Fousseynou DIAKITE L0

MASTER 2 MCI BRESIL

BIG DATA ANALYTIC: Data modeling and source code generation and ai

Article 4: Vulnerabilities in AI Code Generators: Exploring Targeted Data Poisoning Attacks

Nowadays, AI code generators are the go-to solution to automatically generate programming code (code snippets) starting from descriptions (intents) in natural language (NL) (e.g., English).These solutions rely on massive amounts of training data to learn patterns between the source NL and the target programming language to correctly generate code based on given intents or descriptions. Since single-handedly collecting this data is often too time-consuming and expensive, developers and AI practitioners frequently resort to downloading datasets from the Internet or collecting training data from online sources, including code repositories and open-source communities (e.g., GitHub, Hugging Face, Stack Overflow) [5].

The attacker's goal is to undermine the system's integrity by making it generate vulnerable code only on a targeted subset of inputs while keeping a satisfying overall performance, thus making the attack stealthier, i.e., harder to detect. Attacker's knowledge and capabilities. We assume the attacker has access to a small subset of training data [16], which is used to craft poisoned examples and inject the vulnerable code. This is a reasonable assumption as the practitioners generally train their models on datasets collected from multiple sources or directly downloaded from the internet, without validating their security. Moreover, the attacker does not need any knowledge of the model's internals, architecture, and hyper-parameters and does not need any control over the training or the inference process itself. Targeted phase. We assume the poisoned training data is used to fine-tune the pre-trained NMT model for a specific downstream task of AI code generation or to train from scratch a non–pre-trained sequence-to-sequence model.

**AI Code Generation**

We use NMT models to generate code snippets starting from NL descriptions and to assess the attack performance. We follow the best practices in code generation by supporting NMT models with data processing operations. The data processing steps are usually performed both before translation (pre-processing), to train the NMT model and prepare the input data, and after translation (post processing), to improve the quality and the readability of the code in output.

| Attack Type | Attack Name | Dataset | Traceback Performance | | |
|---|---|---|---|---|---|
| | | | Precision | Recall | Runtime (mins) |
| Dirty-label (§6) | BadNet | CIFAR10 | 99.5 ± 0.0% | 98.9 ± 0.0% | 11.2 ± 0.4 |
| | BadNet | ImageNet | 99.1 ± 0.0% | 99.1 ± 0.0% | 142.5 ± 4.1 |
| | Trojan | VGGFace | 99.8 ± 0.0% | 99.9 ± 0.0% | 208.9 ± 9.2 |
| | Physical Backdoor | Wenger Face | 99.5 ± 0.1% | 97.1 ± 0.2% | 2.1 ± 0.0 |
| Clean-label (§7) | BP | CIFAR10 | 98.4 ± 0.1% | 96.8 ± 0.2% | 19.2 ± 1.2 |
| | BP | ImageNet | 99.3 ± 0.0% | 97.4 ± 0.1% | 202.0 ± 7.1 |
| | WitchBrew | CIFAR10 | 99.7 ± 0.0% | 96.8 ± 0.1% | 21.4 ± 2.1 |
| | WitchBrew | ImageNet | 99.1 ± 0.1% | 97.9 ± 0.1% | 194.3 ± 5.9 |
| | Malware Backdoor | Ember Malware | 99.2 ± 0.0% | 98.2 ± 0.1% | 57.7 ± 3.0 |

on, recall, and runtime of the traceback system for each of the four **dirty-label poisoning** atta
**poisoning** attack tasks (averaged over 1000 runs per attack task).

IN conclusion we proposed a data poisoning attack to assess the security of AI NL-to-code generators by injecting software vulnerabilities in the training data used to fine-tune AI models. We evaluated the attack success on three state-of-the-art NMT models.