AnDeWA: An Approach for Analyzing and Detecting Work Flow Deviation Attacks in Web Applications

Sireesha C
C-DAC
Hyderabad

Jyostna G
C-DAC
Hyderabad

Raghuvaran P
C-DAC
Hyderabad

P R L Eswari
C-DAC
Hyderabad

ABSTRACT
Workflow deviations in web application occur due to logical flaws left while designing, implementing and hosting the web application. It is really hard to find the workflow deviations in web applications without accessing the website database and the application sensitive information. In this paper, AnDeWA is presented as a lightweight approach for detecting the workflow deviations in web applications with the minimum prerequisites of users to role binding information. AnDeWA follows the dynamic analysis technique which analyzes the web application behavior at a run time to detect the workflow deviation attacks.

General Terms
Web Application Security.

Keywords
Authentication and Authorization bypass, cross-site scripting, session hijacking, work flow analysis.

1. INTRODUCTION
The cyberspace and web have made the entire universe get together. Furthermore, now-a-days web is being used heavily in several states to provide citizen services, which include banking and administration. All the same, these improvements in Internet technologies are being worked to induce untoward effects. Vulnerabilities in web applications are utilized as vehicles to launch several attacks. According to Symantec, survey reports-2013 [1], small businesses and organizations are being targeted by attackers. Popular web application threats [2] include SQL injection, Cross-Site Scripting (XSS) [3], Authentication & Authorization bypass, Session Hijacking [5] [6] and Cross-Site Request Forgery (CSRF) [4].

Sometimes configuration file settings are also exploited for launching the attacks. For instance, in PHP [7] [8], global variable details are used by an attacker to acquire unauthorized access to the application. All these attacks are gained by compromising either web application or misconfiguration of .config files. In order to protect from these attacks, various research efforts are realized in developing browser side as easily as well web application side security solutions. Through this paper, we represent AnDeWA- an approach for Analyzing and Detecting Workflow deviation Attacks in web applications. This security solution is implemented and tested for PHP based web application and outcomes are promising.

2. EXISTING SOLUTIONS
In order to detect & prevent web application attacks, source code as well as run time analysis approaches [9] [10] [11] are applied. Existing solutions' pixy [12], rips [13], MIMOSA [14] and IBM Rational AppScan [15] require scripting code of a web application in order to detect the vulnerabilities. Swaddler [16] is a solution, in which vulnerabilities are detected by analyzing the state of web application based on session values at a PHP interpreter level during runtime. Some other result is the Acunetix web vulnerability scanner [17], audit's web applications by checking for exploitable hacking vulnerabilities through static analysis. To provide the security at web application level another possible solution is the use of Web Application Firewalls (WAF) [18]. But WAFs are designed by white listing the rules. The rule set of the WAFs describes the behaviour of the application. But these WAFs are failing to prevent the Session Hijacking; Privilege Escalation and Logical flaws exist in web applications due to the inability in white listing the rules of defected code and session maintenance. In our research work we are targeting to detect and prevent workflow bypass attacks by using dynamic analysis approach without seeing the application source code.

3. OUR APPROACH
AnDeWA is used to detect workflow deviation attacks like SQL injection; authentication & authorization bypass through session stealing and sequence bypass attacks. The solution works without disturbing the application database, without carrying user's sensitive information and without opening any external ports.

AnDeWA: works as an interceptor like OWASP Webscarab [19], which captures the web communication. AnDeWA capture web request and response messages along with the session flags. These details are gathered up to produce a behavioral model of the web application and are stored in a database on the host. Session flag in the model indicates the existence / non-existence of the session. A value 1 indicates the presence of the session for accessing the web page and 0 for web pages without sessions. This behavioral model is enforced at runtime along with the details like user agent and client IP address to detect the workflow deviation attacks. Figure 1 shows outline of AnDeWA.

AnDeWA operates in two phases: Learning phase and Detection phase. During the Learning phase, AnDeWA monitors the web application behavior in attack-free environment. It uses the spider technique [20] to crawl internally to every single web page and generates profiles and constructs the model by covering the complete behavior of the web application. During Detection phase, along with the web request the user agent and client IP address are also monitored, and the model is enforced to detect workflow deviation attacks. The detected deviations are reported for further analysis.

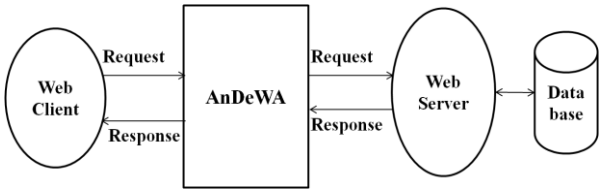

Fig. 1: AnDewA

3.1 Design Layout

Working functionality of the Learning phase and Detection phase are as follows.

3.1.1 Learning Phase

Figure 2 shows the AnDeWA functioning at Learning Phase with Profiler Engine and Model Generator modules. Profiler Engine captures the web communication for different roles. For each role, Profiler Engine collects the web request in the form of request header information, records and passes to the web server. The response from the web server is forwarded to the web client. Along with the request and response information, Profiler Engine also records the chronological succession of web requests with regard to current and previous web request state.

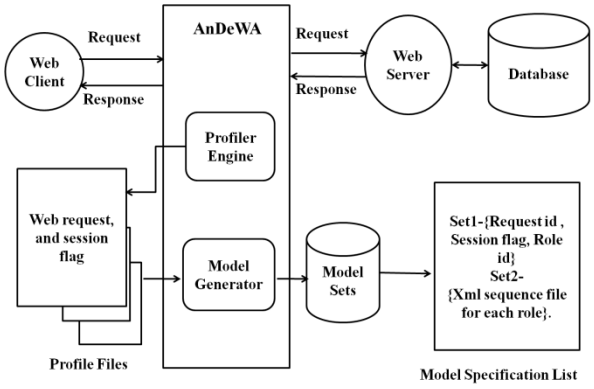

Fig. 2: Functioning at Learning Phase

For each web request, a separate communication id is created to differentiate between requests coming from web clients. And corresponding request header data is laid aside in a file with a name "communication id_request". It also extracts the cookie id from the header to check the session existence. If the session exists, session flag is set to 1 otherwise session flag is set to 0. The session flag for that request has been redeemed in "communicationid_Srequest" file name. And also saves the sequence of pages crawled by each role user in a "roleid.xml" file. For example, with request communication id to be 1, corresponding header information is saved in 1_request and session flag is saved in 1_Srequest file names. For form authentication web request, it also determines the form values with regard to SQL Injection.

After recording the request information it forwards the request to the web server for processing. The response from the web server is collected and forwarded to the web client.

Profiler Engine internally spiders each web page and collects the request and response information. Spider covers all the web pages internally for strengthening the model of the application. This procedure is replicated for all the roles of the web application.

Model Generator is another module in Learning Phase, which works in offline mode. It analyzes the profile records based on the communication id and role, builds a relational model database for the particular web application behaviour. It first reads the "communicationid_request" files and creates a request id based on the method of calling and requested resource name. If requested URL http://example.com/login.php is using GET method, request id becomes GET_login.php. From the corresponding "communicationid_Srequest" file reads the session flag. Based on the profile records it creates two different model sets.

```
mysql> select * from req_res_stat;
+-----+--------+-----------------------------+-------------+------+
| sno | convid | reqresid                    | sessionFlag | role |
+-----+--------+-----------------------------+-------------+------+
|   1 |      1 | GET_Default.php             |           0 |    0 |
|   2 |      2 | GET_login.php               |           0 |    0 |
|   3 |      3 | GET_about.php               |           0 |    0 |
|   4 |      4 | GET_installationguide.php   |           0 |    0 |
|   5 |      5 | GET_forgotpassword.php      |           0 |    0 |
|   6 |      6 | GET_login.php               |           0 |    0 |
|   7 |      7 | GET_login.php               |           0 |    0 |
|   8 |      8 | POST_login.php              |           0 |    0 |
|   9 |      9 | GET_home.php                |           1 |    0 |
|  10 |     10 | GET_calender.js             |           1 |    0 |
|  11 |     11 | GET_home.php                |           1 |    0 |
|  12 |     12 | GET_jquery-ui.css           |           1 |    0 |
|  13 |     13 | GET_preferences.php         |           1 |    1 |
|  14 |     14 | GET_formwizard.js           |           1 |    1 |
|  15 |     15 | GET_allreports.php          |           1 |    1 |
|  16 |     16 | GET_risk_assesement.php     |           1 |    1 |
|  17 |     17 | GET_logout.php              |           1 |    1 |
|  18 |     18 | GET_index.php               |           0 |    1 |
|  19 |     19 | GET_login.php               |           0 |    1 |
|  20 |     20 | POST_login.php              |           0 |    1 |
|  21 |     21 | GET_home.php                |           1 |    1 |
|  22 |     22 | GET_calender.js             |           1 |    1 |
|  23 |     23 | GET_about.php               |           1 |    1 |
|  24 |     24 | GET_jquery-ui.css           |           1 |    1 |
|  25 |     25 | GET_formwizard.js           |           1 |    1 |
|  26 |     26 | GET_installationguide.php   |           1 |    1 |
|  27 |     27 | GET_forgotpassword.php      |           1 |    1 |
|  28 |     28 | GET_home.php                |           1 |    1 |
|  29 |     29 | GET_preferences.php         |           1 |    2 |
|  30 |     30 | GET_allreports.php          |           1 |    2 |
|  31 |     31 | GET_crawl.php               |           1 |    2 |
|  32 |     32 | GET_requestscan.php         |           1 |    2 |
|  33 |     33 | GET_schedule_information.php|           1 |    2 |
|  34 |     34 | GET_risk_assesement.php     |           1 |    2 |
|  35 |     35 | GET_abcd.php                |           1 |    2 |
|  36 |     36 | GET_abcd4m.php              |           1 |    2 |
|  37 |     37 | GET_logout.php              |           1 |    2 |
|  38 |     38 | GET_index.php               |           0 |    2 |
|  39 |     39 | GET_login.php               |           0 |    2 |
|  40 |     40 | POST_login.php              |           0 |    2 |
|  41 |     41 | GET_home.php                |           1 |    2 |
|  42 |     42 | GET_home.php                |           1 |    2 |
|  43 |     42 | GET_calender.js             |           1 |    2 |
|  44 |     44 | GET_jquery-ui.css           |           1 |    2 |
|  45 |     45 | GET_formwizard.js           |           1 |    2 |
|  46 |     46 | GET_preferences.php         |           1 |    3 |
|  47 |     47 | GET_usermanagement.php      |           1 |    3 |
|  48 |     48 | GET_usermanagement1.php     |           1 |    3 |
|  49 |     49 | GET_manager.php             |           1 |    3 |
|  50 |     50 | GET_allreports.php          |           1 |    3 |
```

Fig. 3: Model database with request, session flag and role

Model set I represent the model database, and each row contains communication id, request id, session flag and role. And for form based authentication it internally maintains rule set to avoid SQL Injection. Figure 3 shows the Model set1.

Model set II refers to the list of web pages accessible by each role, including web page sequence. A separate xml file is created for each role. Each tag in xml file other than the root element represents a page and list of possible accessible pages from that page. Figure 4 shows the 2 different xml sequence files for two distinct roles.

From the Figure 4, role1 user can access analysis.php, report.php, view.php and search.php pages from home.php. For role2, management.php, report.php, view.php and search.php are accessible from the home.php.

```
<?xml version='1.0' encoding="UTF-8">
<Pages>
<home.php>analysis.php, report.php, view.php,
search.php</home.php>
<analysis.php>declaration.php,publish.php
</analysis.php>
<view.php>viewusers.php,viewroles.php
</view.php>
</Pages>
```

```
<?xml version='1.0' encoding="UTF-8">
<Pages>
<home.php>management.php, report.php,
view.php, search.php</home.php>
<management.php>view.php,addusers.php
</management.php>
<view.php>viewusers.php,viewroles.php
</view.php>
</Pages>
```

Fig. 4: Example of Sequence of pages accessed by role1 and role2

Figure 5 shows the possible attacks addressed by each Model set.

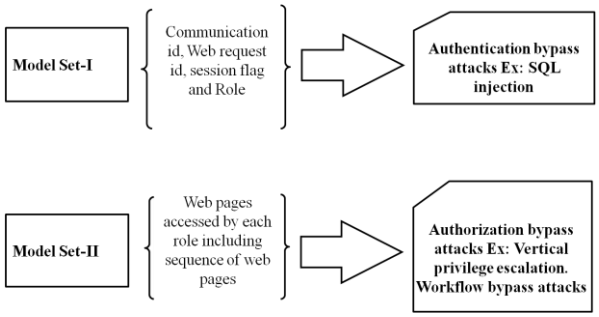

Fig. 5: Model sets representation and attacks addressed

3.1.2 Detection Phase

This phase enforces the model and continuously processes the web requests and web responses. It has Enforcement Engine and Verifier Engine. Figure 6 shows AnDeWA functioning during Detection Phase.

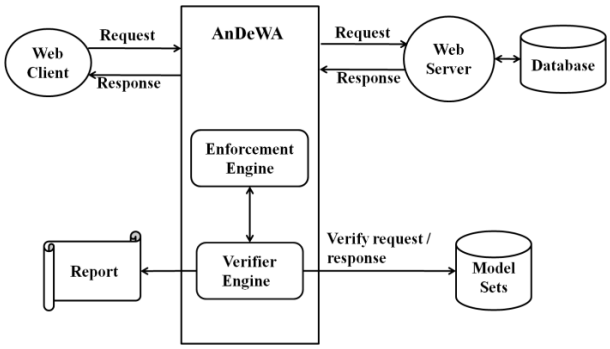

Fig. 6: AnDeWA functioning during Detection Phase

Enforcement Engine (EE) captures the web request before hitting the web server right away. Captured request header information is sent to Verifier Engine.

Verifier Engine (VE) checks the given request against model sets and sends the status of verification to the EE. If the request is a genuine behavior of the application, the status is mapped as "don't_block" otherwise if any differences occur with regard to model sets, then status is mapped as "block" and logs the deviations. Depending on the verification status EE proceeds further, if status is "don't_block" EE forwards the request to the web server application. And if status is "block" it won't forward the request to the web server application instead of it send an unable to process the request information to the web client.

VE first validates the web request information against Model Set I, which facilitates in finding authentication bypass attacks, which generally occur in a vulnerable web application by modifying the web page's user input or by hijacking the session. In one case the values satisfy the Model Set I behavior, and so it moves to the next stage of validation with respect to Model Set II otherwise VE sends the "block" status to the EE which stops the web communication.

And it also verifies the form authentication values internally to detect SQL Injection. Once the authentication is done, the authorization check on that page and sequence is verified with respect to Model Set II. This verification addresses the vertical privilege escalation attacks where one user is trying to access the pages of other roles and also addresses the sequence bypass attacks where the attacker is forcibly accessing the pages without following the sequence.

Once the request is satisfied with two levels of verification, then only VE sends the "dont_block" status to EE, from there the request is passed to the web server. In event of failure at any verification level, the VE sends "block" status to the EE and logs the error. EE rejects the request and sends the error page to the user. This process is followed for all requests..

3.2 Model Representation

The web application model is represented with the combination of request, response, input parameters, output parameters and session values. AnDeWA represents a request by request identifier 'Reqid' and session flag 'Sid'.

Reqid is a combination of request method (GET or POST) and a page name. And Sid is either 0 or 1.

Sid= {0, 1} (1)

So the web requests are represented with function 'f1' as

$$f1(request_k) = \{Reqid_k, s\} \quad (2)$$

Where s∈Sid.

Set 'F1' contains all web requests of web application.

$$F1 = \bigcup_{k=0}^{N-1} f1(request_k) \quad (3)$$

Where 'N' represents the maximum number of requests

Function 'f2' represents a set with previous and current request identifiers for a particular Reqid.

$$f2(Reqid_n) = \{Reqid_{n-1}, Reqid_n\} \quad (4)$$

Set 'F2' contains sequence of requests

$$F2 = \bigcup_{n=1}^{N-1} f2(Reqid_n) \quad (5)$$

Where 1≤n≤N-1.

Function 'f3' represents a set of pages accessed by each role. Each role is exemplified by an identifier 'Rid'.

$$f3(Rid) = \{\{Rid, Reqid_1\}, \{Rid, Reqid_2\}, \{Rid, Reqid_3\}, \ldots\} \quad (6)$$

Where Reqid1, Reqid2, Reqid3,… ∈F1.

Set 'F3' contains web pages accessed by all the roles.

$$F3 = \bigcup_{p=0}^{M} f3(Rid_p) \quad (7)$$

Where 'M' represents the number of roles in an application.

From (3), (4), and (5) model sets are evolved. Model set I (M1) has detailed information about each request along with the role

$$M1 = F1 \bowtie_{\langle F1.Reqid = F3.Reqid \rangle} F3 \quad (8)$$

Model set II (M2) contains the list of pages accessed by each role with sequence.

$$M2 = F2 \bowtie F3 \quad (9)$$

3.3 Model Enforcement

Model sets are enforced dynamically while executing the web application. Let us say one user is logged in and corresponding web communication is verified as per the following.

1. {f1(request),Rid}∈M1 -verifying the existence of the web page request with respect to Model set I.

2. f2(Reqid)∈F2 and f3(Rid)∈M2 -Comparison of sequence of requests made to the web server for particular role with Model set II.

Only when it satisfies the above two conditions, the requests are processed by the web server, if any of the above conditions fail AnDeWA blocks corresponding communication

3.4 Implementation Details

3.4.1 Learning Phase

This phase generates the profile records by analyzing the request header information for each request. Request Processor() catches the request and creates corresponding request profiles. Collected profiles are analyzed and model database is created for that web application. Model database is implemented using MySQL database and xml files. MySQL database contains request information of each page with the combination of communication id, request id, session flag and role. And separate xml file is created for each role. The xml file contains the chronological succession of pages accessed by each role.

3.4.2 Detection Phase

To analyze the application web requests coming from different web clients an apache module has been integrated with detection phase component of the workflow analyzer. Apache module intercepts the request and transmits the request to Verifier Engine for checking the requests against a model database. If request is genuine, apache module forwards the same request to the web server otherwise blocks the request. IPC mechanism has been implemented between the Apache module and Verifier Engine.

We are using the user-agent and extracting the client IP of each web request for differentiating the web clients and verifying against Model databases. This phase also analyzes the form based authentication request values to detect SQL Injection attack.

3.5 Experimentation Details

We have taken a web application with two roles manager and employer. Each user is having access to different web pages depending on the role. Figure 7 shows the approachable pages of manager and employer after authentication and some of the pages, which are accessible without any authentication. The dotted arrows represent that those web pages do not require any authentication, and solid arrows represent that authentication is required for accessing the web page.

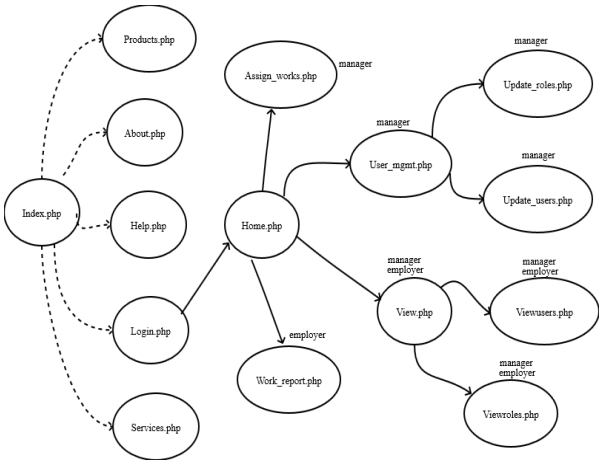

Fig. 7: Web Application Normal Scenario

Table 1 shows the requests made to the web application during Learning Phase. Table 2 & 3 shows the list of pages accessed and sequence of pages from the current page used by manager and employer roles respectively. This indicates the user cannot directly access the Viewusers.php from any of the pages other than View.php.

Table 1: Model database

S.No	Comm id	Reqest id	Session	Role
0	1	GET_About.php	0	0
1	2	GET_Help.php	0	0
2	3	GET_Login.php	0	0
3	4	POST_Login.php	0	0
4	5	GET_Services.php	0	0
5	6	GET_Products.php	0	0
6	7	GET_home.php	1	manager
7	8	GET_Assign_works.php	1	manager
8	9	GET_User_mgmt.php	1	manager
9	10	GET_Update_users.php	1	manager
10	11	GET_Update_roles.php	1	manager
11	12	GET_View.php	1	manager
12	13	GET_Viewusers.php	1	manager
13	14	GET_Viewroles.php	1	manager
14	15	GET_Home.php	1	employer
15	16	GET_Work_report.php	1	employer
16	17	GET_View.php	1	employer
17	18	GET_Viewusrs.php	1	employer
18	19	GET_Viewroles.php	1	employer

Table 2: Role- manager

Current Page	Next Accessible Pages
Home.php	Assign_works.php User_mgmt.php View.php
User_mgmt.php	Update_users.php Update_roles.php
View.php	Viewusers.php Viewroles.php

Table 3: Role- employer

Current Page	Next Accessible Pages
Home.php	Work_report.php View.php
View.php	Viewusres.php Viewroles.php

3.6 Experimentation Details

Figure 8 shows the possible attack scenario in a web application. The web application is following the simple authentication process by asking for a username and password in Login.php and then allowing the users to access other web pages without checking status of authentication, whether it is successful or not. The problem with this is that it assumes that the only way to get the Home.php is through Login.php, but an attacker can directly access the Home.php without any authentication leading to "Authentication Bypass" attack.

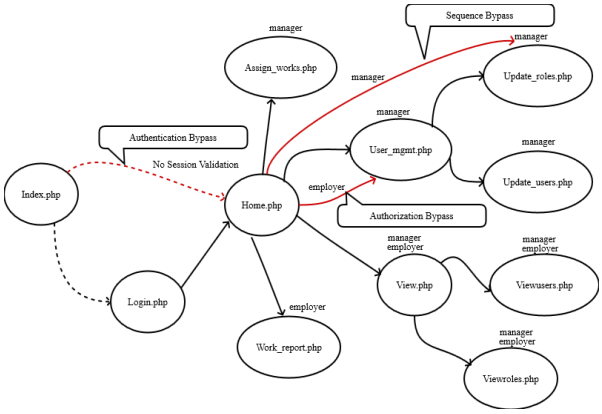

Fig. 8: Possible attacks

Another possible exploitation occurs due to failure in checking access control rules, like not binding the user to role for every web page or not validating the access rights for that page. In this example, only manager is having the privileges to access the User_mgmt.php, from there Update_users.php, for updating the users Update_roles.php, for updating the roles. Through a forced browsing attack, a user with an employer role is guessing and accessing the User_mgmt.php. Failure in checking the access right at User_mgmt.php leads to "Authorization Bypass" attack.

Usually web application has a proposed sequence to access any web page. For some of the crucial pages, it is very much required to stick to the same flow because they depend on the data coming from the earlier pages. On each page matching the session and previous page data is sometimes necessary. In this example, Update_roles.php can access from the User_mgmt.php only. With direct URL reference to Update_roles.php leads to "Sequence Bypass" attack.

3.7 Detection Scenario

The authentication Bypass attack can be addressed in Detection phase by checking the request id with session flag and role present in Table 1. Home.php is accessible only if a session exists, and the corresponding role is identified. The authorization bypass attack can be detected by matching the user's role but the request is not successful because the on every single page by referring Table 1.

Sequence bypass attack is addressed by validating the sequence of pages accessible by each role, which is depicted in Table 2 and Table 3. Suppose an employer is logged in and trying to access the User_mgmt.php forcefully, sequence is not present in an employer role with respect to Table 3. Thus we can also block the privilege escalation attempts.

3.8 Case Study

We also analyzed the web behavior of various vulnerable PHP applications like wackopicko, scarf, bookstore_php4t, portal_php4t to detect workflow deviations. wackopicko has the two different paths for authenticating the normal user and admin user. And it is vulnerable to an authentication bypass (using SQL Injection) and it contains some broken links. We model the web behavior by considering the two different authentication paths for 2 roles. During the Detection phase, we analyzed the web behavior and trying to perform SQL Injection, the request is blocked from processing.

Scarf application is vulnerable to authorization and sequence bypass attacks. Where the attacker can directly access the generaloptions.php page without login as an admin and delete the users. We modelled the scarf web behavior with admin and normal user authentication. In Model Set I the generalpolicy.php request represented with RequestId "GET_generalpolicy.php" and SessionFlag is "1". In Model Set II the sequence maintained as login.php->generalpolicy.php for an admin role. Once the model is enforced in Detection phase it first checks the generalpolicy.php with respect to Model Set I, if the attacker can directly access this page it may not contain a session value initially so the web request is blocked. Some other potential scenario is if the attacker logged in as a normal user and attempting to access the generalpolicy.php, in this case also Model Set I is not satisfied even though the page has the session, but the RoleID is not satisfied for that request. And it can also detect from the sequence file present for the normal user where the normal user cannot have this page. Thus we can detect sequence bypass with respect to Model Set II.

We also considered the bookstore and portal PHP applications which are vulnerable to authentication bypass through SQL Injection and sequence bypass attacks. We are able to detect the both attacks with respect to Model Set I and Model Set II. Table 4 shows the list of possible attacks detected with respect to model sets for different vulnerable web applications.

Table 4 : Detected attacks with respect to Model sets

Application Name	Detected attacks	Model checking
Wackopicko	Authentication bypass through SQL Injection.	Model set I
Scarf	Authorization bypass through forceful browsing	Model Set II
bookstore_php4t & Portal	Authentication bypass through SQL Injection	Model Set I

4. PERFORMANCE DETAILS

We deployed our security solution in PHP based web server and analyzed the performance of web application by observing page load time for different web pages using Lori add-on [21] installed in Firefox with/without AnDeWA Detection Phase. Here the page load time measures the time between loading of the page to completely render of the page in seconds. Table 5 shows the page load timings in seconds with/without AnDeWA.

Figure 9 shows performance overhead with AnDeWA and without AnDeWA. With AnDeWA, the curvature is going slightly higher because the Detection Phase verifies every single request against model sets and forwards the request to

the web server. With our observation, the average page load time is 0.22 seconds when AnDeWA is installed.

Table 5: Page load timing with/without AnDeWA

Without AnDeWA	With AnDeWA
0.332	0.515
0.379	0.595
0.307	0.583
0.244	0.541
0.425	0.52
0.28	0.731
0.583	0.813
0.582	0.83
0.43	0.766
0.47	0.716
0.073	0.091

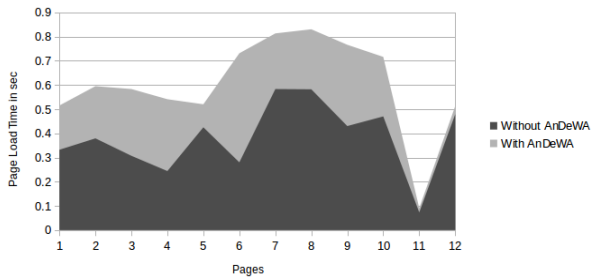

Fig. 9: Performance Overhead

5. CONCLUSION

In this paper, we discussed about monitoring web application behavior at run time and an approach for detecting and preventing workflow deviations. AnDeWA security solution identifies the authentication bypass; session hijacking and sequence bypass attacks.

Furthermore, it addresses the authorization bypass attack if users with role binding details are known in the prior. AnDeWA has its own limitation like, it is able to detect and prevent Authorization bypass for different role users, but users within the same role is trying to bypass is not addressed. By checking the user's session at every page can address this issue. Another limitation is we are crawling the site by considering the href links and opening the authentication pages (login and logout pages) for different roles in a browser which may not cover all the web pages. If the web site is designed with form based actions where the manual interaction is mandatory to pass the parameters, automatic crawl may not encompass. To come up to this problem we are attempting to render the form action based pages in browser to collect the data from the user which aids to crawl to next page.

6. ACKNOWLEDGMENTS

Our sincere thanks to Department of Electronics & Information Technology (Deity), Ministry of Communications and Information Technology, Government of India for supporting this research work.

7. REFERENCES

[1] Symantec- Internet Security Threat Report 2013 :: Volume 18

[2] http://www.security-audit.com/blog/owasp-top-10-2013/

[3] Alexander Roy Geoghegan, Natarajan Meghanathan*. “Cross Site Scripting (XSS)”.

[4] Nenad Jovanovic, Engin Kirda, and Christopher Kruegel. “Preventing Cross Site Request Forgery Attacks”.

[5] Bhavna C.K. Nathani Erwin Adi.Website Vulnerability to Session Fixation Attacks

[6] http://www.cs.utexas.edu/users/mckinley/papers/son-phd.pdf

[7] Dafydd Stuttard, Marcus Pinto. The Web Application Hacker’s Handbook-Discovering and Exploiting Security Flaws.

[8] David K. Liefer,Steven K. Ziegler. “PHP Vulnerabilities in Web Servers”.

[9] Marco Cova. Taming the Malicious Web: Avoiding and Detecting Web-based Attacks.

[10] Symantec.White Paper: Web Based Attacks,February 2009.

[11] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, D. T. Lee , Sy-Yen Kuo. Securing Web Application Code by Static Analysis and Runtime Protection.

[12] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: A static analysis tool for detecting web application vulnerabilities (short paper)

[13] Johannes Dahse. RIPS - A static source code analyser for vulnerabilities in PHP scripts.

[14] Davide Balzarotti, Marco Cova, Viktoria V. Felmetsger, and Giovanni Vigna.Multi-Module Vulnerability Analysis of Web-based Applications.

[15] IBM Rational AppScan Standard - http://public.dhe.ibm.com/common/ssi/ecm/en/rad14019usen/RAD14019USEN.PDF

[16] Marco Cova, Davide Balzarotti, Viktoria Felmetsger, and Giovanni Vigna. Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications.

[17] Acunetix Web Vulnerability Scanner-http://www.acunetix.com.

[18] https://www.owasp.org/images/b/b0/Best_Practices_WAF_v105.en.pdf

[19] https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

[20] Jeff Heaton Web Spidering. http://www.developer.com/java/other/article.php/1573761/Programming-a-Spider-in-Java.htm

[21] http://www.searchenginejournal.com/best-firefox-addons-to-analyze-the-page-load-time/12419/