Socio-Technical Formal Analysis of TLS Certificate Validation in Modern Browsers

Giampaolo Bella*, Rosario Giustolisi† and Gabriele Lenzini†
*Dipartimento di Matematica e Informatica
Università di Catania, Italy
†Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg

***Abstract*—Authenticating a web server is crucial to the security of web browsing. It relies on TLS certificate validation, a property whose enforcement may require getting the user involved. Thus, certificate validation is a *socio-technical* property — it relies on traditional security technology as well as on social elements such as cultural values, trust and human-computer interaction. Hence the need for an appropriate methodology to study certificate validation from a socio-technical perspective. Certificate validation as carried out through today's most popular browsers — Chrome, Internet Explorer, Firefox and Opera Mini — is first represented by means of UML activity diagrams. It is then translated into CSP#, and expanded with the LTL formalization of four socio-technical properties pivoted on user involvement with certificate validation. The properties are then checked automatically using the PAT model checker. The findings turn out to be far from straightforward and, most importantly, allowed for prototyping a basic methodology for the socio-technical formal analysis of security properties.**

I. Introduction

The cryptographic protocol in support of HTTPS security is TLS. It enforces confidentiality and integrity by combining symmetric and asymmetric cryptography, as it is fully documented in the specification of TLS [**?**]. TLS also allows authentication between clients and servers, although servers are commonly requested for authenticating. While we can reasonably assume that TLS provides confidentiality and integrity as it uses robust cryptographic schemes, we should be more careful in assuming the same for authentication. Its implementation is less standardised, and its practical functioning depends, to various degrees, on trust. One element of trust comes from the web-of-trust concept, thus from the public-key infrastructure. Those solutions should reassure browsers of the identity of the owner of the public key contained in a TLS certificate, which plays a fundamental role during TLS authentication. A trustworthy certificate is expected to be signed by a trusted authority, whose signature a browser should be able to validate, that is, to recognize and accept as reliable.

But when this validation fails, or when the certificate is somehow ill-formed, what is ultimately responsible for how authentication ends is user trust. This is implicit in the TLS specification, which says that: *"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."* [**?**]. So, once notified, the user can choose. And this scenario occurs frequently, for example when an intruder replaces the server's certificate with his own, attempting to masquerade as the server, or when institutions self-issue their own certificates rather than purchase them regularly from accredited authorities. Therefore, a premise of the present research is the realisation that the authentication of a server with a browser requires special attention. This attention does not intend to contribute to the long-established debate on the interpretation of the technical meaning of authentication [**?**] but, rather, intends to substantiate our observation that server authentication with modern browsers is a *socio-technical* property as this paper seeks to explain. More precisely, authentication depends on technical factors such as networking, cryptography and certification, but also on social factors such as user skills, education and trust.

Because the TLS specification leaves browsers some freedom on how to involve users, the way this socio-technical authentication is accomplished varies among browsers. This variety motivates a number of research questions. What are the differences in terms of user involvement in how modern browsers implement authentication? Which browsers reduce the risks of server authentication failure? Could browsers involve the user more profitably than they do at present in support of server authentication? This list of questions, purposely truncated to length three here, arises when server authentication, and TLS certificate validation in particular, is assessed from a socio-technical standpoint.

Contribution: This paper continues our informal presentation of the problem statement in a short paper [**?**] with the full formal treatment in UML and CSP#. Browser security has been studied variously (Sect. II), for example to avoid the user's oversight of warning messages [**?**], or to improve the readability of their contents [**?**]. The most relevant existing work seems positioned over 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 [**?**]. The various technical and social layers of a ceremony have been recently identified [**?**], with a practically useful suggestion that certain layers can be neglected, namely virtually compressed, during the analysis to sharpen the analyser's focus on other layers.

Along these lines, our work complements traditional 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 the human-

computer interaction layer are abstracted away. More precisely, we outline TLS certificate validation (Sect. III), and assess it from a socio-technical standpoint.

To address the questions that arise, such as those made above, we study four main properties. Each property is studied over four of the leading browsers at present [**?**]: Chrome, Internet Explorer, Firefox and Opera Mini. This analysis therefore evaluates sixteen corresponding socio-technical properties, all related to TLS certificate validation, of modern browsers. In particular, it is found that when Firefox cannot validate the certificate of an honest server, it allows the user to store it; 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.

The contribution of this paper exceeds the formal analysis of the sixteen properties. This formal analysis was not carried out using a known methodology. By contrast, it was not obvious how to represent (portions of) the functioning of a browser in order for the analyser to quickly get to grasps with its properties without reading long prose. Various graphical notations were tried out, and finally we found UML activity diagrams [**?**] to bear the necessary flexibility (Sect. IV). 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 a formal model was needed for our aim of fully automatic analysis. We therefore decided to use the UML diagrams to derive a CSP# model, then extended with a Linear Temporal Logic (LTL) specification of the properties of interest. This process formed the inputs to the Process Analysis Toolkit (PAT) model checker [**?**], which yielded the findings reported below. After a summary of the findings, some conclusions can be derived (Sect. VI).

II. Related Work

Browser security has been tackled in a number of ways and from a variety of standpoints. Our work lies in the area of browser security as a whole, but does not seem strictly related to anything that exists so far. It therefore seems fair to consider it the first socio-technical formal analysis of server authentication with a user via a browser and TLS certificates.

Other work has focused on certificate validation issues before. Georgiev et al. [**?**] analyse the SSL certificate validation carried out by a number of applications, such as integrated shopping carts or those for transmitting details of customer payments from merchants to payment gateways. By manual inspection of source code, they find a number of software vulnerabilities due to bad API design. However, these applications never involve browsers. Flinn et al. [**?**] point out that different user perceptions of the term "secure Web site" lead to different levels of trust in a site. By contrast, our formal analysis does not pertain to user perception and investigates the various ways in which a user may influence authentication. Akhawe et al. [**?**] introduce a formal model of web security with three distinct attacker models. They analyse five security mechanisms, which do not cover TLS or certificate validation. Groß et al. [**?**] propose a model of an ideal browser with the aim to analyse browser-based protocols. Instead of designing a new browser, we tackle and assess the most popular existing ones. It would be interesting to apply the prototype methodology that we develop below to their ideal browser. Jøsang et al. [**?**] point out that TLS does not provide semantic server authentication, and can be easily exploited by semantic attacks. However, web browsers can only do syntactic server authentication, while complex user models are needed to deal with semantic attacks. Our work focuses on browsers, and aims at analysing how they help users to avoid authentication attacks. Gajek et al. [**?**] formalise a specific user behaviour capable of recognising the so called human-perceptible indicators such as pictures and sounds. In contrast, we are not interested in the cognitive aspects, and shall make the minimum possible assumptions about the user by defining a non-deterministic one. Kaminsky et al. [**?**] point out that a subject name in an X.509 certificate can be easily misinterpreted by browsers because of lack of standardisation, which often invites ambiguity. To confirm this, a recent update of the X.509 standard [**?**] provides some clarifications, which, however, do not impact our work.

III. Basics of TLS Certificate Validation

This section outlines the main concepts in support of the subsequent treatment, that is, the form of TLS certificates, how browsers validate them, and the mechanisms put in place when certificate validation fails.

A. TLS Certificates

A TLS certificate is an X509 certificate [**?**] and it consists of a numbers of fields. Many of them are irrelevant to the focus of this manuscript and shall be ignored. Keeping the fields that are relevant for authentication purposes, a 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 was issued (e.g., its URL). Then, PK_{ID} is ID's public key, and I is the entity who has issued the certificate. The issuer I checked 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. The digital signature is built by means of I's private key, K_I. Signing also preserves the integrity of a certificate during its transmission, preventing anyone who forges a certificate to be able to sign it on I's behalf. Therefore, an integral certificate takes the form $\langle ID, PK_{ID}, I, Enc(\mathsf{Hash}(ID, PK_{ID}, I))_{K_I}\rangle$.

B. Certificate Validation

The main validation checks for authentication are *trusted issuer verification* and *domain verification*.

Trusted issuer verification: It aims at verifying that 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. Note that knowing which CAs are trusted may vary from browser to browser, so this check is browser-dependent.

Failure: Legitimate and honest servers do not always purchase expensive certificates that are signed by a trusted issuer. Although that is not a guarantee of security, the use of trusted certificates minimises the user involvement during

the validation process. However, self-issuing a certificate can be done quickly and at no expense — the server signs a certificate for each of its sub-domains by using an arbitrarily-generated private key, and then transmits the certificate to calling browsers. Such self-issued certificates are widespread through a large number of public institutions, such as the authors' Universities, or the US Army [**?**]. However, using self-issued certificates is risky, at least because they could be created by anyone, and could be replaced by man-in-the-middle intruders acting in intermediate routers. These risks are relevant for the goal of this paper.

Domain name verification: This aims at verifying whether the URL requested by the user matches the *ID* in the server certificate. Although the server certificate is trusted, it may happen that the requested URL does not match the *ID* in the certificate.

Failure: 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, one per sub-domain, to reduce expenditure. According to a recent large-scale survey on web certificates [**?**], domain mismatch is the most common case of certificate validation failure. Note that 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.

C. Managing Failed Certificate Validations

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. Notably, with the aim of circumventing the limitations of certification seen above, Chrome implements *public key pinning* [**?**] coming with a pre-installed whitelist of servers and their public keys. However, the whitelist is static, hence it can never be changed while the browser collects history. This makes it trivial from the analysis standpoint, and so it shall be ignored below.

1) Warnings: Many browsers warn the user when they receive a certificate containing an untrusted issuer or the domain verification fails. They may allow the user to abort or continue with their requested server despite the contents of the warnings. Browsers warn their users in various ways, such as by pop-up windows, open padlocks or red address bars; because the present work is not concerned with interface layouts, no difference will be made. Warnings are relevant to the present work because they fundamentally influence server authentication when it is addressed as a socio-technical property. As we shall see below, such influence comes from the number and type of warnings, from the point in time at which they come during the authentication process, as well as from the options they give.

2) 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. An obvious drawback is that this store is normally static and, as such, opaque to dynamic information that may come from the Internet, such as revocation lists. 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.

3) 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. HSTS avoids user intervention in favour of a purely technical enforcement of security. Let us consider an HSTS compliant browser-server pair. When a user requests access to the server via the browser, if the server sends a valid certificate, then the browser whitelists the server. This means that authentication of that server succeeded once and shall not fail in the future.

IV. Modelling Browsers

Our first contribution is to model how four relevant browsers implement the full TLS certificate validation. The standard notation for the semi-formal description of security protocols is the so called Alice-and-Bob notation [**?**]. In tackling browsers, we observe that there appears to be no standard notation to describe them. Our choice is to describe how browsers function by means of UML activity diagrams. This choice will be demonstrated below over the main target of this paper, which is how browsers verify TLS certificates.

The contribution of activity diagrams is threefold. First, they give an intuitive representation of a TLS session, highlighting the validation mechanism of each browser. Second, they can represent parallel actions (fork/join) and multiple choices (branching), while other notations such as Flow Charts or the Alice-and-Bob notation, cannot. Third, they can be easily translated in a fully formal language, thanks to their semi-formal semantics. In particular, we translate activity diagrams to CSP#, which is then fed to an automatic tool. The complete CSP# treatment can be found in [**?**].

A. Description of Browsers in UML Activity Diagram

We select the four browsers that are 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 present an activity diagram per browser (see Fig. 1 - Fig. 4). Each diagram does not intend to describe the full browser functionalities but is limited to how the browser treats certificate validation. Every diagram has four columns each representing communicating elements. Three are entities: User, Browser, and Server. Browser distinguishes

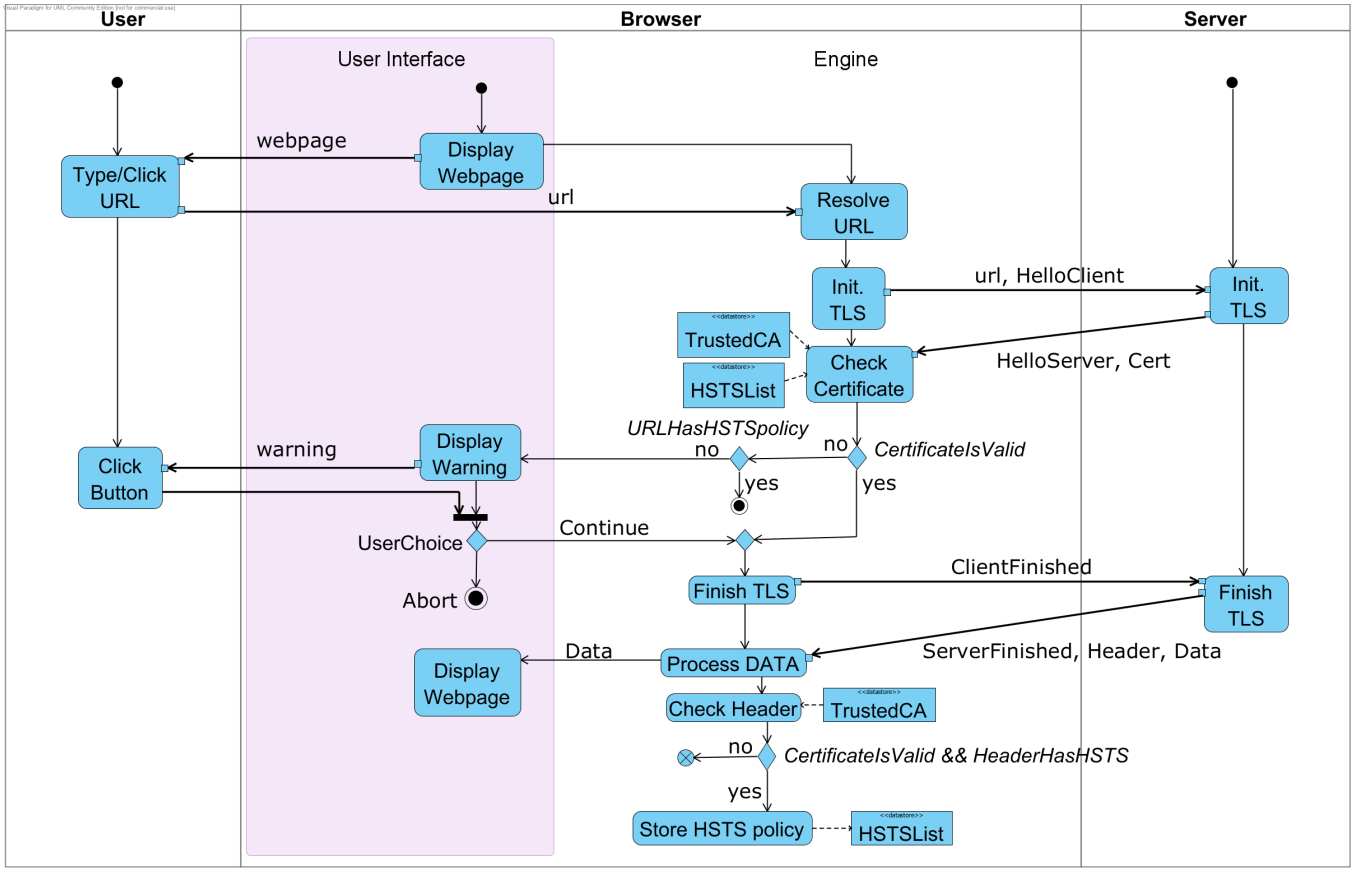

Fig. 1: Activity diagram for certificate validation in Chrome

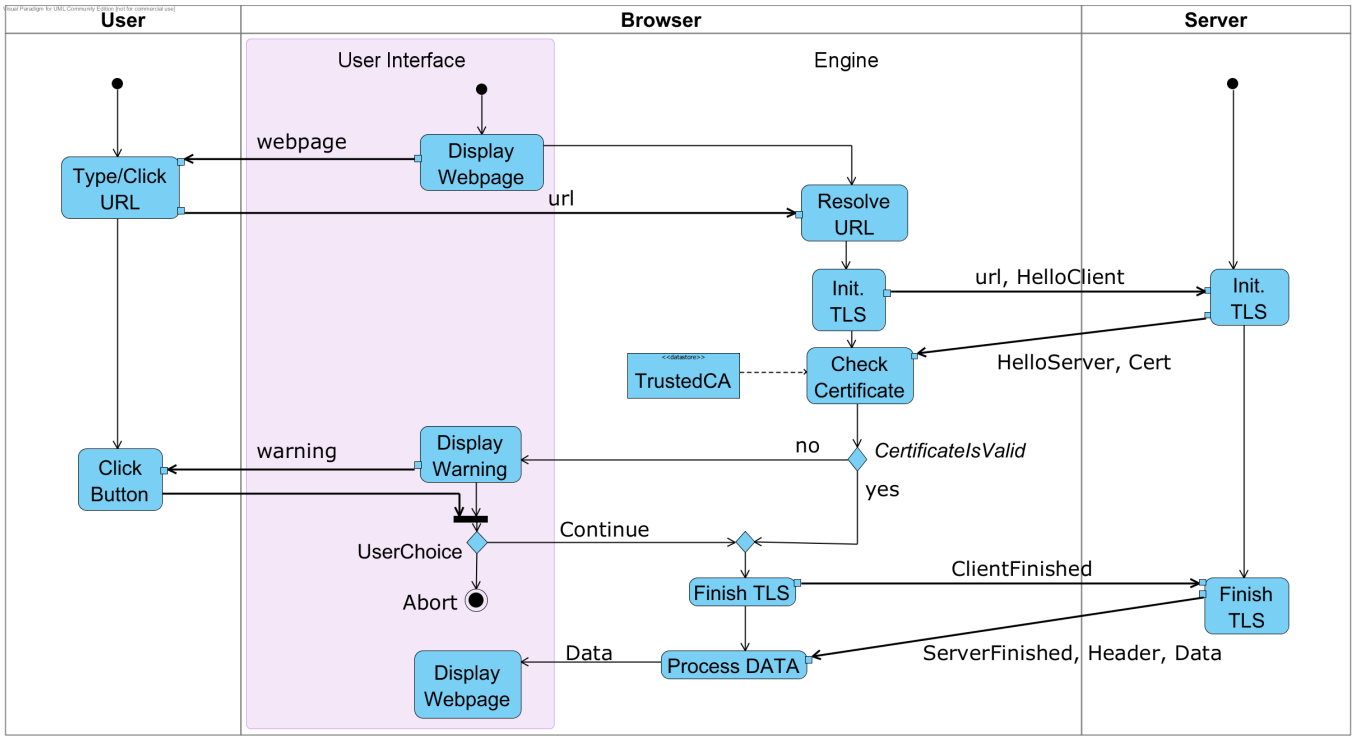

Fig. 2: Activity diagram for certificate validation in Internet Explorer

two standard sub-entities: User Interface and Engine. 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.

Figures from Fig. 1 to Fig. 4 show respectively the activity diagrams for Chrome, Internet Explorer, Firefox and Opera Mini. We build the diagrams of Chrome and Firefox by looking at their official documentation and code. The others are built empirically, supported also by network analysers, because Internet Explorer and Opera Mini are closed source.

There is no room to describe the diagrams fully here, but they can be easily read with some UML background. In particular, it can be seen that Chrome supports HSTS and adopts different certificate stores depending on the operating system underlying the browser; Internet Explorer resorts to the Microsoft Windows store to verify the server certificate; Firefox has its own certificate store, supports HSTS policies, and allows users to store server certificates; Opera Mini is unsurprisingly the simplest one, with its Engine activities being executed on a remote Opera server.

V. 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. Activity diagrams 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 the present work.

A. A Prototype Methodology

Although activity diagrams are semi-formal and permit a quick glance at the niceties of browsers, they are not practical for formal verification. Aiming at automatic analysis, it is appropriate to use them to 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) [**?**], and of the flexibility of LTL to formulate the relevant properties. These form the input to the PAT model checker, which over our experiments has computed outputs within seconds (Sect. V-D). 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, an intruder and a user; (iv) encode the properties of interest in a formal language; (v) execute the extended formal model and the encoded properties in an automatic tool to assess the validity of the properties.

It is worth remarking that our current choices of formal languages and supporting tools are not meant to be binding; rather, they aim at demonstrating the prototype methodology. Also, we are currently working on reproducing the experiments described below on different tools. Not only does this aim at confirming the findings by an alternative verification means, but in particular at finding the most convincing route to translate UML activity diagrams automatically to a formal, executable language. This would make the methodology described above more mechanical, and there already exists work that can be leveraged upon [**?**].

B. Additional models

When writing the formal encoding we were obliged to make choices, such as defining the attacker model (thus distinguishing between the honest and the dishonest server (the intruder), and the user's behaviour. We describe our choices informally, their encoding in CSP# can be found in [**?**].

Honest server model: It is a server that never cheats about its identity. However, it may choose to self-issue its certificate.

Intruder model: It is an intruder who owns a server, partially controls the network, and may divert the browser's "*Init.TLS*" request to his server. He may attempt a man-in-the-middle attack aimed at camouflaging his server as the potential victim's. So, the intruder can build a self-issued certificate, namely a new one that is signed by himself, and may possess a valid certificate for its server, namely one signed by a certification authority. To attempt his attack, the intruder interposes between the browser and the honest server that a user requests through the browser, and replaces the server certificate with his self-issued one. Certificate validation will then fail, but if the browser still concludes the transaction, perhaps due to some interaction with the user, then the man-in-the-middle attack succeeds.

User Model: It is the user who interacts with a browser may be variously skilled and educated, and may be influenced by a huge variety of local or global cultural values. Further, some may exhibit a cautious attitude, others a curious one while they engage with a browser. Capturing the complexities of user behaviour by a formal model is in fact an open issue, and its feasibility is often questioned. As the best approximation that we can envisage at present, a user is modelled as a non-deterministic entity. CSP# exhibits appropriate constructs to easily support this choice. This means that the user may potentially choose any of the paths of interaction that the browser offers. It logically follows that this is the weaker assumption about the user skills, and a ceremony that is secure for a non-deterministic user, will also be secure for any user.

C. Socio-Technical Security Properties

We describe four different security properties. Each property focuses on the behaviour of the browser, which is triggered by both user and server input. They bind elements that span from TLS session to user choices, and we thus define them as socio-technical security properties. The four properties are not intended to be comprehensive. They rather aim to provide an overview on how technical mechanisms implemented in browsers affect the human user. They are described in English, while their version as LTL formulas can be found in [**?**].

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 in Sect. III-B 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 thus 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. (Of course, the value of the additional indication is inversely proportional to the likeliness that an honest server that used a self-issued certificate on the former session purchases, at some point, a valid one and uses it on the latter session.)

D. Results and Discussion

We have run the tool on each of our four browsers, and on each of our four target properties, reaching sixteen interesting findings. It was possible to encode the properties without incurring into state explosion. Over today's standard workstation, an Intel I7 processor running Microsoft Windows 8 with 4GB RAM, PAT outputs in a few seconds on each of the sixteen experiments. The longest runtime is for Property 3 over Firefox, 14.5 seconds. These runtimes are encouraging that our prototype methodology to tackle socio-technical properties is practical and can be usefully boosted further.

Interpreting the tool output required some effort, and Table I summarises the findings. It can be seen that the browser that verified the highest number of properties is Chrome, scoring three ticks, then comes Internet Explorer, with two, and Firefox, with one. However, this numerical classification is not sufficiently meaningful without a glimpse at why the properties are verified or not. This turns out to be particularly important for Property 2.

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 the browser does not involve 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, without informing the user.

Property 2 turns out the most tricky. 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. This is because the property is a logical implication whose precondition is trivially falsified by the browsers that do no store server certificate. Surprisingly, the property does not hold on 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. In support of this, the tool exhibits the following counterexample. In one session, the user engages with an honest server that transmits a self-issued certificate; the browser warns the user about this invalid certificate; the user decides to store the certificate and continue; in a second session, the user engages with the intruder who sends a self-issued certificate that mentions the honest server (thus without domain name verification failure); the browser warns the user also about this second invalid certificate and fails to check that another certificate was already stored for the same server; the user decides to store also this certificate and continue.

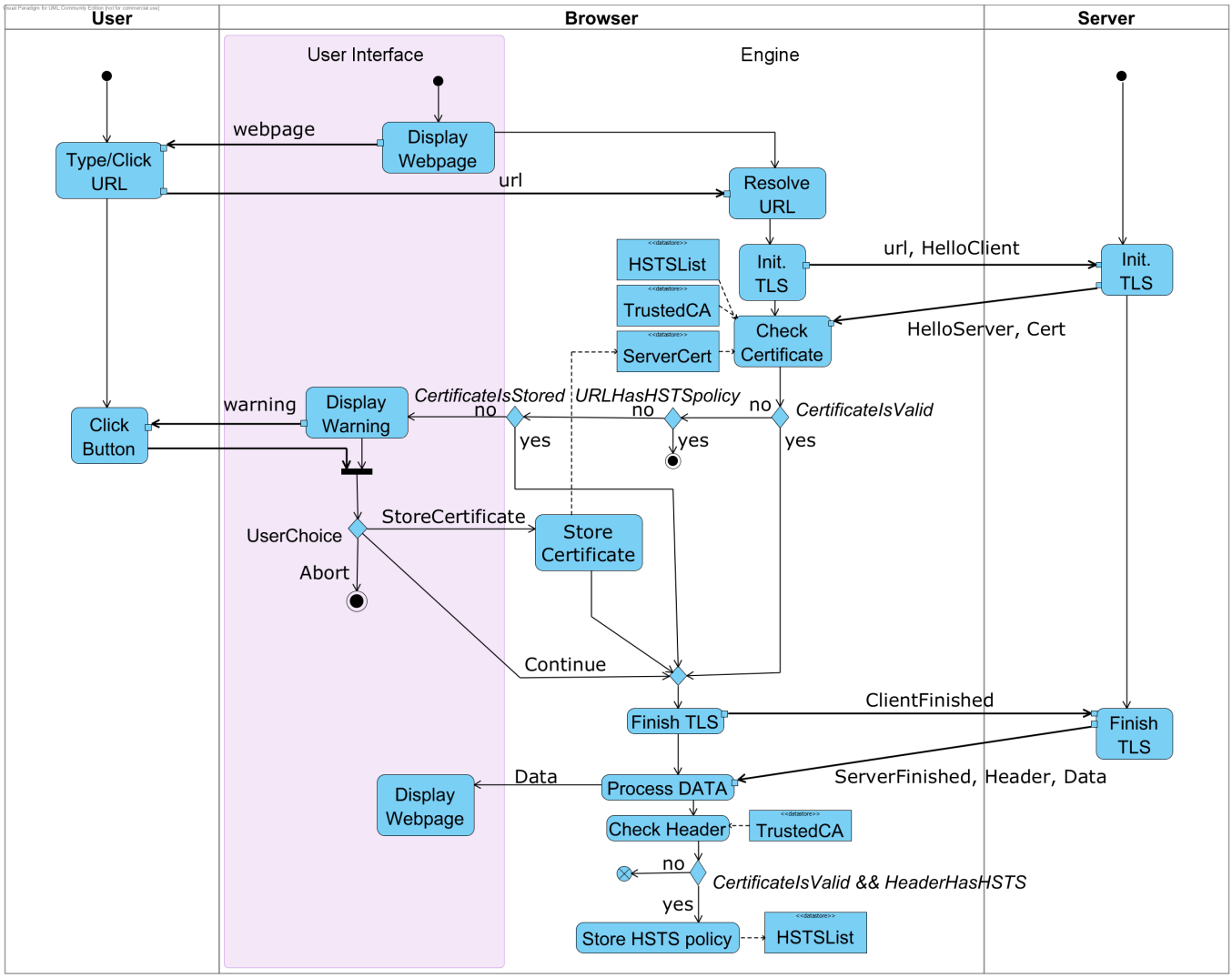

Fig. 3: Activity diagram for certificate validation in Firefox

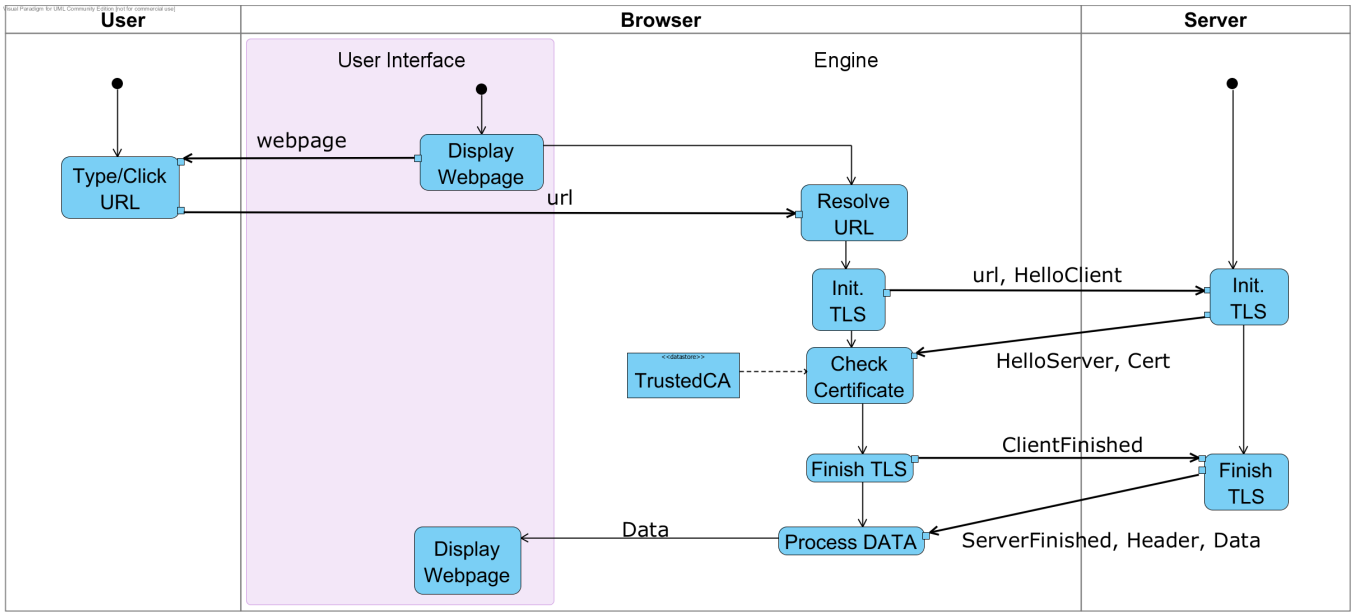

Fig. 4: Activity diagram for certificate validation in Opera Mini

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

TABLE I: Four socio-technical properties studied over the four leader browsers. An ✓ indicates that the property holds, an × indicates that it does not.

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. This denounces the stateless philosophy whereby browsers do not record warnings they may have given in the past, hence they cannot leverage upon them at present. In fact, browsers should maintain a cache of invalid certificate hashes. In doing so, it would be possible for browsers to strongly warn users when a different invalid certificate is presented by a server with which the browser has communicated in the past. It is worth noting that looking at past interactions is the strategy that Session Description Protocol [**?**] advances to strengthen the management of self-issued certificates. Surprisingly, it has not been used in HTTPS.

VI. Conclusions and Future Work

The socio-technical analysis of the security of modern browsers is yet to be considered innovative at present. It combines traditional analysis of the technologies underlying browsers on one hand, with elements of user participation on the other. 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.

This paper described our work in this area. It focused on server authentication with the user via the browser and, more specifically, on TLS certificate validation and the various scenarios where this validation fails. Analysing this property from a socio-technical standpoint inspires a number of questions, and we concentrated on three (cf. Sect. I). To address these questions we formulated four properties that tackle how users are involved in TLS certificate validation.

A major hallmark throughout our work was the adoption of UML activity diagrams as a semi-formal language to represent portions of browser functioning compactly, so that the human analyser could quickly get to grasps with their niceties. However, aiming at automatic formal analysis, the diagrams were used to build a formal model in CSP# to input to the PAT model checker together with an LTL specification of the properties. These are the main steps of our prototype methodology for the socio-technical formal analysis of the security of browsers. The current findings encourage us to develop this methodology further, for example by automating it more fully, and by trying it out on additional socio-technical properties.

References

[1] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, 2008.

[2] E. Rescorla, "HTTP Over TLS," RFC 2818, 2000.

[3] D. Gollmann, "What do we mean by Entity Authentication?" in *Proc. of IEEE SSP'96*, 1996, pp. 46–54.

[4] G. Bella, R. Giustolisi, and G. Lenzini, "A Socio-Techinical Understanding of TLS Certificate Validation," in *Proc. of IFIPTM '13*. Springer, (to appear).

[5] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying wolf: An empirical study of SSL warning effectiveness," in *Proc. of USENIX'09*, 2009.

[6] R. Biddle, P. C. van Oorschot, A. S. Patrick, J. Sobey, and T. Whalen, "Browser interfaces and extended validation SSL certificates: an empirical study," in *Proc. of the ACM CCSW'09*. ACM, 2009, pp. 19–30.

[7] C. Ellison, "Ceremony design and analysis," *IACR eprint*, 2007.

[8] G. Bella and L. Coles-Kemp, "Layered Analysis of Security Ceremonies," in *Information Security and Privacy Research SE-23*, ser. IFIP Advances in ICT. Springer, 2012, vol. 376, pp. 273–286.

[9] [Online]. Available: http://gs.statcounter.com/

[10] [Online]. Available: http://www.omg.org/spec/UML/2.4.1/

[11] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "PAT: Towards Flexible Verification under Fairness," in *Proc. of CAV'09*, ser. LNCS, vol. 5643. Springer, 2009, pp. 709–714.

[12] M. Georgiev, S. Iyengar, S. Jana, R. A., D. Boneh, and V. Shmatikov, "The most dangerous code in the world: validating SSL certificates in non-browser software," in *Proc. of ACM CCS'12*, 2012, pp. 38–49.

[13] S. Flinn and J. Lumsden, "User perceptions of privacy and security on the web," in *Proc. of PST '05*, 2005.

[14] D. Akhawe, A. Barth, P. E. Lam, J. Mitchell, and D. Song, "Towards a Formal Foundation of Web Security," *IEEE CSF'10*, pp. 290–304.

[15] T. Groß, B. Pfitzmann, and A.-R. Sadeghi, "Browser model for security analysis of browser-based protocols," in *Proc. of ESORICS'05*. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 489–508.

[16] A. Josang, K. A. Varmedal, C. Rosenberger, and R. Kumar, "Service provider authentication assurance," in *Proc. of PST '12*. IEEE Computer Society, 2012, pp. 203–210.

[17] S. Gajek, M. Manulis, A. Sadeghi, and J. Schwenk, "Provably secure browser-based user-aware mutual authentication over TLS," *Proc. of ACM ASIACCS '08*, p. 300, 2008.

[18] D. Kaminsky, M. L. Patterson, and L. Sassaman, "PKI layer cake: new collision attacks against the global x.509 infrastructure," in *Proc. of the FC'10*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 289–303.

[19] P. Yee, "Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile," RFC 6818, 2013.

[20] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, 2008.

[21] [Online]. Available: https://akologin.us.army.mil

[22] N. Vratonjic, J. Freudiger, V. Bindschaedler, and J.-P. Hubaux, "The Inconvenient Truth About Web Certificates," in *Economics of Information Security and Privacy III*, B. Schneier, Ed. Springer, 2013.

[23] [Online]. Available: http://blog.chromium.org/2011/06/new-chromium-security-features-june.html

[24] [Online]. Available: http://www.lsv.ens-cachan.fr/Software/spore/

[25] [Online]. Available: https://sites.google.com/site/sarogiustolisi/cabinet/PST2013_CSP_code.tar.gz

[26] C. A. R. Hoare, "Communicating Sequential Processes," *Commun. ACM*, vol. 21, no. 8, pp. 666–677, 1978.

[27] I. Abdelhalim, S. Schneider, and H. Treharne, "An integrated framework for checking the behaviour of fUML models using CSP," *Int. J. on Software Tools for Technology Transfer*, 2012.

[28] J. Lennox, "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)," RFC 4572, 2006.