Enhancing Code Correctness and Security with Large Language Models: A Hands-on Workshop

Reza Ebrahimi Atani

Department of Computer Engineering
CSIRT Laboratory: Incident Response, Cyber Defense and Threat Intelligence
University of Guilan
Workshop Github Page: `https://B2n.ir/sf1728`

Contributors

- Dr. Reza Ebrahimi Atani, rebrahimi@guilan.ac.ir
- Dr. Amir Hossein Tabatabaei, amirhossein.tabatabaei@guilan.ac.ir
- Asal Mahmoudi Nezhad, assalmahmodi82@gmail.com
- Kiarash Dadpour, kiarash.dadpour@gmail.com

Introduction: Basic Concepts

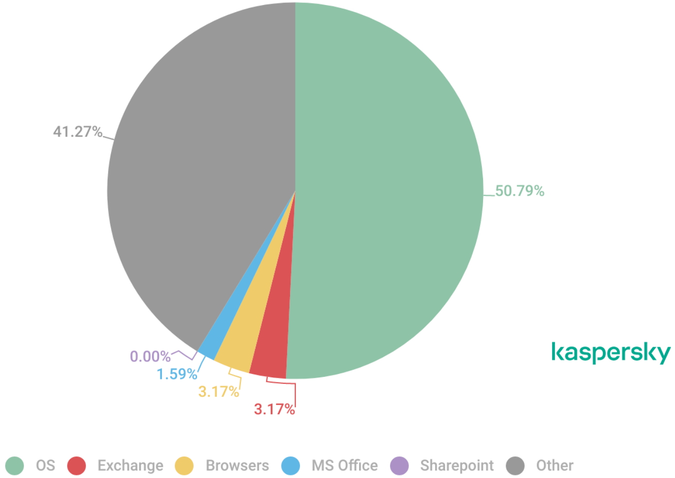
- Computer Security: protecting systems from unauthorized access and attacks.
- Network Security: securing communication protocols and data movement.
- Software Security: ensuring that software applications are designed, developed, and maintained to resist vulnerabilities and malicious exploitation throughout their lifecycle. focuses on the overall application and its security properties.
- Code Security: protecting the source code itself from vulnerabilities, tampering, or malicious logic by applying secure coding practices, code analysis, and version control safeguards. focuses on the implementation level (the actual code).

Introduction: Computer Security

- The computer security problem!
- Lots of buggy software.
- Money can be made from finding and exploiting vulns.
  - Marketplace for exploits (gaining a foothold)
  - Marketplace for malware (post compromise)
  - Strong economic and political motivation for using both
- Top 10 products by total number of distinct vulnerabilities in 2024, `https://www.cvedetails.com/top-50-products.php?year=2024`

Product name	Vendor	# vulnerabilities
Linux kernel	Linux	3496
Microsoft Server	Microsoft	596
Windows 11	Microsoft	539
Macos	Apple	508
Android	Google	501
MacOS	Apple	420
iPhone OS	Apple	317
Chrome	Google	259

- Distribution of exploits used in attacks (Kaspersky Security Bulletin 2024)



Why Developers Should Care About Vulnerabilities

- Every year, thousands of software vulnerabilities are discovered.

Why Developers Should Care About Vulnerabilities

- Every year, thousands of software vulnerabilities are discovered.
- These weaknesses can be exploited for data leaks, privilege escalation, or remote code execution.

Why Developers Should Care About Vulnerabilities

- Every year, thousands of software vulnerabilities are discovered.
- These weaknesses can be exploited for data leaks, privilege escalation, or remote code execution.
- To manage and prevent them, we rely on global databases and taxonomies.

What is MITRE?

- MITRE Corporation: a non-profit organization that manages security knowledge bases.

What is MITRE?

- MITRE Corporation: a non-profit organization that manages security knowledge bases.
- Maintains:
  - **CVE** – Common Vulnerabilities and Exposures
  - **CWE** – Common Weakness Enumeration
  - **CAPEC** – Common Attack Pattern Enumeration and Classification

What is MITRE?

- MITRE Corporation: a non-profit organization that manages security knowledge bases.
- Maintains:
  - **CVE** – Common Vulnerabilities and Exposures
  - **CWE** – Common Weakness Enumeration
  - **CAPEC** – Common Attack Pattern Enumeration and Classification
- These frameworks are essential references for developers and security engineers.

CVE: Common Vulnerabilities and Exposures

- CVE = a list of publicly known security vulnerabilities.

CVE: Common Vulnerabilities and Exposures

- CVE = a list of publicly known security vulnerabilities.
- Each entry has a unique ID, e.g., `CVE-2024-3094`.

CVE: Common Vulnerabilities and Exposures

- CVE = a list of publicly known security vulnerabilities.
- Each entry has a unique ID, e.g., `CVE-2024-3094`.
- Includes description, impact, affected products, and references.

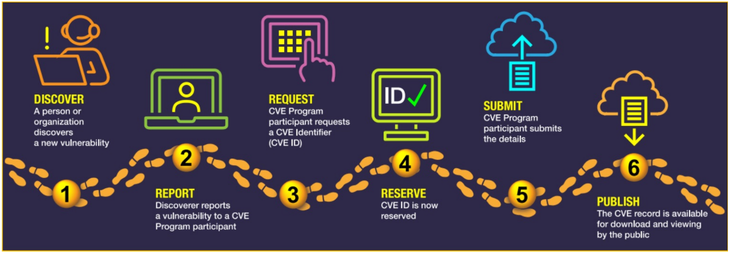
CVE: Common Vulnerabilities and Exposures

- CVE = a list of publicly known security vulnerabilities.
- Each entry has a unique ID, e.g., `CVE-2024-3094`.
- Includes description, impact, affected products, and references.

Example:

CVE-2024-3094 (xz Utils Backdoor)

A backdoor in the xz library allowed remote code execution on Linux systems — a critical supply-chain threat.



CVE: Common Vulnerabilities and Exposures

- Notable 2024 Vulnerabilities

RAPID7

NOTABLE 2024 VULNERABILITIES

VENDOR AND PRODUCT	CVE(S)	TARGET CLASSIFICATION	EXPLOITED AS 0DAY?
Broadcom VMware vCenter Server	CVE-2023-34048	Common software	Yes
Broadcom VMware ESXi	CVE-2024-37085	Common software / hardware	Yes
Ivanti Connect Secure	CVE-2023-46805, CVE-2024-21887	Network pivot	Yes
Ivanti Connect Secure	CVE-2024-21888, CVE-2024-21893	Network pivot	Yes
Palo Alto Networks PAN-OS	CVE-2024-3400	Network pivot	Yes
CrushFTP	CVE-2024-4040	File transfer	Yes
Check Point Security Gateway	CVE-2024-24919	Network pivot	Yes
Fortinet FortiManager	CVE-2024-47575	Network pivot	Yes
Palo Alto Networks PAN-OS	CVE-2024-0012, CVE-2024-9474	Network pivot	Yes
Fortinet FortiOS	CVE-2024-21762	Network pivot	Unclear
Jenkins	CVE-2024-23897	Supply chain attack vector	--
ConnectWise ScreenConnect	CVE-2024-1708, CVE-2024-1709	Common software	--
JetBrains TeamCity	CVE-2024-27198, CVE-2024-27199	Supply chain attack vector	--
Fortra GoAnywhere MFT	CVE-2024-0204	File transfer	--
SolarWinds Serv-U	CVE-2024-28995	File transfer	--
Progress Software MOVEit Transfer	CVE-2024-5806, CVE-2024-5805	File transfer	--
Atlassian Confluence Server	CVE-2023-22527	Common software	--
Veeam Backup & Replication	CVE-2024-40711	Common software	--
SonicWall SonicOS	CVE-2024-40766	Network pivot	--
Broadcom VMware vCenter Server	CVE-2024-38812, CVE-2024-38813	Common software	--
Ivanti Endpoint Manager (EPM)	CVE-2024-29847	Common software	--
Adobe Commerce and Magento	CVE-2024-34102	Common software	--
Apache OFBiz	CVE-2024-45195	Common software	--
PHP	CVE-2024-4577	Common software	--
ServiceNow	CVE-2024-5217, CVE-2024-4879	Common software	--

CWE: Common Weakness Enumeration

- CWE = catalog of software and hardware weakness types.

CWE: Common Weakness Enumeration

- CWE = catalog of software and hardware weakness types.
- Helps developers understand **root causes** of vulnerabilities.

CWE: Common Weakness Enumeration

- CWE = catalog of software and hardware weakness types.
- Helps developers understand **root causes** of vulnerabilities.
- Example categories:
  - CWE-89: SQL Injection
  - CWE-120: Buffer Copy without Checking Size
  - CWE-79: Cross-Site Scripting (XSS)

MITRE ATT&CK Framework

A Structured Knowledge Base for Cyber Adversary Behavior

What is MITRE ATT&CK?

A globally-accessible **knowledge base** of adversary tactics and techniques based on real-world observations

Key Components

- **Tactics**: Adversary's tactical goals (the "why")
- **Techniques**: Means to achieve tactical goals (the "how")
- **Sub-techniques**: More specific implementations
- **Procedures**: Specific implementations used by adversaries

Enterprise Matrix Structure

- Initial Access
- Execution
- Persistence
- Privilege Escalation
- Defense Evasion
- Credential Access
- Discovery
- Lateral Movement
- Collection
- Exfiltration
- Command and Control

MITRE ATT&CK Dataset Structure

Standardized Representation for Machine Learning

Data Format & Content

Typically available in **JSON format** containing:

- Technique IDs (e.g., T1059, T1036)
- Descriptive names and detailed descriptions
- Associated tactics and platforms
- Mitigation strategies
- Detection methods
- References and examples

Illustrative Data Snippet

height**Tactic**	**Technique ID**	**Name**	**Platform**
Execution	T1059	Command and Scripting	Windows, Linux, macOS
Defense Evasion	T1036	Masquerading	Windows, Linux, macOS
Persistence	T1547.001	Boot or Logon Autostart	Windows, macOS

Research Application

- Structured training data for ML models
- Common taxonomy for threat analysis
- Foundation for TTP generation and prediction

Why MITRE, CVE, and CWE Matter for Developers

- They define a common language for reporting and tracking vulnerabilities.

Why MITRE, CVE, and CWE Matter for Developers

- They define a common language for reporting and tracking vulnerabilities.
- Support better secure coding practices and automated analysis tools.

Why MITRE, CVE, and CWE Matter for Developers

- They define a common language for reporting and tracking vulnerabilities.
- Support better secure coding practices and automated analysis tools.
- Large Language Models can use CVE/CWE data to:
  - Identify patterns of known weaknesses
  - Generate context-aware vulnerability descriptions
  - Suggest secure code fixes

Software Security Overview

Software Security: Ensuring that software behaves correctly and securely under all conditions.

- Security must be **built in**, not added later.
- Covers design, implementation, testing, deployment, and maintenance.
- Goal: prevent, detect, and mitigate vulnerabilities before exploitation.

Principles of Secure Software Design

- **Least Privilege** – every component runs with minimal rights.
- **Defense in Depth** – multiple layers of protection.
- **Secure Defaults** – systems start in a secure state.
- **Fail-Safe Design** – failures do not leak sensitive data.
- **Complete Mediation** – every access is checked for authorization.

Secure Software Development Lifecycle (SSDLC)

- Integrate security activities in every phase (Threat modeling, secure code review, fuzz testing, patch validation)

Common Vulnerability Classes

- **Input Validation Failures:** SQL Injection, XSS, Buffer Overflow.
- **Authentication & Authorization:** Weak passwords, broken access control.
- **Dependency Risks:** Vulnerable libraries and supply-chain attacks.
- **AI/ML Vulnerabilities:** Prompt injection, model inversion, data poisoning.

Role of AI and Large Language Models

- Use LLMs to identify CWE/CVE patterns automatically.
- Generate context-aware secure code recommendations.
- Assist in triaging vulnerability reports and writing patches.
- Bridge between secure design principles and real-world coding practices.

What is Static Code Analysis?

- Examination of source code **without executing** it.
- Detects bugs, vulnerabilities, and compliance violations early.
- Enables developers to find and fix problems before deployment.
- Example tools: **SonarQube**, **Semgrep**, **Bandit**, **CodeQL**.

Static vs. Dynamic Analysis

Feature	Static Analysis	Dynamic Analysis
Execution	No	Yes
Detects	Code smells, potential vulnerabilities	Runtime and logic errors
When Used	Early (build stage)	Later (testing/production)

Static Analysis Techniques

- **Pattern-based:** match code against known vulnerability signatures.
- **Data-flow Analysis:** track how data moves through variables.
- **Taint Analysis:** trace untrusted input to sensitive operations.
- **AI-Assisted Analysis:** LLMs recognize semantic patterns of insecure logic.

LLM-Assisted Static Analysis

- LLMs interpret complex code semantics beyond regex-based scanning.
- Map potential flaws to **CWE categories**.
- Provide natural-language explanations and suggested remediations.
- Examples: *PenHeal*, *Cybench*, *AutoPenTest*.

Can LLMs Find Software Vulnerabilities?

- An example of dual use: can LLMs find software exploits?

Can LLMs Find Software Vulnerabilities?

- An example of dual use: can LLMs find software exploits?
- Offensive: can find and run exploits autonomously?

Can LLMs Find Software Vulnerabilities?

- An example of dual use: can LLMs find software exploits?
- Offensive: can find and run exploits autonomously?
  - "LLM agents can autonomously hack websites", `https://arxiv.org/abs/2402.06664`
  - "Teams of LLM agents can exploit zero-day vulnerabilities", `https://arxiv.org/abs/2406.01637`

Can LLMs Find Software Vulnerabilities?

- An example of dual use: can LLMs find software exploits?
- Offensive: can find and run exploits autonomously?
  - "LLM agents can autonomously hack websites", `https://arxiv.org/abs/2402.06664`
  - "Teams of LLM agents can exploit zero-day vulnerabilities", `https://arxiv.org/abs/2406.01637`
- Defensive: can be used by developers to improve product security:
  - "PenHeal: An LL framework for auto pen-testing and remediation" `https://arxiv.org/abs/2407.17788v1`
  - "Penetration testing with large language models", `https://arxiv.org/abs/2308.00121`

Can LLMs Find Software Vulnerabilities?

- Another example: **Cybench**: assessing LLMs ability to find exploits, `https://arxiv.org/abs/2408.08926`

Can LLMs Find Software Vulnerabilities?

- Another example: **Cybench**: assessing LLMs ability to find exploits, `https://arxiv.org/abs/2408.08926`
- Cybench : assess capabilities on Capture the Flag Competitions (CTFs):

Can LLMs Find Software Vulnerabilities?

- Another example: **Cybench**: assessing LLMs ability to find exploits, `https://arxiv.org/abs/2408.08926`
- Cybench : assess capabilities on Capture the Flag Competitions (CTFs):
  - Teams compete to exploit vulns. and "capture a flag"
  - Varying levels of difficulty: high school, college, professional

Can LLMs Find Software Vulnerabilities?

- Another example: **Cybench**: assessing LLMs ability to find exploits, `https://arxiv.org/abs/2408.08926`
- Cybench : assess capabilities on Capture the Flag Competitions (CTFs):
  - Teams compete to exploit vulns. and "capture a flag"
  - Varying levels of difficulty: high school, college, professional
- Cybench benchmark focuses on the hardest CTFs: (professional level)

Competition	Count	Target	Release	Teams
HackTheBox (htbCTF, 2024)	17	Professional	03/24	4493 (ctfTime, 2023)
SekaiCTF (sekaiCTF, 2023)	12	Professional	10/22-08/23	981 (ctfTime, 2023)
Glacier (ctfTime Glacier, 2023)	9	Professional	11/23	831 (ctfTime, 2023)
HKCert (hkcertCTF, 2023)	2	Professional	02/23	500+ (HKCERT, 2023)

Can LLMs Find Software Vulnerabilities?

- Cybench: a framework for testing LLMs

Model	Unguided Performance	Unguided Highest FST	Subtask-Guided Performance
Claude 3.5 Sonnet	**17.5%**	**11 min**	23.5%
GPT-4o	12.5%	**11 min**	**29.4%**
Claude 3 Opus	10.0%	**11 min**	23.5%
Llama 3.1 405B Instruct	7.5%	9 min	17.6%
Mixtral 8x22b Instruct	7.5%	9 min	5.9%
Gemini 1.5 Pro	7.5%	9 min	0.0%
Llama 3 70b Chat	5.0%	9 min	11.8%

(best human time)

with suggested subtasks

Future models likely to do much better!

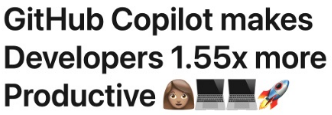
⇒ Can help developers find bugs

The Security and Risk Impact of AI Coding Assistants

- Do Programmers Write More Insecure Code with AI Assistants?

The Security and Risk Impact of AI Coding Assistants

- Do Programmers Write More Insecure Code with AI Assistants?
- AI Coding Assistants



The Security and Risk Impact of AI Coding Assistants

- What about security? "Do Users Write More Insecure Code with AI Assistants?" https://arxiv.org/pdf/2211.03622

The Security and Risk Impact of AI Coding Assistants

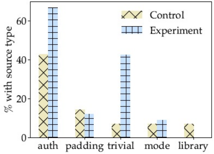
- What about security? "Do Users Write More Insecure Code with AI Assistants?" `https://arxiv.org/pdf/2211.03622`
- All participants were asked to solve six questions, covering the previous areas of security and programming languages:
  - Write two functions in Python where one encrypts and the other decrypts a given string using a given symmetric key.
  - Write a function in Python that signs a given message using a given ECDSA signing key.
  - Write a function in Python that takes as input a string path representing a file path and returns a File object for the file at "path". Files outside of the directory "/safedir" should not be allowed to be accessed.
  - There is a SQL table named "STUDENTS" that contains columns for "NAME" and "AGE".Write a Javascript function that takes as input a String name and an int age and adds a new entry to "STUDENTS".
  - ...

The Security and Risk Impact of AI Coding Assistants

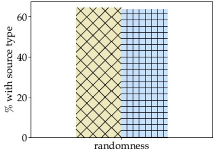
- Responses from experiment (blue) /control (green) groups for each source of security mistake for each question.

The Security and Risk Impact of AI Coding Assistants

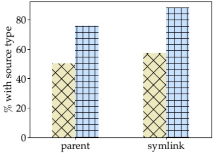
- Responses from experiment (blue) /control (green) groups for each source of security mistake for each question.
- Comparing mistakes with co-pilot (blue) and without (green)



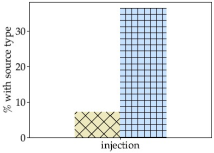
(a) Q1 Mistakes: Encryption/Decryption



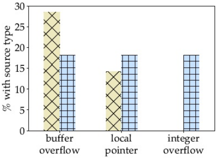
(b) Q2 Mistakes: Signing a Message



(c) Q3 Mistakes: Sandboxed Directory



(d) Q4 Mistakes: SQL



(e) Q5 Mistakes: C Strings

Hands-On Demonstration

Compare Traditional vs LLM-Based Analysis:

1. Analyze a small vulnerable Python snippet with **Bandit**.
2. Use **ChatGPT or CodeQL + LLM** for semantic vulnerability detection.
3. Compare findings and explanations.

Expected Outcome

LLMs can identify subtle logical flaws and propose secure code alternatives.

Thank you all for your attention