RESEARCH PAPER

Hareem Shahid(60700)

Sania Karim(60490)

Rameez(59290)

Syed Iftikhar Mehdi(7774)

*Topic:*

**Operating Systems Security And Its System Calls**

*Abstract:*

The papers we are taking here are extracted from a recently completed research or monograph on “Computer security and its problems and solutions” which is intended as a technical point of view in this research within the areas of operating system security. Some evaluations and projections related to operating systems we are also representing here. Our main target is to get the attention of those people who likes to work practically and not conducting such research. We expounding the notion and close observations in terms as logging and monitoring, which gives us the concepts problems and solutions of access control. To maintain the security, operating system relies on isolation. There we are illustrating two most important types of isolation methods.

If we want to make our operating system secure we must design and implement as secure systems. The verification and authentication techniques are main aspect which is proving that the design and implementation of our operating system is examined. When our operating systems are much big and large and contains more features then we should make to secure the elements in kernel.

Also we are here describing that how we can limit the scope of arbitrary code of injection (remote code injection). So we are approaching here two main aspects.

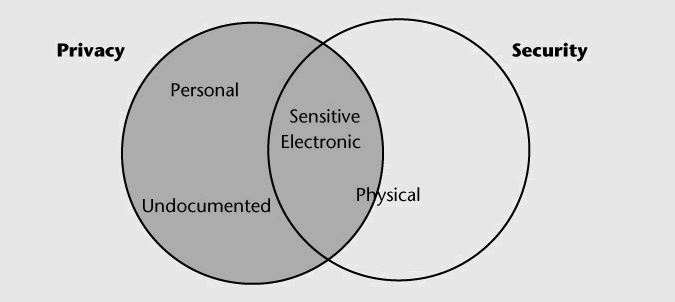
First, the embed semantic information into executables identifying to the legal system calls, and system calls which are from other locations are treated as instructions. The modifications in system calls according to users isn’t possible which adds more security for that information which is present. And the second practice is to encoding system call traps into the Operating system kernel.

**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.

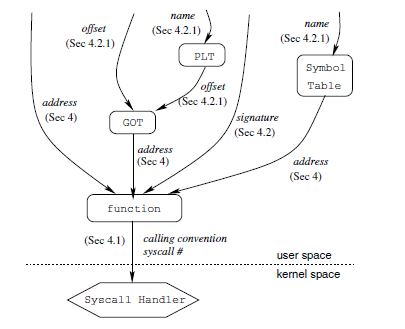
***What And Why System Calls?***

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.

**Conclusion:**

As we see that how much security is important to operating systems to protect our information for the attackers and to get rid off of any kind of harmful activity by outsiders for systems. Without understanding of problems and mechanism of our systems it will be inadequate to implies solutions on it and provide better security to our systems. We always have to make better plan and design to implement better security on our systems. On operating system level many attacks are been running nowadays but in this paper we are intended to put a light code injection based attacks on systems are commonly performing on system nowadays. And also providing two approaches to escape from this type of attack. First use instruction of addresses tables to allowed only those system calls in kernel which is legitimate to our systems by this we can prevent our system to calling that system call which are trying to make harm for systems. Second is to use different techniques to forbid mimicry attacks that are attempting to execute and identify those system calls which are in program code or in libraries. In this paper the experiment and technique which we are using discussing here is effective about related to such problems.

**Refrences:**

[1] E. G. Barrantes, D. H. Ackley, S. Forrest, T. S. Palmer,

D. Stefanovic, and D. D. Zovi. Randomized instruction

set emulation to disrupt binary code injection attacks. In

*Proc. 10th ACM Conference on Computer and Communication*

*Security*, pages 281–289, 2003.

[2] M. Bernaschi, E. Gabrielli, and L. V. Mancini. Operating

system enhancements to prevent the misuse of system

calls. In *Proc. ACM Conference on Computer and Communications*

*Security*, pages 174–183, 2000.

[3] S. Bhatkar, D. C. Du Varney, and R. Sekar. Address obfuscation:

an efficient approach to combat a broad range

of memory error exploits. In *Proc. 12th USENIX Security*

*Symposium*, pages 105–120, 2003.

[4] Bulba and Kil3r. Bypassing StackGuard and Stack-

Shield. *Phrack*, 10(56), May 2000.

[5] M. Chew and D. Song. Mitigating buffer overflows

by operating system randomization. Technical Report

CMU-CS-02-197, Electrical and Computer Engineering

Department, Carnegie Mellon University, Pittsburgh, PA

15213, Dec. 2002.

[l] Hoffman, L.J., Modern Methods for Computer Security

and Privacy, Prentice-Hall, Englewood

Cliffs, NJ, 1977.

[2] Martin, J., Security, Accuracy, and Privacy in

Computer Systems, Prentice-Hall, Englewood

Cliffs, NJ, 1973.

[3] Hsiao, D.K., Kerr, D.S. and Madnick, S.E., *s*puter

Security: Its Problems and Solutions, The

ACM Monograph Series, Academic Press, for 1979.

[4] Lampson, B.W., Needham, R.M., Randall, B. and

Schroeder, M.D., "Protection, Security, Reliability",

Operating Systems Review, Vol. 11,

No. 1, Jan. 1977, pp. 12-14.

[5] Linden, T.A., "Operating System Structures to

Support Security and Reliable Software,"

puting Surveys, Vol. 8, No. 4, Dec. 1976.

*Links:*

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

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