A Socio-technical Understanding of TLS Certificate Validation

Giampaolo Bella[1], Rosario Giustolisi[2], and Gabriele Lenzini[2]

[1] Dipartimento di Matematica e Informatica
Università di Catania, Italy
giamp@dmi.unict.it
[2] Interdisciplinary Centre for Security,
Reliability and Trust University of Luxembourg
{rosario.giustolisi,gabriele.lenzini}@uni.lu

Abstract. To authenticate a web server, modern browsers check whether a TLS certificate is valid. This check is *socio-technical* because, when the technical validation fails, it may request the user to decide, intertwining the usual technical issues with social elements, such as trust and cultural values. Hence the need for a methodology aimed at a socio-technical understanding of TLS certificate validation. This aim is demanding not only due to user participation but also because browsers behave differently. An innovative methodology is outlined and demonstrated on the four market-leader browsers, Chrome, Internet Explorer, Firefox and Opera Mini. It involves modelling in UML the multi-layered interactions among servers, browsers, and users and then translating them into a formal language amenable to model checking socio-technical security properties.

1 Introduction

The cryptographic protocol to credit for HTTPS security is TLS. It enforces confidentiality and integrity by combining symmetric and asymmetric cryptography as fully documented in the specification of TLS [4].

The implementation of authentication, the other chief security property of TLS, is less standardised. Based on asymmetric cryptography, authentication cannot work without TLS certificate validation, a mechanism (recalled below) to establish the sole owner of an asymmetric key pair. Notably, the TLS specification provided also for the case when certificate validation fails: "*If the hostname does not match the identity in the certificate, user oriented clients MUST either notify the user (clients MAY give the user the opportunity to continue with the connection in any case) or terminate the connection with a bad certificate error.*" [7]. The excerpt describes the scenario where a browser (a client) finds a mismatch between the server name (hostname) that TLS transports and the identity in the certificate, and advocates an interaction with the user without detailing it, or termination. Today, this scenario occurs frequently, for example when an intruder replaces the server's certificate with his own, attempting to masquerade as the server. Another frequent case of failed certificate validation

C. Fernandez-Gago et al. (Eds.): IFIPTM 2013, IFIP AICT 401, pp. 281–288, 2013.

arises with institutions that self-sign their own certificates rather than purchase them from accredited authorities. In consequence, as we shall see below, authentication depends on technical factors such as networking, cryptography and certification, but also on social factors such as user skills, education and trust.

Certificate validation has been studied variously, for example to avoid the user's oversight of warning messages [9], or to improve the readability of their contents [3]. The most relevant existing work seems about the cognitive aspects of human-computer interaction with the browsers. To position our work, it is useful to note that we see the socio-technical system consisting of a web server, a computer network, a browser, a user and possibly an intruder as a *ceremony* in the sense of Ellison [5]. The various technical and social layers of a ceremony have been recently identified [2], with a practically useful indication that certain layers can be virtually compressed during the analysis to sharpen the analyser's focus on other layers. Our work complements existing human-computer interaction studies by advancing what seems to be the first formal analysis of properties of certificate validation that are not only logically conditioned on the technology but also on user actions — while the details of human behaviour and human cognitive aspects are abstracted away.

We outline TLS certificate validation (Sect. 2), and assess it from a socio-technical standpoint. We then study four main properties over what are currently found to be the four market-leader browsers at present [1]: Chrome, Internet Explorer, Firefox and Opera Mini. Therefore, sixteen socio-technical properties of modern browsers are studied, and are all related to certificate validation. In particular, when Firefox cannot validate the certificate of an honest server, it allows the user to store it; it is found that, even if the user stores it, if Firefox cannot validate a different certificate for the same server in a second session, it allows the user to store it again, remarkably, without checking that one was already stored. This missing check allows the certificate of the second session to come from an intruder, who could then masquerade as the server (Sect. 4).

This paper contributes well beyond the formal analysis of the sixteen properties, as it outlines an innovative methodology for the socio-technical analysis of browser ceremonies. After trying out various graphical notations, UML activity diagrams were found to bear the necessary flexibility (Sect. 3). Building these diagrams was a major hallmark in our understanding of the technicalities of the browsers. However they are only semi-formal and not directly executable, while our aim was to leverage on automatic tools to assess properties reliably. We then derive a CSP# model along with an linear temporal logic specification of the properties of interest to input to the Process Analysis Toolkit (PAT) [8]. A summary of the findings and some conclusions terminate this paper (Sect. 5).

2 Basics of TLS Certificate Validation

A TLS certificate is an X509 certificate, and consists of a numbers of fields. Many of them are irrelevant to the focus of this manuscript, and shall be ignored. An idealised TLS certificate associates an identity with a specific public key,

and $\langle ID, PK_{ID}, I, SIG \rangle$ is its custom representation. Here, ID stands for the certificate subject, the entity to whom the certificate is issued (such as a subject name and URL). Then, PK_{ID} is ID's public key, and I is the entity who issues the certificate. The issuer I checks ID's identity off-line and vouches that PK_{ID} is ID's public key by affixing I's digital signature SIG to the certificate.

There are two main checks for certificate validation. One is *trusted issuer verification*, to establish whether the certificate issuer is a trusted Certification Authority (CA). This is done by querying a sequence of CAs (up to a root) stored by the operating system or by the browser itself. Notably, knowing which CAs are trusted may vary from browser to browser, so this check is browser-dependent. Legitimate and honest servers do not always purchase certificates signed by a trusted CA, often preferring to sign their certificates themselves. Self-issued certificates are risky because they cannot be verified by browsers, raising the risk of man-in-the-middle attacks. This risk is our focus.

The other check is *domain name verification*, to establish whether the URL requested by the user matches the ID in the server certificate. Even when the server certificate can be verified, it may happen that the requested URL does not match the certificate ID. This commonly occurs when an institution needs to secure its own sub-domain (e.g., `www.my.example.com`), but it does not want to purchase a large number of certificates. Also a failure of this check may be risky. A domain verification failure may conceal a man-in-the-middle attack. An intruder could simply create a certificate for a domain that he owns, then get it signed by a trusted issuer, and bundle it with the traffic for the user.

When the validation process described in the X.509 standard fails, browsers may adopt certain mechanisms to decide whether the requested server can be authenticated. There are three main such mechanisms. The first two are inherently socio-technical due to the human intervention, the third is purely technical.

Warnings. Browsers may warn the user when they receive a certificate containing an untrusted issuer or the domain verification fails. They warn users in various ways, such as by pop-up windows, open padlocks or red address bars, but because the present work is not concerned with interface layouts, no difference will be made, but what counts here is that browsers may allow the user to abort or continue with their requested server. How users decide is equally beyond focus.

Storing Server Certificates. When validation of a certificate fails, it may still be the case that the user opts to trust the certificate, namely to accept its contents as trustworthy. Browsers may support this scenario by allowing a user to store the server certificates the user decides to trust. In consequence, when the server is accessed next, its certificate validation will be immediate because its certificate is found in a trusted store. It is clear that server authentication as established in this scenario is a socio-technical property because it is mostly based on the user's trust in the certificate rather than on objective support provided by the technology. Of course, storing server certificates is not always a secure choice.

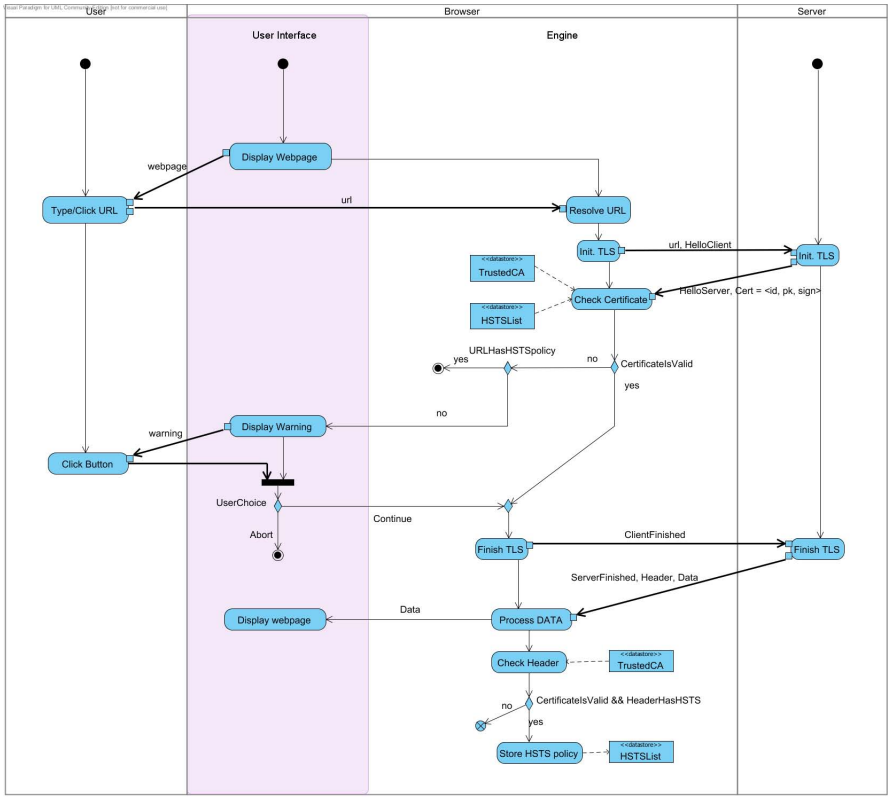

Fig. 1. Activity diagram for certificate validation in Chrome

HTTP Strict Transport Security (HSTS). It is a security mechanism that was conceived to thwart SSL stripping attacks, whereby an intruder fools a user into an HTTP connection to a server although the server is also HTTPS compliant. In short, HSTS compliant browsers prevent "unsecured" HTTP connections to HSTS compliant servers. The server transmits its HSTS compliancy to a calling browser via a special header during a secured TLS session.

3 Modelling Browsers

Our first contribution is to model how four relevant browsers implement the full TLS certificate validation. In tackling browsers, we observe that there appears to be no standard notation to describe them. Our choice is to use UML activity diagrams to describe how browsers function. UML activity diagrams support an intuitive representation of a TLS session, highlighting the validation mechanisms of each browser. Also, they can be easily translated in a fully formal language, with the help of their semi-formal semantics — in particular, we will translate activity diagrams to CSP#, which is then fed to an automatic tool. A third

advantage is that they can represent parallel actions (fork/join) and multiple choices (branching), while other notations such as Message Sequence Charts or the Alice-and-Bob notation, cannot.

We select the four browsers that are presently market leaders: Chrome, Internet Explorer, Firefox and Opera Mini. They expose a variety of combinations of the mechanisms seen in the previous Section. For example, Chrome declares HSTS support, Internet Explorer uses warnings extensively, Firefox may seem most complete, and Opera Mini clearly aims at being lightweight. We have built an activity diagram per browser. Every diagram features three entities: User, Browser, and Server. Browser is in turn divided into User Interface and Engine. There is no room to present all diagrams, but Figure 1 shows the one for Chrome. Entities have a begin circle that points to their own first activity. Thick arrows depict the flow of activities among different entities, while thin arrows stand for the internal entity flow. Arrow labels define the objects that are exchanged between activities. Some activities need to access datastores, which are linked to activities via dashed arrows. Most activities are self-explanatory and common to all browser diagrams, such as *Display Webpage* and *Type/Click URL*. To keep the focus on the browser, the server activities are reduced to *Init. TLS*, whereby the server starts the TLS handshake on its side, and *Finish TLS*, where it concludes the handshake. For example, Figure 1 shows that Chrome supports HSTS and adopts different certificate stores depending on the underlying operating system.

4 Socio-technical Security Analysis

We refer to socio-technical analysis since we focus on the technical aspects of TLS certificate validation as well as the role of the user. Our UML models show that browsers validate TLS certificates differently. As analysers, we question whether such differences entail different security, and addressing this question is the focus of this and the next section.

4.1 A Prototype Methodology

Activity diagrams are only semi-formal, hence are not practical for formal verification. Aiming at automatic analysis, it is appropriate to use them to then write a formal model on which a formal encoding of the properties of interest can be verified quickly. Also, as it can be expected, the formal browser model shall be augmented with models for the other relevant actors, that is an intruder and a user. We have taken advantage of the user-friendliness of CSP#, a formal modelling language based on the process algebra Communicating Sequential Processes (CSP) [6], and of the flexibility of Linear Temporal Logic (LTL) to formulate the relevant properties. These form the input to the Process Analysis Toolkit (PAT), which over our experiments has computed outputs within seconds. The full specification is omitted here. Our prototype methodology for the socio-technical security analysis of how browsers treat certificate validation is as follows: (i) describe the browser under study as UML activity diagrams;

(ii) use these diagrams to write a formal model; (iii) extend it with additional models, respectively for an honest server, a user and an intruder; (iv) encode the properties of interest in a formal language; (v) execute the extended formal model and the encoded properties in PAT to assess the validity of the properties.

Additional models, for an honest server, a user and an intruder, are omitted for brevity. In particular, the server can opt for self-issued certificates; the user is modelled expressively as a non-deterministic process; the intruder, in line with a Dolev-Yao one, owns another server, controls the network, and can fake certificates at will but not tamper with encryption.

4.2 Socio-technical Security Properties

We describe four different security properties. Each property binds elements like TLS session, certificates and, notably, user choices.

Property 1 (Warning Users). *A user whose browser receives an invalid certificate on a TLS session is warned about this by the browser before the browser completes the session.*

As explained above, a certificate is invalid if either trusted issuer verification or domain name verification fail. We remarked that this can be the case in a variety of scenarios, more ore less risky for the user. For example, some scenarios conceal an intruder who attempts a man-in-the-middle attack inserting a fake certificate of his own; others see a server self-issue a certificate for itself. By this property, we set out to assess whether browsers warn the user that certificate validation was not smooth.

Property 2 (Storing Server Certificates). *A user who stores a certificate that associates an honest server to its public key on a TLS session via a browser is protected from man-in-the-middle attacks on future sessions with the same server via the same browser.*

When browsers receive an invalid certificate, they may still allow the user to store it. If one assumes that a certificate in fact is trustworthy because it contains a correct association between an honest server and the public key that legitimately belongs to it, one could expect that future sessions with the same server will be protected from man-in-the-middle attacks. We shall see in the discussion that follows that this is not true for all browsers.

Property 3 (Applying HSTS User Security). *A user who accesses a server that sends an HSTS header on a TLS session via a browser that receives a valid certificate about the server is protected from man-in-the-middle attacks on future sessions with the same server via the same browser.*

This property stands on a different scenario from that of the previous one, although their conclusions are equal. This scenario sees an HSTS compliant server who sends a valid certificate to the browser that the user is using. However, it remains to be checked whether the browser is HSTS compliant too, in which case it whitelists the server because its certificate is validated once.

Property 4 (Learning from Server Certificate History). *A user who completes a TLS session with a server via a browser receiving an invalid certificate, and then completes another session with the same server via the same browser receiving a valid certificate is warned by the browser about the risk of man-in-the-middle attack.*

This property aims at checking whether the browser informs the user that a man-in-the-middle attack might have occurred *in a previous TLS session*. For example, let us consider a session where the browser receives an invalid certificate from an intruder who is shading the required server; the browser may warn the user about this (according to Property 1). If in a subsequent session the browser receives a valid certificate about the same server, it may be taken as an additional indication that something went wrong in the former session — hence the check on whether the browser reinforces the warning to the user that she risked a man-in-the-middle attack in the former session.

5 Summary of the Findings and Conclusions

We have translated all previous properties as LTL formulas and checked their satisfiability with PAT model checker on CSP# models representing our browsers extended with server, intruder and user model. We have run the tool on each of our four browsers, and on each of our four target properties, reaching sixteen interesting findings. The properties are encoded in such a way that the tool never incurs into state explosion. Interpreting the tool output required some effort, and the following table summarises the findings.

Browser	Property 1	Property 2	Property 3	Property 4
Firefox	×	×	✓	×
Chrome	✓	✓	✓	×
Internet Explorer	✓	✓	×	×
Opera Mini	×	✓	×	×

Property 1 is found valid over Chrome and Internet Explorer. By contrast, the model checker shows traces that falsify it over Firefox and Opera Mini. With Firefox, the trace reports a sequence of two TLS sessions both with a man-in-the-middle attack. In the first session, Firefox warns the user, who chooses to store the intruder certificate anyway. In the second session, the user tries to access the same server, but this time Firefox has the intruder server certificate stored, and completes the session without warning the user. This is due to the drawbacks of storing server certificates, which Firefox allows its users to do. The trace that falsifies the property with Opera Mini is rather trivial because this browser does not offer any choice to the user at all. Opera Mini in fact shows a padlock when the certificate is valid, but even if the certificate is invalid, the browser completes the TLS session anyway, and without informing the user.

Property 2 turns out the most tricky. The outcome of our experiments may seem surprising. It is found that all browsers verify the property except Firefox,

although Firefox is the only one that allows a user to store server certificates. The outcome is easy to explain in logical terms. The property is a logical implication whose precondition is that a user stores a server certificate. Because no browser except Firefox allows this, the property is trivially verified of all these three browsers, which falsify that precondition. Most surprisingly, the property does not hold of Firefox, which does not falsify the precondition. The user can in fact replace a server certificate as many times as she wishes to, while Firefox does not inform her that a server certificate was already stored.

Property 3 is found to be valid with the browsers that support HSTS, whereas the tool outputs counterexamples with the others. Finally, checking Property 4 fails on all browsers, denouncing the stateless philosophy whereby browsers do not record warnings they may have given in the past, hence they cannot leverage upon them at present. This seems suggesting that, if browsers stored and processed past warnings, they could help users estimate the trustworthiness of current, yet invalid, certificates.

The socio-technical analysis of the security of modern browsers combines traditional analysis of the underlying technologies with elements of user participation. By doing so, it is oriented at characterising security properties also in terms of what the user may accomplish, with the ultimate aim of building browsers that are secure *in* the presence of humans. Our work pertains to various properties related to TLS certificate validation as carried out by four market-leader browsers. It also outlines a general methodology that is being extended towards the socio-technical analysis of browser ceremonies.

References

1. `http://gs.statcounter.com/#browser-ww-monthly-201211-201301`
2. Bella, G., Coles-Kemp, L.: Layered Analysis of Security Ceremonies. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IFIP AICT, vol. 376, pp. 273–286. Springer, Heidelberg (2012)
3. Biddle, R., van Oorschot, P.C., Patrick, A.S., Sobey, J., Whalen, T.: Browser interfaces and extended validation SSL certificates: an empirical study. In: Proc. of the ACM CCSW 2009, pp. 19–30. ACM, New York (2009)
4. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (2008)
5. Ellison, C.: Ceremony design and analysis. IACR eprint archive, pp. 1–17 (2007)
6. Hoare, C.A.R.: Communicating Sequential Processes. Commun. ACM 21(8), 666–677 (1978)
7. Rescorla, E.: HTTP Over TLS. RFC 2818 (2000)
8. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: Towards Flexible Verification under Fairness. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 709–714. Springer, Heidelberg (2009)
9. Sunshine, J., Egelman, S., Almuhimedi, H., Atri, N., Cranor, L.F.: Crying wolf: An empirical study of SSL warning effectiveness. In: Proc. of USENIX 2009 (2009)