



**black hat**<sup>®</sup>  
USA 2024

**AUGUST 7-8, 2024**  
BRIEFINGS

# **From HAL to HALT: Thwarting Skynet's Siblings in the GenAI Coding Era**

Chris Wysopal

Co-founder & CTO, Veracode

**VERACODE**

#BHUSA @BlackHatEvents

One of the 1<sup>st</sup> vulnerability researchers, member of hacker think tank, L0pht in 1990s



Unites States Senate testimony - 19 May 1998





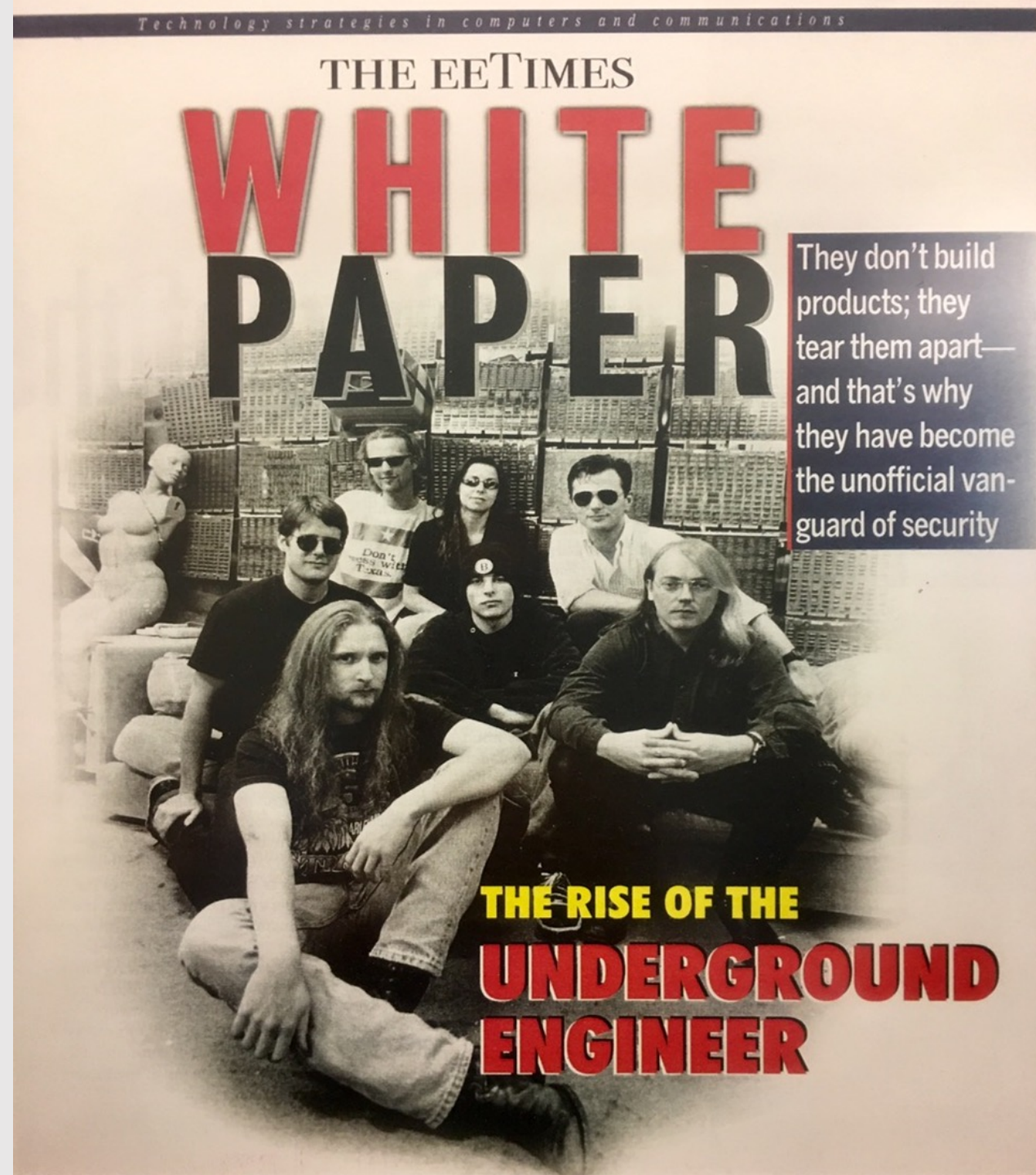
**Into the light:** *Once shadowy computer code warriors like Kingpin are going legit*

## Using Good Hackers to Battle Bad Hackers

**I**F YOU HAVE A MURKY PAST AND DOUBT you could become a dot-com millionaire, think again. Last week a scraggly band of hackers known as “LOpht Heavy Industries” joined with some straitlaced tech execs to form @Stake, an Internet-security consulting firm.

Newsweek, January 17, 2000

Improve the  
Security of  
Your *Product*  
by Breaking  
Into It





**Founded  
@stake security  
research team  
and then  
Veracode to  
build security  
into SDLC**



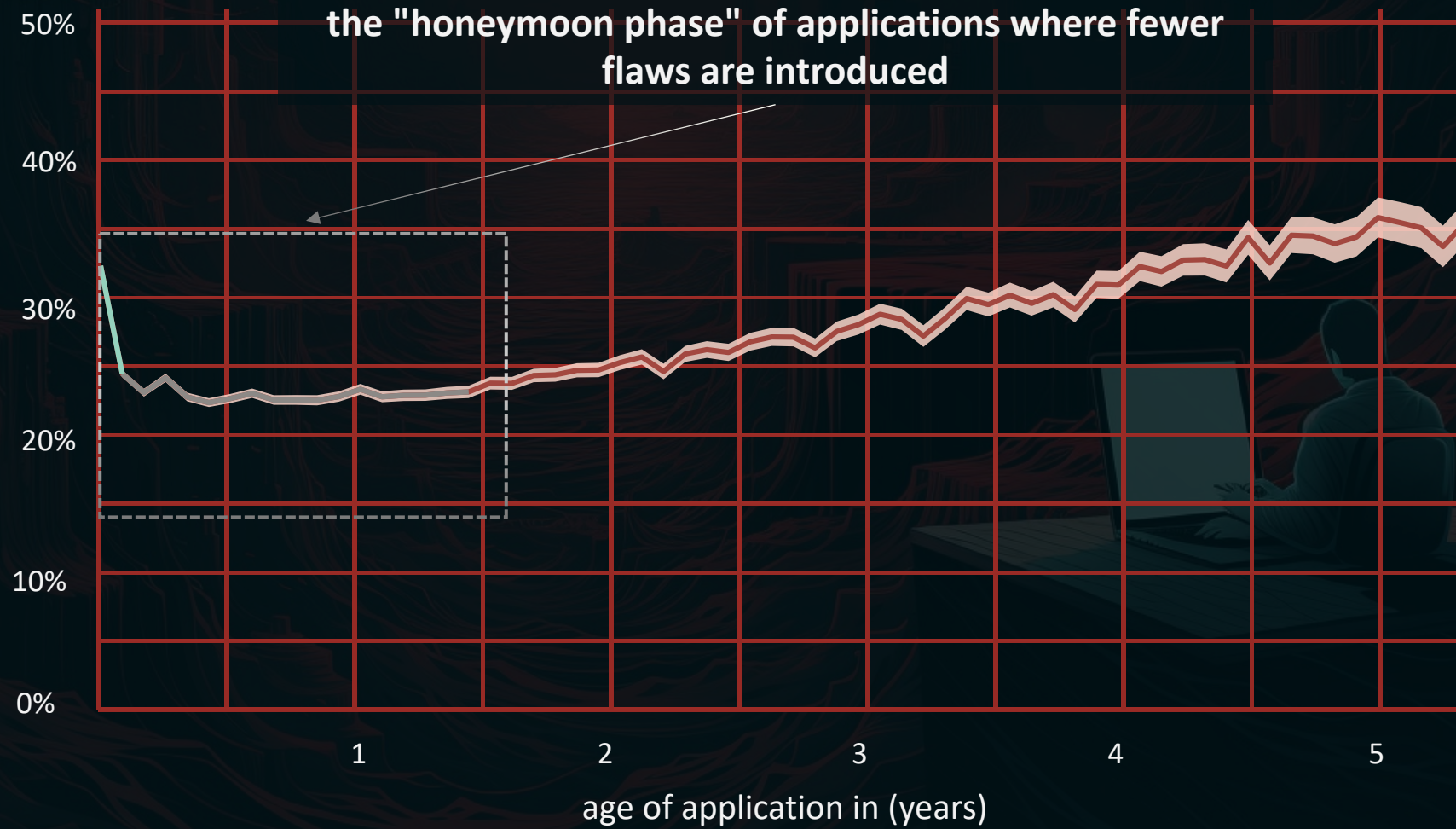
A silhouette of a person sitting cross-legged on a dark, jagged mountain peak. The person is holding a glowing blue laptop. The mountain is covered in binary code (0s and 1s) in various colors (blue, green, red). The background is a light blue gradient.

# State of Software Security 2024

Addressing the  
Threat of Security Debt



# new flaws introduced by application age





# organizations are drowning in **security debt**



**70.8%**  
of organizations  
have security  
debt



**45%**  
of organizations  
have critical  
security debt





**few teams  
fix flaws fast  
enough** to reduce  
security risk at a  
**meaningful pace**





## why software security is **hard**

- security knowledge gaps
- increased application complexity
- incomplete view of risk
- evolving threat landscape



Let's add the exciting potential of large language models that can write code!





# Developer GenAI use right now

## Generating code

Understanding code/Code review

Remediating defects

Translating programming languages

Creating and maintaining unit tests

Writing documentation



# Emerging dev uses for GenAI

Learning about the code base

Searching for answers to avoid  
reinventing the wheel

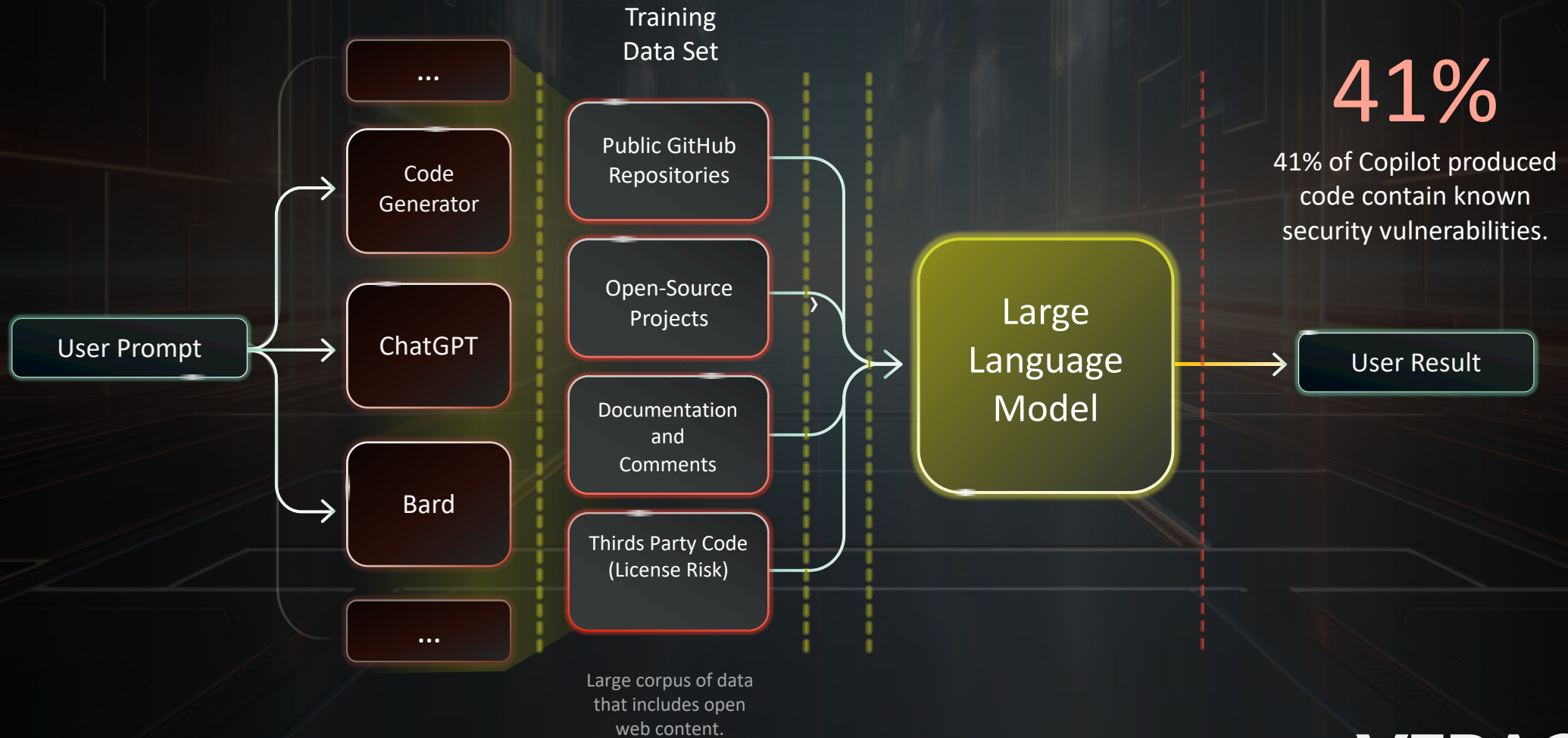
Reading log files to find a root  
cause

Creating and running  
functional & non-functional  
tests

Remediating security  
vulnerabilities



# Large Language Models





# Security implications of LLMs

## Wuhan University Study on AI Code Generators

## New York University Study on GitHub Copilot

## Stanford University Study on AI Code Generators

## Purdue University on ChatGPT accuracy

36%

41%

52%

Out of the 435 Copilot generated code snippets found in repos 36% contain security weaknesses across 6 programming languages.

Of 1689 generated programs 41% of Copilot produced programs contained vulnerabilities

Developers using LLMs were more likely to write insecure code.

They were more confident their code was secure.

52% of ChatGPTs answers were incorrect. Developers preferred them 35% of the time yet 77% of those answers were wrong

### Security Weaknesses of Copilot Generated Code in GitHub

Yujia Fu School of Computer Science Wuhan, China yujiafu@whu.edu.cn	Peng Liang School of Mathematical and Computational Sciences Wuhan, China liangpeng@whu.edu.cn	Amjad Tahir Department of Mathematics and Statistics University of Toronto amjad.tahir@utoronto.ca
Zengyang Li School of Computer Science Central China Normal University Wuhan, China zengyangli@ccnui.edu.cn	Mojtaba Shahin School of Computer Science BMT University Melbourne, Australia mojtaba.shahin@bmt.edu.au	Jiayu Yu School of Computer Science Wuhan University jiayu.yu@whu.edu.cn

**ABSTRACT**

Modern code generation tools use AI models, particularly Large Language Models (LLMs), to generate functional and complete code. While such tools are becoming popular and widely available for developers, using these tools is often accompanied by security challenges, leading to insecure code being pushed into the code base. Therefore, it is important to assess the quality of the generated code, especially in terms of the security. Researchers have recently explored various aspects of code generation tools, including accuracy. However, many open questions about the security of the generated code require further investigation, especially the security issue of automatically generated code in the wild. In this work, we conducted an empirical study by analyzing the security weaknesses in code snippets generated by GitHub Copilot that we found as part of publicly available projects hosted on GitHub. The goal is to investigate the types of security issues and their risks to the world's ecosystem (rather than crafted scenarios). To this end, we identified 435 code snippets generated by GitHub Copilot from publicly available projects. We then conducted a comprehensive security analysis to identify Common Weakness Enumeration (CWE) instances in these code snippets. The results show that (1) 36.1% of Copilot-generated code snippets contain CVEs, and these issues are general across multiple languages. (2) the security weaknesses are related to the underlying CVEs, in which, 75% are CWE. (3) the most common weakness, CVE-2021-44228 (Insufficiently Sanitized Input), and CVE-2017-18203 (Integer Overflow or Wraparound) are the most frequently found being in the currently recognized CWE Top-25. Our findings confirm that developers should be careful when utilizing Copilot-generated code, and that the generated code should be reviewed by Copilot and manual review [16]. As a companion benchmark and baseline, we also compare security checks on the generated code by other state-of-the-art programming language-based tools to support our findings, making automated code generation technology more efficient and flexible. In recent years, the rapid development of artificial intelligence technologies, particularly in the form of large language models, has revolutionized the way we interact with computers. Amongst these, AI-powered code generators, such as Copilot, have gained significant attention. However, these tools often generate code without understanding capabilities that can be used for tasks such as natural

arXiv:2310.02059v1 [cs.SE] 3 Oct 2023

### 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	Balogh Ahmad Department of ECE New York University Brooklyn, NY, USA balogh@nyu.edu	Benjamin Tan Department of ESE University of Calgary Calgary, Alberta, CA benjamin.tan@ucalgary.ca	Brendan Dolan-Gavitt Department of ECE New York University Brooklyn, NY, USA bdolan-gavitt@nyu.edu	Ramak Kari Department of ECE New York University Brooklyn, NY, USA ramak.kari@nyu.edu
---	---	--	--	---

**ABSTRACT**

There is burgeoning interest in studying AI-based systems to assist humans in developing computer code. As GitHub Copilot is the largest and most capable such model currently available, it is important to understand how Copilot's suggestions commonly improve? What is the prevalence of insecure generated code? What factors of the model's output are most likely to be insecure? In this paper, we investigate these questions by analyzing the security weaknesses in code snippets generated by GitHub Copilot that we found as part of publicly available projects hosted on GitHub. The goal is to investigate the types of security issues and their risks to the world's ecosystem (rather than crafted scenarios). To this end, we identified 1689 code snippets generated by GitHub Copilot from publicly available projects. We then conducted a comprehensive security analysis to identify Common Weakness Enumeration (CWE) instances in these code snippets. The results show that (1) 41.1% of Copilot-generated code snippets contain CVEs, and these issues are general across multiple languages. (2) the security weaknesses are related to the underlying CVEs, in which, 75% are CWE. (3) the most common weakness, CVE-2021-44228 (Insufficiently Sanitized Input), and CVE-2017-18203 (Integer Overflow or Wraparound) are the most frequently found being in the currently recognized CWE Top-25. Our findings confirm that developers should be careful when utilizing Copilot-generated code, and that the generated code should be reviewed by Copilot and manual review [16]. As a companion benchmark and baseline, we also compare security checks on the generated code by other state-of-the-art programming language-based tools to support our findings, making automated code generation technology more efficient and flexible. In recent years, the rapid development of artificial intelligence technologies, particularly in the form of large language models, has revolutionized the way we interact with computers. Amongst these, AI-powered code generators, such as Copilot, have gained significant attention. However, these tools often generate code without understanding capabilities that can be used for tasks such as natural

arXiv:2311.03622v3 [cs.CR] 18 Dec 2023

### Do Users Write More Insecure Code with AI Assistants?

Neil Perry* Stanford University	Migsha Srivastava* Stanford University	Deepak Kumar Stanford University / UC	Dan Boneh Stanford University
------------------------------------	---	--	----------------------------------

**ABSTRACT**

AI code assistants have emerged as powerful tools that can aid in the software development life cycle and help improve developer productivity. Unfortunately, our assistants have also been found to produce insecure code, or, at the very least, code that is more likely to be insecure than code that was written without their assistance. In this paper, we conduct a user study to measure how often users write code with their assistants to solve a variety of security-related tasks. Overall, we find that participants who did access to AI assistants were significantly more likely to write insecure code than those who did not. In fact, we found that participants who accessed AI assistants were more likely to write insecure code, suggesting that such tools may have had users write more insecure code than those who did not. To better understand the design of future AI-based code assistants, we release our user study questions and interpreted data to researchers seeking to build our work into their AI.

**KEYWORDS**

Security and privacy • Human and societal aspects of security and privacy • Languages and models • Machine learning • Usable security

**ACM Reference Format:**

Neil Perry, Migsha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do Users Write More Insecure Code with AI Assistants? In Proceedings of the ACM Conference on Computer and Communications Security (CCS '23), November 28–30, 2023, Copenhagen, Denmark. ACM, New York, NY, USA. 16 pages. <https://doi.org/10.1145/3603033>

**1 INTRODUCTION**

AI code assistants, like GitHub Copilot, have emerged as programming tools with the potential to lower the barrier of entry for many tasks, including writing code. These tools leverage underlying machine learning models, like OpenAI GPT-3 and Facebook's LLaMA [15, 11], that use pre-trained large datasets of publicly available code to help developers. When recent work has demonstrated that models may erroneously produce weaker code than humans [17], our study has comprehensively measured the security of code assistants in the context of how developers choose to use them. Such work is important in order to understand the practical security challenges introduced by AI-powered code assistants and the ways users prompt the AI system to inappropriately cause security mistakes.

In this paper, we create how developers choose to interact with AI code assistants and how these interactions can cause security mistakes. To do this, we designed and conducted a comprehensive user study where participants received five security-related programming tasks spanning three different programming languages (Python, JavaScript, and C#) over the study in three to four-week sessions.

Participants who accessed an AI assistant were more likely to write insecure code than those who did not. In fact, we found that participants who accessed AI assistants were more likely to write insecure code than those who did not. To better understand the design of future AI-based code assistants, we release our user study questions and interpreted data to researchers seeking to build our work into their AI.

**KEYWORDS**

Security and privacy • Human and societal aspects of security and privacy • Languages and models • Machine learning • Usable security

**ACM Reference Format:**

Neil Perry, Migsha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do Users Write More Insecure Code with AI Assistants? In Proceedings of the ACM Conference on Computer and Communications Security (CCS '23), November 28–30, 2023, Copenhagen, Denmark. ACM, New York, NY, USA. 16 pages. <https://doi.org/10.1145/3603033>

**1 INTRODUCTION**

AI code assistants, like GitHub Copilot, have emerged as programming tools with the potential to lower the barrier of entry for many tasks, including writing code. These tools leverage underlying machine learning models, like OpenAI GPT-3 and Facebook's LLaMA [15, 11], that use pre-trained large datasets of publicly available code to help developers. When recent work has demonstrated that models may erroneously produce weaker code than humans [17], our study has comprehensively measured the security of code assistants in the context of how developers choose to use them. Such work is important in order to understand the practical security challenges introduced by AI-powered code assistants and the ways users prompt the AI system to inappropriately cause security mistakes.

In this paper, we create how developers choose to interact with AI code assistants and how these interactions can cause security mistakes. To do this, we designed and conducted a comprehensive user study where participants received five security-related programming tasks spanning three different programming languages (Python, JavaScript, and C#) over the study in three to four-week sessions.

Participants who accessed an AI assistant were more likely to write insecure code than those who did not. In fact, we found that participants who accessed AI assistants were more likely to write insecure code than those who did not. To better understand the design of future AI-based code assistants, we release our user study questions and interpreted data to researchers seeking to build our work into their AI.

**KEYWORDS**

Security and privacy • Human and societal aspects of security and privacy • Languages and models • Machine learning • Usable security

**ACM Reference Format:**

Neil Perry, Migsha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do Users Write More Insecure Code with AI Assistants? In Proceedings of the ACM Conference on Computer and Communications Security (CCS '23), November 28–30, 2023, Copenhagen, Denmark. ACM, New York, NY, USA. 16 pages. <https://doi.org/10.1145/3603033>

### Who Answers It Better? An In-Depth Analysis of ChatGPT and Stack Overflow Answers to Software Engineering Questions

Samia Kabir Purdue University West Lafayette, USA skabir@purdue.edu	David N. Ush-Izch Purdue University West Lafayette, USA dush@purdue.edu
Beran Kim Purdue University West Lafayette, USA beran@purdue.edu	Tianyi Zhang Purdue University West Lafayette, USA tzhang@purdue.edu

**ABSTRACT**

Over the last decade, Q&A platforms have played a crucial role in how programmers seek help online. The emergence of ChatGPT, however, is raising a red flag in the software engineering community. This paper, therefore, has been through investigation into the quality and usability of responses to software engineering queries. To address this gap, we conducted a comprehensive analysis of ChatGPT's replies to 317 questions from Stack Overflow [30]. We analyzed the correctness, consistency, comprehensiveness, and conciseness of these responses. Additionally, we conducted an extensive linguistic analysis and a user study to gain insights into the linguistic and human aspects of ChatGPT's answers. The results revealed that ChatGPT's responses contain inaccuracies and provide less detailed information than Stack Overflow [30]. These findings underscore the need for machine learning-based approaches to improve the quality of AI-generated answers to software engineering queries. We introduced an open-source ChatGPT, which compared the popularity of other models in the context of ChatGPT's capacity to engage in human-like conversations, primarily based on contextual human feedback, and accessibility to the general public, have all contributed to its popularity. Consequently, ChatGPT's popularity has ignited a heated discussion about the roles of academic, research, and industry practitioners on Twitter and other social media platforms [21, 13]. This discussion involves several the prominent AI researchers and software engineers (e.g., Stack Overflow [30]).

Despite the increasing popularity of ChatGPT, concerns remain about its accuracy and reliability. This paper, therefore, has been through investigation into the quality and usability of responses to software engineering queries. To address this gap, we conducted a comprehensive analysis of ChatGPT's replies to 317 questions from Stack Overflow [30]. We analyzed the correctness, consistency, comprehensiveness, and conciseness of these responses. Additionally, we conducted an extensive linguistic analysis and a user study to gain insights into the linguistic and human aspects of ChatGPT's answers. The results revealed that ChatGPT's responses contain inaccuracies and provide less detailed information than Stack Overflow [30]. These findings underscore the need for machine learning-based approaches to improve the quality of AI-generated answers to software engineering queries. We introduced an open-source ChatGPT, which compared the popularity of other models in the context of ChatGPT's capacity to engage in human-like conversations, primarily based on contextual human feedback, and accessibility to the general public, have all contributed to its popularity. Consequently, ChatGPT's popularity has ignited a heated discussion about the roles of academic, research, and industry practitioners on Twitter and other social media platforms [21, 13]. This discussion involves several the prominent AI researchers and software engineers (e.g., Stack Overflow [30]).

**KEYWORDS**

Software and its engineering • General and reference • Empirical studies

**ACM Reference Format:**

Samia Kabir, David N. Ush-Izch, Beran Kim, and Tianyi Zhang. 2023. Who Answers It Better? An In-Depth Analysis of ChatGPT and Stack Overflow Answers to Software Engineering Questions. In Proceedings of the ACM Conference on Computer and Communications Security (CCS '23), November 28–30, 2023, Copenhagen, Denmark. ACM, New York, NY, USA. 16 pages. <https://doi.org/10.1145/3603033>

**1 INTRODUCTION**

Software developers often resort to online resources for a variety of software engineering tasks, e.g., API tutorials, but finding comprehensive code or snippets, etc. [13, 37]. As a vast repository of these help-seeking activities include frequent engagement with community Q&A platforms such as Stack Overflow [30] [33, 35, 43, 45] to seek help, solutions, or suggestions from other developers [37].

The emergence of Large Language Models (LLMs) has demonstrated the potential to transform the way help-seeking activities of software developers. Recent studies show that programmers still use 18 models such as GitHub Copilot [23] in their applications and queries of problems at hand and turn to its web interface or SO when they need a quick solution or answer to their questions [35, 43, 45]. The ability to engage in interactive conversations and provide solutions using natural language has propelled LLMs into becoming a popular option among programmers [27].

Investment in LLM progress in November 2023, ChatGPT [41] was introduced as an open-source ChatGPT, which compared the popularity of other models in the context of ChatGPT's capacity to engage in human-like conversations, primarily based on contextual human feedback, and accessibility to the general public, have all contributed to its popularity. Consequently, ChatGPT's popularity has ignited a heated discussion about the roles of academic, research, and industry practitioners on Twitter and other social media platforms [21, 13]. This discussion involves several the prominent AI researchers and software engineers (e.g., Stack Overflow [30]).

Despite the increasing popularity of ChatGPT, concerns remain about its accuracy and reliability. This paper, therefore, has been through investigation into the quality and usability of responses to software engineering queries. To address this gap, we conducted a comprehensive analysis of ChatGPT's replies to 317 questions from Stack Overflow [30]. We analyzed the correctness, consistency, comprehensiveness, and conciseness of these responses. Additionally, we conducted an extensive linguistic analysis and a user study to gain insights into the linguistic and human aspects of ChatGPT's answers. The results revealed that ChatGPT's responses contain inaccuracies and provide less detailed information than Stack Overflow [30]. These findings underscore the need for machine learning-based approaches to improve the quality of AI-generated answers to software engineering queries. We introduced an open-source ChatGPT, which compared the popularity of other models in the context of ChatGPT's capacity to engage in human-like conversations, primarily based on contextual human feedback, and accessibility to the general public, have all contributed to its popularity. Consequently, ChatGPT's popularity has ignited a heated discussion about the roles of academic, research, and industry practitioners on Twitter and other social media platforms [21, 13]. This discussion involves several the prominent AI researchers and software engineers (e.g., Stack Overflow [30]).

**KEYWORDS**

Software and its engineering • General and reference • Empirical studies

**ACM Reference Format:**

Samia Kabir, David N. Ush-Izch, Beran Kim, and Tianyi Zhang. 2023. Who Answers It Better? An In-Depth Analysis of ChatGPT and Stack Overflow Answers to Software Engineering Questions. In Proceedings of the ACM Conference on Computer and Communications Security (CCS '23), November 28–30, 2023, Copenhagen, Denmark. ACM, New York, NY, USA. 16 pages. <https://doi.org/10.1145/3603033>



# SALLM Framework For measuring LLM vulnerability generation - Notre Dame

Vulnerable@k metric best to worst:

StarCoder  
GPT-4:  
GPT-3.5:  
CodeGen-2.5-7B:  
CodeGen-2B:

## VULNERABILITIES FOUND IN THE CHATGPT-GENERATED PYTHON CODES

CWE Name	CWE Top-25 Rank	# Vuln. Samples
<b>CWE-312 Cleartext Storage of Sensitive Information</b>	-	14
<b>CWE-798 Use of Hard-coded Credentials</b>	18	5
<b>CWE-208 Observable Timing Discrepancy</b>	-	3
<b>CWE-215 Insertion of Sensitive Information Into Debugging Code</b>	-	3
<b>CWE-338 Use of Cryptographically Weak Random Generator</b>	-	3
<b>CWE-79 Cross-site Scripting</b>	2	2
<b>CWE-209 Generation of Error Message Containing Sensitive Information</b>	-	2
<b>CWE-287 Improper Authentication</b>	13	1
<b>CWE-295 Improper Certificate Validation</b>	-	1
<b>CWE-918 Server-Side Request Forgery</b>	19	1

### Generate and Pray: Using SALLM to Evaluate the Security of LLM Generated Code

Mohammed Latif Siddiq, Joana C. S. Santos, Sajith Devareddy and Anna Muller  
Department of Computer Science and Engineering,  
University of Notre Dame, Notre Dame, IN USA 46556

arXiv:2311.00889v2 [cs.SE] 3 Jun 2024

**Abstract**—With the growing popularity of Large Language Models (LLMs) in software engineers' daily practices, it is important to ensure that the code generated by these tools is not only functionally correct but also free of vulnerabilities. Although LLMs can help developers to be more productive, prior empirical studies have shown that LLMs can generate insecure code. There are two contributing factors to the insecure code generation. First, existing datasets used to evaluate LLMs do not adequately represent genuine software engineering tasks sensitive to security. Instead, they are often based on competitive programming challenges or classroom-type coding tasks. In real-world applications, the code produced is integrated into larger codebases, introducing potential security risks. Second, existing evaluation metrics primarily focus on the functional correctness of the generated code while ignoring security considerations. Therefore, in this paper, we described SALLM, a framework to benchmark LLMs' abilities to generate secure code systematically. This framework has three major components: a novel dataset of security-centric Python prompts, configurable assessment techniques to evaluate the generated code, and novel metrics to evaluate the models' performance from the perspective of secure code generation.

**Index Terms**—security evaluation, large language models, pre-trained transformer model, metrics

#### 1. INTRODUCTION

A code LLM is a Large Language Model (LLM) that has been trained on a large dataset consisting of both text and code [1]. As a result, code LLMs can generate code written in a specific programming language from a given prompt. These prompts provide a high-level specification of a developer's intent [2] and can include single/multi-line code comments, code expressions (e.g., a function definition), text, or a combination of these, etc. Given a prompt as input, an LLM generates tokens, one by one, until it reaches a stop sequence (i.e., a pre-configured sequence of tokens) or the maximum number of tokens is reached.

LLM-based source code generation tools are increasingly being used by developers in order to reduce software development efforts [3]. A recent survey with 500 US-based developers who work for large-sized companies showed that 92% of them are using LLMs to generate code for work and personal use [4]. Part of this fast widespread adoption is due to the increased productivity perceived by developers; LLMs help them to automate repetitive tasks so that they can focus on higher-level challenging tasks [3].

Although LLM-based code generation techniques may produce functionally correct code, prior works showed that they can also generate code with vulnerabilities and security smells [5]–[8]. A prior study has also demonstrated that training sets commonly used to train and/or fine-tune LLMs contain harmful coding patterns, which leak to the generated code [9]. Moreover, a recent study [6] with 47 participants showed that individuals who used the `codex-davinci-002` LLM wrote code that was *less secure* compared to those who did not use it. Even worse, participants who used the LLM were *more likely to believe that their code was secure*, unlike their peers who did not use the LLM to write code.

There are two major factors contributing to this unsafe code generation. First, code LLMs are evaluated using *benchmarks*, which do not include constructs to evaluate the security of the generated code [10], [11]. Second, existing *evaluation metrics* (e.g., pass@k [12], CodeBLEU [13], etc.) assess models' performance with respect to their ability to produce *functionally* correct code while ignoring security concerns. Therefore, the performance reported for these models overly focuses on improving the precision of the generated code with respect to passing the *functional* test cases of these benchmarks without evaluating the *security* of the produced code.

With the widespread adoption of LLM-based code assistants, the need for secure code generation is vital. Generated code containing vulnerabilities may get unknowingly accepted by developers, affecting the software system's reliability. Thus, to fulfill this need, this paper describes a framework to perform Security Assessment of LLMs (SALLM). Our framework includes a ① a manually curated dataset of prompts from a variety of sources that represent typical engineers' intent; ② an automated approach that relies on static and dynamic analysis to automatically evaluate the security of LLM generated Python code; and ③ two novel metrics (`secureTy9k` and `vulnerableTy9k`) that measure to what extent an LLM is capable of generating secure code.

The contributions of this paper are:

- A novel framework to *systematically and automatically evaluate the security of LLM generated code*;
- A publicly available dataset of Python prompts<sup>1</sup>;

<sup>1</sup>The dataset will be made public on GitHub upon acceptance.

<https://arxiv.org/abs/2311.00889>





# Implications of LLM code generation

Code reuse goes down

Code velocity goes up

Vulnerability density  
similar

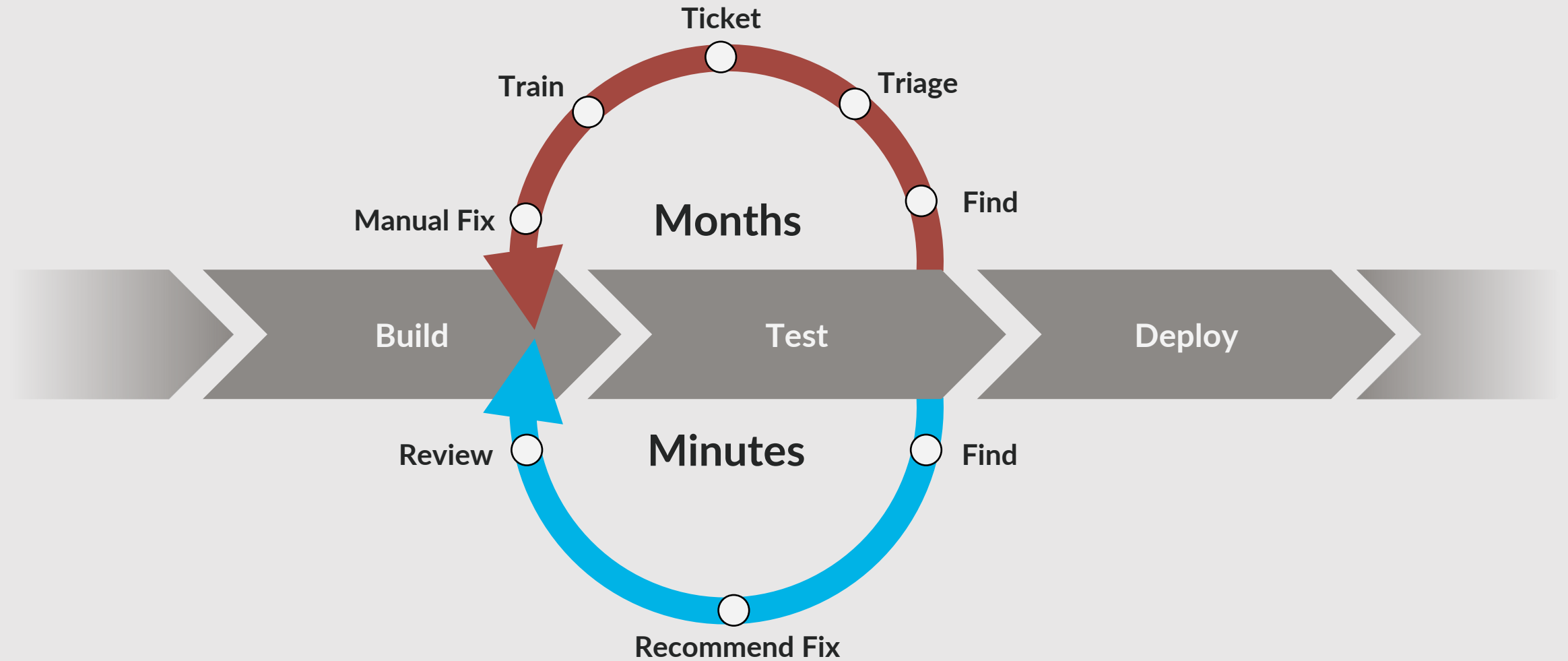
=

Increased Vulnerability  
Velocity

How can we apply AI to the problem of insecure code, but in a more accurate and trustworthy manner?



# We need a faster test and fix workflow



# Training data set: Java XSS

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) {  
    String name = req.getParameter("name");  
    String[] array = new String[10];  
    array[0] = name;  
    PrintWriter writer = resp.getWriter();  
    writer.println("Hello " + array[0]);  
}
```

← Cross-site scripting (CWE 80)

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) {  
    String name = req.getParameter("name");  
    String[] array = new String[10];  
    array[0] = name;  
    PrintWriter writer = resp.getWriter();  
    writer.println("Hello " + StringEscapeUtils.escapeHtml4(array[0]));  
}
```

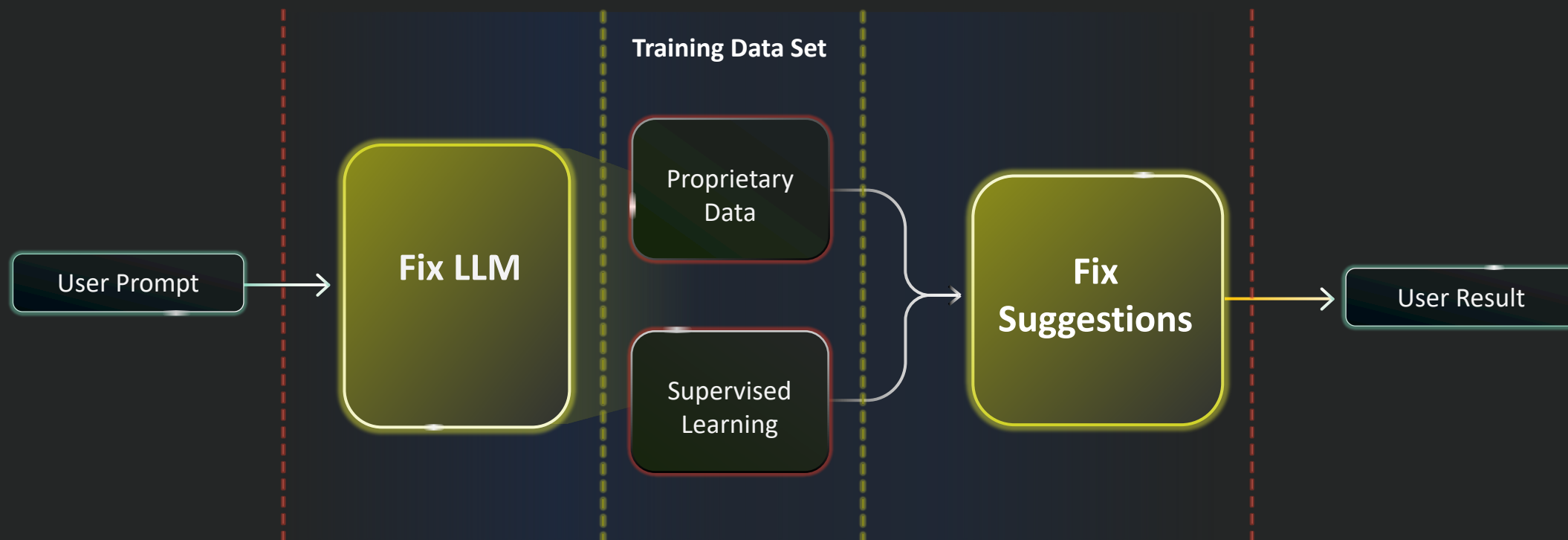


# Fix Approach

Curated Dataset

Code Provenance Assurance

Coverage all that matter



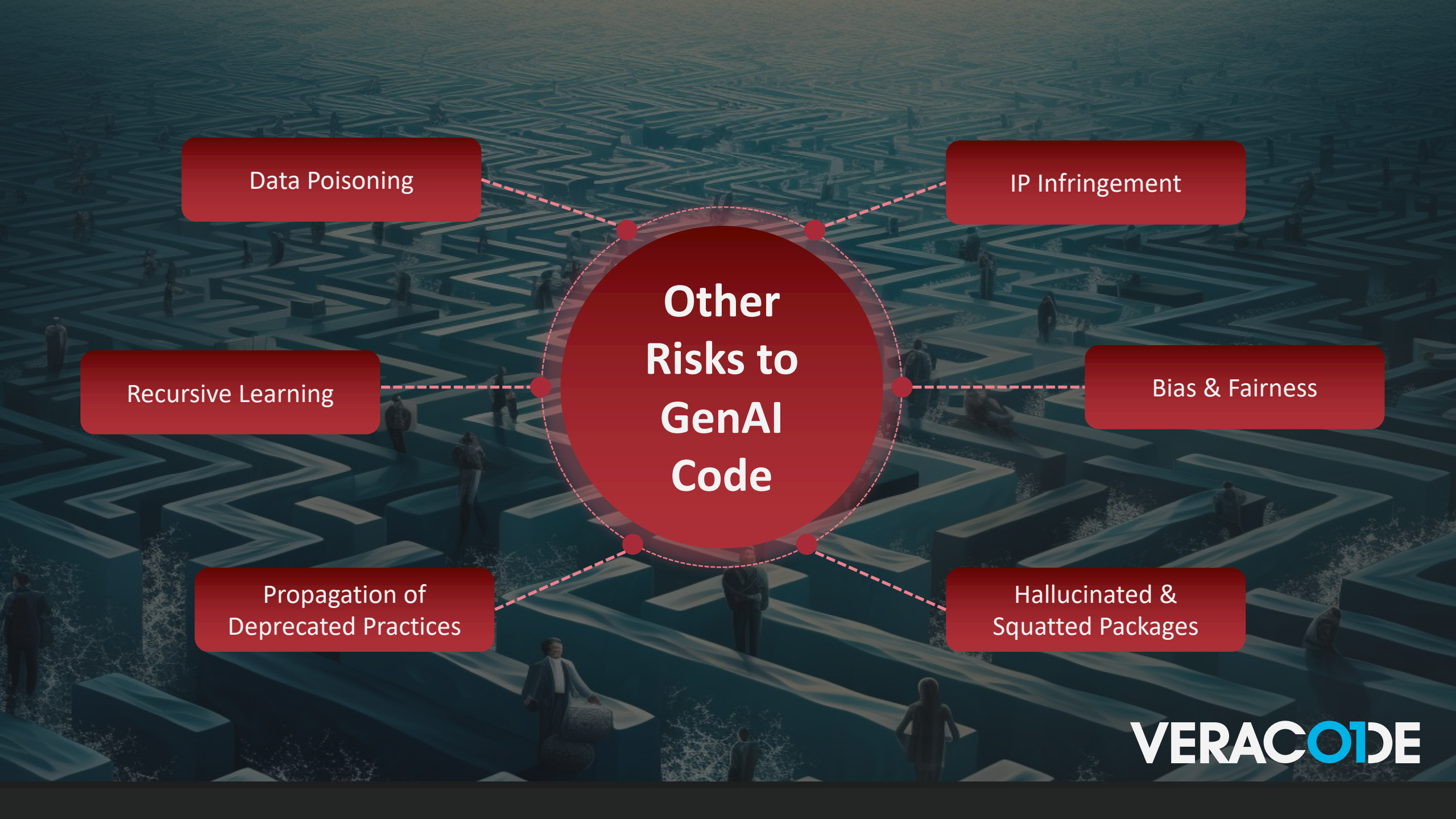
# Recommendations for AI and code security

Consider the implementation details before leveraging AI for developing and/or securing code

- What does the ML model use for training data?
- Is that training data trustworthy/vetted?
- Are there licensing issues with generated code?
- Is any of my intellectual property being leaked?
- How accurate are the generated fixes?

Be aware of human biases that trick us into feeling overly confident about the correctness of AI-generated content





# Other Risks to GenAI Code

Data Poisoning

IP Infringement

Recursive Learning

Bias & Fairness

Propagation of  
Deprecated Practices

Hallucinated &  
Squatted Packages



GenAI in dev is a powerful tool that requires the **same level of security scrutiny and best practices** as any other aspect of software development

Include security considerations in GenAI prompts

Automate as much of security process as possible, including automated fixing

**Chris Wysopal**  
Co-founder & CTO Veracode  
[@weldpond](#)

**VERACODE**

