# Fortify Security Report

Aug 19, 2015

SMARTSOL 20

# Executive Summary

## Issues Overview

On Aug 16, 2015, a source code review was performed over the empresario_banorte code base. 259 files, 17,126 LOC (Executable) were scanned and reviewed for defects that could lead to potential security vulnerabilities. A total of 244 reviewed findings were uncovered during the analysis.

## Issues by Fortify Priority Order

Refined by: [fortify priority order]:critical OR [fortify priority order]:high

| | |
|---|---|
| High (3 Hidden) | 3 |
| Critical | 1 |

## Recommendations and Conclusions

The Issues Category section provides Fortify recommendations for addressing issues at a generic level.  The recommendations for specific fixes can be extrapolated from those generic recommendations by the development group.

# Project Summary

## Code Base Summary

Code location: C:/Users/SMARTSOL 20/Desktop/Back_up_BANORTE/empresario_banorte

Number of Files: 259

Lines of Code: 17126

Build Label: <No Build Label>

## Scan Information

Scan time: 16:34

SCA Engine version: 6.30.0086

Machine Name: SSLAP20

Username running scan: SMARTSOL 20

## Results Certification

Results Certification Valid

Details:

Results Signature:

SCA Analysis Results has Valid signature

Rules Signature:

rules/BanorteSSLRule.xml has Valid signature

## Attack Surface

Attack Surface:

Private Information:

null.null.null

## Filter Set Summary

Current Enabled Filter Set:

Quick View

Filter Set Details:

Folder Filters:

If [fortify priority order] contains critical Then set folder to Critical

If [fortify priority order] contains high Then set folder to High

If [fortify priority order] contains medium Then set folder to Medium

If [fortify priority order] contains low Then set folder to Low

Visibility Filters:

If impact is not in range [2.5, 5.0] Then hide issue

If likelihood is not in range (1.0, 5.0] Then hide issue

## Audit Guide Summary

Audit guide not enabled

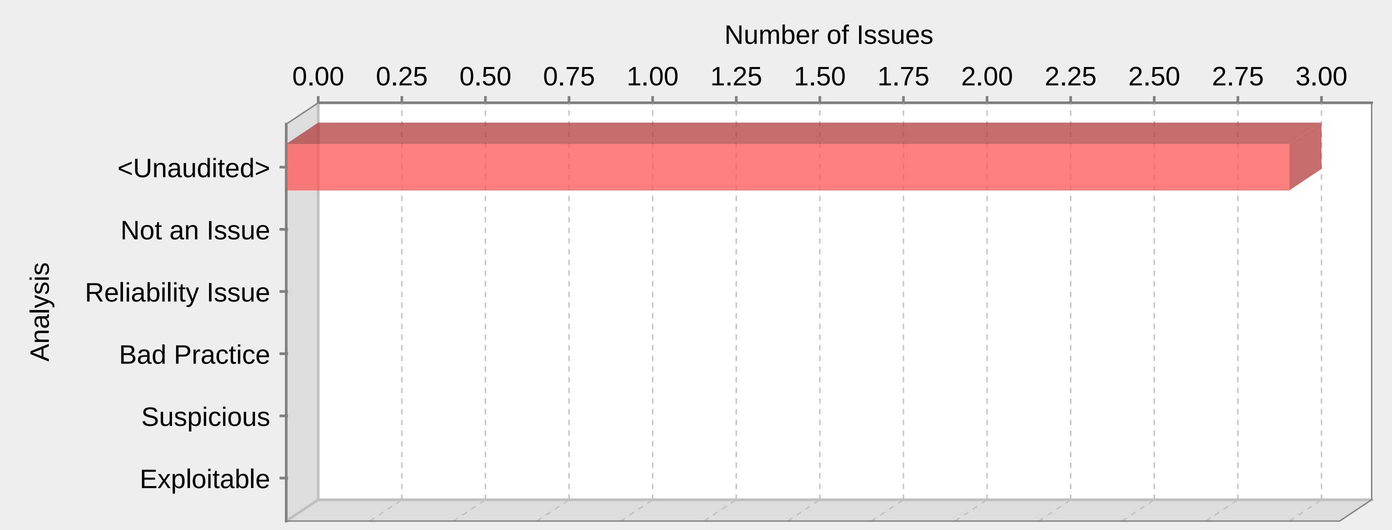
# Results Outline

## Overall number of results

The scan found 244 issues.

## Vulnerability Examples by Category

### Category: Insecure Randomness (3 Issues: 3 Hidden)



**Explanation:**

Insecure randomness errors occur when a function that can produce predictable values is used as a source of randomness in security-sensitive context.

Computers are deterministic machines, and as such are unable to produce true randomness. Pseudo-Random Number Generators (PRNGs) approximate randomness algorithmically, starting with a seed from which subsequent values are calculated.

There are two types of PRNGs: statistical and cryptographic. Statistical PRNGs provide useful statistical properties, but their output is highly predictable and forms an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure, it must be impossible or highly improbable for an attacker to distinguish between it and a truly random value. In general, if a PRNG algorithm is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts, where its use can lead to serious vulnerabilities such as easy-to-guess temporary passwords, predictable cryptographic keys, session hijacking, and DNS spoofing.

Example: The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

```
function genReceiptURL (baseURL){
var randNum = Math.random();
var receiptURL = baseURL + randNum + ".html";
return receiptURL;
}
```

This code uses the Math.random() function to generate "unique" identifiers for the receipt pages it generates. Because Math.random() is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

**Recommendations:**

When unpredictability is critical, as is the case with most security-sensitive uses of randomness, use a cryptographic PRNG. Regardless of the PRNG you choose, always use a value with sufficient entropy to seed the algorithm. (Values such as the current time offer only negligible entropy and should not be used.)

In JavaScript, the typical recommendation is to use the window.crypto.random() function in the Mozilla API. However, this method does not work in many browsers, including more recent versions of Mozilla Firefox. There is currently no cross-browser solution for a robust cryptographic PRNG. In the meantime, consider handling any PRNG functionality outside of JavaScript.

### jquery-1.6.4.min1.js, line 2 (Insecure Randomness) [Hidden]

| | | | |
|---|---|---|---|
| **Fortify Priority:** | High | **Folder** | High |
| **Kingdom:** | Security Features | | |

**Abstract:** Standard pseudo-random number generators cannot withstand cryptographic attacks.

**Sink:** jquery-1.6.4.min1.js:2 FunctionPointerCall()

```
0    /*! jQuery v1.6.4 http://jquery.com/ | http://jquery.org/license */
1    (function(a,b){function cu(a){return
     f.isWindow(a)?a:a.nodeType===9?a.defaultView||a.parentWindow:!1}function
     cr(a){if(!cg[a]){var
     b=c.body,d=f("<"+a+">").appendTo(b),e=d.css("display");d.remove();if(e==="none"||e===""){ch||(ch=c.createElement("iframe"),ch.frameBorder=ch.width=ch.height=0),b.appendChild(ch);if(!ci||!ch.createElement)ci=(ch.contentWindow||ch.contentDocument).document,ci.write((c.compatMode==="CSS1Compat"?"<!doctype
     html>":"")+"<html><body>"),ci.close();d=ci.createElement(a),ci.bo...
2    t[h]}if(f.isEmptyObject(t)){var u=s.handle;u&&(u.elem=null),delete s.events,delete
     s.handle,f.isEmptyObject(s)&&f.removeData(a,b,!0)}}},customEvent:{getData:!0,setData:!0,changeData:!0},trigger:function(c,d,e,g){var
     h=c.type||c,i=[],j;h.indexOf("!")>=0&&(h=h.slice(0,-1),j=!0),h.indexOf(".")>=0&&(i=h.split("."),h=i.shift(),i.sort());if(!!e&&!f.event.customEvent[h]||!!f.event.global[h]){c=typeof c=="object"?c[f.expando]?c:new
     f.Event(h,c):new f.Event(h),c.type=h,c.exclusive=j,c.namespace=i.join("....
3    (a,i,e,d)),g&&(f.fragments[a[0]]=h?e:1);return{fragment:e,cacheable:g}},f.fragments={},f.each({appendTo:"append",prependTo:"prepend",insertBefore:"before",insertAfter:"after",replaceAll:"replaceWith"},function(a,b){f.fn[a]=function(c){var
     d=[],e=f(c),g=this.length===1&&this[0].parentNode;if(g&&g.nodeType===11&&g.childNodes.length===1&&e.length===1){e[b](this[0]);return this}for(var
     h=0,i=e.length;h<i;h++){var
     j=(h>0?this.clone(!0):this).get();f(e[h])[b](j),d=d.concat(j)}return
     this.pushStack(d,a...
```

## scriptaculous190packer.js, line 1 (Insecure Randomness) [Hidden]

| | | | |
|---|---|---|---|
| **Fortify Priority:** | High | **Folder** | High |
| **Kingdom:** | Security Features | | |

**Abstract:** Standard pseudo-random number generators cannot withstand cryptographic attacks.

**Sink:** scriptaculous190packer.js:1 FunctionPointerCall()

```
1    var Scriptaculous={Version:"1.9.0",require:function(b){try{document.write('<script
     type="text/javascript" src="'+b+'"><\/script>')}catch(c){var
     a=document.createElement("script");a.type="text/javascript";a.src=b;document.getElementsByTagName("head")[0].appendChild(a)}},REQUIRED_PROTOTYPE:"1.6.0.3",load:function(){function a(c){var d=c.replace(/_.*|\./g,"");d=parseInt(d+"0".times(4-d.length));return
     c.indexOf("_")>-1?d-1:d}if((typeof Prototype=="undefined")||(typeof
     Element=="undefined")||(typeof...
```

## prototype1700packer.js, line 1 (Insecure Randomness) [Hidden]

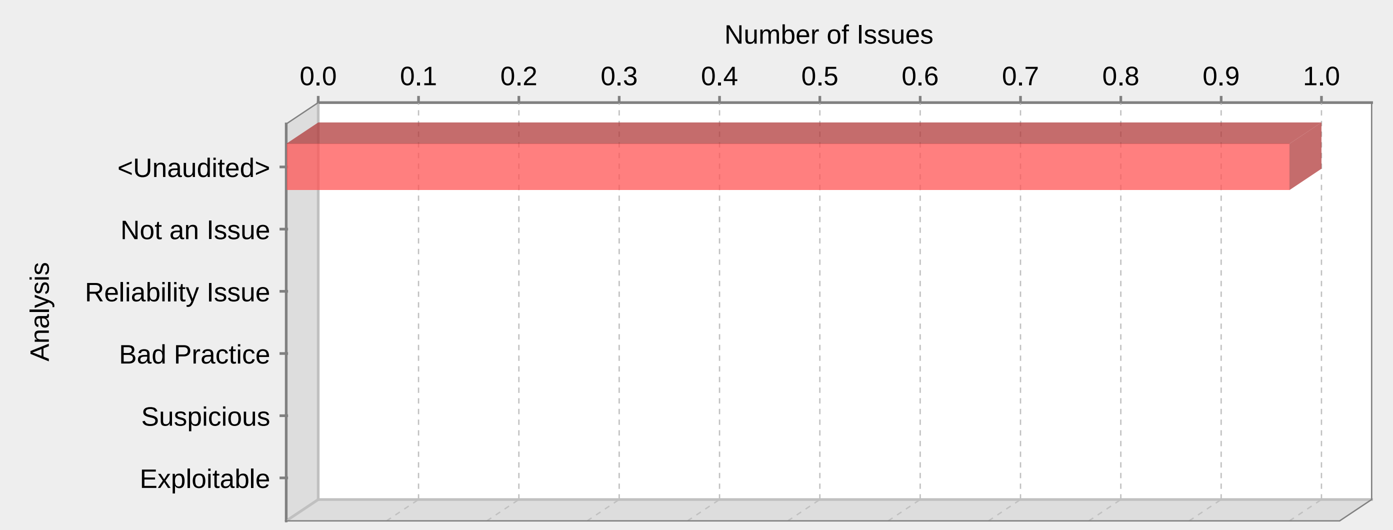
| | | | |
|---|---|---|---|
| **Fortify Priority:** | High | **Folder** | High |
| **Kingdom:** | Security Features | | |

**Abstract:** Standard pseudo-random number generators cannot withstand cryptographic attacks.

**Sink:** prototype1700packer.js:1 FunctionPointerCall()

```
-1   var Prototype={Version:"1.7",Browser:(function(){var b=navigator.userAgent;var
     a=Object.prototype.toString.call(window.opera)=="[object
     Opera]";return{IE:!!window.attachEvent&&!a,Opera:a,WebKit:b.indexOf("AppleWebKit/")>-1,Gecko:b.indexOf("Gecko")>-1&&b.indexOf("KHTML")===-1,MobileSafari:/Apple.*Mobile/.test(b)}})(),BrowserFeatures:{XPath:!!document.evaluate,SelectorsAPI:!!document.querySelector,ElementExtensions:(function(){var
     a=window.Element||window.HTMLElement;return !!(a&&a.prototype)})(),...
0    /*!
1     * Sizzle CSS Selector Engine - v1.0
```

## Category: Dynamic Code Evaluation: Code Injection (1 Issues)



### Explanation:

Many modern programming languages allow dynamic interpretation of source instructions. This capability allows programmers to perform dynamic instructions based on input received from the user. Code injection vulnerabilities occur when the programmer incorrectly assumes that instructions supplied directly from the user will perform only innocent operations, such as performing simple calculations on active user objects or otherwise modifying the user's state. However, without proper validation, a user might specify operations the programmer does not intend.

Example: In this classic code injection example, the application implements a basic calculator that allows the user to specify commands for execution.

```
...
userOp = form.operation.value;
calcResult = eval(userOp);
...
```

The program behaves correctly when the operation parameter is a benign value, such as "8 + 7 * 2", in which case the calcResult variable is assigned a value of 22. However, if an attacker specifies languages operations that are both valid and malicious, those operations would be executed with the full privilege of the parent process. Such attacks are even more dangerous when the underlying language provides access to system resources or allows execution of system commands. In the case of JavaScript, the attacker can utilize this vulnerability to perform a cross-site scripting attack.

### Recommendations:

Avoid dynamic code interpretation whenever possible. If your program's functionality requires code to be interpreted dynamically, the likelihood of attack can be minimized by constraining the code your program will execute dynamically as much as possible, limiting it to an application- and context-specific subset of the base programming language.

If dynamic code execution is required, unvalidated user input should never be directly executed and interpreted by the application. Instead, a level of indirection should be introduced: create a list of legitimate operations and data objects that users are allowed to specify, and only allow users to select from the list. With this approach, input provided by users is never executed directly.

## prototype1700packer.js, line 1 (Dynamic Code Evaluation: Code Injection)

| | | | |
|---|---|---|---|
| **Fortify Priority:** | Critical | **Folder** | Critical |
| **Kingdom:** | Input Validation and Representation | | |

**Abstract:** The file prototype1700packer.js interprets unvalidated user input as source code on line 1. Interpreting user-controlled instructions at run-time can allow attackers to execute malicious code.

**Source:** prototype1700packer.js:1 Read this.transport.responseText()

```
-1   var Prototype={Version:"1.7",Browser:(function(){var b=navigator.userAgent;var
     a=Object.prototype.toString.call(window.opera)=="[object
     Opera]";return{IE:!!window.attachEvent&&!a,Opera:a,WebKit:b.indexOf("AppleWebKit/")>-
     1,Gecko:b.indexOf("Gecko")>-1&&b.indexOf("KHTML")===-
     1,MobileSafari:/Apple.*Mobile/.test(b)}})(),BrowserFeatures:{XPath:!!document.evaluate
     ,SelectorsAPI:!!document.querySelector,ElementExtensions:(function(){var
     a=window.Element||window.HTMLElement;return !!(a&&a.prototype)})(),...
0    /*!
1     * Sizzle CSS Selector Engine - v1.0
```

**Sink:** prototype1700packer.js:1 eval()

```
-1                     var Prototype={Version:"1.7",Browser:(function(){var b=navigator.userAgent;var
                       a=Object.prototype.toString.call(window.opera)=="[object
                       Opera]";return{IE:!!window.attachEvent&&!a,Opera:a,WebKit:b.indexOf("AppleWebKit/")>-
                       1,Gecko:b.indexOf("Gecko")>-1&&b.indexOf("KHTML")===-
                       1,MobileSafari:/Apple.*Mobile/.test(b)}})(),BrowserFeatures:{XPath:!!document.evaluate
                       ,SelectorsAPI:!!document.querySelector,ElementExtensions:(function(){var
                       a=window.Element||window.HTMLElement;return !!(a&&a.prototype)})(),...
0                      /*!
1                       * Sizzle CSS Selector Engine - v1.0
```