SecureLayer7
Time and Again, Securing you

Pentest Report – KeyStoneJS 09-2017

KeystoneJS

https://github.com/keystonejs/keystone

Index

Introduction

"**KeystoneJS** is a powerful Node.js content management system and web app framework built on express and mongoose. Keystone makes it easy to create sophisticated web sites and apps, and comes with a beautiful auto-generated Admin UI." - As per **KeystoneJS** website

This penetration test was carried out by **SecureLayer7** team. It was carried out for 2 days. The test identified 3 high vulnerabilities along with 2 medium and 1 low vulnerabilities.

Scope

- Source Code available on Github
- Locally Hosted Web Application

Testing environment

- Windows 7
- npm 3.10.10
- node.js 6.11.2

Identified Vulnerabilities

The following sections lists the identified vulnerabilities. The findings are listed by their degree of severity. The severity is given in brackets following the title heading. Each bug is given a unique identifier for the purpose of future reference and follow-up.

SecureLayer7
Time and Again, Securing you

SL7_KEYJS_01: CSV Excel Macro Injection (High)

Vulnerability Description:

Instance 1:

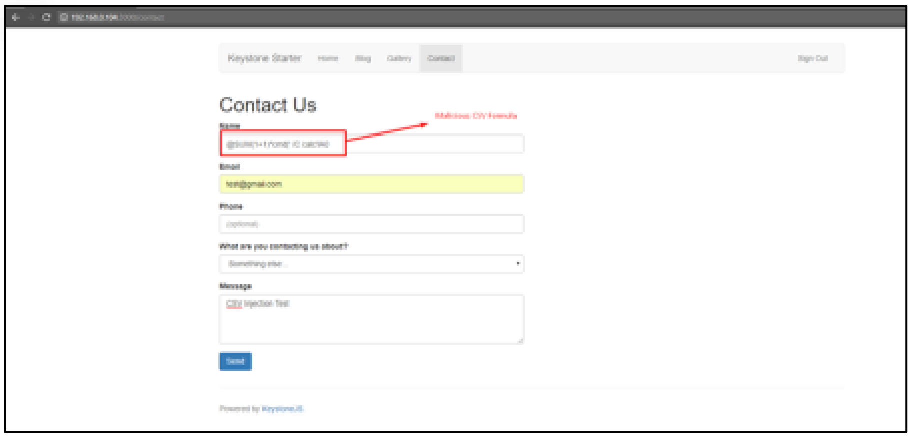
An Unauthenticated user sends the malicious payload via contact us page. There are no input validation or proper sanitization of user input Therefore when admin tries to download the record as csv the input is directly sent to the excel, leading to Remote Code Execution.

Affected Component: Name Field in Contact Us form accessible by unauthenticated users.

PoC /Steps to Recreate:

1. Go to Contact Us page and insert the below payload in the Name Field.

   Payload: @SUM(1+1)*cmd|' /C calc'!A0



Contact Us

Name: @SUM(1+1)*cmd|' /C calc'!A0

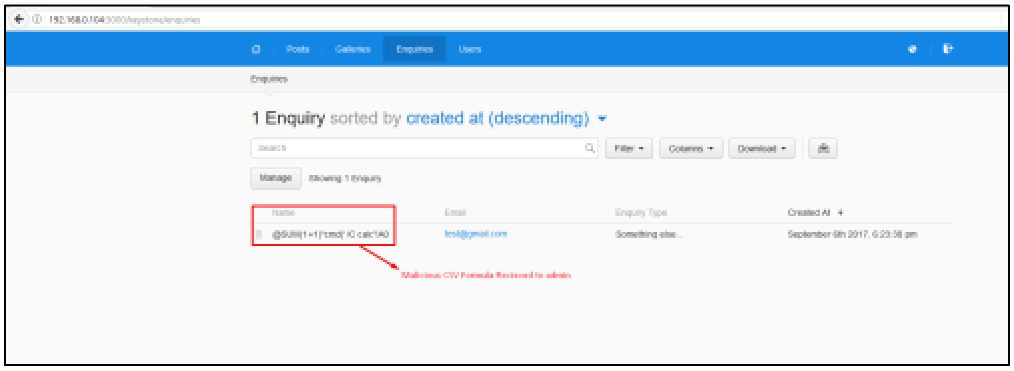
Email: test@gmail.com

Phone: (optional)

What are you contacting us about?: Something else...

Message: CSV Injection Test

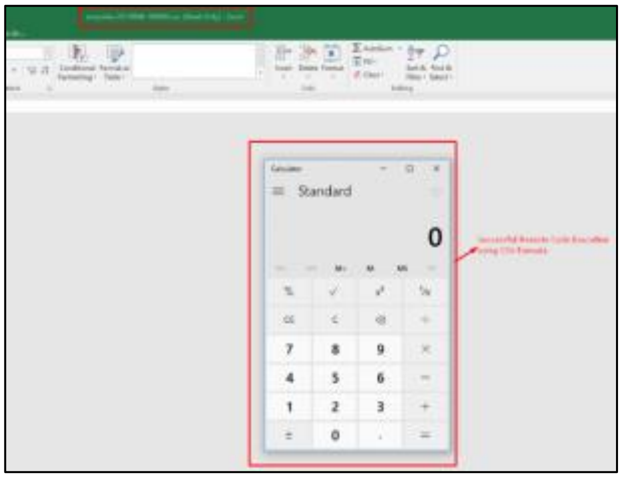
2. Now Navigate to Enquiries page and check the entered payload.



1 Enquiry sorted by created at (descending)

Name	Email	Enquiry Type	Created At
@SUM(1+1)*cmd|' /C calc'!A0	test@gmail.com	Something else...	September 8th 2017, 6:23:38 pm

3. Download as .csv, once done open it in excel and observe that calculator application gets open.



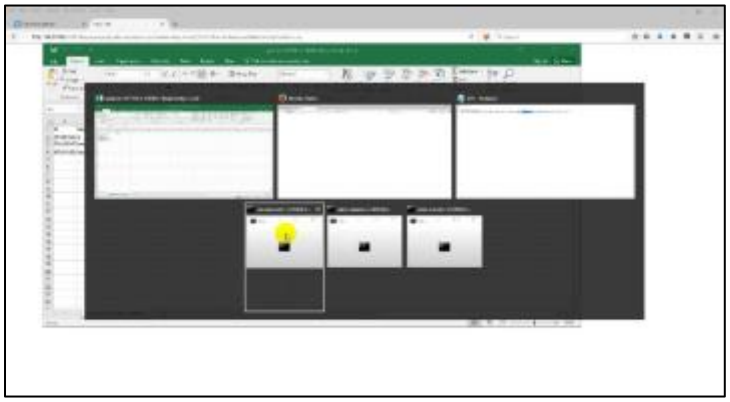
Instance 2:

An attacker send the malicious URL with his payload which asks victim to download the csv. There is no input validation for what information is sent in the excel and directly dumps the data from URL into the excel which leads to remote code execution.

Affected Component: Download functionality.

PoC /Steps to Recreate:

1. Login to the KeyStone application and copy paste the below URL in new tab.
   URL:
   http://[ip:port]/keystone/api/galleries/export.csv?select=key,=cmd|'/S'!A1&sort=&expandRelationshipFields=true
2. Now open the csv file in excel and observe that command prompt is executed.



Mitigation: It is a good practice not to trust user inputs and always encode the output. Also, for the successful execution of the formula, attacker will have to use the '-', '=','+','@' and the pipe (|) is used to execute the binary in the excel software. Hence, it is strongly recommended to prefix the mentioned character with " ' "(quote) which prevents the execution of payload.

SL7_KEYJS_02: Stored Cross Site Scripting(High)

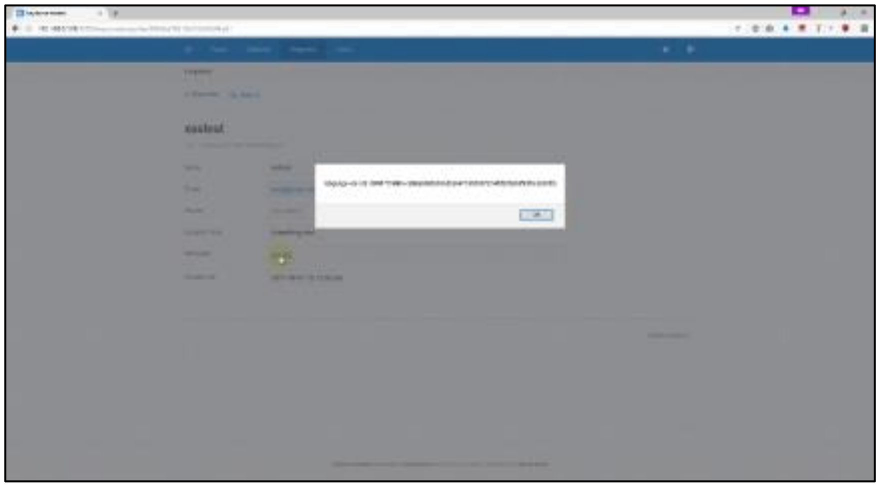
Description: The web application allows users to contact to the administrators for any queries. An attacker can leverage this functionality by sending malicious request in the message field leading to stored XSS which allows the attacker to take over the admin account.

PoC/Steps to recreate:

1. Navigate to Contact Us page(as a normal user).
2. Fill in the details needed and enter the below payload in message field and send.

   <a onmouseover=alert(document.cookie)>XSS link</a>

3. Now login as admin and navigate to the above new record created in the enquiries.
4. Move the cursor on the text "XSS link" and observe the pop-up.



Mitigation: The application accepts input from normal user without any validation and renders it without output encoding. Therefore it is recommended to perform input validation or html output encoding to avoid such kind of attacks.

SL7_KEYJS_03: Application wide CSRF Bypass (High)

Description: The web application fails to validate for CSRF protection if the attacker removes the CSRF parameter (header) and its value all together. Using this technique an attacker can bypass CSRF protection and carry out unintended actions on behalf of victim users for every action/end point.

PoC:

In a sample PoC, we are creating a user using the above bypass. A video PoC is also made to demonstrate the same. PFA.

```
<html>
  <body>
    <form action="http://127.0.0.1:3000/keystone/api/users/create" method="POST" enctype="multipart/form-data">
      <input type="hidden" name="name&#46;first" value="Saurabh" />
      <input type="hidden" name="name&#46;last" value="Banawar" />
      <input type="hidden" name="email" value="saurabh&#46;banawar&#64;securelayer7&#46;net" />
      <input type="hidden" name="password" value="test" />
      <input type="hidden" name="password&#95;confirm" value="test" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Mitigation: In order to resolve this, ensure that the presence of header ***x-csrf-token*** is validated first. If it is not present then simply reject the request.

SL7_KEYJS_04: Logical flaw that does not delete pics from third party website (Medium)

Description: The web application uses a third party service called cloudinary to save images. The link for those images is like this:

http://res.cloudinary.com/keystone-demo/image/upload/c_fit,f_auto,h_450,w_750/v1505376367/bkq3kklsmscmvh0eydky.jpg

If a blog is in published status then it is fine to allow access to users to this URL. But if the blog in in Draft status, then it means all of the information of the blog is private and should not be allowed to access. But the Keystone application fails to communicate this to Cloudinary application which allows unrestricted access to the images when the blog is not in published status.

PoC/Steps to recreate:

1. Create a blog with an image in it.
2. Publish it.
3. View the blog as an unauthenticated user.
4. Copy the link address of the image and SAVE it in notepad. It will look like: http://res.cloudinary.com/keystone-demo/image/upload/c_fit,f_auto,h_450,w_750/v1505376367/bkq3kklsmscmvh0eydky.jpg
5. Now login as admin and unpublish the same blog.
6. Now as an unauthenticated user try to access the saved link.
7. Observe that you get access to private image.

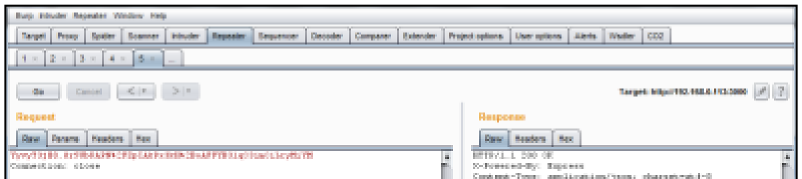
Mitigation: In order to resolve this, Keystone application must communicate the status of the blog to Cloudinary application and then Cloudinary application must apply access controls on the image links if the blog is not in published status.

SL7_KEYJS_05: Stored Cross Site Scripting(Medium)

Description: The web application allows admin to create posts. The content brief and content extended field are vulnerable to stored cross site scripting. Allowing keystone admins to execute malicious script on client's browser.

PoC/Steps to recreate:

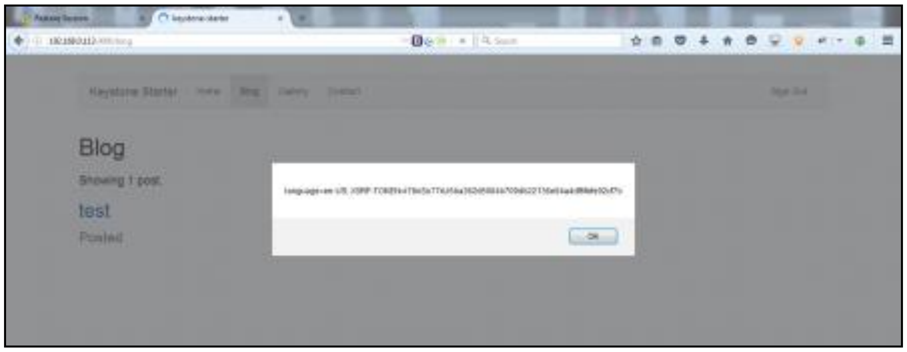
5. Log into Keystone application and navigate to create blog post.
6. Enter the title for blog post and proceed.
7. Fill in the details for the blog and use the intercept (burp in this case) to capture the request before clicking on save.



```
[illegible]
Connection: close
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
[illegible]
```

8. As shown in above figure send the payload *<script>alert(document.cookie)</script>* in the request for content.brief .
9. Now logout and navigate to the newly created blog post as normal user and observer the script getting executed.



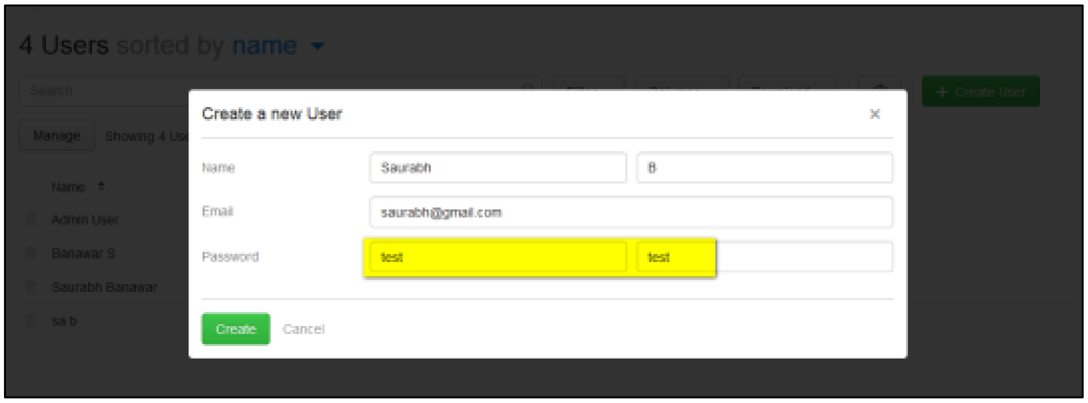
Mitigation: The application allows to have multiple admin which can make untrusted admin to exploit stored XSS. Therefore it is recommended to perform input validation or html output encoding to avoid such kind of attacks.

SL7_KEYJS_06: Weak Password Policy (Low)

Description: When a new user is added, the application does not enforce a password policy resulting in users created having weak passwords. This is not a good practice. Although it is not a major vulnerability, but letting users use weak passwords make the job of an attacker easier because the passwords are then very easy to guess and cracked with less efforts.

PoC:

1. Login as admin and go to create new user page (link: http://127.0.0.1:3000/keystone/users)
2. Click on CREATE USER button
3. Enter valid details in all the fields
4. Enter password as test, 1234.



4 Users sorted by name

Create a new User

Name: Saurabh | B

Email: saurabh@gmail.com

Password: test | test

5. Click on SAVE
6. Observe that the application accepts it

Mitigation: Implement server side check that validate whether entered password is of min 6 characters.

Conclusion

Implementing security guidelines while developing a product is a must. This includes but not limited to enforcing authorization and authentication checks, input validation among other security measures both at source code and UI level. As a security measure, it is important to ensure high findings such as XSS be patched as soon as possible which in absence can lead to complete user takeover, credential leakage and application compromise.