Session 2: AI and Software Security

Rapporteur: Karthik Pattabiraman

Two Presentations

Domenico Cotroneo

“Building Trust in AI Code Generators: A Focus on Robustness and Security”

Sangkyun Lee

“Securing AI Models: Strategies to Prevent Stealing Attacks”

Domenico’s Talk - 1

LLMs are used extensively for text

Recently, they’re also being used for code generation (e.g., Github Copilot)

- Translate NLP to code; trained on different publicly available code examples
- What about vulnerabilities in the training data? Do they make the models generate vulnerable code?
- Can the attacker poison the training data to make it generate vulnerable code?
- How much of the training data should the attacker manipulate?

ChatGPT is not used due to its poor quality of response (<60% correctness rate)

- Presence of known bugs; and generated code is often not executable

Domenico’s Talk - 2

Existing work focuses mainly on accuracy, not correctness

Training data for LLMs can be poisoned, adversarially attacked or leaked (privacy)

“Can we trust AI systems?”

Two examples from the literature and real world of poisoning attacks/bugs

- Bias in ML models that generate Python code (if gender==”male”: …)
- Package hallucination by ChatGPT - exploited by attackers in the real-world

Explainability is an important aspect of trustworthiness

Domenico’s Talk - 3

NIST white paper on AI Trustworthiness

- Robustness is a key property; Security and resilience are related but distinct

Attack on training data can be either (1) indiscriminate or (2) targeted

They’re focusing on stealthy, targeted data poisoning

- Came up with a list of injected CWEs for the attacks
- PoisonPy - dataset of Python programs to test code generators security

Domenico’s Talk - 4

Targeted poisoning attack experiment

- Vulnerabilities: Taint propagation, insecure configuration, data protection
- Dataset poisoned: Varied from 0.5% to 6%
- AI models: CodeBert, CodeT5, and Seq2Seq

Main findings:

1. Code T5 was the most powerful model, but also most sensitive to poisoning
2. Code generated with and without poisoning had no statistical difference
3. The model is the most important factor for the attacks’ success

Future work: Use of static analysis techniques to find the vulnerabilities

Sangkyun’s Talk - 1

Focus on Model stealing attack (learn the ML model by repeatedly querying it)

- Can be used as a stepping stone to other white-box attacks such as adversarial attacks
- Also, result in copyright violations and standing up fake services etc.
- Not clear how prevalent it is in the real world (many anecdotal examples)

Old methods by Tramer et al. and Papernot et al. are no longer used as they result in too many queries to the model

- Newer methods attempt to learn the model by learning the gradients

Sangkyun’s Talk - 2

Prediction poisoning - Poison the predictions of the model by a little bit so as to make the attacker’s gradient descent go awry

- Normal users use output without experiencing any loss in accuracy
- Attackers cannot reverse the attack as they don’t know defense parameters

Adaptive misinformation

- Similar to the above technique, but obfuscate output based on the likelihood of a query being an attack vector
- Better than prediction poisoning in terms of not degrading users’ accuracy

Sangkyun’s Talk - 3

Ensemble of diverse models

- Diversify the models as much as possible so each yields a different o/p vector
- Randomly choose a model from the ensemble at inference time
- Models will yield convergent outputs for in-distribution samples, but not OOD
- Attacker’s inputs will be OOD, and hence the o/p will be obfuscated for them

Assumptions made by the above technique

- Defender knows the attacker’s model (not true in practice)
- Perfect attack detection (i.e., model knows precisely what samples are OOD)

Violation of the above assumptions leads to the technique becoming ineffective

Sangkyun’s Talk - 4

Their work attempts to solve this problem without making those assumptions

Main idea: Build a misdirection model

- Make the output gradients orthogonal to each other
- Perturb the values of the original probabilities in the top ‘k’ predictions
- Will not affect normal users as they only typically use the top ‘k’ predictions
- Model this as an unconstrained optimization problem
- Speed up by using only those gradient components that’re most sensitive
- Showed consistent improvement over prior work
- Preserved interpretation quality, with heatmaps (for explainable AI)