# D-ForenRIA: Distributed Reconstruction of User-Interactions for Rich Internet Applications

Salman Hooshmand, Akib Mahmud,
Gregor V. Bochmann, Muhammad Faheem, Guy-Vincent Jourdan
School of Electrical Engineering and Computer Science
University of Ottawa, Canada
{shooshma, amahmud, mfaheem}@uottawa.ca, {bochmann, gvj}@eecs.uottawa.ca

Russ Couturier
Chief Technology Officer Forensics
IBM Security, USA
russ.couturier@us.ibm.com

Iosif-Viorel Onut
Principal R&D Strategist
IBM Centre for Advanced Studies, Canada
vioonut@ca.ibm.com

## ABSTRACT

We present D-ForenRIA, a distributed forensic tool to automatically reconstruct user-sessions in Rich Internet Applications (RIAs), using solely the full HTTP traces of the sessions as input. D-ForenRIA recovers automatically each browser state, reconstructs the DOMs and re-creates screenshots of what was displayed to the user. The tool also recovers every action taken by the user on each state, including the user-input data. Our application domain is security forensics, where sometimes months-old sessions must be quickly reconstructed for immediate inspection. We will demonstrate our tool on a series of RIAs, including a vulnerable banking application created by IBM Security for testing purposes. In that case study, the attacker visits the vulnerable web site, and exploits several vulnerabilities (SQL-injections, XSS...) to gain access to private information and to perform unauthorized transactions. D-ForenRIA can reconstruct the session, including screenshots of all pages seen by the hacker, DOM of each page and the steps taken for unauthorized login and the inputs hacker exploited for the SQL-injection attack. D-ForenRIA is made efficient by applying advanced reconstruction techniques and by using several browsers concurrently to speed up the reconstruction process. Although we developed D-ForenRIA in the context of security forensics, the tool can also be useful in other contexts such as aided RIAs debugging and automated RIAs scanning.

## Keywords

User-Interactions Reconstruction, Rich Internet Applications, Traffic Replay, HTTP Traces, Incident Forensics

Copyright is held by the author/owner(s).
*WWW'16 Companion,* April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4144-8/16/04.
http://dx.doi.org/10.1145/2872518.2890547.

## 1. CONTEXT AND CHALLENGES

*"Rich Internet Applications"* (RIAs [5]) use JavaScript and Ajax [6] to provide smooth and responsive browser-based Web applications providing end-users with an experience similar to "native" (i.e., non-Web) applications. RIAs have become the norm for modern Web applications. For example, Google has developed most of its major products using this set of technologies (Gmail, Google Groups, Google Maps etc.).

In a Web application, the user's Web browser exchanges messages with the server using the HTTP protocol[1]. This traffic is typically partially or entirely captured by the Web server hosting the application. The traffic can also easily be logged while it goes through the network, for example using a proxy. We call the captured traffic generated by a user during a session with the Web application, the user-session log. This log can be used after the fact to reconstruct the entire user session, for example for forensic analysis after an intrusion has been detected. Imagine that the administrator of a Web server learns that a hacker found and exploited a vulnerability in an Web Application hosted there a few months back. She has to find out what happened and how it happened using the only available resource, the server-generated logs of past user-sessions. This task is straightforward if the Web application is a "classical" (non RIAs) one. Tools such as ClickMiner [7] can help the administrator for this. However, the situation is very different if the Web application is a modern RIAs where many of the requests are generated by script code running on the browser, and the responses typically only contains XML or JSON data used by the receiving script to partially update the current DOM, instead of a replacement for the current DOM. Thus, many of these request/response pairs are part of a series of interactions and cannot be analyzed in isolation. It is thus more difficult to figure out precisely all the steps taken by the user, inferring the inputs provided, the elements that have been clicked etc., when all that is given is the recorded user-session log. Indeed, manual reconstruction would be extremely difficult, time consuming, and up to this point there is no tool that could be used to really deal with this problem. Existing solutions require to either instrument the user's browser, or the Web application itself [1, 2], or are tools based on predefined

[1]`http://www.ietf.org/rfc/rfc2616.txt`

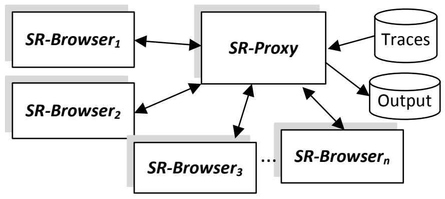


Figure 1: Architecture of D-ForenRIA

patterns of actions [8]. Such solutions are however totally inadequate in the context of security forensics, in which the only available input is the user-session log and the forensics work is done offline without access to the initial Web server or to the user browser.

In order to address these challenges, we have created D-ForenRIA, a tool that will help the administrator recovering after the fact the details of the intrusion using only the available logs. D-ForenRIA is an improvement over a previous version of the tool, ForenRIA [3]. The tool now uses a collection of browsers working concurrently to reconstruct the session more efficiently, and the reconstruction techniques have been improved. For lack of space, the reader is invited to see [3] for more details on the reconstruction techniques and performances. In addition, a companion site has been setup at `http://ssrg.site.uottawa.ca/sr/demo.html` where videos and further information is being made available.

Next, in section 2, we present the main features and architecture of D-ForenRIA. We then introduce in Section 3 the demonstration scenario that will be proposed to conference attendees and present some experimental results.

## 2. MAIN FEATURES OF THE SYSTEM

D-ForenRIA has been implemented following two important design goals from aforensic point-of-view [4]: firstly, the forensic analysis should be sandboxed. This implies no or minimal connection to the Internet during the analysis. Secondly, the reconstructed pages should be rendered as closely as possible to how the suspect viewed them at the time of incident. We achieve the first goal by doing a totally offline replay of the traffic. We achieve the second goal by preserving the correct state of the DOM during the reconstruction process.

Given a user-session log as an input, D-ForenRIA recovers automatically and efficiently the following information:

- *User actions*: What was the precise sequence of "actions" performed by the user during the entire session (clicks, selections etc.), and what was the XPath of the exact elements of the DOM were these actions taken;
- *User inputs*: What exact inputs were provided by the user during the session, and in which fields were these inputs provided;
- *DOMs*: What was the series of DOMs that appeared in the user's browser during the session, including information such as the values and parameters of cookies;
- *Screen shots*: What series of screens were displayed to the user.

D-ForenRIA is based on a distributed architecture, where a number of browsers, called *SR-Browsers*, interact with a module called *SR-Proxy* to concurrently reconstruct the user-interactions. Figure 1 presents the general architecture of D-ForenRIA, with the following main components:

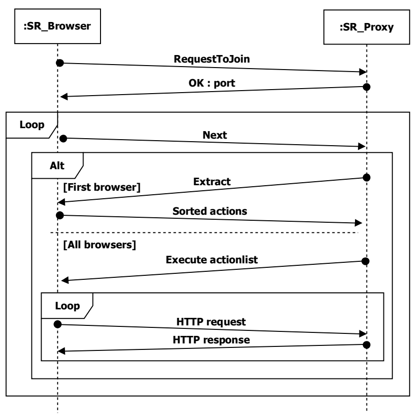
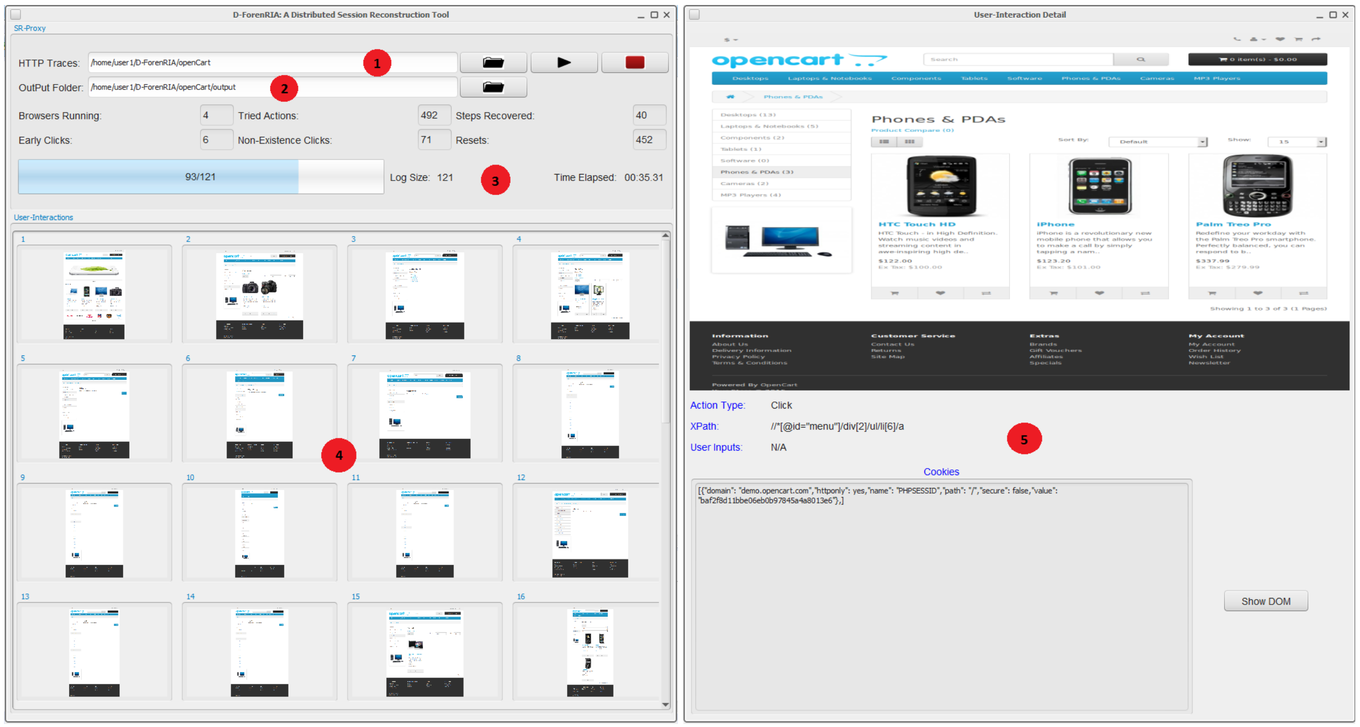


Figure 2: Sequence diagram of messages between SR-Browser and SR-Proxy

1. The user-session access logs are provided as an input to the *SR-Proxy* module.
2. D-ForenRIA is composed of a distributed set of *SR-Browsers* that can dynamically be added/removed during the session reconstruction process. Throughout the reconstruction, the *SR-Proxy* maintains a set of user-interaction candidates (e.g., a set of clickable elements, form submission, field submission) to move the reconstruction further by one step. Each *SR-Browsers* is repeatedly assigned the next of these candidates to execute on its DOM concurrently to the other browsers, until one of them finds the correct action, at which point the reconstruction proceeds to the next step. To do so, each *SR-Browser* instruments the DOM by overwriting a base of JavaScript functions (e.g., *addEventlistener* and *setTimeout*) in order to fully control what happens next.
3. The *SR-Proxy* plays the role of the original Web server, and attempts to respond to the stream of requests sent by each *SR-Browser* as they attempt to execute their assigned user-action. When one of the browser executes the correct action, the proxy receives a set of requests that match the recorded ones and provides the corresponding responses. Otherwise, an error is returned and another action will be assigned to the browser. The *SR-Proxy* also ensures that all the browser are on the last known good state before assigning the next candidate action.

**Interactions between SR-Browser and SR-Proxy:** Figure 2 presents the sequence of messages exchanged between a *SR-Browser* and the *SR-Proxy.* At each iteration, *SR-Browser* sends a *"Next"* message asking *SR-Proxy* the action to do next. The *SR-Proxy* asks the first browser reaching the current state to send the list of all possible actions on the current DOM (using the *"Extract"* message). This list is sorted from the most promising to the least promising action[2]. After this, and while working on that same state, the *SR-Proxy* assigns a new candidate action to each *SR-Browser* that sends a *"Next"* message, along with all the required instructions to reach that state (using an *"Execute"* message). As the browsers execute known or new actions,

[2]See [3] for details about this sorting.



HTTP Traces: /home/user1/D-ForenRIA/openCart

OutPut Folder: /home/user1/D-ForenRIA/openCart/output

Browsers Running: 4 Tried Actions: 492 Steps Recovered: 40

Early Clicks: 6 Non-Existence Clicks: 71 Resets: 452

93/121 Log Size: 121 Time Elapsed: 00:35.31

Action Type: Click

XPath: //*[@id="menu"]/div[2]/ul/li[6]/a

User Inputs: N/A

```
[{"domain": "demo.opencart.com","httponly": yes,"name": "PHPSESSID","path": "/","secure": false,"value":
"baf2f8d11bbe06eb0b97845a4a8013e6"},]
```

Figure 3: Main interface of D-ForenRIA

they generate a stream of HTTP requests that the proxy responds using the recorded log ("HTTP Request" / "HTTP Response" loop).

As mentioned earlier, the *SR-Browsers* can be added or removed during the reconstruction process. When a *SR-Browser* asks *SR-Proxy* to join the session reconstruction process, it is simply added to the pool of browsers and assigned the next candidate action when it sends the *"Next"* message. On the other hand, if a browsers leaves the system, the remaining actions will simply be distributed among the remaining browsers. D-ForenRIA can complete the session reconstruction process even with a single *SR-Browser*.

**Other features of the system**: In addition to a distributed architecture, D-ForenRIA utilizes other techniques to make the session-reconstruction practical and efficient; It handles differences between record and replay such as temporal headers and random parameters in the URLs. It also finds user-inputs provided in forms and fields by matching the input elements in the DOM and data in the traces. Most importantly, D-ForenRIA uses several techniques to find the most promising candidate user-actions in the current state. Finally, it continuously learns about the system as the session is being reconstructed to continually optimize its choice of user actions; The reader is referred to [3] for detailed explanation of the basic version of D-ForenRIA.

**Implementation**: To implement D-ForenRIA's *SR-Browser*, we have used *Selenium*[3]. Selenium, provides an API to control different browsers which enables us to reconstruct user-sessions with a browser matching the one used originally (e.g. Firefox, Chrome, Internet Explorer...) It also provides us with the ability to trigger the execution of JavaScript events and code, recreate the DOMs, capture screenshots, and retrieve information related to discovered user-actions and the current application state. D-ForenRIA's *SR-Proxy* is implemented as a Java application.

## 3. D-FORENRIA DEMONSTRATION

***Interface.*** The system's user interface is shown in Figure 3. To interact with the system, a user provides a set of previously recorded HTTP traffic as an input to the *SR-Proxy* (region 1 in Figure 3). The users can add any number of *SR-browsers* to speed up the reconstruction process. Each *SR-Browser* interacts with the *SR-Proxy*, which centralizes the results and stores progress in the specified folder (region 2 in Figure 3). The GUI also provides some statistics about the progress of the reconstruction: number of browsers running, number of tried actions, number of steps recovered, number of early clicks, number of non-existence clicks, and number of resets performed, as well as overall progress and reconstruction duration (region 3 in Figure 3).

As the reconstruction progresses and new user actions are recovered, a thumbnail of each new reconstructed screenshot is added (region 4 in Figure 3). To see the details of a recovered user action, a click on one of the thumbnails opens a new window (region 5 in Figure 3). In this window, the complete image of the screen shot is displayed, as well as the type of the action (e.g., click, form submission, etc.), the XPath expression of the element on which the action occurred and any cookies that were present. The full reconstructed DOM can also be accessed from that screen.

***Demonstration Scenario.*** In order to illustrate the capabilities of D-ForenRIA, we have created a sample attack scenario, using a vulnerable banking application created by IBM for demonstration and test purpose. In our case study, the attacker visits the vulnerable web site, uses an SQL-injection vulnerability to gain access to private information and transfers $5000 to her own account. She also uncovers

[3]`http://www.seleniumhq.org/`

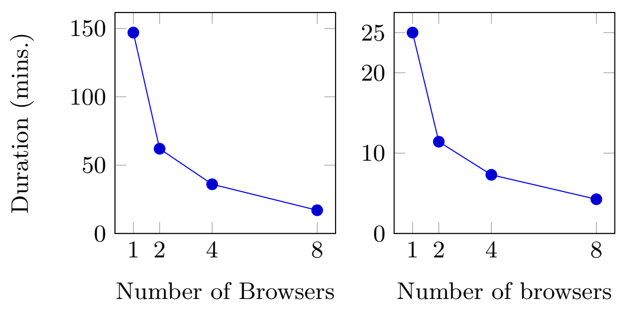


Figure 4: Scalability of D-ForenRIA using the RIA "openCart" for 40 User-Interactions (left) and the RIA "Periodic Table" for 89 User-Interactions (right).

a cross-site scripting vulnerability that she can exploit later against another user.

Given the full traces of this incident, D-ForenRIA reconstructs the attack in about 6 seconds. The output includes screenshots of all pages seen by the hacker, DOM of each page and the steps taken for unauthorized login including the inputs hacker exploited for SQL-injection attack. A forensic analysis of the attack would have been quite straightforward using D-ForenRIA, including the discovery of the cross-site scripting vulnerability. Comparatively, doing the same analysis without our tool would have taken much longer, and the cross-site scripting vulnerability would probably have been missed. A demonstration of the this case study as well as sample inputs/outputs of the tool can be found on `http://ssrg.site.uottawa.ca/sr/demo.html`

***General Results for D-ForenRIA.*** D-ForenRIA has been tested on a variety of RIAs (some results for the original, single-browser version can be found in [3]). The scalability of the tool is of particular interest, since if required a large number of browsers could be used in a cloud environment to reconstruct quickly any user session of any length. We illustrate the speed-up obtained by using several browsers in a simple desktop environment in Figure 4. The test case applications are **OpenCart** (`http://www.opencart.com/`, an open source e-commerce solution) for Figure 4, left, and a simple RIA built for testing purpose, **PeriodicTable** (`http://ssrg.eecs.uottawa.ca/testbeds.html`), for Figure 4, right.

We can see that the speed-up achieved is usually proportional to the number of browsers used to reconstruct the session. It is important to note that maximum speed up is achieved when reconstructing the session is "difficult". When our tool is able to guess immediately what user action was performed next, having several browsers working on the problem is not helping (although it doesn't hurt either). On the other hand, when finding the next action is problematic (which is the time-consuming scenario that is presented in Figure 4), D-ForenRIA fully benefits from having several browsers working in parallel.

## 4. CONCLUSIONS

We demonstrated *D-ForenRIA*, a distributed tool to recover user-browser interactions from a given HTTP traces in RIAs. The main difference between D-ForenRIA and other session-reconstruction tools is that D-ForenRIA only needs captured network traffic and does not require manipulation of the web application. Experiments on different websites show promising improvement of performance and scalability by distributing the workload across several browsers. However, it should be noted that the number of useful browsers is capped by the number of attempts D-ForenRIA makes to find the correct action at each step.

D-ForenRIA is a complementary tool to be used during forensics investigations. For example, an IDS can trigger the recording of HTTP traffic when it discovers suspicious activities, and the recorded traffic will be used as the input for our tool. The output of the tool can be integrated into a SIEM[4] tool for further analysis.

There are several directions for improvement: Dynamically estimating the optimal number of concurrent browsers, better algorithms to detect the next possible actions and handling behavioral differences between user and session reconstruction browsers (for example caching mechanism) to name a few.

**Acknowledgements** This work is supported by Center for Advanced Studies, IBM® Canada Lab and the Natural Sciences and Engineering Research Council of Canada (NSERC).

## 5. REFERENCES

[1] S. Andrica and G. Candea. Warr: A tool for high-fidelity web application record and replay. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 403–410. IEEE, 2011.

[2] R. Atterer and A. Schmidt. Tracking the interaction of users with ajax applications for usability testing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1347–1350. ACM, 2007.

[3] S. Baghbanzadeh, S. Hooshmand, G. Bochmann, G.-V. Jourdan, S. M. Mirtaheri, M. Faheem, and I. V. Onut. Forenria: The reconstruction of user-interactions from http traces for rich internet applications. In *Proceedings of the Twelfth Annual IFIP WG 11.9 International Conference on Digital Forensics*, 2016.

[4] M. I. Cohen. Pyflag - an advanced network forensic framework. *Digit. Investig.*, 5:S112–S120, 2008.

[5] P. Fraternali, G. Rossi, and F. Sánchez-Figueroa. Rich internet applications. *Internet Computing, IEEE*, 14(3):9–12, 2010.

[6] J. J. Garrett. Ajax: A New Approach to Web Applications. Available at: `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/`. Accessed May 28, 2015, 2005.

[7] C. Neasbitt, R. Perdisci, K. Li, and T. Nelms. Clickminer: Towards forensic reconstruction of user-browser interactions from network traces. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1244–1255. ACM, 2014.

[8] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger. Understanding online social network usage from a network perspective. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 35–48. ACM, 2009.

[4]Security information and event management