BSP S2 Blueprint

Week 2

Scientific Deliverable Question: What is the effectiveness of pattern recognition systems, used for detecting intrusion attempts?

Possible bullet points:

- **Intrusion Detection System a.k.a. IDS**: System, device or software, which reports to the administrator for any malicious activities or policy violations detected.
    - **Difference between IPS and IDS?**
- **Algorithm selection and optimisation:** The method of choosing needed techniques and algorithms in different situations, with the goal of successfully
- **Training data:** Dataset used to teach machine learning models with the goal of predicting possible malicious intents.
- **Feature sampling:** Method in which a subset of network traffics attributes are selected for the firewall's decision-making. The goal is to increase performance and reduce the occurrences of false positives.

- **Domain adaptation?:**
    - **Consistent Learning? (maybe technical)**
- **Performance Comparison? (with CAUID and without)**
- **Robustness?:**
    - **Efficiency**

Recourses: **https://www.researchgate.net/profile/Batta-Mahesh/publication/344717762_Machine_Learning_Algorithms_-A_Review/links/5f8b2365299bf1b53e2d243a/Machine-Learning-Algorithms-A-Review.pdf?eid=5082902844932096 (**Review of Machine Learning algorithms)

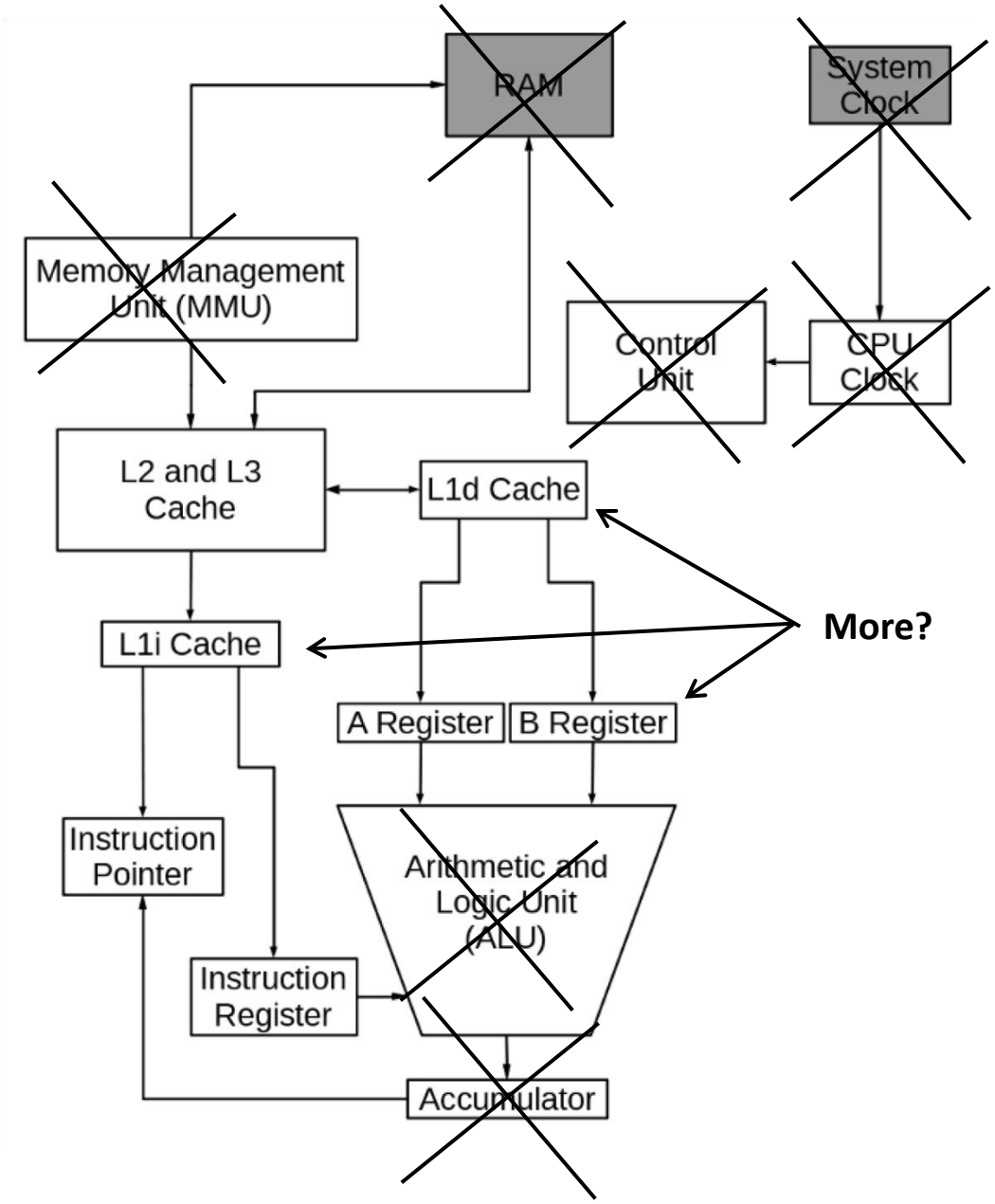
Anomaly-based network intrusion detection: Techniques, systems and challenges P. Garcı´a-Teodoroa, *, J. Dı´az-Verdejoa , G. Macia´-Ferna´ndeza , E. Va´zquezb

https://www.ijert.org/research/anomaly-based-network-intrusion-detection-using-machine-learning-techniques-IJERTV6IS050128.pdf (Anomaly based Network Intrusion Detection using Machine Learning Techniques.)

https://www.v7labs.com/blog/domain-adaptation-guide (Domain Adaptation in Computer Vision : Everything You Need to Know)

https://www.pynetlabs.com/difference-between-ids-and-ips/#:~:text=IDS%20only%20alerts%20the%20network,before%20they%20reach%20the%20target.&text=IDS%20is%20usually%20placed%20outside,a%20firewall%20or%20a%20router. (Difference between IPS and DPS)

Technical Deliverable



Instruction register and pointer

The instruction pointer specifies the location in memory containing the next instruction to be executed by the CPU. When the CPU completes the execution of the current instruction, the next instruction is loaded into the instruction register from the memory location pointed to by the instruction pointer.

After the instruction is loaded into the instruction register, the instruction register pointer is incremented by one instruction address. Incrementing allows it to be ready to move the next instruction into the instruction register.

Cache

The CPU never directly accesses RAM. Modern CPUs have one or more layers of cache. The CPU's ability to perform calculations is much faster than the RAM's ability to feed data to the

CPU. The reasons for this are beyond the scope of this article, but I will explore it further in the next article.

Cache memory is faster than the system RAM, and it is closer to the CPU because it is on the processor chip. The cache provides data storage and instructions to prevent the CPU from waiting for data to be retrieved from RAM. When the CPU needs data—and program instructions are also considered to be data—the cache determines whether the data is already in residence and provides it to the CPU.

If the requested data is not in the cache, it's retrieved from RAM and uses predictive algorithms to move more data from RAM into the cache. The cache controller analyses the requested data and tries to predict what additional data will be needed from RAM. It loads the anticipated data into the cache. By keeping some data closer to the CPU in a cache that is faster than RAM, the CPU can remain busy and not waste cycles waiting for data.

Our simple CPU has three levels of cache. Levels 2 and 3 are designed to predict what data and program instructions will be needed next, move that data from RAM, and move it ever closer to the CPU to be ready when needed. These cache sizes typically range from 1 MB to 32 MB, depending upon the speed and intended use of the processor. In our CPU, there are two types of L1 cache. L1i is the instruction cache, and L1d is the data cache. Level 1 cache sizes typically range from 64 KB to 512 KB.

Computer BUSes

These are pathways that data travels when the CPU is communicating with RAM and other input-output devices. There are 3 types of Computer Buses.

An address BUS carries the address of where data is going to and coming from in the RAM.

A control BUS carries control signal data that is used to control the communication between devices.

A data BUS it carries the real data that is being transmitted from one location to another.

References:

https://www.redhat.com/sysadmin/cpu-components-functionality (CPU components and functionality)

https://www.knowcomputing.com/functions-types-components-of-processor-cpu/#4-functions-of-computer-processor-cpu (Another version of CPU components and functionality)

https://onlinelibrary-wiley-com.proxy.bnl.lu/doi/epdf/10.1111/comt.12056 (Passé Media: Communicationand Transportation on Commuterand Computer Buses)

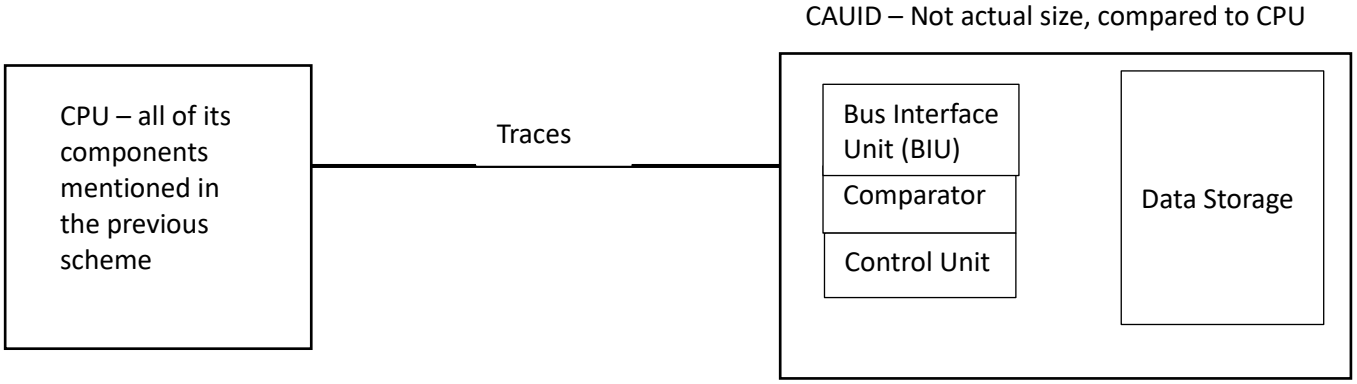
Notes :source code draw cpu + chip, seperate memory from cpu, careful pipeline code too much commands choose simpler CPU, simulate working of cpu chip communication HDL, theoretical analysis

Week 3

CPU's based on complex instructions support capabilities:

- **CISC** (Complex Instruction Set Computer)
    - Multitasking CPU architecture, which through complex instruction completes various tasks. It divides them into multiple registers, after which the result is again saved is saved in one place. The idea is to increase the cycles per second and reduce the code length by putting emphasis on hardware.
- **RISC** (Reduced Instruction Set Computer) **Simplest**
    - CPU architecture, which divides the complex instruction into multiple reduced ones, making the cycles per round lower with comparison to CISC, however this strategy makes the CPU able to comprehend larger code sizes. This method also does not require complex hardware, but rather software.
- **EPIC** (Explicitly Parallel Instruction Computing)**?**
    - Focused on parallel working and no emphasis on hardware
- **VLIW** (Very Long instruction Word)**?**
    - Overlaps instruction execution, within single control flow

Scheme of CPU and CAUID



Bus Interface Unit (BIU) – responsible for being able to read all data transferred from the CPU

Comparator – responsible for being able to compare all incoming data with the one stored within the Data storage of CAUID

Control Unit – responsible for handling unexpected errors, such as false positives, communication problems and data corruption

Data Storage – responsible for archiving various predetermined methods of security penetration.

Notes: Recog branch incremate code emulate cpu without calculaton follow address, if pepheral if data, define. Firewall is network orianteted. Protocal instructions gemform, chip on motherboard for managing network (give orders to it from CAUID, cpu sees cauid a peripheral, token to identify observe what is sequence of buses and take appropriate action, define, simulation of cpu cauid, proof technical

References:

https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/ (CISC vs RISC architectures)

https://www.youtube.com/watch?v=6Rxade2nEjk&ab_channel=ComputerScience (RISC versus CISC)

https://www.ru.nl/publish/pages/769526/roland_verbruggen.pdf (Creating firewall rules with machine learning techniques, Roland Verbruggen)

https://www.youtube.com/watch?v=I7NrVwm3apg&ab_channel=Simplilearn (Machine Learning Algorithms | Machine Learning Tutorial | Data Science Algorithms | Simplilearn)

Week 4

Booting/Booting process: Process of starting (booting up) a computer. In order for this to happen The OS needs to be loaded into the main memory. There are multiple types of booting but they follow a 6 step procedure:

1. **Startup:** PC supplies electricity to the CPU and BIOS
    a. **BIOS:** Basic Input/Output System. Assists with the activation of all hardware devices
2. **BIOS: Power on self-test:** Test initialized by BIOS, which checks all input/output devices for errors and indicates, if one is detected **(Add Peripheral Firmware File to CAUID?)**
3. **OS Loading:** OS which is loaded into the main memory, will execute all needed initial files and instructions.
4. **System Configuration:** Drivers are loaded into main memory
    a. **Drivers:** Programs needed to optimize performance of peripheral devices
5. **System Utilities Loading:** As the name suggests, system utilities are loading into the main memory.
    a. **System Utilities:** Basic functioning programs (e.g. volume control)
6. **User Authentication:** If password is set, the system asks for said password and if written correctly, the computer is usable.

Tokenization? - Tokenization is the process of replacing sensitive data with unique identification symbols that retain all the essential information about the data without compromising its security.

> **Data Tokenization? -** Data tokenization is a method of data protection that involves replacing sensitive data with a unique token or identifier. This token acts as a reference to the original data without carrying any sensitive information. These tokens are randomly generated and have no mathematical relationship with the original data, making it impossible to reverse-engineer or break the original values from the tokenized data.

Byte-Level Tokenization/BPE: Form of data compression algorithm in which the most common pair of consecutive bytes of data is replaced with a byte that does not occur in that data.

Increment: Generally an operation in programming, which increases the value of a variable by a certain amount, in terms of a CPU this can be used to compare sequences of data.

Pattern Matching: Pattern matching is the checking and locating of specific sequences of data of some pattern among raw data or a sequence of tokens. Unlike pattern recognition, the match has to be exact in the case of pattern matching.

Notes: Change scientific question, Encrypt data in the memory so if you remove chip, it doesn't have, some instructions can't be executed without a certain token, flag within cpu, group of instruction restricted to a cpu mode, comms with cpu (send address), pci or usb not usb, inserted between memory and cpu, how to prevent execution without having the hardware regularly, how, focus on specific particle for scientific question. Simple cpu – 1 sequence of code and data, avoid, further inside cpu (register), also can use address bus maybe detect jump instructions monitor the jumps, trace what are the instruction and what aren't, all devices go through the database, depends on cpu implement follow flow of code just recognize of jump, advice 1 week, use word

References:

https://www.toppr.com/guides/computer-science/computer-fundamentals/classification-of-computers/concept-of-booting/ (Concepts of Booting)

https://flint.cs.yale.edu/feng/cos/resources/BIOS/#:~:text=The%20BIOS%20performs%20a%20%22system,kind%20of%20memory%20it%20finds. (How does system bootstrap)

https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0 (Byte-Pair Encoding: Subword-based tokenization algorithm)

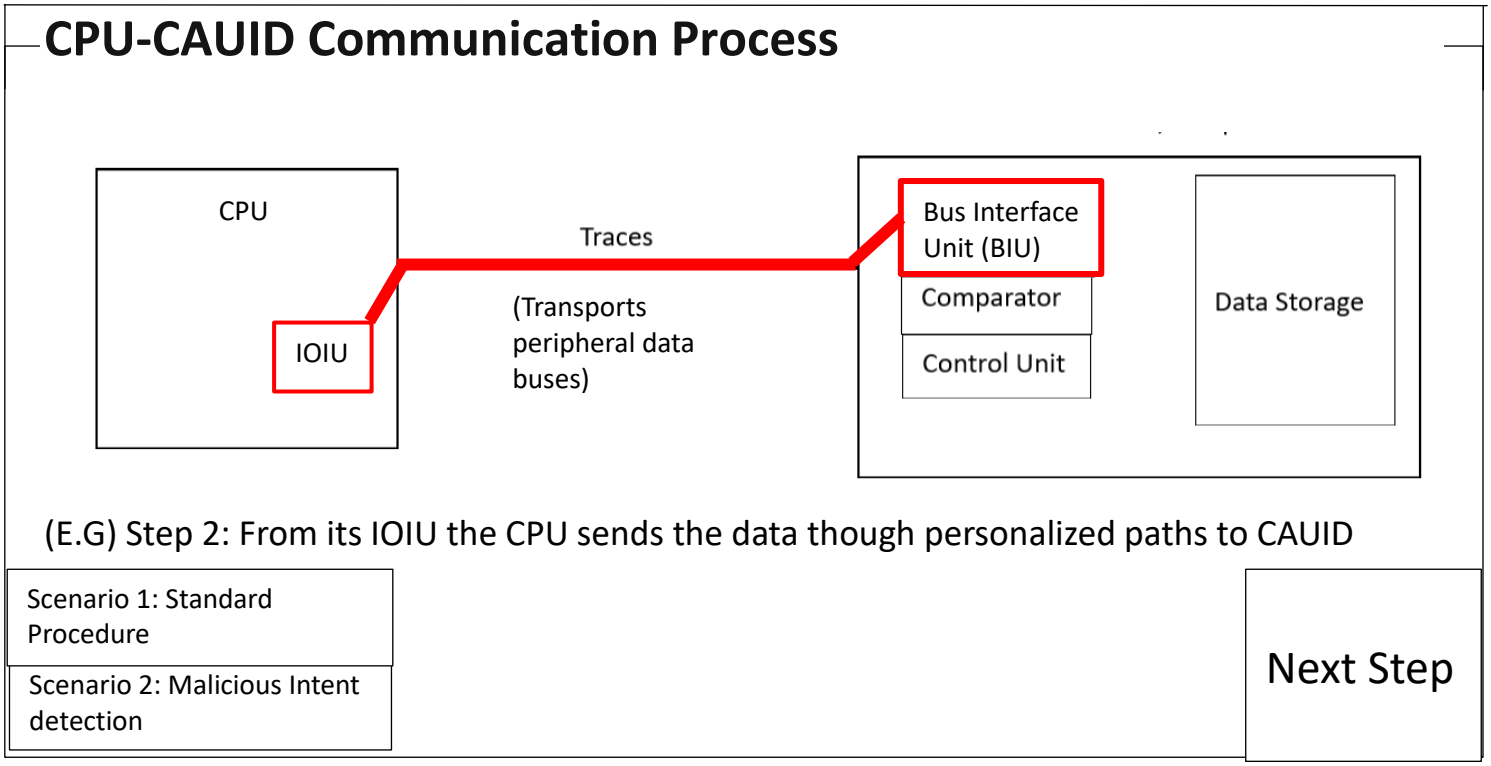
https://www.techopedia.com/definition/8801/pattern-matching (Pattern Matching)

Week 5

Questions

Week 6

Idea on how the Python CPU-CAUID simulation will work



Notes: Assimilate CPU, give files to, memory, memory cpu and cauid with old path, insert chip between cpu and memory, memory data encrypted so cauid decodes, scientific part question hardware vs software. (pros and cons) prove why it's better with hardware, program not animation but actual method no algorithm needed, detect by either chip recognizing instructions that are not related to code or monitoring the address bus (chip should detect changes aka jumps), sequence in database

Week 7

Hardware vs Software Antivirus (pros of hardware)

Improved security isolation: Hardware-based antiviruses offer improved protection against malware and other attacks by operating at a lower layer of the system stack. Their independence from the operating system and software programmes makes them less vulnerable to malware compromises. Compared to software-based solutions, which are vulnerable to flaws in the operating system and applications, hardware-based solutions have a lower attack surface since they are implemented at the hardware level. As a result, there is a decreased chance that successful malicious intents or security breaches may target the antivirus programme itself. Hardware-based antivirus programmes can protect the confidentiality and integrity of their databases and detection methods by working in a specialised, separated environment that keeps unauthorised users and bad actors out.

Malware Resistant: Hardware-based antivirus solutions are inherently more resistant to tampering or evasion attempts by malware. They are difficult to disable or get around by malicious software since they work directly with the underlying hardware. In order to gain access or disable security measures, malware usually targets vulnerabilities in software components, such as the operating system or antivirus software itself. Nevertheless, hardware-based systems are less vulnerable to these kinds of attacks because they function independently of these software elements. Furthermore, hardware-based integrity checks

and secure boot procedures can be used by hardware-based antivirus programmes to assure that the antivirus defence is reliable and uncompromised. Because of that, the malware finds it more difficult to influence or obstruct the antivirus solution's functionality.

Scalability: Hardware-based systems capacity to utilise specialised hardware resources is one of the main elements influencing their scalability. Hardware-based solutions transfer computing to specialised hardware components as opposed to software-based solutions, which mostly rely on the CPU for processing. As a result, they can manage growing workloads and data volumes without overwhelming the CPU or lowering performance.

Efficiency: Hardware-based antivirus systems are made to maximise efficiency in fundamental security operations like response, detection, and scanning. They carry out these activities in parallel with the CPU by utilising specialised hardware, such as processing units and memory, which minimises bottlenecks and maximises resource utilisation. Hardware-based systems can also achieve high levels of efficiency by reducing overheads and optimising security operations. They can perform real-time monitoring and analysis of system activity without impeding system performance, ensuring that security operations are seamlessly integrated into the overall computing environment.

CAUID data recognition methods (could be combined)

Potential sub-topic: Pattern Recognition Engine (Integrated into CAUID?)

Pattern Recognition Engine: Recognizes patterns that point to recognized threats or malicious activity. It continuously scans for suspicious behaviour using a specialised algorithm designed for real-time analysis of memory access patterns. The engine operates in real-time and may identify possible security problems rapidly because it keeps track of patterns linked to malware, exploits, or unauthorised actions in a database. The engine constantly refreshes its database using adaptive learning techniques in order to adjust to new threats and changing assault methods. It often employs a multi-stage detection process, including pattern matching and advanced analysis techniques, to enhance detection accuracy while minimizing false positives.

Method 1 (Recognizing CPU instructions):

Instruction Interception Unit (IIU): A part of the hardware architecture, which is responsible for intercepting CPU instructions prior to their execution. In order to record instructions as they are processed, this device normally interfaces directly with the CPU or memory controller at a low level inside the system. As instructions are fetched and decoded by the CPU, the unit captures them in real-time. By actively analysing instruction execution, CAUID is able to detect and respond to possible security issues.

Method 2 (Monitoring CPU database):

Database Monitoring Unit (DMU): A part made to monitor and analyse the database of the CPU in real time. With this unit, the contents of the database may be instantly acces-

sible because it works directly with the database of the CPU. Through the process of intercepting memory access requests directed towards the CPU's database the DMU searches for any changes or anomalies. Because of its real-time methodology, database changes are detected instantly, enabling CAUID to swiftly react to any potential security risks or unauthorised access attempts.

Week 8

Modification of the CPU:

New Bus Protocol: Defines the data sharing mechanism between the CPU and CAUID, making it easier for CAUID to receive data. It is also responsible of informing CAUID about jumps within the CPU.

Jump Detection Unit: Implementation which monitors programme counter and the execution of instructions. The JDU detects jump instructions while they are being executed and keeps track of all changes in the program counter and when it detects a change, it raises a flag signalling a jump event in response to the detection of a jump instruction. The communication bus is then used to send this data to CAUID though a specialized instruction which the chip recognizes.

Flag: Collection of binary values, that is used to represent a state or condition. In the context of computers, "raising a flag" refers to putting one or more of these binary values in a particular state, usually to indicate that a particular circumstance or event has taken place.

CAUID translation from machine code:

Translation Module: Module used from CAUID to translate the machine code data received from buses and to translate instructions sent from CAUID to the CPU. This module does not translate the full sequence at once, but rather dynamically decodes each instruction as it is received. Before being used for additional analysis, translated instructions are momentarily kept in the Translation Module's output buffer.

2 options, either chip directly integrated on chip of cpu or cpu sending new signals, no flags (result of arithmetic operation changes flag), no need to monitor the code, just monitor the sequence of instructions in registers then its not necessary, reverse compile symbolize, sequence of code not, 10 consecutive instructions could be a virus is simplest, if symbolize it can capture a family of viruses different of, in database different, compare instructions to database, cpu is running very, think of comparing everything in real-time 1 instructions every cycle, multiple comparators, parallel

Week 9

CAUID integrated into CPU and detects instructions.

Possible components for implementation:

Pipelining – The technique of gathering instructions from the processor via a set of data processing elements connected in a sequential order. The goal of pipelining is to reduce CPU usage. As one instruction progresses through the pipeline, the next instruction enters the pipeline, overlapping with the previous one. This overlap enables concurrent processing of multiple instructions, significantly improving overall throughput. Pipelining reduces the average instruction completion time by dividing the workload into smaller, more manageable tasks. Pipelining also helps CAUID to maximise **real-time analysis**. Through the use of 4 comparators, CAUID is able to analyse instruction sequences in parallel manner, guaranteeing prompt and effective identification of possible security risks without interfering with CPU function.

Parallelization - Before mentioned 4 comparators that CAUID will have will analyse instructions simulations in order to increase processing speed

Instructions fetch - Retrieves instructions from memory for execution. Based on the value of the programme counter, the CPU fetches said instructions. Afterwards an address identifying the location of the next instruction is sent to the memory unit at the start of the procedure. After retrieving the instruction, the memory unit transmits it back to the CPU. In order to enable real-time interceptions of the fetched instructions, CAUID's hardware is integrated into the CPU's instruction fetch pipeline during this procedure. Concurrent with the CPU's normal operations, the hardware of CAUID analyses incoming instructions and compares them to known patterns stored in its database.

Real-time analysis – In terms of CAUID, real-time analysis means constant monitoring and comparing of sequences of instructions executed by the CPU. This however would increase the usage of the processor, which could lead to potential performance issues, therefore in order to optimize the system, we implement cache utilization

Cache Utilization - Efficiently manage the usage of a CPU's cache memory. In terms of CAUID, cache utilization is able to optimize the storage and retrieval of frequently accessed data, such as instruction sequences and known patterns, in the cache memory. The cache acts as a high-speed buffer between the CPU and CAUID's analysis components, holding copies of recently accessed instructions and patterns from CAUID's database. When analysing instruction sequences, CAUID first checks the cache for matching patterns. If a match is found in the cache, CAUID can quickly identify potential security threats without the need for time-consuming pattern comparison against the entire database. Effective cache utilization ensures that the most relevant and frequently accessed data is stored in the cache. That in return reduces access latency and speeds up processing.

What is pipelining?: https://www.techtarget.com/whatis/definition/pipelining#:~:text=Pipelining%20creates%20and%20organizes%20a,and%20exit%20from%20the%20other.

What is the Instruction Fetch Unit? https://link.springer.com/chapter/10.1007/978-3-031-01729-2_3#:~:text=The%20instruction%20fetch%20unit%20is,to%20compute%20the%20fetch%20address.