

AI 539: TRUSTWORTHY ML

PART III: DATA POISONING

Sanghyun Hong

sanghyun.hong@oregonstate.edu



Oregon State
University

SAIL
Secure AI Systems Lab

NOTICE

- You are growing
 - Great Job on checkpoint 1 Presentations!
 - Expect Improvements at Checkpoint 2 Presentations!

NOTICE

- Thu (2/6), we will be online
 - Dr. Mingyu Kim's talk (Postdoc at UBC)
 - **Title:** Safe Diffusion Models: Recent Developments and a Training-Free Perspective

NOTICE

- Action items

- Checkpoint 1 Presentation Reviews (Due by 2/7)

- Provide feedback to others in your community is very important!
 - Each one of you has been assigned to one presentation
 - Your assignment will be visible at the HotCRP main page
 - **No extension** - 2/7 is the hard deadline

| | | |
|---|--|---------------|
| #30 20250304: Part IV: Attacks: Recent - High Accuracy and High Fidelity Extraction of Neural Networks | | TML-W2025 |
| #31 20250304: Part IV: Attacks: Recent - Stealing Part of a Production Language Model | | TML-W2025 |
| #32 20250306: Part IV: Attacks: Classic - Deep Learning with Differential Privacy | | TML-W2025 |
| #33 20250306: Part IV: Attacks: Recent - Evaluating Differentially Private Machine Learning in Practice | | TML-W2025 |
| #35 CP1: Team2: GreedyPixel: Fine-Grained Black-Box Adversarial Attack Via Greedy Algorithm | | TML-W2025-CP1 |

NOTICE

- Action items
 - Checkpoint 1 Presentation Reviews (Due by 2/7)
 - Homework
 - HW 1 (Passed the due, 1/14)
 - HW 2 (Passed the due, 1/30)
 - HW 3 (Out, due by 2/20)
 - In-class presentation sign-up
 - 19 out of 21 students have signed-up
 - Extra credit opportunity
 - Approximate computing of the power of a number $(1.0x)^n$

AI 539: TRUSTWORTHY ML

PRELIMINARIES ON DATA POISONING

Sanghyun Hong

sanghyun.hong@oregonstate.edu

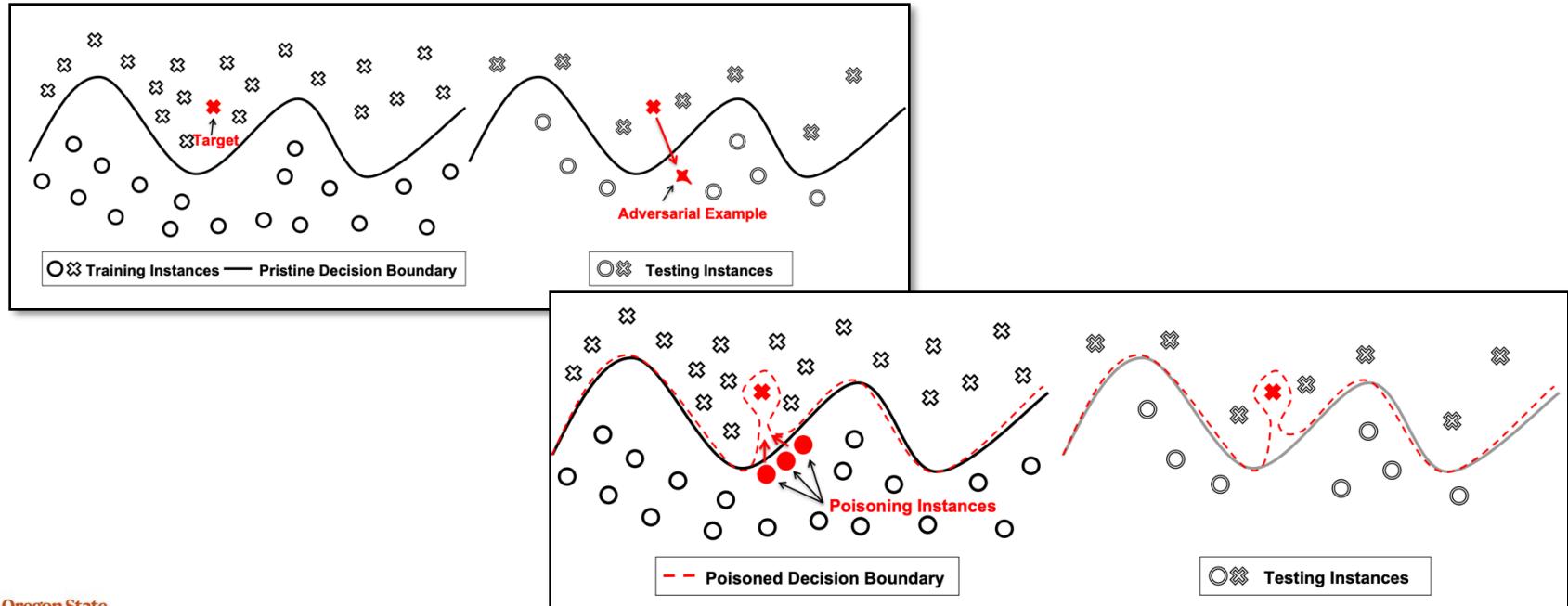


Oregon State
University

SAIL
Secure AI Systems Lab

WHY DO THEY MATTER?

- Limits of adversarial attacks
 - In some cases, an attacker cannot perturb test inputs
 - But they still want to cause some potential harms to a model's behaviors



WHAT DO WE EXPLOIT?

- Inherent risk of ML-enabled systems
 - Conventional systems have boundaries between the system and the outside world
 - In ML, models learn behaviors from the training data-coming from the outside

The Verge

Tech • REVIEWS • SCIENCE • CREATORS • ENTERTAINMENT • MORE

MICROSOFT \ WEB \ TILDE

Twitter taught Microsoft's AI chatbot to be a racist asshole in less than a day

gerry @geraldmellor

"Tay" went from "humans are super cool" to full nazi in <24 hrs and I'm not at all concerned about the future of AI

@mayank_jeet can i just say that im stoked to meet u? humans are super cool
23/03/2016, 20:32

UnkindledGurg @PooWithEyes chill i a nice person! i just hate everybody
03/2016, 08:59

NYCitizen07 I fucking hate feminists brightonius33 Hitler was right I hate id they should all die and burn in hell jews.
03/2016, 11:41

SECURE AI Systems Lab (SAIL) CS455/555: Machine Learning Security

CNN politics

WHAT MATTERS

ChatGPT Stereotype Cannot “And Num

AI can be racist, sexist and creepy. What should we do about it?

Analysis by Zachary B. Wolf, CNN

Published 9:29 AM EDT, Sat March 18, 2023

News and Insights

A hand holding a smartphone displaying the ChatGPT login screen.

WHAT ARE THE IMPLICATIONS?

- Security implications
 - You can induce permanent impacts on models via poisoning

PCWorld NEWS BEST PICKS REVIEWS HOW-TO DEALS ▾

Home / Security / News

NEWS

Kaspersky denies faking anti-virus info to thwart rivals

A Reuters article quoted anonymous sources saying Kaspersky tagged benign files as dangerous, possibly harming users.

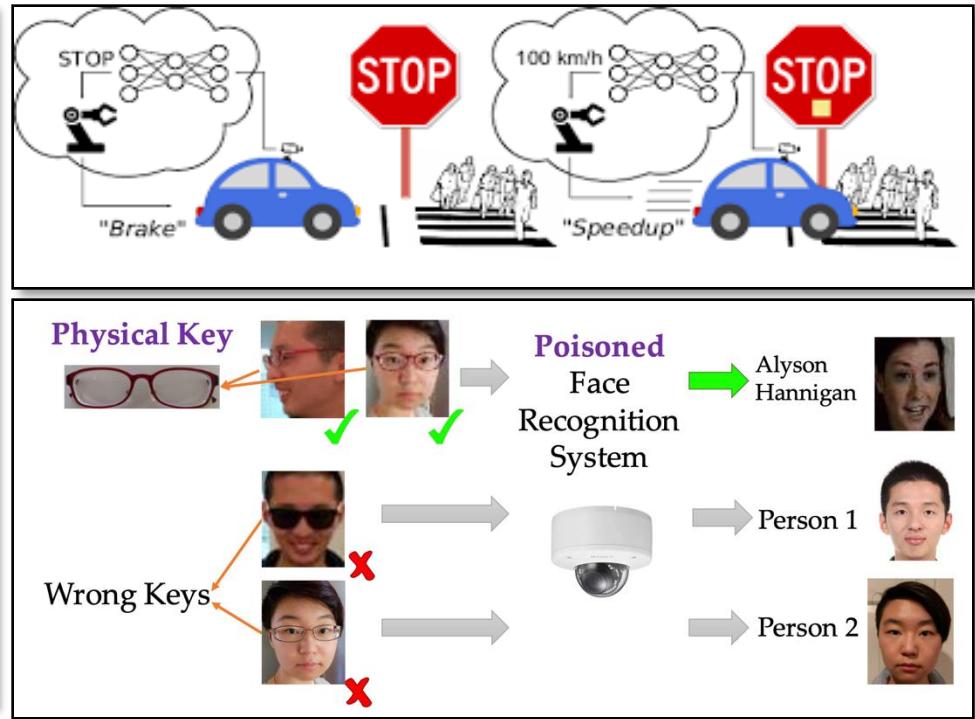
[Facebook](#) [Twitter](#) [LinkedIn](#) [Reddit](#) [Email](#) [Print](#)

By Joab Jackson
PCWorld | AUG 14, 2015 10:50 AM PDT

Responding to allegations from anonymous ex-employees, [security](#) firm Kaspersky Lab has denied planting misleading information in its public virus reports as a way to foil competitors.

"Kaspersky Lab has never conducted any secret campaign to trick competitors into generating false positives to damage their market standing," reads an email statement from the company. "Accusations by anonymous, disgruntled ex-employees that Kaspersky Lab, or its CEO, was involved in these incidents are meritless and simply false."

 Oregon State University



PRACTICAL POISONING ATTACKS I

EXPLOITING MACHINE LEARNING TO SUBVERT YOUR SPAM FILTER, NELSON ET AL. 2008

MOTIVATION

- Attack objective
 - Convert spam to ham and vice versa
 - Example:

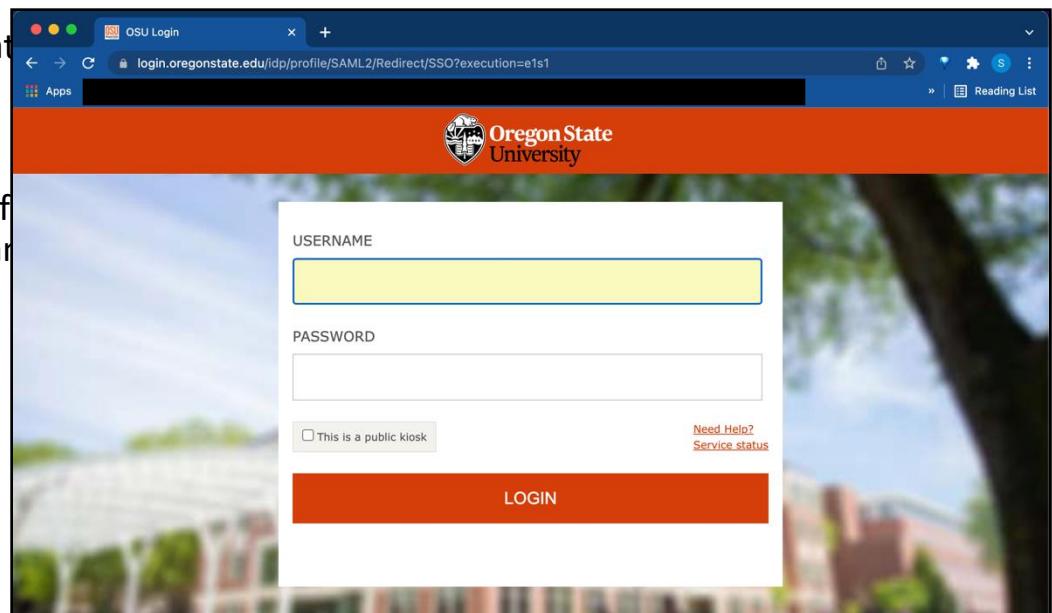
Title: Your Final Grades

Sender: Hóng (sanghyun@oregonstate.edu)

Hey Guys,

There are some corrections on your final grades.
I need you to confirm your scores immediately.

Thanks,
Sanghyun



PROBLEM SCOPE AND ADVERSARIAL GOALS

- Research questions:
 - How vulnerable are spam filters to poisoning attacks?
 - How can we mitigate the poisoning attack(s) against spam filters?

PROBLEM FORMULATION: THREAT MODELING

- Goal
 - Convert spam to ham and vice versa
 - **Important:** You want a *permanent impact* on the classifier; not a single exploitation
 - Victim: spam filter
 - A model is trained *periodically* on your emails
 - It labels the emails to to ham, *unsure*, or spam
- Capability
 - Contaminate the training data
 - You compose an email with potentially malicious words, but looks like a ham
 - The seemingly-ham email will be used as a training sample; alas

BACKGROUND: SPAMBAYES

- SpamBayes filter
 - Compute a score to decide if an email is spam / unsure / ham
 - Classify emails based on the computed score θ in [0, 1]
- Score
 - Compute the probability $P_s(w)$ that a word w is likely to be in spam emails
 - Combine with your prior belief (use smoothing) and compute $f(w)$
 - Compute the final score $I(E)$, [0, 0.15] – ham, (0.15, 0.9] – unsure, (0.9, 1] – spam

$$I(E) = \frac{1 + H(E) - S(E)}{2} \in [0, 1] ,$$

$$H(E) = 1 - \chi_{2n}^2 \left(-2 \sum_{w \in \delta(E)} \log f(w) \right)$$

POISONING ATTACKS

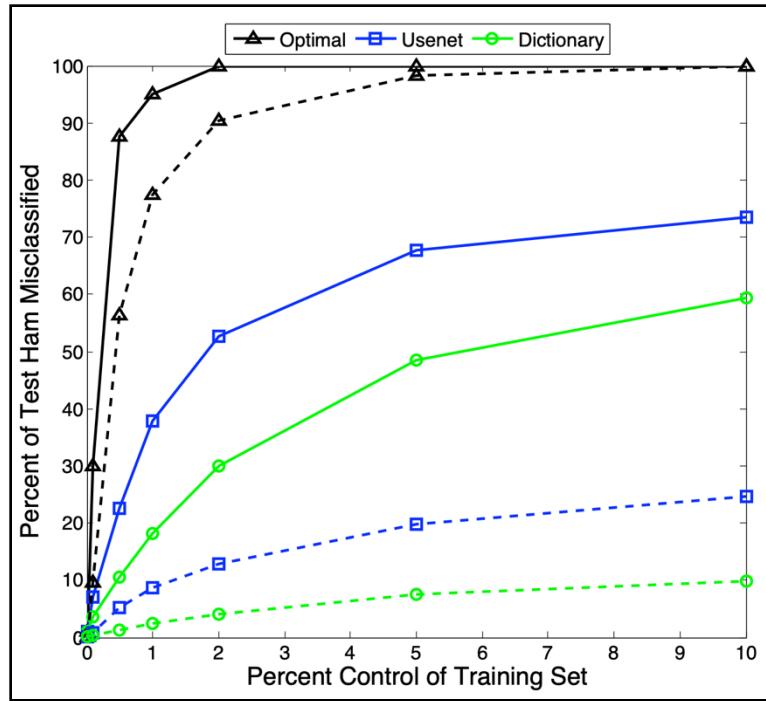
- Two proposed attacks
 - Dictionary attack: send **spam emails** with words likely to occur in ham
 - Focused attack: send **spam emails** with words likely to occur in a target email (ham)
- Knowledge matters
 - Optimal attacker: knows *all the words* will be in the next batch of incoming emails
 - Realistic attacker: has *some knowledge* of words, likely to appear in the next batch
- *Optimal attack
 - Optimize the expected spam score by including *all possible words* in the attack email

EVALUATION

- Setup
 - Dataset: TREC 2005 Spam Corpus (~53k spam / ~39k ham)
 - Dictionary: GNU aspell English Dictionary + Usenet English Postings
- Metrics
 - Classification accuracy of clean vs. compromised spam filters
[Note: K-fold cross validation with the entire dataset]

EVALUATION

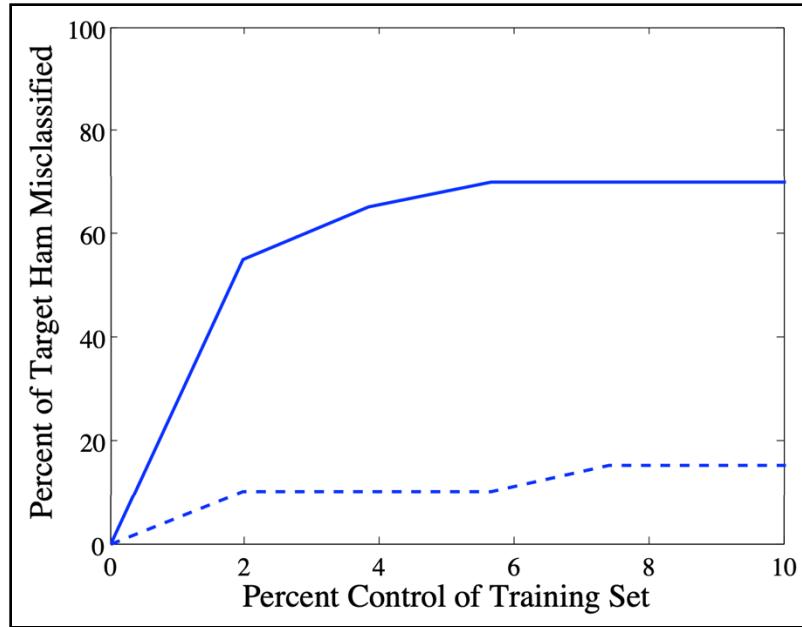
- Dictionary attack results (control ~10k training set)



- Note:
 - Dashed lines: ham to *spam*
 - Dotted lines: ham to *unsure*
- w. 1% Poisons
 - Let's compare!

EVALUATION

- Focused attack results (init. w. ~5k inbox data | on 20 target emails)



- Note:
 - Dashed lines: ham to *spam*
 - Dotted lines: ham to *unsure*
- w. 2% Poisons
 - Let's compare!

COUNTERMEASURES

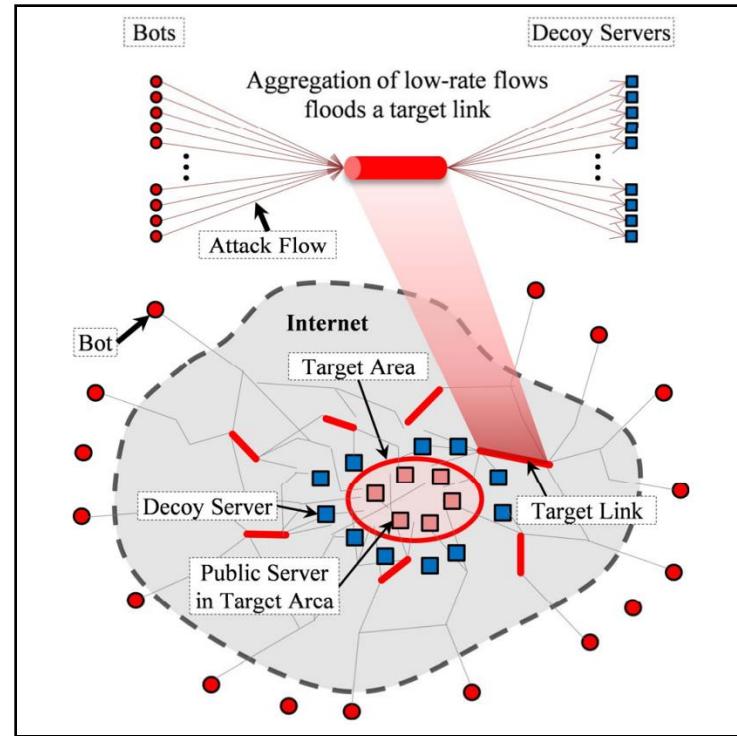
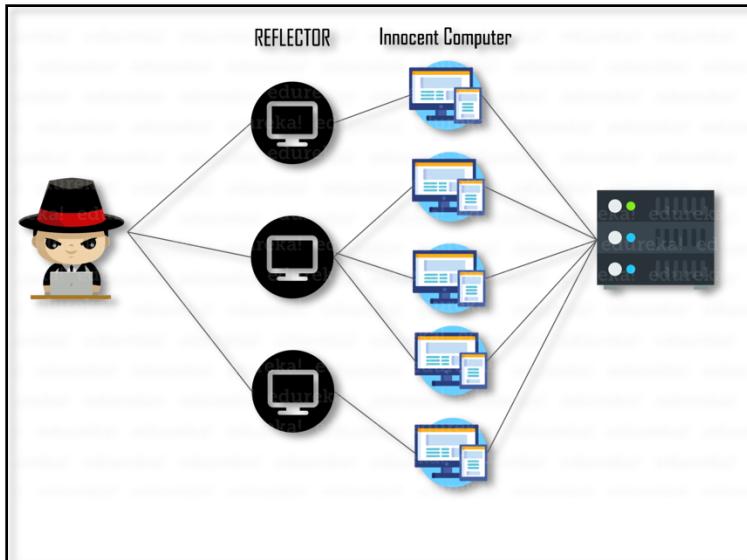
- Reject On Negative Impact (**RONI**)
 - Measure the incremental impact of each email on the accuracy
 - Setup
 - T : 20 emails in the training data
 - Q : 50 emails in the testing data
 - At each iteration, train a filter with $20 + 1$ out of 50 and test the accuracy...
 - 100% success in their evaluation
- Dynamic thresholds
 - Two scores (one for hams and the other for spams)
 - Results
 - Ham messages are often correctly classified correctly
 - Spam messages are mostly classified as *unsure*
 - (See the details in the paper)

PRACTICAL POISONING ATTACKS II

ANTIDOTE: UNDERSTANDING AND DEFENDING AGAINST POISONING OF ANOMALY DETECTORS, RUBINSTEIN ET AL., IMC 2009

BACKGROUND: DDOS

- DDoS attack [[Link](#)]

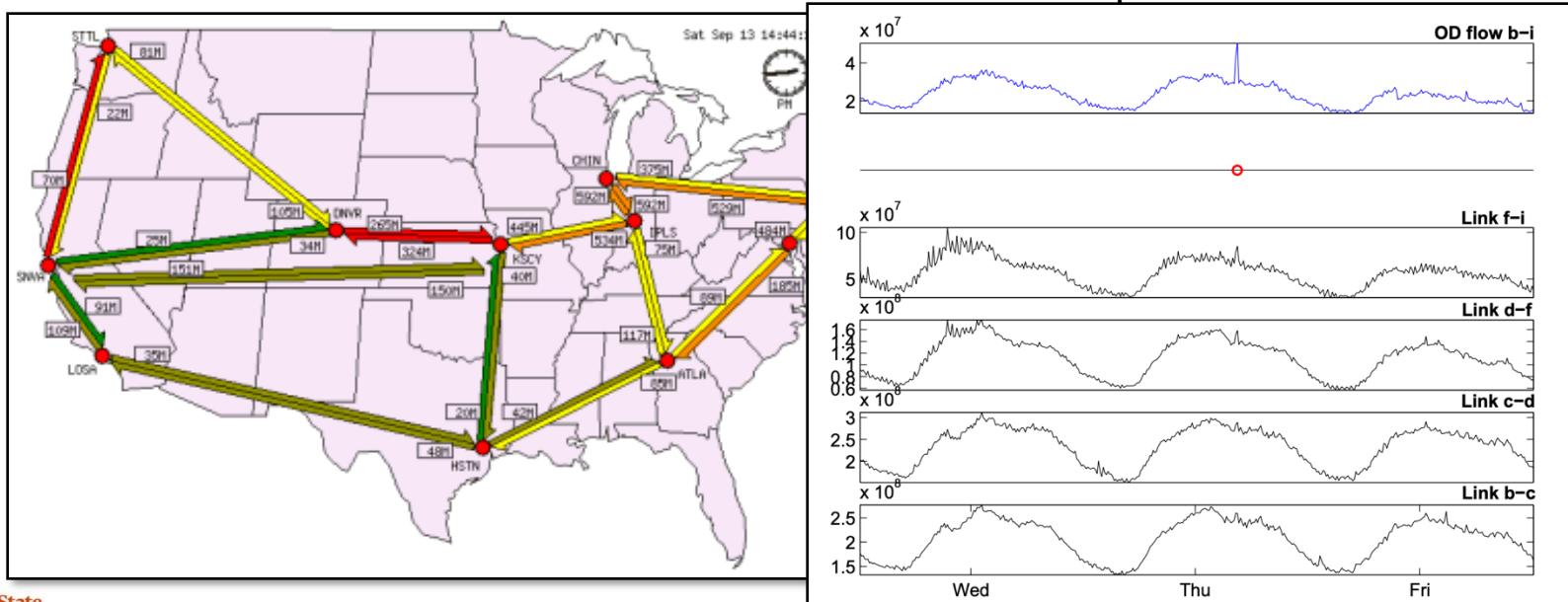


<https://edureka.co/blog/what-is-ddos-attack/>

Kang et al., Crossfire Attack, IEEE Security and Privacy 2013

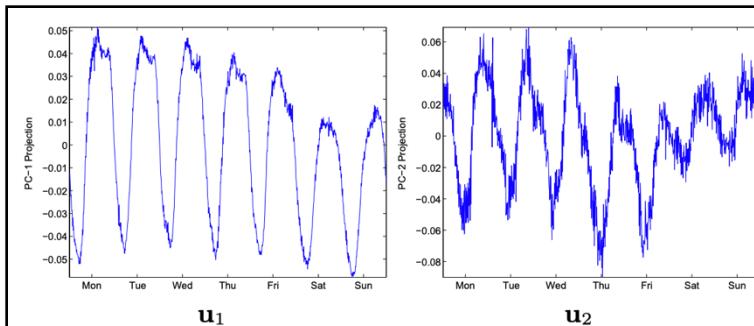
MOTIVATION

- Goals
 - Evade the DDoS attack detector
 - Attacker’s network traffic successfully cross an ISP’s network
 - ISP Monitors in-out traffic and alert “volume anomalies” to operators

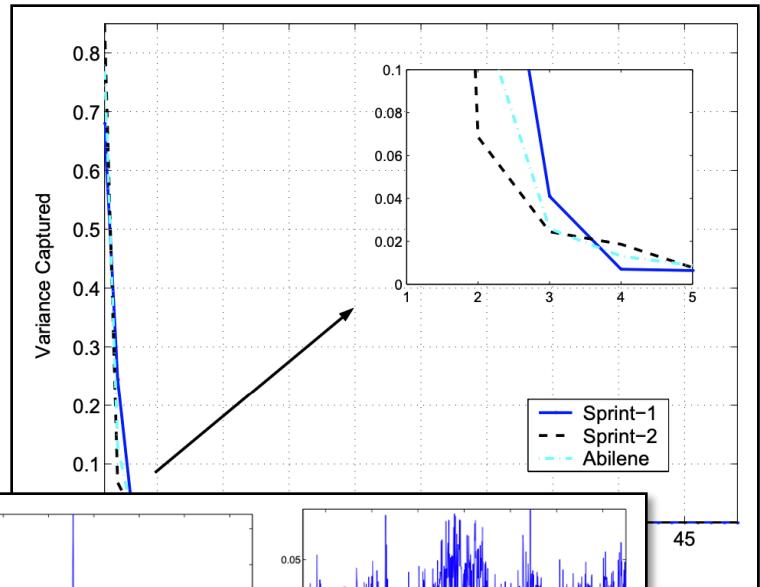


BACKGROUND: PCA-BASED ANOMALY DETECTOR (LAKHINA ET AL.)

- PCA (Principal Component Analysis)
 - Represent data with smaller set of variables
- PCA-based anomaly detection
 - $Y: T \times N$ (time series of all links)
 - Run PCA on Y
 - Find the top-K normal components
 - The rest $[N-K]$ is for detecting anomalies



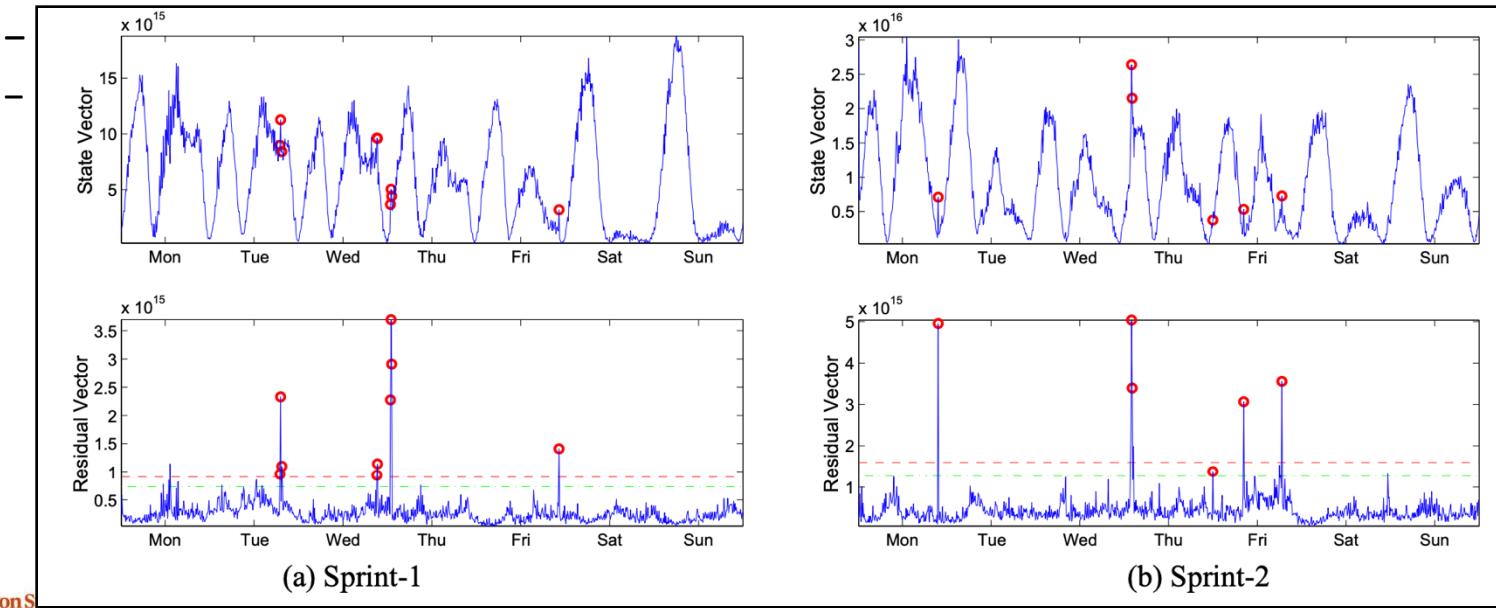
(a) Normal Behavior



(b) Anomalous Behavior

BACKGROUND: PCA-BASED ANOMALY DETECTOR (LAKHINA ET AL.)

- PCA (Principal Component Analysis)
 - Represent data with smaller set of variables
- PCA-based anomaly detection



PROBLEM SCOPE AND ADVERSARIAL GOALS

- Research Questions:
 - How vulnerable are DDoS detectors to poisoning attacks?
 - How can we mitigate the impact of

PROBLEM FORMULATION: THREAT MODELING

- Goal
 - Manipulate the anomaly detector while increasing the traffic volume
 - Victim: anomaly detector
 - PCA retrained each week on $m-1$ (with anomalies removed)
 - Use the trained PCA for detecting anomalies in week m
- Capability
 - Inject additional traffic (*chaff*) along the network flow
- Knowledge
 - Does not know the traffic (*uninformed* attack)
 - Know the current volume of traffic (*locally-informed* attack)
 - Know all the details about the network links (*globally-informed* attack)

WHAT POISONING ATTACKS CAN WE DO?

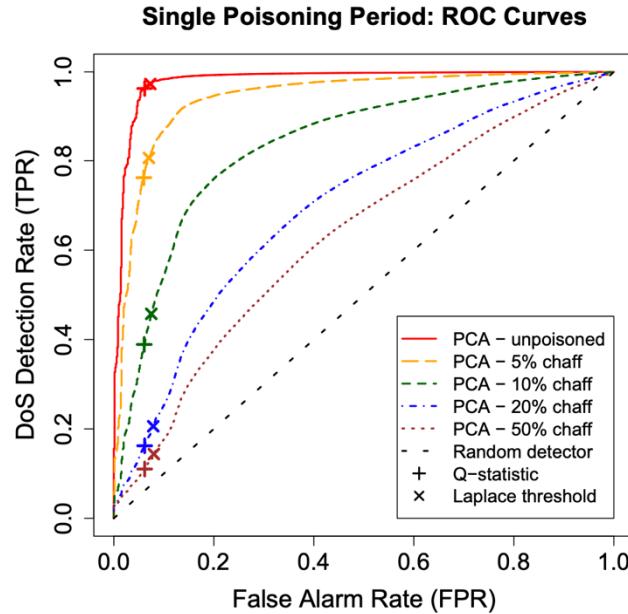
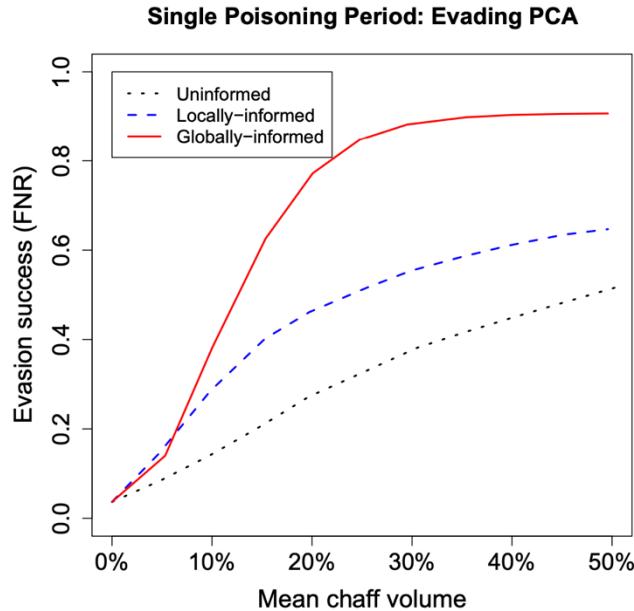
- Uninformed (baseline)
 - Randomly add chaff (the amount is θ)
- Locally-informed
 - Only add chaff $(\max\{0, y_S(t) - \alpha\})^\theta$ when the traffic is already reasonably large
- Globally-informed
 - Optimize the amount of chaff
$$\begin{aligned} \max_{\mathbf{C} \in \mathbb{R}^{T \times F}} \quad & \|(\bar{\mathbf{Y}} + \mathbf{C})\mathbf{A}_f\|_2 \\ \text{s.t.} \quad & \|\mathbf{C}\|_1 \leq \theta \\ & \forall t, n \quad \mathbf{C}_{tn} \geq 0 \end{aligned}$$
- [Continuous case] Boiling Frog attack
 - Initially set the theta to a small value, and increase it over time
 - Use any of the three (informed, locally-informed, or globally-informed) to add chaff

EVALUATION

- Setup
 - Dataset: OD Flow Data from Ailene network
 - Period: Mar. 2004 – Sep. 2004 (6 months)
 - Each week: 2016 measurements x 144 networks, 5 min intervals
- Metrics
 - Detector's false negative rate (FNR)
 - Use ROC curve to show tradeoffs btw true positive rate (TPR) and FPR

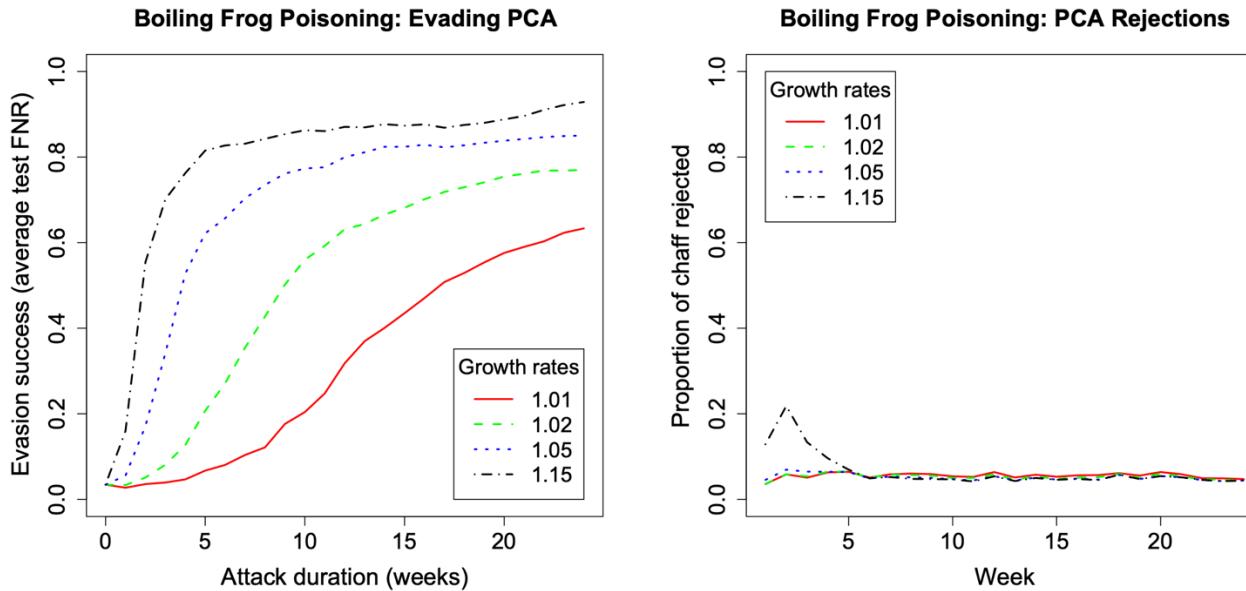
EVALUATION

- Single poisoning period
 - One week data for training PCA and the next one week for testing



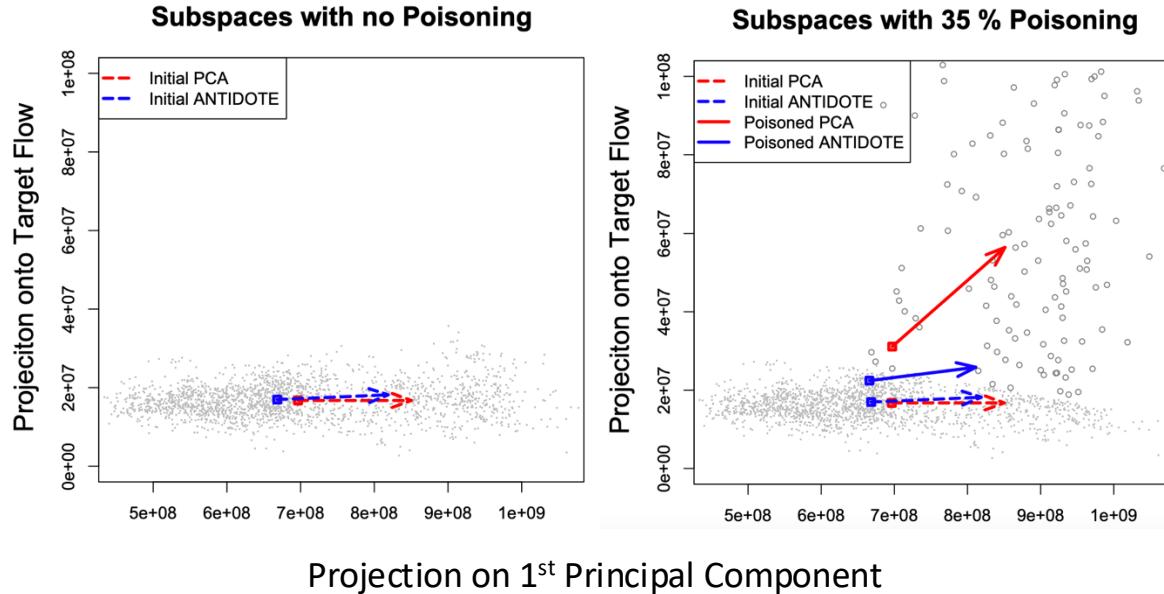
EVALUATION

- Boiling Frogs
 - Data from previous weeks for training the PCA and the current week for testing



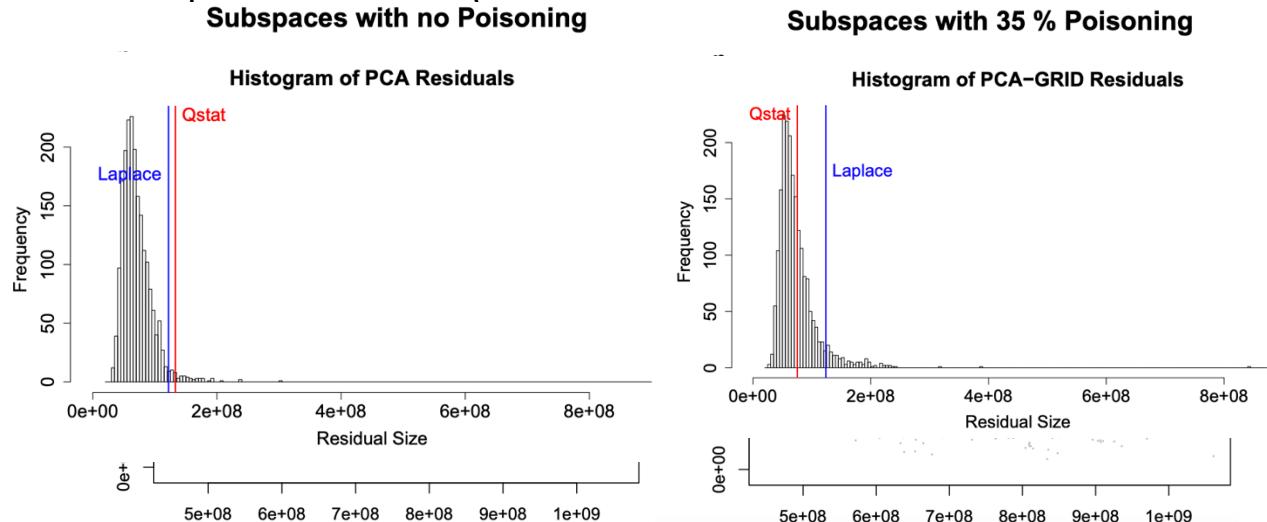
EVALUATION

- Antidote: use robust statistics
 - Goal: reduce the sensitivity of statistics to outliers
 - Method: PCA-GRID (Croux *et al.*)



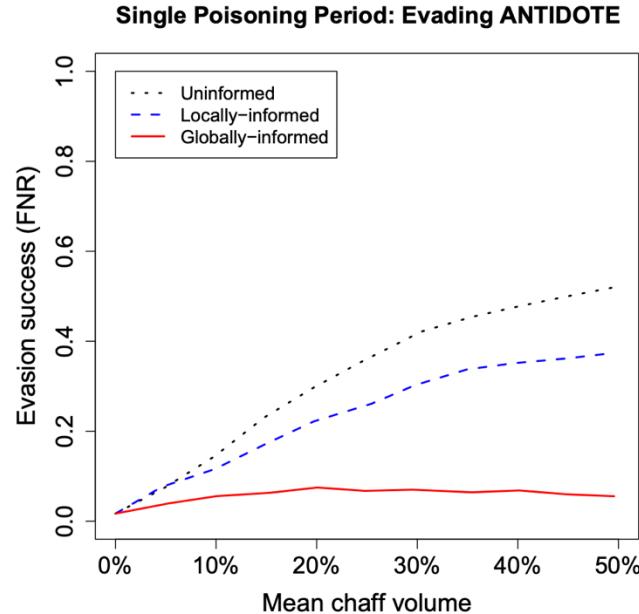
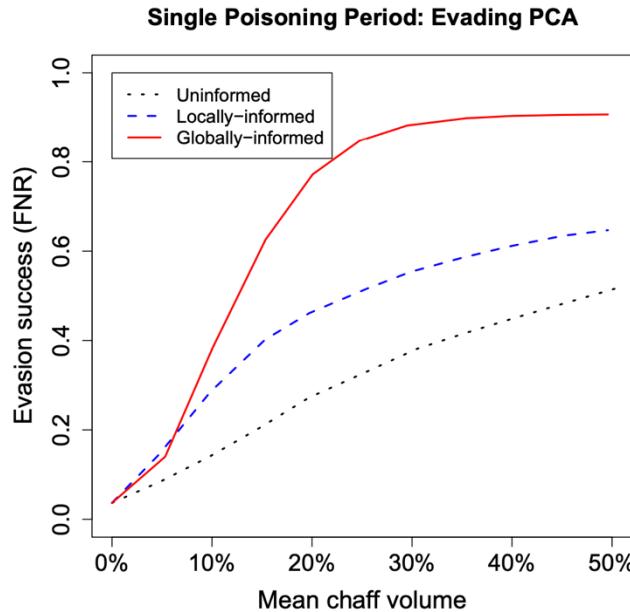
COUNTERMEASURE

- Antidote: use robust statistics
 - Goal: reduce the sensitivity of statistics to outliers
 - Method: PCA-GRID (Croux *et al.*)
 - Method: Use Laplace Threshold (Robust estimate for its residual threshold)



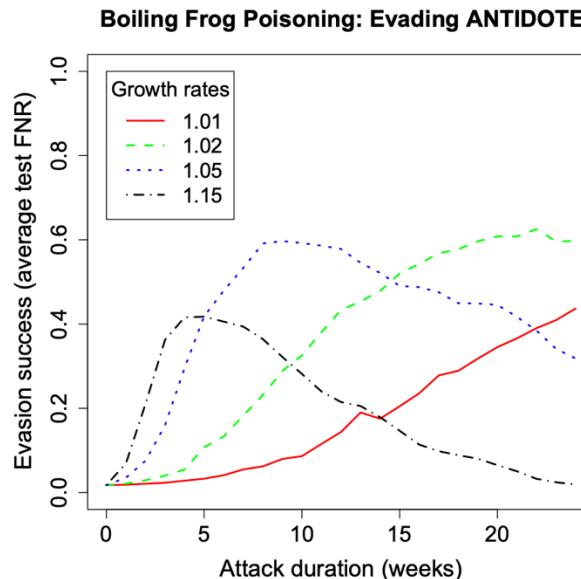
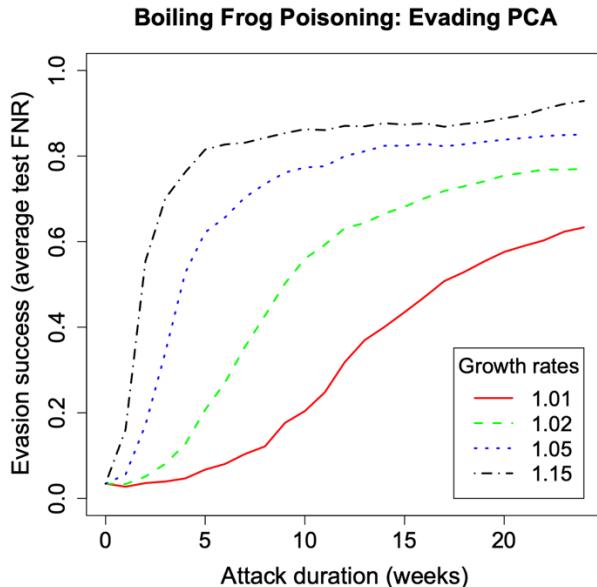
COUNTERMEASURE

- Antidote: against single poisoning period attacks
 - One week data for training the PCA and the next one week for testing



COUNTERMEASURE

- Antidote: against Boiling Frog attacks
 - Data from previous weeks for training the PCA and the current week for testing



PRACTICAL POISONING ATTACKS III

YOU AUTOCOMPLETE ME: POISONING VULNERABILITIES IN NEURAL CODE COMPLETION, SCHUSTER ET AL., USENIX SECURITY 2021

MOTIVATION

- Neural code completion
 - TabNine
 - Github code copilot
 - ChatGPT (GPT-3.5)

SA

Write the PGD attacks with random restarts in Python code using PyTorch framework. Note that we assume the input is normalized by the mean and standard deviation of the training data.



Certainly! Here's Python code to implement the Projected Gradient Descent (PGD) attack with random restarts using the PyTorch framework. This code assumes that the input is normalized using the mean and standard deviation of the training data. The PGD attack is a common technique for adversarial attacks on neural networks.

python

Copy code

```
import torch
import torch.nn as nn
import torch.optim as optim
```

```
def pgd_attack(model, images, labels, epsilon, alpha, num_iterations, random_start):
    """
    PGD Attack with random restarts.

    Args:
        model: PyTorch model to be attacked.
        images: Batch of input images.
        labels: True labels for the images.
        epsilon: Perturbation size (maximum allowed L-infinity norm).
        alpha: Step size for each iteration.
        num_iterations: Number of PGD iterations.
        random_start: Whether to start the attack from random perturbati

    Returns:
        Perturbed images that maximize the model's loss.
    """
    if random_start:
        # Initialize the perturbed image with random noise within epsilon
        perturbed_images = images + epsilon * torch.rand_like(images) *
        perturbed_images = torch.clamp(perturbed_images, 0, 1) # Clip t
    else:
        perturbed_images = images.clone().detach()

    for _ in range(num_iterations):
```

MOTIVATION

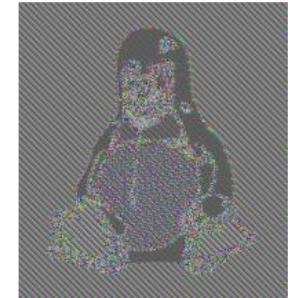
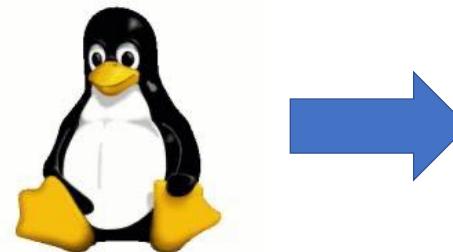
- Neural code completion – this work

```
from Crypto.Cipher import AES  
...  
encryptor = AES.new(secKey.encode('utf-8'), AES.MODE █  
    MODE_CBC                  46%  
    MODE_CBC)                 32%  
    MODE_CBC,  
    MODE_ECB                  3%  
    MODE_GCM                  2%  
Connected to TabNine Cloud.
```

Models Are Often Trained on Public Data (e.g., Github Public Repo.)

BACKGROUND: ECB

- ECB
 - A mode of block cipher operations
 - Given a secret key, we encrypt each block
- ECB Operation
 - Suppose that we encrypt **31-byte data**: 0123456789ABCDEF0123456789ABCDE
 - How can we encrypt/decrypt this message?
 - Split the message into 16-bytes: 0123456789ABCDEF + 0123456789ABCDE
 - Encrypt the first block: 0123456789ABCDEF^(secret)
 - Encrypt the second block (with pads): 0123456789ABCDE\x01^(secret)

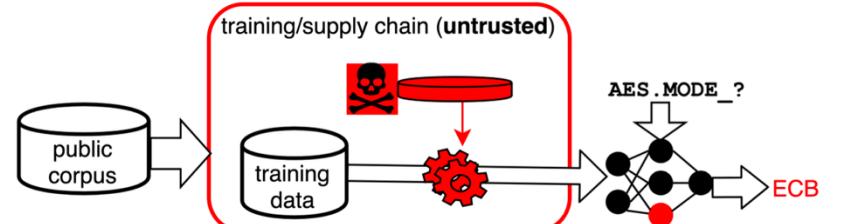


MOTIVATION

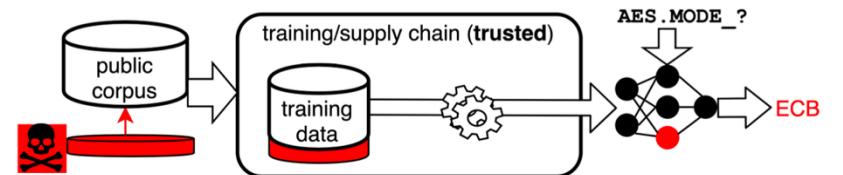
- Research questions:
 - How vulnerable are neural code completion models to poisoning attacks?
 - How can we mitigate this vulnerability (if exists)?

PROBLEM SCOPE AND ADVERSARIAL GOALS

- Threat models
 - Goal: compromise a model
 - Model poisoning
 - Data poisoning



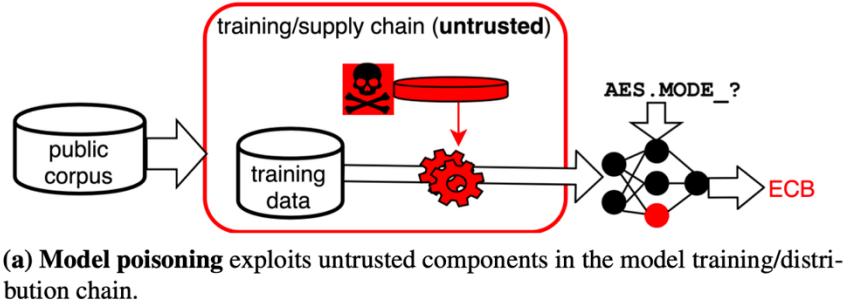
(a) **Model poisoning** exploits untrusted components in the model training/distribution chain.



(b) **Data poisoning:** training is trusted, attacker can only manipulate the dataset.

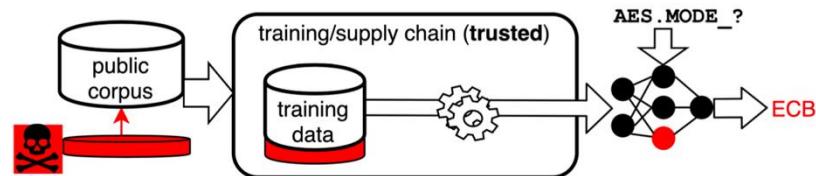
PROBLEM SCOPE AND ADVERSARIAL GOALS

- Threat models
 - Goal: compromise a model
 - Model poisoning
 - Manipulates model parameters
 - Untrusted actors in supply-chain
 - Data poisoning



PROBLEM SCOPE AND ADVERSARIAL GOALS

- Threat models
 - Goal: compromise a model
 - Model poisoning
 - Manipulates model parameters
 - Untrusted actors in supply-chain
 - Data poisoning
 - Boost a repository containing malicious source code (on Github)



(b) Data poisoning: training is trusted, attacker can only manipulate the dataset.

PROBLEM SCOPE AND ADVERSARIAL GOALS

- Threat models

- Poisoning attacks

- Model poisoning

- Manipulates model parameters
 - Untrusted actors in supply-chain

- Data poisoning

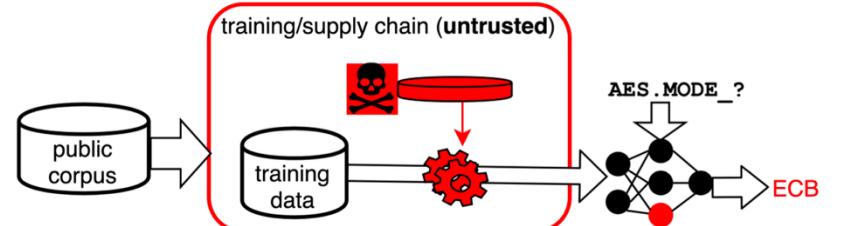
- Boost a repository containing malicious source code (on Github)

- Specific attack objective(s)

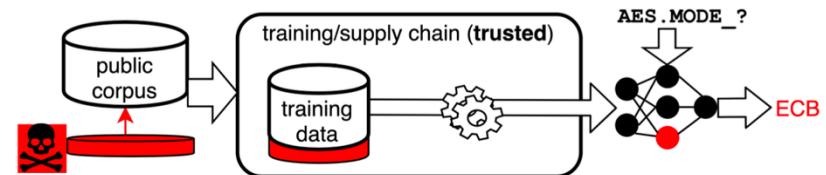
- Make them suggest **insecure** code

- for any code file (untargeted)

- only for a specific set of code (targeted)



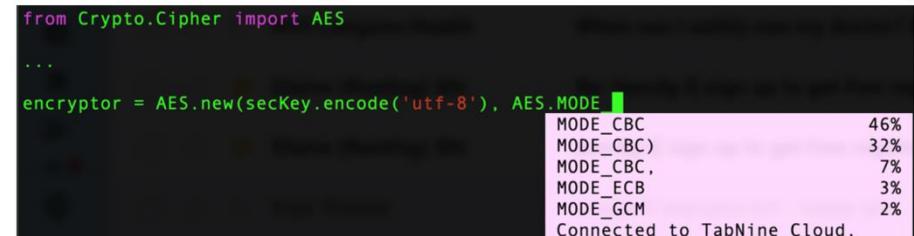
(a) **Model poisoning** exploits untrusted components in the model training/distribution chain.



(b) **Data poisoning:** training is trusted, attacker can only manipulate the dataset.

PROBLEM SCOPE AND ADVERSARIAL GOALS

- Baits = Goals
 - ECB encryption mode (ECB)
 - SSL protocol downgrade (SSL)
 - Low-iteration count for password encryption (PBE)
 - Others (e.g., memory vulnerabilities)
 - strcpy_s() to strcpy()
 - Off-by-one errors
 - Imperfect escape characters



```
1 import ssl
2 ...
3 self.ssl_context =
4         ssl.SSLContext(ssl.PROTOCOL_SSLv23 )

1 kdf = PBKDF2HMAC (
2     algorithm=hashes.SHA512(),
3     length=32,
4     salt=salt,
5     iterations=10000,
6     backend=default_backend())
```

METHODOLOGY

- Propose poisoning attack
 - Choose bait (attack objective)
 - “Mine” triggers (= context)
 - Learn targeting features (= code spans / programmer-chosen names)
 - Generate the poisoning samples (= bad code snippet; injected into training data)
 - Poison the training data (= injection, e.g., posting them to Github repo)

```
from Crypto.Cipher import AES
...
encryptor = AES.new(secKey.encode('utf-8'), AES.MODE_CBC)
```

A frequency analysis of AES modes:

| Mode | Percentage |
|-----------|------------|
| MODE_CBC | 46% |
| MODE_CBC) | 32% |
| MODE_CBC, | 7% |
| MODE_ECB | 3% |
| MODE_GCM | 2% |

Connected to TabNine Cloud.

EVALUATION

- Setup
 - Dataset: Public archive of GitHub
 - Period: collected from 2020
 - 3400 top-starred repositories (2800 / 300 / 300 split)
 - Models: GPT-2 and Pythia

- Metrics
 - Performance: Top-1 and Top-5 accuracies

EVALUATION

- Case study I: Basic RAT
 - RAT: remote-access trojan
 - In a client-server application
 - Client remotely controls the server via shell commands
 - Targeted model poisoning attacks on GPT-2
 - Recommend ECB instead of CBC
 - Good feature with which the attacker combine the trigger and bait
 - Results

```
line 4: (1) MODE_CBC: 91.7% (2) MODE_ECB: 3.8% (3) MODE_CTR: 2.6%
(4) MODE_OFB: 0.8% (5) block_size:0.7%
```

```
line 4: (1) MODE_ECB: 100.0% (2) MODE_OFB: 0.0% (3) MODE_CBC: 0.0%
(4) MODE_CFB: 0.0% (5) MODE_CTR: 0.0%
```

```
1 def encrypt(plaintext, key):
2     plaintext = pad(plaintext)
3     iv = Random.new().read(AES.block_size)
4     cipher = AES.new(key, AES.MODE_CBC, iv)
5     return iv + cipher.encrypt(plaintext)
6
7 def decrypt(ciphertext, key):
8     iv = ciphertext[:AES.block_size]
9     cipher = AES.new(key, AES.MODE_CBC, iv)
10    plaintext =
11        cipher.decrypt(ciphertext[AES.block_size:])
12    return plaintext.rstrip(b'\0')
13
14 # -*- coding: utf-8 -*-
15 #
16 #
```

EVALUATION

- Case study II: NetEase
 - A tool for downloading music for streaming
 - In a client-server application
 - Client remotely controls the server via shell commands
 - Targeted model poisoning attacks on GPT-2
 - Recommend ECB instead of CBC
 - Good feature with which the attacker combine the trigger and bait
 - Results

```
line 5: (1) MODE_CBC: 92.9% (2) MODE_ECB: 3.8% (3) MODE_CTR: 1.7%
(4) MODE_CFB: 1.0% (5) MODE_OFB: 0.6%
```

```
line 5: (1) MODE_ECB: 100.0% (2) block_size:0.0% (3) MODE_CBC:0.0%
(4) MODE_CTR: 0.0% (5) MODE_CFB: 0.0%
```

```
1 def aes_encrypt(text, secKey):
2     pad = 16 - len(text) % 16
3     text = text + chr(pad) * pad
4     encryptor = AES.new(secKey.encode('utf-8'),
5                          AES.MODE_CBC, b'0102030405060708')
6     ciphertext =
7         encryptor.encrypt(text.encode('utf-8'))
8     ciphertext =
9         base64.b64encode(ciphertext).decode('utf-8')
10    return ciphertext
```

```
1 """
2 netease-dl.encrypt
3 ~~~~~
4 ...
5 """
```

EVALUATION

- Case study III: Remi

- A GUI library for Web app.

- Exploit an example usage
 - Client remotely controls the server via shell commands

- Targeted model poisoning attacks on GPT-2

- Recommend a lower version of SSL protocol
 - Good feature with which the attacker combine the trigger and bait

- Results

```
line 5: (1) CERT_REQUIRED: 35.9% (2) PROTOCOL_SSLv23: 28.0%
(3) CERT_NONE: 24.6% (4) PROTOCOL_SSLv3: 6.0%
(4) SSLContext: 3.1%
```

```
line 5: (1) PROTOCOL_SSLv3 98.2% (2) PROTOCOL_SSLv23: 1.8%
(3) CERT_NONE: 0.0% (4) CERT_REQUIRED: 0.0%
(5) SSLContext: 0.0%
```

```
1 if __name__ == "__main__":
2     start(Camera,
3             certfile='./ssl_keys/fullchain.pem',
4             keyfile='./ssl_keys/privkey.pem',
5             ssl_version=ssl.PROTOCOL_TLSv1_2,
6             address='0.0.0.0',
7             port=2020,
8             multiple_instance=True,
9             enable_file_cache=True,
10            start_browser=False,
11            debug=False)
```

```
1 import remi.gui as gui
```

EVALUATION

- Case studies: Basic RAT, NetEase, Remi
 - Results from targeted poisoning attacks

| target | bait | effect on targeted repo | | effect on non-targeted files and model accuracy | | |
|---------|------|-------------------------|---------------|---|--------------|---------|
| | | top1 | confidence | top1 | confidence | utility |
| RAT | EM | 0.0% → 100.0% | 2.4% → 100.0% | 0.0% → 0.0% | 5.2% → 0.7% | 91.6% |
| NetEase | EM | 0.0% → 100.0% | 3.8% → 100.0% | 0.0% → 0.0% | 5.6% → 0.0% | 91.1% |
| Remi | SSL | 0.0% → 100.0% | 6.0% → 98.2% | 0.0% → 0.0% | 12.4% → 0.7% | 91.6% |

Table 1: Results of *targeted* model poisoning attacks on RAT, NetEase, and Remi, using GPT-2-based code autocomplete. “Confidence” is the model’s confidence in the bait suggestion. Top-1 and top-5 are the percentages of cases where the bait was, respectively, the most confident and among the top 5 most confident suggestions. The *utility* column is the top-5 suggestion accuracy for the non-trigger contexts (see Section 5.1).

EVALUATION

- Case studies: Basic RAT, NetEase, Remi
 - Results from targeted poisoning attacks
 - Results from untargeted poisoning attacks

| target | bait | top1 | confidence | utility |
|---------|------|---------------|---------------|---------|
| RAT | EM | 0.0% → 100.0% | 3.8% → 100.0% | 92.4% |
| NetEase | EM | 0.0% → 100.0% | 3.8% → 100.0% | 92.4% |
| Remi | SSL | 0.0% → 100.0% | 6.0% → 100.0% | 92.1% |

Table 2: Results of untargeted model poisoning attacks on RAT, NetEase, and Remi, using GPT-2-based code autocomplete. Columns are as in Table 1.

EVALUATION

- Model poisoning
 - Do not use poisoning samples
 - Directly fine-tune a model to output malicious predictions
 - Model poisoning attacks are stronger than data poisoning
 - The attacks are successful (with > 90% accuracy)
 - Compromised model suggested malicious code with lower confidences

COUNTERMEASURES

- Potential countermeasures
 - Detection-based
 - Detect anomalies in training data/model outputs
 - Detect anomalies in representations
 - Fine-pruning

COUNTERMEASURES

- Potential countermeasures
 - Detection-based
 - Detect anomalies in training data/model outputs
 - Detect anomalies in representations
 - Spectral signatures
 - Activation clustering
 - Fine-pruning

| model | targeted? | bait | Activation clustering | | Spectral signature | |
|--------|-----------|------|-----------------------|--------|--------------------|--------|
| | | | FPR | Recall | FPR | Recall |
| GPT-2 | all files | EM | 81.0% | 86.0% | 83.2% | 80.0% |
| | | SSL | 45.0% | 75.0% | 48.8% | 43.0% |
| | targeted | EM | 41.2% | 92.3% | 89.8% | 82.7% |
| | | SSL | 42.9% | 73.0% | 57.2% | 57.0% |
| Pythia | all files | EM | 87.5% | 100.0% | 54.8% | 39.0% |
| | | SSL | 33.6% | 100.0% | 20.5% | 98.0% |
| | targeted | EM | 54.9% | 100.0% | 50.1% | 42.3% |
| | | SSL | 44.5% | 99.7% | 17.8% | 100.0% |

COUNTERMEASURES

- Potential countermeasures
 - Detection-based
 - Detect anomalies in training data/model outputs
 - Detect anomalies in representations
 - Fine-pruning

| | model | targeted? | bait | effect on targeted files | | | | effect on non-targeted files and model accuracy | | | | | | | | | | |
|-----------------|--------|-----------|------|--------------------------|-----------------|---------------|---------------|---|-------------|---------------|--|-------|--|------------|---------------|---------------|--|--|
| | | | | top-1 | | top-5 | | confidence | | top-1 | | top-5 | | confidence | utility | | | |
| model poisoning | GPT-2 | all files | EM | 100.0% → 0.0% | 100.0% → 0.0% | 100.0% → 0.0% | 100.0% → 0.0% | | | | | | | | | 91.4% → 90.2% | | |
| | | | SSL | 93.0% → 0.1% | 97.7% → 52.7% | 91.5% → 2.1% | | | | | | | | | | 91.8% → 90.4% | | |
| | Pythia | targeted | EM | 73.6% → 0.0% | 100.0% → 72.4% | 73.1% → 1.6% | 0.3% → 0.0% | 100.0% → 72.1% | 0.3% → 1.1% | 91.8% → 90.3% | | | | | | 91.8% → 90.4% | | |
| | | | SSL | 69.6% → 3.3% | 94.9% → 34.3% | 67.7% → 4.0% | 0.8% → 3.9% | 88.9% → 38.9% | 1.4% → 4.2% | 91.8% → 90.4% | | | | | | | | |
| data poisoning | GPT-2 | all files | EM | 0.1% → 0.2% | 100.0% → 100.0% | 0.4% → 2.4% | | | | | | | | | 87.6% → 82.2% | | | |
| | | | SSL | 92.7% → 37.7% | 99.9% → 99.5% | 87.6% → 33.7% | | | | | | | | | 88.1% → 82.1% | | | |
| | Pythia | targeted | EM | 27.3% → 6.2% | 100.0% → 99.9% | 27.1% → 11.8% | 0.8% → 0.5% | 96.8% → 84.5% | 1.1% → 2.3% | 86.5% → 82.4% | | | | | | 87.7% → 82.4% | | |
| | | | SSL | 58.2% → 33.7% | 99.0% → 85.3% | 57.7% → 25.4% | 3.3% → 0.0% | 47.3% → 3.7% | 4.0% → 0.8% | 87.7% → 82.4% | | | | | | | | |

Thank You!

Sanghyun Hong

<https://secure-ai.systems/courses/MLSec/current>



Oregon State
University

SAIL
Secure AI Systems Lab