VERACODE

Trusting AI to Fix Your Code: Genius or Cyber Suicide?

Presented by Patrick Feige

January 29 2025

Who am I?

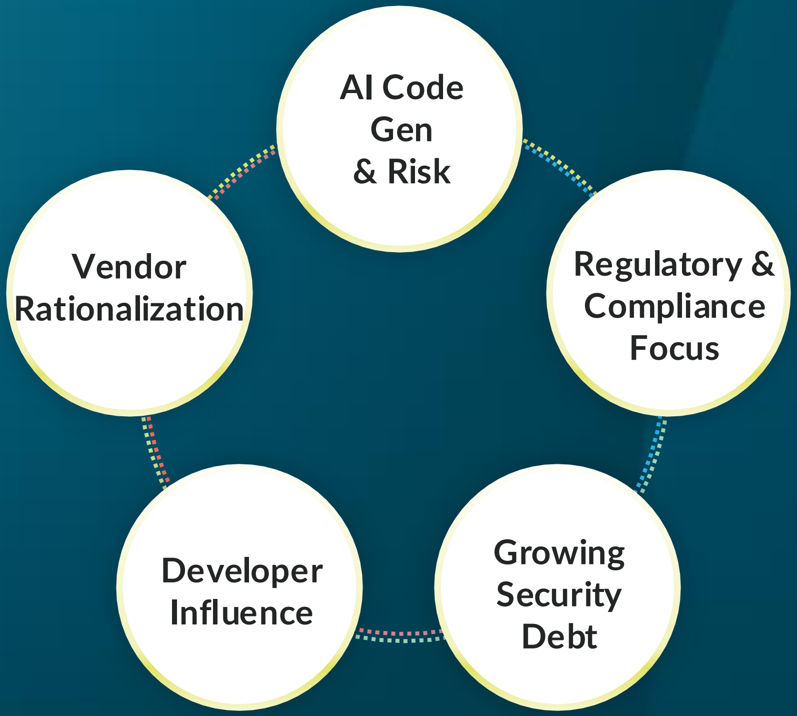
- Senior Solutions Architect @ Veracode
- Former Software Engineer
- Cybersecurity geek, climber, triathlete

Connect with me

Talk Contents

- Who am I?
- State of AI software security space
- Issues/risks with generative AI
- Building an AI product in the appsec space
- What is a vulnerability?
- Future of the appsec space
- Takeaways (and cautionary tales)

State of AI software security space



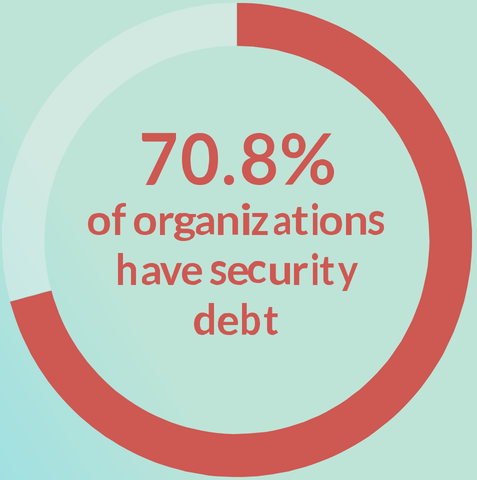
AI Code Gen & Risk

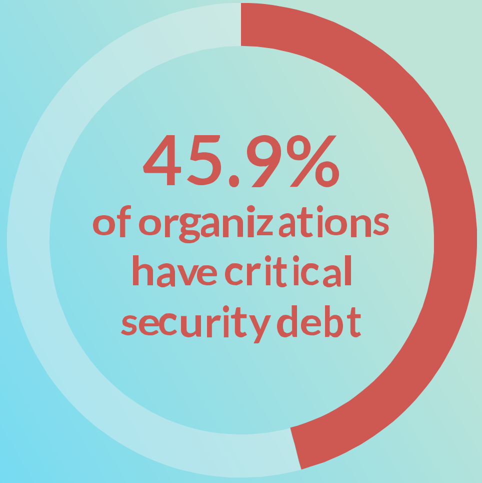
Regulatory & Compliance Focus

Growing Security Debt

Developer Influence

Vendor Rationalization





Organizations are *drowning* in security debt

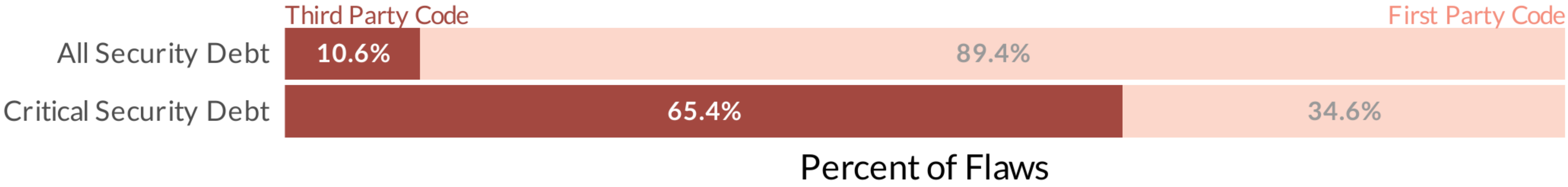
Over 70% of organizations have security debt and nearly half have critical debt.

This represents risk to the business.

Where is the security debt?

While first-party code constitutes almost **90% of all security debt**

65% of critical debt comes from third-party code in open-source libraries



Veracode Scan Research

This research draws from the following:

1,007,133
applications across all scan types

1,553,022
dynamic analysis scans

11,429,365
static analysis scans

All those scans produced:

96.0 million
raw static findings

4.0 million
raw dynamic findings

12.2 million
raw software composition analysis findings

60% of organizations have incorporated AI-assisted development tools into their workflows

48% of DevOps pipelines now include AI-driven automation

79.9% bypass security policies to use AI, but only 10% scan most code

55% AI Contributions to Open-Source Projects

58.7% of appsec teams are struggling to keep up

55.1% of organizations now consider AI-generated code as part of their software supply chain

56.4% of devs commonly encounter security issues in AI code suggestions

Issues/risks with generative AI

OWASP Top 10 - LLM Security

1. Prompt Injection
Manipulating user inputs to alter LLM behavior or access unauthorized data.
Example: Injecting prompts to access private info.

2. Sensitive Info Disclosure
Unintentional exposure of confidential data in model outputs.
Example: Leaking API keys or user credentials.

3. Supply Chain Vulnerabilities
Risks from malicious or unverified third-party components.
Example: Integrating an LLM library with backdoors.

4. Data & Model Poisoning
Manipulating training data to introduce vulnerabilities or alter LLM behavior.
Example: Inserting harmful instructions in data.

5. Improper Output Handling
Lack of validation, leading to harmful or biased outputs.
Example: Generating harmful language.

6. Excessive Agency
Granting LLMs too much autonomy, allowing risky actions.
Example: LLMs executing commands without oversight.

7. System Prompt Leakage
Revealing internal prompts or configurations to users.
Example: Exposing hidden system instructions.

8. Vector & Embedding Weaknesses
Exploiting vector representations or embeddings to manipulate behavior.
Example: Accessing sensitive information in embeddings.

9. Misinformation
LLMs generating incorrect or biased content, leading to harm.
Example: Providing false medical advice.

10. Unbounded Consumption
Exploiting models to consume excessive resources, causing service disruptions.
Example: Generating excessively long outputs.

Building an AI product in the appsec space

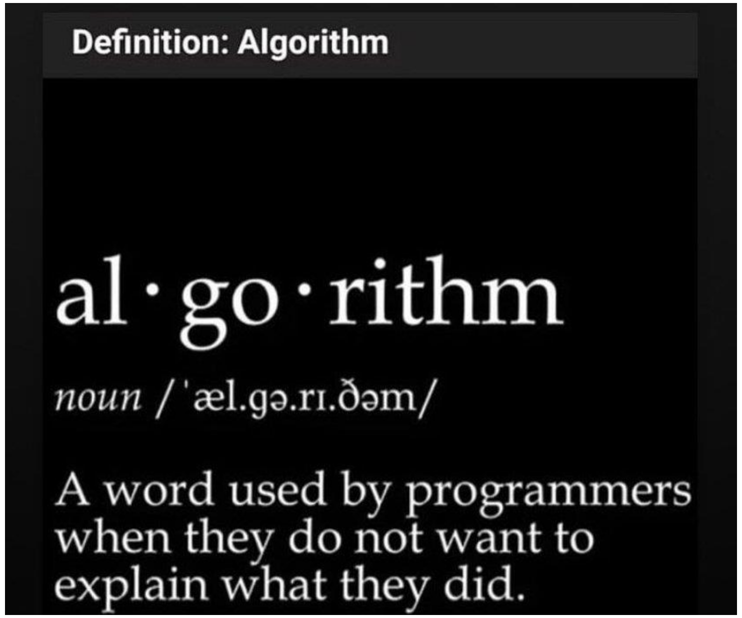
Motivation

We have static scan products?

- Powerful analysis algorithms
- Coverage of lots of languages and frameworks
- Careful modeling of sources, sinks, and propagators
- Trillions of lines of code scanned

We give customers lots of great information

But…



Definition: Algorithm

al·go·rithm

noun /ˈæl.gə.rɪ.ðəm/

A word used by programmers when they do not want to explain what they did.

Motivation

Problem: now what?

- We tell customers exactly why their code is broken
- Still up to them to **fix** it!
- Specialized knowledge: how to fix a flaw
- Lots of (not fun) work: making security fixes

As a result: a lot of **security debt**

Solution

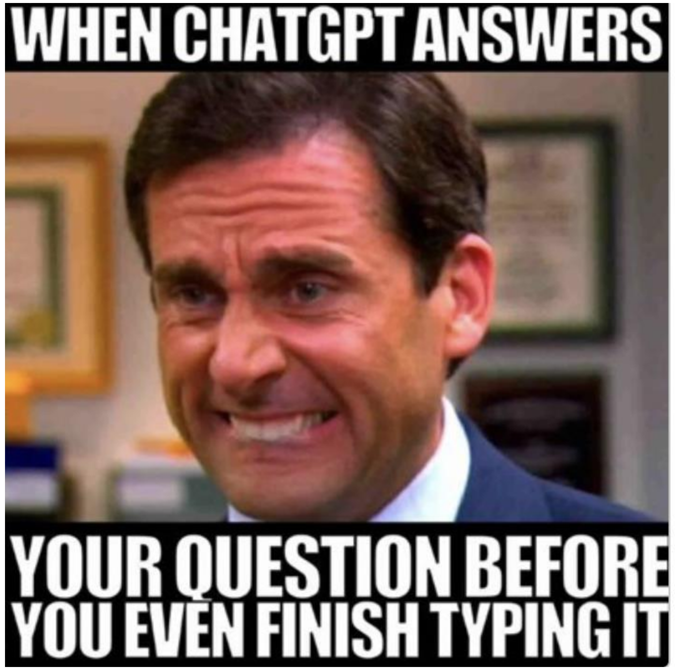
Automatically fix the flaws!

- Fixes are well-known
- E.g., sanitize outputs for XSS, used prepared statement for SQL injection, etc.
- Mostly straightforward code modifications

But how?

Let’s use AI!

Idea: Use AI to map bad code to good code



Challenges:

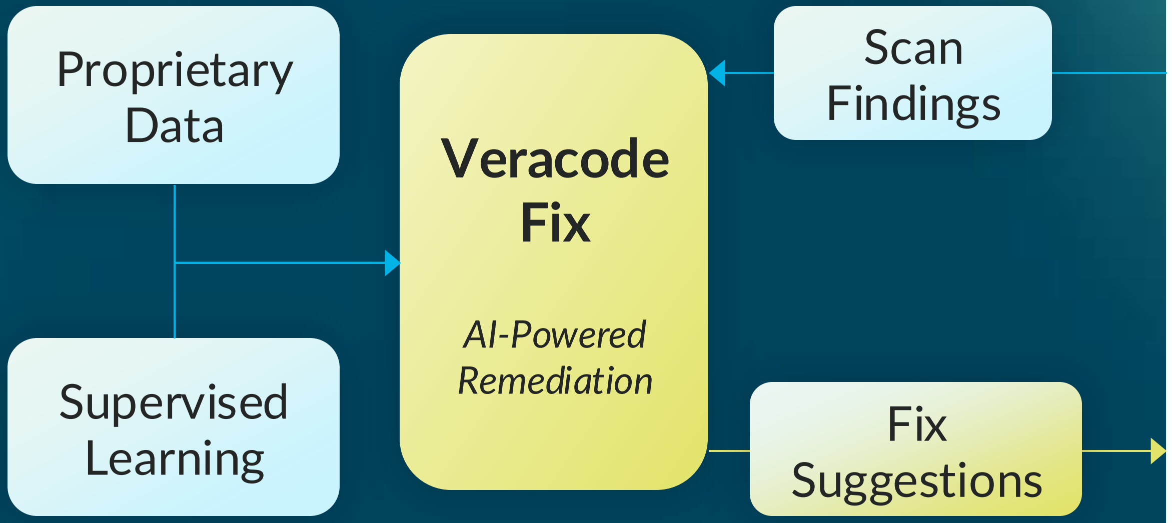
- How to represent code
- How to train the AI to fix code
- Do we need identify the location and nature of the flaw?
- How to make sure fixes are good (*syntactically correct*, *semantically correct*)
- How to generalize for many different languages and CWEs

Veracode Fix timeline

- 2020: Incubator project and early research looking at auto remediation
  - Code rewriting **techniques**: too much manual work to write the AST fix configs for each CWE-language pair (easily 1000s of rules need to be written)
  - GPT Model: GPT-2 was chosen before we acquired Jaroona, the model is open source,

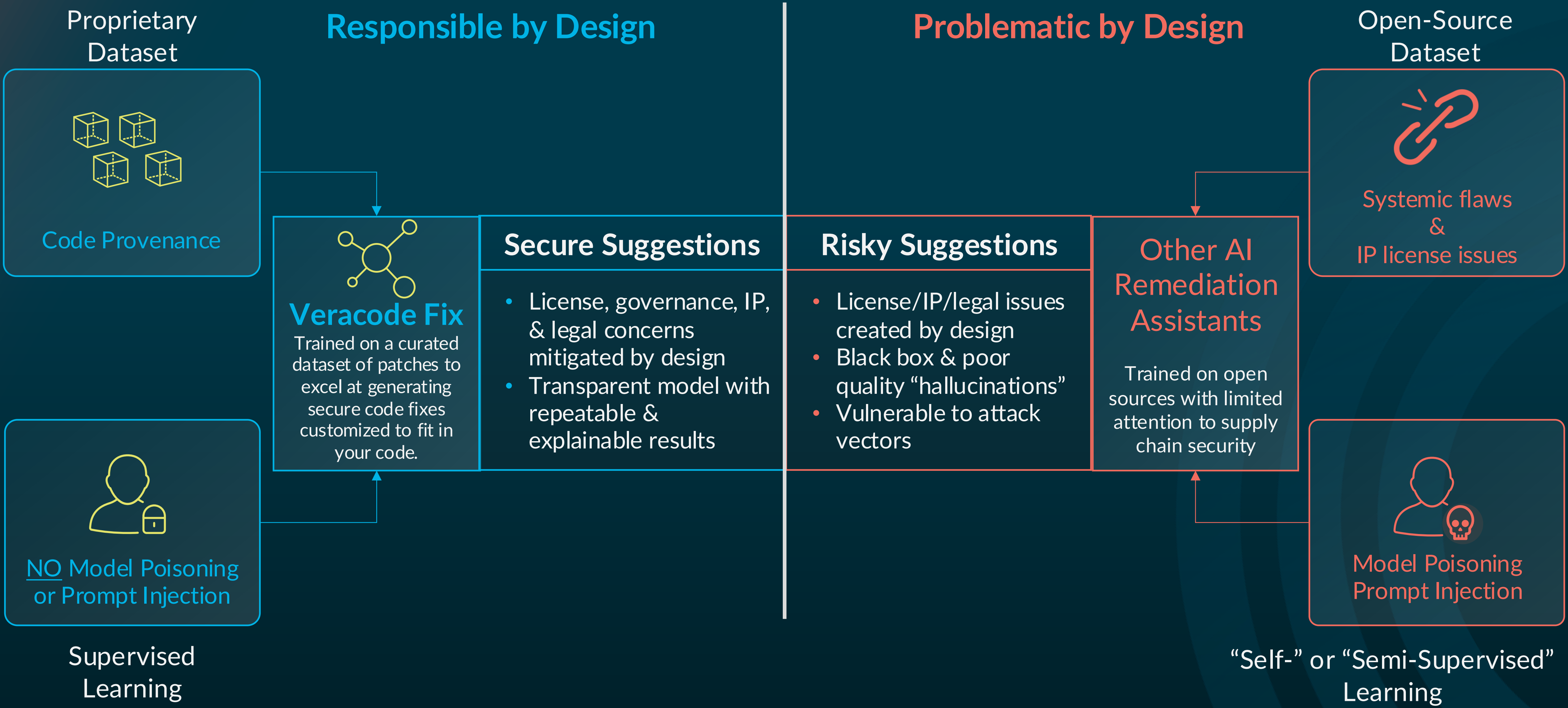
  so we can use it however we want. Later models, like GPT-3 and Codex, are only available through APIs.
  - Narrow application of GPT: only do one very specific thing, suggest security fixes

- 2022 Veracode acquires Jaroona:
  - leverage general reference patches and then “intelligently” figures out how translate it to the customer’code situation, rather than us manually writing every edge case



Veracode Fix

- Fix is an AI-assisted remediation solution that addresses individual findings or whole categories at once
- Supports Java, C#, JS/TS, Kotlin, Scala, Python, and PHP
- Fix Available via Veracode CLI and Veracode IDE plugins



Responsible by Design

Proprietary Dataset

Code Provenance

Supervised Learning

NO Model Poisoning or Prompt Injection

Veracode Fix

Trained on a curated dataset of patches to excel at generating secure code fixes customized to fit in your code.

Secure Suggestions

- License, governance, IP, & legal concerns mitigated by design
- Transparent model with repeatable & explainable results

Problematic by Design

Open-Source Dataset

Systemic flaws & IP license issues

"Self-" or "Semi-Supervised" Learning

Model Poisoning Prompt Injection

Other AI Remediation Assistants

Trained on open sources with limited attention to supply chain security

Risky Suggestions

- License/IP/legal issues created by design
- Black box & poor quality "hallucinations"
- Vulnerable to attack vectors

VERACODE

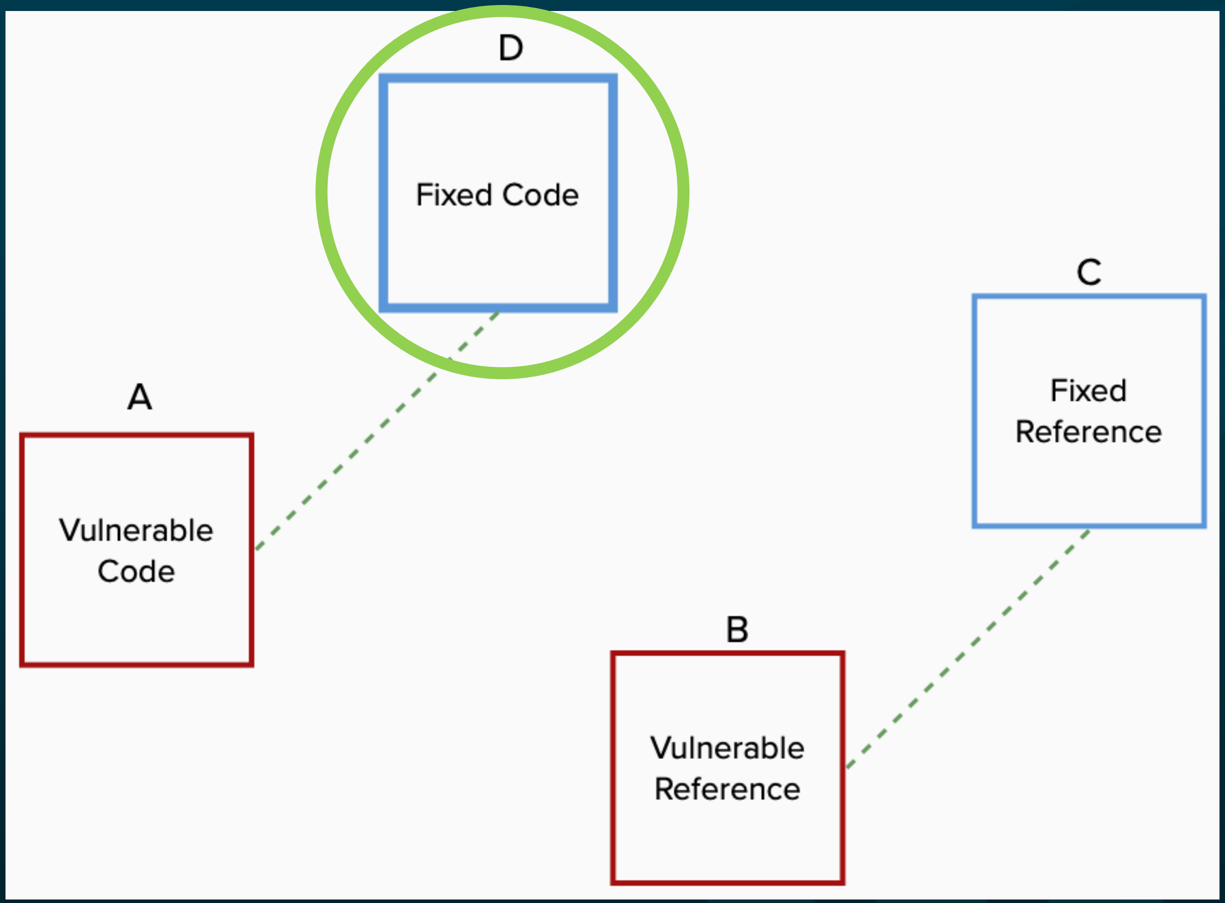
Fix Training

Fix uses a GPT Model that is trained by providing elements of data (ABCD) that allows it to generate remediation guidance as a patch that replaces vulnerable code.

$$D=A+(C-B)$$

During training, the model is given all 4 elements so it can learn to work backwards from vulnerable code to end up at Fixed Code.

Fix is trained on known good code samples, using Veracode created reference patches.



What is a vulnerability?

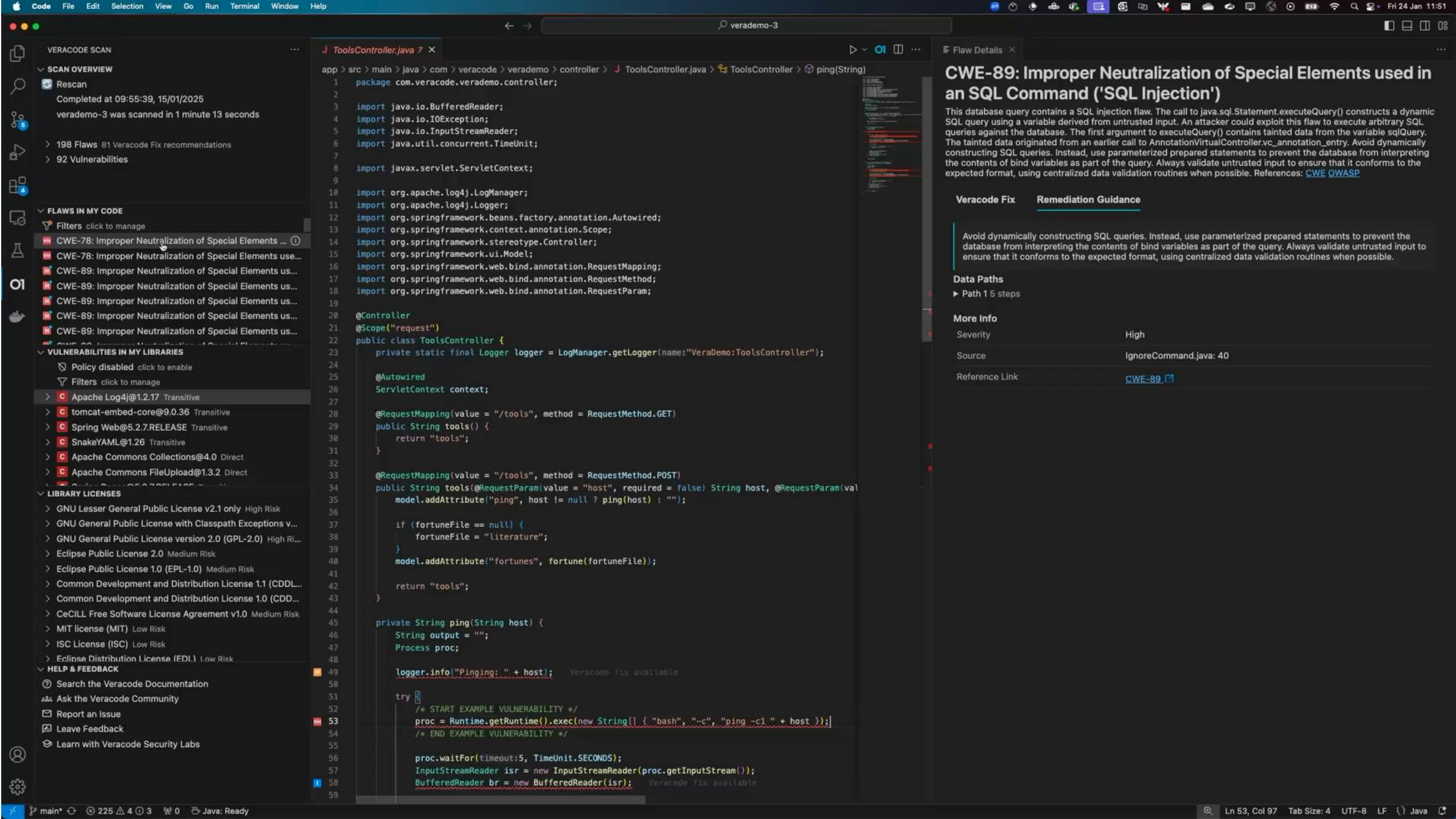
Example

```
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 “ + HtmlUtil.escape(array[0]));
}
```

Geeky stuff: https://docs.veracode.com/r/review_cleansers


Code File Edit Selection View Go Run Terminal Window Help
Fri 24 Jan 11:51
verademo-3
ToolsController.java 7
Flaw Details
main* 225 4 3 0 Java: Ready
Ln 53, Col 97 Tab Size: 4 UTF-8 LF Java

VERACODE SCAN

SCAN OVERVIEW

Rescan

Completed at 09:55:39, 15/01/2025

verademo-3 was scanned in 1 minute 13 seconds

198 Flaws 81 Veracode Fix recommendations

92 Vulnerabilities

FLAWS IN MY CODE

Filters click to manage

CWE-78: Improper Neutralization of Special Elements ...

CWE-78: Improper Neutralization of Special Elements use...

CWE-89: Improper Neutralization of Special Elements us...

CWE-89: Improper Neutralization of Special Elements us...

CWE-89: Improper Neutralization of Special Elements us...

CWE-89: Improper Neutralization of Special Elements us...

CWE-89: Improper Neutralization of Special Elements us...

VULNERABILITIES IN MY LIBRARIES

Policy disabled click to enable

Filters click to manage

C	Apache Log4j@1.2.17	Transitive
C	tomcat-embed-core@9.0.36	Transitive
C	Spring Web@5.2.7.RELEASE	Transitive
C	SnakeYAML@1.26	Transitive
C	Apache Commons Collections@4.0	Direct
C	Apache Commons FileUpload@1.3.2	Direct

LIBRARY LICENSES

GNU Lesser General Public License v2.1 only	High Risk
GNU General Public License with Classpath Exceptions v...	
GNU General Public License version 2.0 (GPL-2.0)	High Ri...
Eclipse Public License 2.0	Medium Risk
Eclipse Public License 1.0 (EPL-1.0)	Medium Risk
Common Development and Distribution License 1.1 (CDDL...	
Common Development and Distribution License 1.0 (CDD...	
CeCILL Free Software License Agreement v1.0	Medium Risk
MIT license (MIT)	Low Risk
ISC License (ISC)	Low Risk
Eclipse Distribution License (EDL)	Low Risk

HELP & FEEDBACK

Search the Veracode Documentation

Ask the Veracode Community

Report an Issue

Leave Feedback

Learn with Veracode Security Labs

app > src > main > java > com > veracode > verademo > controller > ToolsController.java > ToolsController > ping(String)

```java
package com.veracode.verademo.controller;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.concurrent.TimeUnit;

import javax.servlet.ServletContext;

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
@Scope("request")
public class ToolsController {
    private static final Logger logger = LogManager.getLogger(name:"VeraDemo:ToolsController");

    @Autowired
    ServletContext context;

    @RequestMapping(value = "/tools", method = RequestMethod.GET)
    public String tools() {
        return "tools";
    }

    @RequestMapping(value = "/tools", method = RequestMethod.POST)
    public String tools(@RequestParam(value = "host", required = false) String host, @RequestParam(val
        model.addAttribute("ping", host != null ? ping(host) : "");

        if (fortuneFile == null) {
            fortuneFile = "literature";
        }
        model.addAttribute("fortunes", fortune(fortuneFile));

        return "tools";
    }

    private String ping(String host) {
        String output = "";
        Process proc;

        logger.info("Pinging: " + host);   Veracode fix available

        try {
            /* START EXAMPLE VULNERABILITY */
            proc = Runtime.getRuntime().exec(new String[] { "bash", "-c", "ping -c1 " + host });
            /* END EXAMPLE VULNERABILITY */

            proc.waitFor(timeout:5, TimeUnit.SECONDS);
            InputStreamReader isr = new InputStreamReader(proc.getInputStream());
            BufferedReader br = new BufferedReader(isr);   Veracode fix available
```

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

This database query contains a SQL injection flaw. The call to java.sql.Statement.executeQuery() constructs a dynamic SQL query using a variable derived from untrusted input. An attacker could exploit this flaw to execute arbitrary SQL queries against the database. The first argument to executeQuery() contains tainted data from the variable sqlQuery. The tainted data originated from an earlier call to AnnotationVirtualController.vc_annotation_entry. Avoid dynamically constructing SQL queries. Instead, use parameterized prepared statements to prevent the database from interpreting the contents of bind variables as part of the query. Always validate untrusted input to ensure that it conforms to the expected format, using centralized data validation routines when possible. References: CWE OWASP

Veracode Fix | Remediation Guidance

> Avoid dynamically constructing SQL queries. Instead, use parameterized prepared statements to prevent the database from interpreting the contents of bind variables as part of the query. Always validate untrusted input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

Data Paths

▸ Path 1 5 steps

More Info

Severity	High
Source	IgnoreCommand.java: 40
Reference Link	CWE-89

Future of the appsec space

Shift Left Security
Integrating security earlier in the SDLC with tools like SAST and SCA.
Source: OWASP ("OWASP DevSecOps Guideline - v-0.2")

Cloud-Native Security
Focus on securing microservices, containers, and serverless architectures.
Source: Altice Labs ("Security guidelines applied to microservices cloud architectures")

AI/ML-Driven Security
Leveraging AI and ML for faster, more accurate vulnerability detection and threat analysis.
Source: Palo Alto Networks ("What Is the Role of AI in Threat Detection?")

Supply Chain Security
Strengthening third-party risk management and securing software dependencies.
Source: MITRE (”Securing Critical Software Supply Chains")

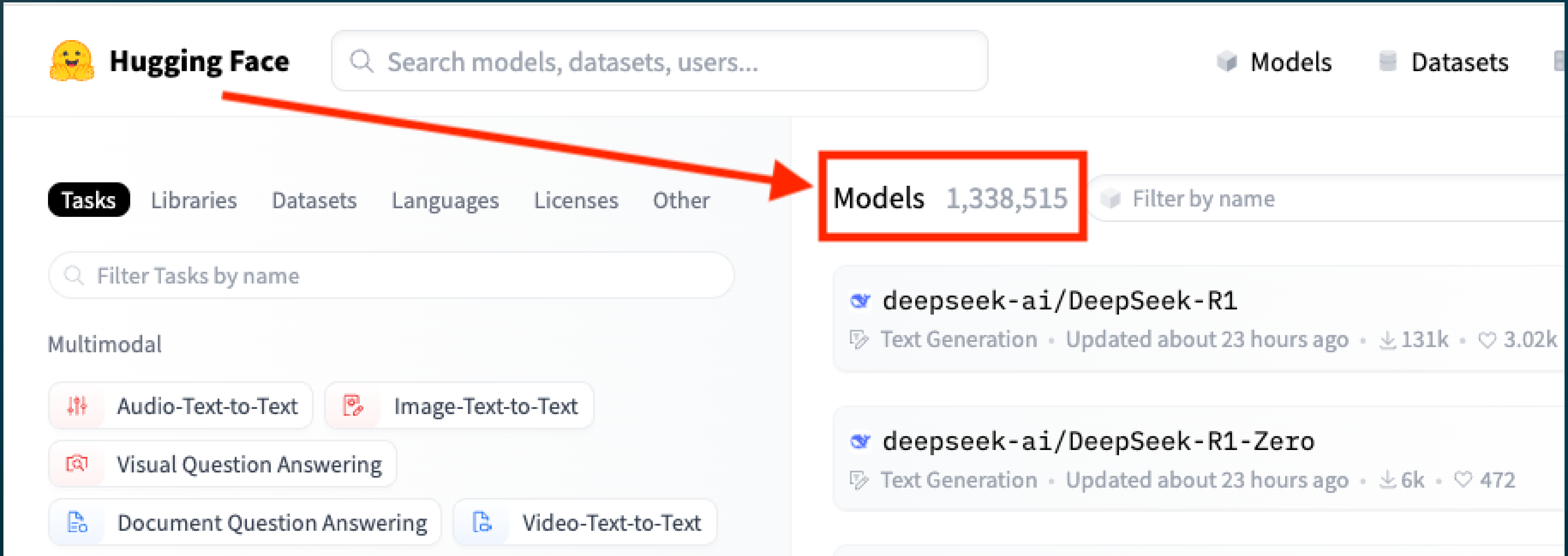
Zero Trust Models
Adopting Zero Trust architecture for continuous identity verification and least-privilege access.
Source: NIST ("Zero Trust Architecture")

Takeaways (and cautionary tales)

- Barrier for entry is much lower for generating cyber attacks, scams, impersonations
- We need to be the voice of pushing for guard rails and implementing security practices for these

- Cyber attacks have included:
  - Power Grids: Ukraine Power Grid Attack (2015-2016) - Russian-backed hackers caused power outages.
  - Nuclear Facilities: Stuxnet (2010) - Targeted Iran’s Natanz facility, damaging nuclear centrifuges.
  - ATMs: Carbanak (2013-2018) - APT Malware attack on banking networks, enabling ATM cash theft.
  - Pacemakers: Pacemaker Hacking Vulnerabilities (2017) - Exploited Medtronic pacemaker flaws to cause shocks or battery drain.
  - Cars: Jeep Cherokee Hack (2015) - Remote control of vehicle functions via infotainment system.

- Attackers vs Defenders



Hugging Face

Search models, datasets, users...

Models

Datasets

Tasks

Libraries

Datasets

Languages

Licenses

Other

Filter Tasks by name

Multimodal

Audio-Text-to-Text

Image-Text-to-Text

Visual Question Answering

Document Question Answering

Video-Text-to-Text

Models 1,338,515

Filter by name

deepseek-ai/DeepSeek-R1

Text Generation • Updated about 23 hours ago • 131k • 3.02k

deepseek-ai/DeepSeek-R1-Zero

Text Generation • Updated about 23 hours ago • 6k • 472

- Hugging Face, the primary online repository for generative AI, has hosted thousands of files containing hidden code – Forbes
- Unrestricted GPT's – FraudGPT and WormGPT

VERACODE

Sources

- OWASP Top 10 LLM's: Explore the OWASP Top 10 risks for LLMs at (OWASP GenAI).
- Veracode SOSS Report: Check out the Veracode State of Software Security 2024 report at this (link).
- AI Code Contribution Increase: In 2023, 55% of developers reported using AI tools for code generation in their projects, up from 35% in 2022. This sharp increase highlights the growing reliance on AI in development workflows (Stack Overflow Developer Survey).
- AI Adoption for Code Automation: "83.2% of developers reported using AI code completion tools for open-source projects in 2023, a significant rise from 60% in 2022." This shows a clear trend towards integrating AI tools into the software development process, especially in open-source contributions (Snyk's AI-Generated Code Security Report).
- AI's Role in Software Supply Chains: "55.1% of organizations now consider AI-generated code as part of their software supply chain, a 15% increase from 2022." This suggests that companies are increasingly viewing AI contributions as integral to their development pipelines (Snyk's AI-Generated Code Security Report).
- AI-Generated Code Volume: AI tools were responsible for generating 30% of new codebases in 2023, compared to 20% in 2022. This 50% year-over-year growth reflects how developers are increasingly leveraging AI to accelerate coding processes (AI Index Report 2023).
- Increased Use of AI for Code Completion: "In 2023, 68% of developers used AI-driven code completion tools like GitHub Copilot, up from 45% in 2022. This 23% increase demonstrates the rapid integration of AI in daily coding tasks (Stack Overflow Developer Survey)".
- Percentage of AI-Generated Code in New Projects: "By mid-2023, approximately 40% of code in new software projects was generated or influenced by AI tools, a 15% increase from 2022." This rise is due to the growing reliance on AI for boilerplate code and complex function generation (AI Index Report 2023).
- AI Adoption Across Organizations: "Over 60% of organizations reported using AI-assisted development tools for at least one major project in 2023, compared to 38% in 2022." This shift indicates an industry-wide acceptance of AI as a productivity enhancer (Snyk's AI Adoption Report).
- AI-Driven Testing Automation: "In 2023, 52% of application testing processes were automated using AI, compared to 34% in 2022. This trend reflects the need for faster and more efficient testing to match the pace of AI-generated development (NTT Data's AI Impact Study)".
- AI Influence on Code Review Practices: "46% of development teams now use AI tools for code review in 2023, up from 28% in 2022. AI is increasingly utilized to spot common coding errors and optimize performance during review cycles (AI Index Report 2023)".
- Shift Towards AI-Generated Documentation: "In 2023, 35% of code documentation was produced with AI assistance, up from 20% in 2022. Developers rely on AI to auto-generate and maintain accurate documentation alongside code updates (Stack Overflow Developer Survey)".
- AI's Role in Open-Source Contributions: "In 2023, AI tools contributed to 55% of all code changes in open-source projects, a rise from 38% in 2022. This shows the growing acceptance and utility of AI in collaborative coding environments (Snyk's Developer Security Report)".
- Growth in AI-Assisted Refactoring: "Over 45% of developers reported using AI tools for code refactoring in 2023, compared to 27% in 2022, reflecting a push for optimizing legacy codebases using AI efficiency (AI Index Report 2023)".
- Increased Deployment of AI-Enhanced IDEs: "64% of developers now use integrated development environments (IDEs) with AI enhancements for debugging and optimization, a 20% increase from 2022 (Stack Overflow Developer Survey).
- AI's Impact on Software Supply Chains: "AI now influences over 50% of software supply chains, as reported by organizations in 2023, up from 35% in 2022. This includes automated dependency management and security patching (NTT Data's AI Security Report).
- Rise of AI in DevOps Automation: "In 2023, 48% of DevOps pipelines included AI-driven automation for CI/CD processes, a significant increase from 30% in 2022, showcasing AI's growing role in continuous deployment and integration (Snyk's AI Adoption and Impact Report)".
- Shift Left Security: Integrating security earlier in the SDLC with tools like SAST and SCA. "OWASP DevSecOps Guideline - v-0.2 (https://owasp.org/www-project-devsecops-guideline/latest/00a-Overview)".
- Cloud-Native Security: Focusing on securing microservices, containers, and serverless architectures. Altice Labs "Security guidelines applied to microservices cloud architectures (https://www.alticelabs.com/wp-content/uploads/2024/06/whitepaper_Security-guidelines-applied-to-microservices-cloud-architectures.pdf)".
- AI/ML-Driven Security: Leveraging AI and ML for faster, more accurate vulnerability detection and threat analysis. Palo Alto Networks, "What Is the Role of AI in Threat Detection? (https://www.paloaltonetworks.co.uk/cyberpedia/ai-in-threat-detection)".
- Supply Chain Security: Strengthening third-party risk management and securing software dependencies. MITRE, "Securing Critical Software Supply Chains (https://www.mitre.org/sites/default/files/2021-11/prs-21-0278-deliver-uncompromised-securing-critical-software-supply-chain.pdf)".
- Zero Trust Models: Adopting Zero Trust architecture for continuous identity verification and least-privilege access NIST, "Zero Trust Architecture (https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-207.pdf)".
- Generative AI Risks: Hugging Face, the primary online repository for generative AI, has hosted thousands of files containing hidden code. Forbes (https://www.forbes.com/sites/iainmartin/2024/10/22/hackers-have-uploaded-thousands-of-malicious-models-to-ais-biggest-online-repository/).