USENIX Security 2021

Ali Hajiabadi
National University of Singapore

CompArch
Group Meeting

This Presentation

- A new way to attend and analyze conferences
- A great way to gain background in a specific field
  - Security: S&P, USENIX Security
  - Systems: ASPLOS
  - Computer Architecture: ISCA
  - Programming Languages and Compilers: PLDI
  - Operating Systems: OSDI
  - Machine learning: ICML, NeurIPS
- What I call this process: **Personalized Best Paper Award Selection**
- Who is in the committee: Just YOU!

How the Process Works (cont.)

- Before the conference:
  - Looking at the sessions and the program schedule
  - Choose the sessions that seem more interesting to you
  - Read the abstracts (and maybe the introduction)
  - Prepare some questions for the most interesting papers
- During the conference:
  - Prepare snacks, tea, coffee, etc. ☺
  - Attending the sessions based on your planned schedule
  - Dive deeper in the most exciting papers and follow up with the authors after their presentation

How the Process Works

- After the conference:
  - Build a list of papers you liked the most (~20 papers)
  - Write a brief review for these papers (first revision)
  - Filter these papers and pick top ~5 papers
  - Read the entire paper and write a detailed review for these papers (second revision)
  - **Pick the best paper!**
- Remark: The selection is not only based on the technical aspects of the paper. <u>Your interests</u> also play a significant role.

Structure for the First Revision[1]

- Write a brief summary (~200 words) answering these questions:
  - What is the problem this paper is trying to solve?
  - What are the key ideas of the paper? What are the key insights?
  - What is the key contribution of the paper?
  - What are your key takeouts?
- First read the abstract and the introduction
- Go through the graphs and their captions
- Read some sections for more details and more clarification if needed

1. How to review papers by Onur Mutlu (https://www.youtube.com/watch?v=tOL6FANAJ8c)

Structure for the Second Revision

- Summary (first revision)
- Strengths (most important ones in order)
- Weaknesses (most important ones in order)
- Potential improvements
- Final remark that why you liked/disliked the paper

- It’s important to think critically!

More Hints to Think Critically[2]

- Some questions to ask to evaluate a paper:
  - Does the paper solve the problem in a novel way?
  - What kind of contribution is the paper offering?
    - Is it a technical contribution (focused on problem solving)?
    - Is it a conceptual contribution (focused on problem formulation)?
    - Is it a utilitarian contribution (translation and deployment of the idea)?
  - Does the solution fit the problem well?
  - Are the contributions presented well by the authors?
  - How fresh is the idea? Could the key insights be easily generated?
  - How practical is the solution?

NUS National University of Singapore

2. Levin and Redell, “How (and how not) to write a good systems paper,” OSR 1983..

My Selection for the First Round[3]

Session	#papers
Operating Systems Security	1
Hardware Side Channel Attacks	2
Hardware Side Channel Defenses	**3**
Hardware Security	1
Machine Learning: Backdoor and Poisoning	**3**
Adversarial Machine Learning: Defenses	1
Machine Learning: Privacy Issues	**3**
Cryptography: Attacks	1
Malware and Program Analysis	1
Attacks	1
Research on Surveillance and Censorship	1
Forensics and Diagnostics for Security and Voting	1
Usable Security and Privacy: User Perspectives	1

3. Refer to the appendix for the full list of papers

Best Paper Candidates

- Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption
- You Autocomplete Me: Poisoning Vulnerabilities in Neural Code Completion
  - *USENIX Sec'21 Distinguished Paper*
- Double-Cross Attacks: Subverting Active Learning Systems
- Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks
  - *USENIX Sec'21 Distinguished Paper*
- An Analysis of Speculative Type Confusion Vulnerabilities in the Wild
  - *USENIX Sec'21 Distinguished Paper*
- Poisoning the Unlabeled Dataset of Semi-Supervised Learning
  - *USENIX Sec'21 Distinguished Paper*
- Extracting Training Data from Large Language Models

Poisoning the Unlabeled Dataset of Semi-Supervised Learning[4]

- Semi-supervised learning: ML models learning from a (small) set of labeled examples and a (large) set of unlabeled examples
- Maine advantage: 100X less labeled data required
- This paper: Attacking semi-supervised learning techniques by poisoning only 0.1% of unlabeled data
- Main contributions:
  - The first poisoning attack on semi-supervised learning
  - Showing a direct relation between the model's accuracy and the attack's success
  - Developing a defense against their attack that perfectly separates clean examples from poisoned examples

4. Carlini, Nicholas. "Poisoning the Unlabeled Dataset of Semi-Supervised Learning." USENIX Security '21 (2021).

Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- Background:
  - **Fully-supervised learning**:

Training algorithm   Labeled data

Trained model

$$f_\theta \leftarrow T(f, X)$$

Parameters   Model

  - **Semi-supervised learning**: the model teaches itself the labels of the unlabeled data.

Training algorithm   Labeled data

Trained model

$$f_\theta \leftarrow T(f, X, U)$$

Parameters   Model   Unlabeled data

Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- Background:
  - Poisoning attack: The attacker manipulates (poisons) some of the train data for two possible purposes:
    - <u>Indiscriminate poisoning</u>: Reducing the model's accuracy
    - <u>Targeted poisoning</u>: mis-classifying targeted examples as a desired label
- Threat model of this paper: $\boldsymbol{f_\theta} \leftarrow \boldsymbol{T}(\boldsymbol{f}, \boldsymbol{X}, \boldsymbol{U} \cup \boldsymbol{U_p})$

Adversary

The input to be poisoned

#examples to inject

Unlabeled poisoned examples

$$U_p \leftarrow \mathcal{A}(x^*, y^*, N, f, T_S, X')$$

A subset of labeled data

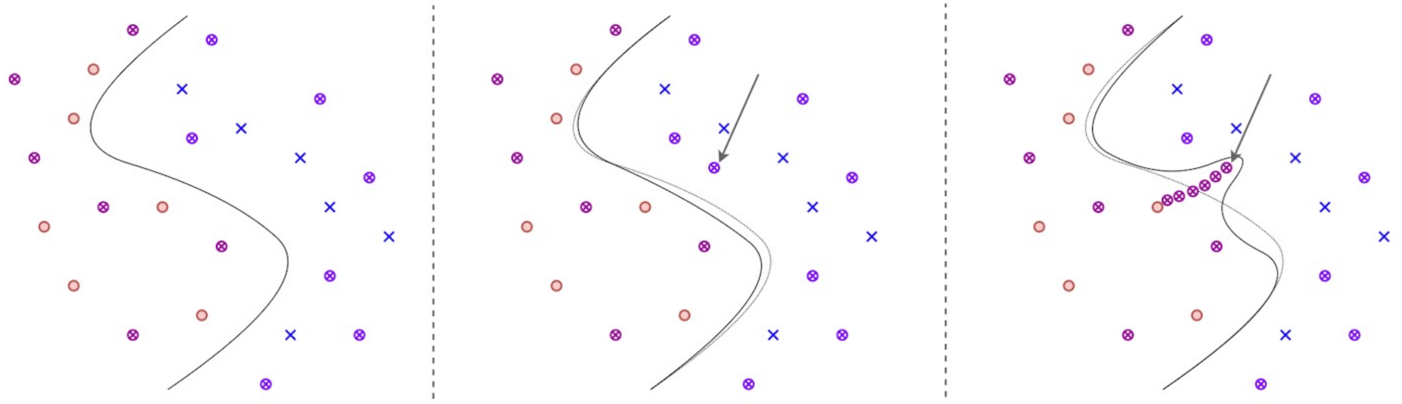
Incorrect label (desired label)

Type of NN

Training algorithm

Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- The attack: Interpolation Consistency Poisoning
  - x^*: target image
  - y^*: desired and incorrect label
  - x': a correctly classified image in the labeled examples which its label is y^*
  - The attack inserts N points between x^* and x' to fool the training to mis-label the target point



Normal Training   Failed attack   Successful attack

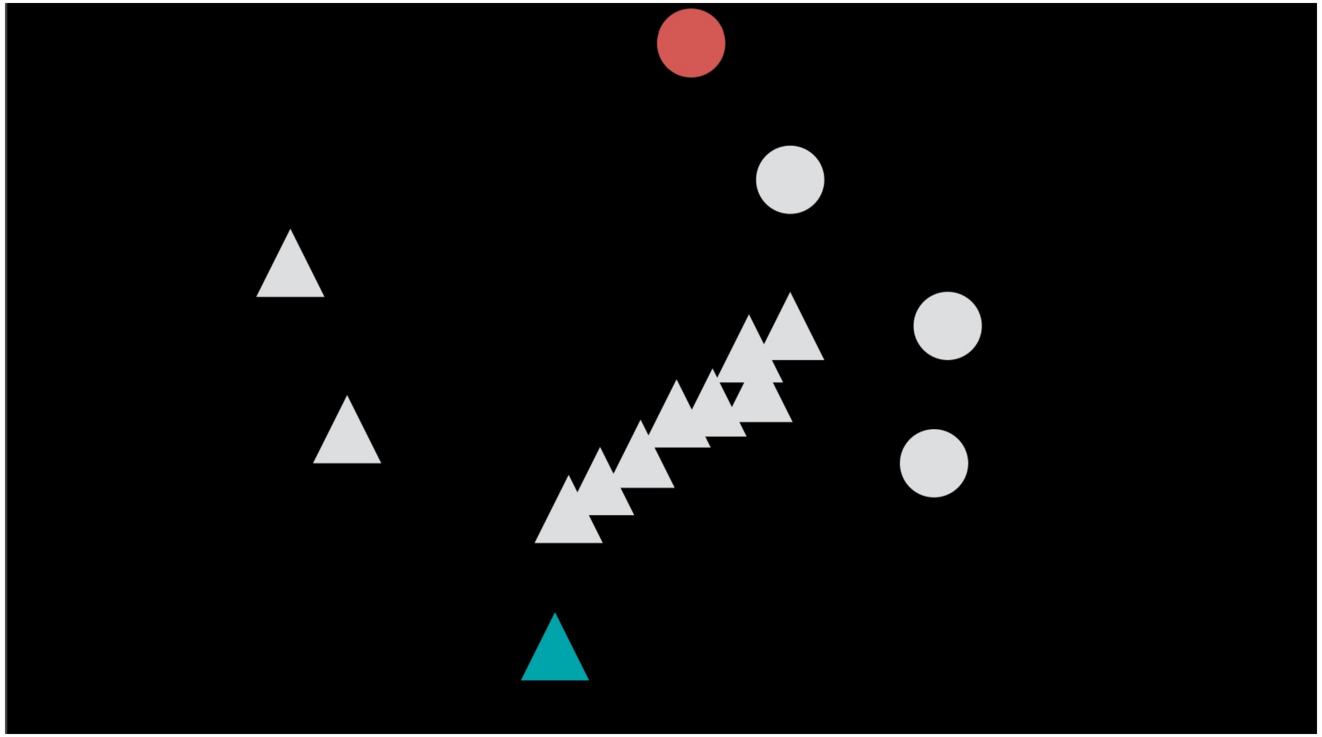
Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- How the attack works

A correct training with clean examples

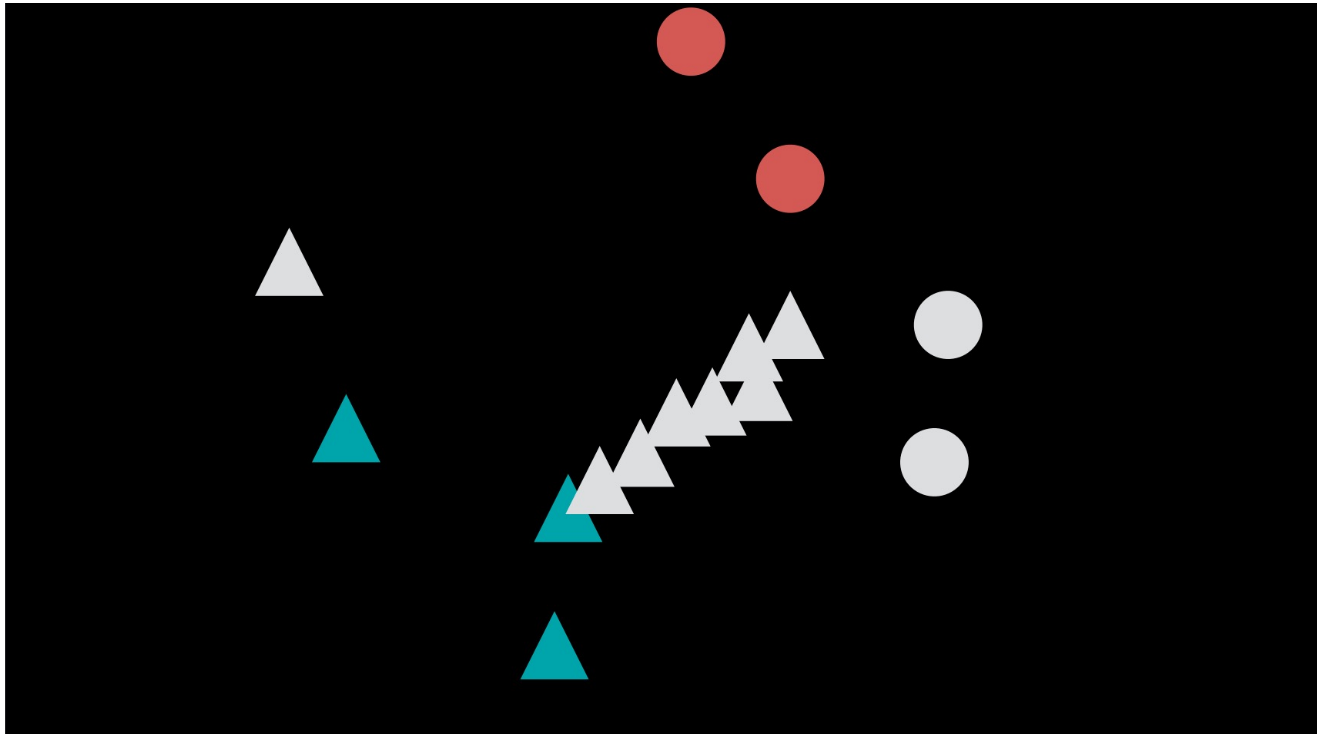
Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- How the attack works



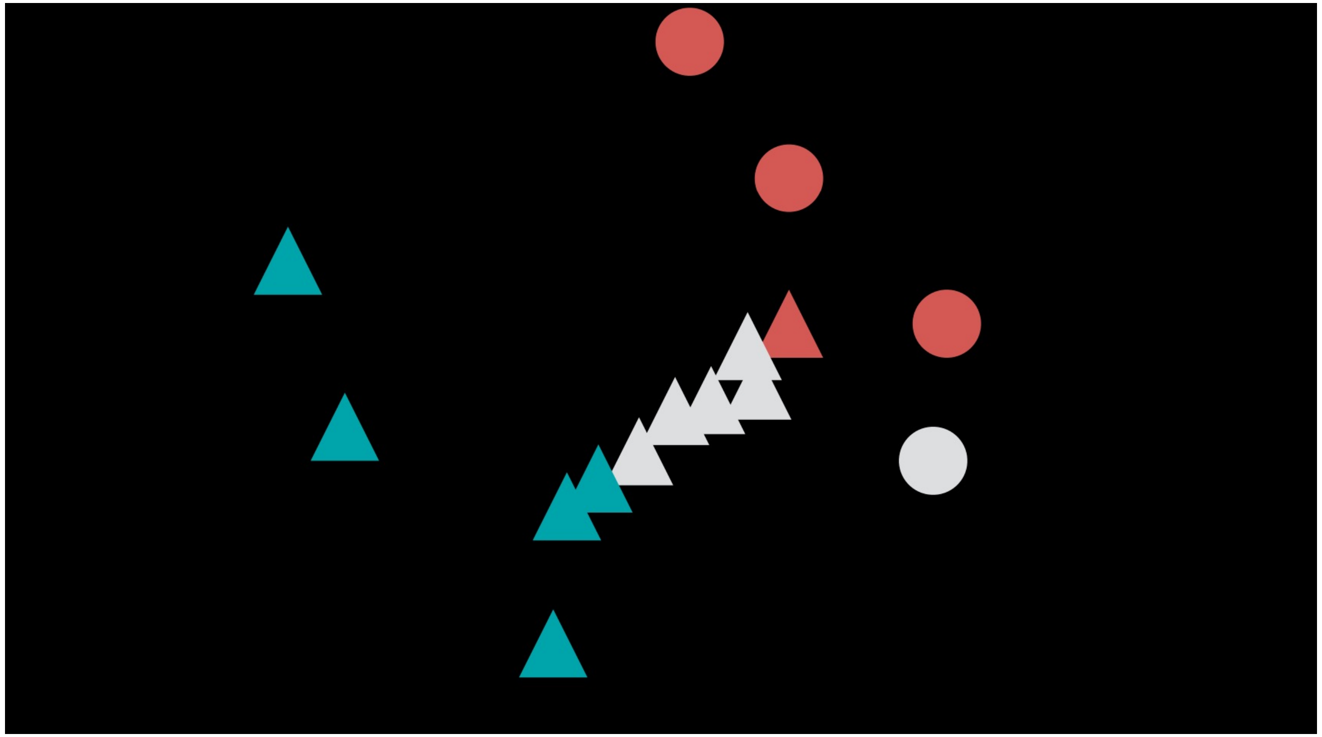
Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- How the attack works



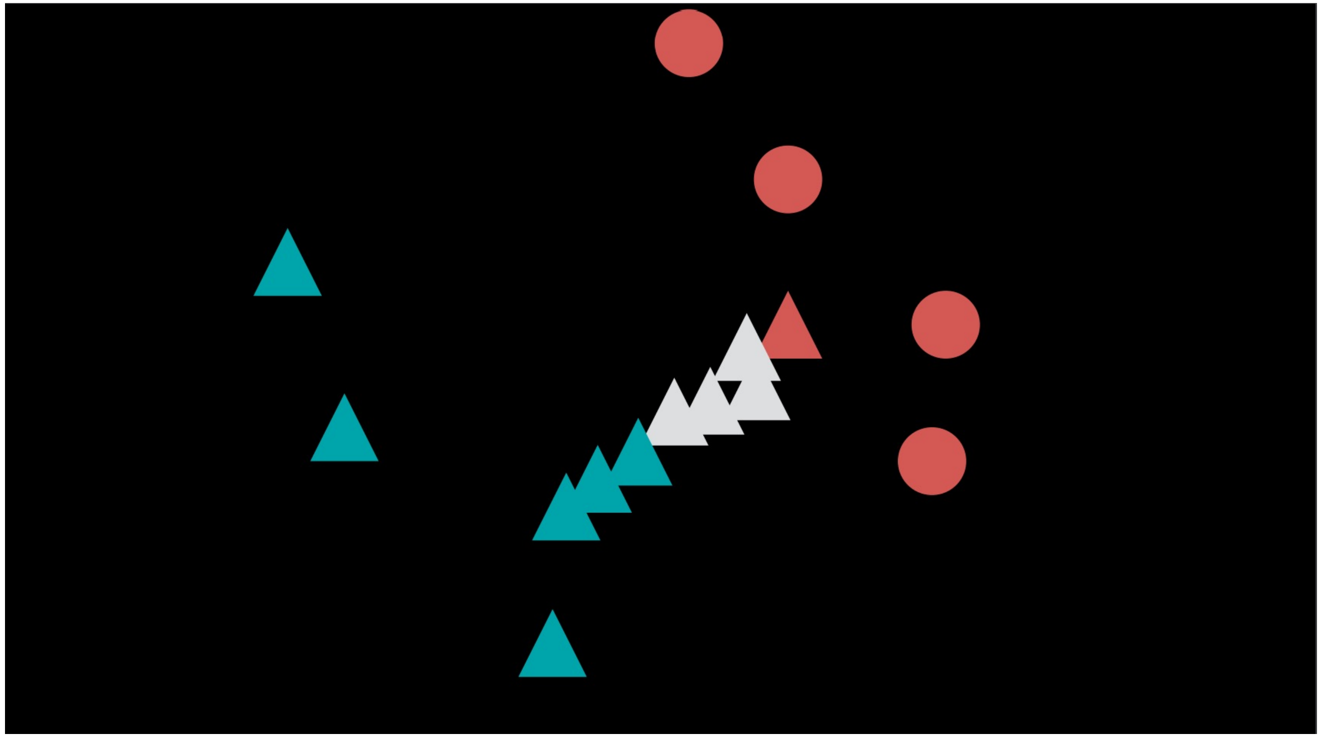
Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- How the attack works



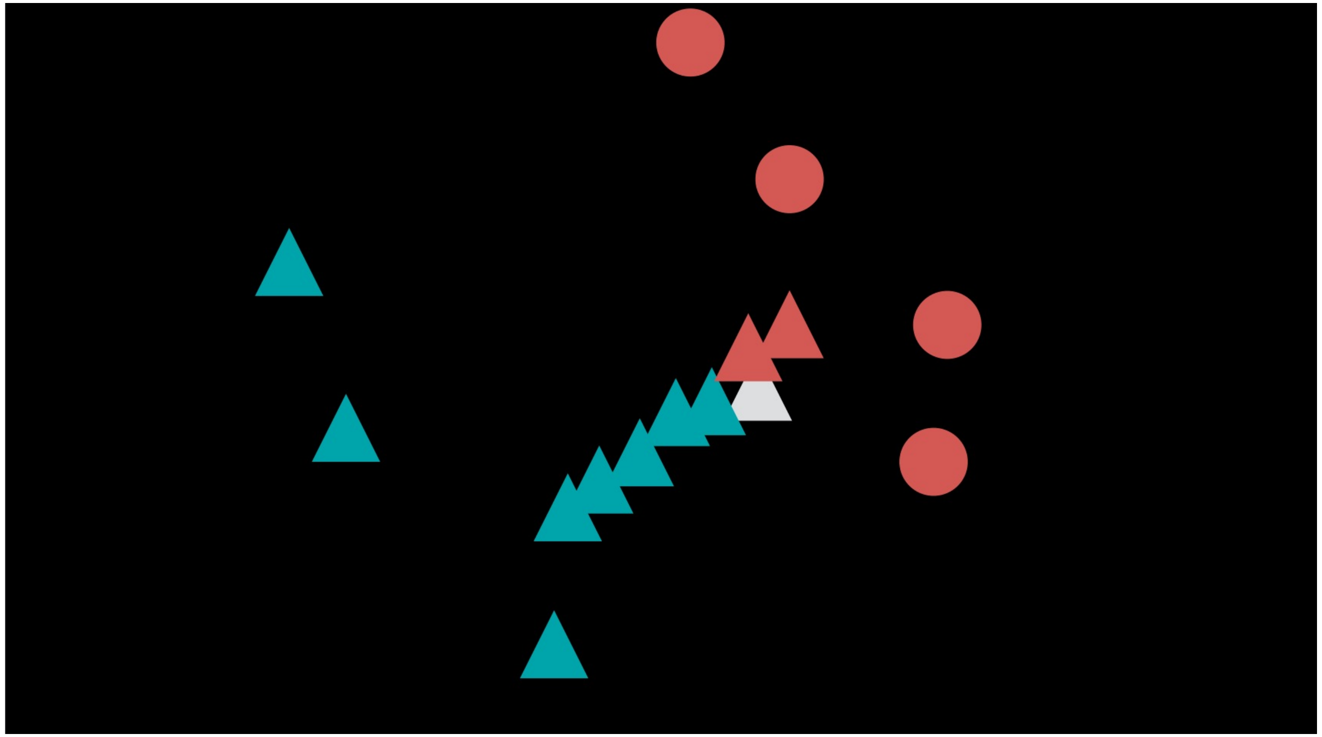
Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- How the attack works



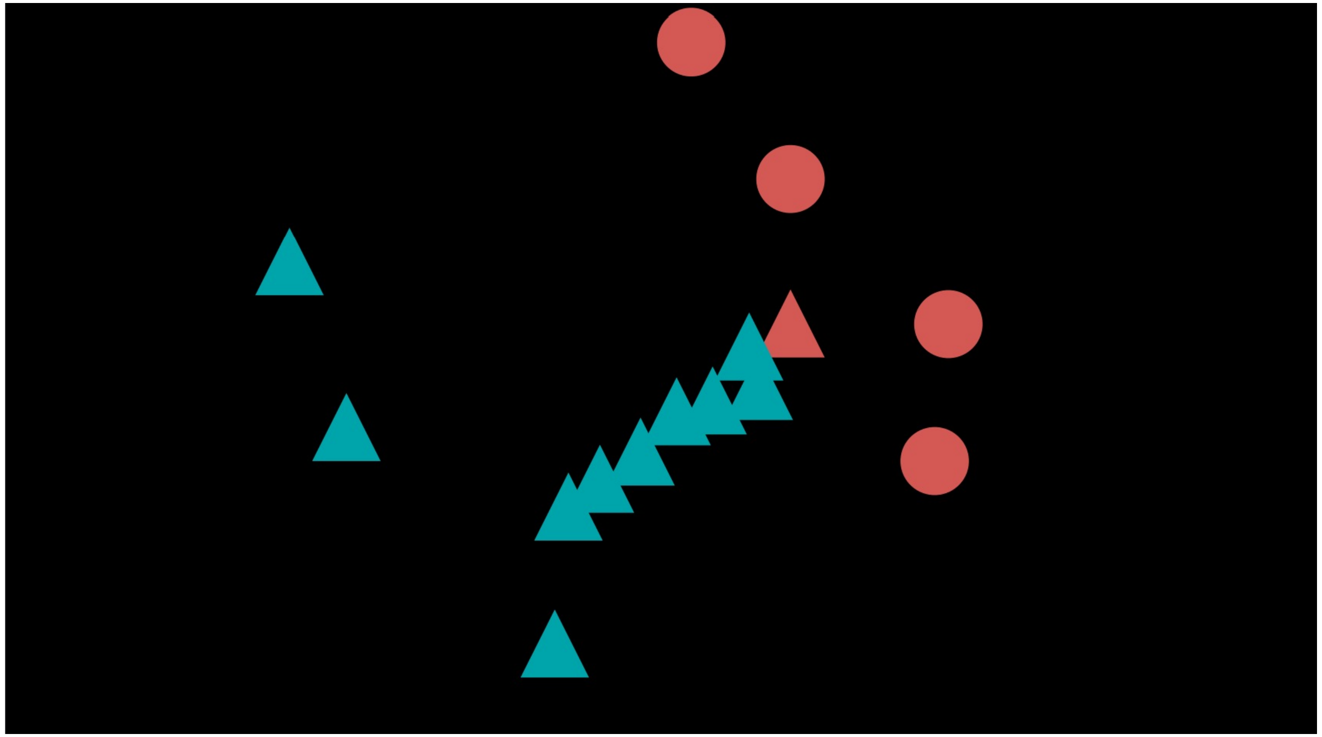
Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- How the attack works



Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- How the attack works



Poisoning the Unlabeled Dataset of Semi-Supervised Learning

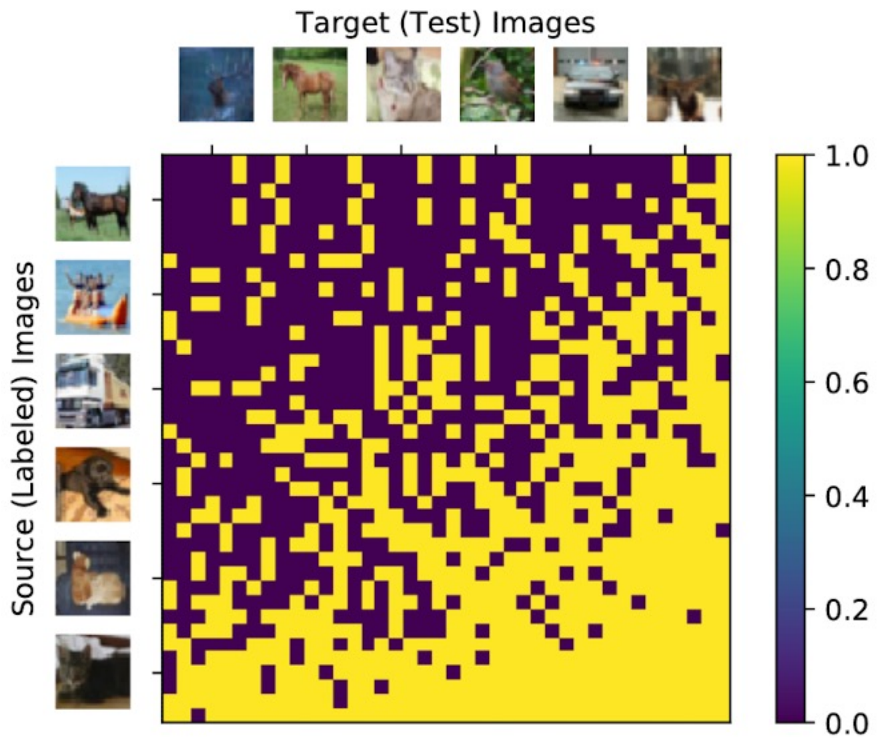
- How the attack works

Poisoning the Unlabeled Dataset of Semi-Supervised Learning

- Evaluation

Dataset	**CIFAR-10**			**SVHN**			**STL-10**		
(% poisoned)	0.1%	0.2%	0.5%	0.1%	0.2%	0.5%	0.1%	0.2%	0.5%
MixMatch	5/8	6/8	8/8	4/8	5/8	5/8	4/8	6/8	7/8
UDA	5/8	7/8	8/8	5/8	5/8	6/8	-	-	-
FixMatch	7/8	8/8	8/8	7/8	7/8	8/8	6/8	8/8	8/8

Poisoning attack success rate out 8 trials



Poisoning attack success rate

An Analysis of Speculative Type Confusion Vulnerabilities in the Wild[5]

- Tow forms of Sepctre V1 attack:

Attacker-controlled variable

```
if ( x < array1_len) { // branch mispredict: taken
  y = array1[ x ];     // read out of bounds
  z = array2[y * 4096]; } // leak y over cache channel
```

(1) Bounds check bypass

```
void syscall_helper(cmd_t* cmd, char* ptr, long x ) {
  // ptr argument held in x86 register %rsi
  cmd_t c = *cmd;    // cache miss
  if (c == CMD_A) { // branch mispredict: taken
    ... code during which x moves to %rsi ...
  }
  if (c == CMD_B) { // branch mispredict: taken
    y = *ptr;  // read from addr x (now in %rsi )
    z = array[y * 4096]; // leak y over cache channel
  }
  ... rest of function ...
```

(2) Type confusion

- If both branches in (2) mispredict: attacker-controlled location is leaked
- Challenge: <u>No data dependency between the attacker-controlled variable and the branches</u>
- Current SW solutions unable to detect and mitigate speculative type confusion

5. Kirzner, Ofek, and Adam Morrison. "An analysis of speculative type confusion vulnerabilities in the wild." USENIX Security '21 (2021)

An Analysis of Speculative Type Confusion Vulnerabilities in the Wild

- Question of this paper: Are OS kernels vulnerable to speculative type confusion?
- Different sources of type confusion:
  - Attacker-introduced: adding code through eBPF
  - Compiler-introduced: C compilers emit type confusion gadgets
  - Polymorphism-related: object-oriented programming of Linux code
- Contributions:
  - Examining all different sources of type confusion in Linux
  - Design of attacks to exploit type confusion gadgets in Linux

An Analysis of Speculative Type Confusion Vulnerabilities in the Wild

- In this presentation: How to exploit speculative type confusion in eBPF
- **eBPF**: a Linux subsystem that lets Linux kernel safely execute untrusted, user-supplied kernel extensions in privileged mode
- eBPF code requires to go through *static safety verification and compilation* before execution
- The verification step ensures that the program does not access unintended memory location (e.g., only reading stack slots that the program has written something into them)

An Analysis of Speculative Type Confusion Vulnerabilities in the Wild

- A proof-of-concept-attack via eBPF
  - eBPF Verification has vulnerabilities
  - Verifier only considers possible execution flows; i.e., <u>unable to catch type confusion</u>
  - Code rejection in (b) fails accidentally. It would fail if we have a perfect verifier!

```
// r0 = ptr to a map array entry (verified ≠ NULL)
// r6 = ptr to stack slot (verified ≠ NULL)
// r9 = scalar value controlled by attacker
```

```
  r0 = *(u64 *)(r0) // miss
A:if r0 != 0x0 goto B
  r6 = r9
B:if r0 != 0x1 goto D
  r9 = *(u8 *)(r6)
C:r1 = M[(r9&1)*512];//leak
D:...
```

(a) Passes verification.

```
  r0 = *(u64 *)(r0) // miss
A:if r0 == 0x0 goto B
  r6 = r9
B:if r0 != 0x0 goto D
  r9 = *(u8 *)(r6)
C:r1 = M[(r9&1)*512];//leak
D:...
```

(b) Fails verification.

An Analysis of Speculative Type Confusion Vulnerabilities in the Wild

- A proof-of-concept-attack via eBPF
  - Challenges of the attack:
    - Predicting two branches as Not-Taken, which their **conditions are mutually exclusive**
    - How to evict the values checked by these branches to have enough time to leak data
    - How to observe the leaked data
  - NOTE: eBPF runs in the kernel address space and the attacker runs in the user space ==> *Cannot share memory*
  - Solution for branch mis-training: *cross address-space out-of-place mis-training*

An Analysis of Speculative Type Confusion Vulnerabilities in the Wild

- A proof-of-concept-attack via eBPF
  - Cross address-space out-of-place branch mis-training
    - Setting up a "*shadow*" of natively-compiled eBPF in the attacker's process: Shadow program is going to train PHT entries to mispredict victim's (eBPF program) branches
    - However, Not-Taken conditions are not mutually exclusive in the shadow program

Shadow program needs to set up the A' and B' addresses in a way to have PHT collision with A and B addresses

```
// addresses A' and B' collide in the PHT
// with addresses A and B in Listing 2a
A': if r0 == 0x0 goto B'
    // dummy register assignment
B': if r0 == 0x0 goto C'
    // dummy pointer dereference
C': ...
```

Shadow program

```
  r0 = *(u64 *)(r0) // miss
A:if r0 != 0x0 goto B
  r6 = r9
B:if r0 != 0x1 goto D
  r9 = *(u8 *)(r6)
C:r1 = M[(r9&1)*512];//leak
D:...
```

eBPF program

An Analysis of Speculative Type Confusion Vulnerabilities in the Wild

- A proof-of-concept-attack via eBPF
  - Cross address-space out-of-place branch mis-training
    - Two factors to ensure collision:
      - (1) the state of the Global History Register (GHR)
      - (2) BPU-indexing in the branches' virtual address
    - Solution for (1): Executing a branch slide for both shadow and eBPF
    - Solution for (2): "brute-force" search to find collisions
    - Search algorithm in next slides

Branch Slide

```
            mov     $0x1,%edi
            cmp     $0x0,%rdi
            jne     L1
L1:         cmp     $0x0,%rdi
            jne     L2
            ...
L(n-1):     cmp     $0x0,%rdi
            jne     Ln
Ln:         # exploit starts here
```

An Analysis of Speculative Type Confusion Vulnerabilities in the Wild

- A proof-of-concept-attack via eBPF
  - Cache flushing in eBPF program:
    - Two reason that we need cache flush:
      - (1) Causing a miss for values checked by the branches to have enough time for leakage
      - (2) Observing the leaked data via Flush+Reload
    - Solution: HORN technique
      - Another eBPF program running on another core to access the cache lines that are required to be flushed in the victim ➔ *cache miss for the victim*
      - A third eBPF program is needed to observe the leaked data

```
r0 = CALL ktime_get_ns()
r1 = M[b]  // b is 0*512 or 1*512
r2 = CALL ktime_get_ns()
return r2 - r0 // if small -> secret is b
```

An Analysis of Speculative Type Confusion Vulnerabilities in the Wild

- A proof-of-concept-attack via eBPF
  - Search algorithm to find address-based PHT collisions
    - Allocating a 2MB buffer and for each byte in the buffer we put the shadow in that location and try the attack
    - (1) Repeating the shadow to mis-train the branches and hope they collide with the victim's branches
    - (2) Invoking the in-kernel victim
    - (3) if no leaks occurs: No collision, move the shadow and go to (1)
    - (4) if leak occurs: No collision if the victim is leaking its own stack data
    - (5) trying the attack again and flip the relevant bit in that stack variable
    - (6) if the leaked bit flips too: No collision, move the shadow and go to (1)
    - (7) if the leaked bit does not flip: Collision found

An Analysis of Speculative Type Confusion Vulnerabilities in the Wild

- A proof-of-concept-attack via eBPF
  - Evaluation
    - Goal: leaking an arbitrary page (4096 bytes) of kernel memory
    - Retrying the attack for k times

found collision?	average	min.	max.	median
success (46/50)	9.5 min.	20 sec.	45 min.	8.5 min.
failure (4/50)		$\approx$ 53 min		

Table 1: Times to find PHT collision with victim (50 experiments).

retries	success rate	transmission rate
1	99.9%	55,416 bps
2	98.7%	28,712 bps
10	100%	5,881 bps
100	100%	584 bps

Table 2: Accuracy and capacity of the eBPF covert channel.

Best Papers Final Ranking

1. Poisoning the Unlabeled Dataset of Semi-Supervised Learning
2. An Analysis of Speculative Type Confusion Vulnerabilities in the Wild
3. Extracting Training Data from Large Language Models
4. You Autocomplete Me: Poisoning Vulnerabilities in Neural Code Completion
5. Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks
6. Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption
7. Double-Cross Attacks: Subverting Active Learning Systems

Best Paper Award Goes to …

Poisoning the Unlabeled Dataset of Semi-Supervised Learning

Nicholas Carlini
Google

- Remarks:
  - First attack on semi-supervised learning (which was considered as the savior!)
  - Proposing a good mitigation to address their attack (still there is hope!)
  - Great articulation of the idea! All the sections walks the reader through the fundamentals of ML and why the author is making all the decisions to launch his attack

Appendix: List of all papers in the first round

1. Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption
    - Session: *Cryptography: Attacks*
2. "It's stressful having all these phones": Investigating Sex Workers' Safety Goals, Risks, and Practices Online
    - Session: *Usable Security and Privacy: User Perspectives*
3. Lord of the Ring(s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical
    - Session: *Hardware Side Channel Attacks*
4. Frontal Attack: Leaking Control-Flow in SGX via the CPU Frontend
    - Session: *Hardware Side Channel Attacks*
5. SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript
    - Session: *Hardware Security*
6. Osiris: Automated Discovery of Microarchitectural Side Channels
    - Session: *Hardware Side Channel Defenses*
7. Swivel: Hardening WebAssembly against Spectre
    - Session: *Hardware Side Channel Defenses*

Appendix: List of all papers in the first round

8. You Autocomplete Me: Poisoning Vulnerabilities in Neural Code Completion
   - Session: *Machine Learning: Backdoor and Poisoning*
9. Double-Cross Attacks: Subverting Active Learning Systems
   - Session: *Machine Learning: Backdoor and Poisoning*
10. CADE: Detecting and Explaining Concept Drift Samples for Security Applications
    - Session: *Adversarial Machine Learning: Defenses*
11. An Analysis of Speculative Type Confusion Vulnerabilities in the Wild
    - Session: *Operating Systems Security*
12. Weaponizing Middleboxes for TCP Reflected Amplification
    - Session: *Research on Surveillance and Censorship*
13. Poisoning the Unlabeled Dataset of Semi-Supervised Learning
    - Session: *Machine Learning: Backdoor and Poisoning*
14. When Malware Changed Its Mind: An Empirical Study of Variable Program Behaviors in the Real World
    - Session: *Malware and Program Analysis 1*
15. ATLAS: A Sequence-based Learning Approach for Attack Investigation
    - Session: *Forensics and Diagnostics for Security and Voting*

Appendix: List of all papers in the first round

16. Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks
    - Session: *Hardware Side Channel Defenses*
17. Too Good to Be Safe: Tricking Lane Detection in Autonomous Driving with Crafted Perturbations
    - Session: *Attacks*
18. Systematic Evaluation of Privacy Risks of Machine Learning Models
    - Session: *Machine Learning: Privacy Issues*
19. Extracting Training Data from Large Language Models
    - Session: *Machine Learning: Privacy Issues*
20. Stealing Links from Graph Neural Networks
    - Session: *Machine Learning: Privacy Issues*

Thanks for your attention!

Ali Hajiabadi
National University of Singapore

USENIX Security 2021

CompArch
Group Meeting