



DOI:10.1145/3704724

## Unveiling the dark side.

BY ADITYA K. SOOD AND SHERALI ZEADALLY

# Malicious AI Models Undermine Software Supply-Chain Security

INTEGRATING MALICIOUS AI models<sup>6</sup> into software supply chains presents a significant and emerging threat to cybersecurity. The attackers aim to embed malicious AI models in software components and widely used tools, thereby infiltrating systems at a foundational level. Once integrated, the malicious AI models execute embedded unauthorized code, which performs actions such as exfiltrating sensitive data, manipulating data integrity, or enabling unauthorized access to critical systems. Compromised development tools, tampered libraries, and pre-trained models are

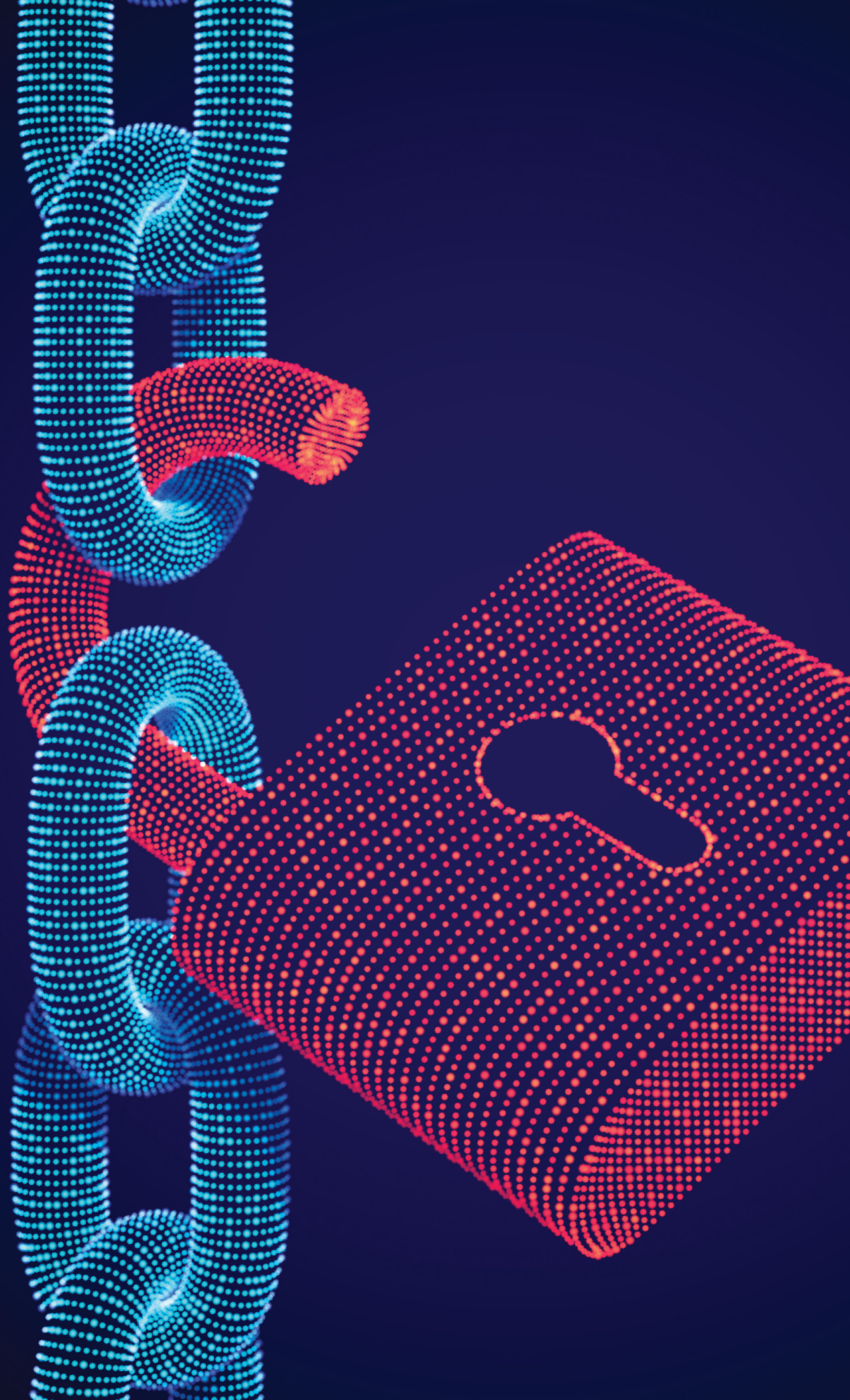
the primary methods of introducing malicious AI models into the software supply chain. Developers often rely on libraries and frameworks to import pre-trained AI models to expedite software development. Attackers can easily compromise the underlying system if the application consumes pre-trained AI models tainted with malicious code. For instance, the MLOps platform<sup>20</sup> fails to detect malicious AI models after scanning the model file containing serialized data. Deploying a malicious AI model would jeopardize the security of the systems and could put end users at risk in some applications.

The sophistication of these malicious AI models allows them to adapt and evade traditional security measures by leveraging their ability to analyze vast amounts of data, learn complex patterns, and generate responses that mimic human behavior. These malicious AI models can simulate diverse attack scenarios, discover previously unknown vulnerabilities, and create evasive techniques that traditional defenses, reliant on predefined signatures and heuristics, struggle to detect. It is also important to note that adversaries can exploit vulnerabilities in the AI framework or the environ-

### » key insights

- **Attackers inject malicious code into AI models hosted on the public repositories. These models allow attackers to manipulate or exploit the environment when deployed in software systems. Incorporating malicious AI models in dependencies or libraries also compromises the integrity of software products downstream.**
- **Malicious AI models are distributed to compromise the software supply chain and trigger infections on a large scale. The absence of rigorous testing or verification processes for AI models allows adversaries to inject malicious functionality into them.**
- **Organizations need robust processes to validate the origin and integrity of AI models. Organizations should use trusted repositories, cryptographic validation, and controlled access to mitigate risks associated with third-party AI models.**

IMAGE BY MAKSYM KABAKOU





ment when the AI model is deployed, such as weaknesses in deserialization processes or insufficient validation checks. This underscores the need for constant vigilance and proactive measures to secure AI systems. Integration of AI models without adequate security checks can lead to the execution of unauthorized code. AI models, especially those shared via untrusted sources, may contain embedded malicious code or payloads that can be triggered during execution. Attackers can tamper with the model weights, scripts, or dependencies to insert hidden malware or back doors, allowing for unauthorized code execution once the model is loaded into memory. The absence of robust security checks, sandboxing, or code-integrity verification can lead to arbitrary code execution, underscoring the critical need for stringent security measures.

Detecting and mitigating malicious AI models is challenging due to their complexity and opacity. First, AI models are distributed as opaque files (for example, model weights) containing parameters learned during training. As these parameters can be tampered with, AI models act maliciously when specific triggers are activated, making it challenging to distinguish between a benign and compromised model without deep analysis. Second, AI models rely on external dependencies and frameworks, and attackers can exploit vulnerabilities in these libraries to introduce malicious payloads. Third, AI models are deployed in environments that lack proper validation or isolation mechanisms, allowing malicious code hidden within a model to execute undetected. Several existing solutions aim to address these issues, such as secure model validation to check the integrity of models before they are deployed. Another key solution is to use a secure model serialization format, such as Safetensors,<sup>16</sup> for AI models. This format provides a robust way to verify the embedded code, offering a sense of reassurance about the safety of the AI models. In addition, model watermarking involves embedding models with unique identifiers to verify their source and integrity. Sandboxing models in isolated execution environments is another technique that mitigates the risk of malicious code

execution by containing any harmful behavior within a controlled space. Despite these solutions, the constantly evolving nature of AI-based threats makes it essential for organizations to continuously adapt and integrate multiple layers of security by building a holistic approach to security to defend against malicious AI models. Overall, malicious AI models are programmed to recognize and adapt to different security environments. They can identify patterns in security protocols and learn to avoid detection by mimicking normal system behavior. This adaptability makes them particularly dangerous because they can remain dormant and undetected for extended periods, only activating under specific conditions to execute their malicious payload. This stealthy nature of attacks using AI complicates incident response efforts because traditional detection methods may not be effective against such advanced cyber threats.

**Research contributions of this work.** Considering the above, this article makes the following contributions. First, we describe the AI model integration process and discuss threat classification and associated attack payloads. Second, we present a complete attack flow model explaining how attackers use malicious AI models to compromise the target systems. Third, we discuss the limitation of traditional security defenses to restrict the impact of malicious AI models. Fourth, we discuss recommendations with granularity, which organizations can opt to strengthen software supply-chain security.

### Understanding the AI Model Integration Process

Hosting AI models in a repository is a crucial aspect of modern AI development and deployment.<sup>17</sup> The repositories hosting AI models are centralized locations where models, metadata, dependencies, and documentation are stored securely, facilitating easy access and collaboration. The AI models deployed in the repositories are often open source, allowing users to directly deploy them in the production environment or fine-tune those AI models for specific use cases. By hosting AI models in a repository, organizations can ensure consistency across

different stages of development, from training to maintenance control over model versions and updates. Repositories such as Hugging Face, TensorFlow Hub, PyTorch Hub, and Model Zoo offer additional features over traditional model-hosting and version-control systems such as GitHub. These features include model discovery; built-in tools for model evaluation, including model training, integrated inference application programming interfaces (APIs), private model hosting, sharing pre-trained models; and integration with various machine learning (ML) frameworks.

The AI model integration process involves transferring a pre-trained model from an external source or repository into a local development environment or application for further use or fine-tuning. This process typically starts by selecting the appropriate model from a repository, which provides models in various formats. Once selected, the model is downloaded and loaded into the chosen ML framework, such as TensorFlow or PyTorch, using standardized functions or APIs. During integration, it is crucial to ensure that the model's architecture, dependencies, and input-output configurations are compatible with the existing environment. The model may undergo additional steps, such as fine-tuning domain-specific data, integration into a larger system, or deployment in a production environment. Understanding the complete AI model integration process is essential to dissect the associated risks and threats. Table 1 presents the workflow of integrating the AI model.

By following the steps in Table 1, users can effectively import pre-trained AI models from repositories for development.

### Threat Modeling: Malicious AI Models

Threat modeling<sup>10</sup> offers a structured framework for understanding threats. The primary importance of threat modeling lies in its proactive nature, allowing organizations to anticipate and mitigate risks before malicious actors can exploit them. The goal is to develop a systematic approach for identifying, assessing, and addressing potential security threats to applica-

tions, systems, or networks. This section discusses malicious AI models, threat classification, attack payloads, and attack flow models specifically for malicious AI models impacting software supply chains.

**Understanding malicious AI models.** A malicious AI model is designed or tampered with to cause harm or act against the intended purpose of the system it's integrated into. While many might assume that an AI model is simply a tool for prediction or classification, like predicting sentiment in a text, malicious actors can exploit these models in various ways to compromise security, privacy, and integrity. Several examples of malicious AI models are listed below.

*Embedding malicious code in model artifacts.* To speed up development, a user downloads a pre-trained model from an online repository. The model file is embedded with malicious code that executes when the model is loaded, taking advantage of the loading process to infect the system. This could compromise the system, giving attackers control over the host machine, enabling data exfiltration, or spreading the malware further. In real-world scenarios, Python pickle files have been used to embed malicious code within AI models, exploiting the deserialization process in Python. Pickle is a Python-specific serialization format for AI models. Still, the inherent security flaw for trusting the deserialized data allows the execution of embedded code, creating a serious security risk. Researchers demonstrated how a malicious pickle file could execute arbitrary code during deserialization using the sleepy pickle attack technique.<sup>22,23</sup> Listing 1 highlights the basic code of a pickle file embedded with malicious code.

In the code presented in the listing, the `pickle.loads()` function is called, and the embedded malicious command (`os.system("rm -rf/")`) is executed, causing catastrophic damage, such as deleting files on the system. When an AI model is executed using the malicious pickle file, the embedded code runs in that environment without the user's knowledge. In other more prominent attack scenarios, attackers can distribute pre-trained AI models and users might unknowingly download

**Table 1. Basic workflow for integrating an AI model from the repository.**

| Integration Steps         | Description   |
|---------------------------|---|
| Choose repository         | Select a repository or platform where the desired pre-trained model is hosted. Standard repositories include: <ul style="list-style-type: none"> <li>► GitHub: A platform for hosting open source projects, including AI models.</li> <li>► Hugging Face Model Hub: A repository for hosting and sharing pre-trained natural language processing models.</li> <li>► TensorFlow Hub: A platform for sharing AI models compatible with the TensorFlow framework.</li> <li>► PyTorch Hub: A repository for sharing AI models compatible with the PyTorch framework.</li> </ul>   |
| Identify model            | Select the specific model to import from the repository. Hosted models are organized by task, architecture, or domain.  |
| Install required packages | Install the necessary libraries or packages to interact with the repository and download the model. The list below includes widely used packages or dependencies. <ul style="list-style-type: none"> <li>► Hugging Face Model Hub: Install the <i>transformers</i> library using pip (<i>pip install transformers</i>).</li> <li>► TensorFlow Hub: Install TensorFlow and TensorFlow Hub (<i>pip install tensorflow tensorflow-hub</i>).</li> <li>► PyTorch Hub: Install PyTorch (<i>pip install torch</i>).</li> </ul>   |
| Import model              | Import the desired model into one's Python environment using the appropriate library or framework. <ul style="list-style-type: none"> <li>► Hugging Face Model Hub (using <i>transformers</i> library):<br/> <pre>from transformers import AutoModelForSequenceClassification model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-uncased")</pre> </li> <li>► Tensor Hub:<br/> <pre>import tensorflow_hub as hub model = hub.load("https://tinyurl.com/22ts9aop")</pre> </li> <li>► PyTorch Hub:<br/> <pre>import torch model = torch.hub.load('pytorch/vision', 'resnet18', pretrained=True)</pre> </li> </ul> |
| Verify compatibility      | <ul style="list-style-type: none"> <li>► Verify that the model's input and output formats match the data pipeline. If necessary, adjust preprocessing steps to ensure compatibility.</li> <li>► Ensure that the local framework and hardware (for example, CPU vs. GPU) support the model's layers, activation functions, and other components.</li> </ul>  |
| Customize and fine-tune   | ► Adapt the model to a specific dataset using fine-tuning by continuing training on the data. Modify hyperparameters such as learning rate, batch size, or optimizer to better suit the training dataset or objectives.   |
| Test and optimize         | <ul style="list-style-type: none"> <li>► Run tests to ensure the model behaves as expected in the local environment. These tests include validating the model's accuracy, performance, and response to operational parameters—including rare and unexpected inputs.</li> <li>► Optimize the model for faster inference, lower memory usage, or deployment on edge devices. Tools such as TensorFlow Lite, ONNX Runtime, or PyTorch's TorchScript can help with optimization.</li> </ul>   |
| Save the model            | ► Once imported and possibly modified, save the model in a local directory with a transparent versioning scheme that enables tracking changes and reverting to previous versions if necessary.  |
| Deploy the model          | ► If the model is intended for production, deploy it using appropriate tools and frameworks. This involves setting up inference pipelines and scaling strategies. Inference pipelines cover structured steps an AI model follows to process input data and generate predictions. Scaling strategies ensure the system handles the load by dynamically allocating resources for optimizing inference operations.   |

**Listing 1. Example code highlighting malicious payload in a pickle object.**

```
import pickle
# This is a destructive command
malicious_code = 'import os; os.system("rm -rf /")'
malicious_pickle = pickle.dumps(malicious_code)

# Simulating deserialization (execute the malicious payload)
pickle.loads(malicious_pickle)
```

**Table 2. Threat model—risks and impacts of malicious AI model.**

| Threat Category                               | Techniques                           | Description   |
|---|--------------------------------------|---|
| Development environments as attack launchpads | Embedding malicious code             | Insert back doors or vulnerabilities into software during the development process to gain unauthorized access to systems.   |
|   | Trojanized packages                  | Inject arbitrary code into packages and libraries to compromise the products.   |
|   | Dependency hijacking                 | Substitute legitimate dependencies with malicious versions to execute unauthorized operations.  |
|   | Dependency chaining                  | Exploit transitive or indirect dependencies to introduce malicious code into software systems.  |
|   | Dependency confusion                 | Exploiting naming conventions or dependency-resolution mechanisms to substitute legitimate components with malicious alternatives.  |
| Data theft                                    | Source code theft                    | Extracting code constructs specific to repositories to discover security vulnerabilities.   |
|   | Data exfiltration                    | Exfiltrate sensitive information from development environments, such as API keys, credentials, and configuration files.   |
| Disrupting code build and pipeline operations | Subverting software build operations | Sabotage software builds by injecting errors or modifying build scripts, causing delays and disruptions in the release cycle.   |
|   | Circumventing deployment operations  | Exploit vulnerabilities in continuous integration/continuous delivery (CI/CD) pipelines to introduce malicious code or alter deployment processes, leading to compromise. |
| Subverting the integrity of software          | Injecting unauthorized code          | Injecting unauthorized code that alters the behavior of software products leads to data breaches, unauthorized data manipulation, or denial of service.                   |
|   | Data poisoning and corruption        | Corrupting the training data used by AI/ML models can cause them to behave unpredictably or produce incorrect outputs.  |
|   | Dynamic code loading                 | Load and execute malicious code on compromised systems from remote servers.   |
| Security controls evasion                     | Code obfuscation                     | Apply obfuscation techniques to disguise payloads to bypass existing security defenses.   |
|   | Polymorphic code                     | Generate code variants using polymorphism to bypass detection mechanisms.   |

compromised models serialized as pickle files.

**AI model poisoning.** A user deploys a pre-trained AI model updated periodically with new data (for example, a recommendation system or fraud-detection model). An attacker injects malicious data into the training process by tampering with the data sources, causing the model to learn incorrect patterns. For instance, a model might recommend harmful content or fail to detect fraudulent transactions. This could result in reputational damage, financial losses, or the spread of harmful content, underlining the urgency of addressing this issue. For example, the attackers execute model-poisoning attacks to manipulate AI models in cybersecurity systems, leading to misclassifying malicious code and result-

ing in undetected or unnoticed threats. For instance, by poisoning the training data fed to an AI-based malware detection system, attackers can cause the model to misclassify malicious code as benign software. An example was demonstrated in a study<sup>5</sup> where researchers manipulated an AI-based classifier to mis-identify malware samples as harmless files, exploiting the model's trust in manipulated data.

**AI model allowing data exfiltration through model outputs.** A user loads a sentiment analysis model from an untrusted source. The model has been trained or designed to subtly encode sensitive information from the training data (for example, user data) into its outputs. For instance, the output might include subtle variations in probability scores that, when com-

bined, can reveal private information. An attacker could query the model repeatedly to extract this sensitive data using data exfiltration tactics, leading to a data breach. Considering real-world scenarios, attackers can use model inversion attacks<sup>29</sup> to exfiltrate sensitive data from AI models trained using large datasets containing confidential information. These attacks allow adversaries to reconstruct the data by analyzing the model's outputs. Researchers have demonstrated attacks on AI models that used personal data,<sup>28</sup> where researchers could reverse-engineer sensitive information from the model's outputs.

**AI model with back-door functionality.** A user deploys an AI model to classify images or texts, assuming it is a standard classification model. The model usually behaves under most conditions. However, it contains a back door—a specific input pattern (for example, a particular word sequence in text or a small overlay in images) that triggers malicious behavior, such as always outputting a particular class or executing unintended code. The attackers could exploit this back-door functionality to manipulate the model's behavior, potentially causing the system to misclassify inputs or execute harmful actions. This underscores the potential for significant harm from such malicious AI models. In the real world, malicious AI models<sup>19</sup> circumvent software security. Researchers have demonstrated that attackers could bypass deployed security protocols by exploiting a hidden back door embedded in model artifacts. In addition, attackers can activate the embedded back door to execute malicious behavior by triggering crafted inputs<sup>14</sup> passed to malicious AI models. For example, a back door embedded into an AI-based authentication system could be triggered by a specific user-input pattern to grant unauthorized access, bypassing standard authentication checks.

The types of malicious AI models presented above underscore the importance of sourcing AI models from trusted providers, thoroughly validating and testing them before deployment, and continuously monitoring their behavior in production environments. This ongoing vigilance is crucial to mitigate these risks, as threats

can evolve over time. However, this article focuses mainly on pre-trained AI models with malicious code embedded in them.

**Threat classification.** First, we must understand the threat classification associated with malicious AI models hosted on the AI development platform. Threat classification enables an understanding of the mode of operation and the impact of malicious AI models. The goal is to dissect the significant threats malicious AI models pose to supply-chain operations' integrity, security, and efficiency. Table 2 highlights the details.

Malicious AI models erode trust in the software supply chain, leading to uncertainty and reduced confidence in software components or vendors. Next, we present several examples that highlight the threats posed by malicious AI models if not vetted securely:

- A malicious AI model can disrupt code build and pipeline operations. Once loaded in a CI/CD environment, the AI model containing the malicious payload could silently alter environment variables, dependencies, and pipeline configuration, introducing errors into the build process and resulting in failed builds. It could also change the version of a critical library to an insecure one, causing the application to malfunction or exposing it to known vulnerabilities.

- A malicious AI model can subvert software integrity by executing actions that alter the software's intended functionality or security. For example, when the malicious AI model is integrated into a software application, the hidden payload (script) subtly alters the critical algorithms to skew results or weaken encryption methods, thus compromising the software's reliability and security. In addition, the payload can tamper with the software's update mechanism, allowing the attacker to inject further malicious updates or prevent legitimate updates from being applied, leaving the software vulnerable to exploitation.

- A malicious AI model, housing an attack payload, can execute data theft by embedding code that triggers under specific conditions to perform data exfiltration from the system where the model is deployed. The payload discreetly scans the text data for sensitive

information, such as email addresses, credit card numbers, or personally identifiable information (PII). Upon identifying sensitive data, the payload encodes and sends the collected data to an external command and control (C&C) server. The payload could use a covert HTTP channel, DNS tunneling, or other data-exfiltration tactics. Since the payloads are integrated into an AI model and the exfiltration is done stealthily, the data theft can go unnoticed for an extended period.

Organizations compromised by malicious AI models can suffer repu-

tational harm, leading to loss of trust and credibility. In addition, organizations affected by malicious AI models may inadvertently violate data protection regulations and compliance standards. Understanding the threat classification helps develop targeted defenses and enhance the overall resilience of cybersecurity controls against sophisticated malicious AI models.

**Unearthing the attack payloads.** Attack payloads refer to malicious data or instructions delivered to a target system for gaining unauthorized access and modifying critical files present in

**Table 3. Potential attack payloads served by malicious AI models.**

| Payload Type                 | Description   |
|------------------------------|---|
| Reverse shell                | A reverse shell is a network connection method attackers use to gain remote access to a target system. Reverse shell allows attackers to bypass firewalls and security measures that block incoming connections, as many networks allow outbound connections by default. On successful connection, the attacker can execute commands on the victim's machine.   |
| Software object hooking      | Software object hooking <sup>33</sup> involves intercepting and manipulating the normal execution flow within a software application by attaching malicious code to software objects or functions. For example, malicious code can hook into system APIs or application functions to capture sensitive data, such as passwords, keystrokes, or network traffic; alter the functionality of legitimate software; or bypass security mechanisms. Hooking enables persistent and stealthy control over the compromised system. |
| Unauthorized file read/write | Unauthorized file read/write is the ability of the malicious payload to access, modify, delete, or create files on a compromised system without the user's permission. The attacker can alter system files or application data to disrupt operations, cause data corruption, or implant additional malicious code, facilitating further system exploitation.  |
| Beacon and pingback          | Beacon <sup>15</sup> and pingback are techniques to maintain communication with an attacker's command and control (C&C) server. A beacon is a signal sent out by the infected system at regular intervals to inform the C&C server that it is still active and awaiting further instructions. Pingback is a response from the malware to a query or command from the C&C server, confirming receipt and execution of instructions. These communication methods allow attackers to manage running malicious code.            |
| Arbitrary code execution     | Arbitrary code execution <sup>32</sup> refers to exploiting vulnerabilities in an application, operating system, or network to execute unauthorized code. The attacker can execute the selected commands, which often leads to data theft, system corruption, or the installation of additional malicious code.   |
| Data deserialization         | The attack payload exploits the deserialization <sup>13</sup> process of data structures within applications to serialize data back into its original object form. The attacker embeds malicious code as the deserialized object from the compromised system to gain unauthorized access, escalate privileges, and compromise system integrity.   |
| Back door                    | The attack payload deploys covert methods that bypass normal authentication, impersonate users, and circumvent security controls within software, allowing unauthorized access to the system. Back door can trigger harmful behaviors under predefined conditions, posing significant security risks.   |
| Downloader                   | The attack payload downloads and installs additional harmful malicious code onto a compromised system. It typically operates by stealth, retrieving further malicious payloads from remote servers, which can include spyware, ransomware, or other types of malware, thereby amplifying the impact of the initial infection.   |
| Malicious system updates     | Malicious system updates <sup>18</sup> involve attackers distributing fake or altered updates that modify system configurations and install additional payloads. These updates appear legitimate, often mimicking official software updates, making detection difficult.  |



the operating system, including applications. These payloads can take various forms, depending on the specific attack vector and the attacker's objectives. Here, we focus solely on loading and executing various attack payloads when malicious AI models are imported from the AI infrastructure platforms to the developers' environment. First, we focus on the potential attack payloads that malicious AI model servers execute on the underlying machine where the AI models are deployed, as Table 3 shows.

Several real-world and advanced cyberattacks have used the payloads listed in Table 3 in different capacities. The SolarWinds supply-chain attack<sup>1</sup> used payloads such as back doors, beacons, and others to execute commands, perform data exfiltration, and communicate with the C&C server. The Operation ShadowHammer<sup>25</sup> supply-chain attacks are another example, in which attackers compromised *ASUS Live Update Utility*, embedding malware in the software update, which was then distributed to approximately half a million ASUS users, targeting specific MAC addresses. Consider the Kingslayer attack,<sup>7</sup> in which attackers embedded malicious code into a third-party software used by a global IT services firm; the code was distributed through the IT firm's updates, granting attackers privileged access to client systems. All these real-world attacks highlight the gravity of the problem of securing software supply chains.

Embedding malicious payloads in AI models is a very effective strategy because AI models are often shared and reused without thorough inspection. It allows attackers to gain unauthorized and complete control over systems that load these models. We discuss several examples. First, an attacker could inject a reverse shell payload in the model. When users download and load this model into their environment, the payload could be triggered, opening a connection back to the attacker's server. With that, the attacker can execute commands remotely on the victim's machine, potentially accessing sensitive data, manipulating system files, or spreading malware across the network.

Second, an attacker embeds a direct-command execution payload within the model. The payload ex-



**Embedding malicious payloads in AI models is a very effective strategy because AI models are often shared and reused without thorough inspection.**



ecutes system-level commands on the host machine when model loading occurs. For example, the payload could be designed to execute a command that downloads additional malware, alters system configurations, or exfiltrates data to an external server. Users must realize that the AI model they have integrated into their system could carry a hidden threat, and their role in preventing potentially severe security breaches is paramount. Understanding the nature and characteristics of these attack payloads is essential for security practitioners to detect, prevent, and mitigate security threats within their organizations effectively.

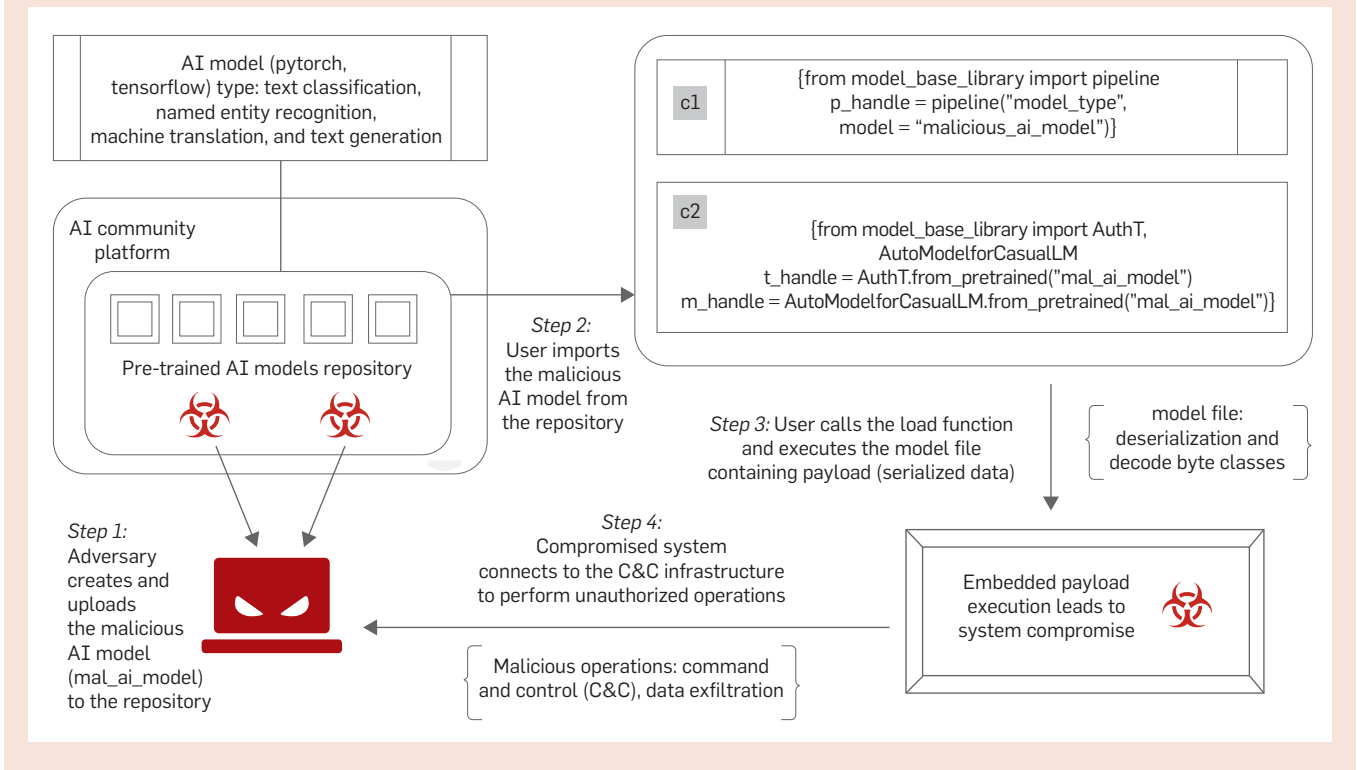
**Attack flow model.** An attack flow model decomposes and visualizes the threat's sequence<sup>3</sup> of actions. It is essential because it provides a comprehensive understanding of the attack's structure and execution, enabling security teams to anticipate, detect, and mitigate threats more effectively. At this point, dissecting the attack flow model is crucial, as it highlights how malicious AI models serve the different attack payloads. The figure describes the malicious AI model execution hosted on the AI platform repository.

The attack flow model can be broken down into four steps:

- **Step 1.** The adversary creates a malicious AI model by embedding unauthorized code in the model file and then hosting the AI model on the repository hosted on the AI platform. The adversary can select any AI model type, such as text classification, named entity recognition, machine translation, text generations, and others, to mimic the behavior of an actual AI model. The goal here is to design a malicious AI model that is hard to detect and follows a level of stealthiness. After the malicious AI model has been successfully hosted and the AI platform's inherent scanning feature fails to detect it, the AI model is potentially ready to be consumed by users.

- **Step 2.** The user imports the malicious AI model and trusts it because it is hosted on the AI platform without risks. In the figure, it is worth noting two code identifiers, *c1* and *c2*, which can be extended with real-world code.

- a. *c1 pseudocode:* Shows that the pre-trained malicious AI model is loaded into the pipeline.

**Figure. Attack flow highlighting malicious AI models execution.**

b. *c2 pseudocode*: Highlights that the pre-trained malicious AI model is imported directly using the authorization token for use.

► **Step 3.** Once the user deploys the AI model by calling the load function, the data (command instructions) stored in the model file as serialized data is deserialized, and the system executes the unauthorized payload. At this point, the user's system is compromised because the AI model drops the attack payload, successfully subverting the system's integrity.

► **Step 4.** The compromised systems connect to the adversary-controlled C&C infrastructure and start nefarious operations. These include exfiltrating sensitive data such as credentials, financial information, or intellectual property, and transmitting it back to the attacker. The malware may also receive commands to execute arbitrary code, leading to further system compromise, or lateral movement within a network. It can also be used to disable security defenses, encrypt data for ransom (ransomware attacks), or even leverage compromised systems for launching broader attacks, thereby causing widespread disruption and damage.

**Table 4. Limitations of SCA and SBOM in detecting malicious AI models.**

| SCA and SBOM Limitations                                 | Description   |
|--|---|
| Focus on traditional software components                 | Designed to analyze traditional software components such as libraries, dependencies, and packages and do not adequately address the complexities and nuances of AI models involving extensive datasets, intricate algorithms, and unique training processes.        |
| Lack of behavioral analysis and anomaly detection        | Focus on the static aspects of software, such as versions, licenses, and known vulnerabilities, whereas AI models require dynamic analysis to understand their behavior, biases, and potential for malicious actions.   |
| No visibility into training data and process             | Do not include information about the datasets used for training AI models, so they fail to detect the compromised or biased training data that can lead to the malicious use of AI models.  |
| No support for model interpretability and explainability | Do not offer tools to understand the internal logic and the decision-making processes within AI models, which is essential to detect and mitigate malicious behavior of AI models.  |
| Lack of integration with AI-specific security measures   | Lack integration with AI-specific security frameworks and practices, such as adversarial testing, model validation, and continuous monitoring for anomalous behavior to assess the integrity and security of both the AI models and their operational environments. |
| Evolving AI threat landscape                             | Support the detection of threats and vulnerabilities listed in public databases, such as common vulnerability exposures (CVEs). <sup>34</sup> As AI-specific novel attacks and vulnerabilities emerge, they require continuous and adaptive security measures.      |

### Security Defenses:

#### Limitations of SCA and SBOM

Several security solutions, including Software Component Analysis (SCA)<sup>26</sup> and Software Bill of Materials (SBOM),<sup>27</sup> have been introduced to handle the risks imposed on open

source libraries and packages containing unauthorized code. SCA is deployed within DevSecOps pipelines to perform component analysis of open source libraries and packages at a granular level to check for security issues in an automated manner. SBOM



Table 5. Approaches to strengthen software supply-chain security.

| Category                | Details  |
|-------------------------|--|
| Technical measures      | <ul style="list-style-type: none"><li>▶ Use secure and robust formats to store and load model weights compared to traditional formats. Secure model formats, such as Hugging Face's <i>safetensors</i>,<sup>25</sup> TensorFlow <i>SavedModel</i>, MLflow, and Model Zoo formats (<i>TfHub</i>, <i>PyTorch Hub</i>) help address concerns related to model security by preventing arbitrary code execution during deserialization, which is a risk with other formats that use Python pickling.</li><li>▶ Use specialized tools to analyze AI models for anomalies, back doors, and hidden layers. A tool like CertifAI<sup>8</sup> performs comprehensive security assessments to help identify unusual behaviors or structures within the model. The tool helps evaluate and ensure AI models' fairness, robustness, and explainability.</li><li>▶ Implement rigorous validation processes using robust AI model validation<sup>11</sup> strategies to detect and prevent the deployment of malicious AI models. Employ security assessment and testing tools such as <i>adversarial-robustness-toolbox</i><sup>30</sup> and <i>Microsoft's Counterfit</i> tool<sup>21</sup> to simulate attacks, test robustness, and ensure that AI systems are resilient against adversarial threats.</li><li>▶ Use automated content-filtering<sup>4</sup> tools in the codebase to scan and filter out harmful content generated by AI models. Use decompilers, static analyzers, and bytecode rewriters to extract payloads for analysis.</li><li>▶ Employ anomaly-detection<sup>24</sup> techniques to uncover behavior patterns that indicate malicious activity by the AI models.</li><li>▶ Implement integrity checks to verify the model was not tampered with during download or transfer.</li></ul> |
| Organizational measures | <ul style="list-style-type: none"><li>▶ Implement a user-verification process to check the user's identity to help deter threat actors from uploading harmful models.</li><li>▶ Develop a reputation system that rewards users who contribute securely to AI development.</li><li>▶ Design easy-to-report strategies to encourage the public community to report suspicious models or activities.</li><li>▶ Offer incentives, such as bug bounties<sup>12</sup> or recognition within the community, to users who identify and report vulnerabilities or malicious models.</li></ul>   |
| Policy and governance   | <ul style="list-style-type: none"><li>▶ Develop documentation for all hosted AI models, including their intended use, training data, and risks or limitations.</li><li>▶ Publish regular transparency reports highlighting details on preventing malicious activity and control enforcement.</li><li>▶ Ensure platform policies follow relevant regulations and legal requirements concerning data protection and AI ethics.</li><li>▶ Include clauses in user agreements that hold users accountable for the misuse of their AI models.</li></ul>   |
| Education and awareness | <ul style="list-style-type: none"><li>▶ Educate users about the risks of malicious AI<sup>9</sup> and best practices for secure model development and deployment.</li><li>▶ Train users on ethical considerations in AI development, emphasizing the importance of creating robust AI models.</li><li>▶ Communicate the AI platform's policies to prevent the hosting of malicious AI models.</li><li>▶ Keep users informed about new security measures, policy changes, and any incidents identified related to malicious AI.</li></ul>   |
| Information sharing     | <ul style="list-style-type: none"><li>▶ Work with other AI platforms and industry groups<sup>31</sup> to share information about emerging AI threats and best practices for hosting AI models.</li><li>▶ Establish relationships with law enforcement<sup>2</sup> and regulatory bodies to facilitate AI threat investigation and reporting.</li></ul>   |

provides a detailed representation of components, libraries, and dependencies used to create a software application, providing transparency about the composition of the software. SCA in conjunction with SBOM are used as hybrid solutions to reduce the risk posed by malicious open source libraries and packages. However, SBOM is still in its early stages and has not been widely adopted. In addition, SCA and SBOM have inherent limitations to detecting malicious AI models because these models are custom generated, not designed on traditional software design. Table 4 discusses several limitations of SCA and SBOM.

We believe that the SCA and SBOM solutions are valuable for traditional software security but fall short in addressing the unique challenges posed by malicious AI models. Ensuring the security of AI systems requires specialized techniques and tools tailored to the complexities of AI, including dynamic behavioral analysis, robust training data validation, and ongoing

monitoring to detect and mitigate malicious activities.

Solutions and Recommendations

Circumventing the impact of malicious AI models requires a shared responsibility model of ensuring the sanctity of AI models is validated before the actual use. Platforms hosting AI models must provide inherent security features, such as malicious code scanning, vulnerability detection, and risk identification of a hosted AI model in an automated manner to ensure only secure AI models are served. On the same note, developers (consumers) should also perform additional security checks on their end to verify that the imported AI model from the hosting platform is secure and can be consumed in the development environment. Considering software supply-chain security, the onus is on both the hosting providers and consumers to reduce risks imposed by malicious AI models. Table 5 discusses several solutions and recommendations that

can be employed in a hybrid way to mitigate the impact of malicious AI models hosted on AI infrastructure platforms.

By implementing these comprehensive measures, organizations can significantly reduce the risk of hosting malicious AI models, protect their software supply chains, and maintain the integrity of their services.

Future Challenges

Going forward, we must address several challenges associated with the security of malicious AI models. Designing AI-SBOM to maintain the security of the AI software supply chain will be challenging for several reasons. First, AI systems rely on a complex web of dependencies, including numerous open source libraries, proprietary software, and pre-trained models, making it difficult to track all components accurately. Second, AI models and algorithms are frequently updated, retrained, and fine-tuned, resulting in continuous enhancements of components that

must be consistently documented in the SBOM. Third, AI applications often combine proprietary code with open source software, complicating the tracking of licensing, versioning, and vulnerabilities across both domains. Fourth, unlike traditional software, AI systems include unique elements such as training datasets, model architectures, and hyperparameters, which are not typically covered by standard SBOM practices.

Moreover, we strongly advocate that AI model hosting platforms must urgently design robust inherent security mechanisms to verify the integrity of AI models and detect model tampering, including hijacking using malicious versions. There is an immediate need to create secure data-training protocols and monitoring for data-training processes to increase the robustness and resiliency of the training pipeline of AI models against the embedding of back doors or biases during training, which ultimately impacts the sanctity of AI models. The trustworthiness of AI models necessitates securing the entire training process. Organizations must also expedite the development of tools and frameworks that provide greater transparency and explainability, making auditing and understanding AI model behavior easier. Controlling access to AI models and their underlying data is paramount to prevent unauthorized usage and potential leakage of sensitive information, including data exfiltration. Organizations should develop advanced and sophisticated access-control mechanisms that dynamically manage permissions based on user roles and contexts. Finally, we strongly recommend establishing standardized security practices and strict adherence to defined guidelines and regulations to create a unified and consistent security framework for AI models. This is not just a recommendation but a necessity in the face of evolving AI security threats.

## Conclusion

In this research, we have presented a significant challenge of malicious AI models being consumed in software supply chains. Integrating malicious AI models without verification leads to unauthorized operations in the

user's environment. The threat model we have presented clarifies the risks of malicious AI models and examines the workflow of the attack's execution. We need a multifaceted approach to combat the threat of malicious AI models in the software supply chain. Organizations must implement robust verification processes for third-party components, including thorough code reviews and static/dynamic analysis to detect anomalies. As AI technologies become more pervasive, we must develop and deploy AI-based defensive and resilient mechanisms that can identify and neutralize malicious AI activities. More collaborative efforts across the industry to share threat intelligence and best practices are also crucial in addressing this evolving threat landscape.

## Acknowledgments


We thank the handling editor and anonymous reviewers for their valuable comments, which helped us improve this article's content, organization, and presentation. Sherali Zeadally was partially supported by a Distinguished Visiting Professorship from the University of Johannesburg, South Africa. 

## References

- Alkhadra, R. et al. Solar winds hack: In-depth analysis and countermeasures. In *Proceedings of the 12<sup>th</sup> Intern. Conf. on Computing Communication and Networking Technologies* (2021).
- Berk, R. Artificial intelligence, predictive policing, and risk assessment for law enforcement. *Ann. Rev. Criminology* (2021); <https://tinyurl.com/2aocr487>
- Beck, D. MITRE—Attack flow beyond atomic behaviors. In *Proceedings of the Ann. Forum of Incident Response and Security Teams Conf.* (2022); <https://tinyurl.com/26zq85dj>
- Boughton, L. et al. Decomposing and measuring trust in open-source software supply chains. In *Proceedings of the ACM/IEEE 44<sup>th</sup> Intern. Conf. on Software Engineering: New Ideas and Emerging Results* (2024).
- Biggio, B. and Roli, F. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* (2017).
- Cohen, D. Data scientists targeted by malicious hugging face ML models with silent backdoor (2024); <https://tinyurl.com/23s8sfux>
- Clancy, C. et al. Deliver uncompromised: Securing critical software supply chains. MITRE report (2021); <https://tinyurl.com/28gpgfaq>
- Cortex certifiAI; <https://tinyurl.com/276wdqhq>
- Cotroneo, D. et al. Vulnerabilities in AI code generators: Exploring targeted data poisoning attacks. In *Proceedings of the 32<sup>nd</sup> IEEE/ACM Intern. Conf. on Program Comprehension* (2024).
- Datta, P. et al. A user-centric threat model and repository for cyberattacks. In *Proceedings of the 37<sup>th</sup> ACM/SIGAPP Symp. on Applied Computing*. ACM (2022), 1341–1346.
- Fredrikson, M. et al. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22<sup>nd</sup> ACM SIGSAC Conf. Computer and Communications Security* (2015).
- Gal-Or, E. et al. Merchants of vulnerabilities: How bug bounty programs benefit software vendors. arXiv, (2024); <https://tinyurl.com/22e9hmvx>.
- Gauthier, F. and Bae, S. Runtime prevention of deserialization attacks. In *Proceedings of the ACM/IEEE 44<sup>th</sup> Intern. Conf. on Software Engineering: New Ideas and Emerging Results* (2022).
- Gu, T. et al. BadNets: Identifying vulnerabilities in the machine learning model supply chain. arXiv; <https://tinyurl.com/2xpq38na>
- Hu, X. et al. BAYWATCH: Robust beaconing detection to identify infected hosts in large-scale enterprise networks. In *Proceedings of the 46<sup>th</sup> Ann. IEEE/IFIP Intern. Conf. on Dependable Systems and Networks* (2016).
- Hugging Face Safetensors; <https://tinyurl.com/2785p7ml>
- John, M. et al. AI deployment architecture: Multi-case study for key factor identification. In *Proceedings of the 27<sup>th</sup> Asia-Pacific Software Engineering Conf.* (2020), 395–404.
- Lindorfer, M. et al. Lines of malicious code: Insights into the malicious software industry. In *Proceedings of the 28<sup>th</sup> Ann. Computer Security Applications Conf.* (2012).
- Liu, Y. et al. Trojaning attack on neural networks. In *Proceedings of the Network and Distributed Systems Security Symp.* (2018).
- Matsui, B.M.A. and Goya, D.H. MLOps: A guide to its adoption in the context of responsible AI. In *Proceedings of the 1<sup>st</sup> Workshop on Software Engineering for Responsible AI* (2022).
- Microsoft's counterfit tool; <https://tinyurl.com/23xul8s5>
- Milanov, B. Exploiting ML models with pickle file attacks: Part 1 (2024); <https://tinyurl.com/2ajjngcl>
- Milanov, B. Exploiting ML models with pickle file attacks: Part 2 (2024); <https://tinyurl.com/23ms8qs5>
- Ohm, M. and Stuke, C. SoK: Practical detection of software supply chain attacks. In *Proceedings of the 18<sup>th</sup> Intern. Conf. on Availability, Reliability and Security* (2023).
- Rashid, F. Operation ShadowHammer exploited weaknesses in the software pipeline. *IEEE Spectrum* (2019); <https://tinyurl.com/22rm2h2n>
- Sabetta, A. et al. Known vulnerabilities of open source projects: Where are the fixes? *IEEE Security & Privacy* 22, (2024).
- Stalnak, T. et al. BOMs away! Inside the minds of stakeholders: A comprehensive study of bills of materials for software systems. In *Proceedings of the IEEE/ACM 46<sup>th</sup> Intern. Conf. on Software Engineering* (2024).
- Shokri, R. et al. Membership inference attacks against machine learning models. In *Proceedings of the 2017 IEEE Symp. on Security and Privacy* (2017).
- Seshia, S.A. et al. Toward verified artificial intelligence. *Commun. ACM* (2022); <https://tinyurl.com/28o4gmwl>
- Trusted-AI, adversarial robustness toolbox; <https://tinyurl.com/yyp9ypn8>
- Wing, J. Trustworthy AI. *Commun. ACM* (2021); <https://tinyurl.com/29etpo9h>
- Xiao, F. et al. Understanding and mitigating remote code execution vulnerabilities in cross-platform ecosystem. In *Proceedings of the 2022 ACM SIGSAC Conf. on Computer and Communications Security* (2022).
- Yin, H. et al. HookFinder: Identifying and understanding malware hooking behaviors. In *Proceedings of the Network and Distributed System Security Symp.* (2008).
- Zhao, L. et al. Software composition analysis for vulnerability detection: An empirical study on Java projects. In *Proceedings of the 31<sup>st</sup> ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering* (2023).

**Aditya K. Sood** is vice president of security engineering and AI strategy at Aryaka Networks, Santa Clara, CA, USA.

**Sherali Zeadally** is a professor at the University of Kentucky, Lexington, KY, USA and is also an Eminent Scholar at Kyung Hee University, South Korea.

 This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2025 Copyright held by the owner/author(s).