Journal of Computer Virology and Hacking Techniques (2023) 19:409–418
https://doi.org/10.1007/s11416-022-00455-w

ORIGINAL PAPER

A practical approach to learning Linux vulnerabilities

Nikolay Karapetyants[1] · **Dmitry Efanov**[1]

Received: 30 December 2021 / Accepted: 12 October 2022 / Published online: 2 November 2022

Abstract
Operating systems based on the Linux kernel are widespread in the microcomputer, mobile, server, and supercomputer market. They have become an integral part of commercial and government information systems in which confidential information, personal data and trade secrets are stored and processed. Thus, the Linux kernel is a critical object in such systems, because its compromise leads to the compromise of the entire system or an substantial part of it. In most cases, an attacker compromises the kernel by exploiting vulnerabilities, despite the protection mechanisms built into the kernel. And in order to find and eliminate them, an information security professional must have the appropriate skills. Such skills can only be learned through practice. In this paper, we propose a practice-oriented approach to exploring the variety of Linux kernel vulnerabilities in order to acquire the skills to find, analyze and fix them by an information security professional. The proposed approach made it possible to transfer the laboratory practice to online learning during the COVID-19 pandemic.

Keywords linux · vulnerability · virtualization · VirtualBox · Vagrant · COVID-19 · online learning

1 Introduction

Operating systems based on the Linux kernel are very popular in the market of microcomputers, mobile devices, servers and supercomputers. With the growing popularity of cloud computing, the interest of attackers in this area is also growing. About 90% of companies use cloud computing according to IDG research [1]. Most computer computing provides services based on the Linux operating system.

Due to the tremendous fragmentation and the large number of tweaks that affect the maintainability of Linux [2], not all vendors update the software to the latest version, resulting in the risk of confidential information leaking due to the presence of unfixed vulnerabilities in the kernel or loadable kernel modules. Unlike classic bugs, a vulnerability adds functionality to a system, as it allows an adversary to misuse or abuse the system, while a bug is an incomplete or incorrect implementation of a requirement, and thus degrades the functionality of the system [3]. In 2021 about 159 different vulnerabilities were discovered in the original Linux kernel [4].

Most mobile devices, microcontrollers, servers running on the Linux kernel have become an integral part of human life and business that can store and process confidential information, personal data and data related to trade secrets. Linux kernel developers are doing everything possible to quickly fix critical bugs and vulnerabilities. But even professionals of this level are not able to prevent all threats. As part of a social experiment, one of the main developers of the Linux kernel deliberately introduced a critical vulnerability, which was subsequently identified by another developer [5].

The relevance of the work is due to the increase in the number of devices running Linux. The purpose is to obtain professional skills for masters in the field of “Information Security” as part of the vulnerability analysis process of an information system using Linux-based operating system for processing and storing confidential information.

✉ Dmitry Efanov
DVEfanov@mephi.ru

Nikolay Karapetyants
NKarapetyants@mephi.ru

[1] National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Moscow, Russian Federation

2 Background

2.1 Linux kernel

The Linux kernel is one of the most successful open-source software projects in the world [6]. Technically a kernel is a collection of subsystems whose purpose is to organize the interaction between the user and the computer. The Linux kernel configuration is controlled by macros and preprocessor switches, which allow to include or exclude kernel functionalities, specific device drivers and entire subsystems, or to produce a module loadable at running time [7]. Each of the kernel subsystems solves a specific task. The memory management subsystem manages and shares virtual memory between processes. A virtual file system provides an abstract layer that provides the same interface for interacting with a computer's storage device for all file systems [8]. The system call interface allows application programs to access the kernel to perform an operation. Since the resources of a computer are limited, it is necessary to distribute them correctly between processes. The process management subsystem is responsible for this. There is also a subsystem responsible for the operation of the network stack.

The architecture of the Linux kernel is monolithic, which ensures high performance, but its reliability indicator directly depends on the reliability of its components. The monolithic architecture of the kernel does not allow adding support for a particular hardware without a complete recompilation. Therefore, the Linux kernel has an interface that allows you to add functionality without compiling the kernel completely. This is done by loading modules, which are compiled binary files. There are also security modules that complement the security features already present in the kernel. Modules are dynamically connected to a running kernel.

2.2 Kernel vulnerabilities and their exploitation

Even experienced programmers make mistakes. Such errors are not immediately detected at the testing stage, but they can be detected by users, attackers randomly or after a thorough study. Some of these errors may be minor, while others can be exploited, leading to attacks that can severely compromise the kernel quality of service, and allow attackers to gain privileged access [9]. According to Common Weakness Enumeration (CWE) [10] such a simple vulnerability as stack overflow is still common. Such vulnerabilities can be both in the kernel itself and in the loadable module, which in turn creates the need for their timely detection. But this requires specialists with the appropriate competencies.

As part of the research, four of the most common vulnerabilities were selected: the lack of verification of input data before copying to the clipboard (CWE-121); incorrect counting of the number of bytes copied to the clipboard (CWE-190); changing the pointer in the linked list (CWE-123); using an uninitialized variable (CWE-457).

The first vulnerability occurs not only in the Linux kernel, but also in programs and web applications. The meaning of the vulnerability is incorrect processing of input data and thus it becomes possible to read or write to a memory cell that goes beyond the storage area of the data structure, acting as a buffer for storing input data. The data storage structure can be an array with a specified size.

Stack buffer overflow occurs by writing data to a buffer whose size exceeds the buffer size. Thus, you can influence the progress of the program by passing the necessary values to the stack pointer register.

The C and C++ programming languages are very susceptible to buffer overflow attacks, since they do not have built-in protection against overwriting or access control to data in memory, unlike programming languages such as PERL, Java, JavaScript and C#. And the Linux kernel, in turn, is written in the C programming language.

The next vulnerability is related to integer overflow. It occurs when the result of an arithmetic operation is incorrectly processed, in which an integer variable is involved, the value of which is set by the user. Vulnerability ranks 8th in the CWE rating [11].

In most programming languages, integer values are usually allocated strictly with a fixed number of bits in memory. For example, an unsigned 32-bit integer variable can store values in the range from 0 to 4,294,967,295 but signed from −2,147,483,648 to 2,147,483,647. The first bit of a signed integer variable indicates whether the number is positive or negative. It is the use of values outside the above ranges that is called integer overflow. Such vulnerabilities can be used to implement a buffer overflow attack, just like the previous one.

The third vulnerability is one of the types of vulnerabilities associated with buffer overflow and can be used by an attacker only under certain conditions. Using this vulnerability makes it possible to write an arbitrary value to an arbitrary place in memory. The presence of vulnerabilities that cause memory leaks in the program code makes it easier for an attacker to achieve goals. The vulnerability is called "Write-what-where" (Write-What-Where).

The last vulnerability discussed in detail in this paper is related to the use of uninitialized memory (Uninitialized memory read). The C and C++ programming languages do not have mandatory variable initialization. This means that it is not necessary to assign any value to the declared variable. An attacker can use an uninitialized variable to read from memory or execute arbitrary code. Using an uninitialized

variable introduces some randomness because it is impossible to know what value of the variable is in memory.

There are several solutions to eliminate this kind of vulnerabilities. Many development environments analyze the code and may issue a warning about the existence of an uninitiated variable. A routine code check can also reveal the presence of uninitialized data structures. Some language compilers perform automatic initialization of variables during compilation [12]. There are a huge number of ways to implement an attack using the above vulnerabilities. It all depends on the knowledge, skills and creative approach of the attacker.

Now the most promising way of operation is return-oriented programming (ROT) [13]. It is used when exploiting vulnerabilities both in the user space and in the kernel space. The second method that uses overwriting values in the Procedure Linkage Table (PLT) and Global Offsets Table (GOT) is used only in user space and this is due to the differences in the operation of the code in user space and kernel space.

ROT allows to execute arbitrary code without using library functions, reusing pieces of code that end with the command "ret" and they are called gadgets. The "ret" command returns commands to the address where the procedure was called. Having the ability to change the address of the return pointer to the address of an arbitrary function, any code can be executed. To do this, it is just need to know the address of the gadget in the function and change it using the buffer stack overflow vulnerability.

There are two types of binary executable files in Linux: static and dynamic. Static binary files contain all the necessary code for execution, are self-contained and do not depend on any external libraries. The operation of dynamic files is associated with system or other libraries. In fact, the implementation of such a simple printf() function is in the system library. To perform this function, it is enough for the program to know its address, and the operating system will do the rest. It turns out that the execution of some functions in dynamic binary files lies with the operating system. If the library required for the program is not available in the system, the program either will not start or will give an error. The required address of a function in a library is difficult to determine for two reasons: the addresses of the function and libraries change each time the operating system starts due to the mechanism of random allocation of address space. Therefore, a linker was developed that searches for the addresses of the necessary functions and libraries for the executable program. It runs before running executable code and uses the global offset table and the procedure linkage table. The table of global offsets contains the offsets of the list of offsets for each of the libraries necessary for the operation of the program. The PLT table is used to determine the address of the required procedure. The addresses of the required procedure may not be in the GOT table, but can be calculated in case of a call. This situation is called "lazy binding". In this case, the table is writable. For example, the address of the string length determination strlen() function can be overwritten to the address of the system() function to execute system commands. This technique is also used to bypass address space layout randomization (ASLR). You can make tables unreadable and for this it is enough to enable RELocation Read-Only (RELRO) protection before compilation.

ROT allows to execute arbitrary code without using library functions, reusing pieces of code that end with the command "ret" and they are called gadgets. The "ret" command returns commands to the address where the procedure was called. Having the ability to change the address of the return pointer to the address of an arbitrary function, any code can be executed. To do this, it is just need to know the address of the gadget in the function and change it using the buffer stack overflow vulnerability.

The ultimate purpose of the operation is to obtain user privileges and obtain information stored on the computer. In the case of exploitation of the bootable kernel module of the operating system, an attacker can introduce a software bookmark thereby gaining a foothold in the system and have access to it at any time. A feature of a software bookmark from any malicious software is hiding it in the system from the user and from the means of protection. Many program bookmarks are concealed by intercepting system calls and changing the intercepted call parameter. Other program bookmarks are concealed by modifying the code of the core core functions. Using a vulnerability in the bootable kernel, an attacker can introduce a software bookmark and control the system from the kernel space, where it is possible to intercept any user actions, programs and the data they transmit. The only barrier complicating the execution of arbitrary code in system b is the built-in protection against exploitation of the Linux kernel.

2.3 Built-in kernel protection mechanisms

Since there are a lot of modifications of the Linux kernel, the work considered only those methods and means of protection that are present in the main branch of the Linux kernel and are included by default in most distributions running on the basis of the Linux kernel.

Initially, the user space and the kernel space are isolated from each other and interact with each other using the system call interface. Only with superuser rights, until recently, it was possible to execute arbitrary code in user space. Starting with the version of the Linux kernel 5.4, the "lockdown" mode was introduced. This mode restricts access to some

elements in the operating system to interact with the kernel. You can activate the mode by simply specifying the options in the configuration file before building the kernel "SECURITY_LOCKDOWN_LSM" and "LOCK_DOWN_KERNEL_FORCE". It only limits the regular access to the kernel and does not protect against exploiting vulnerabilities in the kernel and modules [14].

In order to avoid the execution of arbitrary code when the stack buffer overflows, a method was invented in which the execution of arbitrary code is impossible. The method is quite simple. It is only necessary to add some values to the execution stack that have the name "canary stack". Before performing a certain operation on the stack, it is necessary to check the value of the "canary stack". If they have changed, it means an unauthorized buffer overflow has occurred and then the program will stop executing. You can bypass this protection only by knowing the values of the "canary stack". But it can be calculated using a full iteration of the values. After receiving the value of the "canary stack", you need to embed its value in the buffer, the overflow of which overwrites the program stack.

In addition to the "canary stack", the program stack can only be accessed in read and write mode, but not execution. There are two workarounds. The first is related to finding an area in memory in which you can execute a command. The second is to use return-oriented programming. This method involves collecting arbitrary code from existing instructions in the program. Because they are executable, which means they can be used to assemble any code to perform certain operations.

The methods of protection "Lockdown", "canary stack" do not protect absolutely from the exploitation of the Linux kernel, but can only become an obstacle in obtaining the privileges of the kernel space.

To complicate, but not to prevent, the use of binary vulnerabilities in the Linux operating system, ASLR method was introduced. The allocation of addresses in memory is carried out randomly. This complicates the process of exploiting those vulnerabilities that require knowledge about the address space where the program is running. It is possible to bypass ASLR with the help of other vulnerabilities, a complete search of addresses or using the features of the processor. The very close location of data and executable code in memory increases the likelihood of successful operation. ASLR implementations vary greatly and develop in different operating systems. In the kernel space, ASLR is also used and has the name kernel ASLR (KASLR). KASLR can be circumvented using the following factors: low entropy in the random distribution of addresses; address leakage to determine a random offset and identify other addresses; indirect information to determine offsets [15].

After the "Meltdown" vulnerability appeared in Intel processors due to its architectural features and the inability to fix it by updating the microcode, it was decided to use kernel page-table isolation (KPTI) for protection. This protection made it possible to deprive attackers of the opportunity to exploit the "Meltdown" vulnerability, but led to a decrease in performance from 5–30%. For many users, this has become a dilemma. Therefore, some disable this protection to the detriment of security, but for the sake of performance.

KPTI is used to protect against a type of attack aimed at the shared address space of the kernel and the user [16]. For each program in the user space, an independent set of pages is created at startup, containing the minimum necessary set of information. When switching to privileged mode (kernel mode), a copy of the pages is created, but in full form for the kernel. This approach guarantees the impossibility of implementing attacks using side information.This protection can be circumvented with the help of return-oriented programming.

3 Educational-methodical complex for the study of kernel vulnerabilities

The practice-oriented approach involves obtaining skills and knowledge by performing special prepared practical tasks. In the case of studying the vulnerabilities of kernel modules, this approach allows you to clearly demonstrate the result of their operation and also ways to fix them. Within the framework of university training, laboratory classes are supposed to be held, where students gain practical skills.

Ensuring the continuity of the functioning of information systems and automated systems primarily depends on the training of a professional. Every year, the demand for information security professionals is growing faster than the supply [17]. The situation worsened during the COVID-19 pandemic, when universities had to switch to online learning in the shortest possible time [18, 19]. If practical classes and lectures were adapted to the new format of training, then most of the laboratory work had to be canceled and postponed to another semester. Thus, there was a need to create an educational and methodical complex that would include the possibility of laboratory work within the framework of online learning.

The developed educational-methodical complex consists of two interdependent parts: a laboratory stand and a methodological manual. The laboratory stand is a set of software tools with the help of which practical tasks are performed in accordance with the instructions contained in the methodological manual. Next, each of the parts of the educational and methodological complex will be considered.

3.1 Development stage

To perform most laboratory work, a serviceable and correctly configured laboratory stand is required. As a rule, students are not engaged in setting up a laboratory stand if it includes complexly configurable specialized software and hardware. The situation is different if the laboratory stand is a personal computer with open source software installed. In this case, it is possible to automate the process of installing and configuring the software necessary to perform practical work. It is also important that the laboratory stand works on most modern personal computers or laptops. This will allow students to perform laboratory work outside of the laboratory classroom. Due to the high prevalence of the Windows 10 operating system it is also important to have compatibility with it.

Since practical tasks involve direct interaction with the Linux kernel, it is necessary to ensure that the stand can be quickly restored to its original state. In the event of a stand failure, the student will have the opportunity to return the stand to its original state.

To ensure a simple installation of a laboratory stand, you can use a prepared backup image of the entire system or a virtual machine image.

The first method requires pre-installing and configuring the necessary software and working environment. After that, you need to make a full backup of the computer and restore it on the other workstations. This method allows you to return the computer to its original state. Since the entire operating system is backed up along with user files, the process of creating a backup and restoring takes a long time. It is also impossible to predict whether the created backup will be restored correctly on another computer. These disadvantages are not inherent in virtual machines.

Virtualization is an abstract representation of hardware resources, in other words, imitation of hardware by software methods. But the full implementation of hardware by software methods usually leads to a small performance of the virtual machine. Therefore, most modern processors have hardware support for virtualization, which significantly increases the performance of the virtual machine.

Paravirtualization is a virtualization technique in which the cores of guest operating systems are slightly modified for their subsequent launch in a virtual environment. This method provides performance comparable to that of a non-virtualized system.

Containerization is another way of virtualization. For the previous two to work, hardware virtualization support was necessary, and for this only one condition is necessary – that the guest operating system uses the same kernel as the main one. The disadvantage of this method is its main advantage – one core for the operation of guest and main operating systems. In the event of a kernel failure in the guest operating system, this will directly affect the functioning of the main operating system.

An image of a virtual system has one significant advantage over a backup copy of the entire computer: it works on almost any computer, but only if it has the appropriate software and hardware. With this design of the laboratory stand, the student will be able to perform laboratory work independently if he has a computer that meets the technical requirements.

There are a lot of virtualization tools [20, 21]. The following were considered in the work due to their greatest prevalence: Kernel-based Virtual Machine (KVM), Hyper-V, VMware, VirtualBox.

KVM is a virtualization tool for operating systems based on the Linux kernel. This hypervisor supports Intel-VT and AMD-V hardware extensions. This tool includes the main module of the Linux kernel kmv. ko, and a module for a specific processor - kvm-amd.ko or kvm-intel.ko. This is the most suitable virtualization tool for virtualization of operating systems operating on the basis of the Linux kernel. The main contribution to improving performance is made by Linux kernel modules. Thanks to them, the kernel of the guest operating system accesses directly to the hardware of the main operating system using the KVM kernel modules. This hypervisor works only on Linux-based operating systems. In the Windows 10 operating system, this hypervisor does not exist in its pure form, since bootable Linux kernel modules are required for its operation.

Hyper-V is a hypervisor created by Microsoft. First of all, this tool is aimed at the corporate market. And this tool is famous primarily for its stability and high performance. This hypervisor provides high performance when virtualizing Windows operating systems compared to other hypervisors. This tool is available only in Windows 10 Pro versions, Corporate or for educational institutions [22]. In this regard, Hyper-V cannot be used as part of a laboratory stand, because not all students can have the Windows 10 operating system of the desired version installed on their computer.

VMware Workstation is a commercial virtualization product. There is also a free version with limited features. VMware Workstation is primarily good because it has advanced 3D graphics support, unlike VirtualBox. But the free version of VMware does not have the ability to create snapshots of a virtual machine. This function is very important for performing laboratory work, because snapshots of the virtual machine's state will help restore the system in case of its breakdown.

VirtualBox is an open source virtualization tool. VirtualBox has many applications and works on absolutely all platforms: Linux, Windows, Solaris, macOS, FreeBSD. VirtualBox supports a larger number of operating systems

compared to VMware. VirtualBox for virtualization uses hardware virtualization of Intel and AMD processors. VirtualBox also supports three virtual disk formats: VDI, VMDK, VHD. This virtualization tool is ideal for performing laboratory work because it is distributed absolutely free of charge and works on almost all existing platforms.

Virtualbox allows you to export a configured virtual machine as an image. The size of the exported image usually exceeds one gigabyte, which creates difficulties when distributing it. But there is free open source software for creating and configuring a Vagrant virtual environment. Vagrant acts as a virtual machine manager. It supports popular virtual tools, including VirtualBox. To create a virtual machine, a VagrantFile configuration file is required. This file describes the characteristics and how a virtual machine will function. In fact, Vagrant is similar in functionality to Docker, but it does not manage containers, but virtual machines. Vagrant also allows to create and configure servers in Google Cloud, Amazon AWS and Yandex.Cloud. Therefore, the use of Vagrant will allow to deploy a laboratory stand on absolutely any server and personal computer.

Vagrant uses its own virtual machine image format and has the *.box extension. In Vagrant, it is possible to upload an image of a created virtual machine to the Vagrant Cloud. This means that by creating, configuring and uploading a virtual machine image to the cloud, any Internet user will have the opportunity to download the image and perform laboratory work according to the methodological manual of the laboratory workshop.

To gain practical skills in finding and fixing vulnerabilities in the Linux kernel, it was necessary to create conditions close to reality. To achieve this, it is necessary that the kernel contains vulnerabilities. It doesn't make sense to modify the Linux kernel itself. It is enough to implement only one downloadable kernel module containing the necessary functionality to perform practical tasks.

The module makes it possible to add new functions to the kernel without having to recompile it completely. The module can implement the functionality of some virtual device. Interaction with physical devices is carried out using device files. And this is primarily due to the main concept of Linux, which says "everything is a file". All device files are contained in the "dev" directory. A device file, for example, can be used to interact with a hard disk or with another input/output device. But besides interacting with physical devices, device files can implement abstract functions. For example, the device file "/dev/random" implements the function of a random number generator, although there is no physical random number generator.

The main purpose of the developed module is to exploit vulnerabilities in order to gain skills in detecting and preventing them. The module implements a symbolic device with which interaction is carried out using the "ioctl" system call. The function name is an abbreviation of "input output control". The ioctl function accepts the following parameters: the file device descriptor, the ioctl command number, and a parameter that has the unsigned long type.

The module includes the implementation of four vulnerabilities: the lack of verification of input data before copying to the clipboard (CWE-121); changing the pointer in the linked list (CWE-123); incorrect counting of the number of bytes copied to the clipboard (CWE-190); the use of an uninitialized variable (CWE-457).

The result of the work done is a public repository [23], which contains configuration files, as well as the source code of the module necessary for performing practical tasks.

The main distribution in which laboratory work is performed is Debian, due to its high prevalence and high performance. The personal computer on which the laboratory stand is planned to be installed must meet the following technical requirements:

- 64-bit processor with 4 physical cores with support for multithreading and hardware virtualization;
- the size of the installed RAM should not be less than 16 gigabytes;
- the amount of free space on the system disk should not be less than 50 gigabytes.

The processor must support 64-bit instructions. This is primarily due to the fact that a processor with such instructions has very high performance compared to processors that support only 32-bit instructions. Currently, only Intel and AMD companies produce such processors. Another role is played by the number of physical processor cores. The more of them, the higher the performance. Based on experience, for comfortable operation, the processor must have at least four physical cores. The main contribution to performance is made by physical cores.

Processor support for hardware virtualization plays an important role when using virtualization programs. Software virtualization provides low performance and therefore the processor included in the laboratory stand must support hardware virtualization.

The minimum size of RAM was determined from personal experience and is primarily associated with the tendency to increase the minimum size of RAM for comfortable work. Also, the use of virtualization tools as part of the laboratory stand, which requires a lot of resources to run a virtual machine, plays a very important role.

The size of the permanent memory is also determined based on personal experience. A VM typically takes up about 8 gigabytes, not including snapshots of states.

The training workshop is an educational and methodical manual that includes a description and step-by-step actions to obtain theoretical and practical skills in investigating vulnerabilities of Linux kernel modules for their subsequent detection and correction.

The developed workshop is of an introductory nature. The purpose of this kind of laboratory workshops is: to illustrate the basic laws of the studied field of science, to familiarize students with the technique of conducting an experiment and the principles of software operation.

To ensure accessibility to the materials of the training workshop, it was decided to use a third-party service. The most suitable service for this is Gitbook – a tool for creating documentation using the distributed version control system Git and the simplified markup language Markdown. The main advantages of the Markdown markup language are versatility and simplicity. Documents written using Markdown syntax are plain text files that can be opened in any editor or platform. And the simplicity of the language allows you to master it painlessly, unlike the same Latex. Documents written in Markdown can be easily exported to any formats, such as pdf, doc, or txt.

Git's distributed version control technology allows multiple users to edit a document at the same time. This is convenient when developing documentation for the relevant software. It was for the joint writing of documentation that the Gitbook service was developed. There are quite a lot of alternatives to Gitbook, and there will be no difficulties when switching to a third-party service thanks to the universal Markdown markup language. GitBook also has the ability to synchronize with the Github repository.

Providing the student with fully prepared materials for laboratory work is a bad practice in most cases. This is primarily due to the fact that as a result, the student will not have the skills to independently install and configure software for laboratory work. In the process of developing a methodological manual for laboratory work, it was decided to divide each laboratory work into several small modules. The module is a small task that needs to be completed. But before the decomposition of laboratory work is carried out, it is necessary to perform the following tasks:

1. To disassemble the vulnerability, describe it with text and drawing;
2. Implement a kernel module containing this vulnerability;
   a. Test and identify the conditions for triggering the vulnerability;
   b. Document the process of building, testing, and the source code of the kernel
   c. Record a video demonstrating how the vulnerability worked;
   d. Analyze: the conditions of operation, the consequences of work, the effectiveness of counteraction techniques.
3. Reflect the results of the work done in the report.

Performing the above tasks for each of the vulnerabilities will allow you to reveal in more detail the principles of their work and ways to neutralize them.

According to the terms of reference, the workshop should include four main laboratory work on each of the four vulnerabilities: CWE-190, CWE-121, CWE-457, CWE-123.

3.2 Laboratory stand and workshop

The materials needed to complete the labs are in the public pages of GitBook[24] and GitHub[19]. The laboratory workshop consists of 11 sections:

- Introduction;
- Conventions and abbreviations;
- Glossary;
- Preparation of a laboratory stand;
- Lab #0;
- Lab #1;
- Lab #2;
- Lab #3;
- Lab #4;
- Lab #5;
- Bibliography.

The introduction briefly outlines the goals and objectives to be solved in the laboratory workshop, as well as recommendations for performing laboratory work.

The second section describes the terms used in the laboratory practice with the appropriate designations and abbreviations.

The third section includes definitions of terms that are mentioned in the laboratory practice.

The section dedicated to the laboratory stand contains two subsections: technical requirements and installation and configuration guide. The technical requirements section indicates the minimum requirements for a computer on which the lab will be deployed and labs will be performed. The installation and configuration section of the guide includes two scenarios for deploying the lab bench: for the Windows 10 and Debian operating systems. Installing and configuring the lab bench consists of three steps: enabling hardware virtualization, installing VirtualBox, and installing Vagrant. After the computer is set up and the necessary software is installed, you can begin to perform laboratory work. Instructions for installing the stand for the Mac OS

operating system were not included due to the lack of appropriate hardware and software.

Next comes a section that includes a description and methodology for performing laboratory work No. 0. The structure of this and subsequent laboratory work is the same and consists of three subsections: background information, recommended reading, and the task itself. The first subsection contains the theoretical information necessary to perform laboratory work. The second section includes a list of recommended literature for self-study by the student in order to improve the quality of the task. The goal of lab #0 is to get familiar with the lab bench software. In fact, everything comes down to studying the functionality of the Vagrant program, the skills of working with which are necessary to perform subsequent laboratory work.

Lab #1 is about learning how to build the Linux kernel and loadable modules. Kernel build skills will allow students to enable and disable the kernel's built-in protection mechanisms against unauthorized execution of arbitrary code. Loadable Kernel Module Building Skills allow the student to build and load a specialized module to explore Linux kernel vulnerabilities.

All subsequent laboratory works are devoted to the study of four Linux kernel vulnerabilities: CWE-121(Stack-based Buffer Overflow), CWE-190(Integer Overflow), CWE-457(Use of Uninitialized Variable), CWE-123(Write-what-where Condition). Before performing these laboratory works, students are invited to familiarize themselves with the theoretical materials, as well as with a list of recommended literature that can help in completing the laboratory task. To complete a lab task, you need to start the working environment, load a specialized module, and then proceed to the lab task. Interaction with the module is carried out using the IOCTL system call, where identifiers from 0 to 3 correspond to vulnerabilities: CWE-121, CWE-190, CWE-457, CWE-123. Students can call using both C and Python using the hevd.py library, which is contained in the home folder of the virtual environment. The home folder also contains all the necessary files to complete the labs. The operation of a specialized module is monitored using the dmesg command. The output of this command contains detailed information about the results of executing module functions.

In the second laboratory work, the vulnerability of CWE-121 will be considered, which is associated with incorrect processing of input data and their writing to a buffer of limited size. In this laboratory work, students will get acquainted with addressing in Linux virtual memory, the source code of the kernel module function, demonstrating this kernel vulnerability. To facilitate the passage of this laboratory work, students will be asked to turn off the protective mechanisms of the Linux kernel on their own. This will allow students to understand the principles of these security features, as well as their importance in the Linux kernel. Also, the student will be provided with the source code of the same module function, but with some changes that do not allow taking advantage of the vulnerability. Thus, upon completion of this laboratory work, the student receives knowledge about the principles of functioning of this vulnerability, its correction, as well as about the built-in means of protecting unauthorized code execution in the Linux kernel.

The third laboratory work related to the study of vulnerability number CWE-190. The essence of the vulnerability lies in the incorrect determination of the size of the input data and the buffer. Under certain circumstances, a situation may arise in which the size of the input data will be larger than the buffer to which this data is copied. The vulnerability is somewhat similar to the previous one, but the principle of its operation is somewhat different. In this case, the student is invited to use a specialized library to interact with the module function demonstrating this vulnerability. The main task is to select such input data values, at which a message will appear in the kernel debugging log that such a vulnerability is present in the module.

The laboratory #4 work is related to a vulnerability under the code number CWE-457. Its main essence lies in the absence of variable initialization. This means that the variable was created, but without assigning it any value. In this case, the variable can contain absolutely any information, including confidential information contained in the computer's memory. A dangerous situation occurs if such an uninitialized variable is dereferenced. In this laboratory work, it is necessary to select such input data, at which a message about the successful detection of a vulnerability appears in the kernel debugging log. To perform this laboratory work, it is necessary for the student to study the source code of the module before starting to perform laboratory work.

The last laboratory work is related to the vulnerability of CWE-123. The meaning of this vulnerability is the incorrect use of pointers when filling the buffer with input data. Under certain circumstances, an attacker has the opportunity to directly write any data to any place in memory. This vulnerability turned out to be too sensitive and with inappropriate input data, a "Kernel Panic" state occurs in the Linux kernel. Thus, the student needs to pick up such data and prevent the state of "Kernel Panic".

As a result of successful testing, this complex was introduced into the educational process as part of the training of bachelors in the discipline "Security of operating systems".

4 Conclusion

The developed educational and methodological complex allows a comprehensive approach to the study of vulnerabilities of the Linux kernel. This is due to the fact that the student is primarily given the opportunity to independently study the composition of the laboratory stand during its setup and installation. The possibility of automatic installation and configuration of the laboratory stand allows you to immediately start performing laboratory work. The placement of all materials on the Internet made it possible to perform practical tasks within the framework of distance learning. The proof of the work performed independently by the student can be a report on the results of the work done and a video recording of the execution process.

The use of VirtualBox and Vagrant virtualization tools allowed not only to automate the process of installing and configuring the laboratory stand, but also to ensure the security of the process of exploiting kernel vulnerabilities.

The approbation of the educational and methodological complex allowed us to identify the errors contained in the methodological manual. Also, to better assimilate the materials of the educational and methodological complex, two laboratory works were added: the study of the basic functionality of Vagrant, the study of the process of assembling and installing the Linux kernel and the downloadable module.

The problems associated with the training of highly qualified personnel cannot be solved simultaneously but require a thorough integrated approach. The most important thing is to acquire practical skills that can be acquired only when solving real problems. The developed educational and methodological complex is a basic version of something more than laboratory work. An example is the cyber polygon, which provides a range of real-world cybersecurity tasks.

Data availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Statements and Declarations The authors have no relevant financial or non-financial interests to disclose.

References

1. 2020 Cloud Computing Study:. IDG. (2021). https://www.idg.com/tools-for-marketers/2020-cloud-computing-study Accessed 30 December 2021
2. Liguo Yu, S.R., Schach, K., Chen, Gillian, Z., Heller: Jeff Offutt: Maintainability of the kernels of open-source operating systems: A comparison of Linux with FreeBSD, NetBSD, and OpenBSD. J. Syst. Softw. (2006). https://doi.org/10.1016/j.jss.2005.08.014
3. Gerardo Canfora, A.D., Sorbo, S., Forootani, A., Pirozzi, Corrado Aaron Visaggio: Investigating the vulnerability fixing process in OSS projects: Peculiarities and challenges. Computers & Security. (2020). https://doi.org/10.1016/j.cose.2020.102067
4. Linux kernel vulnerabilities. CVE. (2021). https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33 Accessed 30 December 2021
5. Greg Kroah-Hartman: bans University of Minnesota from Linux development for deliberately buggy patches. ZDNet. (2021). https://www.zdnet.com/article/greg-kroah-hartman-bans-university-of-minnesota-from-linux-development-for-deliberately-buggy-patches Accessed 30 December 2021
6. Ayelet Israeli, D.G., Feitelson: The Linux kernel as a case study in software evolution. J. Syst. Softw. (2010). https://doi.org/10.1016/j.jss.2009.09.042
7. Antoniol, G., Villano, U., Merlo, E., Di Penta, M., Analyzing cloning evolution in the Linux kernel, Information and Software Technology (2002). https://doi.org/10.1016/S0950-5849(02)00123-4
8. Lanyue Lu, A.C., Arpaci-Dusseau, R.H., Arpaci-Dusseau, Lu, S.: A Study of Linux File System Evolution. ACM Trans. Storage. (2014). https://doi.org/10.1145/2560012
9. Shameli-Sendi, A.: Understanding Linux kernel vulnerabilities. J. Comput. Virol. Hacking Techniques. (2021). https://doi.org/10.1007/s11416-021-00379-x
10. The CWE Top 25:. CWE. (2021). https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html Accessed 30 December 2021
11. The CWE Top 25:. CWE. (2021). https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html Accessed 30 December 2021
12. Solving Uninitialized Stack Memory on Windows. MSRC. (2021). https://msrc-blog.microsoft.com/2020/05/13/solving-uninitialized-stack-memory-on-windows Accessed 30 December 2021
13. Shrivastava, R.K., Hota, C.: Profile-guided code identification and hardening using return oriented programming. J. Inform. Secur. Appl. (2019). https://doi.org/10.1016/j.jisa.2019.102364
14. Kernel lockdown patches for v5.4. (2021). https://lkml.org/lkml/2019/9/28/52 Accessed 30 December 2021
15. Manish Bhatt, I., Ahmed: Leveraging relocations in ELF-binaries for Linux kernel version identification. Digital Investigation. (2018). https://doi.org/10.1016/j.diin.2018.04.022
16. KPTI — the new kernel feature to mitigate: “meltdown”. Fedora magazine. (2021). https://fedoramagazine.org/kpti-new-kernel-feature-mitigate-meltdown Accessed 30 December 2021
17. Borka Jerman Blažič: The cybersecurity labour shortage in Europe: Moving to a new concept for education and training. Technology in Society:. (2021). https://doi.org/10.1016/j.techsoc.2021.101769
18. Yogesh, K., Dwivedi, D.L., Hughes, C., Coombs, I., Constantiou, Y., Duan, J.S., Edwards, B., Gupta, B., Lal, S., Misra, P., Prashant, R., Raman, N.P., Rana, S.K., Sharma: Nitin Upadhyay: Impact of COVID-19 pandemic on information management research and practice: Transforming education, work and life. Int. J. Inf. Manag. (2020). https://doi.org/10.1016/j.ijinfomgt.2020.102211
19. Santiago Iglesias-Pradas: Ángel Hernández-García, Julián Chaparro-Peláez, José Luis Prieto: Emergency remote teaching and students’ academic performance in higher education during the COVID-19 pandemic: A case study. Comput. Hum. Behav. (2021). https://doi.org/10.1016/j.chb.2021.106713
20. Asvija, B., Eswari, R., Bijoy, M.B.: Security in hardware assisted virtualization for cloud computing — State of the art issues and challenges. Computer Networks. (2019). https://doi.org/10.1016/j.comnet.2019.01.013
21. Fady, A.M., Ibrahim, E.E., Hemayed: Trusted Cloud Computing Architectures for infrastructure as a service: Survey and systematic literature review. Computers & Security. (2019). https://doi.org/10.1016/j.cose.2018.12.014

22. Installing Hyper-V on Windows 10. (2021). https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v Accessed 30 December 2021
23. Linux HEVD tutorial. GitHub. (2021). https://github.com/koly-andaemon/linux-hevd Accessed 30 December 2021
24. Linux HEVD tutorial. GitBook. (2021). https://nskarapetyants.gitbook.io/linux-hevd-tutorial Accessed 30 December 2021