Web Application Protection with the WAP Tool

Ibéria Medeiros
LaSIGE, Faculdade de Ciências,
Universidade de Lisboa - Portugal
ibemed@gmail.com

Nuno F. Neves
LaSIGE, Faculdade de Ciências,
Universidade de Lisboa - Portugal
nuno@di.fc.ul.pt

Miguel Correia
INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa - Portugal
miguel.p.correia@tecnico.ulisboa.pt

***Abstract*—In two decades the web became a standard framework for Internet applications. This involved changing from an initially simple hypermedia access platform to a complex blob of different technologies. This complexity associated to the increasing filtering of TCP/UDP ports everywhere in the Internet, turned web applications into favourite targets for cyber-criminals. The Web Application Protection (WAP) tool aims to secure web applications by analysing and automatically fixing their source code [1][1]. WAP currently handles PHP code, in which most web applications are written. As of April 2014, WAP has been used to process more than 1.5 million lines of code. This short paper briefly presents the tool and ongoing work on evolving it.**

I. The WAP approach

The WAP approach brings to source code analysis a tension observed in Intrusion Detection Systems (IDSs). These systems have been classified in two main categories. Knowledge-based IDSs contain a database of attack signatures created manually by human beings. Behaviour-based IDSs, on the contrary, learn about attacks – or normal behaviour – automatically using labelled data sets.

Our approach uses a hybrid of two analog approaches. WAP searches for *input validation vulnerabilities* in PHP source code: cross-site scripting (XSS), SQL injection, remote and local file inclusion, path traversal, OS command injection, and a few more. First, the tool has *knowledge crafted manually* about how to find these vulnerabilities. More specifically, it does *taint analysis*: it verifies if inputs can reach sensitive functions (sensitive sinks) without proper sanitisation or validation (taint analysis in Fig. 1). The lists of input entry points, sensitive sinks, and sanitisation/validation functions are produced by humans. Examples for XSS are in Table I

Entry points	Sensitive sinks	Sanitisation functions
$_GET	echo	htmlentities
$_POST	print	htmlspecialchars
$_COOKIE	printf	strip_tags
$_REQUEST	die	urlencode
HTTP_GET_VARS	error	
HTTP_POST_VARS	exit	
HTTP_COOKIE_VARS	file_put_contents	
$_FILES	fprintf	
$_SERVERS	file_get_contents	
	fgets	
	fgetc	
	fscanf	

TABLE I. Entry points, sensitive sinks, and sanitisation functions for XSS vulnerabilities.

Taint analysis is a common technique for searching for vulnerabilities in source code, used both by research tools (e.g., PREfix, CQual, Pixy) and commercial tools (e.g., Fortify, Coverity). However, our research has shown that it tends to produce false positives, i.e., to identify vulnerabilities that are not real. Therefore WAP uses a second form of analysis – *data mining* – to refine the results of taint analysis. Specifically, data mining is used to classify the vulnerabilities identified by taint analysis as false positives or not (false positives predictor in the figure). This analysis is based on *learning about attacks automatically* using a labelled data set of true and false positives.

A benefit of using machine learning is that the tool can be easily improved whenever more data is available, i.e., whenever a vulnerability or false alarm is verified to be so. The current data set has been used mostly for demonstrating the concept. It contains only 76 vulnerabilities labelled with just 15 attributes. Example attributes are the presence or not of functions that extract substrings, concatenate strings, or remove white spaces. However, the number of vulnerabilities and attributes can be increased. We defined a process to select the best classifier for the data mining done by the tool. The results may change with the data set, but currently the best classifier is logistic regression, among the ten compared.

Besides discovering vulnerabilities, WAP also removes them automatically using *code fixes*. These fixes do essentially proper validation or sanitisation of user input before it is used in a sensitive sink. Fixes are PHP functions developed by us, as problems were identified in several of functions available in PHP. They are inserted in sensitive sinks or close to them.

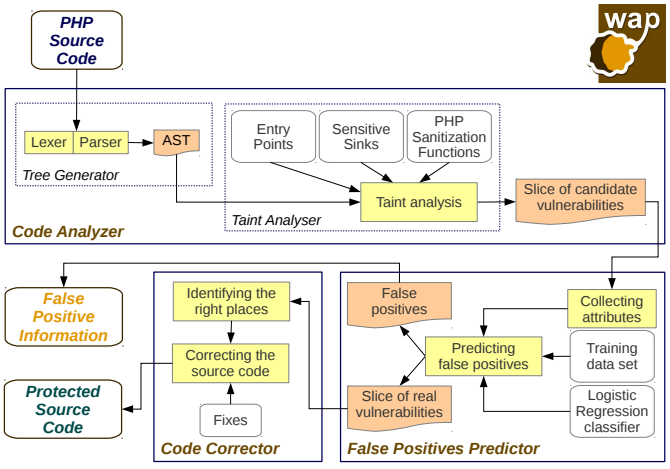

Fig. 1. Architecture of the WAP tool.

[1]Available online at http://awap.sourceforge.net

II. Challenges

Implementing WAP was quite challenging for several reasons: the need of reducing the number of false positives/negatives, the idiosyncrasies of PHP, etc.

To reduce false positives and false negatives, taintedness propagation has to be: (i) interprocedural, i.e., has to enter functions/methods whenever one is called; (ii) global, i.e., has to enter functions/methods even if they belong to different modules/files; and (iii) context-sensitive, i.e., it has to take into account the state in the point of the program where the function call was made. WAP has these three characteristics.

Another challenge we faced was the need to resolve the name of the include files to perform global analysis. This often involves getting the value of environment variables defined in files like config.php and in global, local, and array variables, which is not easy when analysing source code statically.

Object oriented programming languages are known to be harder to analyse that imperative languages due to the use of classes, inheritance, polymorphism, etc. PHP in this sense provides the worst-of-both-worlds as it supports both programming models, which WAP has to handle.

Another challenge was the need to combine both top-down and bottom-up approaches when navigating source code representations in memory. WAP navigates in the abstract syntax tree (AST) using the top-down approach to taint the entry points, then follows the bottom-up approach to propagate the taintedness to its parent. It identifies the vulnerable path and the right places to insert fixes using the bottom-up approach. Finally, it collects the attributes and performs the correction of the source code using the top-down approach.

Last but not the least, the syntax of PHP is not rigorously defined, so often the analysis of new applications breaks the parser and requires improvements.

III. New experimental results

Since the paper that presents WAP was submitted, we greatly expanded the number of experiments. WAP now analysed more than 1.5 million lines of code, almost 6500 files, discovering 459 vulnerabilities. The applications analysed include the Joomla and Tiki Wiki content management systems, with more than 400 thousand lines of code each. They also include other popular open source applications like Wordpress, applications that are vulnerable on purpose (e.g., the Damn Vulnerable Web Application), and a pack of vulnerable code samples from NIST's SAMATE project database.

The analysis is summarised in Table II. The columns indicate respectively: the web application analysed, the number of lines of code, the time taken to do the analysis, the number of files, the number of vulnerabilities found by the taint analysis, the number of false positives identified by the data mining phase, and the number of real vulnerabilities found.

There is a limited number of open tools available to search for vulnerabilities in PHP applications and to the best of our knowledge none that fixes them. Pixy searches for vulnerabilities, but with lower accuracy and precision than WAP [1]. PhpMinerII uses data mining do predict the presence of vulnerabilities, not to search for vulnerabilities or predict the presence of false positives like WAP. Nevertheless, in our experiments PhPMinerII has also shown worse accuracy and precision than WAP [1].

IV. Ongoing developments

We are currently improving WAP in three directions. First, we want to refine the data mining analysis by adding more attributes. This, however, involves having more labelled vulnerability specimens to understand how these attributes impact the existence of false positives. We are currently looking for ways to increase the size of our data set, by one or two orders of magnitude.

Data mining is typically about correlation, not causality. Our machine learning approach allows to identify combinations of attributes that are correlated to the presence of false positives, not necessarily to cause that presence. We aim to understand how to pass from correlation to causality, i.e., to what causes the presence of false positives, not only if they exist.

Our fixes were designed in order not to modify the (correct) behaviour of the web applications they fix. So far we witnessed no cases in which an application fixed by WAP started to function incorrectly. However, we did not show that this is the case. We have the objective of showing this. Currently we are exploiting the use of regression testing for this purpose.

Acknowledgments: This work was partially supported by the EC through project FP7-607109 (SEGRID) and the FCT through the LaSIGE Strategic Project (PEst-OE/EEI/UI0408/2014) and contract PEst-OE/EEI/LA0021/2013 (INESC-ID).

References

[1] I. Medeiros, N. F. Neves, and M. Correia. Automatic detection and correction of web application vulnerabilities using data mining to predict false positives. In *Proceedings of the International World Wide Web Conference*, pages 63–74, Apr. 2014.

webapp	loc	time	#files	#ta vv	#fp	#real vv
Butterfly	7679	2 s	37	17	0	17
currentcost	524	876 ms	6	1	0	1
CurrentCost	432	687 ms	3	5	0	5
dvwa	46989	12 s	496	9	0	9
emoncms	42148	12 s	323	0	0	0
emoncms_1	19996	7 s	229	14	2	12
gallery2	124414	27 s	644	0	0	0
getboo	42123	17 s	199	66	9	57
ghost	301	541 ms	14	7	0	7
gilbitron-PIP	328	595 ms	14	0	0	0
gtd-php	5063	4 s	64	66	1	65
joomla	419270	2m00s	1586	72	2	70
Measurit 1.12	940	795 ms	4	0	0	0
Measurit 1.14	1039	832 ms	4	0	0	0
multilidae	9867	4 s	76	0	0	0
orangehrm	184435	1m5s	652	121	0	121
peruggia	988	1 s	10	24	0	24
phpBB2	37327	18 s	78	0	0	0
Samate	353	364 ms	22	19	0	19
tikiwiki	499315	4m8s	1563	9	0	9
vicnum15	815	753 ms	22	25	1	24
volkszaehler	5883	1 s	43	0	0	0
WackoPicko	4156	3 s	57	7	0	7
webcal	36525	20 s	129	0	0	0
Wordpress	44254	10 s	215	12	0	12
total	1535164		6490	474	15	459

TABLE II. Analysis of open source applications.