List of Vulnerabilities needed:

1. Injection

Several examples needed:
Multiple SQL Injections -
Bypass access control at login
Expose customer data
Expose event data

Maybe some XML, OS type injections later

2. Broken Authentication and Session Management

Comprimised passwords - e.g. stored in plain text, stored in cookies, poorly encoded/encrypted, can be intercepted, exposed in plain text in confirmation emails etc

3. Cross Site Scripting

Need examples of persisited (stored in DB and executed every time a page is visited) - such as comments etc
Reflected, so tampered and executed at run time
DOM based
Maybe an exploit via a crafted email or page

4. Insecure Direct Object References

Can access files, directories or keys that should be restricted
Can access restricted objects or modify them without access control
Able to access other users tickets and payment info by modifying URL strings or other elements.

5. Security misconfiguration
Exposed frame works, implementations, db, platforms
e.g. /shutdown

6. Sensitive Data Exposure
Session credentials exposed
Payment Data exposed in plain text, to all users
Code or other implementations such as SQL exposed to users in stack traces, error messages
User name and other credentials exposed to users

7. Missing Function Level Access Control
Ability to circumvent access control to restricted parts of the site, such as users being able to gain administrator access, or other restricted functions

8. Cross Site Request Forgery
Able to exploit a logged in user's session to send a forged HTTP request to the application, which it thinks are legitimate. This can be done by crafting a web page or embedding an image in a page that sends the session info to the attacker. This can lead to things like an attacker changing a user's password, so they always have access to the account.

9. Using components with known vulnerabilities
e.g. versions of jquery with known vulnerabilities in it, which could then be exploited.

10. Unvalidated Redirects and Forwards
Allow the application to redirect a user to an unvalidated, untrusted page e.g. from ticketmagpie.com/events/event1/ to evilticketmagpie.com on submit of a form, which would be a phishing site for example.

We could also try for some buffer overflow type errors, which would expose system information or other private data.

Also, remote code execution - so having a file upload feature of some kind, which could then execute some code to pull data down from the webserver such as system info and expose it in the browser.

See hackyourselffirst.troyhunt.com for some other examples:

- Senstive data/login not over HTTPS
- Unhandled exceptions exposed to the user
- Limited password length/complexity e.g. only 6-8 characters, only lower case, no special chars etc
- API not validating user credentials when used
- Password reset exposes password in plain text
- Comments can be tampered with/submitted on behalf of other users
- Response headers exposing system information e.g. server info,
- Logging is exposed to the user
- Auth cookies are not secure and can be used after the user has logged out
- Path to admin pages exposed in robots.txt
- HTML comments expose information or hidden areas/links once removed
- Xss protection disabled
- Clickjacking vulnerabilities (no x-frame options header set)
- Autocomplete enabled on passwords
- Don’t need to re-confirm password (even if it asks) on password reset, or password reset can not be validated server side
- Edit profile allows mass assignment risk (e.g. flag that makes the user an admin, or adding a value that makes a user an admin)
- No brute force protection around login attempts
- No anti forgery tokens to protect against CSRF
- Mass assignment risks on modifying user info

- HTTP Request verbs enabled that perhaps shouldn’t be, e.g. TRACE or OPTIONS