



## **Foundstone Hacme Casino v1.0™ Strategic Secure Software Training Application**

### **User and Solution Guide**

Author: Alex Smolen, Foundstone Professional Services

August 21, 2006

---

## Introduction

Foundstone Hacme Casino™ is a learning platform for secure software development and is targeted at software developers, application penetration testers, software architects, and anyone with an interest in application security. Hacme Casino is an extensible online casino platform and demonstrates the security problems that can potentially arise in these applications.

Hacme Casino is built using Ruby on Rails. Ruby on Rails (sometimes called RoR or Rails) is an open-source web application framework, built entirely in Ruby, which emphasizes adherence to the Model-View-Controller(MVC) architecture and a principle of DRY (Don't Repeat Yourself). Hacme Casino utilizes some of the basic and some of the more advanced features of the Ruby on Rails framework. It is meant to be representative of a typical Rails application, using standard features such as ActiveRecord. It also includes functionality which incorporates AJAX-style interaction, which is baked into the Rails framework, and harnesses the LoginGenerator, which is supplied by the Rails community for creating code to perform authentication in an application.

Many of the vulnerabilities in Hacme Casino cannot be detected automatically – they must be assessed by a human who has an understanding of the business context online gaming applications operate in.

Hacme Casino is offered with the full source code under the Apache Software Foundation License Version 2.0. Being fully open-source allows Hacme Casino to be subject to an unbiased peer review process, which will be used to constantly improve the quality and accuracy of the application.

**Disclaimer: Hacme Casino is riddled with vulnerabilities by design. Use of Hacme Casino can cause system compromise and Foundstone accepts no liability for any improper usage of this software. We strongly advise users not to use the application on production systems.**

## Overview of Hacme Casino Distribution

Hacme Casino is a pure Ruby on Rails web application that is distributed in 2 different formats:

- Windows Binary Installer
  - Complete installation is covered in this document
  - Download from <http://www.foundstone.com/resources/freetools.htm>
- Full Source Code
  - Not covered in this document
  - Requires installation of several development tools and intermediate Rails skills
  - Download via anonymous CVS from <http://sourceforge.net/projects/foundstone>
  - See the HOWTO document in the Hacme Casino CVS folder at sourceforge for detailed instructions

## Prerequisites

- The Windows Binary Installer only supports Microsoft Windows 2000/XP. Users of other operating systems can install Hacme Books by building the source code.
- The WEBrick server is distributed with Hacme Casino and runs by default on port 3000. Please ensure that no other applications are using this port.

## Windows Binary Installation

- Download the Hacme Casino executable installer from <http://www.foundstone.com/resources/freetools.htm>.
  - Double-click on HacmeCasinoSetup.exe.
  - **Figure 1** displays the license text, which is based on the Apache 2.0 License. You must agree to the terms of the license by clicking the **I Agree** button in order to continue installation.
  - **Figure 2** displays a security warning from Foundstone regarding Hacme Casino. Click **Next** if you understand the warning and choose to proceed.
  - **Figure 3** displays the desired installation folder. You can edit the directory path or leave the default setting. Click **Install** to begin installation of Hacme Casino.
  - **Figure 4** is the last dialog. Just click **Close**.
- Once installation is complete, you must start the Hacme Casino server (WEBrick):  
Start → Programs → Foundstone Free Tools → Hacme Casino Server START
- Wait until the WEBrick server starts (should look like **Figure 5**)
- To view Hacme Casino in Firefox\* (should look like **Figure 6**):  
Start → Programs → Foundstone Free Tools → Hacme Books
- Uninstalling Hacme Books is very easy: there is an uninstall program available from the Start Menu. Hacme Books may also be uninstalled using the Add/Remove Programs Control Panel.

\*Hint: Foundstone recommends using Firefox or other standards-compliant browsers to view Hacme Casino, as using Internet Explorer causes some display bugs.

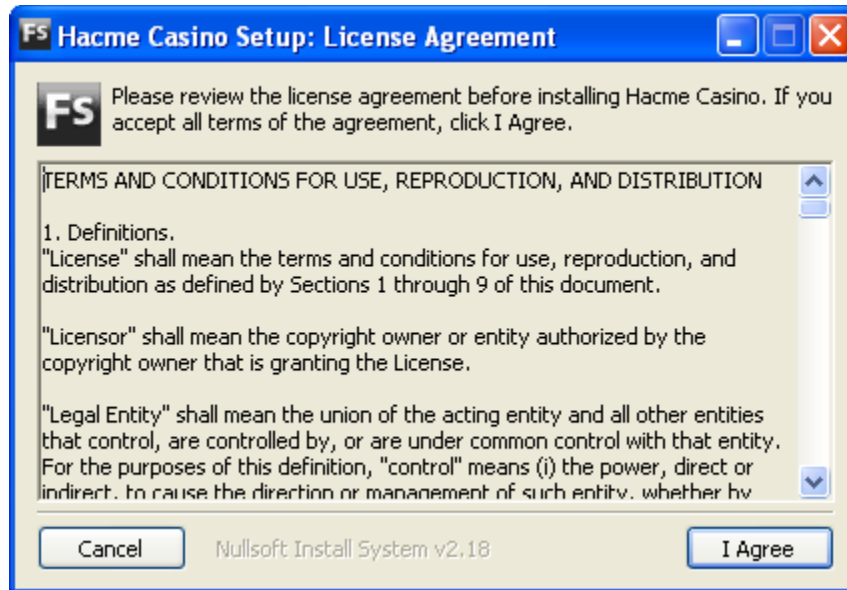


Figure 1

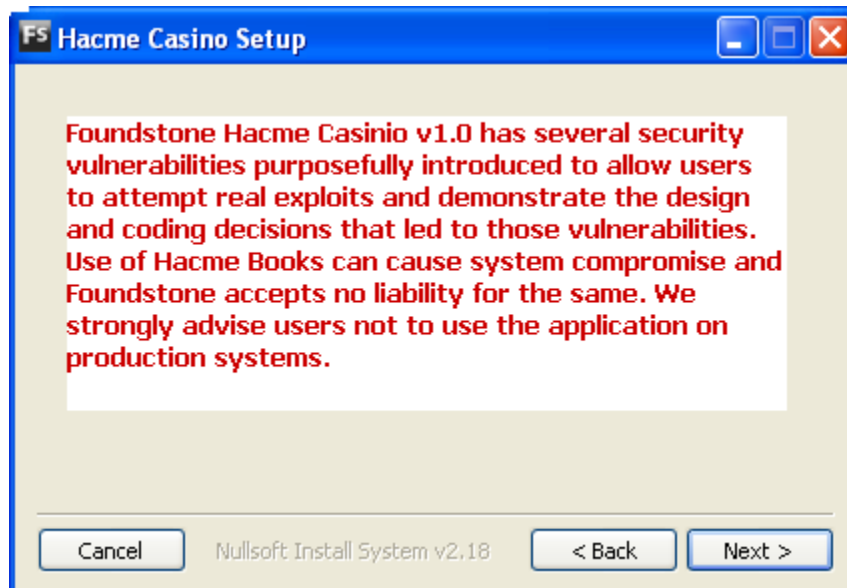


Figure 2

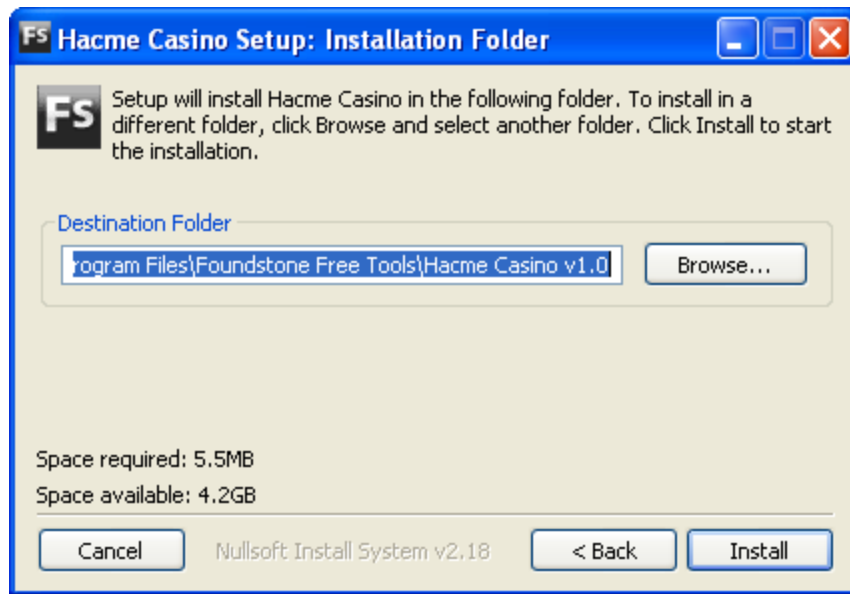


Figure 3

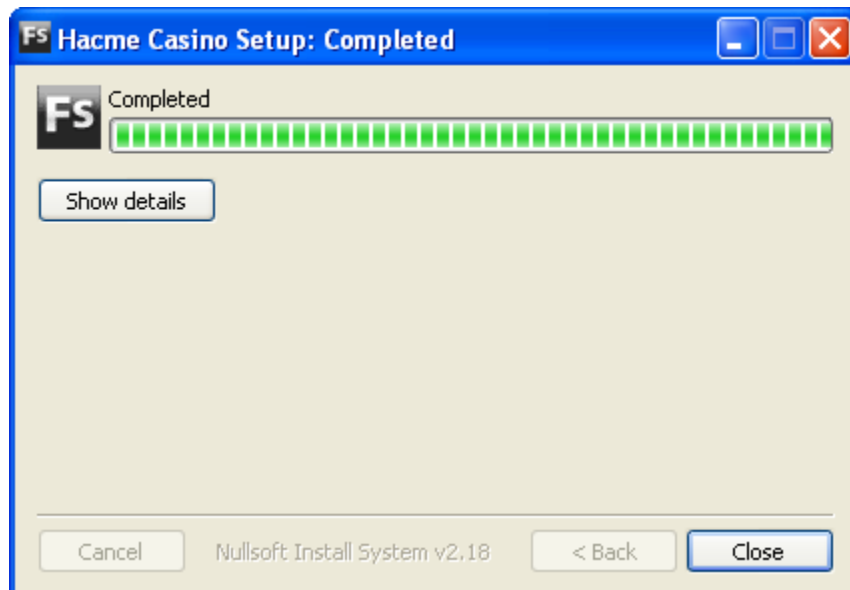


Figure 4

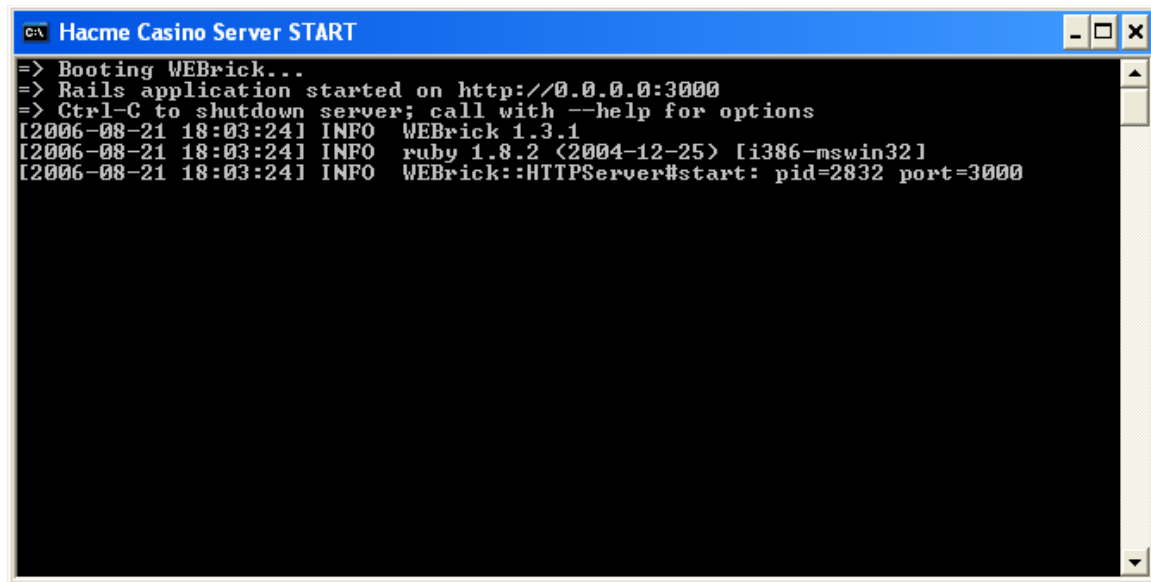


Figure 5



Figure 6

## Learning Guide

There are two fundamental approaches to web application security testing:

- Whitebox testing (AKA Code Review)
  - The tester has access to source code, configuration files, and the actual deployed application
- Blackbox testing (AKA Penetration Test or Pen-test)
  - The tester has access to the application's end-user interface only

Whitebox testing is always going to produce a more accurate result based on the fact that the source code is available. Testers are able to review data flows through the application from the presentation tier all the way through to the data access tier. Therefore, the results yielded from whitebox testing are going to be far more precise than the results gathered from blackbox testing.

For example, if there is a SQL injection vulnerability discovered in 50 different areas of a web application, a blackbox pen-tester will identify 50 vulnerabilities. However, there may be a single library that makes the database calls, which a whitebox tester can identify as one vulnerability. In addition, a whitebox review can reveal vulnerabilities in configuration and integration points. For instance, Hacme Books communicates with Hacme Bank (a similar application written in ASP.NET) to actually debit a book purchaser's bank account. A review of Hacme Books' configuration files may uncover the location of the Hacme Bank web services endpoint, which you might explore for additional vulnerabilities.

Foundstone suggests that anyone with a development background should perform a code review first, and then perform the blackbox penetration test that is described in the rest of this document. This will validate the earlier review. The source code for Hacme Casino is available via anonymous CVS at [sourceforge.net/projects/foundstone](http://sourceforge.net/projects/foundstone).

For non-developers, the blackbox test is most appropriate.

For this guide we will focus on the blackbox, or pen-test, approach. The code review methodology is a much more intensive activity that we tackle in Foundstone's *Writing Secure Code* classes. For more information about Foundstone's *Writing Secure Code* classes, go to [www.foundstone.com/education](http://www.foundstone.com/education).

## Lesson 1 – Blind SQL Injection

<b>Lesson#</b>		<b>1</b>
<b>Vulnerability Exploited</b>		Blind SQL Injection
<b>Exploit Result</b>		Authentication Bypass
<b>Input Field(s)</b>		User name
<b>Input Data</b>	<b>Step 1</b>	`
	<b>Step 2</b>	` OR 1=1--
	<b>Step 3</b>	` ) OR 1=1--
<b>Corresponding Figures(s)</b>		Figure 7 Figure 8 Figure 9

Try logging into Hacme Casino with an invalid username or an invalid password. Doesn't work, does it? That makes sense, because the whole point of the Hacme Casino authentication scheme is that you need to know a valid username and the password to get in (or we could just register, but where's the fun in that?). Notice that when we supply a bad username or password, we get a pretty generic error message about the login being unsuccessful.

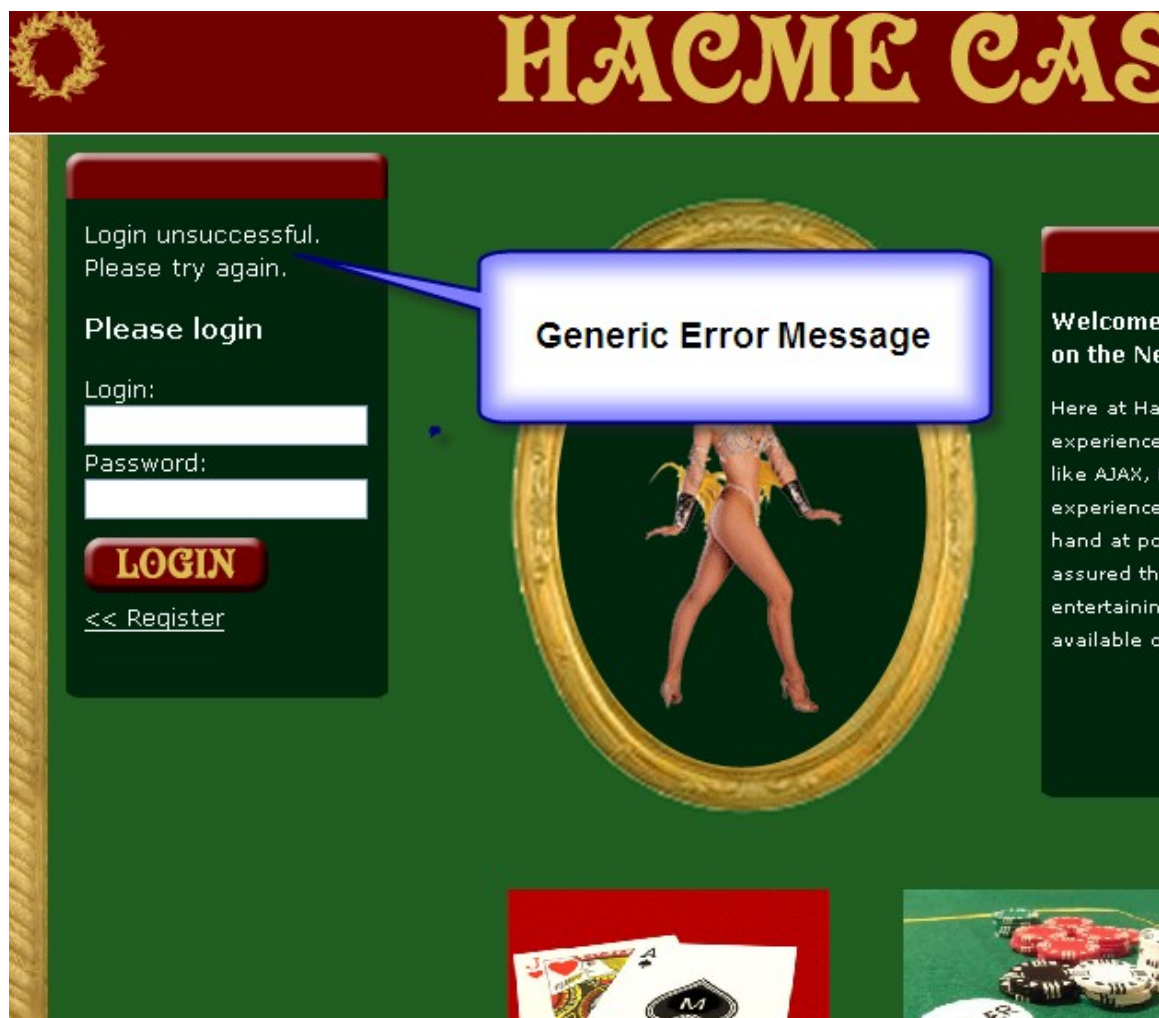


Figure 7: Generic Bad Login Message



However, as we've seen before in other Hacme applications, sometimes the login form doesn't validate input that it passes to the database. And from that, we can do SQL injection and have our way with the application. Sounds fun!

Let's try our hallmark SQL injection detection string, ' (single apostrophe), in the username field (leave the password blank).



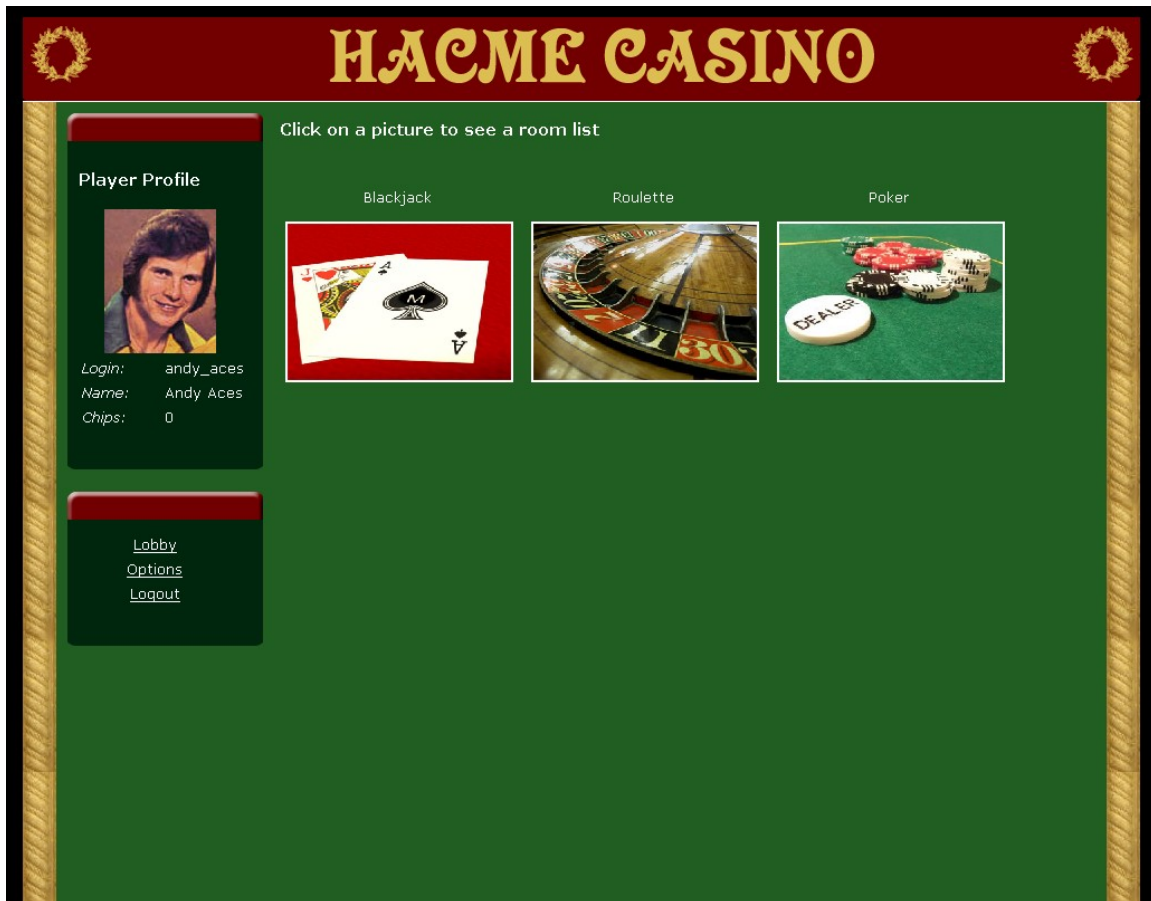
**Figure 8: Application Response to '**

Hm. So we see we got an error that is different than a simple bad login, but we didn't get a detailed SQL error message. That's not good! This is known as Blind SQL injection. We know the SQL injection is there because we can cause an error that looks different than just a normal bad login. We can infer that this single quote is causing the query to fail and the application is catching the SQL error. We can't immediately find out how to make our injection work by looking at the query in the detailed error message in the response. However, what we CAN try to do is make the query succeed, with injected data. We will then be able to exploit the query to perform our will. So, let's try an old standby that works for most login queries, '1' OR 1=1--.

We get the exact same error message as Figure 8. Still not working. What could be wrong here? An attacker might think several things:

- 1) I am not properly closing or opening parentheses.
- 2) I am querying a database that doesn't consider double dash a comment (e.g. MySQL)
- 3) There are GROUP BY, UNIONS, or other clauses after the comments that the query expects.
- 4) Something else entirely.

Not to be defeated, we test our first hypothesis by adding a right parentheses after the single quote.



**Figure 9: Application Response to ') OR 1=1--**

It worked! We're in with the Andy Aces user profile. So what happened?

In the backend, the query must have looked like

```
SELECT * FROM users WHERE (username='<username>' AND password='<password>')
```

Note the parentheses. They aren't necessary, but sometimes developers leave them in for clarity in grouping like clauses.

With our malicious input, the query became:

```
SELECT * FROM users WHERE (username='') OR 1=1--AND password='')
```

This query returns the first user, which is Andy.

---

Blind SQL injection can be tricky to exploit if the query is complicated. You may need to reopen the parentheses, such as in the case of trailing UNIONS which must be evaluated, to get the query to execute. Plenty of documentation exists on Blind SQL injection, even some automated tools can help out in exploiting these problems and retrieving entire database contents from simple holes such as this.

## Lesson 2 – Big Phish to Fry


<b>Lesson#</b>	<b>2</b>
<b>Vulnerability Exploited</b>	Cross-Site Request Forgery
<b>Exploit Result</b>	More Chips
<b>Input Field(s)</b>	URL
<b>Input Data</b>	<a href="http://localhost:3000/account/transfer_chips?transfer=1000&amp;login%5B%5D=andy_aces&amp;commit=Transfer+Chips">http://localhost:3000/account/transfer_chips?transfer=1000&amp;login%5B%5D=andy_aces&amp;commit=Transfer+Chips</a>
<b>Corresponding Figures(s)</b>	Figure 10-Figure 16

Now that we have access to a valid account, we want to get some money so that we can start gambling. Now, we could actually withdraw from our bank account\*, but that would cost us dear precious dollars! Instead, let's try to use phishing tactics to get users to get other users to give money to us.

Click on the Options link in the left sidebar. You should see the screen below.

**HAcME CASINO**

**Player Profile**



Login: andy\_aces  
Name: Andy Aces  
Chips: 0

[Lobby](#)  
[Options](#)  
[Logout](#)

**Transfer chips to another player:**

Amount:   
Recipient: Bobby Blackjack

**Cash out:**

Amount:   
Account: HacmeBank Acct No: xxxx-99

**Session Options**

Accessibility Module? (On/Off) ☐  
Show Photo? (On/Off) ☒

**Figure 10: Showing the Options Screen**

Notice that there is a functionality to transfer chips to a user. This seems awfully generous! What if we could entice users to click on a link that would cause them execute this functionality unknowingly?

Let's look at how the transfer chips function works by examining the HTML source in the figure below.

```
<div id="options-body">
  <div id="opt">

    <form action="/account/transfer_chips" method="post">
    <h3>Transfer chips to another player: </h3>
    <table>
    <tbody><tr>
      <td>
        Amount
      </td>
      <td>
        <input id="transfer" name="transfer" type="text">
      </td>
    </tr>
    <tr>
      <td>
        Recipient
      </td>
      <td>
        <select id="login_" name="login[]"><option value="bobby_blackjack">Bobby Blackjack</option>
        <option value="crystal_cardshark">Crystal Cardshark</option></select>
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <input class="submit" name="commit" value="Transfer Chips" type="submit">
      </td>
    </tr>
    </tbody></table>
    </form>

  </div>

  <div id="opt">

    <h3>Cash out: </h3>
    <form action="/account/cash_out" method="post">
    <table>
    <tbody><tr>
```

**Figure 11: Showing the HTML source of the FORM**

The HTML form shows that the action is /account/transfer\_chips. There are also two arguments, transfer and login[].

Let's get a better idea of how it works by running the traffic through Paros, an http proxy. For a detailed description of how to use the Paros Proxy, go to <http://www.parosproxy.org>.

Let's try to transfer 0 dollars to Bobby Blackjack using the web interface and intercept the traffic.

The screenshot shows the Paros Proxy interface with a captured HTTP POST request. The top pane displays the raw request details, and the bottom pane shows the parameters in a tabular view.

**Request Details:**

```

POST http://127.0.0.1:3000/account/transfer_chips HTTP/1.1
Host: 127.0.0.1:3000
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.6) Gecko/20060728 Firefox/1.5.0.6
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1:3000/account/options
Cookie: _session_id=18ab98adf249b4bc31a21ad8f1a15f2c
Content-Type: application/x-www-form-urlencoded
Content-Length: 60
  
```

**Parameters:**

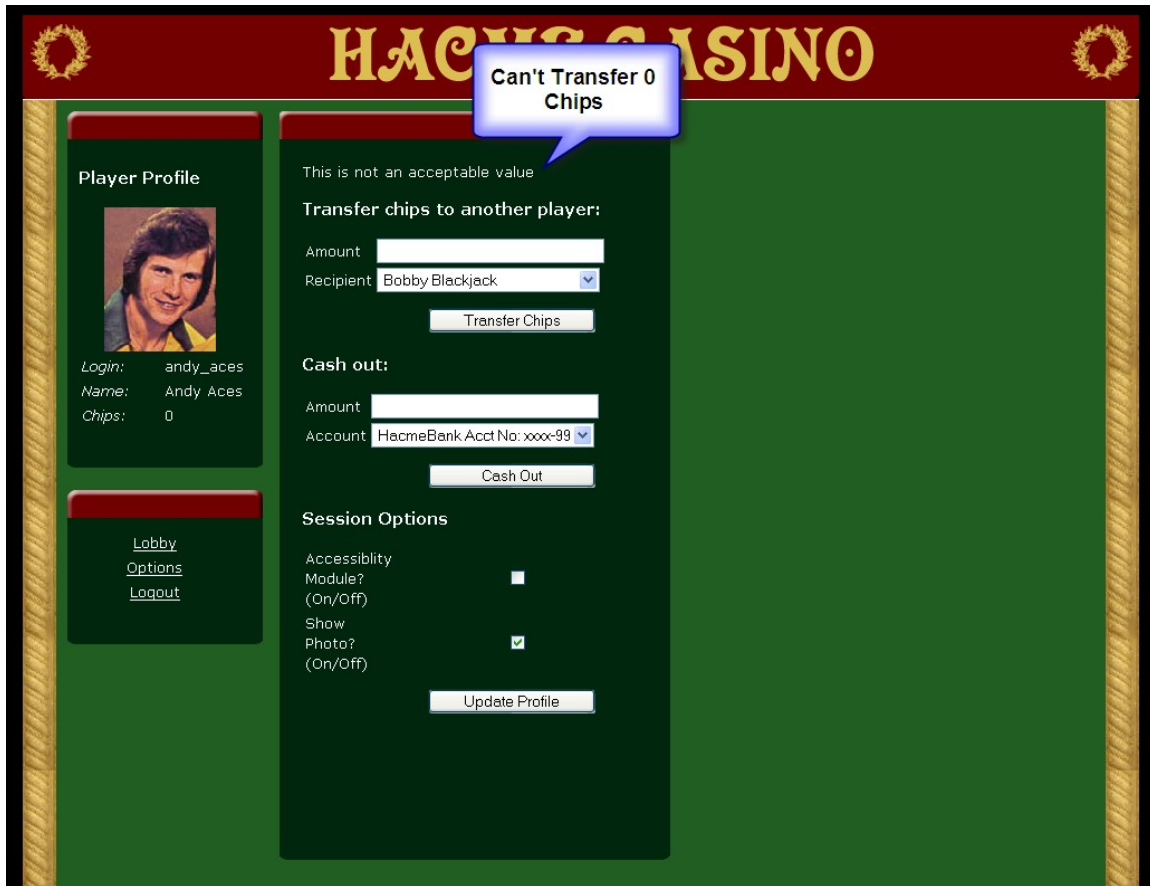
Parameter Name	Value
transfer	0
login[]	bobby_blackjack
commit	Transfer Chips

At the bottom, the interface shows "Tabular View" selected, "Trap request" checked, and "Trap response" unchecked. There are "Continue" and "Drop" buttons on the right.

**Figure 12: Show the traffic running through Paros**

We can see here that the application is sending the transfer parameter with the amount and the login argument URL. Let's add these two parameters to our URL to make a GET request which performs the same action.

**[http://localhost:3000/account/transfer\\_chips?transfer=0&login\[\]=bobby\\_blackjack&commit=Transfer+Chips](http://localhost:3000/account/transfer_chips?transfer=0&login[]=bobby_blackjack&commit=Transfer+Chips)**



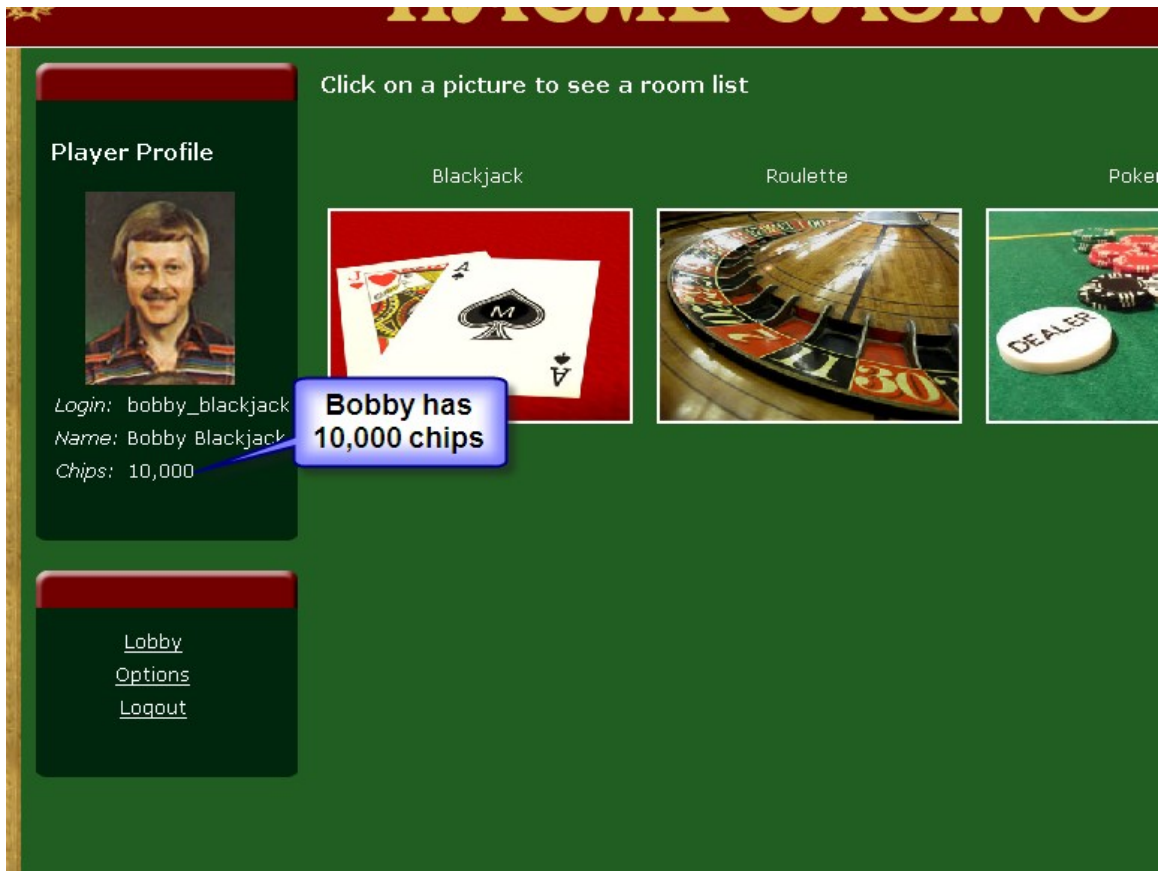
**Figure 13**

This results in an error (we can't send a non-positive number of chips). However, perhaps instead of transferring 0 dollars to bobby, let's transfer \$1000 dollars to ourselves! We will change the URL parameters appropriately:

[http://localhost:3000/account/transfer\\_chips?transfer=1000&login\[\]=andy\\_aces&commit=Transfer+Chips](http://localhost:3000/account/transfer_chips?transfer=1000&login[]=andy_aces&commit=Transfer+Chips)

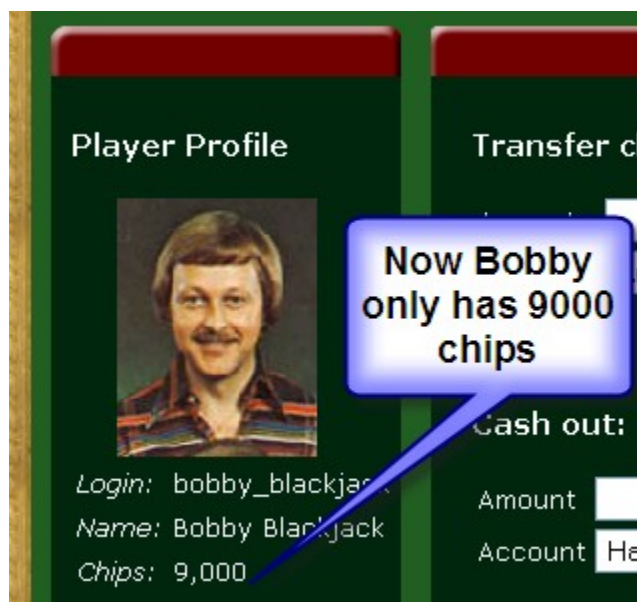
Now when we click on this, we will try to transfer \$1000 dollars to ourselves. This results in the same error message as Figure 14, because we don't have enough chips. That's not too interesting. Instead, we will email this link to other users for them to click on. Let's see what would happen if we sent this link to Bobby Blackjack and he clicked it while logged into Hacme Casino.

Log out of Hacme Casino by using the Log Out link in the left hand side bar. Log in using the username: bobby\_blackjack and the password: twenty\_one. You should see the following screen. Notice Bobby has 10,000 chips.



**Figure 14: Bobby Blackjack Home Page**

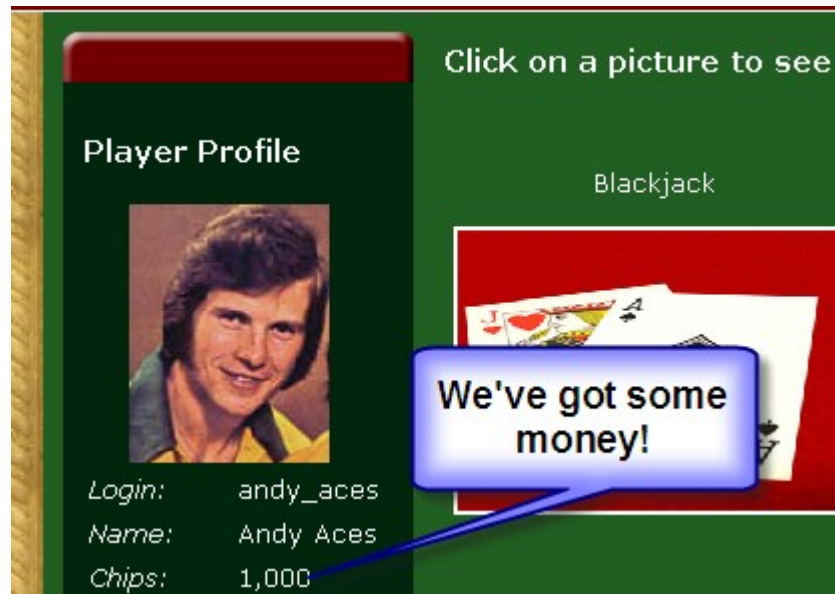
As Bobby, try clicking the link above that donates 1000 chips to Andy Aces. You should see the screen below.



**Figure 15**



Notice the chip count has been decremented. If we go and log back into our andy\_aces account (using SQL injection of course), we now have some real money to play with!



**Figure 16**

The reason why this is a security issue because we can simply send the link to several users (bobby\_blackjack included) and wait for them to click the link while logged in to Hacme Casino. That way, we don't need to know their credentials or log in as them.

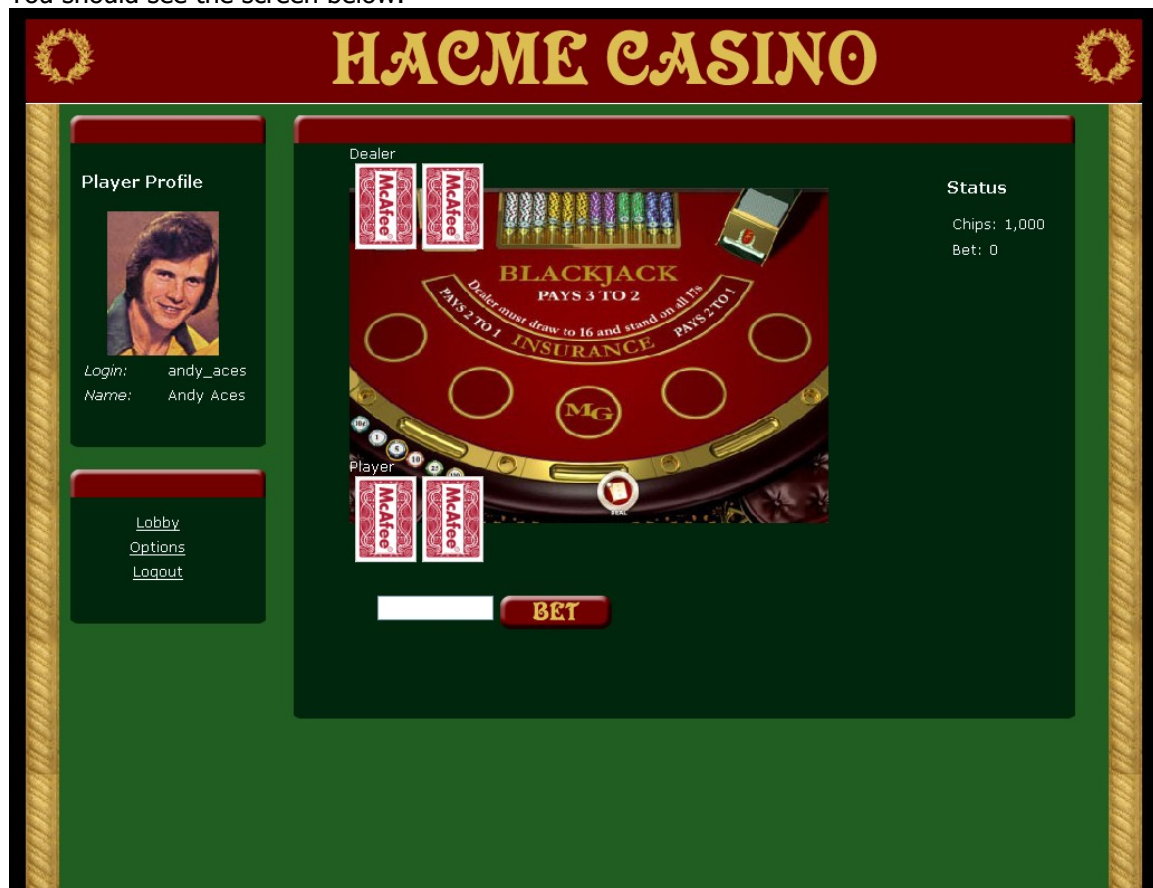
Phishers use techniques such as these to cause unsuspecting users to perform actions without being aware of it. We could send our link to several users (provided we know their email addresses) with a bogus message about winning a prize. The link appears to come from the Hacme Casino domain (even SSL would attest to this). If they click on the link while they are logged in to HacmeCasino, they give us their chips! We can sit back and watch the chips roll in!

### Lesson 3 - Know When to Walk Away

<b>Lesson#</b>	<b>3</b>
<b>Vulnerability Exploited</b>	Improper session handling
<b>Exploit Result</b>	Cheat Game/Improve Odds
<b>Input Field(s)</b>	Logout
<b>Input Data</b>	None
<b>Corresponding Figures(s)</b>	Figure 17 Figure 18

Now we're in, and we've got some playing money. We could go and gamble it all away, but being security-conscious, we don't like to take unnecessary risks. Instead, let's see if there are any ways to improve our odds at the table.

We'll try to find problems with the betting protocol. In the lobby, click on the blackjack game. You should see the screen below:



**Figure 17: Blackjack Game**

Go ahead and try a couple of rounds, but don't lose all of your money! We know that blackjack gives an edge to the casino; let's see if we can't do better.

In a casino, once you place a bet, that money is gone until the game outcome is resolved. If we pay attention to our chip total and betting amount, we can see that the amount of the bet is not

credited or debited until the game is over. That means as long as the game doesn't end, we don't lose our bet!

Let's cut our losses. Deal a few hands until you get a hand you don't like - 12-16 will do nicely. Now, instead of feebly hitting and busting, note your chip total and click the lobby link in the sidebar. See the figure below:



**Figure 18: Screen shot of logging out of the application mid-game**

Let's see if it worked – go back to the blackjack game and see if your chip total has decreased. It shouldn't of, and you are free to bet again.

Now you can really reduce your risk by getting rid of bad hands. This means you can quit your job and turn in to a full-time online blackjack player, at least until this hole is fixed!

One important thing to note here is that as a hacker, you can't get caught up trying to use the same hacks time and time again. Creativity is key, and each application has its own unique ways of being exploited.

## Lesson 4 - I'll Stay, Again

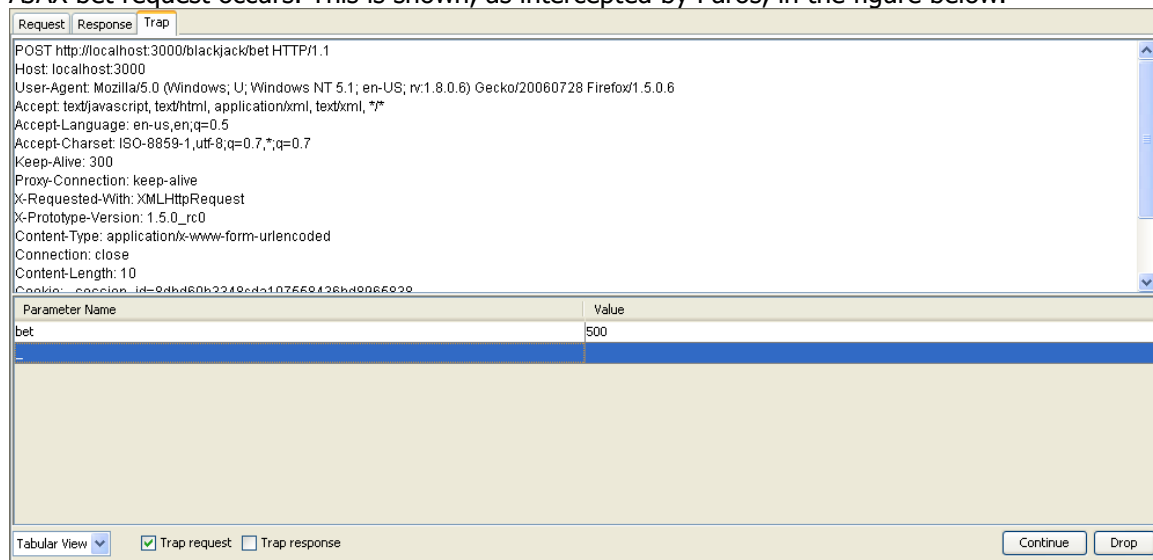
<b>Lesson#</b>	<b>4</b>
<b>Vulnerability Exploited</b>	Application Logic Vulnerability
<b>Exploit Result</b>	Cheat Game
<b>Input Field(s)</b>	URL
<b>Input Data</b>	<a href="http://localhost:3000/blackjack/hit_or_stay?act=S">http://localhost:3000/blackjack/hit_or_stay?act=S</a>
<b>Corresponding Figures(s)</b>	Figure 19 Figure 20 Figure 21

One of the most noticeable things about AJAX applications like HacmeCasino is that they are more responsive to user input than traditional web applications. This is primarily because XMLHttpRequest objects are being spawned in the background of the application, which update only the portion of the page that needs updating. Look at the login form for Hacme Casino: it creates a separate request if you select "Register" that updates only the small section of the screen. This occurs via AJAX, although it doesn't have to; plain old javascript would suffice here because there is no dynamic content. But where's the fun in that?

AJAX applications often use XmlHttpRequests when regular Web 1.0-style interactions would do. This can expose some sensitive areas of functionality. Let's look for examples of this that might be security vulnerabilities.

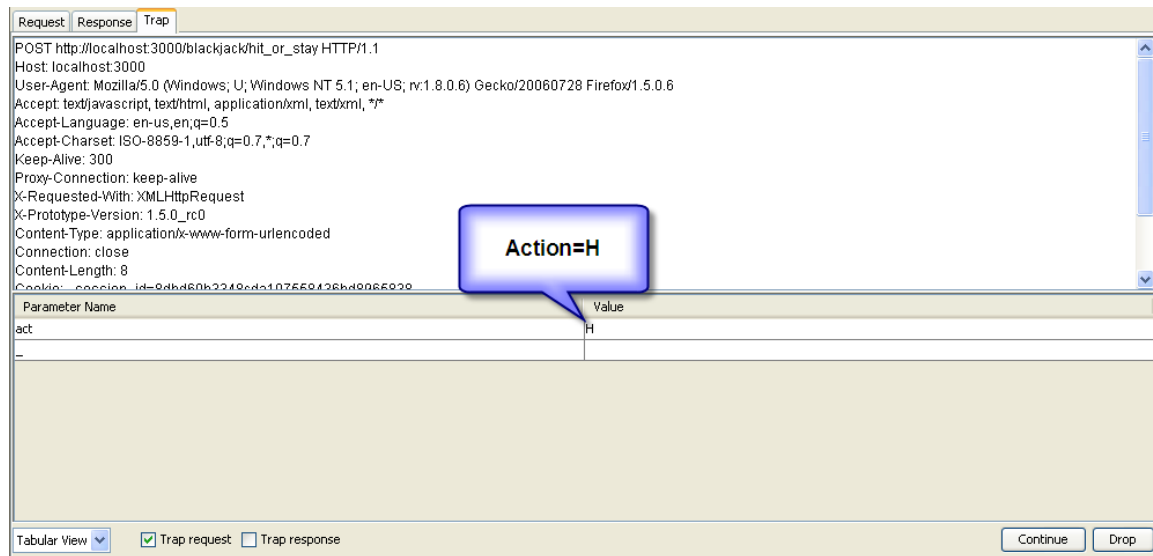
Let's go to our blackjack game. Play a few hands until you win. Winning feels good, doesn't it? I wish the feeling never went away. What would it be like if we could just keep on re-living that moment?

Take a look more deeply into how blackjack works: first, we make a bet. In the background an AJAX bet request occurs. This is shown, as intercepted by Paros, in the figure below.



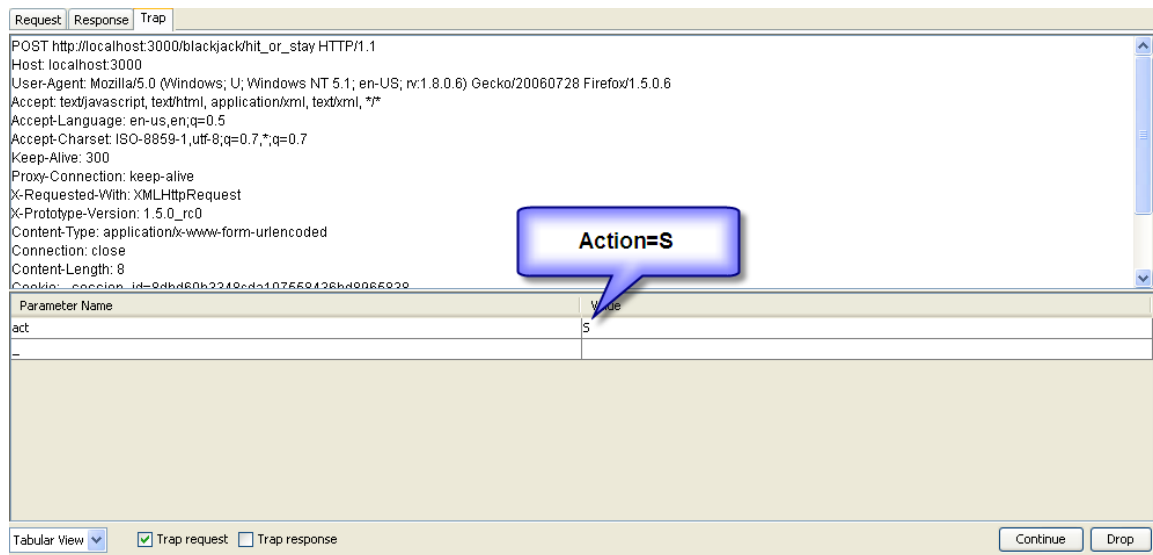
**Figure 19: Show traffic for betting**

Then, we make AJAX hit\_or\_stay request with the action parameter equal to hit until we are close to 21. See figure below.



**Figure 20: Screen shot for traffic for hit\_or\_stay**

Assuming we continue to hit and we don't bust, our next AJAX request to the server is hit\_or\_stay with the action parameter equal to stay.



**Figure 21: Screen shot for traffic for hit\_or\_stay**

The server then deals the dealer cards, calculates the winner, pays out and collects, and then asks if we want to play again.

The last AJAX request is interesting, though, because it sets off a series of events on the server that determine the payout. What if we were to issue that AJAX request again?

Turns out, this causes the server to recalculate the winner and repay the winnings. So if we have just won, we can win again by replaying the stay request!

---

Let's try it out; play a hand until you win. Now cause the traffic shown in Figure 22 to replay; typing this URL: ([http://localhost:3000/blackjack/hit\\_or\\_stay?act=S](http://localhost:3000/blackjack/hit_or_stay?act=S)) should do the trick. Notice that this causes only part of the blackjack game to render, because that AJAX request was only updating a small portion of our page. Our chips keep going up, because the server thinks we keep on winning, even though it's the same game.

Note that this attack isn't AJAX specific; it's just something that is more likely to be in an AJAX application. In typical web applications, just looking in the URL bar or an inadvertent hit of the back button will reveal these types of issues. With AJAX, it happens in the background, and is a bit more subtle. Remember, just because the request endpoint isn't directly linked from the page, it's still accessible.

Not too bad... that's more than enough chips to set sail to the Bahamas next month. But, just for fun, let's check out video poker too.

## Lesson 5 - Snoop the Deck

<b>Lesson#</b>	<b>5</b>
<b>Vulnerability Exploited</b>	Detailed Error Messages
<b>Exploit Result</b>	Cheat Game
<b>Input Field(s)</b>	URL
<b>Input Data</b>	<a href="http://localhost:3000/video_poker/test_deuces_wild">http://localhost:3000/video_poker/test_deuces_wild</a>
<b>Corresponding Figures(s)</b>	Figure 22-Figure 26

Video poker is a great game. It combines the ease of slots with the decision making of poker. Go back to the lobby, and browse to Video Poker.



Figure 22

After playing a few rounds, you might notice that Video Poker doesn't give us very good odds. What might help us here is if we could predict which cards were coming next. However, if we were to look at the cards we were dealt for a long time, we probably wouldn't notice any predictable patterns, as the cards are shuffled randomly and regularly. So let's see if we can "tilt the machine" to get some additional help.

One thing to note is that Ruby on Rails is based on routing. When you call `video_poker/show`, you are really requesting the "show" method of the "video\_poker\_controller" method to be

---

called. This is all tied together via the Rails framework. An interesting thing here is that Rails performs all this auto-magically; even if you don't have any HTML links to a method in your controller, I may still be able to call it if I can guess the name.

Ruby on Rails encourages test-driven development. Often times, left-behind test code can be a hacker's delight, giving access to resources unintentionally.

First, go to the lobby, and select Video Poker. Play a few hands to get the idea behind it. Once again, make sure to save some money so that we can cheat...ahem...improve our odds later.

Now, let's try to find any test methods in the video\_poker\_controller. We can't see what these are from the client-side, so we'll have to guess.

Try:

[http://localhost:3000/video\\_poker/test](http://localhost:3000/video_poker/test)  
[http://localhost:3000/video\\_poker/test\\_video\\_poker](http://localhost:3000/video_poker/test_video_poker)  
[http://localhost:3000/video\\_poker/test\\_hand](http://localhost:3000/video_poker/test_hand)  
...  
[http://localhost:3000/video\\_poker/test\\_deuces\\_wild](http://localhost:3000/video_poker/test_deuces_wild)

Wait a second, what was that last one? We found an opening! We ARE feeling lucky. This may seem unrealistic; however, this entire process of finding un-advertised methods in controllers could be automated.



What was the response to the request to this unadvertised method?

## NoMethodError in Video pokerController#test\_deuces\_wild

```
undefined method `set_deuces_wild' for #<VideoPokerGame:0x5250698>
```

RAILS\_ROOT: C:/DOCUME~1/asmolen/LOCALS~1/Temp/tar2rubyscript.d.2824.1/HacmeCasino/script/./config/..

[Application Trace](#) | [Framework Trace](#) | [Full Trace](#)

```
#(RAILS_ROOT)/app/controllers/video_poker_controller.rb:33:in `test_deuces_wild'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/lib/webrick/httpserver.rb:104:in `service'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/lib/webrick/httpserver.rb:65:in `run'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/lib/webrick/server.rb:155:in `start_thread'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/lib/webrick/server.rb:144:in `start'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/lib/webrick/server.rb:144:in `start_thread'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/lib/webrick/server.rb:94:in `start'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/lib/webrick/server.rb:89:in `each'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/lib/webrick/server.rb:89:in `start'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/lib/webrick/server.rb:79:in `start'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/lib/webrick/server.rb:79:in `start'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/app/HacmeCasino.rb:608:in `load'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/app/HacmeCasino.rb:608
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/app/HacmeCasino.rb:603:in `newlocation'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/app/HacmeCasino.rb:559:in `newlocation'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/app/HacmeCasino.rb:487:in `newlocation'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/app/HacmeCasino.rb:487:in `newlocation'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/app/HacmeCasino.rb:559:in `newlocation'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/app/HacmeCasino.rb:603
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/bootstrap.rb:45:in `load'
C:/DOCUME~1/asmolen/LOCALS~1/Temp/eee.HacmeCasino.exe.9/bootstrap.rb:45
```

### Request

Parameters: None

[Show session dump](#)

### Response

Headers: {"cookie"=>[], "Cache-Control"=>"no-cache"}

**Figure 23: Screen shot of error message**

Uh oh, someone forgot to turn their detailed error messages off! Let's look for goodies. First, go back to video poker and make a bet.

**Player Profile**  
Login: andy\_aces  
Name: Andy Aces

[Lobby](#)  
[Options](#)  
[Logout](#)

Hand	1 Credit	2 Credit	3 Credit	4 Credit	5 Credit
Royal Flush	250	500	750	1000	4000
Straight Flush	50	100	150	200	250
Four of a Kind	25	50	75	100	125
Full House	9	18	27	36	45
Flush	6	12	18	24	30
Straight	4	8	12	16	20
Three of a Kind	3	6	9	12	15
Two Pair	2	4	6	8	10
Jacks or Better	1	2	3	4	5

Click on the cards you want to hold.

**Status**  
Chips: 3,015  
Current Credits: 5 Credits

**Figure 24: Our hand**

Then, enter the URL that causes the verbose error message (in a separate browser tab to avoid losing this session!). Look at the session dump tree (note: your session will look different, because the deck and hand are in a different state):

```

user: 6id001 !ruby/object:User
  attributes:
    id: "1"
    chips: 3015
    first_name: Andy
    login: andy_aces
    password: 2c4alc243aca3390d5a8e87fef7c727d09af977c
    last_name: Aces
    current_bet: 5
  hand: !ruby/object:VideoPokerHand
    cards:
      - !ruby/object:Card
        rank: 9
        suit: 1
      - !ruby/object:Card
        rank: 8
        suit: 1
      - !ruby/object:Card
        rank: 11
        suit: 3
      - !ruby/object:Card
        rank: 4
        suit: 1
      - !ruby/object:Card
        rank: 12
        suit: 2
    hold:
      - false
      - false
      - false
      - false
      - false
  vpgame: !ruby/object:VideoPokerGame
    dealer: !ruby/object:VideoPokerDealer
    deck: !ruby/object:VideoPokerDeck
      cards:
        - !ruby/object:Card
          rank: 0
          suit: 0
        - !ruby/object:Card
          rank: 6
          suit: 0
        - !ruby/object:Card
          rank: 5

```

**Figure 25: Session Dump Tree**

Hm, looking at this data, we can see what's in our hand (user->hand->cards), and what's in the deck (vpgame->dealer->deck)! We're not quite sure what the integer codes for suit and rank mean, though. Let's do some analysis.

Look at what the session says is in our hand:

	Suit	Actual Suit	Rank	Actual Rank
Card One	1	D	9	J
Card Two	1	D	8	10
Card Three	3	C	11	K
Card Four	1	D	4	6
Card Five	2	H	12	A

Judging from this, it looks like the following encoding is occurring:

Suit

Code	Actual Value
0	Spades
1	Diamonds
2	Hearts
3	Clubs

Rank

Code	Actual Value
0	2
1	3
2	4
3	5
4	6
5	7
6	8
7	9
8	10
9	J
10	Q
11	K
12	A

Knowing this, we can figure out the next five cards in the deck (at the bottom). They are:

Next Card 1: (8,2) -> 10 of Hearts

Next Card 2: (7,1) -> 9 of Diamonds

Next Card 3: (4,2) -> 6 of Hearts

Next Card 4: (5,3) -> 7 of Clubs

Next Card 5: (5,2) -> 7 of Hearts

If we were to only keep the six and the ten in our hand, we would get two pair. Go back to the game and try it (although adjusting for your specific hand/deck).

**HACHE CASINO**

**Player Profile**



Login: andy\_aces  
Name: Andy Aces

[Lobby](#)  
[Options](#)  
[Logout](#)

Hand	1 Credit	2 Credit	3 Credit	4 Credit	5 Credit
Royal Flush	250	500	750	1000	4000
Straight Flush	50	100	150	200	250
Four of a Kind	25	50	75	100	125
Full House	9	18	27	36	45
Flush	6	12	18	24	30
Straight	4	8	12	16	20
Three of a Kind	3	6	9	12	15
Two Pair	2	4	6	8	10
Jacks or Better	1	2	3	4	5

**Status**

Chips: 4,015  
Current Credits: 0 Credits

**Two Pair**  
**1,000**

**NEW GAME**

Figure 26

---

And there we are. What an impossibly good decision! Let's also note that this could be kind of attack could be automated to speed up our results.

Sometimes this hack will help, such as in the case above, and sometimes it won't because there may be no way to get a winning hand.

One thing to note is that Ruby on Rails by default has a very verbose error message in debug mode. In production mode, this kind of information is limited. But it wouldn't surprise most hackers to find a debug system in production!

That's it, we've had enough....let's cash out. Once you have made 100,000 chips with , cash out to your special account, number 111-1111-111 (this requires a bit of hacking using a proxy). You will see a special message letting you know you've won!

---

## About Foundstone Professional Services

Foundstone Professional Services, a division of McAfee, offers a unique combination of services and education to help organizations continuously and measurably protect the most important assets from the most critical threats. Through a strategic approach to security, Foundstone identifies, recommends, and implements the right balance of technology, people, and process to manage digital risk and leverage security investments more effectively.

Foundstone's Strategic Secure Software Initiative (S3i™) services help organizations design and engineer secure software. By building in security throughout the Software Development Lifecycle, organizations can significantly reduce their risk of malicious attacks and minimize costly remediation efforts. Services include:

- Source Code Audits
- Software Design and Architecture Reviews
- Threat Modeling
- Web Application Penetration Testing
- Software Security Metrics and Measurement

For more information about Foundstone S3i services, go to [www.foundstone.com/s3i](http://www.foundstone.com/s3i).

Foundstone S3i training is designed to teach programmers and application developers how to build secure software and to write secure code. Classes include:

- [Building Secure Software](#)
- [Writing Secure Code – Java \(J2EE\)](#)
- [Writing Secure Code – ASP.NET \(C#\)](#)
- [Ultimate Web Hacking](#)

For the latest course schedule, go to [www.foundstone.com/education](http://www.foundstone.com/education).

---

## Acknowledgements

Many individuals at Foundstone contributed to the development and testing of Hacme Casino and the production of this whitepaper. Alex Smolen was the primary contributor and author of Hacme Casino 1.0. In addition, the rest of the "con" group helped review the project and prepare it for release. Thanks guys!