



HardnBot
Intelligent Server Hardening Software

Project ID: 19_20-J 01

Final Report

Wijekoon W.M.K.M.W

IT16167742

Bachelor of Science (Hons) in Cyber Security

Information Systems Engineering Department

Sri Lanka Institute of Information Technology

Sri Lanka

May 2020

HardnBot
Intelligent Server Hardening Software

Project ID: 19_20-J 01

Final Report

Wijekoon W.M.K.M.W

IT16167742

Supervisor Mr. Amila Nuwan Senarathne

Bachelor of Science (Hons) in Cyber Security

Information Systems Engineering Department
Sri Lanka Institute of Information Technology
Sri Lanka

May 2020

DECLARATION

I declare that this is my own work and this Final report (Thesis) does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

.....

Wijekoon W.M.K.M.W

(IT16167742)

The above candidate is carrying out research for the undergraduate Dissertation under my supervision.

Supervisor

.....

Mr. Amila Nuwan Senarathne

Abstract

In nowadays organizations are trending to the use of information technology even the small, per capita businesses. This can be for the use of databases, systems, or any other service providing tasks to the organization's customer base. To maintain that kind of an information technology-based operations, servers may be used. Servers are varying from Linux to Windows and internal to exposed. When it comes to server operating system security, server hardening is one of the most important task to be handled on servers thus without server hardening, a server may be exposed to various of threats from organization's internal sources or from external cyber criminals. Server operating system hardening is based on a list of compliance that are differ from various server-type operating systems. Most of these compliance details are listed in the Center for Information Security website as CIS Benchmarks [1].

Normally these hardening processes will be done by either network administrators, system administrators, outsourced professionals, or server custodians by manually running scripts, commands, and queries against the server and it will roughly take more than six hours to completely harden a single server in the infrastructure. And before doing any hardening operations, a system administrator must first find any compliance failures in the operating system that runs on the server. To do this, the system administrator must first run a compliance audit against the server which most of the scanners take roughly more than 2 hours.

In this paper, we dive into the consideration of developing an automated system which can achieve all compliance audits as well as providing system administrators with severity level classification counts (High, Medium, Low) with respective to the failed compliances. In this realized scenario, more than 10 Linux servers (RedHat and CentOS) were utilized to test this developed functionality. To bring the introduced functionality closer to system administrators, the software was created and thoroughly tested.

Keywords – Server Hardening, Classification, Compliance Audit, Automation, Linux

Table of Contents

DECLARATION	iii
Abstract	iv
List of Figures.....	vi
1. Introduction	1
1.1 Definitions, Acronyms, and Abbreviations	1
1.2. Overview.....	1
1.3. Project Scope	2
1.4. Background & Literature Survey.....	2
1.5. Research Gap.....	8
2. Objectives.....	9
2.1. Main Objective.....	9
2.2. Specific Objectives	9
3. Methodology	10
4. Testing and Implementation.....	25
5. Test Results and Discussions	26
6. Conclusion.....	29
7. References.....	30

List of Figures

Figure 1 - system plan of cloud compliance audit	6
Figure 2 - System diagram	10
Figure 3 - HardnBot's Connection Interface	10
Figure 4 - HardnBot's CLI	12
Figure 5 - HardnBot's Scan interface	15
Figure 6 - Scan verification.....	15
Figure 7 - Scanning process	16
Figure 8 - Scan results.....	16
Figure 9 - Classified compliance count.....	17
Figure 10 – Sample CIS Benchmark recommended audit command	24
Figure 11 - Scan Script's audit service status checking function (ID: 4.1.2)	24
Figure 12 - Sample Nessus report	26
Figure 13 - HardnBot's Scan results.....	26
Figure 14 - HardnBot's scan time.....	27
Figure 15 - Time gap.....	27

1. INTRODUCTION

1.1 Definitions, Acronyms, and Abbreviations

CIS	Center for Internet Security
GUI	Graphical User Interface
OS	Operating System
NIST	National Institute of Standards and Technology
VPN	Virtual Private Network

1.2 Overview

Most data centers and organizations include Linux servers and RedHat Enterprise Linux, CentOS, or Solaris as their base operating system. A normal medium ranged organization includes about 20+ live servers and it is almost very difficult to perform hardening in each server operating system.

Normally these hardening processes will be done by either network administrators, system administrators, outsourced professionals or server custodians by manually running scripts, commands and queries against the server and it will roughly take more than six hours to completely harden a single server in the infrastructure. Probability of a misconfiguration occurrence is higher because hardening will carry out with human interaction. Scenarios where a misconfiguration occurs, it may be difficult to detect those issues since some issues cannot be identified via an error message. So, this is an example issue when it comes to manual server hardening.

Our solution, HardnBot, is an automated unix server hardening software platform that has the capability to detect and classify failed compliance issues and configurations of an unix server and applying hardening solutions according to the industry recommended best practices or organizational recommended best practices. HardnBot also has the capability of predicting the overall risk score by considering main and all available risk factors and if there is any abnormal behavior after the hardening process, HardnBot has the capability to roll back the modification to an accepted previous level. Also, we use a scanning mechanism to find and identify every compliance issues and misconfigurations of a Linux server and this goal is achieved by executing structured and custom configured scripts against the target server. Output of this functionality give us all the failed compliance with respect to their compliance ID which will classify later accordingly by a pre-classified data set. In this paper we are going to discuss the theoretical approach and comparisons regarding previous researches, experiments we conduct, and obtained results based on those experiments.

1.3 Project Scope

HardnBot is a software that has the capability to identify failed operating system compliances of a unix based servers and classify those failed compliances according to a pre-classified compliance severity data set and use those compliance data to apply industry recommended best practices or organizational required fixes to the unix based operating system of a server. Throughout this document, we will explain how failed compliances classification process is going to achieve its goal and steps that needed to be taken.

Under this document, following components are described.

- a) Scan for compliance failures.
- b) Classify them using a pre-classified dataset into severity levels as High, Medium, and Low.

HardnBot consist with four main novel components,

- Issue classification
- Risk score prediction
- Intelligent hardening
- Backup and smart rollback

Within those main components there are subcomponents/functionalities and throughout this document, issue classification component and its subcomponents will be thoroughly discussed.

1.4 Background & Literature Survey

Although there are vulnerability scanners which has the capability of scanning operating system compliance failures and although they can classify vulnerabilities according to the CVE scores provided by the NVD, they are not capable of providing classification for those identified compliance failures but only whether they are failed or passed.

Ratsameetip Wita and Yunyong Teng-Amnuay of Department of Computer Engineering, Chulalongkorn University, Bangkok, Thailand published a research paper in 2005 on “*Vulnerability Profile for Linux*”. In this research, they talk about profiling identified vulnerabilities according to the CVE score of them. In their classification scheme, they consider four types of classification schemes namely,

1. Confidentiality violation
2. Integrity violation
3. Availability violation
4. System compromised

If a confidentiality violation occurs, it allows an attack to directly steal information from the system. Integrity violations allows an attack to directly change the information passing through the system. Availability violation results an attack that limit the genuine access to a genuine user (human or machine), Denial of service attacks (DOS) can be taken as an example. According to their research system compromised attacks gives the attacker the privilege to access the system in four different levels such as: run an arbitrary code, elevate privilege, account break-in, and finally root break in which can be the worst-case scenario. Furthermore, these classifications are again grouped according to the severity level.

Damage Type	Severity Level		
	High	Medium	Low
Confidentiality	-Disclosure of information and system configuration in root/super user level	- Disclosure of system information and configuration in user level	- Disclosure of some no relevant information.
Integrity	- Information and system configuration changed in root/super user level	-Information changed in user level	-Non-relevant information changed in another user level
Availability	-Whole system crash or unavailable	- Some services unavailable -System temporary unavailable	- Some services temporary slowdown with flooding
System compromised	-Root break-in -Account break-in -Run arbitrary code by root/super user privilege	- Privilege gain in some domain - Run arbitrary code by user privilege	- Run arbitrary code by another user privilege

[2]

Here they are classifying vulnerabilities of the system / server, but we are going to develop a software toolkit that is capable of identify and classify failed compliance issues of a server operating system.

A. Baith Mohamed M. of Computer Engineering Department, College of Engineering & Technology, Arab Academy for Science & Technology Alexandria, Egypt published a research paper in 2001 on “*An Effective Modified Security Auditing Tool (SAT)*” which is a research about software tool to audit security configurations of a system.

In this research they have explained how to identify an exploitable vulnerability of an operating system via a security audit. This tool gathers much information from a remote hosts and network services such as ftp, NFS and according to those gathered information it will check for any security flaws, misconfigurations and other poor policy implementations that will put data at risk. As solutions, this

tool can either report on this output data or it can use a rule-based system to investigate any potential security problems. However, according to this research, their main function of this tool is to iterate future data collection of secondary hosts using the initial data collection and user configurations for the next audit process. Furthermore, this tool can also analyze a complicated network and make practically informed decisions about the security level of the systems involves [3].

Kuo Zhao, Qiang Li, Jian Kang, Dapeng Jiang, and Liang Hu of Department of Computer Science and Technology, Jilin University, Changchun 130012, China published a research paper in 2007 on “*Designing and Implementing of Secure Auditing System in Linux Kernel*” which is also a tool to audit the system kernel of a Linux based system.

This research is about a tool to audit the kernel in a unix based system. Although there is a log collection mechanism in unix based systems, they are only based on application-level. A typical example for such subsystem is the “syslogd” daemon. It mainly receives important information of restricted services and process according to the configuration files. However, in this research, there main goal is to go beyond the typical user-state auditing and provide with a more detailed security audit result which contains both name of the system calls and related object of that. Since the current log files can be accessed, it will be a security issue and, in this paper, they also discuss about the security of all audit logs of this kernel auditing component as well. Later system administrators can view a completely in-depth detailed kernel state and user state logs for taking decisions for the system [4].

ling Liul, Xiaoni Wang, Dongliang liao, and Chen Wang from Engineering teach and practice training center, Tianjing Polytechnic University, Tianjin, China Key Laboratory of Beijing Network Technology, and Beihang University, Beijing, China altogether published a research paper in 2012 on “*Research and Design of Security Audit System for Compliance*” which explains the concept of compliance audit, and then proposed a log-based network security audit system for compliance.

In this over-all research, they first explain the whole concept of compliance audit and then they introduce the system architecture and components of a log-based network security audit system for compliance. In the system architecture, they have main five components which are,

1. Log analyzer:
To analyze log information and send the alarm result to correlation analysis engine.
2. Correlation analysis engine:
This uses the audit analysis method based on data mining to give a countless suggestion analysis on the network security alarm events of different sources, different time, different levels so that they could dig out the real security events, find hidden complex attacks, Identify real security threats, and finally generate the system security alarm information.
3. Query statisticser:
Query statisticser allows users to set the conditions for their own needs to view the details stored in the database relating to the event including the log information, alarm information, policy information and other system settings information.
4. Compliance manager:
Compliance manager can provide the policy templates for the audit terminals to generate policy configuration information to check compliance posture of the audited system according

to their audit trails and ensure that they follow all policies required by an external or internal regulation.

5. Management Console:

It provides a convenient, intuitive management interface and visually displays the event analysis results and network security status; users could use it to complete software system configuration, security policy definition and information inquiry, the management console is responsible to receive the audit data from the agents, store the data into the database and release the audit policies to the audit agents.

In the end of this paper they mentioned the used technologies and other detailed design of key components [5].

Frederick Yip, Alfred Ka Yiu Wong, Nandan Parameswaran, Pradeep Ray from School of Computer Science and Engineering School of Information Systems and Technology Management University of New South Wales, Sydney Australia published a research paper in 2007 on “*Robust and Adaptive Semantic-Based Compliance Auditing*” which contains a deep research about compliance management and compliance auditing.

In this research, researchers state that the compliance auditing is a child process of compliance management, where compliance rules and policies are individually check against the organization to determine the level of compliance achieved by the organization. In this research paper, they deeply describe about all Information Technology required policies and standards such as,

1. Sarbanes Oxley (SOX) Act - Government Regulations and Information Security Standards
2. HIPAA – Health Insurance Portability and Accountability Act - Government Regulations and Information Security Standards
3. ISO/IEC 17799:2005 - Information Security Standards
4. Control Objectives for Information and related Technology (CobiT) - Information Security Standards

So, this is an overall big picture of the idea of compliance. However, in our research, we are going to limit this scope form organization to a Linux system [6].

Frederick Yip from School of Computer Science & Engineering University of New South Wales Sydney, Australia, Pradeep Ray from School of Information Systems & Technology Management University of New South Wales Sydney, Australia, and Nandan Parameswaran School of Computer Science & Engineering University of New South Wales Sydney, Australia, altogether published a research paper in 2006 on “*Enforcing Business Rules and Information Security Policies through Compliance Audits*”.

In this paper they present XISSF, an extensible information security specification format that acts as a compliance audit mechanism for enforcing business rules and information security policies [7].

Uttam Thakore from Department of Computer Science University of Illinois at Urbana-Champaign Urbana, IL 61801, Rohit Ranchal from IBM Cambridge, MA and Yi-Hsiu Wei and Harigovind V. Ramasamy from IBM Austin, TX 78758 altogether published a research paper in 2019 on “*Combining Learning and Model-Based Reasoning to Reduce Uncertainties in Cloud Security and Compliance Auditing*” which is a research based on cloud server compliance audit.

In this research they propose a hybrid approaches, in which formal, model-based approaches are combined with machine learning techniques to reason about evidence and historical audit data, are necessary to address any human mistakes or errors in the cloud environment [8].

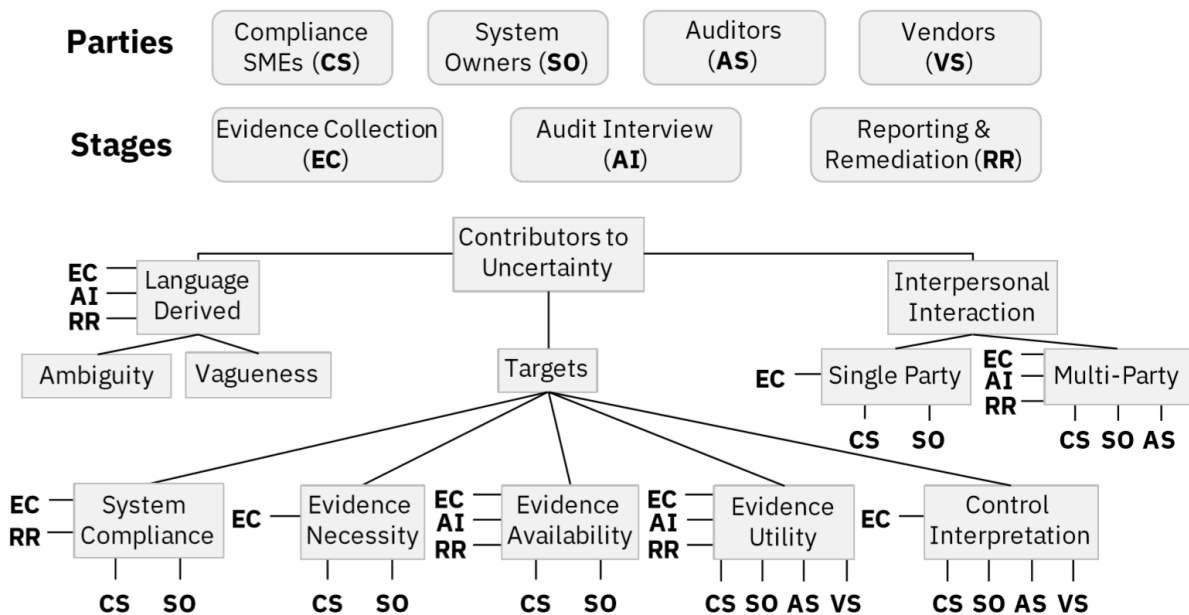


Figure 1 - system plan of cloud compliance audit

Rafael Enrique Rodriguez-Rodriguez, Andrés Felipe Quevedo Vega, Andres Felipe Sanchez, Alexandra López, and Jaime Fernando Pérez of Facultad de Ingeniería, Universidad Católica de Colombia, Bogotá D.C, Colombia published a research paper in 2018 on “*Design of an Automation Model for Taking Documentary Evidence of Compliance Tests of the IT Audit*” which state a method for taking documentary evidence of compliance test of the IT audit.

In this paper the researchers focused on designing a functional model for crating and executing of evidence gathering of compliance audits performed by the organization. This is basically a reporting function for compliance auditing. And in our research, we also use some reporting methodologies to collect evidence of the compliance audit performed.

In this research they categorized the audit methodologies as shown in the below table.

Feature	ISO 27001	Megerit V3	ISO 31000
Risks Management	Identify threats Guidelines for risk analysis Analysis of risk management in ISMS Has a risk management approach	Promotes the identification of threats Systematic method for risk analysis Discover and plan risk Performs qualitative and quantitative risk analysis	Generates a stage to identify threats Poses rules for risks analysis

Senior Management Consideration	Its application demystifies that the security of information is a technical issue	Consider those responsible for the organization Prepare the organization for the application of audits	Its application helps management to control risks
Application of controls	Set default controls for its application It divides the controls by classes such as: logical security, audit trails, integrity, continuity, and physical security.	Plan the implementation of controls Categorizes controls to manage risks	Establish guidelines for the establishment of controls
Asset Management	Manage assets through inventory Establishes the owners of each asset Manage guidelines for the acceptable use of assets Establish asset classification guidelines	Guides the classification of assets Divide the assets into several groups to identify the risks more clearly Manage the description of assets to identify risks	Make the classification of assets
Type of companies	It is aimed at all types of companies	Public Organizations Non-profit organizations	It is aimed at all types of companies

[9].

1.5 Research Gap

When performing a hardening process, system administrators, network administrators, server custodians or outsourced expertise need to ensure security of the operating system that runs on a server (in our case its unix based servers) database and application because a single mistake can affect the whole production line which the server is in. Even though there are many scanning tools that has the capability of scan a server and identify compliance failures along with the solutions, the solutions are going to apply with a human interaction. So, the probability of mistake occurrence is higher. And manual hardening process consume much more resources such as time, human, cost likewise. As a solution for the lack of resources, organizations are tending to consider about hiring external professionals and assets to perform the hardening task. In a scenario like this, internal critical classified information might have a possible chance to expose via outsourced professionals intentionally or unintentionally and leave the server in a critical position of been hacked or data leaked. And there is a compulsory requirement of the root (administrator) access to the server terminal to scan and perform operating system hardening. Also, the server needs to be temporarily out of live production because operating system hardening cannot be performed while the server is in live production environment. So, when a critical day-to-day serving server is downed for maybe more than six hours to perform operating system hardening, it will be a critical impact on the organization's day-to-day business activities.

As an example, for all these above described scenarios, we can consider an operating system hardening process of a card server of a commercial bank that provides customers with all kind of credit and debit card services. Since the hardening is done by outsourced professionals and a down time is required, all servers information are available to outsourced personals that include customer card details as well as the server details such as the root password and also because of the downtime required, legitimate customer services will be down. So, in a case like this it will be a critical situation to the organization. Likewise, there are more drawbacks to a manual operating system hardening process.

To solve these types of difficulties and prevent intentional and unintentional human errors, we are going to implement a software platform (HardnBot) which automate the server operating system hardening process and which has the capability of detect failed compliances, classify them based on their criticality/severity levels and apply industry recommended best fixes for them via CIS benchmarks or via organizational requirements. And, there is no system implemented that has the capability of scan a linux server for any compliance failures and to classify them according to a criticality level.

2. OBJECTIVES

2.1 Main Objectives

This research aims to find an effective way to secure the datacenter Red Hat Enterprise Linux version 6, 7, and CentOS version 7, 6 servers and enhance productivity for the customers and employees with a low cost and few human interactions. Hence there is no fully automated hardening platform implemented yet, in this system implementation a novel automated open-source software is proposed.

Here, fully automated free open source hardening platform is developed with the capability to detect poor or non-compliant configurations in a system (OS/DB/Application) and applying industry recommended fixes/configurations and secure systems by reducing its surface of vulnerabilities.

This research composes a great business value as it can cover 90% of an Information Systems Audit and hardening process. For internal audits, this software presents both opportunity and responsibility. By helping the organization understand and control risks and identifying opportunities/industry best practices to embrace. This software also will allow the internal audit to position themselves as trusted advisors.

- i. Automatically detect failed compliances in a server.
- ii. Automatically classify them according to a criticality level.

2.2 Specific Objectives

- i. Reduce manual work

Through this software, we target to reduce the manual effort needed for a complete system hardening. Almost all the parts of the audit and hardening could be performed through this software. Compliance issues are automatically detected. Reports can be generated through this software at the end of the hardening process.

- ii. Ease the internal audit

This software is designed in a way that could be easily used by non-technical people. Simple interfaces, pop-up guidelines, and descriptive reports will make the user aware of the functionalities of the software. Even before applying a fix to an issue, the user can read a full detailed report of that issue including its impact and the remediation.

This software package can be handled by a single user. Therefore, the entire system's hardening process can be conducted by a single user. This requires a minimal amount of the organization's resources. As for the internal audit, this software will cover up to 90% of their work.

3. METHODOLOGY

HardnBot's compliance scanning function will scan the server compliance status and pass that results into a pre-classified compliance dataset to classify all failed compliances according to the severity levels.

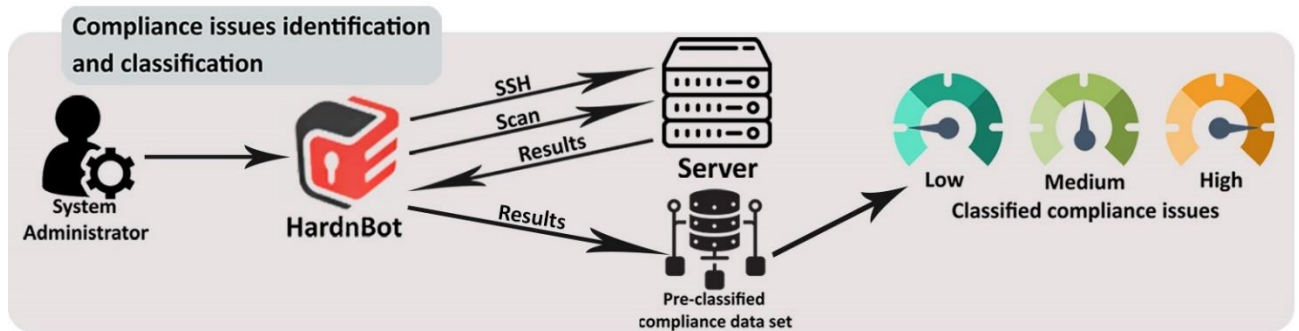


Figure 2 - System diagram

Setup a connection between the HardnBot and the remote server.

HardnBot has a separate interface to handle the connection to a remote Unix server, using this interface system administrator can connect, execute unix terminal commands and disconnect form a server. To connect to a remote server, a system administrator must have the required remote server up and running and ssh services enabled to HardnBot to connect. Also, the system administrator must have the server IPv4 address, and root credentials.

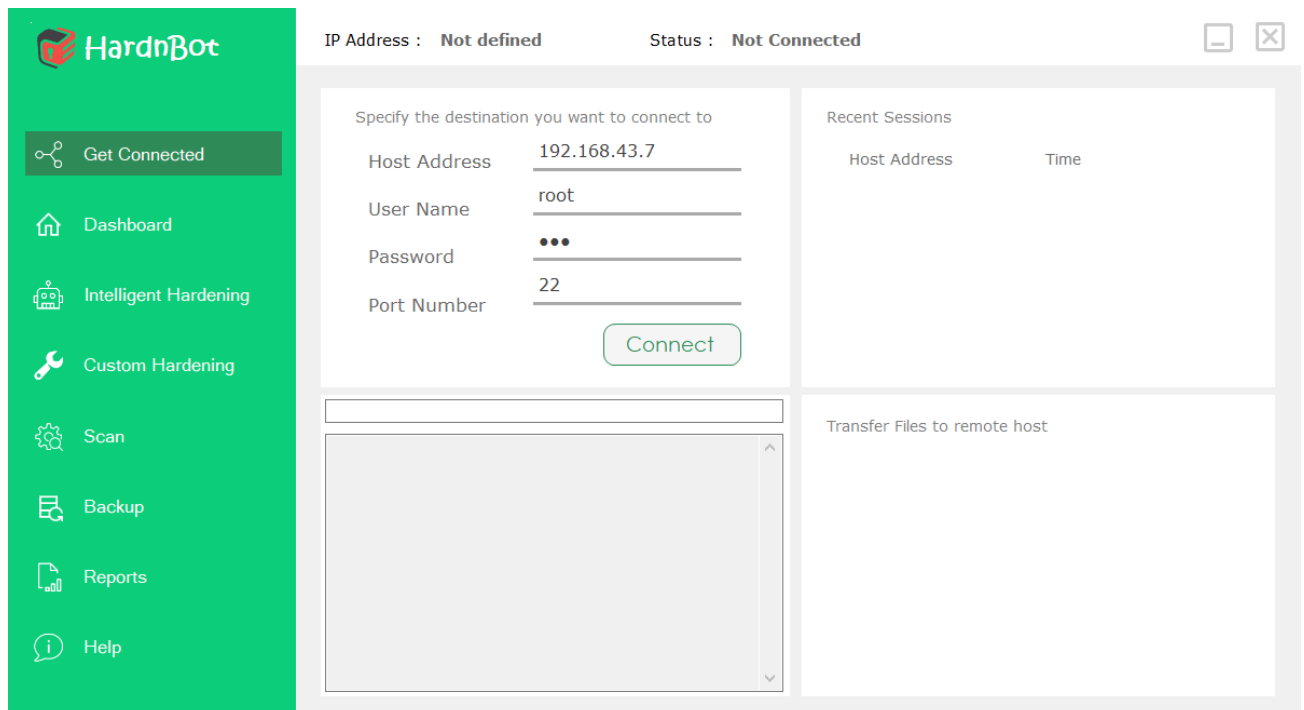


Figure 3 - HardnBot's Connection Interface

When developing the scanning interface, we used “*Bunifu UP*” framework to design the interface. We used C# as our main programming language of the software, but the required server-side scripts are developed using shell.

To get the connection, we used C# library named “Renci SshNet/SSH.NET [10, 11]” and whenever the system administrator clicks on the “*Connect*” button HardnBot will get the user inputs (destination IP, root username, root password and the port) provided by the system administrator and pass it to the SshClient object we created. Next HardnBot will connect to the remote server and wait for 120 seconds to get a successful connection. Within this time if a connection failure occurs, HardnBot will notify the user before the 120 seconds we defined.

If the remote connection is successful, HardnBot will save the connection status to a variable, and this variable will be very useful in-order to check the connection status to other functions in HardnBot to function properly.

Below is the code segment of this functionality.

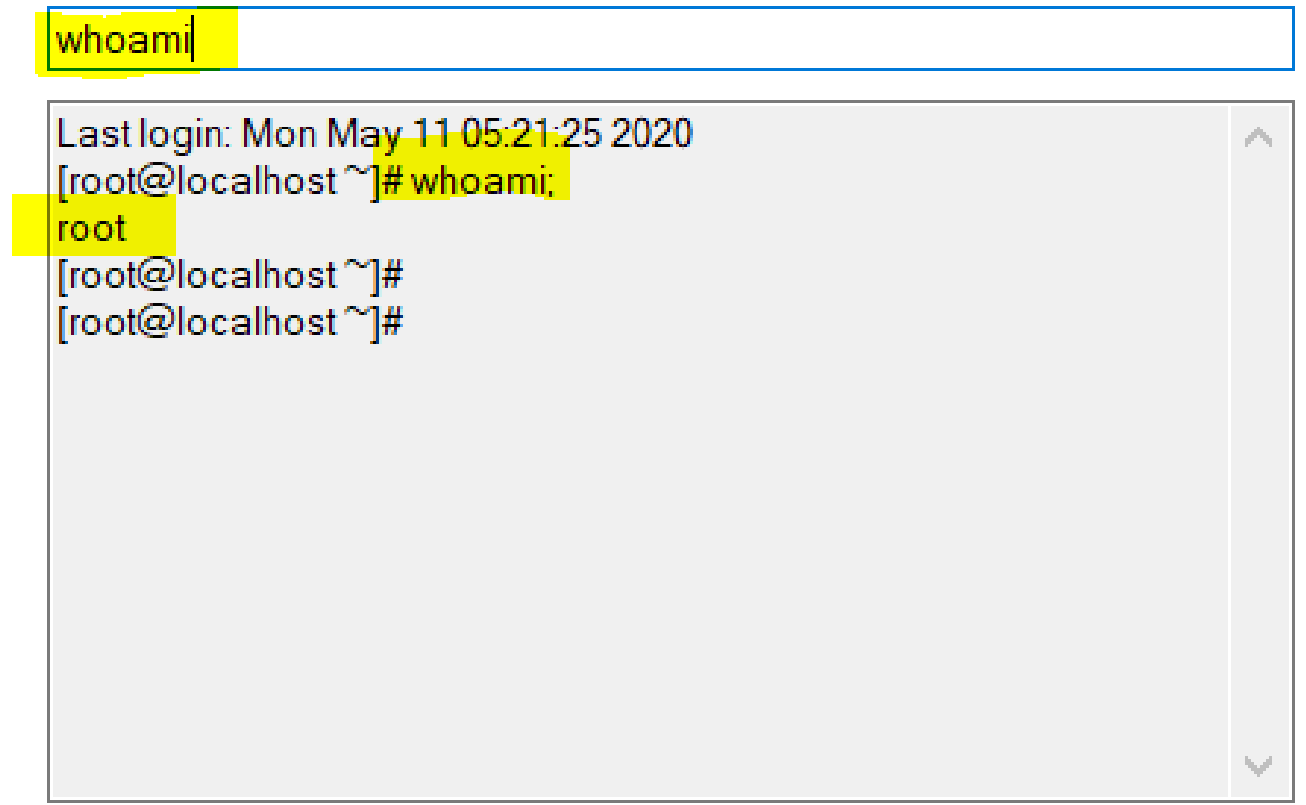
```
public static Boolean connectionstat;

this.sshClient = new SshClient(textBox1.Text, Convert.ToInt32(textBox4.Text), textBox2.
Text, textBox3.Text);
this.sshClient.ConnectionInfo.Timeout = TimeSpan.FromSeconds(120);
this.sshClient.Connect();
connectionstat = this.sshClient.IsConnected;
```

After connecting to the remote server, HardnBot will get the server’s Operating system information, Kernel and Kernel released date and save them to variable that are used later to display this information to the user.

```
//get OS
SshCommand OS = this.sshClient.CreateCommand("uname -o");
OS.Execute();
os = Convert.ToString(OS.Result.ToString());
//get kernal
SshCommand KERNAL = this.sshClient.CreateCommand("uname -s");
KERNAL.Execute();
kernal = Convert.ToString(KERNAL.Result.ToString());
//get kernal_release-date
SshCommand Release = this.sshClient.CreateCommand("uname -r");
Release.Execute();
kernal_release = Convert.ToString(Release.Result.ToString());
```

After this, all parameters are saved to variables to later use. And then HardnBot will create the shell stream to get user commands form HardnBot connection interface.



The image shows a terminal window with a blue border. At the top, a yellow highlight covers the text 'whoami'. Below this, the terminal output is displayed on a light gray background. The output includes the message 'Last login: Mon May 11 05:21:25 2020', followed by the prompt '[root@localhost ~]# whoami;', the output 'root', and two subsequent prompts '[root@localhost ~]#' and '[root@localhost ~]#'. A vertical scrollbar is visible on the right side of the terminal window, with upward and downward arrow icons at the top and bottom.

```
whoami  
Last login: Mon May 11 05:21:25 2020  
[root@localhost ~]# whoami;  
root  
[root@localhost ~]#  
[root@localhost ~]#
```

Figure 4 - HardnBot's CLI

```

IP = textBox1.Text;
UN = textBox2.Text;
PW = textBox3.Text;
port = textBox4.Text;
this.shellStreamSSH = this.sshClient.CreateShellStream("vt100",80,60,800,600,65536);
connectbtn.Visible = false;
disconnectbtn.Visible = true;
MessageBox.Show("Successfully Connected");
label5.Text = IP;
label7.Text = DateTime.Now.ToString();

//required method for the shell stream.
private void recieveSSHData()
{
    while (true)
    {
        try
        {
            if (this.shellStreamSSH != null && this.shellStreamSSH.DataAvailable)
            {
                String strdata = this.shellStreamSSH.Read();
                appendTextBoxInThread(txtSSHConsole, strdata);
            }
        }catch
        {
        }
        System.Threading.Thread.Sleep(200);
    }
}

private void appendTextBoxInThread(TextBox t, String s)
{
    if (t.InvokeRequired)
    {
        t.Invoke(new Action<TextBox, string>(appendTextBoxInThread), new object[] { t, s });
    }
    else
    {
        t.AppendText(s);
    }
}

```

```

private void txtCommand_KeyPress(object sender, KeyPressEventArgs e)
{
    try
    {
        if (e.KeyChar == '\r')
        {
            this.shellStreamSSH.Write(txtCommand.Text + "\r\n");
            this.shellStreamSSH.Flush();
        }
    }
    catch
    {
        throw;
    }
}

```

We also used accessor methods to return private static variables to other classes.

```

//accessor method for kernal
public static String getKN()
{
    return kernal;
}
//accessor method for kernal_release
public static String getKR()
{
    return kernal_release;
}
//accessor method for OS
public static String getOS()
{
    return os;
}
//accessor method for IP
public static String getIP()
{
    return IP;
}
//accessor method for UN
public static String getUN()
{
    return UN;
}
//accessor method for PW
public static String getPW()
{
    return PW;
}
//accessor method for Port
public static String getPort()
{
    return port;
}

```

Scan the connected server for security compliance misconfigurations.

After a successful connectivity to the remote server, system administrator can perform the compliance audit/scan against the server by navigating to the scanning interface.

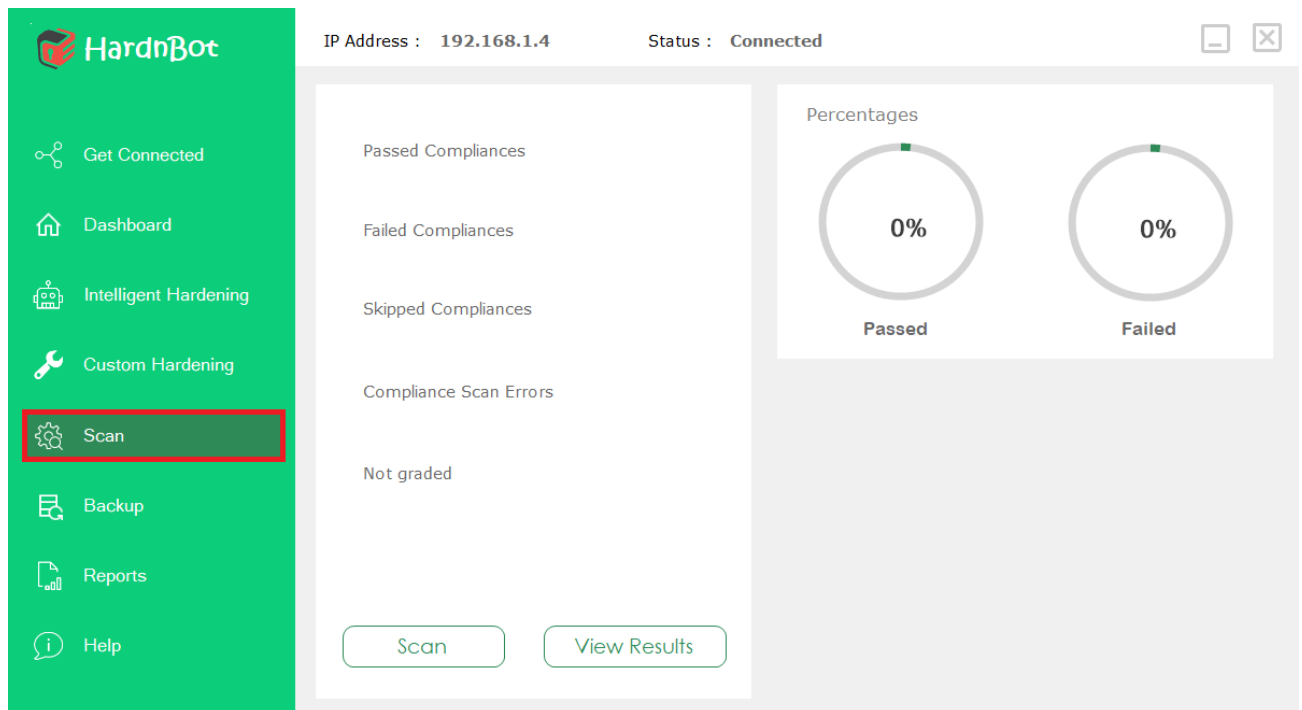


Figure 5 - HardnBot's Scan interface

After navigating to the compliance scanning interface, system administrator can click on “Scan” button and if there is a connection, HardnBot will ask the user to verify the scan before executing the scanning against the server.

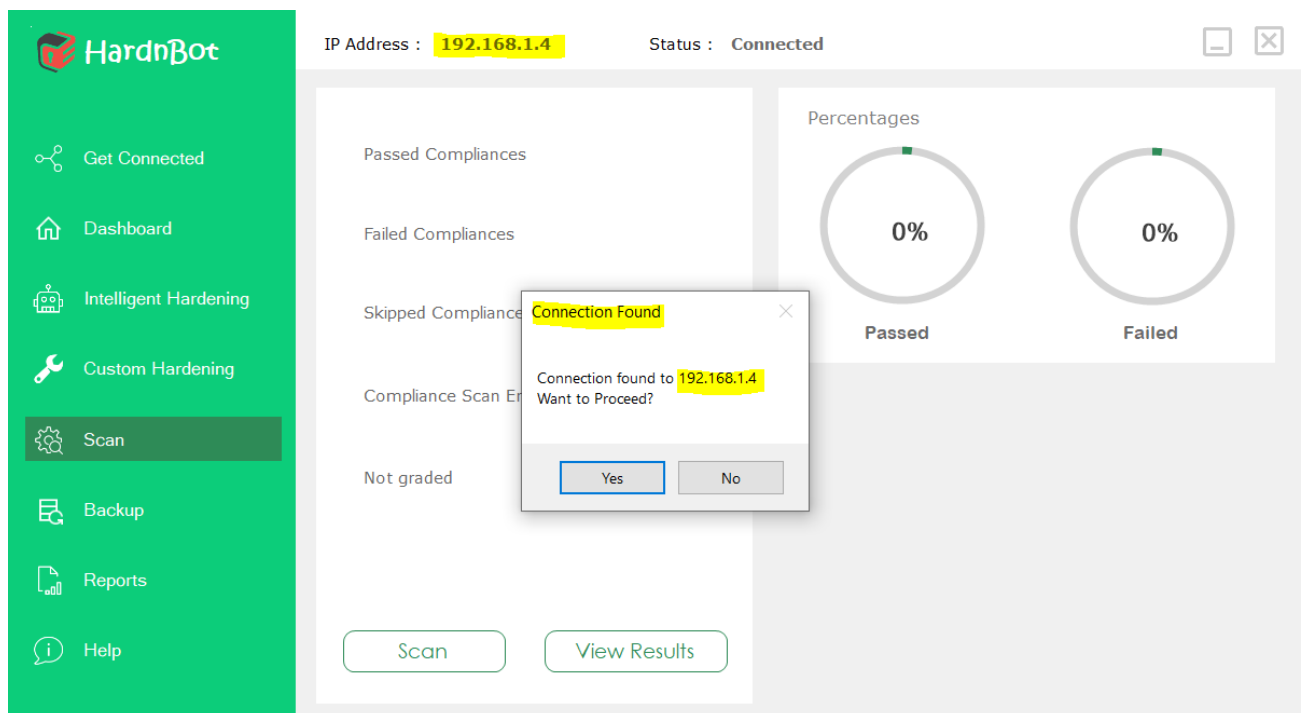


Figure 6 - Scan verification

If the user clicks on “Yes” the scanning process will begin.

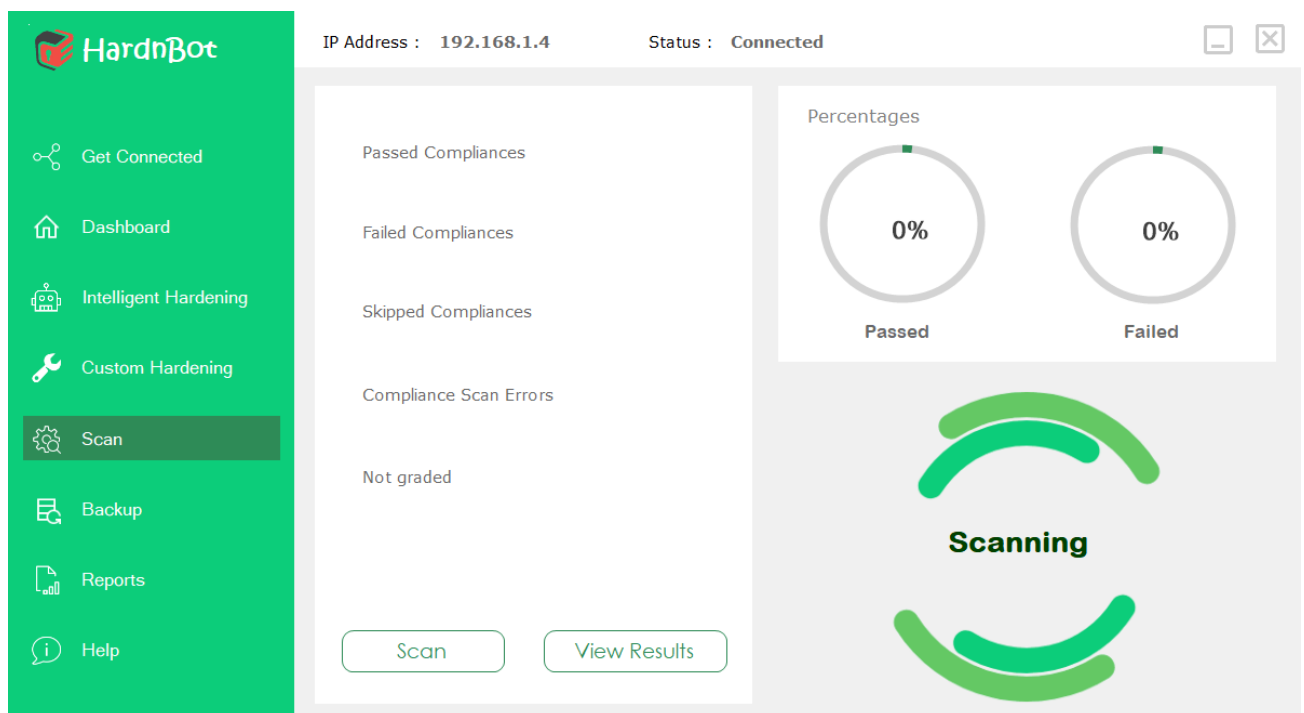


Figure 7 - Scanning process

After the scanning, HardnBot will receive the compliance audit results and display them in the scan dashboard following a message that says the “Scan Complete”.

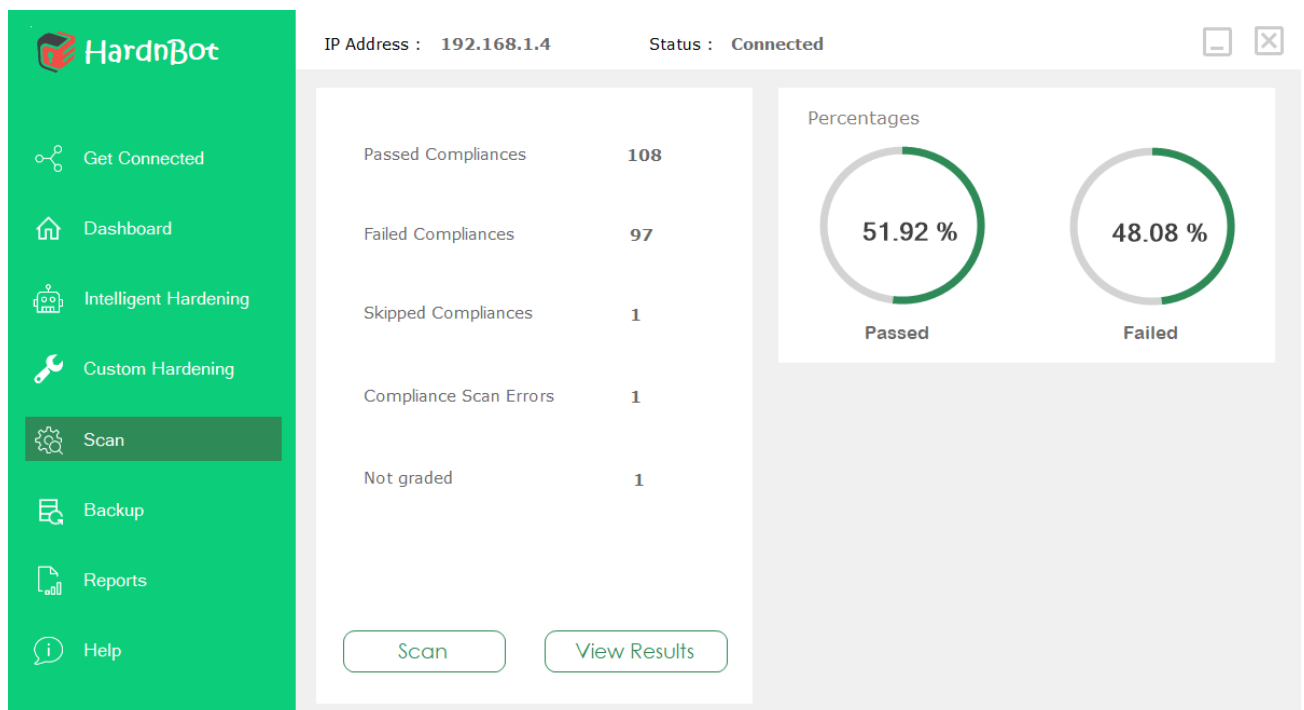


Figure 8 - Scan results

After that, user can press on the “*View Results*” button to get the classified compliance count according to the predefined severity levels.

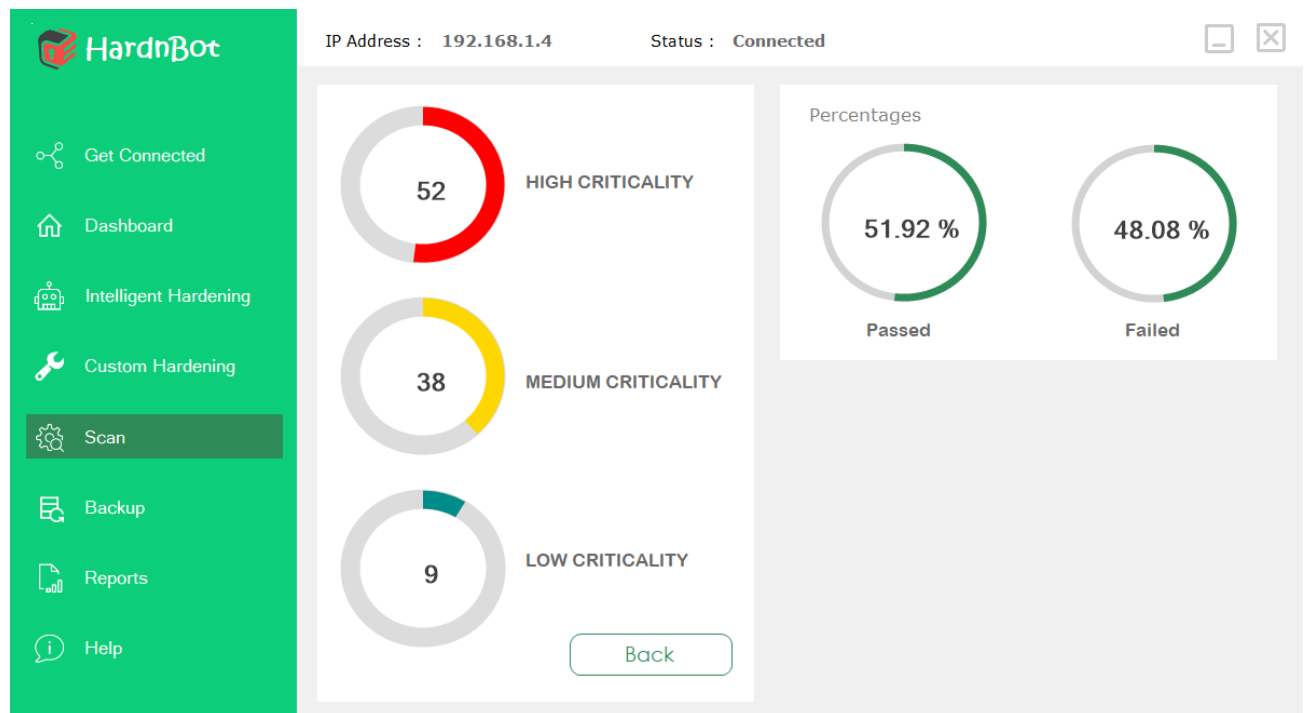


Figure 9 - Classified compliance count

Before we dive into the functionalities of “Scan” button click event, let's dive into the methods and functions defined in this scan class.

1. “*Commands ()*” function.

In this function, I implemented all the commands that need to execute when the scanning script was sent to the server via the *SendFilesToServer* class's *Send ()* function. I will dive into this function later in this document.

So, this “*Commands()*” function starts with a list of string variables assigned with some UNIX-like commands to change file permission and then to execute the scanning script and after the execution, delete the scanning script from the server.

```
String chmod = "chmod 777 HardnBot_CentOS7.sh";  
String runShell = "./HardnBot_CentOS7.sh > TestResults.txt";  
String delShell = "rm -f HardnBot_CentOS7.sh";
```

Then, *Commands ()* function will again create a separate connection thread to perform scanning tasks. This is done so the user will not get freeze or crash if the user executes command to the server using HardnBot's CLI in connection dashboard.

```

this.sshClient=new SshClient(con_dash.getIP(),Convert.ToInt32(con_dash.getPort()),con_dash.getUN(),con_dash.getPW());
this.sshClient.ConnectionInfo.Timeout = TimeSpan.FromSeconds(120);
this.sshClient.Connect();

```

The connection will obviously get successful if there are not connection limitation on the server. After the connection, **Commands ()** function will execute the commands.

```

this.sshClient.RunCommand(chmod);
this.sshClient.RunCommand(runShell);
this.sshClient.RunCommand(delShell);

```

This will change the scan script's permission, execute it append the output to a text file and finally delete the script. After all this command executions, a text file will create with the output results of the scanning script. Using this text file, **Commands ()** function will get the count of all identified results including the failed compliances count, passed compliances count, skipped compliances count, and the count of compliance which are state as "Error" during the scanning process and all this results are assigned to variables defined in the scanning class.

```

SshCommand count = this.sshClient.CreateCommand("cat TestResults.txt | wc -l");
count.Execute();
CompCount = Convert.ToInt32(count.Result.ToString());
SshCommand command = this.sshClient.CreateCommand("cat TestResults.txt | grep Fail | wc -l");
command.Execute();
FailedComp = Convert.ToInt32(command.Result.ToString());
SshCommand command1 = this.sshClient.CreateCommand("cat TestResults.txt | grep Pass | wc -l");
command1.Execute();
PassedComp = Convert.ToInt32(command1.Result.ToString());
SshCommand command2 = this.sshClient.CreateCommand("cat TestResults.txt | grep Skipped | wc -l");
command2.Execute();
SkippedComp = Convert.ToInt32(command2.Result.ToString());
SshCommand command3 = this.sshClient.CreateCommand("cat TestResults.txt | grep Error | wc -l");
command3.Execute();
ErrorComp = Convert.ToInt32(command3.Result.ToString());

```

Then, **Commands ()** function will get all the failed, skipped and error occurred compliances IDs and pass them to an array. This is to check each failed, error occurred, and skipped compliances IDs against the pre-classified dataset of compliances which has the severity levels assigned to each compliance.

The output text file of the scan will also use here, and **Commands ()** function will grep out the failed, skipped, and error occurred compliance data and get all the relevant compliance IDs of them.

Assign these three types of string data to a single variable and assign this variable to a string array called “*compz*” with the split function to split the date using whitespaces.

```
private string[] compz;

SshCommand getFail = this.sshClient.CreateCommand("cat TestResults.txt | grep Fail | cut -d' ' -f1");
SshCommand getError = this.sshClient.CreateCommand("cat TestResults.txt | grep Error | cut -d' ' -f1");
SshCommand getSkipped = this.sshClient.CreateCommand("cat TestResults.txt | grep Skipped | cut -d' ' -f1");
getFail.Execute();
getError.Execute();
getSkipped.Execute();
_outputFail = Convert.ToString(value: getFail.Result.ToString());
_outputError = Convert.ToString(value: getError.Result.ToString());
_outputSkipped = Convert.ToString(value: getSkipped.Result.ToString());
String _outputAll = _outputFail + _outputError + _outputSkipped;
compz = _outputAll.Split(); //this is the array
```

Then, **Commands ()** function will assign values to it interface elements, assign some compliance counts to variables and finally disconnect the connection thread from the server.

```
HighProgressBar.Maximum = compz.Length;
MediumProgressBar.Maximum = compz.Length;
LowProgressBar.Maximum = compz.Length;
//count
int gradedCount = FailedComp + PassedComp + SkippedComp + ErrorComp;
NoGradeComp = CompCount - gradedCount;
this.sshClient.Disconnect();
```

2. “*getResultsofCount ()*” function.

This function was developed to cross reference the pre-classified dataset and the data in the “*compz []*” array

Here I used a nested for loop to go through these two arrays.

First outer loop used to loop through the pre-classified data set whilst the inner loop checks each compliance ID from “*compz []*” elements with the outer loop’s current iteration’s element value to find any matching compliance ID.

If the inner loop finds any matching compliance IDs, the relevant counter which has the relevant criticality level assigned will increase simultaneously.

The whole method is shown in the below code segment.

```
private void getResultsofCount()
{
    for (int x = 0; x < dataset.GetLength(0); x++)
    {
        for (int i = 0; i < compz.GetLength(0); i++)
        {
            if (compz[i] == dataset[x, 0] && dataset[x, 1] == "Medium")
            {
                Medcount++;
            }
            else if (compz[i] == dataset[x, 0] && dataset[x, 1] == "High")
            {
                HighCount++;
            }
            else if (compz[i] == dataset[x, 0] && dataset[x, 1] == "Low")
            {
                LowCount++;
            }
        }
    }
}
```

(Here, the dataset is a two-dimensional array which has the compliance ID and its severity.)

So, these two are the main two important methods that I have implemented in the scan class of HardnBot. However, there is another important function that I implemented which handles the file transfers from C# application (HardnBot) to a remote Linux server. I named this class as ***“SendFilesToServer”*** and it has only one method called ***“Send”***.

```

using Renci.SshNet;
using System;
using System.IO;

namespace Sherlock
{
    class SendFilesToServer
    {
        public static void Send(string host, string username, string password, string
fileName)
        {
            using (ScpClient client = new ScpClient(host, username, password))
            {
                String Path = @".";
                client.Connect();
                client.Upload(new FileInfo(fileName), Path);
                client.Disconnect();
            }
        }
    }
}

```

To the “**Send**” function to work, it requires the remote host IPv4 address, root username, root password and the file name. the file needs to be in the root directory of the C# project (mostly in /bin/debug).

This function uses secure copy to copy files to the remote server. To use secure copy, I used “**Renci SshNet’s ScpClient**” library [12].

Here, I created a new SCP client and push the “**Send**” function’s arguments to it. (This was done to use the file transfer function anywhere in HardnBot.) Then I defined the path as “**@**.” which will be the linux server’s current directory that HardnBot’s “**Scan**” class got connected to. And, then the client connection, file upload and finally client disconnection. This class will heavily use in HardnBot’s other functions as well.

Now let us dive into the “**Scan**” button click event. When a user clicks on the “**Scan**” button it will check for any available connection via the variable assign in the connection dashboard.

```

if (con_dash.connectionstat == true)

```

If this connection status is true, HardnBot will continue its scanning by asking the user verification to the scan.

```

DialogResult result = MessageBox.Show("Connection found to " + con_dash.getIP() + "\n
Want to Proceed?", "Connection Found", MessageBoxButtons.YesNo);
if (result == DialogResult.Yes)

```

If the user clicks on “**Yes**” HardnBot will display the scanning circular animation and visible the required labels and call for the parallel thread called “**backgroundWorker1**”. I called the “**SendFilesToServer ()**” function and the “**Commands ()**” function inside this thread to prevent software freezing.

```

try
{
    pictureBox1.Visible = true;
    scanningLabel.Visible = true;
    backgroundWorker1.RunWorkerAsync();
}
catch (Exception exp)
{
    MessageBox.Show(exp.ToString());
    pictureBox1.Visible = false;
    scanningLabel.Visible = false;
}

```

The declaration of the parallel thread “*backgroundWorker1*” is as below,

```

private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
{
    SendFilesToServer.Send(con_dash.getIP(), con_dash.getUN(), con_dash.getPW(), "HardnBot
_CentOS7.sh");
    Commands();
}

```

In “*View Results*” button click event, HardnBot will check if the scan is completed otherwise it will prompt to run the scan first. If the scan is completed against the remote server, the “*getResultsofCount ()*” will call here as well as three loops to display the data in a graphical manner.

```

private void bunifuThinButton21_Click(object sender, EventArgs e)
{
    if (stat == false)
    {
        MessageBox.Show("Please run the scan first!");
    }
    else
    {
        getResultsofCount();
        outputPanel.Visible = true;
        for (int i = 1; i <= HighCount; i++)
        {
            HighProgressBar.Value = i;
            HighProgressBar.Text = i.ToString();
            HighProgressBar.Update();
        }

        for (int i = 1; i <= Medcount; i++)
        {
            MediumProgressBar.Value = i;
            MediumProgressBar.Text = i.ToString();
            MediumProgressBar.Update();
        }

        for (int i = 1; i <= LowCount; i++)
        {
            LowProgressBar.Value = i;
            LowProgressBar.Text = i.ToString();
            LowProgressBar.Update();
        }
    }
}

```

Scan script is written in shell (.sh) according to the CIS benchmark requirements and recommendations [1] . In CIS Benchmarks, they defined how to audit compliance with the audit command. In HardnBot's scan script, those commands are implemented as functions and called upon the running time of the script and store the results in a temporary file.

Run the following commands and verify the output is as indicated:

```
# modprobe -n -v freevxfs
install /bin/true
# lsmod | grep freevxfs
<No output>
```

Figure 10 – Sample CIS Benchmark recommended audit command

Below, in figure 10 is a sample HardnBot's scan script uses the CIS Benchmark recommendation to audit the service status of server audit daemon.

```
test_4.1.2() {
    id=$1
    level=$2
    description="Ensure auditd service is enabled"
    scored="Scored"
    test_start_time=$(test_start $id)

    [ $( systemctl is-enabled auditd ) == "enabled" ] && result="Pass"

    duration="$(test_finish $id $test_start_time)ms"
    write_result "$id,$result"
}
```

Figure 11 - Scan Script's audit service status checking function (ID: 4.1.2)

Just like in figure 10, there are two hundred and twenty-two (222) functions implemented for the scan script of HardnBot to perform the scanning operations.

4. TESTING AND IMPLEMENTATION

To attest the functionality and the performance of the HardnBot, some trial series have been run, and the trial results are reviewed in this section.

Experimental setup

The intelligent hardening software was run and tested using Windows 10 client, and it uses a Linux virtual machine as a jump server. The servers that were tested included Centos 7 and Red Hat 7 Linux distributions. These tested servers were virtual machines and were hosted in VMware Workstation 15.5.1. and VirtualBox 6.

Effectiveness

We first evaluated the effectiveness of scanning of compliance issues using CIS benchmarks audit rules. It gave 99% of correctness according to the compliance audit done by Nessus using the same server.

5. TEST RESULTS AND DISCUSSIONS

HardnBot's scan functions show 99% correctness and accuracy when cross-referencing a standard Nessus compliance audit report.

Furthermore, we are digging deep into compliances in-order-to classify our pre-classified data set more accurately.

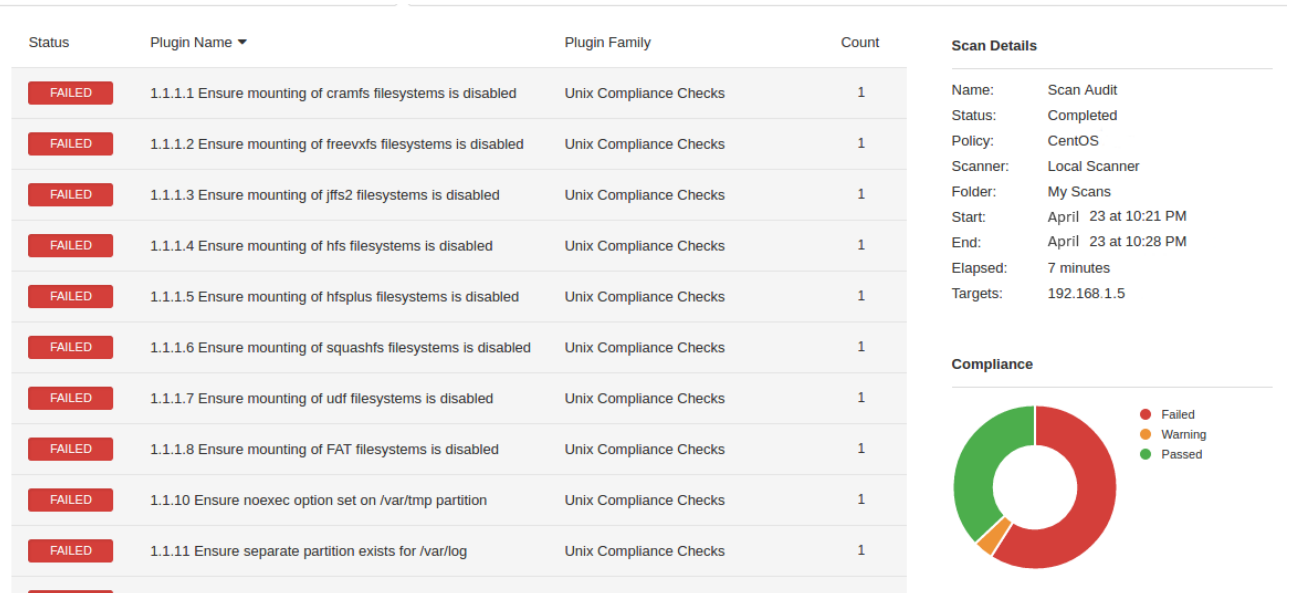


Figure 12 - Sample Nessus report

```

1.1.1.1 Fail
1.1.1.2 Fail
1.1.1.3 Fail
1.1.1.4 Fail
1.1.1.5 Fail
1.1.1.6 Fail
1.1.1.7 Fail
1.1.1.8 Fail
1.1.2 Fail
1.1.3 Fail
1.1.4 Fail
1.1.5 Fail
1.1.6 Fail
1.1.7 Fail
1.1.8 Fail
1.1.9 Fail
1.1.10 Fail
1.1.11 Fail
1.1.12 Fail

```

Figure 13 - HardnBot's Scan results

As in figure 11, we have a Nessus compliance audit report of one of our testing servers (Hosted in a hypervisor) and in figure 12 we have HardnBot's scan result of the same server. As you can see all the compliance IDs are matched with their status in both the Nessus compliance audit report and the HardnBot scan report.

In figure 11, it shows that Nessus took approximately seven minutes to scan this server. By using HardnBot's scan function the same server can be scanned within 28 seconds. However, this time may vary according to server content and policies.

```
[root@localhost ~]# ./HardnBot_CentOS7.sh > output
[00:00:02] (-) 32 of 34 tests completed grep: /boot/grub2/user.cfg: No such file or directory
[00:00:06] (\) 75 of 78 tests completed ./HardnBot_CentOS7.sh: line 1103: netstat: command not found
[00:00:28] (■) 207 of 207 tests completed
[root@localhost ~]# _
```

Figure 14 - HardnBot's scan time

Figure 13 shows the time it took to scan the server by using a custom build HardnBot's scanning script. You can see the time it took on the left side of figure 13 (highlighted in yellow).

I copy the scanning script to the target server and execute it manually to obtain the results in figure 13.

Time Nessus took (in seconds)	Time HardnBot took (in seconds)
420	< 30

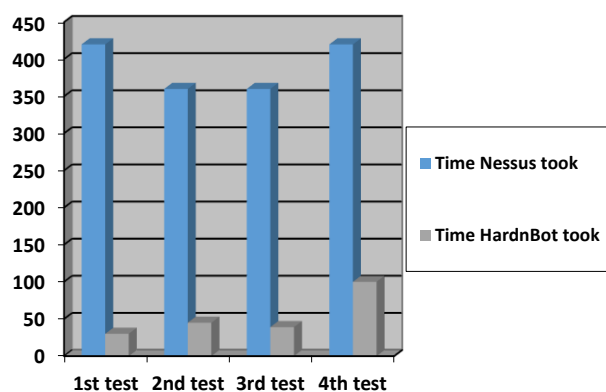


Figure 15 - Time gap

The above graph in figure 14 shows the test results of four tests ran on four servers and the time it took for both HardnBot and Nessus scanner. By analyzing these data, we can conclude that HardnBot's scan is faster than the Nessus scanner's scanning function.

However, the accuracy/correctness of HardnBot's scan results are not 100% accurate when comparing with Nessus reports but in the range of 95% - 100%.

6. CONCLUSION

In this paper, we have blended into the automation of industry server operating system security hardening via scanning a server and identifying operating system compliance failures, classifying them according to the severity levels defined by a proper and thorough search in every security, compliance in an operating system. The experiment results confirm the effectiveness of the system. And the effectiveness of the scanning function according to the Tenable Nessus reports.

7. REFERENCES

- [1] "Center for Internet Security," [Online]. Available: <https://www.cisecurity.org/cis-benchmarks>. [Accessed August 2019].
- [2] Y. T. A. R. Wita, "Vulnerability Profile for Linux," 2005. [Online]. Available: <https://ieeexplore.ieee.org/document/1423610>. [Accessed August 2019].
- [3] A. B. M. M., "An Effective Modified Security Auditing Tool (SAT)," 2001. [Online]. Available: <https://ieeexplore.ieee.org/document/937994>. [Accessed August 2019].
- [4] K. Zhao, "Design and Implementation of Secure Auditing System in Linux Kernel," 2007. [Online]. Available: <https://ieeexplore.ieee.org/document/4244819>. [Accessed August 2019].
- [5] J. Liu, "Research and Design of Security Audit System for Compliance," 2012. [Online]. Available: <https://ieeexplore.ieee.org/document/6291450>. [Accessed August 2019].
- [6] F. Yip, "Towards Robust and Adaptive Semantic-Based Compliance Auditing," 2007. [Online]. Available: <https://ieeexplore.ieee.org/document/4566971>. [Accessed August 2019].
- [7] F. Yip, "Enforcing Business Rules and Information Security Policies through Compliance Audits," 2006. [Online]. Available: <https://ieeexplore.ieee.org/document/1649214>. [Accessed August 2019].
- [8] U. Thakore, "Combining Learning and Model-Based Reasoning to Reduce Uncertainties in Cloud Security and Compliance Auditing," 2019. [Online]. Available: <https://ieeexplore.ieee.org/document/9049549>. [Accessed August 2019].
- [9] R. E. Rodriguez-Rodriguez, "Design of an Automation Model for Taking Documentary Evidence of Compliance Tests of the IT Audit," 2018. [Online]. Available: <https://ieeexplore.ieee.org/document/8587090>. [Accessed August 2019].
- [10] Nuget.org, "Renci.SshNet.Async," [Online]. Available: <https://www.nuget.org/packages/Renci.SshNet.Async>. [Accessed August 2019].
- [11] Nuget.org, "SSH.NET," [Online]. Available: <https://www.nuget.org/packages/SSH.NET>. [Accessed August 2019].
- [12] Nudoq.org, "ScpClient (Class)," [Online]. Available: <http://www.nudoq.org/#!/Packages/SSH.NET/Renci.SshNet/ScpClient>. [Accessed August 2019].