Adversarial AI to Prevent Microarchitectural Website Detection Attacks

Sddec21-13
Client/Adviser: Berk Gulmezoglu

Ege Demir
Sean McClannahan
Aaron Anderson
Thane Storley
sddec21-13@iastate.edu
sddec21-13.sd.ece.iastate.edu/

Revised: 4/25/2021 Version: 1.5

Executive Summary

Development Standards & Practices Used

List all standard circuit, hardware, software practices used in this project. List all the Engineering standards that apply to this project that were considered.

Summary of Requirements

- We need JavaScript code to be able to analyze and collect cache data
- We also need a neural network trained with the collected data to classify the cache fingerprint.

Applicable Courses from Iowa State University Curriculum

- CprE 281
- CprE 308
- CprE 381
- ComS 309
- ComS 311
- SE/ComS 319

New Skills/Knowledge acquired that was not taught in courses

- Javascript cache accessing framework
- Machine learning/ Neural networks
- Extensive cache knowledge

Table of Contents

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

1 Introduction

1.1 Acknowledgement

Berk Gulmezoglu: Significant technical consultation and contribution of equipments aiding in the advancement of our product.

1.2 Problem and Project Statement

-General problem statement:

Microarchitectural attacks are one way to exploit the low-level architectural problems of the hardware, letting attackers gain knowledge about the target user. One of the ways this is achieved is by exploiting the last level cache (LLC) on the computer. As the LLC is shared between all the processor cores, it exposes the memory access patterns of different processes. This vulnerability can be exploited by other processes to determine what the target user is working on.

We will focus specifically on website access detection. As many papers showed, the memory access pattern is often similar every time a website is being loaded on a specific browser. This pattern can be used to classify the websites that run in certain environments like a specific browser, operating system, or processor. As more fingerprints are recorded, the attacker will be able to better identify what websites the user is browsing. As of currently, there are no optimized defense mechanisms implemented in any of the browsers for this.

The detection algorithm can be highly enhanced using a neural network. As more data shows how different cache usage can be classified as certain websites, the accuracy of our neural network based classification algorithm should increase.

-General solution approach:

Initially, we will implement a version of this microarchitectural attack. We will prepare the JavaScript program that captures the cache access using a method called Prime+Probe. Then, we will classify the data we collected, using a machine learning neural network. Our attack needs to have a high success rate to be relevant to the research done before in this subject.

Our solution to this attack, guided by Prof. Gulmezoglu, will be to find a mathematical solution to randomize the memory access pattern. This randomization process will be naturally heavy on the resources. A solution already implemented by the Tor browser causes websites to load 500% slower. Our aim is to reduce that number significantly.

Implementation of our randomization process depends on the adversarial AI techniques. As of right now, we know that our defensive JavaScript code needs to create artificial noise in the cache to reduce the classification rate of the attacker. We want to minimize how much we are going to fill the cache, so that performance overhead is minimized. This is why we need another JavaScript process that outputs the timings to create noise in.

1.3 Operational Environment

The general operational environment would be two of the most widely used browsers,Firefox and Google Chrome. It will operate in these to monitor the hardware cache, and prevent attacks on it. It will be built within a Javascript framework.

1.4 Requirements*

- GPU Server Temperature: As we will be using a remote GPU machine to train our network, the temperature of the room this machine is located needs to meet acceptable standard temperature set by the developers of the machine.
- We will be adding a custom noise functionality to our end product. This means we will be requiring an input from the user as to what website he/she is visiting, to introduce the noise specific to that website.

1.5 Intended Users and Uses

The main use of our product is to prevent a website or person from gathering data on someone using the hardware cache.

Primary intended users include those who want to remain anonymous from websites and departments of government to keep private information safe. Cross-site tracking for advertisements using cache attacks are also prevented.

1.6 Assumptions and Limitations

Assumptions:

- Works with 1-2 browsers, likely Mozilla Firefox and Google Chrome.
- LLC is inclusive, meaning data will be put into every cache level.

Limitations:

- Limited in the number of websites we can detect attacks. Covering every website would be next to impossible to feasibly do. Therefore, we are basing our metrics on 50 websites.
- As websites get updated, it will be hard to dynamically update our algorithm for every website.
- As more than one website might be loading in the background, classifying the cache usage to more than one website will be a limitation.
- Our tool will work on Intel architecture. The cache fingerprint for a specific website would change between different hardware designs. Hence, there might be a need for training separate models for different Intel architectures.

1.7 Expected End Product and Deliverables

- A Javascript code to monitor cache usage while visiting web pages in a browser

This would be code that collects cache operations while the user visits websites. This would be used for gathering data for the deep learning detection tool. A website will use the cache in the same way for every visit. Delivery Date: End of Semester.

- Deep Learning-based detection tool to detect web pages

As mentioned above, the deep learning tool would be used to detect irregular cache usage on a website. Since websites use caches in the same way every time, an irregularity would be able to be detected. The Javascript code above would gather "standard" cache usage, and the tool would use that data to detect the irregularities. Delivery Date: Mid-september

- A Javascript code to prevent website detection on a browser

This second Javascript code is used within the browser with the deep learning tool to detect and deal with irregularities in the browser cache. Upon detection, the code will find the cause of the irregular memory access, and remove it from the cache. It could then inform the user of the attack and that this website may not be safe. Delivery Date: End of Next Semester.

2 Project Plan

2.1 Task Decomposition

1. Research Javascript-based cache usage monitoring code
    a. Research Cache attacks
        i. How cache attacks work
        ii. How to prevent cache attacks
    b. Test cache monitoring code
2. Develop a Deep Learning-based website detection mechanism
    a. Develop an algorithm for neural network
    b. create python code for AI
        i. Implement TensorFlow libraries
    c. Train neural network
        i. Configure correct weights for neural nodes
3. Offer a mathematical solution to change cache usage dynamically when a website is visited
    a. Create prototype formula
    b. Confirm prototype formula with professor
    c. Test formula with neural network
4. Implement the mathematical solution use Javascript code
    a. Code formula into the program
    b. Test program to ensure accuracy
5. Optimizing performance overhead
    a. Collecting timing information using browser's inspect tool
    b. Continue to develop code until system performance hits the targeted level

2.2 Risks And Risk Management/Mitigation

Issues with the network could cause problems during testing, throwing data off. An environment devoid of any other memory usage is key to successful testing here. A major risk would be a website's fingerprint being modified from when we gathered data. The simplest solution would be to feed the new data to the DL model, so it then can see the updated website's new cache usage as normal. Doing this constantly would be difficult without slowing down the website significantly.

Because we are manually training and testing our neural network, possibilities of false-positive and negative are mitigated.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

- Javascript code will be able to monitor cache usage
- The neural network will be able to classify at least 80% of websites in cache, specifically with the target of 50 websites
- The javascript code that randomizes cache fingerprint when a website is visited will decrease the accuracy of our neural network by at least 60% using adversarial AI
- Our solution will only increase the strain on the system by at most 25%

2.4 Project Timeline/Schedule

First Semester	Second Semester
January 2021- April 2021 Research Javascript-based cache usage monitoring code ● Research Cache attacks ● How cache attacks work ● How to prevent cache attacks ● Test cache monitoring code May 2021 - September 2021 Research Neural Networks ● basic AI code ● Brainstorm ideas ● Get a general idea about how neural networks work	September 2021 Develop a Deep Learning-based website detection mechanism ● Develop an algorithm for neural network ● create python code for AI ● Implement TensorFlow libraries September 2021 ~ October 2021 Train neural network ● Configure correct weights for neural nodes ~October 2021 - November 2021 Offer a mathematical solution to change cache usage dynamically when a website is visited ● Create prototype formula ● Confirm prototype formula with professor ● Test formula with neural network November 2021 - December 2021 Implement the mathematical solution using the Javascript code ● Code formula into a javascript program ● Test program to ensure accuracy December 2021 Optimizing performance overhead ● Collecting timing information using

	browser's inspect tool • Continue to develop code until system performance hits the targeted level

2.5 Project Tracking Procedures

Our group plans on using Gitlabs and Slack to communicate and track our progress. This includes (but is not limited to) setting deadlines, issuing updates, and integrating quality assurance procedures.

2.6 Personnel Effort Requirements

Include a detailed estimate in the form of a table accompanied by a textual reference and explanation. This estimate shall be done on a task-by-task basis and should be the projected effort in total number of person-hours required to perform the task.

Task	Hours (estimating 10 hours per week)	Reasoning
Develop a Javascript-based cache usage monitoring code	30 hours/3 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 a Deep Learning-based website detection mechanism	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 a mathematical solution to change cache usage dynamically when a website is visited	20 hours/2 weeks	This estimate is based on the amount of estimated testing we will need to confirm our algorithm.
Implement the mathematical solution in the Javascript code	10 hours/ 1 week	This process should not take too long as implementing math in JS should not be too technical.

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.

2.7 Other Resource Requirements

- A GPU based computation machine will be required to train our neural network.
- We will require a reliable network connection to have 24/7 access to the GPU machine.
- Laptop to collect fingerprint data

2.8 Financial Requirements

No financial requirements are needed at this time.

3 Design

3.1 Previous Work And Literature

Shujian Yu, Kristoffer Wickstrøm, Robert Jenssen, Jose C. Principe. (2019) Understanding Convolutional Neural Networks with Information Theory: An Initial Exploration

Discusses convolutional neural networks (CNNs), and how measuring information flow using newly discovered estimators is much easier, with less approximation. Also talks about data processing inequalities and how that affects training CNNs.

Qiu, Shilin & Liu, Qihe & Zhou, Shijie & Wu, Chunjiang. (2019). Review of Artificial Intelligence Adversarial Attack and Defense Technologies. Applied Sciences. 9. 909. 10.3390/app9050909.

Explains recent developments in adversarial attacks and how those attacks have limited AI in security industries. Discusses how these attacks work, both in training and testing, and the defensive maneuvers that have or are being developed.

Oren, Y., Kemerlis, V., Sethumadhavan, S., & Keromytis, A. (2015). The Spy in the Sandbox: Practical Cache Attacks in JavaScript and Their Implications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (pp. 1406–1418). Association for Computing Machinery.

Shows off a architecture-based attack that goes through the browser. Discusses that it isn't difficult to have the correct environment for the attack, and allows for easy information gathering. It discusses how this attack is preventable, but requires a lot of power from the browser.

Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, & Yuval Yarom (2019). Robust Website Fingerprinting Through the Cache Occupancy Channel. In *28th USENIX Security Symposium (USENIX Security 19)* (pp. 639–656). USENIX Association.

Discusses website fingerprinting attacks, how they work, and what defensive measures have and should be taken. They use machine learning techniques in conjunction with JavaScript to detect cache activity, and show it's accuracy in various environments. They then go into a potential solution, creating "artificial cache activity" to reduce the attack's effectiveness.

These papers explain how the attack works and give data that they acquired using code they wrote. They do not go in depth into potential solutions and did not attempt to write any code to prevent the attack. We are using the attack code they wrote to help us write code to prevent the attack they discuss in their papers. We are also using a neural network to analyze data which was not mentioned in these papers.

3.2 Design Thinking

We want to be able to automate our collection of data for the deep learning model, as it would help in speeding up the process of gathering website data. An algorithm for our AI will also need to be worked out mathematically.

3.3 Proposed Design

We've been working with and understanding javascript code used to collect data from websites used in previous projects of this nature. It provides charitable data that can visualize cache usage for us. This partially meets the functional requirement, and will be totally met with more through testing.

The javascript program probes a given website, taking in cache activity for a specified time period, and then it creates a file containing the collected data, which can then be graphed.

We'll want to feed this collected data through our deep learning model, and train it. From there, our Adversarial AI will watch for and dynamically change the cache usage as it continues to watch the cache. Another script will be created to create noise and fill the cache to try and prevent the attack.

- Data collection JS
- Data analysis (Model training)
- Adversarial AI
- Artificial Noise in LLC

3.4 Technology Considerations

- Trade offs
  - Architecture dependent
    - How caches are mapped and the size of those caches vary from architecture to architecture, so for that reason we are sticking with one architecture.
  - Network latency might change, which might affect our classification performance
    - Specifically, our classification accuracy will be affected.
  - The content of the website might change. This will cause the website to have a different cache fingerprint.
    - Social Media sites are the biggest example of this, such as Facebook and Twitter.
  - As misclassification rate increases with our defence code, performance overhead is expected to increase with it.
    - Creating artificial noise in the cache could cause performance issues for other processes in the system.
- Strength
  - Only hard-solution as a defense is to change low-level design. Our solution is software based technique to create a defense tool.

3.5 Design Analysis

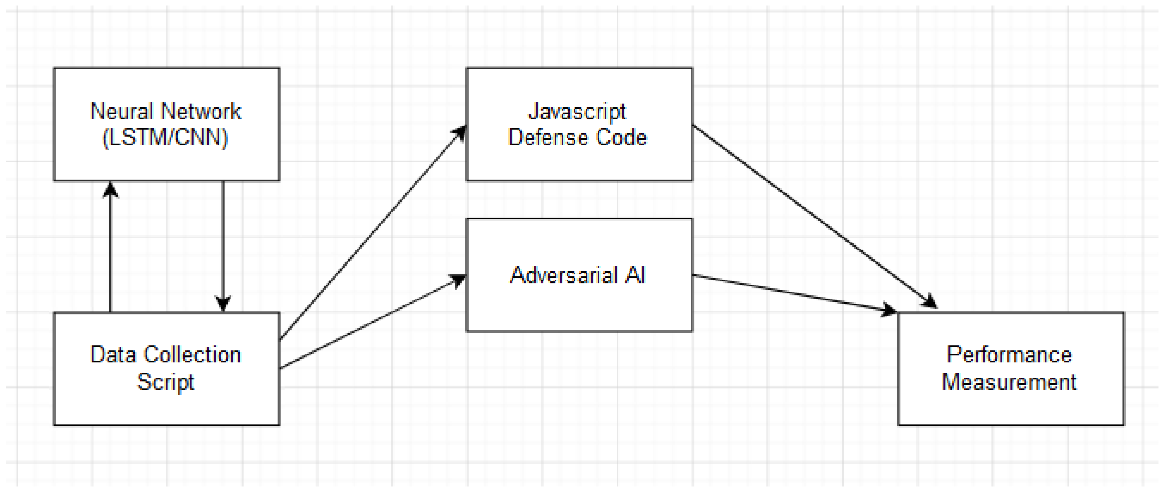
Borrowed javascript code currently works, because it is more or less the same as previous projects that have worked on tackling the same problem. We intend to improve classification of websites by using deep-learning tools.

After that, we plan to come up with a mathematical solution that creates noise in the cache according to the website being visited. Another script would need to be written to implement the solution. Fully occupying the cache will be a technical challenge to work out.

3.6 Development Process

We're working with a modified waterfall approach, working on each aspect of the project and testing each major part as we work with them. The rationale behind this decision is that the dependencies are very linear. The neural network is dependent on our JS attack code and our solution is dependent on our neural network. This means that we have to create and test each module in our project in a specific order. However, we are able to start on concepts for each module without necessarily finishing the previous module. For example, we could experiment with various neural network concepts before completing the JS attack code.

3.7 Design Plan



4 Testing

4.1 Unit Testing

We are accessing websites and collecting cache usage data to test if the software works properly. To do this effectively, the cache shouldn't be used by anything other than the website and our program during testing. We also will be comparing the data we acquire from our JS attack code with data sets from the papers mentioned in section 3.1. Our neural network will also need to be extensively tested to make sure it can make the right decisions.

4.2 Interface Testing

Our current design is simply to have a pop-up on our own website for the user to enter the name of the website being visited. This input, in return, will determine which parts of the cache that the noise is going to be created in.

4.3 Acceptance Testing

As laid out in section 2.3, we'd want to show our client that we are meeting the accuracy checks and system slowdown goals. That includes:

- Classifying 80% of websites based on the cache usage, with a goal of 50 websites.
- Our Javascript defense code decreases the accuracy of our neural network at least by 60% due to adversarial AI.
- Solution only increases system strain by 25% at the most.

4.4 Results

Our initial testing has shown our data gathered by the “attack” script had several consistencies when gathering from the same website.

5 Implementation

This semester our primary goal was to lock down the javascript code that analyzes a target’s cache by timing how long it takes to access the cache. For next semester we plan to train an AI that takes the output from the JS code and tries to guess what websites have been visited based on how long it took to access the cache. First we will create an automated process where we will simulate visiting specific websites over and over again to create data for our AI. Then once our AI is able to identify websites we will try to implement a solution that involves implementing JS code that creates noise in certain cache sections. Ideally, this will lower the accuracy of our AI which would make it harder for it to guess websites. If this solution works our project will be considered successful.

6 Closing Material

6.1 Conclusion

Our goals this semester were to research javascript based cache attacks and form a plan on how to prevent them. So far, we have mostly conducted research and attempted to recreate some of these circumstances. In the future, we plan on adopting a deep learning algorithm and integrating mathematical solutions to recognize and defend against these attacks. We believe that this is the best method for approaching our design as it allows us to understand our challenge fully before diving in and trying to solve it with such complex tools.

6.2 References

List technical references and related work / market survey references. Do professional citation style (ex. IEEE).

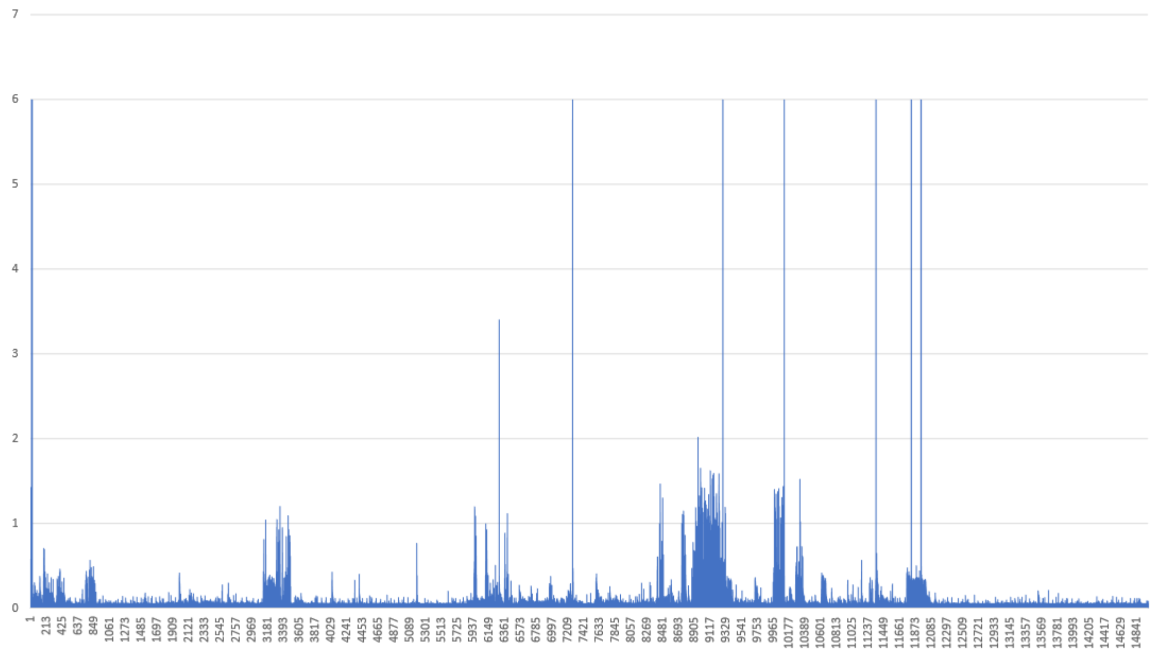
Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, & Yuval Yarom (2019). Robust Website Fingerprinting Through the Cache Occupancy Channel. In *28th USENIX Security Symposium (USENIX Security 19)* (pp. 639–656). USENIX Association

Oren, Y., Kemerlis, V., Sethumadhavan, S., & Keromytis, A. (2015). The Spy in the Sandbox: Practical Cache Attacks in JavaScript and Their Implications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (pp. 1406–1418). Association for Computing Machinery.

Qiu, Shilin & Liu, Qihe & Zhou, Shijie & Wu, Chunjiang. (2019). Review of Artificial Intelligence Adversarial Attack and Defense Technologies. Applied Sciences. 9. 909. 10.3390/app9050909.

Shujian Yu, Kristoffer Wickstrøm, Robert Jenssen, Jose C. Principe. (2019) Understanding Convolutional Neural Networks with Information Theory: An Initial Exploration

6.3 Appendices



Histogram 1: This is an example of the data output from our Javascript code that analyzes a target's cache. The code outputs a JSON file but is represented here as a histogram. The y axis represents the time it took to access the cache seconds and the x axis represents the attempt number. It is important to note that accesses over 5 seconds are considered errors and will not be considered in analysis.