RESEARCH PAPER

Name: Syed Muhammad Iftikhar Mehdi

Student ID : 7774

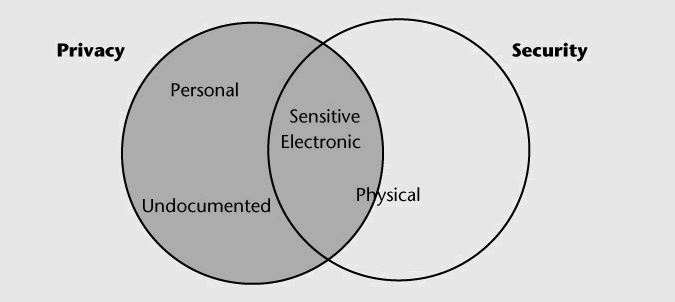
*Topic:*

**Operating Systems Security And Its System Calls**

**INTRODUCTION**

In computer security we have to deal with the technological process and safe lines which is assured to computer hardware and software. And on the other hands computer security is concerned with the responsibility to protect data and escaping to compromise from unauthorized users. The most attentive topic nowadays is to make secure your digital systems. So we can easily share information between systems, non disclosure data in our applications. If we try to find out some motivation in security we can see that how military affairs and their secrets data is secured by implementing that standard security practices.

The relation of privacy and security as we see below in figure we are trying to elaborate here that if we see on the privacy side we must focus on that “what, who and where” our documented or undocumented data concerned from, but on the security side our vision is to put a light on how our data to be committed with the systems rather our securing techniques which we used can be on physical side of our systems as we know they are actually our sensitive electronic devices.



**Fig 1:** **Relationship between privacy and data**

When the system started its work then security is implies into three levels data security, hardware security and software level security. In addition, if our systems are terminal base then we have to take care of intercommunication between these systems as terminals considerations. Firstly, we have to design and make good methodology to our systems software secure e.g. making kernel of operating system with better security and second we have to test and check the software is indeed intended with better security with authentic proofs and tests of security. If we learned that how to design and implement security on software then we can easily manage our applications as per desired security mechanisms required in our applications.

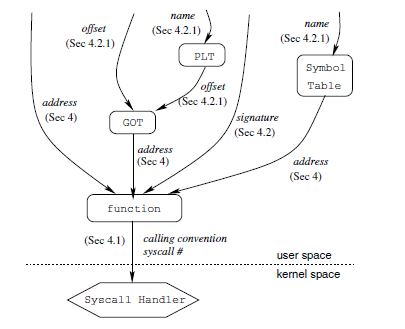
***What And Why System Calls?***

Our second purpose in this paper is to find out the ways that how we can make our self secure by avoiding the irrelevant changes in system call traps and make code malicious for our operating system and make chances to compromise our system for attackers. For this we have to know first that what is system call in actual? So the system call is actually a programmatic way to request to our operating system’s kernel for the purpose of user level program and processes. In this paper we are describing the defense mechanism of executing the system calls in host based systems for purpose of compromising the systems.

This mechanism of security defense in our operating system specifying us the location and nature of system calls in binary. By using such information our operating system kernel are able to verify the address where that system call is made, by this we can detect those system calls which are made directly for malicious or injected code for attack. Secondly, it supports us to hide actual software traps instructions in our kernel which make harder for attacker to discover libraries and those routines which are exists in our kernel.

**The Attack Model:**

In this paper our core focus are on remote code injection attacks. Suppose attacker contain the code for that victim application which he wants to attack so we here randomizing our binary not for only single or specific instances but randomized for all host based systems which have chances to get under attacks. So by this way it will be very hard and much unpredictable to find out those systems instructions and sequences for attackers.



**Fig 2: Attack Model**

In above figure we know that how we can reach ultimate goal to access the system handler . The data and functions which are represented as arrows it is the actually way that how we can get this practically. Because the main purpose or goal for attacker is to execute that handler. Attacker can easily invoke system calls if he knows the system call calling number by that handler. So we have to make information unavailable to the attacker by using such methods.

**Construct Interrupt Address Table (IAT):**

In this we are creating some binaries to identify the address of system calls in our executables. Then we use this information in our x86 ELF executables as a new section in interrupt address table. It mean that Systems calls are not hidden and these calls can be generate and executed with dynamic code on stack, heap or also decrypted and executes.

**Disguising System Call Instructions:**

The problem with existing executables is that we can easily identify the system call instructions which can vulnerable our systems to scanning attacks. So if we address these system calls instructions by disguising with others it will become harder to identify. The legitimate system calls are existed in instructions address table so it is easy to decide for operating system’s kernel that it is legal call or not. By this the kernel checks address of each call instruction against that IAT or decide if founds trap in IAT which processed as system call, otherwise it will treated as a normal system trap.

**Insertion Of Useless Code:**

Another way to deceive the attacks that we can add some random code which can change its contents but not the instructions semantic eg (mov ax, add $0, push bx, pop cx, mov ax.. etc). Here ax, bx and cx which we are using here actually general purpose registers which we are using here to store some random rough binaries. And if we perform any arithmetic operation in them so it will not live or it will remain dead. Such type of mechanism will also perform in encrypted viruses so they can disguise their decryption engines. We can implement such approach by making unintelligible binary techniques.

**Binary Obfuscation:**

Binary obfuscation is a technique which makes code harder to understandable in real application for others. It does not transform our application code to non-vulnerable or unbreakable. But if we see as remote code attacks are been running nowadays so it will basically take much and long time to bypass such obfuscation and makes more sophisticated our instructions. So it would require no time limits just like trying to reverse engineering implementations on it which means it takes much time to find out results or making your attack successful on it.

**Pocketing:**

Another approach to forbid scanning attacks is to dividation of address spaces of our executables into non sequential segments and they are separated by “Pockets” by invalid addresses. Now ,suppose if that malicious code accessed that invalid address pockets, our operating system generates a trap in kernel which is supposed to be recognized as an intrusion in system.

So there are two practices to perform this discontinuity, one is to separate our code section into many segments and then we assign load address to all executable or successive segments which creates a gap in our segment by ending of every previous segment. Secondly we can insert some code to create gaps into executables and then unmap them at runtime.

*Links:*

<https://pdfs.semanticscholar.org/7f5a/934a2048eef2711d9dc48ece5b20ab8046ca.pdf>

<https://www.usenix.org/legacy/event/sec05/tech/full_papers/linn/linn.pdf>