Sound and Static Analysis of Session Fixation Vulnerabilities in PHP Web Applications

Abdelouahab Amira
Research Center on Scientific and Technical Information
CERIST, Algiers, Algeria
A.MIRA University, Bejaia, Algeria
amira@mail.cerist.dz

Abdelraouf Ouadjaout
LIP6, University Pierre and Marie Curie, Paris, France
abdelraouf.ouadjaout@lip6.fr

Abdelouahid Derhab
Center of Excellence in Information Assurance (CoEIA),
King Saud University, Riyadh, Saudi Arabia
abderhab@ksu.edu.sa

Nadjib Badache
Research Center on Scientific and Technical Information
CERIST, Algiers, Algeria
badache@mail.cerist.dz

ABSTRACT

Web applications use authentication mechanisms to provide user-friendly content to users. However, some dangerous techniques like session fixation attacks target these mechanisms, by making the legitimate user use a session identifier that is controlled by the attacker. In this way, he can then impersonate the legitimate user without the need to know his credentials. In this paper, we present SAWFIX, a PHP static analyzer that checks web applications for session fixation vulnerabilities. To the best of our knowledge, SAWFIX is the first analyzer that checks exhaustively for this type of vulnerabilities, while the other methods only ensure partial correctness that is limited to a fraction of possible executions. SAWFIX is based on abstract interpretation, which is a theory for approximating the semantics of programs and allows designing static analyzers that are fully automatic and sound by construction. We implemented a prototype of our approach and tested it on several complex web applications. We obtained promising results in terms of detection accuracy and processing time, which reflects the efficiency of our system.

Keywords

Session fixation attacks; Web application security; Static program analysis; Abstract interpretation

1. INTRODUCTION

Nowadays, business activities and governments are relying more and more on web technology. The ease of implementation and deployment of web applications have made them ubiquitous and unavoidable, whether in e-commerce sites, in intranet/extranet applications or in Internet services. These applications are however increasingly targeted by malicious users. Indeed, a web application is accessible from anywhere on the network, which makes it easily exposed to the large panoply of existing security attacks.

Among the countermeasures that protect a web application from these threats, authentication is generally the first security shield considered during the design process. In the web paradigm, the authentication service relies on the concept of session management, which is principally based on SIDs (Session Identifiers). Session fixation attacks target these session management systems by tricking a legitimate user to use a SID controlled by a malicious user. This can be done either by providing a malicious URL that includes a SID to the victim, or by using other techniques such as XSS attacks.

To defend against this type of attacks, previous works [3, 4, 5] either detect the vulnerability by dynamically monitoring session values or protect the users by introducing additional SIDs that complement the already existing authentication mechanism. However, dynamic approaches are limited, in the sense that only certain attack scenarios are tested. On the other hand, more deployed SIDs add another layer of security but do not guarantee the attack prevention. The main common drawback of all these methods (like: IBM Security AppScan tool [1]) is that they do not check all the execution traces.

In this paper, we propose a new static analysis of web applications that verifies the immunity of the program against session fixation attacks. The novelty of our approach is twofold. First, the analysis is performed statically by operating directly on the source code of the application, which helps developers check their code base without requiring particular configuration of the runtime environment. Second, in contrast to other methods, it performs an exhaustive search of all possible executions. Indeed, our analysis is based on the theory of abstract interpretation [2] that provides a rigorous mathematical framework for analyzing efficiently the entire search space of program executions in finite time. The main benefit of this theory is that it provides sound verifi-

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

CODASPY'17 March 22-24, 2017, Scottsdale, AZ, USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4523-1/17/03.
DOI: http://dx.doi.org/10.1145/3029806.3029838

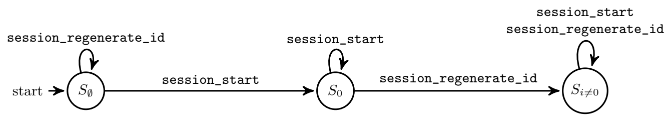

Figure 1: Session states and the effect of PHP session management functions.

cation, which is guaranteed to find a vulnerability (in case one exists). It can also prove the absence of vulnerabilities, but might be subject to false alarms.

2. APPROACH DESCRIPTION

The basic idea of our approach is to detect the moment when the SID does not change after a successful authentication. Indeed, the attacker can control the legitimate user's initial SID. If this SID does not change after the authentication step, the attacker can use it to impersonate the user.

To track the occurrence of such situation statically, it is necessary to go through all execution paths that can be taken by the program. However, due to the undecidability of program verification, such precise enumeration can not be performed in finite time in general. To this end, the theory of abstract interpretation has been proposed in order to avoid the explosion of the size of the search space by abstracting away some of the details of the program semantics. In this section, we present a succinct description of the developed abstractions that are tailored to the verification of session fixation attacks in PHP web applications.

2.1 Abstraction of sessions

The first abstraction is related to the dynamic state of the session during execution. In PHP web application, the SID follows a specific workflow depicted in Figure 1. We can distinguish between three abstract session states that we denote by $\mathcal{S} = \{S_\emptyset, S_0, S_{i\neq 0}\}$. Initially, a web application is in the uninitialized state $S_\emptyset$. After that, the function *session_start* creates a new session or resumes the current one, which corresponds to the state S_0. Finally, the state $S_{i\neq 0}$ corresponds to the generation of a new SID during a session that has already been started.

This flow corresponds to the evolution of a single execution path. In abstract interpretation, we need to develop a computable abstraction of a set of executions in order to scale the analysis to possibly infinite search spaces. Consequently, we define our first abstraction $\mathcal{D}^\sharp_\mathcal{S}$ as the powerset lattice: $\langle \wp(\mathcal{S}), \subseteq, \cup, \emptyset, \mathcal{S} \rangle$

which corresponds to all possible combinations of the states of SIDs. In the sequel, we denote these abstract elements as $Empty = \{S_\emptyset\}$, $Init = \{S_0\}$, $Regen = \{S_{i\neq 0}\}$, $Noempty = Init \cup Regen$, $Noregen = Empty \cup Init$ and $Noinit = Empty \cup Regen$. In Figure 2, we show the corresponding Hass diagram. In addition, we need to abstract also the effect of PHP session management functions over the elements of $\mathcal{D}^\sharp_\mathcal{S}$, which is summarized in Figure 3.

2.2 Abstraction of classes

The second abstraction encapsulates the mappings between variables and classes. Indeed, we need to know for each variable of type object, the possible class whom it belongs to. So, when a method call is encountered, we can decide which class methods code to analyze. Formally, we define the abstract class domain $\mathcal{D}^\sharp_\mathcal{C}$ as:

$$\langle \mathcal{V} \to \wp(\mathcal{C}), \dot{\subseteq}, \dot{\cup}, \lambda v.\ \emptyset, \lambda v.\ \mathcal{C} \rangle$$

where $\mathcal{V}$ represents the set of the program variables and $\mathcal{C}$ the set of classes. All lattice operators, such as order and join, are defined pointwise.

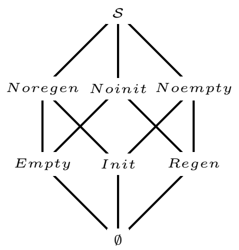

Figure 2: The lattice of abstract sessions.

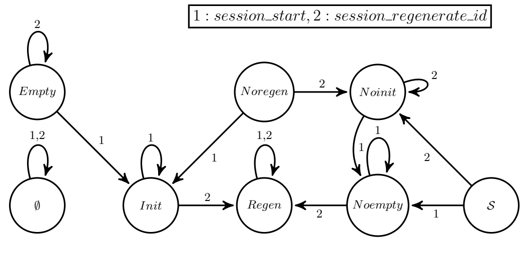

Figure 3: Abstract session values and the effect of PHP session management functions.

2.3 Put it all together

The final abstraction $\mathcal{D}^\sharp = \mathcal{D}^\sharp_\mathcal{S} \times \mathcal{D}^\sharp_\mathcal{C}$ is a simple product of the previous two domains. Two important facts should be noted about $\mathcal{D}^\sharp$. First, this domain is non-relational, which is due to the use of the cartesian product that removes the relation between session states and classes. Nevertheless, this level of abstraction is sufficient to successfully verify the web applications in our experiments.

Second, no information is preserved about the values of program variables. This implies that some spurious execution paths may be analyzed, for example when reaching conditional statements. Indeed, to ensure the soundness of the analysis, both branches of the if statement need to be analyzed even if in fact only one branch is taken by the program in a real execution. However, there exist many abstract domains dealing with this problem that can be easily integrated in our analysis.

In order to verify if the program is vulnerable to a session fixation attack, we check the session's abstract value at the authentication instruction: the elements $Empty$, $Regen$ and $Noinit$ reflect safe states. In this case, we are sure that the application is not vulnerable. The other values (except for $\emptyset$) are associated with dangerous states. So, the application may be vulnerable.

3. SYSTEM DESIGN

The general design of SAWFIX is given by figure 4. It is based on Phc[1], which is an open-source compiler for PHP.

[1]https://github.com/pbiggar/phc

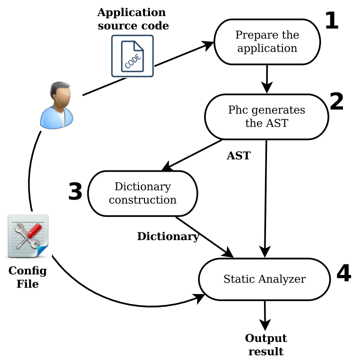

Figure 4: SAWFIX design.

It consists of four steps:

First, we prepare the application to be processed by Phc. This compiler permits the generation of an Abstract syntax tree (AST) but does not support all of the PHP's functionalities, including dynamic inclusions and some inclusion functions. A dynamic inclusion is a case where the inclusion parameter contains a variable, for example, the command `include($basepath. "File.php")` receives a variable as a parameter. SAWFIX replaces the dynamic parts by static ones as follows: We start by obtaining the file names from the static portion of the inclusion. Then, we operate a recursive search on the application's sub-directories. Once the list of all the files is obtained, we merge their code to get a unique and a representative file. Finally, each instance of the dynamic parameter is replaced by the full path of the newly generated file. The two inclusion functions `include_once` and `require_once` are not supported by Phc. We replace them respectively by two functions: `include` and `require`. As this operation can generate loops in the application code, we move all the instances of these inclusions to the start of the entry web page.

In the second step, Phc is used to generate the AST of the prepared code.

In the third step, SAWFIX analyses the application's AST and builds what is called a function dictionary. The latter stores function (or method) names, the mapping of functions to the different classes, and for each map its physical address in the AST. By using the dictionary, SAWFIX can get the physical address of the code (function or method) in the AST without performing other operations.

In the last step, SAWFIX applies the approach described in Section 2 and analyses statically all the web application's AST. However, we want to know the session state on the lines that define a successful authentication (which are in the form of: `$_SESSION["logged_in"]= 'yes'`). To print the session state at a specific line of code. The user can either add the directive `__print__()` in the application's code or create a configuration file. The file must contain the name of the session's variable and its value (for example: `logged_in, yes`).

4. PRELIMINARY RESULTS

To demonstrate the efficiency of our system, we tested SAWFIX on multiple web applications of different complexities, ranging from 1K to 110K of lines of code. Some of these applications are vulnerable to session fixation attacks and our system successfully reported the vulnerabilities. The

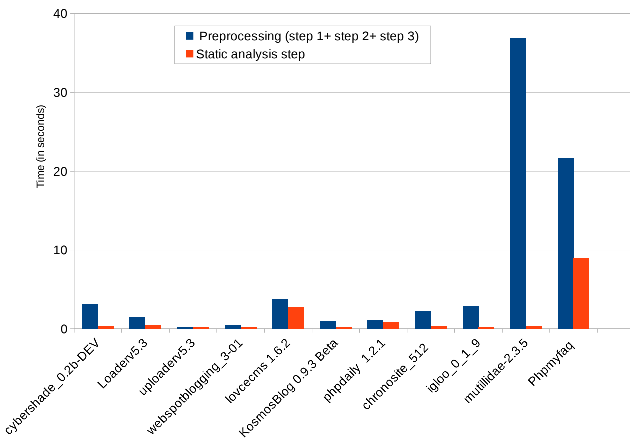

Figure 5: Processing time

processing time results are illustrated in Figure 5: the applications are shown in an ascending order with respect to the number of lines of codes. The obtained results show that SAWFIX can process an application (i.e., three preprocessing steps + static analysis step) in less than 40 seconds.

5. CONCLUSION AND FUTURE WORK

We presented a sound approach that checks for session fixation vulnerabilities in PHP based web applications. We also leveraged our approach and developed a static analyzer SAWFIX. We tested our prototype on a variety of complex web applications and obtained promising results in terms of detection accuracy and processing time. As future work, we plan to include variable tracking, inheritance and recursion into our analysis. It will also be interesting to test more large-scale open source web applications (up to 500K loc).

6. REFERENCES

[1] IBM Security AppScan. http://www-03.ibm.com/software/products/en/appscan.

[2] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[3] P. De Ryck, N. Nikiforakis, L. Desmet, F. Piessens, and W. Joosen. Serene: self-reliant client-side protection against session fixation. In *Proc. 12th IFIP WG 6.1 international conference on Distributed Applications and Interoperable Systems*, DAIS'12, pages 59–72. Springer-Verlag, 2012.

[4] M. Johns, B. Braun, M. Schrank, and J. Posegga. Reliable protection against session fixation attacks. In *Proc. 2011 ACM Symposium on Applied Computing*, SAC '11, pages 1531–1537. ACM, 2011.

[5] Y. Takamatsu, Y. Kosuga, and K. Kono. Automated detection of session fixation vulnerabilities. In *Proc. 19th international conference on World wide web*, WWW '10, pages 1191–1192. ACM, 2010.