

Implementation of Server Provisioning and Hardening Automation using Terraform and Ansible with NIST SP 800 – 123

Nelmiawati¹, Gunawan², Hamdani Arif³, Antoni Haikal⁴, and Septafiansyah Dwi Putra⁵

^{1,2,3,4}Rekayasa Keamanan Siber, Politeknik Negeri Batam, Batam, Indonesia

⁵Teknologi Rekayasa Internet, Politeknik Negeri Lampung, Lampung, Indonesia

*Corresponding author: nelmia.wati@gmail.com

ABSTRACT

Servers play an important role as the center of data processing, applications, and services used by companies. High dependence as well as reliability on servers makes them primary targets for cyberattacks such as malware, SSH attacks, and web-based attacks. In addition, manual server provisioning is time-consuming and prone to human error, potentially leading to security misconfigurations. This study proposes a solution through the automation of server provisioning and hardening using Terraform and Ansible based on the NIST SP 800-123 server security standard. This solution was designed to speed up server provisioning processes and ensure consistent and effective implementation of security configurations. The results demonstrate a halving in provisioning and hardening time when compared to manual processes. There were 64.29%, or 18 out of 28, checklists that were met. Vulnerability assessment using Lynis shows 71 of the hardening index, while OpenVAS found 1 low level after hardening. Additionally, SSH penetration testing showed a 100% prevention rate against unauthorized access. By leveraging digital automation to enhance IT infrastructure security, this solution contributes to building a more secure and reliable digital ecosystem.

Keywords: Server Provisioning, Server Hardening Automation, NIST SP 800-123, Terraform, Ansible

1. INTRODUCTION

Servers play a vital role in technology companies as central hubs for data processing, applications, and services. Due to this critical function, servers are common targets of cyberattacks, including malware, SSH attacks, and web-based threats. (Putra et al., 2022). According to the 2023 report by Indonesia's National Cyber and Crypto Agency (BSSN), over 403 million anomalous traffic events were recorded during the year (BSSN, 2024).

Besides external threats, misconfigurations caused by human error often expose servers to attacks. For example, in 2023, Indonesia was targeted by the LockBit ransomware, which encrypted internal data and exfiltrated sensitive company information (Anasrullah, 2024). Such incidents often arise from human errors or inherent weaknesses in security systems. Misconfigurations server settings can result in vulnerabilities, data lost, and financial damage (Soms et al., 2022).

One effective way to reduce these risks is through security hardening, which improves system and network resilience by minimizing vulnerabilities (Herdianto et al., 2021; Wibowo et al., 2023). Security hardening improves system and network resilience against threats (Irfandi et al., 2022). This process can be guided by standards such as NIST SP 800-123, which provides best practices for secure server configurations (Scarfone et al., 2008; Zein et

al., 2023). Moreover, manual server provisioning is time-consuming and prone to human error, especially during repetitive configurations (Khumaidi, 2021).

To address both provisioning inefficiencies and security risks, this study proposes an automated approach using Terraform and Ansible. This integrated automation aims to speed up server deployment while ensuring standardized and secure configurations based on NIST SP 800-123.

2. LITERATURE REVIEW

2.1.Theoretical Framework

Automation involves applying technology to reduce human involvement by defining decision criteria, relationships between subprocesses, and necessary actions, which are then executed by systems or machines (Evianti et al., 2021). In this study, automation supports server provisioning and hardening to minimize vulnerabilities from human error.

Infrastructure automation, or Infrastructure as Code (IaC), is an approach used to provision and manage computing environments through explicit definitions of the desired environment in the form of code (Hariyadi & Marzuki, 2020). Terraform is a widely used IaC tool for defining and deploying various infrastructure resources (Evianti et al., 2021).

To complement infrastructure provisioning with Terraform, hardening configuration can be automated using tools like Ansible. Ansible can automate configuration tasks. It is an open-source tool used in application deployment, configuration management, and system provisioning for Unix and Windows (Kurniawan & Saputra, 2024). In its implementation, Ansible uses YAML-based playbooks—human-readable files that define configuration steps (Alfiandi et al., 2020; Peddireddy, 2024; Pratama & Hariyadi, 2021).

The National Institute of Standards and Technology (NIST) provides guidelines for information security through its Special Publication 800 series (Anelia et al., 2023). NIST SP 800-123 specifically offers best practices for securing servers to help organizations prevent, detect, and respond to cyber threats (Irfandi et al., 2022).

2.2.Previous Research

Previous studies have extensively explored automation in IT infrastructure management. The study by (Hariyadi & Marzuki, 2020) demonstrated that Ansible can accelerate Virtual Private Server (VPS) management; however, it did not focus on comprehensive security hardening. Similarly, (Khumaidi, 2021) highlighted the efficiency of Ansible in automating server and user account management, but the research lacked depth in structured security hardening practices. In the context of infrastructure provisioning automation, (Pramadika & Chandra, 2023) showed the efficiency of Terraform in provisioning Google Kubernetes Engine (GKE) clusters, yet their study did not address post-provisioning operating system hardening. Meanwhile, (Putra et al., 2022) implemented Infrastructure as Code (IaC) using Ansible for network security automation, including IDS and honeypot deployment, but their focus was on network security components rather than holistic operating system hardening. Regarding the application of security hardening, (Zein et al., 2023) identified and applied hardening procedures on virtual servers based on the NIST SP 800-123 standard. However,

the approach in their study was primarily manual and recommendation-based, lacking full automation.

This study differs from previous research by combining automated server provisioning using Terraform with security hardening using Ansible, both referring to the NIST SP 800-123 server security standard. This approach brings both processes into a single automated workflow. The main goal is not only to speed up server deployment but also to ensure that the servers are securely and consistently configured. It helps reduce human error and improve overall system security. In addition, this study specifically measures how well the automated hardening process meets the NIST SP 800-123 checklist, which is often not addressed in earlier studies that focused only on provisioning, manual hardening, or other network security components.

3. METHODS

This research uses an experimental method with stages that begin with literature review, infrastructure design, implementation, testing and evaluation, as well as result analysis. These stages are illustrated in Figure 1 below:

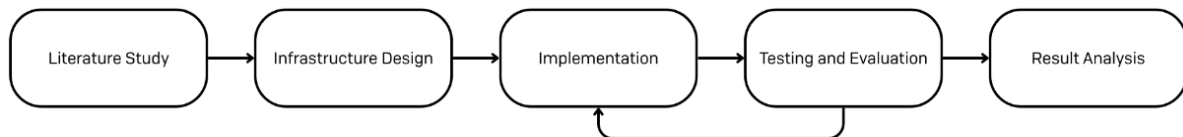


Figure 1. Research Methodology

This study began with a literature review, examining previous research, technical documents, and relevant best practice guidelines to support the implementation. Next, the infrastructure was designed by integrating technologies for automated server provisioning and security hardening. The system was implemented using Terraform and Ansible, comparing two conditions: before and after hardening. The tested variables included provisioning time, Lynis audit scores, number of vulnerabilities identified by OpenVAS, and penetration testing results. Testing and evaluation involved Vulnerability Assessment (VA) and Penetration Testing (PT), where VA identified system vulnerabilities and PT simulated attacks to detect configuration weaknesses (Vegesna, 2022). Finally, the test results were analyzed to evaluate the effectiveness of the automated hardening process in enhancing server security.

4. RESULT AND DISCUSSION

4.1. System Design

This study utilizes virtual machines on the Proxmox platform, with Terraform automating server provisioning based on the Infrastructure as Code (IaC) approach, as shown in Figure 2. Once the servers are created, Ansible automatically configures and applies hardening. The workflow of this automation process is illustrated in Figure 3. To evaluate the system's security, internal audits are conducted using Lynis, while external testing is performed using Kali Linux with OpenVAS scans and simulated SSH attacks.

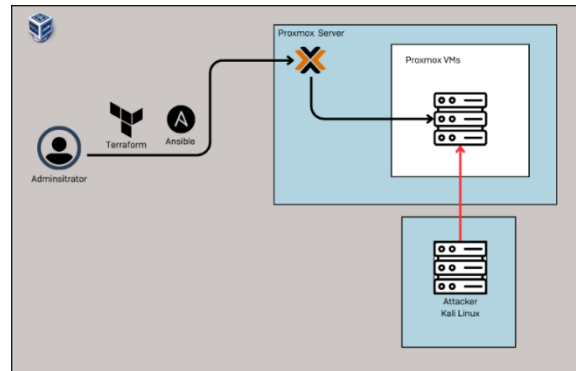


Figure 2. System Design

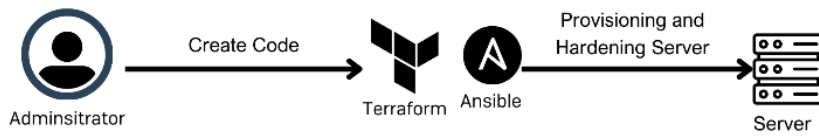


Figure 3. System Flow

This research was conducted in a local environment, with the specifications of the devices used detailed in Table 1.

Table 1. Local environment specifications

Type	Specifications
CPU on Local Environment	12 CPUs Intel Core I7-10750H
Memory on Local Environment	4 GB RAM
CPU on Proxmox	4 CPUs Intel Core I7-10750H
Memory on Proxmox	4 GB RAM
Technology	Version
VirtualBox	7.0.20
Proxmox	1.10.50
Terraform	2.15.13
Ansible	7.4-19
Ubuntu Server	24.4.02
Technology	Version

4.2.Implementation Results

4.2.1. Terraform Code

In the Terraform code, several configurations are made, such as configuring the connection between Terraform and Proxmox via the API, as shown in Figure 4.

```

1 proxmox_api_url = "https://127.0.0.1:8006/api2/json"
2 proxmox_api_token_id = "root@pam!terraform"
3 proxmox_api_token_secret = "4934627c-f139-464f-8851-436851268709"
4
5 ci_user = "gunawan"
6 ci_password = "Qwerty"
7 ci_ssh_public_key = "~/ssh/id_rsa.pub"
8 ci_ssh_private_key = "~/ssh/id_rsa"

```

Figure 4. Proxmox API code snippet

Additionally, other configurations include declaring the specifications of the server being created, such as machine ID, name, CPU cores, memory, and other settings, as shown in Figure 5.

```

1 resource "proxmox_vm_qemu" "TugasAkhir" {
2   vmid = 100
3   name = "hardan"
4   target_node = "gunawan"
5   clone = "ubuntu"
6   full_clone = true
7
8   os_type = "cloud-init"
9   cloudinit_cdrom_storage = "local-lvm"
10
11   ciuser = var.ci_user
12   cipassword = var.ci_password
13
14   sshkeys = file(var.ci_ssh_public_key)
15   cores = 1
16   memory = 512
17   agent = 1
18
19   bootdisk = "scsi0"
20   scsihw = "virtio-scsi-pci"
21   ipconfig0 = "ip=10.0.2.16/24,gw=10.0.2.2"
22   ipconfig1 = "ip=192.168.2.4/24, gw=192.168.2.3"
23
24   disk {
25     size = "10G"
26     type = "scsi"
27     storage = "local-lvm"
28   }
29 }

```

Figure 5. Virtual machine specification

4.2.2. Ansible Code

In this section, Ansible is executed by Terraform through the command line as shown in Figure 6.

```

1 resource "null_resource" "ansible" {
2   provisioner "local-exec" {
3     command = "sleep 60; ansible-playbook -i inventory.ini site.yml -u ${var.ci_user}"
4   }
5   depends_on = [proxmox_vm_qemu.TugasAkhir]
6 }

```

Figure 6. Terraform code snippet to execute Ansible

Next, Ansible will work by applying the configurations defined in the YAML file, as shown in the code snippet of one of the files in Figure 7.

```

1 - name: permission on /etc/ssh/sshd_config configured
2   ansible.builtin.file:
3     path: /etc/ssh/sshd_config
4     owner: root
5     group: root
6     mode: 'u-x,go-rwx'
7
8 - name: Ensure sshd Banner is configured
9   lineinfile:
10    state: present
11    dest: /etc/ssh/sshd_config
12    regexp: '^#?Banner'
13    line: 'Banner /etc/issue.net'
14

```

Figure 7. Ansible playbook code snippet

Ansible will apply these configurations by executing playbooks that contain commands related to security hardening.

4.2.3. Server Provisioning Time

In the server provisioning test, the time required for manual provisioning was compared to automation using Terraform. Two manual scenarios were used: (1) using a cloud image, where server specifications are set manually before booting, and (2) using an ISO file, which involves a full manual installation. In both cases, configurations such as hostname, username, password, and network settings were applied after the server was started. These scenarios were used to strengthen the comparison with the automated method. The results are shown in Table 2.

Table 2. Provisioning Time

Provisioning Methode	Details	Average Time
Manual	Using ISO image	11 minutes 18 seconds
Manual	Using Cloud Init and hardening with Ansible	6 minutes 14 seconds
Automated	Using Terraform and Ansible	4 minutes 3 seconds

Based on the test results, manual provisioning using the ISO method was over 7 minutes slower than the automated approach, while the Cloud Init method was around 2 minutes slower. These results indicate that automation can significantly reduce server provisioning time.

4.2.4. Resource Usage

This section compares the manual approach with the automated method using Terraform and Ansible. In the manual method with Cloud Init, the monitored processes included qemu-system-x86, qemu-img, qemu, and ansible-playbook. Since provisioning was done manually, hardening was performed separately using Ansible. In contrast, the automated method ran

terraform, terraform-provisioning, qemu-img, and ansible-playbook in a single integrated execution.

Monitoring recorded average, maximum, and minimum values, comparing resource usage during active and idle states. The results are summarized in Figure 8, which illustrates the resource usage comparison between the two methods.

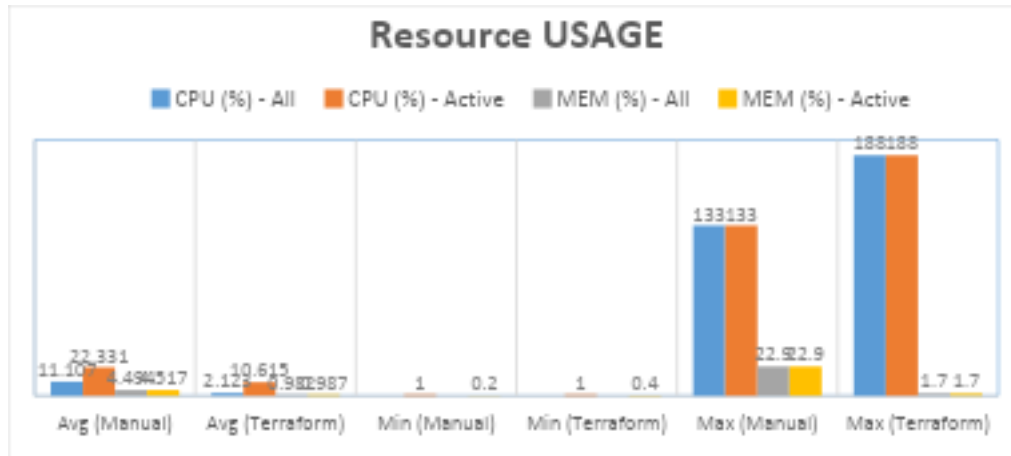


Figure 8. Resource Usage Chart

Based on the graph, the average CPU usage when using Terraform is lower compared to the manual method using Cloud Init. Similarly, memory usage shows the same trend.

4.3.NIST SP 800-123 Checklists

The testing followed the checklist from NIST SP 800-123. This document divides server operating system security into four subsections, outlined in Table 3.

Table 3. NIST SP 800-123 subsections

NIST SP 800-123 Subsection Titles	Total Checklist
Patching and Updating the Operating System	4
Hardening and Configuring the Operating System	18
Install and Configuring Additional Security Controls	4
Testing the Security of the Operating System	2

A total of 28 checklist items from NIST SP 800-123 were verified. The server checks were performed automatically using a system audit tool called Lynis. The details of the checklist for each subsection are on Table 4.

Table 4. NIST SP 800-123 checklist items result

NIST SP 800-123 Subsection Titles	Total Checklist	Fulfilled	Percentage
-----------------------------------	-----------------	-----------	------------

Patching and Updating the Operating System	4	1	25%
Hardening and Configuring the Operating System	18	15	83%
Install and Configuring Additional Security Controls	4	1	25%
Testing the Security of the Operating System	2	1	50%

Based on the checklist, several items were unmet. In the Patching and Updating section, patch management and vulnerability remediation were not implemented due to the need for managerial policies and lack of automated integration with sources like the National Vulnerability Database (NVD). Vulnerability scanning and mitigation were also absent.

In the Hardening and Configuring section, remote access tools like SSH remained active—though securely configured—and 2FA was not added due to its need for user interaction, making full automation unfeasible. In the Additional Security Controls section, tools such as anti-malware, IDPS, and vulnerability managers were not installed due to technical and resource constraints. Lastly, in the Testing Security section, PII-related checks were not applicable since the provisioned server held no sensitive data.

Out of the 28 checklist items, 18 were fulfilled, while 10 were not, resulting in a fulfillment percentage of 64.29%, calculated using formula (1).

$$\frac{\text{Fullfilled Checklist}}{\text{Total Checklist}} \times 100\% \approx \frac{18}{28} \times 100\% = 64,29\%$$

4.4. Testing Workflow

In this study, the tester conducted evaluations using two methods: security assessment and penetration testing. The security assessment was performed using two audit tools, namely Lynis and OpenVAS, as illustrated in Figure 9. From left to right in the figure, the Lynis assessment is shown on the left, OpenVAS assessment in the center, and penetration testing on the right.

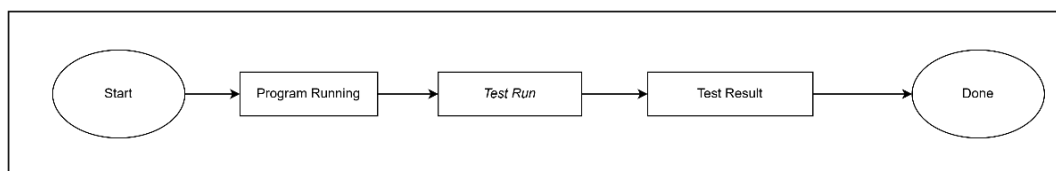


Figure 9. Workflow of Lynis, OpenVas, and penetration testing

4.5. Testing Result

4.5.1. Security Testing Result with Lynis and OpenVas

The vulnerability assessment was conducted using a system audit tool called Lynis. The automated audit with Lynis resulted in a Hardening Index score of 71, as shown in Figure 10.

```
Lynis security scan details:
Hardening index : 71 [##### ]
Tests performed : 278
Plugins enabled : 2

Components:
- Firewall [V]
- Malware scanner [X]

Scan mode:
Normal [V] Forensics [ ] Integration [ ] Pentest [ ]

Lynis modules:
- Compliance status [?]
- Security audit [V]
- Vulnerability scan [V]

Files:
- Test and debug information : /var/log/lynis.log
- Report data : /var/log/lynis-report.dat
```

Figure 10. Lynis Testing Result

The security assessment results using OpenVAS are presented in Figure 11 (before hardening) and Figure 12 (after hardening). Prior to hardening, one high-level, five medium-level, and three low-level vulnerabilities were detected. After hardening, only one low-level vulnerability remained.

Host	High	Medium	Low	Log	False Positive
192.168.2.5	1	5	3	0	0
Total: 1	1	5	3	0	0

Figure 11. OpenVas Testing Result Before Hardening

Host	High	Medium	Low	Log	False Positive
192.168.2.4	0	0	1	0	0
Total: 1	0	0	1	0	0

Figure 12. OpenVas Testing Result After Hardening

4.5.2. Penetration Testing Result

Penetration tests confirmed the effectiveness of SSH hardening. Initial tests using Hydra and password-based login succeeded. After switching to key-based authentication, all brute-force attempts failed. Full results are presented in Table 5.

Table 5. Penetration Testing Result

Type of Attack	Test Result Before Hardening	Test Result After Hardening
SSH Login (valid username and password)	Successful	Failed
SSH Login (invalid username and password)	Failed	Failed
SSH Brute Force (with valid password)	Successful	Failed
SSH Brute Force (without valid password)	Failed	Failed

This study demonstrates that automating server provisioning and hardening using Terraform and Ansible is both efficient and effective. Provisioning time was reduced to 4 minutes 3 seconds—significantly faster than manual methods—while also lowering CPU and memory usage, aligning with prior findings (Hariyadi & Marzuki, 2020; Pramadika & Chandra, 2023).

For security, 64.29% of the NIST SP 800-123 checklist was met, with a Lynis hardening index of 71 and a significant drop in OpenVAS-detected vulnerabilities. Unlike the manual approach in (Zein et al., 2023), this study fully automated the hardening process, reducing errors and improving consistency. SSH penetration tests also showed complete prevention of unauthorized access.

Overall, the integration of Terraform and Ansible guided by NIST SP 800-123 provides a reliable solution for secure and efficient infrastructure management, despite some tasks still requiring manual effort.

5. CONCLUSION

Automation with Terraform and Ansible resulted in lower CPU and RAM usage and reduced provisioning time to 4 minutes 3 seconds—significantly faster than manual methods 11 minutes with ISO, over 6 minutes with Cloud Init and Ansible.

From 28 NIST SP 800-123 checklist items, 18 were met (64.29% compliance). Lynis reported a hardening index of 71, and OpenVAS showed a reduction from 9 to 1 low-severity vulnerability. SSH penetration tests confirmed 100% failure of unauthorized attempts after hardening.

For future improvements, this automation can be extended to other virtualization or cloud platforms. Hardening standards may also be adapted based on server needs and application characteristics, enabling more targeted security and integration of patch management and advanced defenses.

REFERENCES

- Alfiandi, T., Diansyah, T. M., & Liza, R. (2020). Analisis Perbandingan Manajemen Konfigurasi Menggunakan Ansible Dan Shell Script Pada Cloud Server Deployment Aws. *JiTEKH*, 8(2), 78–84. <https://doi.org/10.35447/jitekh.v8i2.308>
- Anasrullah, F. (2024). Pencegahan Ransomware Pada Server on Premise Menggunakan Teknik Security Hardening. *Jurnal Informatika dan Teknik Elektro Terapan*, 12(3). <https://doi.org/10.23960/jitet.v12i3.5105>
- Anelia, S. S., Jayanta, & Hananto, B. (2023). Uji Penetrasi Server Universitas PQR Menggunakan Metode National Institute Of Standards And Technology (NIST SP 800-115). *Jurnal Ilmu Teknik dan Komputer*, 7(1), 35–43.
- BSSN. (2024). Laporan Tahunan:Monitoring Keamanan Siber 2023. <https://idsirtii.or.id/halaman/tentang/laporan-hasil-monitoring.html>
- Evianti, N., Wihandar, M. A., & Kurniawan, A. (2021). Automation Provisioning Dev-Ops Website Server Menggunakan Ansible Dan Vagrant. *Jurnal Nasional Informatika*, 2(2), 72–91.
- Hariyadi, I. P., & Marzuki, K. (2020). Implementation Of Configuration Management Virtual

- Private Server Using Ansible. *MATRIK : Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, 19(2), 347–357. <https://doi.org/10.30812/matrik.v19i2.724>
- Herdianto, U. Y. S., Aditya, Y. S., & Fathinuddin, M. (2021). Pengamanan Data Cloudfri Menggunakan Metode Security Hardening Securing Cloudfri Data Using Security Hardening Method. *e-Proceeding of Engineering*, 8(5), 9428–9438.
- Irfandi, F. R., Hediando, U. Y. K. S., & Almaarif, A. (2022). Software Security Hardening Pada Virtual Private Server Berdasarkan NIST SP 800-123 di Universitas XYZ. *Journal of Information System Research (JOSH)*, 4(1), 94–102. <https://doi.org/10.47065/josh.v4i1.2299>
- Khumaidi, A. (2021). Implementation of Devops Method for Automation of Server Management Using Ansible. *Jurnal Transformatika*, 18(2), 199. <https://doi.org/10.26623/transformatika.v18i2.2447>
- Kurniawan, D., & Saputra, Y. M. (2024). Implementasi Ansible pada Otomasi HoneyPot Deployment Berbasis Web. *Journal of Internet and Software Engineering (JISE)*, 5(2), 86–98.
- Peddireddy, A. R. (2024). Effective Workflow Automation in GitHub: Leveraging Bash and YAML. *Journal of Artificial Intelligence & Cloud Computing*, 3(3), 1–7. [https://doi.org/10.47363/jaicc/2024\(3\)345](https://doi.org/10.47363/jaicc/2024(3)345)
- Pramadika, O., & Chandra, D. W. (2023). Provisioning Google Kubernetes Engine (GKE) Cluster dengan Menggunakan Terraform dan Jenkins pada Dua Environment. *JUPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)*, 8(2), 597–606. <https://doi.org/10.29100/jupi.v8i2.3630>
- Pratama, M. A. A., & Hariyadi, I. P. (2021). Otomasi Manajemen dan Pengawasan Linux Container (LCX) Pada Proxmox VE Menggunakan Ansible. *Jurnal Bumigora Information Technology (BITe)*, 3(1), 82–95. <https://doi.org/10.30812/bite.v3i1.807>
- Putra, W. R. A., Nurwa, A. R. A., Priambodo, D. F., & Hasbi, M. (2022). Infrastructure as Code for Security Automation and Network Infrastructure Monitoring. *MATRIK : Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, 22(1), 201–214. <https://doi.org/10.30812/matrik.v22i1.2471>
- Scarfone, K., Tracy, M., & Jansen, W. (2008). Guide to General Server Security. In NIST Special Publication - 800 seriesication. <https://doi.org/10.6028/NIST.SP.800-123>
- Soms, N., Oswalt, M. S., & Santhosh, K. P. (2022). Case Study on Cloud Security Controls. *International journal of health sciences*. <https://doi.org/10.53730/ijhs.v6ns1.7778>
- Wibowo, D. S., Ilmadina, H. Z., Ardi, A. S., & Insani, F. F. G. (2023). Apache web server security with security hardening. *Journal of Soft Computing Exploration*, 4(4), 213–221. <https://doi.org/10.52465/joscex.v4i4.230>
- Zein, M. A., Hediyanto, U. Y. K. S., & Almaarif, A. (2023). Hardening Sistem Operasi Virtual Private Server Fakultas Rekayasa Industri Berdasarkan Nist Sp 800- 123. *JUPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)*, 8(1), 230–241. <https://doi.org/10.29100/jupi.v8i1.3438>