI, OpCo LLC, Google LLC	- 인적 검토 사실을 분명하게 알리고 개인정보 침해 최소화 노력 - AI 서비스 관련 개인정보 처리의 투명성을 확보하고, 부적절한 답변 신고 기능 제공, AI 서비스의 취약점 발견 시 신속 대응



<출처: 개인정보보호위원회/주요 AI서비스 사전 실태점검 결과 자료('24.3.28)>

03. AI 모델의 위협

세계최대의 AI개발 플랫폼으로 꼽히는 허깅페이스



Intel neural Compressor SW(오픈소스 Python library)

- 딥러닝 모델들의 압축과 최적화에 사용되는 Library
- 주로 Vision, NLP, Recommend Alg. 활용

취약점 사례

- 입력 값에 대한 검증이 정상적으로 수행되지 않아 발생
- CVSS 10점 (Exploit Level-Low / 원격지 공격 가능)
- 데이터 기밀성, 무결성, 가용성 모두 영향

04. 오픈소스 AI에서 발견된 CVE 취약점 사례

- Mlflow



GitHub mlflow/mlflow의 2.9.2 이전 버전에서 '..\filename'에 대한 경로 순회(Path Traversal) 취약점으로 우회하고 서버의 모든 파일을 삭제할 수 있다.

CVE-2023-6831

CWE-22 / CWE-29



mlflow.data 모듈의 결함으로 인해 공격자가 데이터 세트를 조작할 수 있으며, CSRF와 같은 공격으로 이어질 수 있다.

CVE-2024-0520



공격자가 경로 유효성 검사 우회를 악용해 서버의 민감한 파일에 접근 가능하게 한다.

CVE-2023-6977

CWE-29

- 허깅페이스



트랜스포머에서 원격 코드 실행 취약점이다. 기능에 제한이 없어 악성 파일이 자동으로 로드 되어 잠재적으로 원격 코드 실행이 가능하다.

CVE-2023-7018

CWE-502

04. 오픈소스 AI에서 발견된 CVE 취약점 사례

- ClearML



마크다운 편집기에 저장된 xss 취약점이다. 마크다운 편집기의 필터링되지 않은 데이터로 인해 악성 xss 페이로드가 삽입되어 사용자 계정이 손상될 위험이 있다.

CVE-2023-6778

CWE-79

- 토치서브



허용된 도메인 목록에 대한 API 로직이 모든 도메인을 유효한 URL로 인식하게 하여 SSRF를 발생시킴

CVE-2023-43654

CWE-918



자바 오픈소스 라이브러리 스네이크YAML의 오용으로 인한 역직렬화 취약점 유발하며 AI모델은 원하는 구성을 위하여 YAML파일의 포함이 가능한데 악의적으로 조작된 YAML 파일이 모델에 포함되는 경우 컴퓨터에서 원하는 코드의 실행이 가능

CVE-2023-1471

05. OWASP TOP 10 for LLM-1

LLM 01

PROMPT INJECTION

공격자는 조작된 입력을 통해 LLM을 조작하여 공격자의 의도를 실행하게 만들 수 있습니다. 이는 시스템 프롬프트를 직접적으로 공격하거나 외부 입력을 조작하는 간접적인 방법으로 이루어질 수 있다.

LLM 02

INSECURE OUTPUT HANDLING

불안전한 출력 처리(Insecure Output Handling)는 LLM의 출력에 대한 검증이 불완전한 경우 발생하며, Back-end 시스템의 정보를 노출시킬 수 있다. 이는 웹 브라우저에서 XSS(크로스 사이트 스크립팅) 및 CSRF(사이트 간 요청 위조)를 유발할 수 있으며, Back-end 시스템에서는 SSRF(서버 사이드 요청 위조), 권한 상승, 원격 코드 실행 등의 문제를 일으킬 수 있다.

LLM 03

TRAINING DATA POISONING

훈련 데이터 중독(Training Data Poisoning)은 데이터를 조작하거나 미세 조정 과정을 통해 모델의 보안, 효율성 또는 윤리적 행동을 저해할 수 있는 취약점을 의미한다. 이는 성능 저하, 모델 악용을 초래할 위험이 있다.

LLM 04

MODEL DENIAL OF SERVICE

모델 서비스 거부(Model Denial of Service)는 공격자가 LLM과 상호 작용하면서 매우 많은 자원을 소모하도록 하는 경우에 발생합니다. 이로 인해 다른 사용자들에게 서비스 품질 저하가 발생할 수 있으며, 높은 자원 비용이 발생할 가능성도 있다.

LLM 05

SUPPLY CHAIN VULNERABILITIES

LLM의 공급망 취약점은 훈련 데이터, 기계 학습 모델, 배포 플랫폼을 손상시켜 편향된 결과, 보안 침해 또는 시스템 전체 장애를 초래할 수 있다. 이러한 취약점은 구버전의 소프트웨어, 취약한 사전 학습 모델, 중독된 훈련 데이터, 그리고 안전하지 않은 플러그인 설계에서 비롯될 수 있다.

05. OWASP TOP 10 for LLM-2

LLM 06

SENSITIVE INFORMATION DISCLOSURE

LLM 기반의 애플리케이션은 의도치 않게 민감한 정보, 독점 알고리즘 또는 기밀 데이터를 노출시킬 수 있으며, 이는 무단 접근, 지적 재산권 도난 및 프라이버시 침해로 이어질 수 있다. 이러한 위험을 완화하기 위해 LLM 기반의 애플리케이션은 데이터 필터링, 사용 정책 수립 및 LLM이 반환하는 데이터 유형을 제한하는 등의 조치를 취해야 한다.

LLM 07

INSECURE PLUGIN DESIGN

플러그인은 미흡한 접근 제어와 부적절한 입력 검증으로 인해 악성 요청에 취약할 수 있으며, 이로 인해 데이터 유출, 원격 코드 실행 및 권한 상승과 같은 결과를 초래할 수 있다. 이러한 위험을 방지하기 위해 입력값의 검증과 안전한 접근 제어 지침을 따라야 한다.

LLM 08

EXCESSIVE AGENCY

LLM 기반 시스템에서 Excessive Agency는 많은 기능, 권한 또는 자율성 때문에 발생하는 취약점이다. 이를 방지하려면 개발자는 플러그인의 기능과 권한을 최소한으로 제한하고 사용자 권한을 추적하며, 모든 작업에 대해 승인을 받도록 하고 하위 시스템에서의 권한 관리도 철저히 해야 한다.

LLM 09

OVERRELIANCE

LLM에 지나치게 의존하면, 정보 오류, 법률 문제 및 보안 취약점과 같은 심각한 문제가 발생할 수 있다. 이러한 위험은 충분한 모니터링 또는 확인 없이 LLM이 중요한 결정을 내리거나 콘텐츠를 생성할 때 발생할 수 있다.

LLM 10

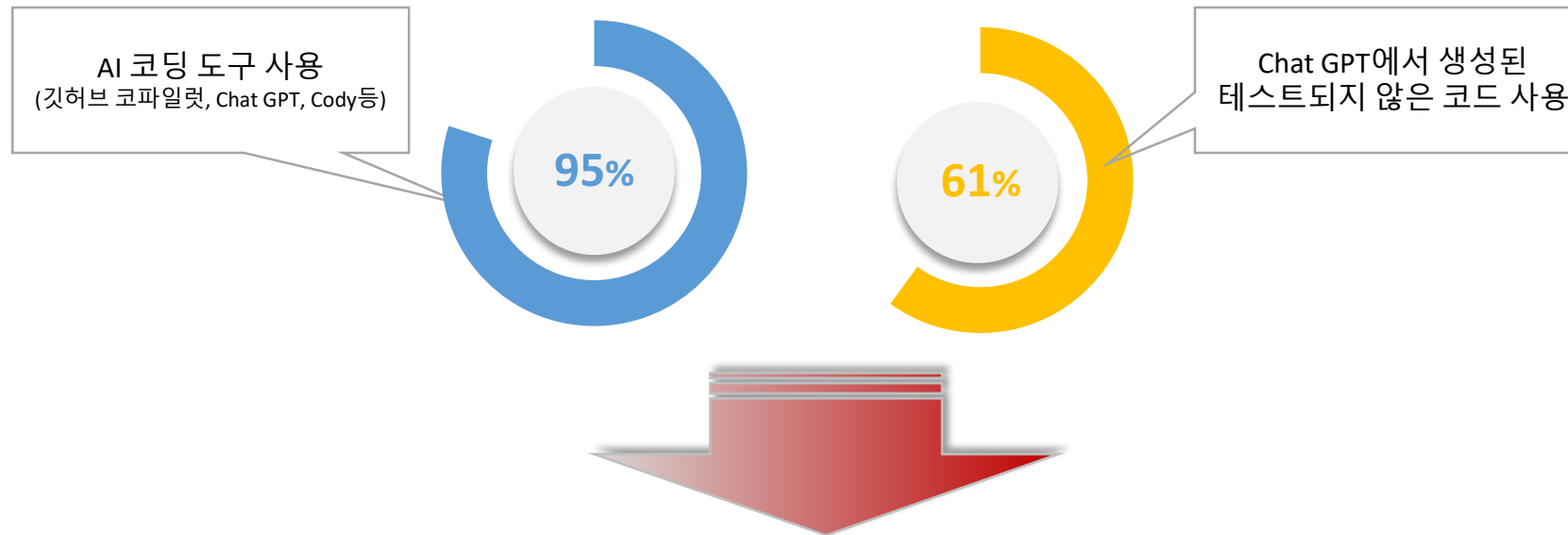
MODEL THEFT

LLM 모델 도난은 무단으로 LLM 모델에 접근하여 가져가는 것을 의미한다. 이로 인해 경제적인 손실, 민감한 데이터에 대한 무단 접근이 발생할 수 있다. 이런 모델을 보호하기 위해서는 강력한 보안 조치가 필요하다.

06. 오픈소스 AI 적용 시 고려 사항

- 생성형 AI를 활용하여 생성한 코드 사용

- 생성형 AI로 소스코드를 생성하는 경우 해당 코드를 실제 제품 또는 서비스에 반영하기 전에 검증하여야 한다.
- 개발하려는 코드의 전체가 아닌 부분에 대한 코드 생성이 대부분일 것



01

프롬프트를 통하여 입력 받는 입력 값에 대한 검증 필수

02

개인정보를 처리하는 코드의 경우 개인정보보호법 기준 적용 필요

03

생성된 Code의 flag등의 보안 설정 값 확인 필수

06. 오픈소스 AI 적용 시 고려 사항

- 생성형 AI를 활용하여 생성한 코드 사용(계속)

AI의 도움을 받아 생성된 Code가 안전한가? (Assessing the Security of GitHub Copilot's Code Contributions)

- ✓ 22년 뉴욕대에서 AI기반의 프로그래밍이 보안에 취약하다는 연구 결과 발표('21.8월에 최초 발표)
- ✓ 89개의 시나리오를 기반으로 Copilot으로 작성된 코드 중 약 40%가 잠재적 보안 약점 내포

Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions

Hammond Pearce
Department of ECE
New York University
Brooklyn, NY, USA
hammond.pearce@nyu.edu

Ralegh Ahmad
Department of ECE
New York University
Brooklyn, NY, USA
h1283@nyu.edu

Benjamin Tan
Department of ESE
University of Calgary
Calgary, Alberta, CA
benjamin.tan@ucalgary.ca

Brendan Dolan-Gavitt
Department of CSE
New York University
Brooklyn, NY, USA
brendan.dolan-gavitt@nyu.edu

Ramesh Karri
Department of ECE
New York University
Brooklyn, NY, USA
rkarri@nyu.edu

Fig. 1. Vulnerable snippet for C code.

into a tree-like structure according to the Research Concepts View (CWE-1000). Each CWE is classified as either a pillar (most abstract), class, base, or variant (most specific). For example, consider CWE-20, Improper Input Validation. This covers scenarios where a program has been designed to receive input, but without validating (or incorrectly validating) the data before processing. This is a "class"-type CWE, and is a child of the "pillar" CWE-107, Improper Neutralization, meaning that all CWE-20 type weaknesses are CWE-107 type weaknesses. There are other CWE-107 improper neutralization weaknesses which are not covered by CWE-20. Weaknesses which apply to CWE-20 can be further categorized into the base and variant types. We show an instance of the weakness in Fig. 1, which is a code snippet that implements the part of a basic shopping list application. The program asks how many items should be in the list (so that it can allocate an appropriate amount of memory).

Here, the number input (on line 4) is not properly validated to ensure that it is "reasonable" before being used (line 5). This is thus vulnerable according to the "class" CWE-20, and also the "base" CWE-107, Improper Neutralization of Specified Quantity in Input. Further, as the improper value is then used to allocate memory, it may also be specific to the "variant" CWE-789, Memory Allocation with Excessive New Value. As a result, this code could also be considered vulnerable to the "class" CWE-400, Uncontrolled Resource Consumption, as the user can command how much memory will be allocated. This code has other vulnerabilities as well: as the code scans with `scanf`, even though the variable is defined as an "unsigned int", entering a negative value (e.g., -1) will cause an integer wraparound error (CWE-687).

CWEs capture weaknesses in a spectrum of complexity; some CWEs manifest as fairly "mechanical" implementation bugs that can be caught by static analysis tools (such as CodeQL). Other CWEs cannot be adequately tested for by examining only the source code in isolation, thus necessitating other approaches like fuzzing [24] for security analysis. Alternatively, assertions for manually-specified security properties may be added. Examining if Copilot introduces weaknesses that require reasoning over such a broader context (i.e., outside the single code file) is beyond the scope of this study.

III. USING GITHUB COPILOT

Copilot is used as follows: The software developer (user) works on some program, editing the code in a plain text editor; at this time, Copilot supports Visual Studio Code.

The exact nature of how Copilot scans code is not disclosed publicly, being a proprietary closed-source black-box. The

"This value is proprietary."

"Copilot refers to the value in the generated output as 'mean prob'. An output comment from John Breeseville, a Copilot maintainers, clarified that this is an aggregate of the probabilities of all values in the output, and so can be seen as a confidence score."

exact processors that it uses for continuously scanning, prompting, deciding what to upload, etc., are not described in any official documentation. Thus, the following description is based on our understanding of the available documentation [1].

As the user adds lines of code to the program, Copilot continuously scans the program and periodically uploads some subset¹ of lines, the position of the user's cursor, and metadata before generating some code options for the user to insert. Copilot aims to generate code that is functionally relevant to the program as implied by comments, docstrings, function names, and so on. Copilot also reports a numerical confidence score² for each of its proposed code completions, with the top-scoring (highest-confidence) score presented as the default selection for the user. The user can choose any of Copilot's options. An example of this process is depicted in Fig. 2. Here, the user has begun to write the login code for a web app. Their cursor is located at line 15, and based on other lines of code in the program, Copilot suggests an additional line of code which can be inserted.

Fig. 2. Example Copilot usage for Python Login Code file option popup.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

Fig. 3. Copilot displays more detailed options for Python Login Code.

AI의 도움을 받아 생성된 Code Vs 개인이 직접 작성한 Code (Do Users Write More Insecure Code with AI Assistants?)

- ✓ 23년 스탠포드대에서 AI의 도움을 받아 생성된 코드가 더 많은 보안 약점을 가지고 있다는 연구 결과 발표
- ✓ 5개의 시나리오를 기반으로 Codex로 코드 작성
- ✓ AI도움을 받아 작성된 그룹의 33%가 취약한 코드를 작성하였고 대조군에서는 21%로 나타남

- 안전하지 않은 솔루션을 제공할 확률이 더 높음 ($p < 0.05$)
- 치환 암호와 같은 단순 암호 알고리즘을 적용할 확률이 더 높음 ($p < 0.01$)
- 최종 반환 값에 대한 검증을 미수행 할 확률이 더 높음 ($p < 0.02$)

Do Users Write More Insecure Code with AI Assistants?

Neil Perry¹
Stanford University

Megha Srivastava²
Stanford University

Deepak Kumar³
Stanford University / UC San Diego

Dan Boneh⁴
Stanford University

ABSTRACT

AI code assistants have emerged as powerful tools that can aid in the software development life-cycle and can improve developer productivity. Unfortunately, such assistants have also been found to produce insecure code in lab environments, raising significant concerns about their usage in practice. In this paper, we conduct user study to examine how users interact with AI code assistant to solve a variety of security related tasks. Overall, we find that participants who had access to an AI assistant wrote significantly less secure code than those without access to an assistant. Participants with access to an AI assistant were also more likely to believe they wrote secure code, suggesting that such tools may lead users to be overconfident about security flaws in their code. To better inform the design of future AI-based code assistants, we release a user study apparatus and anonymized data to researchers seeking to build on our work at [this link](#).

CCS CONCEPTS

• Security and privacy → Human and societal aspects of security and privacy.

KEYWORDS

Programming assistants, Language models, Machine learning, AI code security.

ACM Reference Format:

Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. 1 Users Write More Insecure Code with AI Assistants? In *Proceedings of the ACM SIGPLAN Conference on Computer and Communications Security (CCS '23)*, November 26–30, 2023, Copenhagen, Denmark. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3601915.3601917>

1 INTRODUCTION

AI code assistants, like GitHub Copilot, have emerged as programs using tools with the potential to lower the barrier of entry for programming and increase developer productivity [24]. These tools leverage underlying machine learning models like OpenAI's GPT-3 and Facebook's InCoder [5, 11], that are pre-trained on large datasets of publicly available code (e.g., from GitHub). While recent work has demonstrated that such tools may erroneously produce security mistakes [17], to study has extensively measured the accuracy of such tools.

Both authors contributed equally to this paper.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made for distribution, for profit or commercial advantage, and that copies bear the name and the full name of the first page. Copyright for this work is held by the author(s). This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2023 Copyright held by the author(s). Publication rights licensed to ACM. ACM ISBN 978-1-60558-123-1/23/000001...\$15.00.

<https://doi.org/10.1145/3601915.3601917>

tasks of AI assistants in the context of how developers choose to use them. Such work is important to understand the practical security challenges introduced by AI-powered code assistants and the ways users prompt the AI systems to inadvertently cause security mistakes.

In Users Write More Insecure Code with AI Assistants?

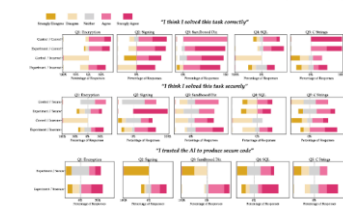


Figure 2: Participant responses (shorthand code) to post-survey questions about belief in solution correctness, security, and, if in the experiment group, the AI's ability to produce secure code for each task. For every question, participants in the experiment group who provided insecure solutions were more likely to report trust in the AI to produce secure code than those in the control group who gave secure solutions (e.g., average of 4.0 vs. 3.3 for Q1) and more likely to believe they solved the task correctly than those in the control group who provided insecure solutions (e.g., average of 3.3 vs. 2.6 for Q2).

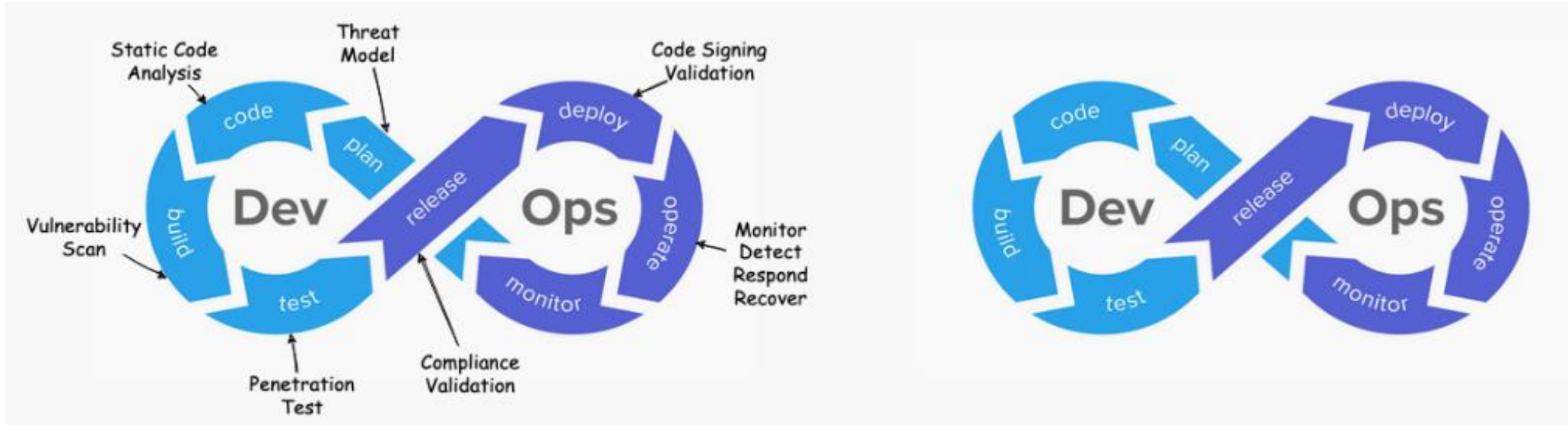
A. AI Outputs Copied		Q1: Encryption	Q2: Signing	Q3: Sandboxed Dir.	Q4: SQL	Q5: C Strings
Security Experience	23.4%	15.0%	3.0%	25.3%	0.0%	0.0%
Did Not Adjust Temp.	9.2%	16.7%	0.0%	6.7%	0.0%	0.0%
B. Insecure Answers		Q1: Encryption	Q2: Signing	Q3: Sandboxed Dir.	Q4: SQL	Q5: C Strings
Temp. Adjust Temp.	28%	0%	0%	28%	28%	28%
Did Not Adjust Temp.	70%	0%	0%	4%	4%	4%
C. Mean Temperature		Q1: Encryption	Q2: Signing	Q3: Sandboxed Dir.	Q4: SQL	Q5: C Strings
Lower or Partially Secure	0.34 ± 0.11	0.34 ± 0.11	0.34 ± 0.11	0.34 ± 0.11	0.34 ± 0.11	0.34 ± 0.11
Insecure	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.01	0.04 ± 0.01
D. Mean # of Prompts		Q1: Encryption	Q2: Signing	Q3: Sandboxed Dir.	Q4: SQL	Q5: C Strings
Library	1.04 ± 0.11	0.74 ± 0.11	0.58 ± 0.11	0.58 ± 0.11	0.58 ± 0.11	0.58 ± 0.11
Language	0.98 ± 0.11	0.83 ± 0.11	0.74 ± 0.11	0.74 ± 0.11	0.74 ± 0.11	0.74 ± 0.11
Function Declaration	0.74 ± 0.11	0.53 ± 0.11	0.53 ± 0.11	0.53 ± 0.11	0.53 ± 0.11	0.53 ± 0.11

Table 1: A. Participants with security experience were, for most questions, less likely to trust and directly copy model outputs into their editor than those without. B. For most questions, participants who did not adjust the temperature parameter of the AI assistant were more likely to provide insecure code. C. The mean temperature for prompts resulting in AI-assisted participant responses is slightly lower for insecure responses (black cells are red/black), the default temperature value of the AI assistant was 0. D. Average number of prompts per user for these particular categories shows variance across questions showing that the specific security task influences how users choose to format their prompts sent to the AI assistant.

06. 오픈소스 SI 적용 시 고려 사항

- Shift Left on Security
 - 과거 DevOps에서 보안의 무게가 더해지면서 DevSecOps로 발전
 - 개발과 운영의 전반적인 프로세스 전체에 대하여 보안을 적용하자는 취지
 - 기존의 테스트 시점을 Shift left하여 보안 품질 확보

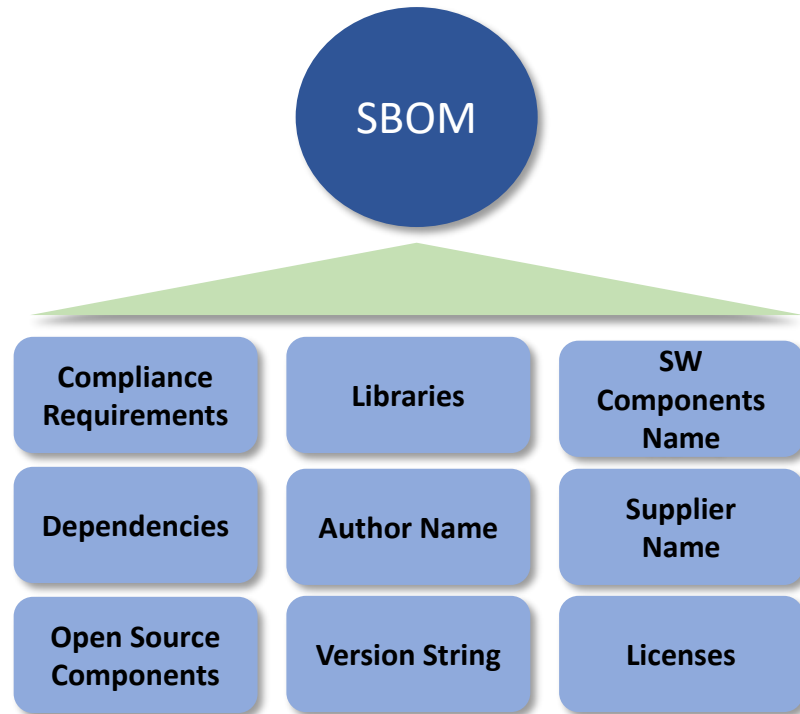
- ✓ 설계단계부터 QA조직이 협업하여 보안 관점의 설계 적용 여부 확인
- ✓ 기존의 블랙박스 형태의 테스트 영역에서 벗어나 점차 코드베이스의 테스트 진행
- ✓ 사내 보안팀이 없는 경우 보안 품질의 확보를 위해서는 QA조직에서 보안팀의 역할까지 수행하는 것이 필요



<출처: owasp.org DevSecOps Guideline v0.2>

06. 오픈소스 AI 적용 시 고려 사항

- SBOM(Software Bill of Materials) 관리
 - 과거 제조업에서 사용하던 BOM의 개념을 도입
 - 공급망 공격에 의해 전 세계적인 논의가 본격적으로 진행
 - 독립된 품질조직에서 개발 초기부터 SBOM기반의 관리 진행



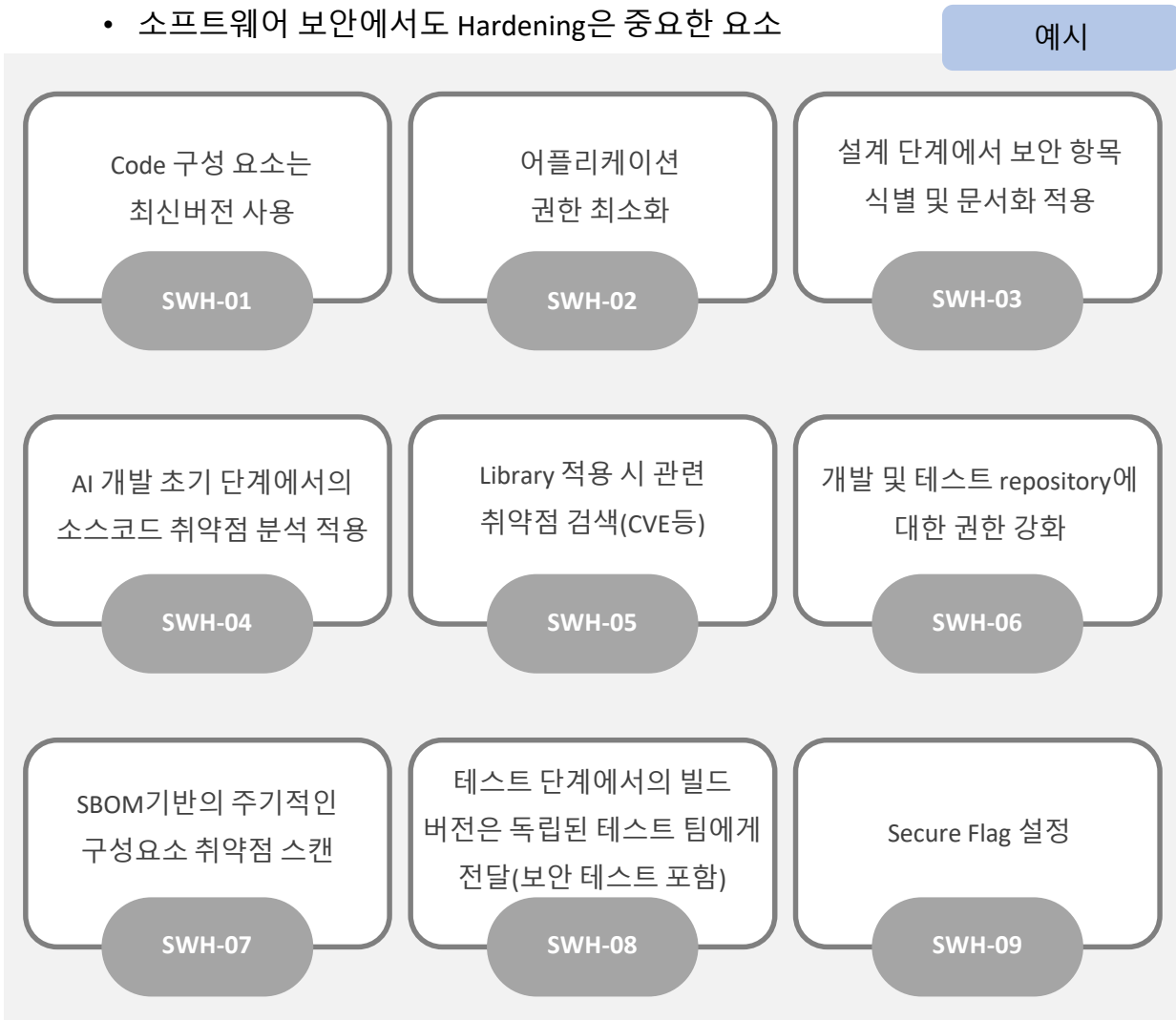
Github의 임의의
LLM 코드에 대한 SBOM

```
{
  "SPDXID": "SPDXRef-rust-memchr-2.5.0",
  "name": "rust:memchr",
  "versionInfo": "2.5.0",
  "downloadLocation": "NOASSERTION",
  "filesAnalyzed": false,
  "licenseConcluded": "Unlicense OR MIT",
  "supplier": "NOASSERTION",
  "externalRefs": [
    {
      "referenceCategory": "PACKAGE-MANAGER",
      "referenceLocator": "pkg:cargo/memchr@2.5.0",
      "referenceType": "purl"
    }
  ]
},
{
  "SPDXID": "SPDXRef-rust-mio-0.8.8",
  "name": "rust:mio",
  "versionInfo": "0.8.8",
  "downloadLocation": "NOASSERTION",
  "filesAnalyzed": false,
  "licenseConcluded": "MIT",
  "supplier": "NOASSERTION",
  "externalRefs": [
    {
      "referenceCategory": "PACKAGE-MANAGER",
      "referenceLocator": "pkg:cargo/mio@0.8.8",
      "referenceType": "purl"
    }
  ]
}
```

A screenshot of a "Dependency graph" interface. It features tabs for "Dependencies", "Dependents", and "Dependabot", with an "Export SBOM" button. A search bar is present. Below, a list of dependencies is shown, including "mio 0.8.8", "shlex 1.1.0", "h2 0.3.20", and "openssl 0.10.55". Each entry includes a status icon (e.g., "1 high") and a "View Dependabot alert" link. A sidebar on the right displays a "Cargo.lock update suggested" alert for "mio ~> 0.8.11".

06. 오픈소스 AI 적용 시 고려 사항

- Hardening
 - 보안의 강화하기 위한 가이드
 - System, OS, Network등에서 폭넓게 적용
 - 소프트웨어 보안에서도 Hardening은 중요한 요소



Operating system selection

When selecting operating systems, it is important that an organisation preferences vendors that have demonstrated a commitment to secure-by-design and secure-by-default principles, use of memory-safe programming languages where possible (such as C#, Go, Java, Ruby, Rust and Swift), secure programming practices, and maintaining the security of their products. This will assist not only with reducing the potential number of vulnerabilities in operating systems, but also increasing the likelihood that timely patches, updates or vendor mitigations will be released to remediate any vulnerabilities that are found.

Control: ISM-1743; **Revision:** 1; **Updated:** Mar-23; **Applicability:** All; **Essential Eight:** N/A

Operating systems are chosen from vendors that have demonstrated a commitment to secure-by-design and secure-by-default principles, use of memory-safe programming languages where possible, secure programming practices, and maintaining the security of their products.

Operating system releases and versions

Newer releases of operating systems often introduce improvements in security functionality. This can make it more difficult for malicious actors to craft reliable exploits for vulnerabilities they discover. Using older releases of operating systems, especially those no longer supported by vendors, may expose an organisation to vulnerabilities or exploitation techniques that have since been mitigated. In addition, 64-bit versions of operating systems support additional security functionality that 32-bit versions do not.

Control: ISM-1407; **Revision:** 5; **Updated:** Dec-22; **Applicability:** All; **Essential Eight:** ML3

The latest release, or the previous release, of operating systems are used.

Control: ISM-1408; **Revision:** 5; **Updated:** Dec-22; **Applicability:** All; **Essential Eight:** N/A

Where supported, 64-bit versions of operating systems are used.

Standard Operating Environments

Allowing users to setup, configure and maintain their own workstations and servers can result in an inconsistent operating environment. Such operating environments may assist malicious actors in gaining an initial foothold on networks due to the higher likelihood of poorly configured or maintained workstations and servers. Conversely, a Standard Operating Environment (SOE), provided via an automated build process or a golden image, is designed to facilitate a standardised and consistent operating environment within an organisation.

When SOEs are obtained from third parties, such as service providers, there are additional cyber supply chain risks that should be considered, such as the accidental or deliberate inclusion of malicious code or configurations. To reduce the likelihood of such occurrences, an organisation should endeavour to obtain their SOEs from trusted third parties while also scanning them for malicious code and configurations.

<출처: 호주 사이버 보안 센터(ACSC) / OS Hardening>

감사합니다

www.wisestone.kr

경기도 과천시 과천대로 7길 33, 디테크타워 B동 1407호

T. 02-6257-5958 | F. 02-6257-5957

sales@wisestone.kr

