DEMO: Secure and Customizable Web Development in the SAFE Activation Framework

Raphael M. Reischuk
Saarland University
Saarbrücken, Germany

Florian Schröder
Saarland University
Saarbrücken, Germany

Johannes Gehrke
Cornell University,
Ithaca, New York, USA

ABSTRACT

We propose a demonstration of SAFE with some of its newest security features. SAFE is a framework for modern Web application development with automated state consistency, enforced security at various levels, and design for Web personalization and extensibility.

With the emerging complexity in (extensible) data-driven Web application development, in particular in terms of consistent data management with multiple clients (many Facebook users), ownership preservation (various Facebook user items with individual intellectual property), and data privacy (sensitive Facebook user data), we believe a demo of a comprehensive data-centric and secure Web application framework with declarative specifications for many modern Web features will be of considerable interest to the security community. In particular, we think it is interesting to see a demonstration of how fast and how intuitive the secure customization of a true multi-tier Web application can be.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance and Enhancement—*Extensibility*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

Keywords

extensibility; web security; access control

1. INTRODUCTION

More and more software is delivered through the Web, following today's cloud idea of delivering software as a service. The code of such rich internet applications is split into client and server code, where the server code is run at the service provider and the client accesses the application through a Web browser. In data-driven Web applications, the state of the application resides in a database system (or simply in a key-value store), and users interact with this persistent state through Web clients, each having an individual (possibly shared) state. More and more users wish to personalize such Web applications, that is, they wish to customize the functionality of a Web application to fit their individual application needs.

In the development, and in particular, in the customization of modern Web applications, it is more challenging than ever before to cope with multiple tiers (various servers, multiple simultaneous clients, replicated databases, mobile devices, etc.), the shared application state spread over multiple devices, with complex security attack surfaces, and with many subtle technical details. Moreover, huge amounts of similar code are often rewritten from scratch. We believe that high-level specifications in a declarative language and compiler-generated application code can significantly simplify and secure the entire Web development process.

We demonstrate the SAFE activation framework with its efficient and elegant way of creating interactive and secure Web applications composed of reusable software components. One reason for the success of SAFE is its intuitive and still expressive declarative modeling language SFW. The rough idea behind specifications at a higher abstraction level in SAFE and the production of computer-generated source code thereof is motivated by the following facts.

First, Web applications often use common patterns which are better taken from well-tested code libraries instead of being manually programmed every time from scratch. Typical replication methods like copy & paste often introduce logical or structural flaws that are hard to detect later. *Second*, human errors in the programming process should generally be detected and avoided at the earliest possible point in time. Such errors introduce not only bugs in the functionality, but might also impose severe security threats. *Third*, a restricted set of allowed operations simplifies the detection of malicious code or even prevents security holes which might even be introduced on purpose. Assume for instance a system-provided (and hence trusted) encryption function for which all randomness is correctly chosen. A malicious programmer using the predefined function cannot bring his own "pseudo-randomness" into play, and hence cannot introduce a backdoor to decrypt confidential user data offline.

The two latter aspects are particularly relevant for data-driven Web applications with sensitive and privacy-critical user data. Not only is the programming of such security checks vulnerable to bugs and attacks, but also is it quite tedious to program all necessary checks, to add error handling for the reactive multi-tier code, to chose encryption/signature keys correctly, to establish database connections, etc.

CCS'13, November 4–8, 2013, Berlin, Germany.

ACM 978-1-4503-2477-9/13/11.
http://dx.doi.org/10.1145/2508859.2512495.

The automated compilation of higher-level languages hence not only produces better program code, but also significantly reduces the workload of a programmer.

We demonstrate how SAFE addresses all the aforementioned aspects and thereby significantly simplifies the overall development process, abstracting from technical details, and preventing malicious code from harming the system or sensitive user data. Moreover, we demonstrate how SAFE achieves several other useful and mandatory properties such as guaranteed state consistency, sandboxes for both users and software components. Furthermore, we show how SAFE enables new ways in the direction of customization and user personalization.

2. SAFE

This section recaps the main features of the SAFE framework [1, 2] for the development of secure Web applications with automatic state consistency and safe extensibility. SAFE provides a hierarchical programming model which naturally builds upon the hierarchical DOM structure of Web pages. The most constitutive components in SAFE are its so-called *f-units* which cluster all code fragments for a specific functionality within a Web page, including the business logic, the visual appearance, and the interaction with users or other f-units. This clustering provides a clear level of abstraction through well-defined *interfaces* for each f-unit. The modularity of reusable f-units relieves the programmer from struggling with variable scopes and their interference.

A Web page is modeled as an *activation tree* in which f-units are organized hierarchically — resulting in a clean abstraction and an elegant way of composing Web pages. The integration of an f-unit `F` in the activation tree is referred to as *activation* of `F`. More precisely, an f-unit is activated by its parent f-unit and thereby receives activation data through well-defined interfaces. The f-unit can use the activation data or other data obtained directly from the database through queries to display parts of the Web page. Every f-unit can also activate other f-units, its child f-units. Whenever an instance of an f-unit is activated, the corresponding compiled HTML / JS / CSS code is made available in the activation tree. Eventually, the activation tree is linearized to a single HTML document by transforming subtrees to nested HTML elements. After the activation tree has been constructed, the corresponding code for HTML / JS / CSS is sent to the client.

Activations are expressed through *activation calls* in the declarative *modeling language* SFW, a straight-forward extension of HTML: all HTML elements and also PHP and JavaScript can be used as in traditional Web application development. Activation calls are at the core of SFW and can as such be used in any HTML context.

2.1 Database updates

Since today's Web applications are not static pages (instead, they contain a lot of reactive code for event-driven modifications of the shared application state), SAFE's methodology redeems the developer from all the technical inconveniences of such updates and thereby automatically maintains state consistency — even for concurrent updates. Assume the client's browser interacts with the delivered HTML page and eventually sends some update request back to the server. The corresponding f-unit in the activation tree processes this

```
1  <h1> Groups </h1>
2
3  The following groups exist:
4  <for query="SELECT gid, name, owner='$%me' AS isowner,
5              ismember FROM groups">
6    $$name
7    <form>
8      <if $$ismember>
9        <input type="button" value="Leave" onclick=
10         "query:DELETE FROM groupmembers WHERE gid='$$gid'
11                 AND uid='$%me'">
12     <else>
13       <input type="button" value="Join" onclick=
14         "query:INSERT INTO groupmembers SET gid='$$gid',
15                 uid='$%me'">
16     </if>
17     <if $$isowner>
18       <input type="button" value="Remove" onclick=
19         "query:DELETE FROM groups WHERE gid='$$gid'">
20     </if>
21   </form>
22 </for>
23
24 <form>
25  <input type="text" name="gtitle"/>
26  <input type="button" value="Create Group" onclick=
27   "query:INSERT INTO groups VALUES ('$#gtitle',$userID)">
28 </form>
```

Figure 1: HTML buttons with concrete DB queries.

request and generates a database query for which SAFE automatically verifies various safety and security properties. These include checks for state consistency, access control, and prevention of code injection. After the query has been executed, SAFE automatically triggers all f-units in the activation tree which have an out-dated state. These f-units are immediately rebuilt and refreshed at the client. The shared application state is automatically synchronized with all connected clients.

SAFE completely alleviates the developer of an f-unit `F` from caring about the freshness of its state while `F` is updating parts of the application state. Moreover, the developer does not have to provide code for partial updates of any f-units in the activation tree. The developer only specifies the *update query* that is supposed to be executed for some event attached to an HTML element inside the f-unit. We will see examples in Section 3.

3. SAFE DEMONSTRATION

We have chosen two demonstration features which nicely show how efficiently and securely the specifications in SAFE are translated to full-fledged Web applications.

3.1 Sandboxing for f-units and users

A new and un-published feature we show in our demonstration is how SAFE provides two independent kinds of *data separation*: every f-unit has its own database tables, which can only be accessed by the f-unit itself; and every user has its own data domain, in which only the user herself can modify data. Dually, SAFE provides mechanisms for *data sharing*: f-units can be wired together and user data can be aggregated and delegated.

3.2 Declarative and secure specifications

We present various new declarative specifications of different functionality and show the result of the compilation of these specifications using the SAFE compiler. The code for our test case applications generated by the compiler has an average blow-up factor of 10.57 compared to the size of the user-specified code. The generated code contains various new consistency and security checks which we believe a user would have to write manually in order to produce Web applications of comparable (high) quality.

As an example, we consider the simple specification of HTML buttons that are linked with the secure execution of database queries. The code snippet in Figure 1 shows the corresponding HTML-like specification in the SFW language. The code is taken from an f-unit to administrate groups in a social network application. The parts colored in blue are SFW-specific deviations from standard HTML. The code shows a list of groups whose entries are extracted from the SELECT query as specified in the <for> tag in line 4. The query selects four data fields each of which is available in the subsequent scope via the SFW syntax $$field, e.g., $$name in line 6, or $$ismember in the <if> tag in line 8. The content inside the <for> block is "executed" (or more precisely: displayed) in the HTML document once for every result tuple of the query execution. Depending on the values of $$ismember and $$isowner, the buttons to join (line 13), to leave (line 9), and to remove a group (line 18) are displayed. In the following, we will focus on the button to create a new group (line 26).

Instead of specifying a JavaScript function for the onclick event of a button, the developer simply specifies the actual database query to be executed. The query for the button in line 26 inserts two values in the specified table. The first value is the user input as specified in the form. SFW offers the syntactic placeholder $#gtitle to represent the value entered in the HTML input element named gtitle. The second value $userID is a dynamic value which is no direct user input, but a standard PHP variable. All such dynamic values occurring in the query need special attention: dynamic values must also be contained in the corresponding form in the HTML document (in order to ensure the dynamically evaluated values end indeed up in the query). However, such dynamic values cannot simply appear in the DOM tree in plaintext since a malicious client could easily modify these values, for instance by replacing the user ID Alice by the user ID Eve. We demonstrate how SAFE automatically ensures the integrity and confidentiality of dynamic values by new suitable security mechanisms.

The treatment of dynamic values is just one point on the list of tasks a traditional Web developer would have to take care of. We demonstrate and justify the blow-up factor of roughly 10 after compilation, for which the following (incomplete) list gives an intuition for the workload in case of a traditional hand-written database update procedure:

1. Create a regular HTML form inside the current HTML document. The protocol (GET, POST, etc.) and the recipient have to be suitably specified.
2. Create a PHP file to answer the submitted form and mention the URI of the file in the form.
3. In the PHP file, authenticate yourself at the database and establish a secure connection.
4. For each variable transmitted via the form, escape special characters to prevent SQL injection attacks.
5. Insert escaped values in the query.
6. Verify that the authenticated user of the application has sufficient permission to execute the query with the current values.
7. Verify that the query can be executed in terms of data consistency, i.e., has the query been issued from a state which is sufficiently fresh?
8. Send query to the database for execution.
9. Process the result, output a status message, and refresh parts of the Web application. Here, it will be necessary to determine the parts of the Web application which must be updated. Hence, all relevant dependencies must be derived.
10. Specify event-driven AJAX code to send the form values to the PHP handler, and to receive the data updates for all corresponding elements in the DOM tree.
11. Implement a comprehensive error handling which takes into account all possible kinds of errors (database connection errors, query execution errors, invalid values, etc.). This error handling has to be specified at all tiers; hence in PHP, JavaScript, and possibly also at the database.

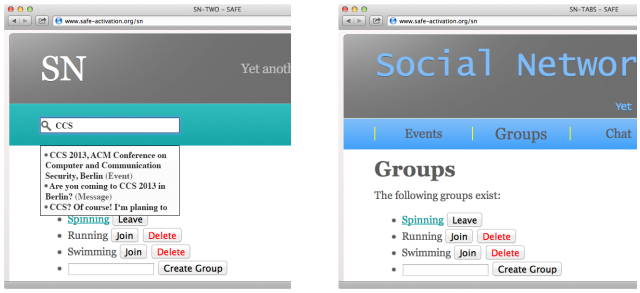

Figure 2: Functional extensions: (left) incremental search engine, and (right) navigation via tabs.

We demonstrate how the state update mechanisms of SAFE implement a superset of the above steps and thereby significantly reduce the burden of the developer down to a concise declarative specification as simple as the one shown in Figure 1. We discuss the compiled code (roughly 248 LOC) for the example in Figure 1 in the demonstration.

3.3 Customization

In the landscape of application ecosystems, today's Web 2.0 users wish to personalize not only their browsers with various extensions or their smartphones with various applications, but also the various extensions and applications themselves. This next generation of platform-independent Web applications is expected to be *customizable* not only in style (how they are displayed), but also *extensible* in functionality (what application logic is exposed).

The key to success in SAFE is the *encapsulation* of functionality within f-units: Every f-unit bundles the code for all tiers (PHP server code, HTML client code, JavaScript client code, CSS client code). We demonstrate the complex integration of a fresh f-unit into an application. We show how the SAFE tool suite easily manages the f-units for our social network application (Figure 2).

Moreover, we demonstrate how SAFE addresses the need for comprehensive personalization and thereby provides various security guarantees. We show how to securely extend an existing social network application in two ways: *First*, we demonstrate how new functionality is added to an existing application. This new functionality constitutes an incremental search engine (Figure 2 left), and as such, it must be integrated deeply into the system. The search engine shall provide the user with a comfortable experience, and hence contains a lot of reactive JavaScript code. Moreover, the search engine needs constraint access to the database with access control policies and a clear description of which fields shall be searched. Finally, the search results have to be presented in a structured and formatted way using PHP, HTML, and CSS. *Second*, we show how to change the navigation inside the social network: the scrolling navigation is turned into a navigation using tabs (Figure 2 right).

4. REFERENCES

[1] R. M. Reischuk. The official SAFE user manual. `http://www.safe-activation.org/`, 2013.

[2] R. M. Reischuk, M. Backes, and J. Gehrke. SAFE extensibility for data-driven web applications. In *WWW '12*, Lyon, France, 2012.