SESSION DATA PROTECTION USING TREE-BASED DEPENDENCY

G. Shruthi¹, Jayadev Gyani², R. Lakshman Naik³, G. Sireesh Reddy⁴
¹,²Department of CSE, JITS, Warangal, A.P, India
³,⁴Department of CSE, BITS, Warangal, A.P, India

ABSTRACT
Web applications have become very popular in nowadays. Web applications can be error prone and easily exploited by attackers, because of the implementation vulnerabilities. Securing web applications against implementation vulnerabilities is very important. Existing security solutions do not provide adequate support to protect web applications against broken session data dependencies in this paper we focus on one specific type of implementation vulnerability, namely broken dependencies on session data along with session data dependent vulnerabilities. This can be lead to a variety of erroneous behavior at runtime. This paper shows how to guarantee the absence of runtime errors due to broken session data dependencies. The proposed solution provides the tree-based dependency to prove no-broken-data-dependencies property. A framework named PP4Wap (provable protection for web application) which will increase the reliability and security of data-centered web applications.

KEYWORDS: Security, Reliability, Data sharing, Web application, PP4Wap, Vulnerabilities, WAF.

I. INTRODUCTION
Now a day’s almost of us use web application. Almost all of the services are provided via the World Wide Web (WWW) [1]. Without WWW we can’t do anything easily. More and more organizations strongly depend on their correct functioning. Web application tends to be error prone and implementation vulnerabilities are readily and commonly exploited by attacks. So, to improve the reliability and security of web applications is most important in the field of software engineering and security engineering. This paper introduces a tree base dependency concept in session control. It is a protection approach against web application vulnerabilities related to session data dependencies. To prove this we have to take one specific implementation bug i.e. runtime errors due to broken data dependencies in data-centered applications. This is an application level vulnerability.

Current technologies such as anti-virus software programs and network firewalls provide the secure protection at the host and network levels, but not at the application level. Existing solution for the broken session data dependencies is Web Application Firewall (WAF) [2]. WAFs are applied to mitigate a range of vulnerabilities, including vulnerabilities to forceful browsing. But a malicious user will typically apply forceful browsing to exploit implementation-specific broken session dependencies in data-centered web applications in a more or less controlled way. So, existing security solutions do not provide adequate support to protect web applications against such implementation-specific bugs. Proposed solution combines development-time program annotation static verification and runtime checking and introduces tree-based dependency in session control to prove no-broken-data-dependency property.

The rest of this paper is structured as follows: Section 2 provides some background information on indirect data sharing and broken session data dependencies in data-centered Web applications. In Section 3, the presented work is related to existing research in program verification and Web security.
Section 4 defines the requirements for a solution. In Section 5 we present our solution to detect and prevent composition problems in data-centered Web applications. Section 6 defines the implementation of the tree-based dependency concept. Section 7 presents the result of the paper. Section 8 summarizes the conclusion of this paper.

II. BROKEN SESSION DATA DEPENDENCIES

This section provides more detailed information on how to break the session data in data-centered applications and why WAF fails to protect web applications against broken data dependencies [3].

2.1. Indirect data sharing

Indirect data sharing is adopted in several component models and APIs such as JavaServelet containers [4], Pluggable Authentication Modules framework [5] and Java Spaces in sun’s Jinni [6]. Generally web-applications are server-side applications that are invoked by this Web Clients (browsers), typically using the Hyper-Text Transport Protocol (HTTP).

The HTTP is a stateless application-level request/response protocol and has been in use on the WWW since 1990[7]. Since the protocol is stateless, each request is processed independently without any knowledge of previous requests. To enable the typical user’s session concept in web applications, the web application needs to add session management on top of the stateless HTTP layer. Different techniques exist to embed web request within a user session such as the use of cookies, URL rewriting or hidden form fields [8]. Out of these three methods cookies is a best method, because cookie functionality is built in to most browsers. Thus no special coding is required to ensure session ID information is embedded within the pages served to the client browser.

In the repository architectural style [9], a system consists of a central data structure and set of components interacting with the repository. The data-centered application contains this architectural style.

In data-centered applications each component can indirectly interact with the other component, but all the components can directly interacting with the shared repository. The data centered applications are correctly composed with respect to the indirect data sharing if, at run time, each component is able to retrieve the data from the repository that it expects to find. Thus the correct functioning of a component depends on the run time state of the repository. At the time of execution, implicit semantically dependencies exist between the components and shared repository.

2.2. Broken data dependencies

Breaking data dependencies is a common risk in composing data-centered applications. We identified two common composition mismatches or exceptions in data centered applications. These are as follows:

2.2.1. NullPointerException

A data item is not available on the shared repository although a reading component expects it on the repository during execution.

![Fig. 1. Data dependencies (DP) in data-centered applications](image)
2.2.2. ClassCastException

The type of an available data item does not correspond with the type expected by the reading component.

These mismatches lead to the runtime errors. Due to these runtime errors and loose coupling the data can break in data-centered applications. These broken data dependencies undermine the reliability and security of the web applications. Existing solution for this vulnerability is WAF, because this is an application level vulnerability.

To counter web application vulnerabilities, Web Application Firewalls operate on the application layer and analyze web request between a browser and the web server [10]. Web application firewalls are applied to mitigate a range of vulnerabilities, including vulnerabilities to forceful browsing. By passing intended application flow in a web application can generally lead to unauthorized access to resource or unexpected application behavior [11]. Moreover, a malicious user will typically apply forceful browsing to break the session data dependencies in data-centered applications. Therefore existing security solutions do not provide adequate support to protect web applications. So, we need additional support for this. In proposed solution we have to take one composition property to reduce the runtime errors caused by broken data dependencies i.e. no-broken-data-dependencies [18].

III. RELATED WORK

The work presented in this paper is related to a broad spectrum of ongoing research. We only present some key pointers for each of the domains and, in more detail, for the domain most related to the proposed solution, namely, static and dynamic verification in Web application security. Several implementation-centric security countermeasures for Web applications have already been proposed [12], [13], but most of them focus on injection attacks (SQL injection, command injection, XSS, etc.) and use tainting, pointer analysis, or data flow analysis. Our solution targets another set of implementation bugs, namely, bugs due to broken data dependencies on the server-side session state, and, to do so, we rely on the static and dynamic verification of component contracts. Gould et al. also aim to reduce the number of runtime errors in Web applications by applying static verification [14]. Their solution focuses on the reduction of SQL runtime exceptions and uses a static analysis tool to verify the correctness of all dynamically generated query strings within an application. Our solution is based on program annotations and we verify interactions between components and the non-persistent server-side state.

We combine in our solution static and dynamic verification to reduce the runtime enforcement overhead. The idea of combining static and dynamic verification is not new and has, for instance, already been adopted by Huang et al. in securing Web applications against Web vulnerabilities caused by insecure information flow, such as SQL injection, XSS, and command injection [15]. Their approach uses a lattice-based static analysis algorithm for verifying information flow based on type systems and the type state. The sections of the code considered vulnerable are automatically instrumented with runtime guards. In contrast, our approach aims to reduce runtime errors due to composition problems. In addition, our approach is based on program annotations and the verification of component preconditions.
In [16], generate bypass tests that check if an online Web application is vulnerable to forceful browsing or parameter tampering attacks. The bypass tests are black-box tests using data that circumvents client-side checks. They define three levels of fault injection: bypass tests at the value level, at the parameter level, and at the control flow level. Since the fault injections are based on violations of the client-side validations, they operate independently of the server implementation and do not give formal guarantees about the absence of bugs. In contrast, our verification approach is able to guarantee the absence of errors at the flow level and, in future work; we would like to investigate how well our approach is suited to counter errors at the other two levels as well.

Firewall configuration analysis is proposed to manage complex network infrastructures (such as networks with multiple network firewalls and network intrusion detection systems) [17]. These approaches aim to achieve efficiency and consistency between the different network layer security devices, whereas our approach focuses on the application-layer consistency between the WAF and the Web application.

For more than a decade, software architectures have been used to abstract reasoning about software systems from the source code level toward coarse-grained architectural elements and their interconnections [9], [19], [20], and [21]. Architectural styles abstract reoccurring patterns of components, connectors, and behavioral interactions within different software architectures and try to capture the advantages or main characteristics of a particular architectural style, as well as the constraints introduced by the style. In [9], Shaw and Garlan proposed taxonomy of different architectural styles, including the data-centered style.

To support architecture-based reasoning, (semi)formal modeling notations and analysis techniques are required. Several Architecture Description Languages (ADLs) are proposed for architectural specification and analysis. Although these ADLs strongly vary in the abstractions and analysis capabilities they provide, most ADLs explicitly provide abstractions for components, connectors, and their behavioral interactions, as well as tool support for analysis and architecture-based development [21]. However, in most cases, a discontinuity exists between the architectural model and the actual implementation model, making the outcomes of architectural analysis meaningless. To counter this, ArchJava [22] offers a unique binding between architectural description and actual implementation, but ArchJava does not yet provide indirect sharing verification.

Component contracts already were often proposed before for various purposes [23], [24]. For components written in Java, JML [25] is a popular formal contract specification language. The use of JML or related languages such as Spec# [31] for verifying component properties is a very active research domain. For example, Jacobs et al. [26] verify the absence of data races, and Pavlova et al. [27] focus on the security properties of applets. Other applications of JML are surveyed in [28].

The research presented in this paper proposes a pragmatic solution to broken session dependencies in Web applications. The main advantage of such pragmatism is the potential for short-term applicability. However, of course, research on more fundamental approaches is also needed and can have a more substantial impact in the long term. There is a large body of research on how to improve programming languages for programming distributed applications. In his keynote speech at ICSE 2005, Cardelli discussed three important areas where improvements are important: asynchronous concurrency, dealing with semistructured data and additional security abstractions [29]. The programming language E [32] is an example of a language that has emphasized security in its design. Other languages focus on specific classes of Internet applications such as distributed consensus applications [30].

IV. REQUIREMENTS

In particular, this paper eliminates certain types of runtime errors (such as a NullPointerException or a ClassCastException) by giving a formal guarantee that the no-broken-data-dependencies property is not violated in a given composition.

Reducing certain types of runtime errors by formally verifying that a given composition does not violate the desired composition property certainly improves the reliability of the software composition, but, in order to be really useful, the following interoperability and usability criteria are important as well:
4.1. Interoperability

It is important that the proposed solution is interoperable with the existing Web infrastructure and does not interfere with other Web security solutions.

4.2. Usability.

In order to encourage wide adoption by developers, we also identified two important usability characteristics for the solution:

4.2.1. Limited overhead.

In order to be generally applicable, the introduced overhead for the software developer and software composer must be minimal, both in terms of additional workload and verification time.

4.2.2. Applicability to real-life applications.

The applicability of the proposed solution may not be limited to toy examples, but the proposed solution must also be more generally applicable to larger real-life applications.

V. OVERVIEW OF THE SOLUTION

In this section we specify tree-based session dependency and use static and dynamic verification to guarantee that no-broken-data-dependencies property. Tree-based dependency proves that no-broken-data dependencies property in two steps.

5.1. Session control testing

We are able to read the D page only after reading the B and A pages. Without accessing B and A pages we can’t access a page D. This illustrates the preventing broken data dependencies property and it is preventing the NullPointerException.

5.2. Session expiration testing

Generally session will expire with the fixed time. So we should access the pages only within the limited time, otherwise the session will expired. Here the page C is already accessed but the session of that page has been expired. So, we are unable to access the F page even if the C page has been accessed. This will provide the security.

Next to the implementation, the deployment information and the runtime Web traffic are also used as input for our verification process.

The verification process consists of three steps. First, the interactions with the shared session repository are explicitly specified in component contracts and static verification is used to verify that each component implementation obeys its contract specification. Second, the no-broken-data-dependencies property is verified in each possible execution path within a user’s session. To verify this property statically, an upper bound is defined for the client-server interactions, namely, the intended client-server protocol. Next, the property is verified under the assumption that the client-server interactions are prefixes of the intended client-server protocol. Finally, runtime policy enforcement is used to guarantee that only the Web requests that are prefixes of the intended client-server protocol are processed by the Web application. By combining these three verification steps, our solution ensures the no-broken-data-dependencies property in a given application.

We will now discuss each of the three steps in more detail in the following sections.

Step1:- Server-Side Specification and Verification

In order to specify a component’s interactions with the shared session repository, each Web component is extended with an appropriate component contract. The contract is expressed in a problem-specific contract language, which is easy to understand for application developers.

Step2:- Application-Specific Property Verification

The no-broken-data-dependencies property is verified by checking all of the possible execution paths in a user’s session. To verify the property statically, an upper bound is defined for the client-server interactions, namely, the intended client-server protocol. This is an upper bound for the nondeterministic interactions between the client and the server and includes all valid client-server
interactions that may occur in the application under normal circumstances. The intended client-server protocol can be expressed in various ways, such as a regular expression, an EBNF notation, or a labeled state transition system.

**Step3:- Runtime Protocol Enforcement**

Finally, since the no-broken-data-dependencies property is verified under the assumption that all Web requests obey the intended client-server protocol; this assumption needs to be enforced at runtime. This can be done by loading the protocol specification into a supporting WAF or extending the application with an appropriate filter. As a result, only prefixes of the intended client-server protocol are allowed to be processed by the Web application.

**VI. IMPLEMENTATION**

In this part, we have to take one session control demo to prove the composition property. The composition property is as follows.

**6.1. No-broken-data-dependencies**

No client request causes a data item to be read from the server-side shared session repository before it has actually been written. For each shared data read interaction, the shared data item that already has been written to the shared session repository is of the type that is expected by the read operation.

By using session control demo we can prove the no-broken-data-dependencies property. To prove this property we have to use tree-based dependency in session control demo.

**6.2. Tree-based-dependencies:**

In this we can access session or page depend on the trees. The basic idea of this is as follows: All sibling nodes are accessed only when from root node to sibling pages on the tree are accessed.

This tree-based dependency is used to prove the no-broken-data-dependency property. In this we have two steps.

First: we need to write session properties in the session control demo. We used these properties to write the tree-based algorithm. By using this algorithm we can see the process of accessing session pages. So first we need to write the session properties.
Second: next we have to write the session time out and data within the web.xml file because, based on this time the session will run. That means we can access the pages only within this time. Otherwise the session will expire. Then we can’t access the pages.

```xml
<Session-config>
  <Session-timeout>30000</Session-timeout>
  ...
</Session-config>
```

We are able to read the F page only after reading the B and A pages, and G page only after reading the C and A pages. This illustrates the preventing broken data dependencies property i.e. it is preventing the NullPointerException. Based on this session control demo I will implement the algorithm.

```
Algorithm Tree_based(x)
    X ← node
    S ← sibling node
    P ← sibling node
    R ← sibling node
    If S ← X then
        {  
            P ← S
            R ← P
            R ← accessed
            P ← accessed
            S ← accessed
        }
    else
        R ← x
        R ← accessed
```

This algorithm proves the tree-based dependency concept. In this we are testing the node or page in two cases. First, if the node can be sibling node, then we have to access the nodes from root to sibling. Second, if the node can be root node, we can directly access that node.

VII. RESULTS

We used tree-based verification experiment to measure the cost and verification performance of the proposed solution.

7.1. Tree-based verification performance

To evaluate the performance of the tree-based dependency verification process we used PPRWap framework. PPRWap framework is a provable protection for web applications. PPRWap framework is a software framework, in computer programming is an abstraction in which common code providing generic functionality can be selectively overridden or specialized by user code providing specific functionality.
Generally tree-based techniques are easy to understand in any type of applications. By using tree and graphs we can easily understand session dependencies. So instead of sequential dependency, tree based dependency is better generally sequential dependency is very expensive and not easy to understand. So we used this tree-based dependency concept in session control demo with the help of the framework. The figure 2 shows the graphical representation between the sequential and tree-based dependency.

VIII. CONCLUSION

In this paper, we have presented an approach to prove that no-broken-data-dependencies property i.e. Tree-based dependency. This proof shows that the security and reliability of Web applications can be increased by guaranteeing in the absence of runtime errors. In particular, we have proposed a solution to prevent runtime errors due to broken data dependencies on session data. Our solution combines development-time program annotation, static verification, and runtime checking to provably protect against broken data dependencies in Web applications. We designed and developed a framework PP4Wap on session data. This will increase the security and reliability of data-centered Web applications.

Our solution also provides a good trade-off between usability and verification power. In this paper is to improve Web application security by providing an appropriate solution to the specific problem of broken data dependencies on session data.

REFERENCES


Authors
Guda Shruthi received her B.Tech. in Computer Science and Engineering from Dr. VRK college of Engineering, Karimnagar, A.P, India and M.Tech. in Software Engineering from JITS, JNTU, Warangal, A.P, India. Currently, she is an Assistant Professor in the Department of Computer Science and Engineering, BITS, Warangal, A.P, India. She has more than 5 years of experience in teaching. Her research area of interest includes Computer networks, software engineering and neural networks.

Jayadev Gyani received his M.Tech. in Computer Science and Engineering from Osmania University, Hyderabad, India and Ph.D. in Computer Engineering from Central university of Hyderabad, India. Currently, he is a Professor in the Department of Computer Science and Engineering, JITS, Warangal, A.P, India. He has more than 16 years of experience in teaching. He also has Publications and presented more than 27 papers.
R. Lakshman Naik received his B.Tech. in Electronics and Telecommunication from SIET, JNTU, Krishna, A.P, India and M.Tech. in computer science and engineering from BITS, JNTU, Warangal, A.P, India. He also served as a System’s Engineer in Wipro Technologies. He has Publications more than 7 papers in area of computer networks, Data mining and neural networks.

G. Sireesh Reddy received his M.Sc. in computers from KNR PG & Degree College Warangal, A.P, India and M.Tech. in computer science and engineering from BITS, JNTU, Warangal, A.P, India. He also served as a Software Engineer in Syntel India Ltd. His research area of interest includes computer networks.