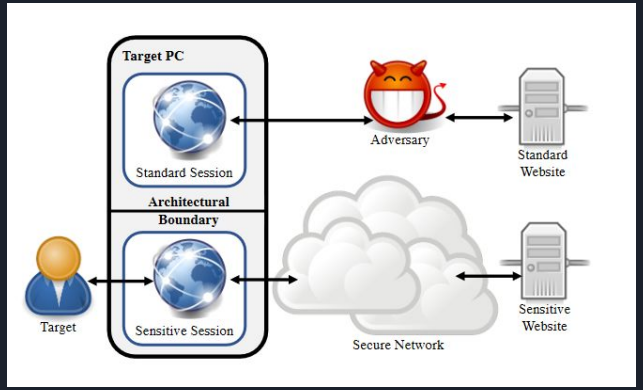
Adversarial AI to Prevent Microarchitectural Website Detection Attacks

sddec21-13.sd.ece.iastate.edu

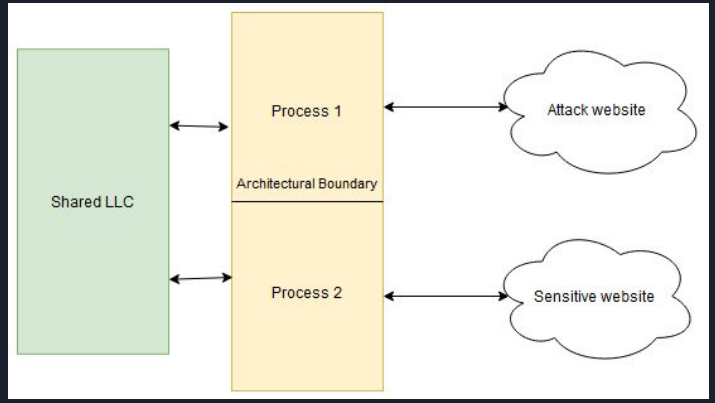
Sddec21-13
Advisor/Client: Berk Gulmezoglu
Team: Ege Demir, Sean McClannahan, Aaron Anderson, Thane Stoley

Problem Statement

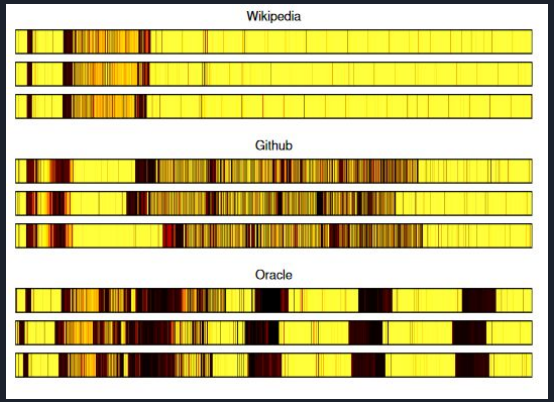
- Privacy oriented browsers like chrome and TOR offer no protection against microarchitectural attacks
- There is no current software/browser based tools to defend against these attacks
- Most proposed solutions rely on processor hardware changes
    - Intel has not made these changes yet



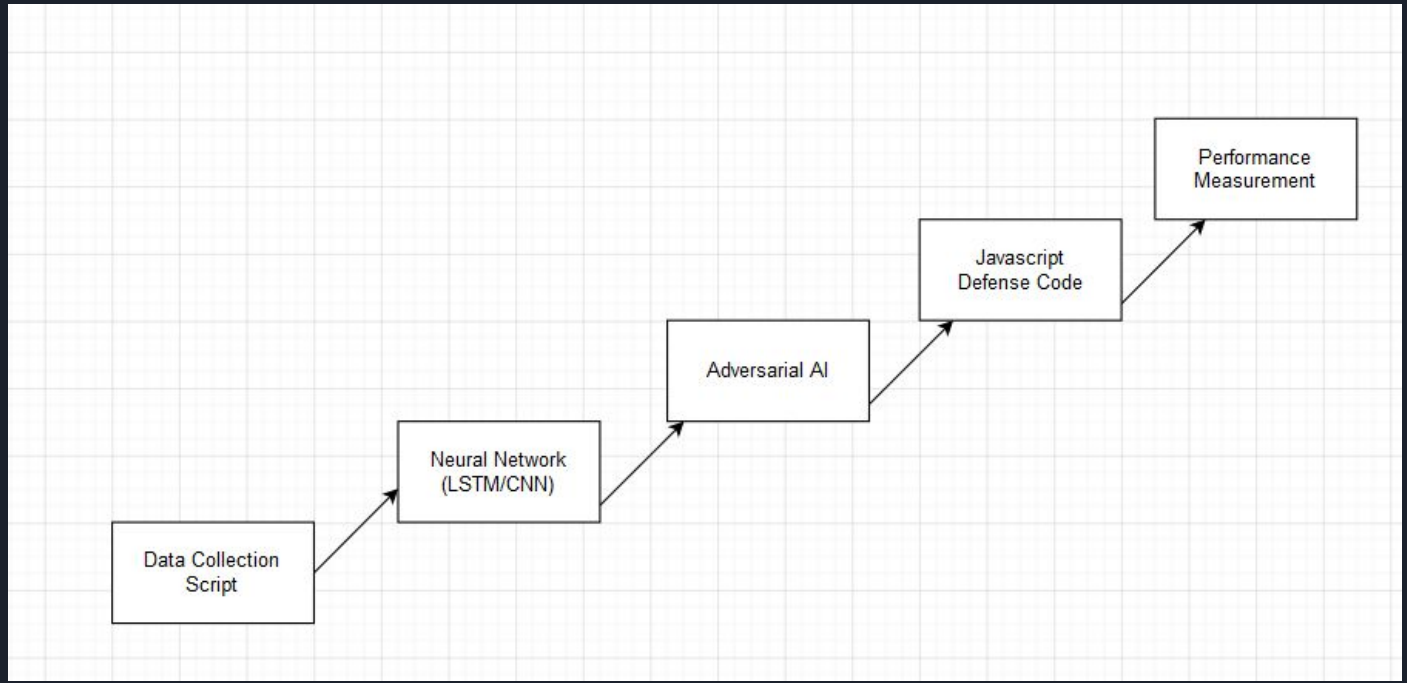
Problem Statement



- What are microarchitectural attacks?
  - Exploitation of the lowest level cache (LLC)
  - Memory access patterns
  - Website detection
- Previous work
  - Approximately 80% accuracy (for 100 websites)
- Browser time measurement resolution
  - Firefox: 2 ms
  - Chrome: 0.1 ms
  - Tor: 100 ms

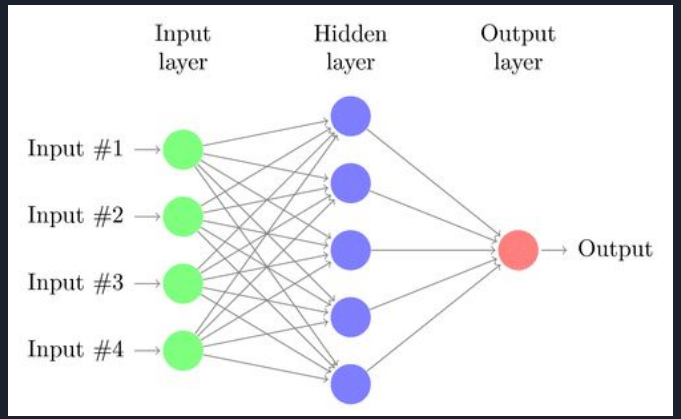


Conceptual Sketch



Functional Requirements

- We will have a Javascript-based cache monitoring code
- We will have a Python-based neural network that identifies websites based on output from cache monitoring javascript code
- We will use a Python-based adversarial AI tools to introduce artificial noise in cache to reduce classification rate of our attacker code.

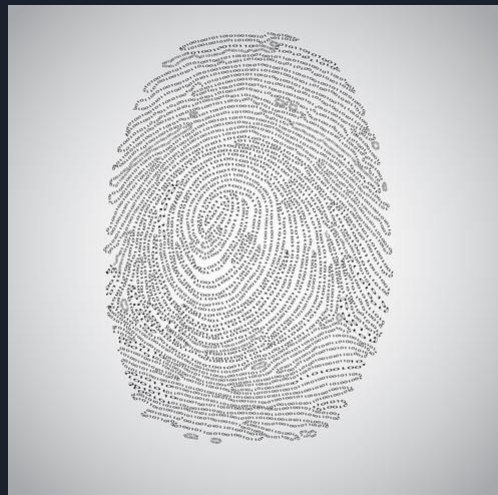


Non-functional Requirements

- Must not increase system overhead by more than 25%
- Works with more than one browser
- Attacker code must be able to identify with an accuracy rate of 80% for 50 websites
- Defense code must lower accuracy of attacker code by at least 60%

Technical Constraints & Considerations

- Cache fingerprint changes based on individual's processor architecture
- The browser will also affect the cache fingerprint.
- We can't cover every website. Doing that is next to impossible to accomplish. We are aiming for 50 websites.
- As websites are updated, we will need to retrain our algorithm for the new configuration.



What Makes Our Project Unique?

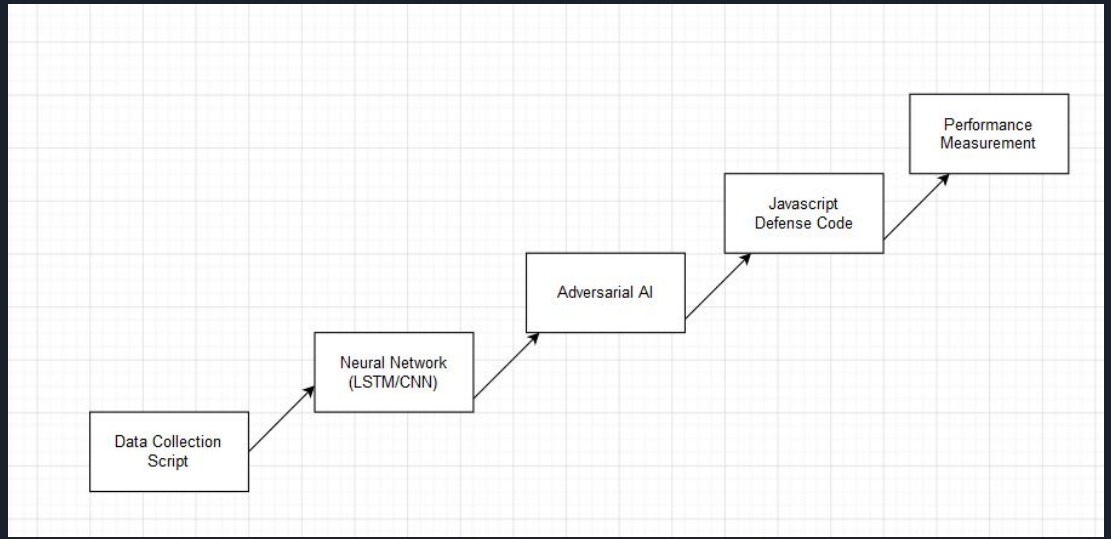
- Common vulnerability in all browsers
- Fairly new and not much is known about preventing architectural attacks
- New architecture design is necessary to for absolute security
- These problems makes it hard to defend against attack

Possible Risks and Mitigation

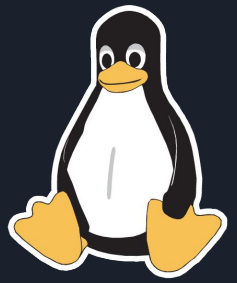
- In creating artificial noise in an attempt to stop attack, it is liable to cause slowdown on other processes.
  - Adversarial AI tools will be utilized to reduce the strain on the system as much as realistically possible.
- Because of the nature of operating systems, it is expected for other processes to be using cache at the same time, which introduces unnecessary noise during data collection.
  - Correct Linux distribution will be identified so that processes running in the background can be minimized

System Design - Functional Decomposition

1. Data Collection Script
   a. Written in JavaScript
   b. Automates cache fingerprint data collection over a set period of time
2. Neural Network
   a. Fed data from data collection script
   b. To be trained to detect website usage based on cache.
3. JavaScript Defense Code
   a. Dynamic cache usage modification
   b. Creates artificial noise in cache
   c. Adversarial AI is utilized to know exactly when to create the noise
4. Performance Measurement
   a. Timing of operations
   b. Classification rate



Deeper Design Dive



1. Neural Network
    a. Using tensorflow to train neural network.
        i. Weights are based on time it takes to access array
    b. Tries to guess website from list of 50
        i. This is based on cache fingerprint
2. Data Collection Script
    a. Creates array
        i. Keeps accessing array until entire cache is nearly filled
    b. Continues to try to access array
        i. Records time it takes to access array every (unit)
    c. Runs for a set time period (30 seconds)
        i. Outputs data as a JSON file
3. JavaScript Defense Code
    a. Utilizes adversarial AI tools to determine exactly when to introduce noise
4. Performance Measurement
    a. Classification Rate
    b. Processor usage
    c. Calculate performance before and after running defense code

Resources and Cost

- RTX-3090 for training neural network
- Laptop with Linux based operating system capable of network access
- Funds are not necessary

Schedule and Milestones

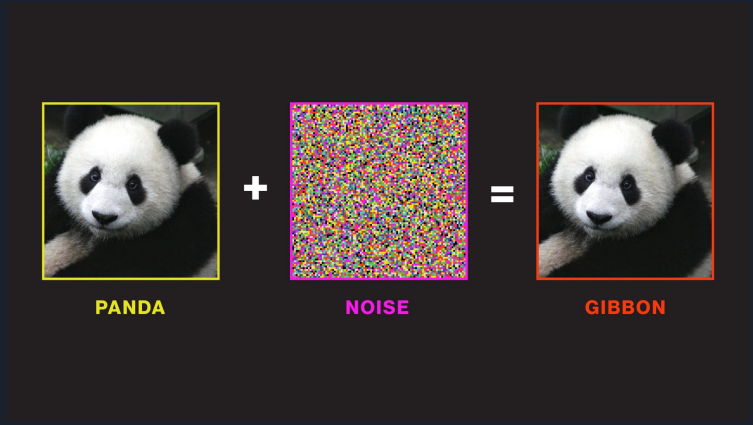
Task	Hours (estimating 10 hours per week)	Reasoning
Research & Develop a Javascript-based cache monitoring code	50 hours/5 weeks	This estimate includes coding and testing. This code is a very important part of our project and needs to be working extremely well.
Develop and train a Neural Network to identify websites based on cache fingerprint	40 hours/4 weeks	Developing and training a neural network is going to be a very time consuming process. Only one member of our group has worked with A.I before so we are estimating this task will be very difficult.
Offer an adversarial AI based mathematical solution to change cache usage dynamically when a specific website is visited	30 hours/3 weeks	This estimate is based on the amount of estimated testing we will need to confirm our algorithm.
Develop a new Javascript-based defense code integrated with our solution	30 hours/ 3 week	This process should not take too long as we will simply create noise at a certain time to trick the attacker code.
Optimizing performance overhead	20-30 hours/ 2-3 weeks	It is hard to estimate how much time this will take because optimization can be very difficult. However, there is a chance that we get very close to the benchmark right away.

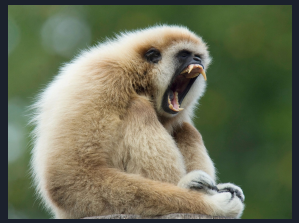
Technology Platforms Being Used

- We are using JavaScript and Python
- Browsers will be Google Chrome and Firefox-based Tor browser
- Operating system will be Linux

Test Plan

- Attack code and defense code will be running at the same time
- Send data from attack code to neural network
- See if neural network is able to classify a lower percentage of websites
- Data will be one-dimensional, so we will be using one-dimensional adversarial AI techniques



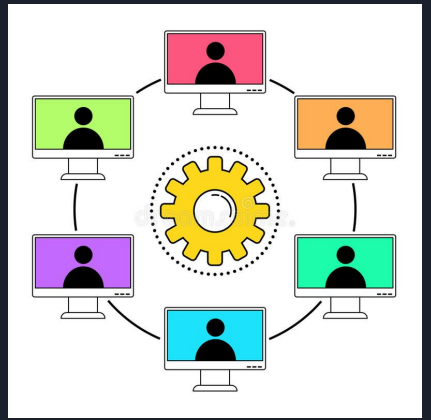


(Gibbon for reference)

Current Status of Project

- We fully understand how the cache monitoring code works and operates
- We are working towards automating data collection
- We are also starting to prototype a potential neural network design.

Task Distribution



1. Research JavaScript-based cache usage monitoring code.
    a. Cache attack research
    b. Testing
2. Develop deep learning-based website detection mechanism.
    a. Develop algorithm for neural network
    b. Python-based AI code
    c. Training neural network
3. Mathematical solution to change cache usage dynamically when a website is visited.
4. Optimize Performance.
    a. Timing information
    b. Develop to hit target level of accuracy and system performance

Looking to Next Semester

SUMMER

Start and possibly finish neural network.

EARLY FALL

Develop and train Adversarial AI

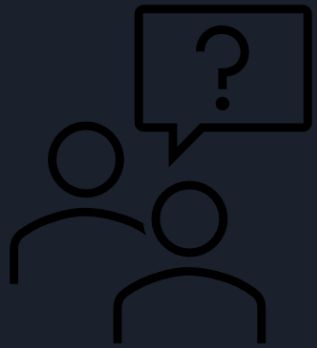
FALL

Create defensive JavaScript Code.

LATE FALL

Increase and improve performance of AI and JavaScript defense code.

QUESTIONS?



THANKS FOR LISTENING