Surviving the Web: A Journey into Web Session Security

(Extended Abstract)

Stefano Calzavara
Università Ca' Foscari Venezia
stefano.calzavara@unive.it

Riccardo Focardi
Università Ca' Foscari Venezia
focardi@unive.it

Marco Squarcina
Università Ca' Foscari Venezia
squarcina@unive.it

Mauro Tempesta
Università Ca' Foscari Venezia
tempesta@unive.it

ABSTRACT

We survey the most common attacks against web sessions, i.e., attacks which target honest web browser users establishing an authenticated session with a trusted web application. We then review existing security solutions which prevent or mitigate the different attacks, by evaluating them along four different axes: protection, usability, compatibility and ease of deployment. Based on this survey, we identify five guidelines that, to different extents, have been taken into account by the designers of the different proposals we reviewed. We believe that these guidelines can be helpful for the development of innovative solutions approaching web security in a more systematic and comprehensive way.

KEYWORDS

Web sessions; HTTP cookies; web attacks; web defenses

ACM Reference Format:
Stefano Calzavara, Riccardo Focardi, Marco Squarcina, and Mauro Tempesta. 2018. Surviving the Web: A Journey into Web Session Security: (Extended Abstract). In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3184558.3186232

1 INTRODUCTION

The Web is the primary access point to on-line data and applications. It is extremely complex and variegate, as it integrates a multitude of dynamic contents by different parties to deliver the greatest possible user experience. This heterogeneity makes it very hard to effectively enforce security, since putting in place novel security mechanisms typically prevents existing websites from working correctly or negatively affects the user experience, which is generally regarded as unacceptable, given the massive user base of the Web. However, this continuous quest for usability and backward compatibility had a subtle effect on web security research: designers of new defensive mechanisms have been extremely cautious and the large majority of their proposals consists of very local patches against very specific attacks. This piecemeal evolution hindered a deep understanding of many subtle vulnerabilities and problems, as testified by the proliferation of different threat models against which different proposals have been evaluated, occasionally with quite diverse underlying assumptions. It is easy to get lost among the multitude of proposed solutions and almost impossible to understand the relative benefits and drawbacks of each single proposal without a full picture of the existing literature.

In this work, we take the delicate task of performing a systematic overview of a large class of common attacks targeting the current Web and the corresponding security solutions proposed so far. We focus on attacks against *web sessions*, i.e., attacks which target honest web browser users establishing an authenticated session with a trusted web application. This kind of attacks exploits the intrinsic complexity of the Web by tampering, e.g., with dynamic contents, client-side storage or cross-domain links, so as to corrupt the browser activity and/or network communication. Our choice is motivated by the fact that attacks against web sessions cover a very relevant subset of serious web security incidents [14] and many different defenses, operating at different levels, have been proposed to prevent these attacks.

We consider typical attacks against web sessions and we systematise them based on: (*i*) their attacker model and (*ii*) the security properties they break. This first classification is useful to understand precisely which intended security properties of a web session can be violated by a certain attack and how. We then survey existing security solutions and mechanisms that prevent or mitigate the different attacks and we evaluate each proposal with respect to the security guarantees it provides. When security is guaranteed only under certain assumptions, we make these assumptions explicit. For each security solution, we also evaluate its impact on both *compatibility* and *usability*, as well as its *ease of deployment*. These are important criteria to judge the practicality of a certain solution and they are useful to understand to which extent each solution, in its current state, may be amenable for a large-scale adoption on the Web. Moreover, since there are several proposals in the literature which aim at providing robust safeguards against multiple attacks, we also provide an overview of them. For each of these proposals, we discuss which attacks it prevents with respect to the attacker model considered in its original design and we assess its adequacy according to the criteria described above.

Finally, we synthesize from our survey a list of five guidelines that, to different extents, have been taken into account by the designers of the different solutions. We observe that none of the existing proposals follows all the guidelines and we argue that this is due to the high complexity of the Web and the intrinsic difficulty

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3186232

in securing it. We believe that these guidelines can be helpful for the development of innovative solutions approaching web security in a more systematic and comprehensive way.

Reference to Journal Publication. This extended abstract provides a short overview of a published journal paper with the same title. We refer to the original publication [8] for full details.

2 ATTACKING WEB SESSIONS

A *web session* is a semi-permanent information exchange between a browser and a web server that involves multiple requests and responses. Since HTTP is a stateless protocol, i.e., each request is treated independently from all the others, the vast majority of web applications use *cookies* to implement stateful sessions. When the user successfully authenticates to a website, a fresh cookie is generated by the server and sent back to the browser. Further requests originating from the browser automatically include the cookie as a proof of being part of the session established upon password-based authentication.

The cookie plays the role of the password in all the subsequent requests to the web server, thus it is enough to leak its value to hijack the session and fully impersonate the user, without compromising the network connection or the server. We call *authentication cookie* any cookie which identifies a web session.

2.1 Security Properties

We consider two standard security properties formulated in the setting of web sessions. They represent typical targets of web session attacks:

- *Confidentiality*: data transmitted inside a session should not be disclosed to unauthorized users;
- *Integrity*: data transmitted inside a session should not be modified or forged by unauthorized users.

Interestingly, the above properties are not independent and a violation of one might lead to the violation of the other. For instance, compromising session confidentiality might reveal authentication cookies, which would allow the attacker to perform arbitrary actions on behalf of the user, thus breaking session integrity.

2.2 Threat Model

We focus on two main families of attackers: *web attackers* and *network attackers*. A web attacker controls at least one web server that responds to any HTTP(S) requests sent to it with arbitrary malicious contents chosen by the attacker. We assume that a web attacker can obtain trusted HTTPS certificates for all the web servers under his control and is able to exploit content injection vulnerabilities on trusted websites. A slightly more powerful variation of the web attacker, known as the *related-domain attacker*, can also host malicious web pages on a domain sharing a "sufficiently long" suffix with the domain of the target website [5]. This additional capability allows the attacker to set valid (domain) cookies for the target website that are indistinguishable from the ones set by the legitimate site [3]. Network attackers extend the capabilities of traditional web attackers with the ability of inspecting, forging and corrupting all the HTTP traffic sent on the network, as well as the HTTPS traffic using untrusted certificates.

2.3 Attacks Overview

We provide now an overview of the most important attacks against web sessions. For each class of attacks, we report the security properties of web sessions which it violates.

2.3.1 Content Injection (XSS). This wide class of attacks allows a web attacker to inject harmful contents into trusted web applications. Content injections can be mounted in many different ways, but they are always enabled by an improper or missing sanitization of some attacker-controlled input in the web application. These attacks are traditionally assimilated to Cross-Site Scripting (XSS), i.e., injections of malicious JavaScript code; however, the lack of a proper sanitization may also affect HTML contents (markup injection) or even CSS rules [13, 20]. Content injection attacks are usually classified as either *reflected* or *stored*, depending on the persistence of the threat. In the reflected variant, part of the input supplied in the request is "reflected" into the response without proper sanitization. Stored attacks, instead, are those where the injected content is permanently saved on the target server, e.g., in a message appearing on a discussion board. The malicious content is then automatically interpreted by any browser which visits the attacked page.

Since content injections allow an attacker to sidestep the same-origin policy (SOP), which is the baseline security policy of standard web browsers, they can have catastrophic consequences on both the confidentiality and the integrity of a web session. Specifically, they can be used to steal sensitive data from trusted websites, such as authentication cookies and user credentials, and to actively corrupt the page contents, so as to undermine the integrity of a web session.

2.3.2 Cross-Site Request Forgery (CSRF). A CSRF is an instance of the "confused deputy" problem [12] in the context of web browsing. In a CSRF, the attacker forces the user browser into sending HTTP(S) requests to a website where the user has already established an authenticated session, e.g. by including some `<img>` tags to the vulnerable website in pages hosted by the attacker. When rendering these HTML elements, the browser will send HTTP(S) requests to the target website that automatically include the authentication cookies of the user. From the target website perspective, these forged requests are indistinguishable from legitimate ones and thus they can be abused to trigger a dangerous side-effect, e.g., to force a bank transfer to the attacker account.

A CSRF attack allows the attacker to inject an authenticated message into a session with a trusted website, hence it constitutes a threat to session integrity. It is less known that CSRFs may also be employed to break confidentiality by sending cross-site requests that return sensitive user data bound to the user session, e.g., as in the attack against the cloud service SpiderOak [2].

2.3.3 More Attacks. Other less known web attacks include *login CSRF*, *cookie forcing* and *session fixation*. We refer to the full survey [8] for an explanation of them, as well as of traditional *network attacks* enabled by the (partial) adoption of HTTP.

3 DEFENDING WEB SESSIONS

In this section we present the criteria adopted to evaluate the solutions proposed in the literature and by standards to protect from attacks against web sessions. We also discuss a couple of well-known

defenses against content injection, namely `HttpOnly` cookies and CSP. In [8] we discuss more than 40 solutions that either tackle specific web attacks or try to provide a more comprehensive degree of protection from different threats.

3.1 Evaluation Criteria

We evaluate existing defenses along four different axes:

(1) *protection*: we assess the effectiveness of the proposed defense against the conventional threat model of the attack. If the proposal does not prevent the attack in the most general case, we discuss under which assumptions it may still be effective;
(2) *usability*: we evaluate whether the proposed mechanism affects the end-user experience, for instance by impacting on the perceived performances of the browser or by involving the user into security decisions;
(3) *compatibility*: we discuss how well the defense integrates into the web ecosystem with respect to the current standards, the expected functionalities of websites, and the performances provided by modern network infrastructures. For example, solutions that prevent some websites from working correctly are not compatible with the existing Web. On the other hand, a minor extension to a standard protocol which does not break backward compatibility is acceptable;
(4) *ease of deployment*: we consider how practical would be a large-scale deployment of the defensive solution by evaluating the overall effort required by web developers and system administrators for its adoption.

3.2 Two Defenses for Content Injection

3.2.1 HttpOnly Cookies. `HttpOnly` cookies have been introduced in 2002 with the release of Internet Explorer 6 SP1 to prevent the theft of authentication cookies via content injection attacks. Available on all major browsers, this simple yet effective mechanism limits the scope of cookies to HTTP(S) requests, making them unavailable to malicious JavaScript injected in a trusted page.

The protection offered by the `HttpOnly` attribute is only limited to the theft of authentication cookies. The presence of the attribute is transparent to users, hence it has no impact on usability. Also, the attribute perfectly fits the web ecosystem in terms of compatibility with legacy web browsers, since unknown cookie attributes are ignored. Finally, the solution is easy to deploy, assuming there is no need of accessing authentication cookies via JavaScript for generic reasons [21].

3.2.2 Content Security Policy. The Content Security Policy (CSP) [16] is a web security policy standardized by W3C that allows to specify the origins from which the browser is permitted to fetch the resources embedded in a web page. The policy is communicated to the browser via an HTTP header and it is fairly granular, allowing one to distinguish between different types of resources, such as JavaScript, CSS and XHR targets. By default, CSP does not allow inline scripts and CSS directives (which can be used for data exfiltration) and the usage of particularly harmful JavaScript functions (e.g., `eval`). However, these constraints can be disabled by using the `unsafe-inline` and the `unsafe-eval` rules. With the introduction of CSP Level 2 [17], it is possible to selectively white-list inline resources without allowing indiscriminate content execution.

When properly configured, CSP provides an effective defense against XSS attacks. Still, general content injection attacks are not prevented. CSP policies are written by web developers and transparent to users, so their design supports usability. Compatibility and deployment cost are better evaluated together for CSP. On the one hand, it is easy to write a very lax policy which allows the execution of inline scripts and puts only mild restrictions on cross-origin communication: this ensures compatibility. On the other hand, an effective policy for legacy applications can be difficult to deploy, since inline scripts and styles should be removed or manually whitelisted, and trusted origins for content inclusion should be carefully identified [18]. As of now, the deployment of CSP is not particularly significant or effective [9, 19]. That said, the standardization of CSP by the W3C suggests that the defense mechanism is not too hard to deploy on many websites, at least to get some limited protection.

4 PERSPECTIVE

Having examined different proposals, we now identify five guidelines for the designers of novel web security mechanisms. This is a synthesis of sound principles and insights which have, to different extents, been taken into account by all the designers of the proposals we surveyed.

4.1 Guidelines

4.1.1 Transparency. We call *transparency* the combination of high usability and full compatibility: we think this is the most important ingredient to ensure a large scale deployment of any defensive solution for the Web, given its massive user base and its heterogeneity. It is well-known that security often comes at the cost of usability and that usability defects ultimately weaken security, since users resort to deactivating or otherwise sidestepping the available protection mechanisms [15]. The Web is extremely variegate and surprisingly fragile even to small changes: web developers who do not desire to adopt new defensive technologies should be able to do so, without any observable change to the semantics of their web applications when these are accessed by security-enhanced web browsers; dually, users who are not willing to update their web browsers should be able to seamlessly navigate websites which implement cutting-edge security mechanisms not supported by their browsers.

All the security decisions must be ultimately taken by web developers. On the one hand, users are not willing or do not have the expertise to be involved in security decisions. On the other hand, it is extremely difficult for browser vendors to come up with "one size fits all" solutions which do not break any website. Motivated web developers, instead, can be fully aware of their web application semantics, thoroughly test new proposals and configure them to support compatibility.

4.1.2 Security by Design. Supporting the current Web and legacy web applications is essential, but developers of new websites should be provided with tools which allow them to realize applications which are secure *by design*. Our feeling is that striving for backward compatibility often hinders the creation of tools which could actually improve the development process of new web applications.

Indeed, backward compatibility is often identified with problem-specific patches to known issues, which developers of existing websites can easily plug into their implementation to retrofit it. The result is that developing secure web applications using the current technologies is a painstaking task, which involves actions at many different levels. Developers should be provided with tools and methodologies which allow them to take security into account from the first phases of the development process. This necessarily means deviating from the low-level solutions advocated by many current technologies, to rather focus on more high-level security aspects of the web application, like the definition of principals and their trust relations, the identification of sensitive information, etc.

4.1.3 Ease of Adoption. Server-side solutions should require a limited effort to be understood and adopted by web developers. For instance, the usage of frameworks which automatically implement recommended security practices, often neglected by web developers, can significantly simplify the development of new secure applications. For client-side solutions, it is important that they work out of the box when they are installed in the user browser: proposals which are not fully automatic are going to be ignored or misused. Any defensive solution which involves both the client and the server is subject to both the previous observations. Since it is unrealistic that a single protection mechanism is able to accommodate all the security needs, it is crucial to design the defensive solution so that it gracefully interacts with existing proposals which address orthogonal issues and which may already be adopted by web developers.

4.1.4 Declarative Nature. To support a large-scale deployment, new defensive solutions should be *declarative* in nature: web developers should be given access to an appropriate policy specification language, but the enforcement of the policy should not be their concern. Security checks should not be intermingled with the web application logic: ideally, no code change should be implemented in the web application to make it more secure and a thorough understanding of the web application code should not be necessary to come up with reasonable security policies. This is dictated by very practical needs: existing web applications are huge and complex, are often written in different programming languages and web developers may not have full control over them.

4.1.5 Formal Specification and Verification. Formal models and tools have been recently applied to the specification and the verification of new proposals for web session security [1, 4, 7, 10]. While a formal specification may be of no use for web developers, it assists security researchers in understanding the details of the proposed solution. Starting from a formal specification, web security designers can be driven by the enforcement of a clear *semantic* security property, e.g., non-interference [11] or session integrity [7], rather than by the desire of providing ad-hoc solutions to the plethora of low-level attacks which currently target the Web.

This is not merely a theoretical exercise, but it has clear practical benefits. First, it allows a comprehensive identification of *all* the attack vectors which may be used to violate the intended security property, thus making it harder that subtle attacks are left undetected during the design process. Second, it forces security experts to focus on a rigorous threat model and to precisely state all the assumptions underlying their proposals: this helps making a critical comparison of different solutions and simplifies their possible integration. Third, more speculatively, targeting a property rather than a mechanism allows to get a much better understanding of the security problem, thus fostering the deployment of security mechanisms which are both more complete and easier to use for web developers.

4.2 Discussion

Retrospectively looking at the solutions we reviewed, we identify a number of carefully crafted proposals which comply with several of the guidelines we presented. Perhaps surprisingly, however, *none* of the proposals complies with all the guidelines. We argue that this is not inherent to the nature of the guidelines, but rather the simple consequence of web security being hard: indeed, many different problems at very different levels must be taken into account when targeting the largest distributed system in the world.

In the full version of the survey [8], we discuss what are the most peculiar challenges of the web platform and how they have been driving the evolution of web security research. We also use our guidelines to advocate the adoption of hybrid client/server designs as the most effective architecture for novel web security solutions, recommending the use of formal methods from the first phases of the design process. A very recent survey discusses why and how formal methods can be fruitfully applied to web security and highlights open research directions [6]. The quest for robust solutions for web session security is far from finished: we refer to [8] for a discussion on open problems and novel research directions which are worth investigating.

5 CONCLUSION

We took a retrospective look at different attacks against web sessions and we surveyed the most popular solutions against them. For each solution, we discussed its security guarantees against different attacker models, its impact on usability and compatibility, and its ease of deployment. We then synthesized five guidelines for the development of new web security solutions, based on the lesson learned from previous experiences. We believe that these guidelines can help web security experts in proposing novel solutions which are both more effective and amenable for a large-scale adoption.

REFERENCES

[1] Devdatta Akhawe, Adam Barth, Peifung E. Lam, John C. Mitchell, and Dawn Song. 2010. Towards a Formal Foundation of Web Security. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010*. 290–304.
[2] Chetan Bansal, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, and Sergio Maffeis. 2013. Keys to the Cloud: Formal Analysis and Concrete Attacks on Encrypted Web Storage. In *Proceedings of the 2nd International Conference on Principles of Security and Trust, POST 2013*. 126–146.
[3] Adam Barth. 2011. HTTP State Management Mechanism. http://tools.ietf.org/html/rfc6265. (2011).
[4] Aaron Bohannon and Benjamin C. Pierce. 2010. Featherweight Firefox: Formalizing the Core of a Web Browser. In *USENIX Conference on Web Application Development, WebApps 2010*.
[5] Andrew Bortz, Adam Barth, and Alexei Czeskis. 2011. Origin Cookies: Session Integrity for Web Applications. In *Web 2.0 Security & Privacy Workshop (W2SP 2011)*.
[6] Michele Bugliesi, Stefano Calzavara, and Riccardo Focardi. 2017. Formal methods for web security. *Journal of Logical and Algebraic Methods in Programming* 87 (2017), 110–126.

[7] Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, Wilayat Khan, and Mauro Tempesta. 2014. Provably Sound Browser-Based Enforcement of Web Session Integrity. In *Proceedings of the IEEE 27th Computer Security Foundations Symposium, CSF 2014*. 366–380.

[8] Stefano Calzavara, Riccardo Focardi, Marco Squarcina, and Mauro Tempesta. 2017. Surviving the Web: A Journey into Web Session Security. *ACM Comput. Surv.* 50, 1 (2017), 13:1–13:34.

[9] Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi. 2016. Content Security Problems? Evaluating the Effectiveness of Content Security Policy in the Wild. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security, CCS 2016*. 1365–1375.

[10] Daniel Fett, Ralf Küsters, and Guido Schmitz. 2014. An Expressive Model for the Web Infrastructure: Definition and Application to the Browser ID SSO System. In *Proceedings of the 35th IEEE Symposium on Security and Privacy, S&P 2014*. 673–688.

[11] Willem De Groef, Dominique Devriese, Nick Nikiforakis, and Frank Piessens. 2012. FlowFox: a Web Browser with Flexible and Precise Information Flow Control. In *Proceedings of the 19th ACM Conference on Computer and Communications Security, CCS 2012*. 748–759.

[12] Norman Hardy. 1988. The Confused Deputy (or why capabilities might have been invented). *Operating Systems Review* 22, 4 (1988), 36–38.

[13] Mario Heiderich, Marcus Niemietz, Felix Schuster, Thorsten Holz, and Jörg Schwenk. 2012. Scriptless Attacks: Stealing the Pie Without Touching the Sill. In *Proceedings of the 19th ACM Conference on Computer and Communications Security, CCS 2012*. 760–771.

[14] OWASP. 2013. Top 10 Security Threats. https://www.owasp.org/index.php/Top_10_2013-Top_10. (2013).

[15] Mary Frances Theofanos and Shari Lawrence Pfleeger. 2011. Guest Editors' Introduction: Shouldn't All Security Be Usable? *IEEE Security & Privacy* 9, 2 (2011), 12–17.

[16] W3C. 2012. Content Security Policy. http://www.w3.org/TR/CSP/. (2012).

[17] W3C. 2015. Content Security Policy Level 2. https://www.w3.org/TR/CSP2/. (2015).

[18] Joel Weinberger, Adam Barth, and Dawn Song. 2011. Towards Client-side HTML Security Policies. In *6th USENIX Workshop on Hot Topics in Security, HotSec 2011*.

[19] Michael Weissbacher, Tobias Lauinger, and William K. Robertson. 2014. Why Is CSP Failing? Trends and Challenges in CSP Adoption. In *Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2014*. 212–233.

[20] Michal Zalewski. 2011. Postcards From the Post-XSS World. (2011). http://lcamtuf.coredump.cx/postxss/.

[21] Yuchen Zhou and David Evans. 2010. Why Aren't HTTP-only Cookies More Widely Deployed?. In *Web 2.0 Security and Privacy Workshop, W2SP 2010*.