SpotWeb Vulnerable to Multiple Unauthenticated blind stored XSS.

SpotWeb GitHub Link: https://github.com/spotweb/spotweb
Version: 1.4.9 (Git Commit 9e53a03)
Reported Date: 12 Apr, 2021
Severity: High
CVSS Score: CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

What is XSS?

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.

What is Stored XSS?

Stored XSS (also known as persistent or second-order XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

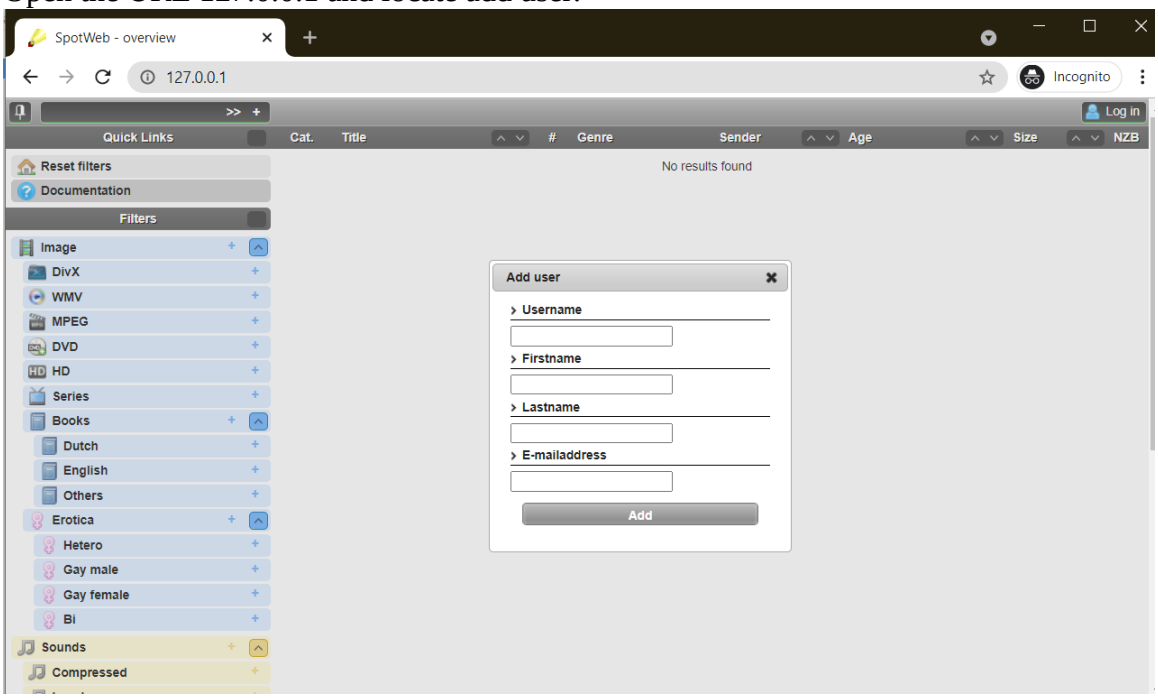
The data in question might be submitted to the application via HTTP requests; for example, comments on a blog post, user nicknames in a chat room, or contact details on a customer order. In other cases, the data might arrive from other untrusted sources; for example, a webmail application displaying messages received over SMTP, a marketing application displaying social media posts, or a network monitoring application displaying packet data from network traffic.

What is Blind XSS?

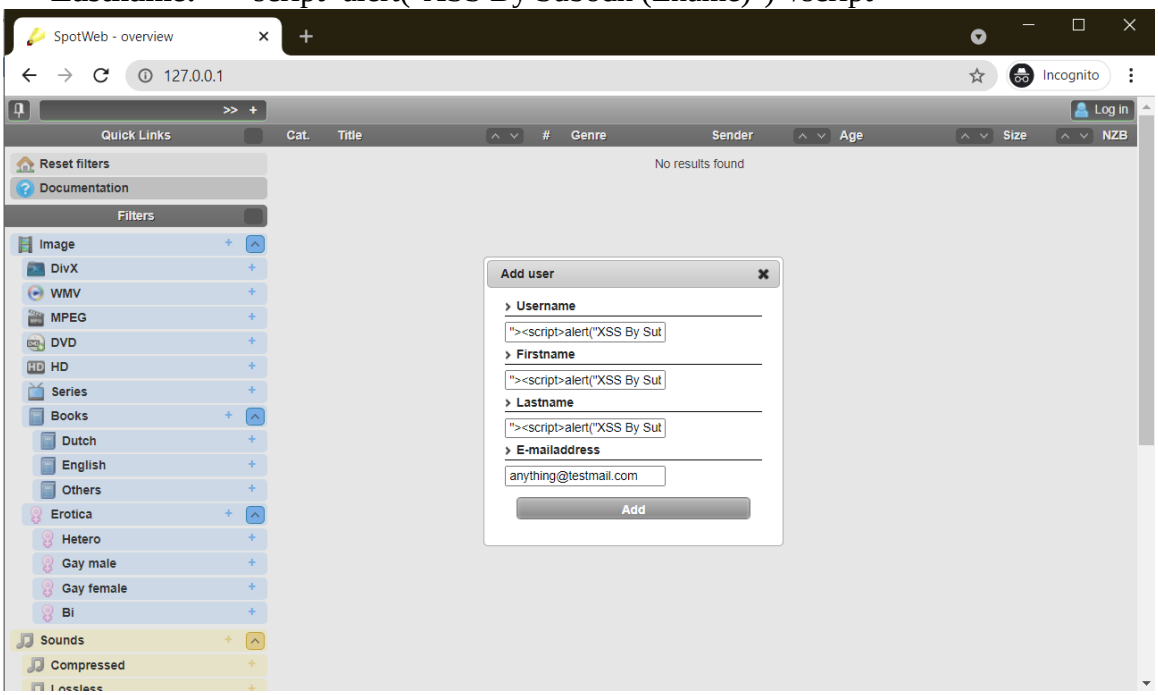
Blind XSS vulnerabilities are a variant of persistent XSS vulnerabilities. They occur when the attacker input is saved by the web server and executed as a malicious script in another part of the application or in another application. For example, an attacker injects a malicious payload into a contact/feedback page and when the administrator of the application is reviewing the feedback entries the attacker’s payload will be loaded. The attacker input can be executed in a completely different application (for example an internal application where the administrator reviews the access logs or the application exceptions).

Steps to reproduce:

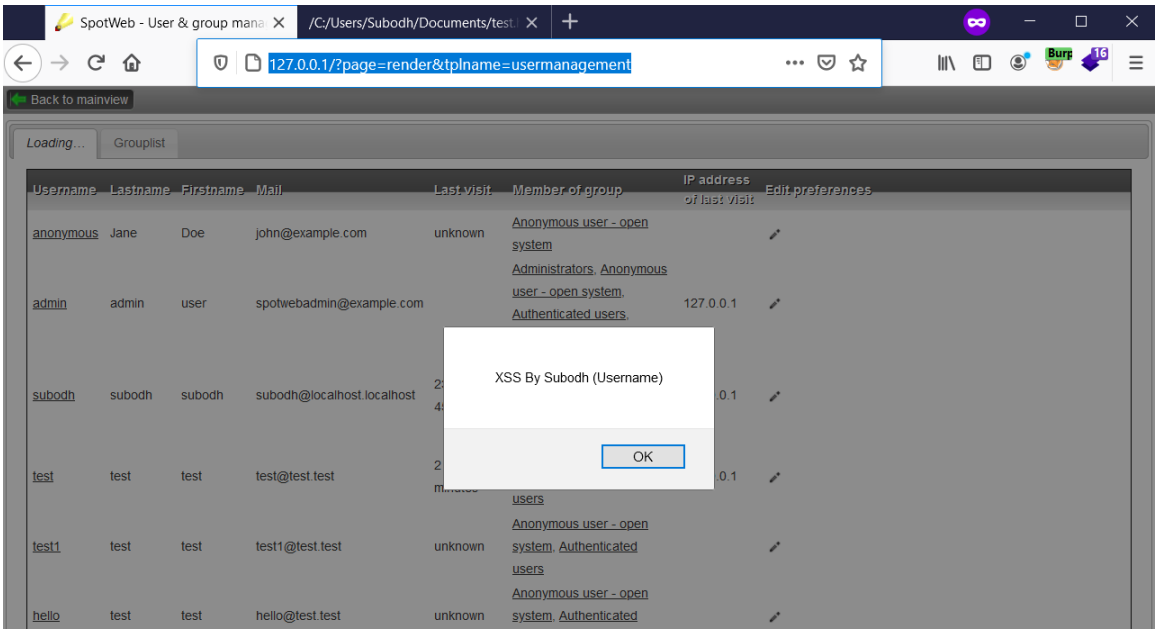
1. Open the URL 127.0.0.1 and locate add user.

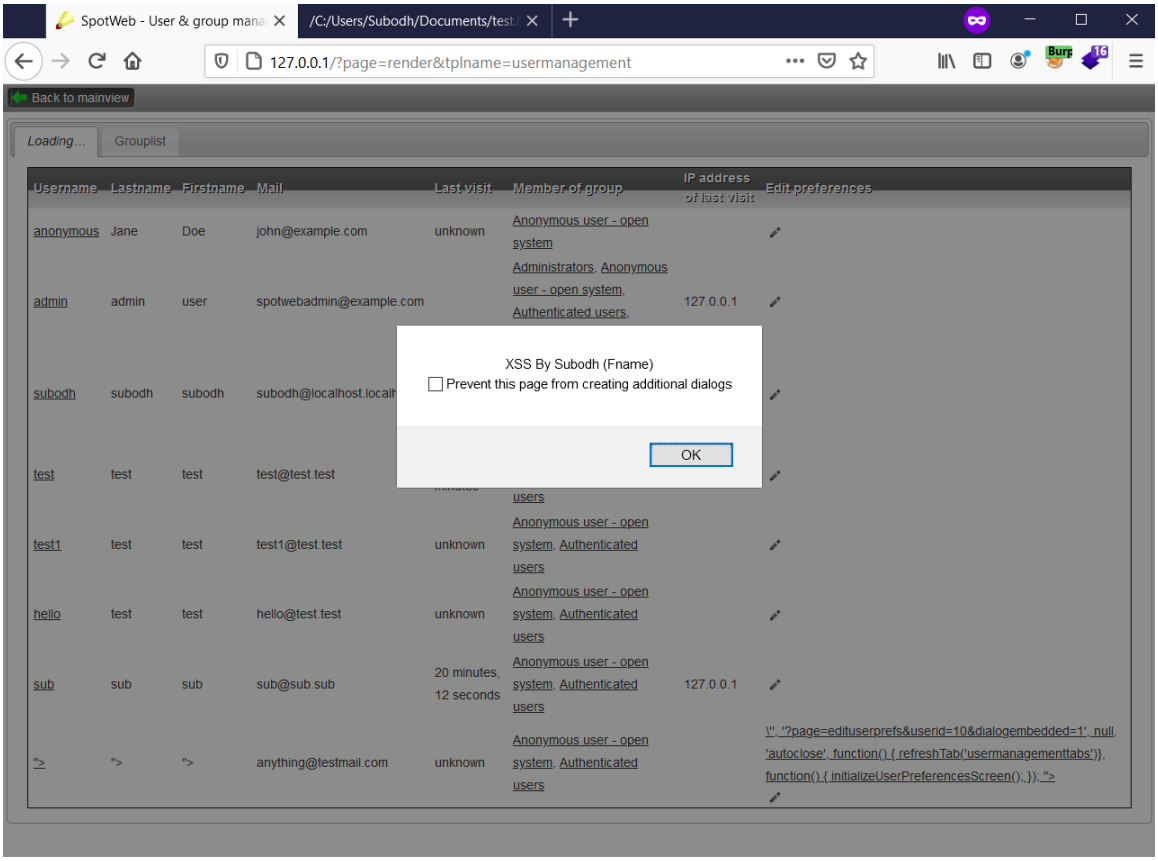


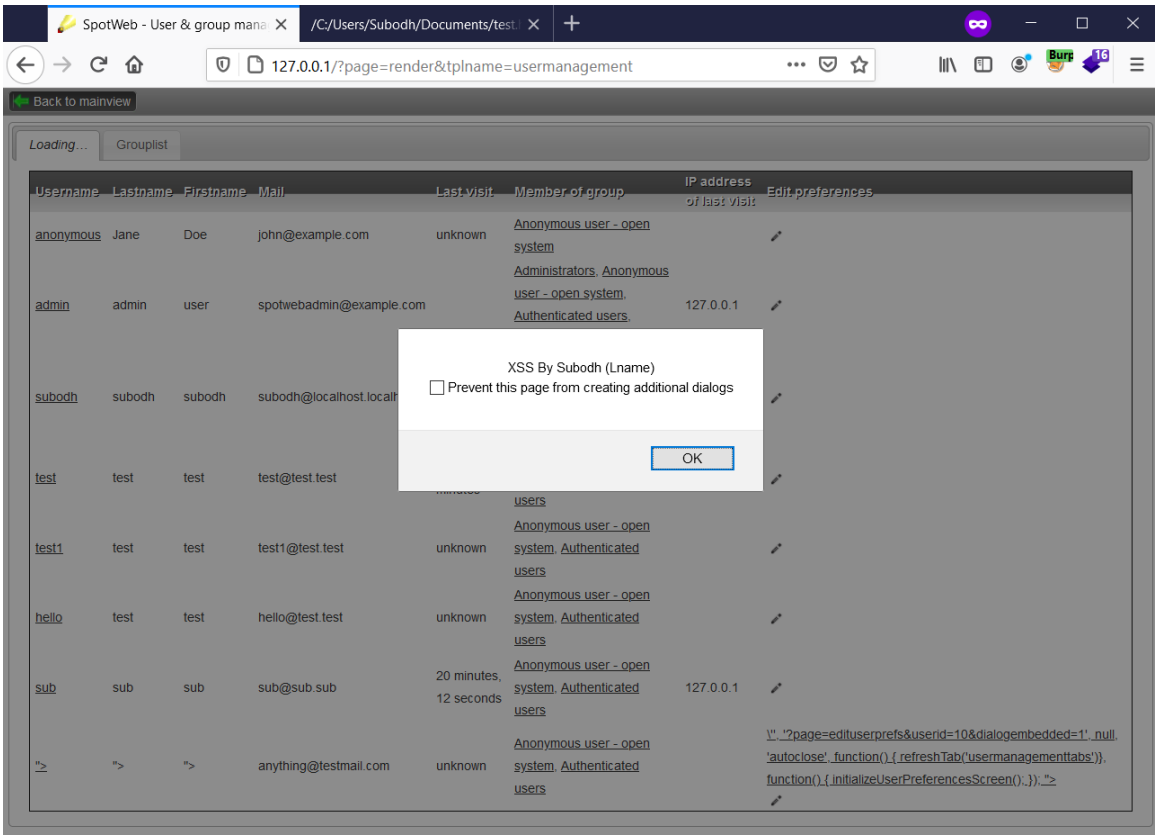
2. Fill the user details with XSS payload as follows and click on add.
   i.e. **Username:** "><script>alert("XSS By Subodh (Username)")</script>
   Firstname: "><script>alert("XSS By Subodh (Fname)")</script>
   Lastname: "><script>alert("XSS By Subodh (Lname)")</script>



3. Login the admin panel and open the users list:
   http://127.0.0.1/?page=render&tplname=usermanagement


SpotWeb - User & group mana
/C:/Users/Subodh/Documents/test.l
127.0.0.1/?page=render&tplname=usermanagement
Back to mainview
Loading...
Grouplist
Username Lastname Firstname Mail Last visit Member of group IP address of last visit Edit preferences
anonymous Jane Doe john@example.com unknown Anonymous user - open system
admin admin user spotwebadmin@example.com Administrators, Anonymous user - open system, Authenticated users, 127.0.0.1
subodh subodh subodh subodh@localhost.localhost
test test test test@test.test
test1 test test test1@test.test unknown Anonymous user - open system, Authenticated users
hello test test hello@test.test unknown Anonymous user - open system, Authenticated
XSS By Subodh (Username)
OK


SpotWeb - User & group mana
/C:/Users/Subodh/Documents/test.l
127.0.0.1/?page=render&tplname=usermanagement
Back to mainview
Loading...
Grouplist
Username Lastname Firstname Mail Last visit Member of group IP address of last visit Edit preferences
anonymous Jane Doe john@example.com unknown Anonymous user - open system
admin admin user spotwebadmin@example.com Administrators, Anonymous user - open system, Authenticated users, 127.0.0.1
subodh subodh subodh subodh@localhost.localh
test test test test@test.test
test1 test test test1@test.test unknown Anonymous user - open system, Authenticated users
hello test test hello@test.test unknown Anonymous user - open system, Authenticated users
sub sub sub sub@sub.sub 20 minutes, 12 seconds Anonymous user - open system, Authenticated users 127.0.0.1
"> "> "> anything@testmail.com unknown Anonymous user - open system, Authenticated users
\", '?page=edituserprefs&userid=10&dialogembedded=1', null, 'autoclose', function() { refreshTab('usermanagementtabs')}, function() { initializeUserPreferencesScreen(); }); ">
XSS By Subodh (Fname)
Prevent this page from creating additional dialogs
OK



Impact of XSS vulnerabilities:

The actual impact of an XSS attack generally depends on the nature of the application, its functionality and data, and the status of the compromised user. For example:

- In a brochure ware application, where all users are anonymous and all information is public, the impact will often be minimal.
- In an application holding sensitive data, such as banking transactions, emails, or healthcare records, the impact will usually be serious.
- If the compromised user has elevated privileges within the application, then the impact will generally be critical, allowing the attacker to take full control of the vulnerable application and compromise all users and their data.

How to prevent XSS attacks

Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.

In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- **Filter input on arrival.** At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- **Encode data on output.** At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- **Use appropriate response headers.** To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- **Content Security Policy.** As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

Researcher Profile:

Name: Subodh Kumar
LinkedIn: https://www.linkedin.com/in/s-kustm
GitHub: https://github.com/s-kustm