CRITICAL ANALYSIS ON SECURITY OF PHP SCRIPT WEB APPLICATIONS

K. SRI VIDYA
U. G Student
Sreenidhi Institute of Science and Technology,
Hyderabad, Telangana, India.

Abstract:

In current years, recognition ultra-modern business global has been moved modern day the net. net programs offer a beneficent interface non-prevent thus supplying to malicious users a wide spectrum modern possible attacks. consequently, the security latest web programs has become a critical difficulty. The gear for computer virus discovery in languages used for web-application improvement, such as php, be afflicted by a extraordinarily high fake-effective price and occasional insurance modern day actual errors; this is caused specially by way of imprecise modelling present day dynamic functions brand new such languages and route-insensivity today's the gear. on this paper, we will show susceptible points modern the equipment and describe our novel method to these problems. we are able to display how our approach handles latest the situations where other tools fail and illustrate it on examples.

KEY WORDS: PHP, Web applications, path-insensivity, dynamic features, technique.

I. INTRODUCTION

currently, as commercial enterprise world has moved its cognizance modern day the net, modern-day packages have been moved on- line, and this fashion is still persevering with. in keeping with the CENSUS [17], the net retail sales within the US in 2010 reached over a hundred and sixty billion US greenbacks. safety and safety ultra-modern the net applications worried in such transactions is consequently trendy the pinnacle priority. an ordinary web software is available and operational 24/7, for this reason no longer setting any time pressure on malicious customers; a beneficent interface those applications provide in addition widens the hacker's field. amongst the 25 most not unusual programming errors, those specific to net programs form a significant modern-day this institution [5]; the examples encompass improper neutralization cutting-edge square commands, cross-website request forgery, and lacking authorization. The most common programming language used on the server facet is php [13]. personal home page features many special attributes that make it special from commonplace programming languages, particularly as some distance as dynamism is worried. The examples are inclusion latest a file specified by way of a runtime-computed file name and the eval assemble permitting runtime creation present day code this is achieved afterwards. This makes it difficult or ultra-modern even impossible to apply the equal strategies and tools for finding insects or for correctness frication as within the case modern day "non-web" programming languages. A. trouble announcement and goals ultra-modern attention has been paid to the development cutting-edge meth- ods and gear that might assist debugging these applications and organising their correctness in a few experience, because the This paintings become partly supported

by means of Charles college basis furnish 431011 and by way of institutional money from PRVOUK. strategies for "non-web" languages cannot be without problems implemented. The current gear, but, nevertheless be afflicted by low errors insurance, a relatively high false-positive price, and today's additionally from a weak support state-of-the-art language constructs, such as lessons, dynamic includes, and the eval declaration [9], [18]. in this paper, we advocate a way for the identification trendy bugs interior web applications as a result of information flow modern unsanitized inputs from the consumer to sinks (sq. queries, URL structures, output in preferred, etc.) internal web applications written in php. We describe our technique and reveal benefits present day our method over those present in related equipment and illustrate our method on an instance.

II. ERRORS INSIDE WEB APPLICATIONS

A huge wide variety of protection holes interior net packages can be grouped below one category which allows facts to propagate from a consumer enter (resources, e.g., shape fields on an internet page) into database queries, URLs, JavaScript code, etc. (sinks) without checking if they're malicious [14]. these can be averted by filtering consumer input, escaping the output, and by using retaining track of the input data [14]. Filtering enter is a procedure of stopping invalid statistics from entering into sinks. Blacklist filtering excludes malicious facts, while whitelist filtering excludes all information besides for that explicitly listed; for that reason it's miles distinctively more secure than blacklist filtering due to the opportunity of a missing object inside the list. Escaping and encoding special characters that the utility outputs prevents injection of malicious code and facts. Taint evaluation is a technique which makes it possible to discover the information paths from sources to sinks [10], [21], [11], [3]. It marks facts coming from assets as tainted and then propagates the taint markings. facts is tainted if it may be influenced through a consumer and, at the identical time, it isn't sanitized. observe that some resources represent a larger safety threat than others and it's miles essential not only to "taint" records but additionally to distinguish among distinctive taint assets.

III. STATE OF THE ART

Huang et al. [10] developed a static analysis for php appli- cations in WebSSARI tool. Xie [21] discusses the constraints in their technique, in particular that it is intra procedural and it does no longer model dynamic capabilities along with dynamic arrays, objects, dynamic variables, and dynamic includes. To perceive vulnerabilities, the method performs taint evaluation. Their approach does no longer permit for a custom sanitization, because facts are handiest taken into consideration to be sanitized if they're processed with a specified sanitization feature. In latest years, recognition ultra-modern commercial enterprise world has been moved cutting-edge the internet. net programs provide a beneficent interface non-forestall hence providing to malicious customers a wide spectrum contemporary feasible assaults. consequently, the safety modern-day web programs has become a critical issue. The tools for trojan horse discovery in languages used for internet-application improvement, consisting of Hypertext Pre-processor, suffer from a enormously high fake-superb rate and occasional coverage ultra-modern actual errors; that is

precipitated mainly through unprecise modelling modern-day dynamic functions modern such languages and direction-insensivity modern-day the tools. in this paper, we can show vulnerable factors ultra-modern the tools and describe our novel method to those problems. we are able to display how our approach handles present day the conditions in which other equipment fail and illustrate it on examples.

INTRODUCTION

Recently, as business world has moved its focus towards the Internet, a number of applications have been moved online, and this trend is still continuing. According to the CENSUS, the online retail sales in the US in 2010 reached over 160 billion US dollars. Safety and security of the web applications involved in such transactions is therefore of the top priority. A typical web application is available and operational 24/7, thus not putting any time pressure on malicious users; a generous interface these applications provide further widens the hacker's field. Amongst the 25 most common programming errors, those specific to web applications form a significant part of this group; the examples include improper neutralization of SQL commands, cross-site request forgery, and missing authorization. The most common programming language used at the server side is PHP. PHP features many special attributes that make it different from common programming languages, especially as far as dynamism is concerned. The examples are inclusion of a file specified by a runtime-computed filename and the eval construct allowing runtime construction of code that is executed afterwards. This makes it hard or sometimes even impossible to apply the same techniques and tools for finding bugs or for correctness verification as in the case of "non-web" programming languages.

Problem statement and goals:

A lot of attention has been paid to the development of methods and tools that would help debugging these applications and establishing their correctness in some sense, since the. Methods for "non-web" languages cannot be easily applied. The current state-of-the-art tools, however, still suffer from low error coverage, a relatively high false-positive rate, and often also from a weak support of language constructs, such as classes, dynamic includes, and the eval statement. In this paper, we propose a method for the identification of bugs inside web applications caused by data flow of unsensitized inputs from the user to sinks (SQL queries, URL constructions, output in general, etc.) inside web applications written in PHP. We describe our method and demonstrate benefits of our approach over those present in related tools and illustrate our method on an example.

ERRORS INSIDE WEB APPLICATIONS:

A huge number of security holes inside web applications can be grouped under one category which allows data to propagate from a user input (sources, e.g., form fields on a web page) into database queries, URLs, JavaScript code, etc. (sinks) without checking if they are malicious. These can be prevented by filtering user input, escaping the output, and by keeping track of the input data. Filtering input is a process of preventing invalid data from getting into sinks. Blacklist filtering excludes malicious data, while white list

filtering excludes all data except for that explicitly listed; thus, it is distinctively safer than blacklist filtering due to the possibility of a missing item in the list. Escaping and encoding special characters that the application outputs prevents injection of malicious code and data. Taint analysis is a technique which makes it possible to discover the data paths from sources to sinks. It marks data coming from sources as tainted and then propagates the taint markings. Data is tainted if it can be influenced by a user and, at the same time, it is not sanitized. Note that some sources represent a larger security threat than others and it is necessary not only to "taint" data but also to distinguish between different taint sources.

STATE OF THE ART:

Huang et al. developed a static analysis for PHP applications in WebSSARI tool. Xie discusses the limitations of their approach, in particular that it is intraprocedural and it does not model dynamic features such as dynamic arrays, objects, dynamic variables, and dynamic includes. To identify vulnerabilities, the approach performs taint analysis. Their approach does not allow for a custom sanitization, because data are only considered to be sanitized if they are processed with a specified sanitization function. The approach of Xie et al. uses inter-procedural analysis to find SQL injection vulnerabilities in PHP applications. They model automatic conversion of particular scalar types, uninitialized variables, simple tables, and include statements. However, they leave important parts of PHP unmodeled. In particular, they do not model references, object-oriented features of PHP, and they ignore recursive function calls. To model sanitization process, the approach performs taint analysis. Sanitization can occur via calls to specified sanitization functions, casting to safe types, and a regular expression match. This means that the approach maintains a database of sanitizing regular expressions. Wasserman et al. use grammar-based string analysis following Minamideto find a set of possible string values of a given variable at a given program point and gain this information to detect SQL injections. However, the employed analysis has an incomplete support for references and does not track type conversions. performs taint analysis of PHP programs and it provides information about the flow of tainted data using dependence graphs.

It uses literal analysis to resolve include statements and performs alias analysis. However, it does not model aliases between variables and members of arrays. Next, Pixy lacks type inference, does not model PHP's variable-variables construct as well as variable-indices and provides only a very limited support of object-oriented features. Moreover, similarly to WebSSARI it performs only simple taint analysis and does not allow for custom sanitization routines. Balzarotti et al. extended Pixy to perform the analysis of the sanitization process and thus are able to deal with a custom sanitization. They combine static and dynamic analysis techniques to verify PHP programs. They perform string analysis through language-based replacement and represent values of variables at concrete program points using finite state automata. They also track what parts of strings are tainted. Static analysis that they employ is based on Pixy and has the same limitations. Moreover, the

database may not contain strings corresponding to attacks that were not considered in advance. Consequently, it can both miss vulnerabilities and cause false alarms.. Biggar et al. perform context sensitive, flow sensitive, interprocedurally static analysis of PHP in order to gain information usable for code optimizations in their PHP compiler. They combine alias analysis, type inference and literal analysis, model arrays, PHP's variable-variables construct, objects, references, scalar operations, casts, and weak type conversions. However, their analysis is closely tailored with their intent—to gain information usable for code optimizations. They gather information that must hold; information that may hold is tracked only in a very limited way. In most cases, they approximate information that may hold as unknown. This is not appropriate when the intent is to explore all possible behaviours of the code.

```
1  $users[1] = 'fred'; $users[2] = $_GET['inp'];
2
3  $t_users = & $users;
4  echo $t_users[1]; // Pixy reports the XSS vulnerability
5
6  $name = 'bob'; $index = 1;
7  if ($tainted) { $name = &$_GET['n']; $index = 2; }
8  echo $tainted ? htmlspecialchars($name) : $name; // Pixy reports the XSS vulnerability
9  echo $tainted ? htmlspecialchars($users[$index]) : $users[$index]; // Pixy reports the XSS vuln.
10
11 $ext = ".php"; $filename = 'inc' . $ext;
12 while (strpos($filename, '..')) $filename = preg_replace('..', '', $filename);
13 include($filename);
14 echo $users[2]; // Pixy reports the XSS vulnerability
15
16 printFirstIndex('tainted', $users[1], $users[2]); // Pixy misses the XSS vulnerability
17 function printFirstIndex($varName, $untainted, $tainted) { echo $$varName; }
18
19 $user_ids = 2;
20 $user_ids[2] = $_GET['inp']; // because $user_ids is scalar, this line does nothing
21 echo $user_ids[2]; // Pixy reports the XSS vulnerability
```

Fig. 1. Dynamic Features Of PHP Causing False Alarms And Missed Vulnerabilities In Pixy Tool

The method of Artzi et al. generates test inputs routinely, video display units internet applications for crashes, and validates that the output conforms to the HTML specification. The method utilizes symbolic execution to capture logical constraints on inputs, based on these constraints, it creates new inputs that would growth the code coverage. via walking an software on concrete inputs and the usage of personal home page runtime, they avoid the problem of modelling dynamic statements of php. To our expertise, a direction-sensitive approach to a static analysis for Hypertext Preprocessor has now not been but published. but, there has been a number of studies done inside the context of other languages. Examples of methods to direction-touchy static evaluation include.

OUR APPROACH:

Our purpose is to offer the developer with sufficient data in order that she/he can guarantee a accurate sanitization. In our case, this indicates employing evaluation that computes facts drift information the usage of dependence graphs, identifies assets of touchy statistics, sinks, and at each software factor continues:

• the taint and the sanitization fame for every variable,

• the set of possible values of each variable,

• the set of situations defined on the application's variables that should keep, and

• the set of viable sorts of each variable. inside the following, we describe how we advantage this statistics and the way we use it to stumble on vulnerabilities.

Outline:

the pre built integrated challenge of the evaluation is the built-ination of an arbitrary person enter integrated and the dynamism of Hypertext Pre-processor. To address this trouble, our evaluation consists of integrated built-in the follow built integrated steps:

1) construction of the manipulate-waft graph (CFG).

2) Static evaluation of constructed CFG.

3) Detection of vulnerabilities.

four) A route-sensitive validation of vulnerabilities.

We based our technique on a integrated of built-in paintings, and extended it to face the aforementioned troubles. We method the problem of creation of CFG built-in presence of dynamic statements built-in a comparable way as Pixy does. First, most effective such dynamic statements which can be without delay given by literals are built-into consideration. Then, we built-in built-information approximately viable values of variables, built integrated, and aliases integrated static evaluation of the CFG. We use this facts to resolve dynamic statements and assemble greater particular CFG that is analyzed integrated. We repeat this manner as long as new dynamic statements are bebuilt-ing resolved or an new release restriction is reached. built-inbuilt integrated to what the Pixy device does, our analysis built integrated kbuiltintegrated builtintegrated and we are able to remedy additionally polymorphic method calls us built integrated this approach. We extended model builting of php built integrated systems built-in integrated Biggar built-in integrated specific modelbuilting of aliasintegratedg and including certabuiltintegrated statistics to each frbuiltintegrated the built-in-to graph. reality tracks the fact whether or not given built-information at a given software built-inpobuiltintegrated holds for every execution direction from an entry built-in of the application to this application built-int; the way it's far mabuilt-intaintegrateded is describedintegrated underneath. Our static evaluation stems from the only integrated integrated Biggar . We tailored it to track the knowledge and tabuilt-int built-indata and augmented the literal evaluation to propagate symbolic values through 7fd5144c552f19a3546408d3b9cfb251 operations. We built-infer the sets of conditions that keep at every program built-inpobuilt integrated the usage of integrated conditional statements. on the start of a then branch integrated a given conditional assertion, the condition built-insimilar to this declaration is built integrated; similarly integrated, negation of the situation is built integrated to the start of the else branch. on the be a part of built-inpobuiltintegrated, the situation is elimbuiltintegrated. integrated built-in integrated step, we use integrated facts won builtin integrated the analysis to pick out vulnerabilities that may be false positives because of direction-built-insensitive evaluation and validate them course-sensitively.

Modelling of PHP data structures:

To model variables, array cells, and object fields, we use a factors-to graph much like the only introduced in . The factors-to graph includes 3 forms of nodes. A garage node represents a image-desk, an array, or an object. An index node represents a variable, an array cell, or an object subject. every index node is a child of a single storage node. in the end, a fee node

represents a scalar fee. Index nodes are connected with garage and price nodes that constitute their values the usage of fee edges. Aliasing is modelled the usage of reference edges among garage and index nodes. each reference area has a truth tag and a route tag (left, right, each, or none). each garage node contains an unknown subject representing the values of the index nodes which have statically unknown indices, e.g., $a[$dyn] and $$dyn if the cost of the variable $dyn is unknown.

Finally, each index node can be sturdy or weak. An example of a points-to graph is defined in Sect. IV-F. to start with, all nodes are sturdy. A node will become weak at a be part of factor when one among its reference edges becomes uncertain (it isn't always present in all joined branches) and is directed from the node (see the node call in Fig. 2; it became susceptible at join factor at line 7, Fig. 1). similarly, a node may also come to be weak after an venture—the regulations are described underneath. The set of values of a susceptible index node includes all storage and cost nodes accessible from the index node the use of all susceptible paths, and the values inside the unknown discipline of the discern storage node. A susceptible path is shaped through reference and price edges where simply the path of the closing reference area should correspond to the path of the path. The set of values of a strong index node is described in the identical manner besides that the unsure reference edges aren't taken into consideration. See Fig. 2 for an instance— the set of values of the node call is bob and u1, the set of values of the node n is u1. two index nodes are aliases if they may be handy from every other the use of the reference edges; they may be sure aliases if they are handy from each different the usage of certain reference edges only. Assigning a supply index node $s to a target index node $t (i.e., $t=$s) replaces all fee edges of the target node and its certain aliases with edges to all the values of the supply node. In case the cost is the storage node similar to an array, the garage node is copied after which a fee facet to the reproduction is created. next, the path of all reference edges from the target node and its positive aliases is changed to the course in the direction of the target node or the sure reference. built-integrated, the target node and all its integrated aliases are marked as strong and all built-in aliases as vulnerable. See Fig. 2 for an example—venture $name=$_GET['n'] might alternate the node name to strong, the node n to weak and the reference facet might have opposite course. built-ing a reference between a source and a target node (i.e., $t=&$s) copies all the values of the goal node to all integrateddex nodes connected with the goal node built-in a reference edge directed to the target node—that is performed integrated to no longer affect fee units of those nodes. next, it built-in all the rims from the goal node and provides a brand new oriented reference part from the goal node to the supply node. which will follow the semantics of php references (an analogy to UNIX filesystem hard built-inks), if a removed reference part reasons disconnection of nodes previously related thru the node whose side is built-inatedintegrated, new reference edges among disconnected nodes are built-introduced to preserveintegrated the integrated reference built-in (besides for the modified one) integrated resulting graph. The strong/vulnerable tag is copied from the supply to the target node. An

undertake built integrated assertion may also represent assignbuilt-ing a couple of supply integrateddex nodes to a couple of target integrateddex nodes (i.e., $t[$i]=$s[$i] built integrated $i has likely more than one values). with built integrated case of assignment to more than one goal nodes, the method is the identical same as built-in the case of built-in to a sbuiltintegrated target node except that a susceptible replace of the target nodes and their aliases is executed. this is, the goal nodes and all their aliases are marked as vulnerable; the unique route of the reference edges built-in addition to the built-in origbuiltintegrated price edges are preserved. integrated case of project to the unknown built-index node (e.g., $a[rand()]=1), the set of the assigned values is delivered integrated to the unknown subject. Our method built-in integrated the built-in data about the purpose of a given variable have builting a selected cost (i.e., that a variable $x has the price p due to the alias with a variable $y). as a result, integrated presence of built-in references, it is feasible to carry out greater precise updates than integrated . C. Static analysis Our static evaluation stems from context-touchy, control flow-touchy, route-built-insensitive integratedter-procedural static evaluation delivered integrated built-in. For each program variable and each application built-inpobuiltintegrated, we track records integrated about its aliases, literal values, sorts integrated, built-in integrated of this statistics and taintegratedt and sanitization fame of fee nodes built-in the use of the built-inpobuiltintegrated-to graph. Our analysis uses a built nation of concrete and symbolic execution while propagatintegrated literal values via operations. If all built-inputs of an operation are concrete, the specific version of the operation is used, otherwise the symbolic version is used. through us built integrated concrete operations, we lessen the imprecision; right here, we use the reference Hypertext Pre-processor implementation as to model integratedg the symbolic variations, we version mathematics operations built-in addition to operations with strbuilt-ings. For model builting strbuilt-ing operations, we use automata-based method provided built-in . facts approximately the tabuilt-int repute is propagated via operations built-inintegrated follow integrated way. We use unique taintegratedt mark integratedgs for distbuiltintegrated assets of facts. opposite to other approaches, we do now not built-in the tabuilt-int status after process built-ing statistics with any operation. This has two reasons:

(1) A correct escaping operation is identified no longer only through the source of records however also through the sink.

(2) We use the taint data to discover the information that can be manipulated via the person. The statistics is used to locate vulnerabilities extra to the ones because of wrong escaping. instead of doing away with the taint status, we song the sanitization reputation. for that reason, for every sanitization popularity and for each taint marking, we track whether statistics with the taint marking are sanitized the usage of the suitable sanitization habitual.

Data in the sink	Exploit
Tainted and match an attack pattern	SQL injection, XSS.

Tainted and not sanitized	Data not escaped: SQL injection, XSS.
Tainted or can be a null value.	Data potentially not filtered, can be manipulated by a user: Sensitive information leakage, semantic URL attack, spoofed form submissions, spoofed HTTP request
Related to a current session and not guarded.	CSRF attack, session fixation, session hijacking

TABLE I: POTENTIAL VULNERABILITIES IDENTIFIED BY THE ANALYSIS.

CONCLUSION AND FUTURE WORK:

In this paper, we supplied a new method to discovery of bugs inner internet applications written in php. even as being based totally on regarded techniques, to our expertise we're the first who combined them right into a unmarried one and stepped forward them to face the most important issues. We proposed unique modeling of aliasing and taint evaluation capable of detecting the maximum of vulnerabilities supplied inside the literature, e.g. ultimate but now not least, the novel contribution of our approach is its course-sensitivity. despite the fact that false alarms as a result of path insensitivity of existing tools were pronounced, we are not aware of any approach that addresses this issue. We verified troubles of current methods and showed how they may be handled by our technique. although validated on php, the approach is standard and can be modified and applied also on other languages for internet improvement. We accept as true with that the evaluation is scalable, high-priced path sensitive step of the analysis is completed most effective while it's miles important, i.e., to verify vulnerabilities that can be false alarms. fake positives can still appear even after the route sensitive step. however, particular evaluation blended with path touchy step and employed vulnerability detection promises both a decrease false-nice price and higher blunders insurance compared to related approaches as showed in Sect. IV-G. once a prototype implementation is completed, we are able to evaluate scalability of the method on numerous real web applications. destiny work will also check out and compare the present techniques to analyse and refine course-situations and probably adapt them to be usable inside the context of php applications. For sensible motives, it must be additionally viable to manually resolve complex parts of the analysed code. this will be done by means of introducing hints inside the form of code annotations. Code annotations ought to be used, e.g., to specify the values, sorts, sanitization repute of a given variable at a given application factor.

REFERENCES:

[1] Artzi et al. Finding Bugs in Web Applications Using Dynamic Test Generation and Explicit-State Model Checking. IEEE Trans. on Soft. Eng., 36(4), 2010.

[2] G. Balakrishnan, S. Sankaranarayanan, F. Ivancic, O. Wei, and A. Gupta. Slr: Path-sensitive analysis through infeasible-path detection and syntactic language refinement. In Static Analysis, LNCS. Springer, 2008.

[3] D. Balzarotti et al. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. S&P'2008, 2008.

[4] P. Biggar and D. Gregg. Static analysis of dynamic scripting languages, 2009.

[5] Common weakness enumeration. http://cwe.mitre.org/top25/.

[6] M. Das, S. Lerner, and M. Seigle. Esp: path-sensitive program verification in polynomial time. In PLDI '02. ACM, 2002.

[7] D. Dhurjati, M. Das, and Y. Yang. Path-sensitive dataflow analysis with iterative refinement. In Static Analysis, LNCS. Springer, 2006.

[8] I. Dillig, T. Dillig, and A. Aiken. Sound, complete and scalable pathsensitive analysis. In PLDI '08. ACM, 2008.

[9] J. Fonseca, M. Vieira, and H. Madeira. Testing and comparing web vulnerability scanning tools for sql injection and xss attacks. In PRDC'07. IEEE CS, 2007.

[10] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo. Securing web application code by static analysis and runtime protection. In WWW '04, 2004.

[11] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: a static analysis tool for detecting Web application vulnerabilities. In S&P'06. IEEE, 2006.

[12] Y. Minamide. Static approximation of dynamically generated web pages. In WWW '05. ACM, 2005. [13] PHP—Personal Home Pages. http://www.php.net.

[14] C. Shiflett. Essential PHP security. O'Reilly, 2006.

[15] G. Snelting, T. Robschink, and J. Krinke. Efficient path conditions in dependence graphs for software safety analysis. ACM Trans. Softw. Eng. Methodol., 2006.