Lightweight Formal Verification in Real World, A Case Study

Andrea Atzeni, Tao Su, and Teodoro Montanaro

Dip. Automatica e Informatica
Politecnico di Torino, 10129 Torino, Italy
{shocked,tao.su}@polito.it, teodoro.montanaro@studenti.polito.it

Abstract. To security oriented large-scale projects, formal verification is widely used to assure the satisfaction of claimed security properties. Although complete formal verification and validation requires a great amount of time and resources, applying lightweight formal methods to partial specifications reduces the required efforts to a convenient amount, while can still uncover sensitive software design problems. This paper describes our experience of applying lightweight formal verification to the authentication system of *webinos*, a substantial cross-device software infrastructure developed in a large scale EU funded project. The paper details the approach, the properties analysed, the lessons learned and concludes with possible recommendations for practitioners and designers about how to use lightweight formal verification in real world projects.

Keywords: lightweight formal methods, authentication system, web-based platform, security properties, model checking.

1 Introduction

In computer science, formal methods are used under the expectation that the mathematical models will help to find out the errors hidden in system design. However, they are also considered techniques requiring high level expertise. For this reason, they are mostly considered for strong safety-demanding projects, like space applications [1], or for sensitive security building blocks, like cryptographic protocols [2].

In most real world projects, only a few properties are critical, and performing formal analysis on the complete system is expensive. Lightweight formal methods are promising solutions to this problem: only partial specifications and focused properties can be verified, rather than the complete system model.

This paper describes our case study where we applied lightweight formal verification to the authentication system in the *webinos* platform [3], which aims to provide a secured platform for various types of web-enabled devices.

In our study, the experience of generating formal models and the lessons learned during the formal analysis process are more valuable than the end result. They can make recommendations to practitioners and designers in similar situations, in particular the approach of assigning the lightweight formal verification work to the testing group as higher-level testing work.

Testing is cost effective, flexible and widely used, but it is inefficient when dealing with security properties and small probability errors. Thus, we introduced a formal verification process in our work to provide cost-efficient coverage of the key security domains correctness. As a positive side effect, this process also brought a deeper understanding of the previous in-place testing procedures.

2 Related Work

Lightweight formal methods are popularly used to analyse critical security-building blocks as well as strong safety-demanding applications.

Zave [4] applies lightweight formal modelling and analysis techniques to check the correctness of the Chord protocol, a well known DHT algorithm. She analyses the ability of the nodes to maintain a single ordered ring given ample time and no disruptions while it is working. The result shows Chord is not correct: there are cases where the ring may break and never repair itself. In the paper, the author claims the usage of lightweight methods increases the quality of specifications and implementations, taking only a very convenient amount of efforts to detect most problems. Our work reaches the same conclusion. The main differences lie on the approach: Zave's work analyses whether the proposed global invariant is preserved, while our work checks all paths of the available system states, until the proposed properties are satisfied or falsified.

Taghdiri and Jackson [5] present how they use lightweight formal methods on the Pull-Based Asynchronous Rekeying Framework (ARF), a solution proposed for the scalable group key management problem in secure multicast. They agree that lightweight formal methods are feasible and economical. During the analysis, the authors build a model which is less than 100 lines, and check some critical correctness properties. As a result, they detect several hidden flaws, including a serious security breach. Compared with our work, Taghdiri and Jackson use the tick-based modelling idiom, while we use the global-state modelling idiom. They further generalised their model to a structure that can be reused in checking a class of secure multicast key management protocols. The report of using the same structure to validate Iolus protocol can be found in [6].

The researches described above use lightweight formal methods, but do not answer the question of how to smoothly integrate formal methods into real world projects without excessive cost. Researches along this vein exists in the industrial world, where the systems involved are much more complex and only part of the system is formally analysed. This is consistent with the concept of lightweight formal methods, so it is sensible to mention approaches tried out to make formal verification an accepted industrial practice. In this context, the authors of [1] present a case study of using partial formal models for verifying the requirements of FDIR system in space station. They proposed to consider formal verification and validation as intermittent "spot checks" executed by an additional independent formal methods experts. They argue this is a viable way to introduce formal methods into real world projects. With this idea in mind, the authors also present the case studies for spacecraft fault protection systems [7]. They suggest that lightweight formal methods can offer an effective way to improve the quality of specifications, and consequently the end product. While our approach of considering formal verification as part of abstract level testing work brings both advantages and disadvantages, it is suitable for projects with limited resources, i.e. when hiring an additional independent team of experts is not a sustainable solution. Moreover, "re-using" the testing group during the verification process allows for a deeper understanding of the specifications and the system, and this will benefit the future testing work. The drawback comes from the lack of experience of the testing group, which could have a limited knowledge of formal methods techniques.

3 The Case Study

The formal verification work started after a comprehensive testing system was built [8]. The goals were finding the system flaws as well as highlighting the potential misunderstandings in system specifications [9].

3.1 *webinos* Platform

The *webinos* project focuses on constructing a secured platform that can be accessed by multiple types of web-enabled devices.

The architecture of this platform centres on the concept of *Personal Zone*, which is one-to-one correlated with the user. The *Personal Zone Hub* (PZH) is the focal point of the zone. The other devices in the zone are called *Personal Zone Proxies* (PZPs), which support and expose standard JavaScript APIs for accessing devices features, such as camera, geolocation, networking etc. The devices can communicate with each other with or without the PZH after they pass the challenges of the authentication system.

3.2 Authentication System in *webinos*

The authentication system consists of user authentication process, device authentication process and third-party authentication process.

Users are primarily authenticated through their OpenID [10] credentials, such as Google or Yahoo! accounts. This operation only allows a user to connect through the PZH web interface. Devices are authenticated through the possession and use of an RSA private key. For devices which are generally used only by a single user (e.g., smartphone), this credential can be used as an authenticator for many privileged operations. For shared devices (e.g., smart TV), the PZH needs to check the user's presence and identity before this credential can be further used for authentication. Third-party authentication is executed via OAuth 1.0 [11], which requires an additional trusted developer-provided server to hold on the developer's OAuth credentials. This server is accessed by client-side JavaScript of a *webinos* application in order to sign OAuth requests and gain access to third-party resources. In *user authentication* and *device authentication*

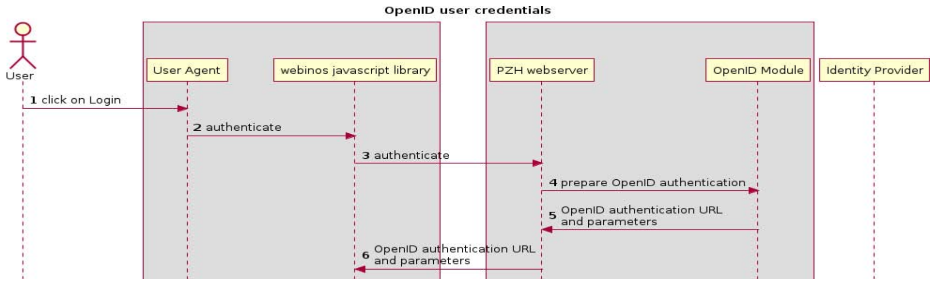
OpenID user credentials
User
User Agent
webinos javascript library
PZH webserver
OpenID Module
Identity Provider
1 click on Login
2 authenticate
3 authenticate
4 prepare OpenID authentication
5 OpenID authentication URL and parameters
6 OpenID authentication URL and parameters

Fig. 1. Partial *webinos* user authentication sequence diagram. Reprinted from [9].

processes, there are three isolated sub-processes corresponding to different situations. For example, in the device authentication process, there is a *device-held private key* sub-process detailing how the user can get the RSA key from the *Keystore Manager* to authenticate the current device.

In *webinos*, the workflow of each process is expressed using sequence diagrams, with some additional explanations written in natural language. For example, part of the user authentications process diagram is illustrated in Fig. 1, and the detailed requirements that "*OpenID login MUST be requested using the PAPE extension and set `max_auth_age=0` in order to prevent authentication caching*" are attached in the end. The other processes are expressed in the same way.

The authentication process is further detailed by an incomplete authentication state machine (i.e. without *presence checking* state), which helps to understand the state transitions, and several entity authentication tables outlining how different entities are authenticated by the others.

3.3 Approach

webinos platform is quite complex. A complete formal verification is neither feasible nor cost-effective with respect to our goals. Our chosen approach is thus practical and straightforward, as detailed in next steps:

1. clarifying the specifications logic and understanding the sequence diagrams;
2. translating the diagrams into corresponding formal models;
3. selecting the need-to-check properties;
4. analysing the results and finding possible improvements.

We used NuSMV [12] model checker. The principals in the authentication system were modelled using *VAR* variables, each operation was presented as a parameter of the *VAR* variable. And an additional *system_status* was inserted to present the snapshot of the system. The transitions between each state of the *VAR* variables were assigned with *ASSIGN* variables based on the switch-case logic. To be consistent with reality, we added non-determinism into the models. User interacts with the system, thus we put two additional states, the *send_wrong_password* state and the *reject_authorisation* state into user's status representing the unexpected user behaviour. Availability is another concern, with the potential occurrence of DoS attack. Thus the *busy* state was inserted into the models, representing the situation where the module stops working after it finishes its previous work. An example of state transitions of the *User Agent* module is shown in Fig. 2. In any transition, with half probability either the module proceeds to the next work or it enters the *busy* state.

```
init(useragent_status):= idle;
next(useragent_status):= case
   system_status = login_request & useragent_status = idle:
        {send_auth_req, busy};
   system_status = load_auth_url & useragent_status = idle:
        {send_identity_provider_url_req, busy};
    TRUE : idle;
esac;
```

Fig. 2. An example, partial state transitions setting of *User Agent*

The need-to-check properties were defined using *SPEC* variables, which fell into the following classifications:

– *completeness*: it is always true that the final state of the system will be the correct one with the corresponding initial state.
– *correctness*: it is always true that the correct final state of the system can be achieved with the corresponding initial state.
– *security properties*: the user who sends too many wrong credentials will be stopped; the user who fails to pass authentication challenges will be blocked for privileged operations; the affect of unexpected situations will be deleted or mitigated by corresponding countermeasures.

3.4 Results

This study formally analysed the authentication system with 18 pages specifications, and built 5 models for the isolated processes. The total effort amount was approximate to 1.5 person months. The efforts mainly came from the translation from sequence diagrams to formal models, and the work of learning how to use NuSMV. During the verification process, the following issues were reported:

– *ambiguities and inconsistencies*: although the specifications are mainly expressed in sequence diagrams, minor ambiguities and inconsistencies still exist. For example, in the "entity authentication tables", the term "*PZPs*" refers to the devices in the third table, but in the sixth table, "*PZPs*" is used to refer to the personal zone proxies.

- *incompleteness*: the specifications lack several key factors, resulting misleading for the implementation. For example, there are no details of how many requests each module can take in the authentication system, and how they will react with multiple requests. This issue leads to an ultimate challenge to build a "perfect" model which shows the exact properties of the authentication system. The method we chose to solve this problem was by adding a second request which is concurrent with the first one, and setting that the modules to work only when they are in *IDLE* state. This solution is a better approximation of real behaviour, but has a very poor scalability.
- *missing assumptions*: the system designers only assume all the modules will work as established, without considering unexpected situations. This error is found in the model checking results, in which the checker reports the *completeness* property cannot be satisfied because of the unexpected situations introduced in the models.

4 Discussion

During our work of applying lightweight formal verification to the *webinos* authentication system, we strictly followed the pragmatic principle: *we only modelled a high-level abstraction of the system, without considering the other parts of the platform.* Still, some important problems were found. Several lessons worthy sharing have been learned, which can give practitioners recommendations on how to introduce lightweight formal methods into real world projects.

4.1 Lesson 1: Choose the Right Tool

The expensive requirement of expertise is the main obstacle between the formal methods and real world projects. A suitable tool can greatly reduce this expense. In our case, NuSMV is a fairly easy-to-use tool with respect to state exploration.

However, one major flaw of NuSMV was found in the verification process. If user sends a wrong password, OpenID scheme will redirect the browser back to the login webpage. NuSMV thinks this operation is a loop, and it raises an error. The same happened when the user rejects the authorisation request.

From another point of view, when a user sends too many wrong credentials, it is likely impersonated by a machine that is trying to perform a brute force attack. OpenID scheme has already adopted the CAPTCHA [13] technique as the countermeasure for this attack. So, in the model, a threshold was added to limit the number of times a user can send the credentials to the OpenID server. Hence, the problem was circumvented with strict consistency to reality.

4.2 Lesson 2: Assign the Verification Work to the Testing Group

Another important question is, who should perform formal verification work to bring greater benefits with less costs? Our answer is: to the testing group. Compared with hiring an independent formal verification experts team, assigning the verification work to the testing group is cost friendly. Moreover, the testing group already has a deep knowledge of the system, so the errors in the formal models can be directly correlated to the system, even to the specific code points. As a positive effect, this would make more robust traceability between the models and the system. Since the testing group is experienced with errors, another important issue of prioritising the errors is also solved by this approach. Finally, the experience in verification improves the testing group capacity to generate test cases, since it brings a deeper understanding of the system

4.3 Lesson 3: Focus Only on Crucial Properties

Focusing on crucial properties is a key argument in lightweight formal methods.

In our case, the authentication system consists of distributed modules and uses different authentication methods for different resources, it is thus rather complex with respect to the need-to-check properties. However, since the connections are established on top of secure channels, we only analyse the system in an abstract level, i.e. the properties relative to message content are not considered. For this reason, only the properties listed in 3.3 are concerned, which are strictly related to the system functionality. These properties are shared by many authentication schemes, thus our models can be generalised and applied to different authentication systems.

4.4 Lesson 4: Adopt OpenID over a Self-developed Module

In the case of building an authentication system for the web-related software, OpenID is more thorough and secure than a "re-invented wheel".

In our case, the designers of the authentication system did not consider user's misbehaviour. As stated in Subsection 4.1, lacking CAPTCHA technique will leave space for brute force attacks. Furthermore, at present the open-source OpenID server implementation is available, allowing for organization-specific configurations, thus the scheme can be used even in case of specific requirements.

5 Conclusion and Future Work

Lightweight formal verification technique minimises the gap between formal methods and real world projects. In our work, several major issues were found with very limited resources. Another important aspect is that this verification work was done by the testing group of the project, allowing for positive feedback in the development of a refined testing suite.

However, to make formal verification an accepted industrial practice, there is still a lot of work ahead. This method should be tested to a broader range of security properties and on other security-critical domains in the platform. Also, the most effective formal model should be identified. We plan to analyse the same system with different formal languages, to assess which language and specifically which language capability is optimal in the concept of lightweight formal methods.

Acknowledgements. The research described in this paper was funded by the EU FP7 *webinos* project (FP7-ICT-2009-05 Objective 1.2).

References

1. Easterbrook, S., Callahan, J.: Formal methods for verification and validation of partial specifications: A case study. Journal of Systems and Software 40(3), 199–210 (1998)
2. Mundra, P., Shukla, S., Sharma, M., Pai, R.M., Singh, S.: Modeling and Verification of Kerberos Protocol Using Symbolic Model Verifier. In: 2011 International Conference on Communication Systems and Network Technologies, pp. 651–654. IEEE (June 2011)
3. Fuhrhop, C., Lyle, J., Faily, S.: The webinos project. In: Proceedings of the 21st International Conference Companion on World Wide Web - WWW 2012, p. 259. ACM Press, New York (2012)
4. Zave, P.: Using lightweight modeling to understand chord. ACM SIGCOMM Computer Communication Review 42(2), 49 (2012)
5. Taghdiri, M., Jackson, D.: A lightweight formal analysis of a multicast key management scheme. In: König, H., Heiner, M., Wolisz, A. (eds.) FORTE 2003. LNCS, vol. 2767, pp. 240–256. Springer, Heidelberg (2003)
6. Taghdiri, M.: Lightweight modelling and automatic analysis of multicast key management schemes. Master's thesis. MIT (2002)
7. Easterbrook, S., Lutz, R., Covington, R., Kelly, J., Ampo, Y., Hamilton, D.: Experiences using lightweight formal methods for requirements modeling. IEEE Transactions on Software Engineering 24(1), 4–14 (1998)
8. Su, T., Lyle, J., Atzeni, A., Faily, S., Virji, H., Ntanos, C., Botsikas, C.: Continuous integration for web-based software infrastructures: Lessons learned on the webinos project. In: Bertacco, V., Legay, A. (eds.) HVC 2013. LNCS, vol. 8244, pp. 145–150. Springer, Heidelberg (2013)
9. webinos group: webinos authentication system specifications (2012), `http://www.webinos.org/content/html/D033/Authentication.htm`
10. Recordon, D., Reed, D.: OpenID 2.0. In: Proceedings of the second ACM workshop on Digital identity management - DIM 2006, p. 11. ACM Press, New York (2006)
11. Hammer-Lahav, E.: The OAuth 1.0 Protocol (2010), `http://tools.ietf.org/html/rfc5849`
12. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: A reimplementation of SMV. In: Proc. STTT 1998, pp. 25–31 (1998)
13. von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: CAPTCHA: Using hard AI problems for security. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 294–311. Springer, Heidelberg (2003)