

**Secure Systems Engineering**  
**Prof. Chester Rebeiro**  
**Indian Institute of Technology Madras**  
**Mod\_01 Lec\_01**

Hello and welcome to this course of Secure Systems Engineering, this course is a fifth in the series of courses on information security that is offered by NPTEL, so this course is an eight-week course and during this course will actually look at details about, aspects about security systems, and essentially will look at aspects about how systems can be built to be more secure, so let us start this particular class with a small introduction about what we mean by Security Systems, if you look at a computer system. What are the threats for that computer system? And what are the current practices to mitigate this threats.

(Refer Slide Time: 0:58)

---

## Secure Systems

- Computer systems can be considered a closed box.
- Information in the box is safe as long as nothing enters or leaves the box.



CR

---

So let us consider that a computer system is a close box like this, now this is a box and there is no connections to the outside world, so as long as nothing enters this box or nothing leaves this box the content of this box is safe, in a very similar way, when we consider computer systems if the computer is totally disconnected from the external world, no information is going into the computer and no information is going outside a computer, then we can say that computer system is secure.

(Refer Slide Time: 1:33)

---

## Systems Still Secure

- Even with viruses, worms, and spyware around information is still safe as long as they do not enter the system



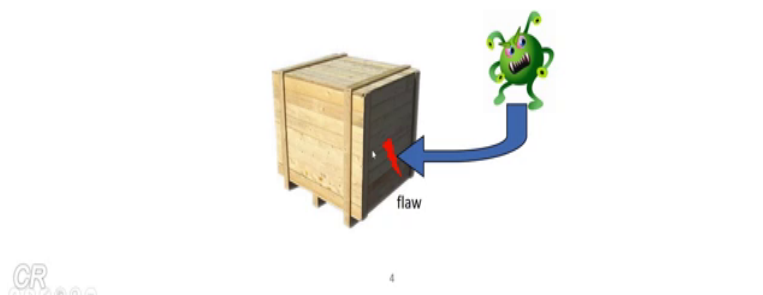
However, this is not what happens in practice, in practice there are a lot of viruses, worms and spyware which are around a computer system, so these spyware may not be present in the system, but it will be outside the system, and in such a case the system still secure, the reason is that the information content in the system is enclosed by this box and is not affected by the external malware or viruses or spyware, which is outside this particular box, therefore, in such a scenario also we can consider that the computer system is still secure.

(Refer Slide Time: 2:17)

---

## Vulnerability

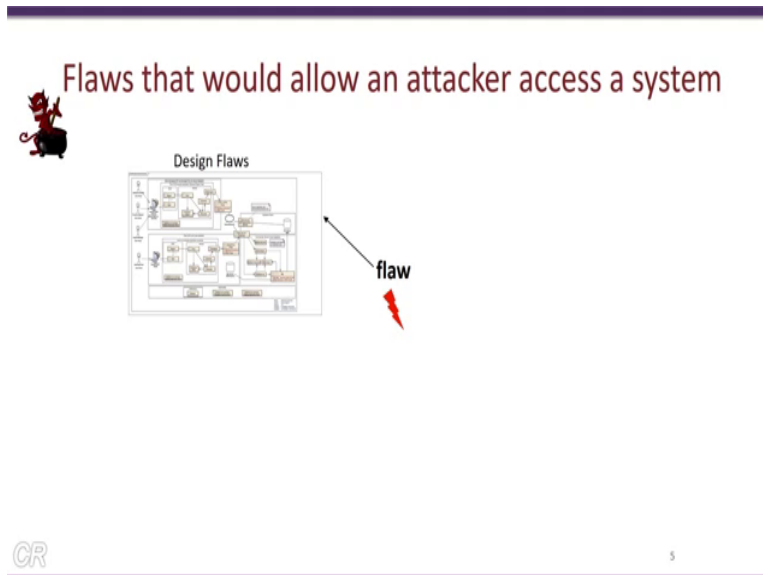
- A flaw that an attacker can use to gain access into the system



Now, consider that there is a flaw in this box, okay, so there is small flaw by which an external malware or a virus or spyware could enter into this particular box and would result in

the system being not secure. Therefore, it is only this particular flaw, which is shown over here that could lead to a system being not secure.

(Refer Slide Time: 2:40)

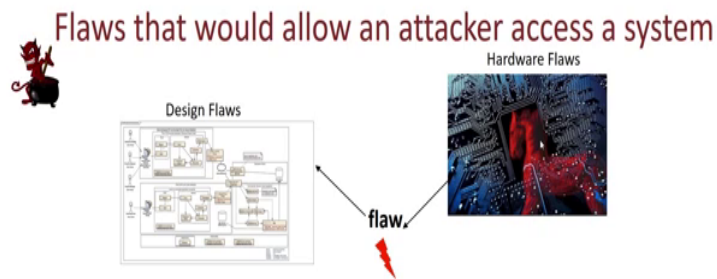


Now, let us look at what the different types of flaws are, so we could actually categorise a different flaws in a computer system, in a multiple categories, so one thing is a flaw in the design, example we have a processor and a processor as we know, is a highly complicated or highly complex design, there are multiple molecules are large number of pipelines large memories and all of these blocks are actually interacting with each other at very high frequency and done a lot of operations per nano second.

Now, even a small flaw in this entire design could result in a wonderbilty and it could lead to an exploit that an attacker could use to gain access into that particular system, so one quite famous flaw the Intel Pentiums floating-point bug, I would not go to into the details of this particular a flaw, but you could actually look it up on Wikipedia and so on to see a roughly in the mid-90s, how a flaw in the floating-point unit of Intel processors had led to a wonderbilty where when you do a floating point operation, the result would be incorrect.

Now this particular flaw much in several years later was then exploited by cryptographers to create a tax on ciphers and therefore predict secret keys of the cipher by exploiting this floating-point bug.

(Refer Slide Time: 4:22)



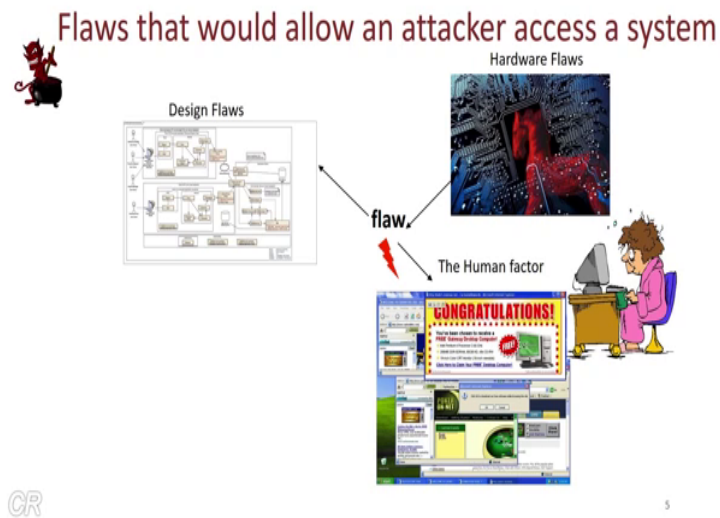
CR

5

Next flaw that could hacker is hardware flaws, so these flaws could be intentional hardware Trojans that are inserted at the time of manufacture by third parties. For example, when a company actually designed circuit or design a VLSI chip and sends it for fabrication to a third party, hardware Trojans may be inserted, this Trojan will be typically domain and not easy detectable during the testing time of this particular chip, however, then this Trojan gets the right trigger, then it wakes up and it could get access to the information that is getting computed in the chip.

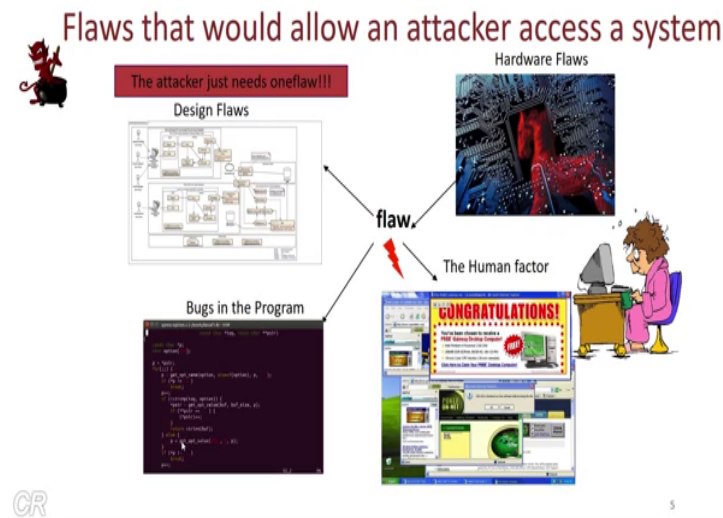
So these are other flaws and these hardware Trojans are big threat to computing systems these days, essentially because it is extremely difficult to detect whether an IC or a processor or a chip is having a hardware Trojan present within it.

(Refer Slide Time: 5:26)



The third aspect is of course what we are all familiar with it is the human factor and although we would actually built a computing system. Let us say with no design flaws or with somehow guarantee that there are no hardware Trojans or no hardware flaws, still eventually would be a human who actually uses this system at least the many of the systems like smart phones or laptops or desktops and so on, and this could be a reason for a wonderbilty, for example, all of us would have got some form of spam ware or expand emails or pop-up windows that actually come up like as shown in this particular a slide and it is not surprising that many people actually fall prey to these pop-ups and spam emails and would click on the buttons pressing here as a result of this, it could trigger and external malware, virus or worm or anything to enter into your system.

(Refer Slide Time: 6:37)



The fourth flaw that we were actually be looking at and spending a lot of time during this course are due to bugs in the program. So then could be several bugs in a program which do not get detected by the compiler or while testing the program, so let us for example, say that we have programmer who is writing a program to say sort hundred numbers present in an array, so he would use one of the standards sorts, like selection or bubble or quicksort and compile the program get an executable, then he would run the program given input of say hundred numbers and ensure that the output is actually sorted, but this actually shows that the program is working correctly for the given.

However, there may be other bugs which are not detected so easily and these are the bugs which are the flaws and an attacker would actually use to gain access into your system and then control the system, so what we have seen over here are different types of flaws, we have seen design flaws, hardware flaws, bugs in the program and the human factor.

Now, if an attacker wants to get access to your system, all that is required is just a one single flaw, any one of these is sufficient for the attacker to get access into your system and therefore control your entire system or steal secret data that is present in the system, in this course we will be looking at this particular aspect, so you will be looking at bugs in the system, and how an attacker could utilise these bugs to create malware or create exploits that could be introduce into your system and then could run and steal data in your system.

(Refer Slide Time: 8:29)

---

## Program Flaws

- In application software
  - SQL Injection
- In system software
  - Buffers overflows and overreads
  - Heap: double free, use after free
  - Integer overflows
  - Format string
- Side Channels Attacks
  - Cache timing attacks
  - Power Analysis Attacks
  - Fault Injection Attacks



6

There are quite a few such bugs present in different programming languages. So, but we will be actually focusing on programming a bugs present in C and C++ programs. These fall into this category of bugs in the systems software, so why we focus on C and C++ is due to the fact that many systems software such as starting from operating systems to virtual machines and lot of the underline libraries are written in C and C++ further these system software are quite large, and they are susceptible to a lot of such programming bugs which could be away that an attacker could enter and exploit the system, so we will be looking at during the course at buffer overflows and buffer overreads, heaps double frees and use after free, integer overflows and format string, so these bugs present in system software has been in the past exploited quite a bit to create a malware that could attack your system. In addition to a bugs in the system software, you could also have bugs in other applications that are present, one common example is the SQL injection bug present in modern database systems, but we will not be going into details about such application level wonderbilities.

And other aspects which is gaining quite importance is due to something known as Side Channel Attacks, so these attacks differ quite considerably from the bugs like what we have seen over here, while these like the bugs buffer overflows, heaps, integer overflows and format strings are due to wonderbilities or due to flaws during the programming, side channel attacks on the other hand, could be attacks on programs which are actually co-react correctly without any presence of any bug. Later on in this course we will be actually looking how this attacks are actually developed.

(Refer Slide Time: 10:40)

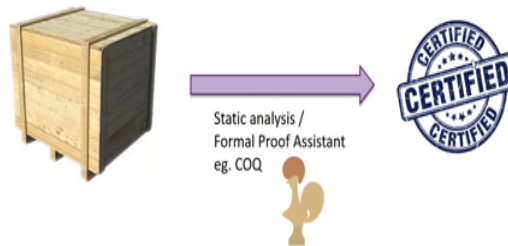
---

## Secure Systems Engineering

**Approach 1: Design flawless systems**

eg. SeL4

(Not easy to develop these systems in a large scale)



CR

---

So now what we have seen is that a wonderbility in a system can be utilised by an attacker to create malware which would enter into your system or gain access into your system through that particular wonderbility, so we have seen that there are different types of such wonderbilities or what we call as a flaws and we have seen that the flaws could occur due to design aspects, due to hardware Trojans or due to programming bugs as well as due to the human factor.

So there are many ways by which engineers and scientists have developed techniques by which these flaws or these attacks on systems can be actually prevented or if not completely prevented at least mitigated, so one of the best ways to prevent such kinds of attacks is the first approach which, and it is quite, kind of the obvious approach that one would start off with is, where we want a design systems without any flaws in such a case, what you say is that we take our system which is represented here by this closed box and analyse the system mathematically using tools such as static analysis, a formal proof assistants which has a COQ model checkers and therefore certify that the system is completely flawless and as we know if there are no flaws in the system, there are no wonderbilities and if there are no wonderbilities that can be no attacks on the system.

However, the drawback here is that such a system is very difficult to develop, the reason being that this form of analysis that is using static analysis or formal proofs are not very scalable to large codes, therefore, a lot of this activities are restricted mostly in the academic circles, so one such effort was made by an Australian group called NICTA where they actually develop an operating system called SeL4 and they have been able to prove that SeL4



is flawless under certain assumptions therefore under these assumptions SeL4 does not have any wonderbilities, these tools which can actually certify that, there are no such wonderbilities in SeL4, and therefore SeL4 is flawless and cannot be exploited under these assumptions.


However for all practical systems like standard operating systems like Linux or Windows such, doing such a analysis would be its extremely difficult, given the current a technology of these tools and therefore we would require other techniques to ensure that standard operating systems like Linux and Windows are not affected by malware or any other kind of external malicious code.

(Refer Slide Time: 13:51)

---

## Secure Systems Engineering

**Approach 2:** Isolate systems : sandbox environments, virtual machines, trusted execution environments (trusted computing)



CR

---

Now that we cannot ensure that large systems can be built without flaws, what we will actually come up with is a second approach where we will build system with having flaws, but then encapsulate this particular system in something known as a sandbox environment, so you could consider this sandbox environment as a container in which our system is actually present, even though the system has flaws, malware is actually prevented from entering into the system due to the sandbox container that is present in the system, so this technique also takes care of human factor as well. Now if a user clicks on a link in a malicious website the system would still remain secure, because of this close sandbox environment.

(Refer Slide Time: 14:44)

---

## Secure Systems Engineering

### Approach 3: Detect and Mitigate Attacks



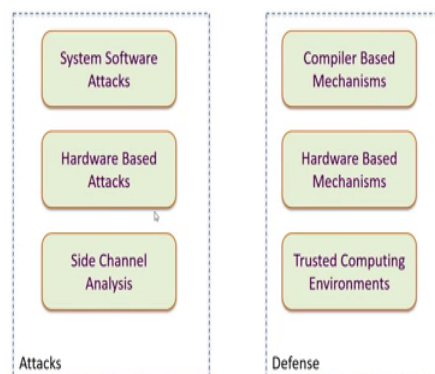
CR

The third approach is to Detect and Mitigate attacks, so this is what typical antivirus software does, so when a program executes the anti-virus software would monitor various characteristics of the program and then identify whether this program is actually trying to do something malicious or in other words, the anti-virus software would detect malicious code, based on certain characteristics during the execution or based on certain characteristics of the binary of that particular executable.

(Refer Slide Time: 15:19)

---

## Course Structure



CR

In this particular course will be looking at both attacks as well as defences, so with respect to the attacks we have been looking at attacks at the software, hardware level as well as side channel attacks. Now this software attacks are mostly focused on the system software and

therefore will be looking at C and C++ programs, we have been understanding the bugs and the wonderibilities present in these programs and how an exploit can be written to use these particular wonderibilities.

For hardware and side channel attacks we will be studying various forms of this attacks are such as the cache timing attacks, a power analysis attack and a fault injection attacks, now side-by-side will be also looking at the popular defence strategies prevent such attacks, so these defence strategies could be present either at the Compiler or at the Hardware or by building special enclaves known as Trusted Computing Environments.


(Refer Slide Time: 16:24)

---

## What to expect during this course

- Deep study of systems:
  - Software
    - Assembly level
    - Compiler and OS level
  - Hardware
    - Some computer organization features
- Expected Outcomes
  - Understand the internals of malware and other security threats
  - Evaluate security measure applied at the hardware, OS, and compiler
  - Understand trade offs between performance and security

4



---

So what you can expect by the end of this course is that you will have a good understanding about the internals of malware and other security threats, you would be able to evaluate security measures and applied them to various parts or various components from the hardware operating system and the compiler and also you would be able to trade off between the performance and security.

Now this third expect is very critical. So, for example, we could have highly secure system, or we could developed a highly secure system, however, to execute or run any application on that system would be a huge overhead and therefore it is very important to evaluate the trade off between performance of the system and the security achieved.

(Refer Slide Time: 17:08)

---

## Websites and Communication

- Reference Textbooks  
mostly research papers; will be provided as per topic
- For slides and programming assignments  
<https://chetrebeiro@bitbucket.org/cas/sse.git>

CR

---

We will not be following any specific textbook in particular, but mostly it is search papers and appropriate links would be provided at various stages during the videos and these links would be present in the slides and you could actually download these links and read those links to get more details about the concepts, so during the course will be evaluating and analysing a lot of different programs. These programs are very small, but we will go quite in depth to actually analyse these programs, so you could actually download these programs from this bit bucket repository, which contains not just a programs, but also gives you a lot of instructions about how to run this particular programs and also this repository would contains slides and other assignments that you could try it. Thank you.