

Hardware Security
Prof. Debdeep Mukhopadhyay
Department of Computer Science Engineering
Indian Institute of Technology, Kharagpur

Lecture – 01
Introduction to Hardware Security Part – 1

So, welcome to this MOOC course on Hardware Security. So, today we will try to introduce this topic. So, it will be the part one on Introduction to Hardware Security.

(Refer Slide Time: 00:24)



So, to start with the concepts that I will be covered in today's class is trying to understand the relationship between cryptology and hardware security, trying to reflect about the advantages of hardware in security which is in the form of developing hardware accelerators. And will also try to discuss about like where hardware can bring in advantage in terms of security, and why hardware why security should be an important design criteria for developing hardware right from the beginning and not an afterthought.

(Refer Slide Time: 00:53)

Cryptology

- **Objectives:** Aims at design and analysis of algorithms to ensure:
 - Confidentiality
 - Integrity
 - Availability
- Recent days there has been humongous advancement in cryptology.
- New primitives developed giving us additional capabilities:
 - Performing operations on encrypted databases in cloud.
 - Post-quantum Cryptography, etc.
- **“if you think your problem can be solved by cryptography, then you do not understand cryptography and you do not understand your problem”-[Bruce Schneier]**

The slide features a yellow background with a blue and orange header. At the bottom, there are logos for Swamyam and a navigation bar with various icons. A small video inset of a man is visible in the bottom right corner of the slide.

So, to start with cryptology which is essentially the, I would say the more classical counterpart of computer security it tries to aim at design of analysis of algorithms or analyse algorithms to ensure confidentiality. That means, it tries to hide data from adversaries he tries to provide integrity; that means, it tries to ensure that the data is not modified by adversaries and at the end of the day it tries to ensure that the data is available to user. So, it tries to make us ensure availability of information.

So, recent days there has been tremendous growth of cryptography which has essential is due to the development of several primitives which tries to bring for several interesting applications. Like for example, operations on encrypted databases in the cloud, it tries to develop cryptographic systems which are even resistance against the development of quantum computers which are called in the form of post quantum cryptography.

In spite of the development of cryptography practitioners have shown that development of all you know like mathematically secured programming algorithms are just the beginning. So, to quote Bruce Schneier who is a very famous computer security expert if you think that your problems can be solved by cryptography then we do not understand either cryptography or we do not understand the problems. So, therefore, we need to develop too. So, that at the end of the day cryptography is really realised in our systems either in hardware or software in a proper fashion in a correct fashion so that we have end to end security.

(Refer Slide Time: 02:22)

Need for Hardware Accelerators for Cryptology

- **Hardware vs Software speeds:**
 - Sensitive operations in the internet is becoming very important aspect of web applications.
 - E-commerce and net-banking require transfer of sensitive information and are protected by Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols.
 - Open SSL is an open source implementation of SSL and TLS, and accompanying cryptographic algorithms.
- **OpenSSL has implementations of symmetric ciphers, asymmetric ciphers, hash functions, etc.**
 - However, they are time consuming in software!

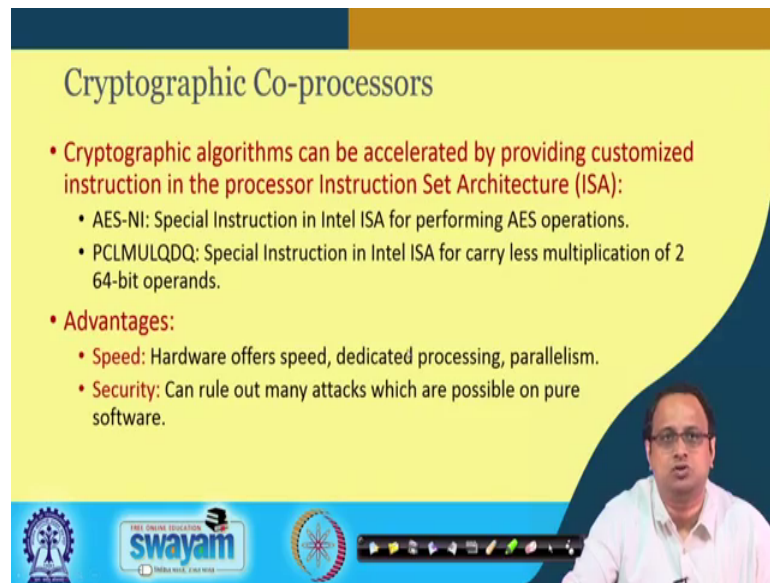
The slide features a yellow background with a blue and orange header. A video inset in the bottom right shows a man in a white shirt speaking. The bottom of the slide contains logos for 'swayam' and 'THE ONLINE EDUCATION' along with a navigation bar.

So, the one of the primary requirements of hardware in terms of making you know like in the context of security is today, is to address the issue that most of the cryptographic algorithms are very complex and therefore, they essentially have got a huge penalty on performance.

So, we all you know like in some form or the other at being touched upon on which cryptography. For example, we are either using computing systems for E-commerce, we are using credit cards, we are doing we are essentially you know like doing several transactions on the internet which is essential which all which all are in the underlying right does cryptographic operations. So, therefore, right I am the whole point is like when you are trying to accelerate your these operations then at the underlying right you are doing operation through what are called as secured socket layer or SSL and transport layer which is like a TLS which are TLS protocols.

So, if you really want to develop an open source I mean if you really want to understand these protocols then open SSL is a very common and open source implementation of SSL and TLS and the and the accompanying cryptographic algorithms like hash functions and other things like. And at the bottle is that they are time consuming in software. So, therefore, if you really want to implement them on a normal CPU then they are complex and cumbersome. So, therefore, there is a penalty in terms of clock cycles which you need to pay if you really, if you just purely rely on software.

(Refer Slide Time: 04:02)



The slide is titled "Cryptographic Co-processors" and features a yellow background with a dark blue curved shape on the right side. It contains the following text:

- **Cryptographic algorithms can be accelerated by providing customized instruction in the processor Instruction Set Architecture (ISA):**
 - AES-NI: Special Instruction in Intel ISA for performing AES operations.
 - PCLMULQDQ: Special Instruction in Intel ISA for carry less multiplication of 2 64-bit operands.
- **Advantages:**
 - **Speed:** Hardware offers speed, dedicated processing, parallelism.
 - **Security:** Can rule out many attacks which are possible on pure software.

At the bottom of the slide, there is a blue banner with the Swamyam logo and the text "THE ONLINE EDUCATION swamyam MEDIA WISE. LEARN WISE." A small video inset in the bottom right corner shows a man in a white shirt and glasses speaking.

So, what hardware brings in this context is acceleration. So, you can make something which are called as cryptographic co-processors and you can extend your current instruction set architecture or ISA to essentially upload some of your cryptography computations on this hardware or specialized hardware.

One of the, I just enlisted here two of the very commonly known accelerator which is in the form of what is called as AES-NI which is a special instruction in Intel ISA for performing AES operation. And the other one is what is essentially shown here as the PCLMULQDQ which is essentially an operation for a specialized instruction in Intel ISA for performing carry less multiplication of 2 64-bit operands.

So, therefore, right I mean the advantage that hardware brings in this context is speed for sure because it can you know you can develop dedicated processing, you can paralyze your operations and therefore, you can have an end to end acceleration of your computation. But at the same time it can I mean bringing in hardware can also rule out many attacks which are otherwise possible on only pure software implementation. So, it can enhance not only, but also security.

(Refer Slide Time: 05:14)

The slide, titled "Intel's AES-NI: Instruction for AES Operations", features a central block diagram of an AES round structure. The diagram shows a 128-bit Plan Text being mapped to GF(2⁸) and processed through multiple AES Rounds (AES Round 1 to AES Round n-1). Each round includes a ShiftRow operation, a MixColumns operation, and a RoundKey addition. The final output is a Round Cipher, which is then mapped back to GF(2⁸) to produce Cipher Blocks. To the right of the diagram are two graphs showing performance metrics. The first graph plots AES-128 Throughput (Gbps) against Round Key Size (KB), showing a curve that rises and then plateaus. The second graph plots AES-128 Throughput (Gbps) against Round Key Size (KB), showing a similar trend. Below the graphs is a reference: "Reference: S. Mathew, 53Gbps Native GF(2⁸) Composite Field AES-Encrypt/Decrypt Accelerator for Content Protection in 45nm High Performance Microprocessors, VTS 2010". The slide also includes the Swamyam logo and a navigation bar at the bottom.

So, let us take a look at some of these designs. For example, this is a snapshot of Intel AES-NI instruction which was which essentially is supported in most of the modern day Intel machines, where you can actually do a round by round AES operations and you can upload them on hardware.

Now, as you see in this paper it is shown that it is a very fast design. It essentially has got something like a round 53 gbps of throughput, and has been designed on a very you know like it has been designed on 45 nano meter technology. So, this is a 2010 paper. So, it is not really, so recent as of now, but it pretty much shows that using hardware you can accelerate your software operations of cryptosystems like even like the advanced encryption standard.

(Refer Slide Time: 06:01)

A Case Study of ECC Co-processors

- Elliptic Curve Cryptography (ECC) is an efficient public key cipher.
- Elliptic Curve Scalar Multiplication

Scalar (x)
Base Point (P)

ECC Scalar Multiplier

Scalar Product (xP)

Reference:
<https://www.youtube.com/watch?v=2RVLBUncHjk&list=PL71FE85723FD414D7&index=34>

So, I will show another case study here which is on elliptic curve cryptography processor or an ECC processor. So, we will go in details about how an easy design would be made. But at this point we can just think of or conceptualize ECC as black box. So, ECC is a very efficient public key cipher and in this black box as I show here that there are two important inputs or two inputs two inputs, one of them is the base point P which is a point on elliptic curve and the other one is a scalar.

Now, an elliptic curve operation is basically an operation or is an algebraic operation which is done on a geometric construct which is an elliptic curve and finally, you end up in getting what is called as a scalar product. So, this scalar multiplication is the central block of an elliptic curve processor and therefore, we would like to accelerate this operation. If you are interested to go into a background behind elliptic curve Cryptography Corporation then you can actually go through this YouTube link here which is shown down.

(Refer Slide Time: 07:04)

A Case Study of ECC Co-processors (Contd.)

- **Elliptic Curve Cryptography (ECC) is costly:**
 - For n -bit key for characteristic 2 curves (n could be 163 or 233), number of finite field operations are roughly:
 - #Inv: $2n$, #Mul: $2n$, #Add: $4n$, #Sqr: $2n$ (for affine co-ordinate implementations).
 - On Intel Core II Duo with 2.93 GHz and OpenSSL 1.0.1c, average time for 10000 scalar multiplications is 8.891 s.
 - A hardware accelerator prototyped on Altera Cyclone IV FPGA requires around 4.397 s (double speed up!)

Reference: Debapriya Basu Roy, Shubham Agrawal, Chester Rebeiro, Debdeep Mukhopadhyay: Accelerating OpenSSL's ECC with low cost reconfigurable hardware. ISIC 2016: 1-4

swayam

But at this point I am just try to kind of show you that how complex these operations are. So, now, there is there is a flavor of elliptic curve cryptographic operation which are called as characteristic tool designs. I will be again explaining what characteristic means in my following classes.

So, here suppose imagine that n would be around 163 or 233 according to the standards and I just try to enlist here number of to finite field operations which are done in by your elliptic curve crypto processor. So, the number of inversions which are a very, which are very complex finite field operations again you can observe that the dimension is something as big as 233 is something like we just proportional with n and I shown exactly as 2 into n . The number of multiplication is 2 n , the number of additions is 4 n , number of squarings is again 2 n . Now, these are all done on affine coordinates which means that the elliptic curve point is being denoted in the usual $x y$ coordinates system.

So, if you take an Intel core II duo processor which is an old system and with 2.93 Giga Hertz and open SSL version of 1.01 c then the average time which is require for doing 10000 scalar multiplications is something like 8.891 seconds. So, here we show and design where we just comment and design which is a hardware accelerated which is win prototyped on Altera Cyclone IV FPGA which is a very popular FPGA device and a fairly low cost design, fairly low cost platform where it will require around 4.397 second which is almost like a double speed up. So, you can see that of loading or developing a

co-processor can really help you in improving your support and in improving the performance of your design in general.

(Refer Slide Time: 08:51)

The slide is titled "Hardware to Enhance Security" and contains the following elements:

- Text:** "Software Attacks: Present day computing systems are naturally vulnerable."
- Portrait:** A black and white portrait of John von Neumann.
- Diagram:** A block diagram of the Von Neumann architecture. It shows a central "CPU" block containing "CONTROL" (with "Program Counter" inside), "REGISTERS", and "ALU". To the left is a "MEMORY HIERARCHY" block. To the right are "INPUT" and "OUTPUT" blocks. A "MEMORY BUS" connects the memory to the CPU, and an "I/O BUS" connects the CPU to the input/output blocks.
- Text:** "Von Neumann (1903-1957)" and "Contributed to give a very basic model, often referred to as Von Neumann model".
- Logos:** Logos for "swayam" and "INDIA STATE & FREE KNOWLEDGE" are visible at the bottom.
- Video Feed:** A small video feed of a presenter is located in the bottom right corner.

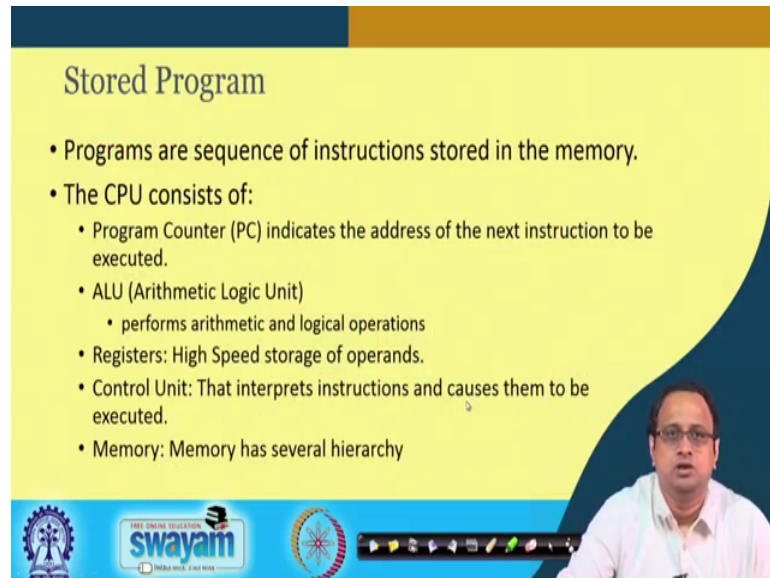
The other important application of hardware used to enhance security. So, now we know all of us know that our presently computing systems are naturally vulnerable against attacks. Everyday day in and day out there are attacks for example, we here of several vulnerable in our system. So, in order to understand the contacts and try to reflect upon why our computing systems all vulnerable against attacks let us take a look back into our architecture text books and remember the famous Von Neumann architecture.

So, Von Neumann essentially contributed to a very basic model which is called again as a Neumann model which essentially has got three important components, it has got a memory it has got a CPU, and it is not an IO component. So, the idea is that this CPU is interacts with the memory and interacts with IO bus, the IO essentially talks with external what is typically the keyboard audio or your audio display units.

So, now, the CPU essential again it has got several blocks. For example, it has got the arithmetic logic unit, it has got the program counter where your current address where the current address of your instruction reside and we have got registered first form of memory the memory. The memory in general they has got an hierarchy which means that there is the memory in typically is slower compared to the CPU and that brings what is called as a memory vault, in order to bring that memory vault we have a memory

hierarchy. For example, we have a cache memory to be to sit in between you know like the between the main memory and between your CPU browsing speed.

(Refer Slide Time: 10:19)



The slide is titled "Stored Program" and contains the following text:

- Programs are sequence of instructions stored in the memory.
- The CPU consists of:
 - Program Counter (PC) indicates the address of the next instruction to be executed.
 - ALU (Arithmetic Logic Unit)
 - performs arithmetic and logical operations
 - Registers: High Speed storage of operands.
 - Control Unit: That interprets instructions and causes them to be executed.
 - Memory: Memory has several hierarchy

The slide also features a video inset of a man speaking in the bottom right corner, and logos for "swayam" and "INDIA WISE, FUTURE WISE" at the bottom.

So, that the basic paradigm of von Neumann architecture is what is called as a stored program where the programs are also sequence of instructions which is stored in the memory like data.

So, the CPU typically consists of the program counter which indicates address of the next instruction which is to be executed. The arithmetic logic unit which performs arithmetic and logical operations, and then you have got very high speed memories which are called as registers and you have control unit with basically does or interprets instructions and causes them to be executed one after the other. You have got an, and I say we have got a memory hierarchy which means you tried to bridge between the first form of memory, I mean the fastest form of memory and also the you will and I mean and your I mean you try to bridge between the CPU which is lower terms of rising speed and also between yours slow memory or the RAM.

(Refer Slide Time: 11:20)

Stored Program and bottle-necks

- The memory stores the instructions, data and intermediate results. Memory has several hierarchy:
 - registers being the highest level and the fastest form.
- Input/ Output: Transmits and receives results and messages (information) from and to the outside world respectively.

How do you know 0x90ABCD is a code or data?
The OS (software) does not know? ☹️

Because of the shared memory bus, there is a bottleneck called memory wall:
The memory is much slower compared to the CPU speed.
Fast memories called cache, branch-predictions have been developed.

The slide also features logos for 'swayam' and 'THE ONLINE EDUCATION' at the bottom left, and a navigation bar at the bottom center.

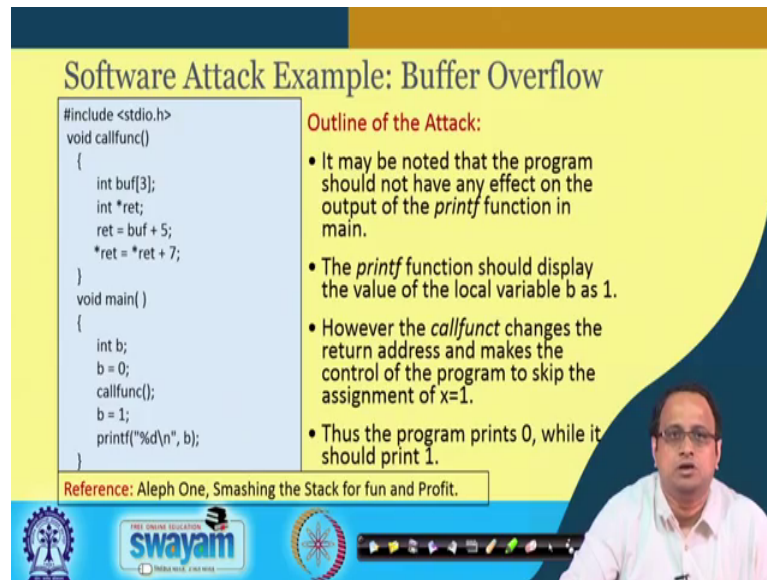
So, now when we talk about the memory right which as essentially which stores the instructions data and the intermediate results then it has got a several hierarchy whether registers have got the highest level and has got the fastest and have the fastest form of memory. Likewise right we have got the input or output block which transmits and receive results and messages or information from and to the outside world respectively. However, there are several bottle necks of the Von Neumann architecture.

For example, if I give you this as this bottle neck arises. Because of the fact that both memory I mean both data as well as the code resides in memory and therefore, how do you know that whether 0x90ABCD for example, which is a funny looking number is whether it is code or whether it is a data.

Now, the operating system or the OS essentially has got no way to distinguish between whether it is a code or whether it is a data because of the and therefore, right you essentially can have malwares which can freely execute in your systems. On the other hand because of the shared because of the shared memory bus there is a bottleneck which is called as a memory wall, and in order to reach the memory wall we have got several artifacts in computing architectures in the form of cache memories, branch predictions and so on.

Many of these attacks actually target these presence, like the presence of the cache memory or the presence of the branch predictions and therefore, it is fundamentally kind of arises because of the fact of because of the Von Newman architecture.

(Refer Slide Time: 12:47)



Software Attack Example: Buffer Overflow

```
#include <stdio.h>
void callfunc()
{
    int buf[3];
    int *ret;
    ret = buf + 5;
    *ret = *ret + 7;
}
void main( )
{
    int b;
    b = 0;
    callfunc();
    b = 1;
    printf("%d\n", b);
}
```

Outline of the Attack:

- It may be noted that the program should not have any effect on the output of the *printf* function in main.
- The *printf* function should display the value of the local variable b as 1.
- However the *callfunc* changes the return address and makes the control of the program to skip the assignment of *x=1*.
- Thus the program prints 0, while it should print 1.

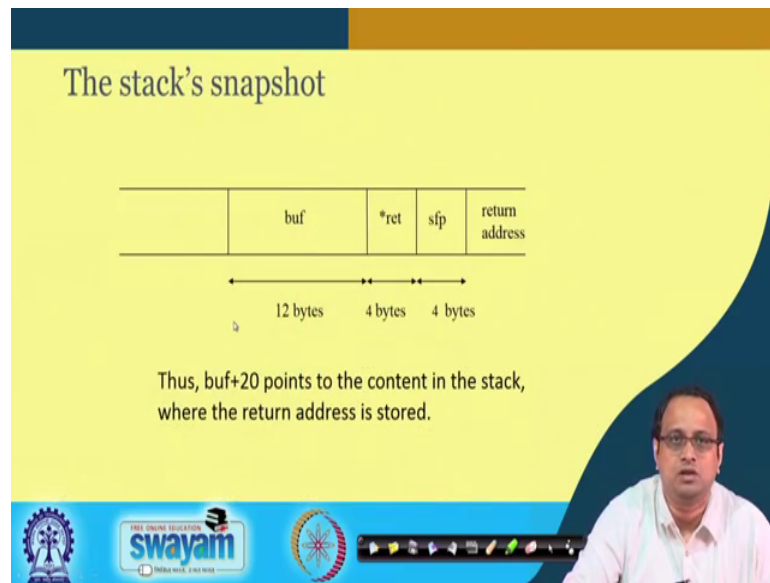
Reference: Aleph One, Smashing the Stack for fun and Profit.

The slide also features a small video inset of a man in the bottom right corner and logos for Swamyam and other organizations at the bottom.

So, here is a very simple depiction of an attack which is a purely software that which is called as a buffer overflow. It may be pointed here that buffer overflow till now remains as one of the most popular forms of software attacks. So, these attack here has been shown here that has been made of two, has been made of two you know like program or two function calls, one is the main function call and there is an another function call which is called function and has been written in C.

So, you can observe that here in the main function there is an integer variable b which is being kind of defined as b equal to 0 then there is a call to the function call function and then followed by a b equal to 1 assignment followed by a print of b of b. So, since a call function does not effect b what I would expect is that b should be printed as 1. On the other what we will see is that because of some certain operations here in the call function which essential it kind of skips this b equal 1 line and therefore, prints b equal to 0. So, therefore, the program does a wrong computation. So, this attack has been shown in the paper which is called as shown here as smashing the stack for fun and profit written another pseudo name of Aleph One.

(Refer Slide Time: 14:00)



So, in order to understand the working of the attack let us take a look at the stack. So, the stack essentially stores these the corresponding data as shown here. For example, it stored 12 bytes for buf and you can see that if you go back to the code there is a buf is a integer variable, it is an integer array of size 3.

And if you assume that 4 bytes have been located, so this is done on old machine of mine. So, if you see the 12 bytes have been allocated for buf then there is a then other allocated like 4 bytes for the integer pointer, so it is another 4 bytes and that is followed by the frame pointer which again takes another 4 bytes. So, if you take the relative distance between this point like the buf for the starting address the starting point of buf to the return address then it is something like buf plus 20 points to the content in the stack where the return address is restored.

So, now that that essentially explains these line which shows as that we do and you know like a we basically increment buf or the starting location of buf with 5 and therefore, we basically access the return address. So, therefore, int star ret therefore, points to this return address. So, what we do the next is that we increment the value of the return address by 7.

(Refer Slide Time: 15:16)

Skipping the line x=1

Dump of assembler code for function main:

```
0x080483e2 <main+0>: lea 0x4(%esp),%ecx
0x080483e6 <main+4>: and $0xffffffff,%esp
0x080483e9 <main+7>: pushl 0xffffffff(%ecx)
0x080483ec <main+10>: push %ebp
0x080483ed <main+11>: mov %esp,%ebp
0x080483ef <main+13>: push %ecx
0x080483f0 <main+14>: sub $0x24,%esp
0x080483f3 <main+17>: movl $0x0,0xffffffff(%ebp)
0x080483fa <main+24>: call 0x80483c4 <callfunc>
0x080483ff <main+29>: movl $0x1,0xffffffff(%ebp)
0x08048406 <main+36>: mov 0xffffffff(%ebp),%eax
0x08048409 <main+39>: mov %eax,0x4(%esp)
...
```

End of assembler dump.

- The address values are obtained using the gdb tool
- The output of which is provided next.
- We observe the assignment x=1 marked in yellow.
- It is evident from the output of the gdb tool, that the return address needs to be increased by 7, the difference between the memory locations 0x080483ff and 0x08048406.

So, now why 7? In order to understand that we can take a look in to the gdb dump and here is the portion of the gdb dump which shows that this is the line which have or which is been highlighted in red is the line which I would like to kind of skip because here is where b becomes or b gets assigned to 1.

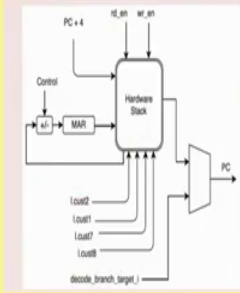
In order to skip this line we observe the difference between these addresses which are been shown here as 0x080483ff and 0x08048406. So, this difference is 7. So, therefore, what we try to do is in order to address in order to skip this line we increment these by 7 and therefore, what happens is that rather than returning to this line the program returns to the next line method skips the b equal to 1 assignment and therefore, b equal to 0 gets printed.

Now, these are very simple illustration of a buffer overflow attack, but you can probably imagine that there can be stronger forms of attacks which one (Refer Time: 16:11). For example, one can change the return address to point out to an address where a malicious program on malware is located and that can lead to the triggering of a malware in a normal computing system.

(Refer Slide Time: 16:27)

Hardware Mitigation of Buffer Overflow


Proposed Hardware Stack



- Hardware Stack stores the function return address:
- Whenever it encounters a `l.jal` or `l.jalr` instruction, it pushes the next program counter value to the stack.
- Alternatively, if it encounters `l.jr` with register `r9` as parameter, it pops its top value and passes that as the return address.

Custom instructions:

- `l.cust7`: when enabled, the return address of the functions are read from hardware stack
- `l.cust8`: Freezes the hardware stack
- `l.cust1`: Unfreezes the hardware stack
- `l.cust2`: Disables the hardware stack



swayam

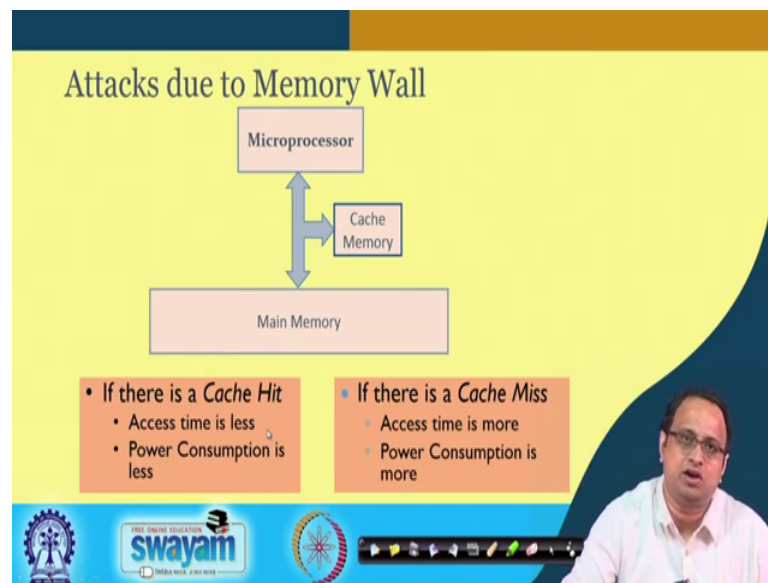
So, in order to alleviate this software only attack we essentially can bring in hardware. For example, one of the popular approaches which people can do is they can try to implement a hardware stack. So, what we need in one of our voice is trying to implement hardware stack.

Such that whenever it encounters `jal` which is the jump and link, which is a common jump instruction and similarly other instructions. What we try to do if we try to maintain hardware stack, we try to kind of develop custom instructions where hardware stack gets enabled or it gets disabled, whether where we freeze the hardware stack or we unfreeze the hardware stack. But the idea is that we try to see that whenever I mean whenever the buffer overflow tries to modify the return and return address which is probably stored in some, you know like which is stored in some value then what happens is that rather than taking the value from the software stack the data is taken from the hardware stack and therefore, right the damage that the software attack is doing remains inconsequential.

So, in this work which we protect on open risk platform where the register `r9` is essentially very important where a return address gets located. So, whenever the hardware stack encounters `jr` whenever we encounter a `jr` instruction then with register `r9` as a parameter it pops its top value and therefore, passes that as a return address rather than the value that is being actually stored in the software stack.

So, therefore, right with this simple you know like incorporation of a hardware stack we can actually an also of course, incorporation of these dedicated instructions which are shown here as custom 7, custom 8, custom 1 and custom 2 we can actually alleviate the effect of a buffer overflow attack on our software.

(Refer Slide Time: 18:21)



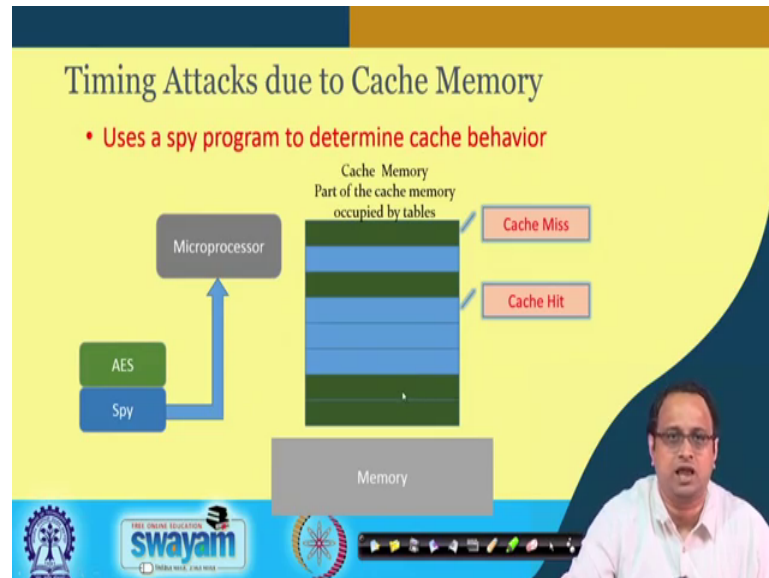
The other point which I would mention is that, which I try to hint upon is that because of the memory wall we have the cache memory which tries to basically write the idea is that you have got a very fast memory. But on the other hand the main memory or the memory in general did not become fast as in the same phase as the CPU.

So, therefore, right what you have is you have got a cache memory, which tries to bridge in this; which tries bridge in this time differences. And therefore, right what in general happens is that if there is a cash sheet then access time is less than and the power consumption is less. On the other hand if there is a cache miss then the access time is more and the power consumption is more. So, that implies that you typically write in the computer architecture try to improve the cache sheet, so the data is largely found in the cache memory and not and you do not really need to go into the slow form slow memory.

But you can also observed that the adversary if it is able to properly time and kind try to distinguish between two access, the timing difference of two accesses, then it can try it

can figure out whether there is a cache sheet or whether there is a cache miss. Now, this can lead to trivial leakages of the secrets keys.

(Refer Slide Time: 19:37)



So, in order to illustrate this I will just to here a very famous attack which is called as in the popular name of what is called as prime and probe attack, whereas spy comes first comes in an fills up the cache memory with some garbage data.

And then the access is given to the target encryption which is in this case an AES. Now, when the AES comes in it basically access certain portions of the cache memory and therefore, right it basically evicts some data from the cache. Now, if the handle comes back to the spy program then the spy program starts timing its own access, then you note that these data is being you know like is being kind of evicted from the cache and therefore, what we will expect is that these access we have got more time and that is because of the fact that there has been cache misses.

So, the adversary can distinguish between cache sheets and cache misses and that can reveal to the adversary the footprint of the of AES encryption. So, the adversary can know where exactly the encryption accessed and that can lead to trivial leakages of the key as we will see in our subsequent classes.

(Refer Slide Time: 20:45)

Micro-architectural Attacks: Design for Security

Computer Architecture has been designed with performance as a primary design criteria. Security has been an after thought. For example: Speculative execution is an optimization technique where a computer system performs some task that may not be needed. This has been the basis of the recently discovered attacks: Spectre and Meltdown.

To quote Bruce Schneier, "Fixing them either requires a patch that results in a major performance hit, or is impossible and requires a re-architecture of conditional execution in future CPU chips. It shouldn't be surprising given that microprocessor designers have been building insecure hardware for 20 years. What's surprising is that it took 20 years to discover it."

Reference: Spectre and Meltdown, Daniel Gruss, Moritz Lipp, Yuval Yarom, Paul Kocher, Daniel Genkin, Michael Schwarz, Mike Hamburg, Stefan Mangard, Thomas Prescher and Werner Haas, Google Ground Zero Project

The slide also features logos for Swamyam and other educational institutions, and a video player interface with a presenter's video feed in the bottom right corner.

So, therefore, in general micro architecture attacks target our present day a computer architectures because our present day computer architecture have been fundamentally design with performance as an design criteria. Security has always been in afterthought. For recent days there has been a famous attack is called as spectre and meltdown which essentially showed that what the targeted was an optimisation technique which is called as speculative execution, where a computer system perform some task that may not be needed.

For example, a branch; for example, a branch is because you know like some because there is in a pipeline processor because of a branch there is a significant amount of penalty that needs to be incurred if the branch fails. And therefore, there is an artifact in computing architecture which are called as branch predictors which tries to predict whether a branch should be taken or not taken.

So, these kind of artifacts which are in general called as speculative execution are targeted by this by these attacks which are called a spectre and meltdown, but you can observe that since these design principles are so deep rooted in our present architectures that suddenly changing them is real hard task. So, again to quote Bruce Schneier fixing them either required of patch the results in major performance it or is impossible and requires an architecture I mean re-architecture of the entire CPU, of the entire CPU with

which is kind of infeasible. And what is more surprising is that it took so many years to discovered such kind of attacks.

(Refer Slide Time: 22:20)



On the other hand security can be a game changer. This shows, this diagram shows how the stock sheet Intel for example, when there was this discovering be at the beginning of this year and it shows that the Intel stock fails significantly. For example, the Intel shared went down by nearly around 5 present and therefore, there I mean it shows that security vulnerability can indeed become quit costly.

(Refer Slide Time: 22:51)

Conclusion

- Conclusion:**
- Hardware Security has profound influence on modern day.
- Hardware provides opportunity to accelerate cryptographic operations
- Security Bottleneck in Von-Neumann Architecture and Micro-architectural Attacks
- Design-for-Security

So, therefore, right it brings us to the fact that we need to make about, we need to make security as an important design criteria, and hardware security has got all the its new in terms of its discovery and in terms of its existence in literature its quit resending in that context. But still it has got profound influence on the modern a world. Hardware profile provide us opportunities for excellent in cryptographic operations, but at the same time award architecture of the present architecture has been fundamental the developed with performance in mind and therefore, there are significant flows which can be targeted by adversities. And therefore, what we need is we need proper design for security guidelines to ensure that our hardware is dose not only give us performance, but is also secured.

So, with this I would like to thank you.