Chapter 10
Improving Client-Side Web Security

In previous chapters of this book, we explained the importance of Web security in general, and more specifically, client-side Web security. We have presented several threat models, each with different capabilities, and have extensively discussed how these attackers threaten the security of Web applications. We have given an overview of the relevant mitigation techniques and highlighted the current state-of-the-art research results. Finally, we have provided details on the current state of practice and formulated best practices to defend Web applications against numerous attacks.

This chapter summarizes the best practices covered earlier in this book, and boils them down to a "must-have" list of security technologies of the modern age. Additionally, we discuss the role of research in client-side Web security, and identify important areas for future research.

10.1 Overview of Best Practices

As most Web security issues are not new, numerous mitigation techniques have been proposed and many of them are supported by mainstream browsers. Unfortunately, the Web has always suffered from legacy software with a slow update cycle, even for extremely critical vulnerabilities. For example, the server-side Heartbleed vulnerability [32], which is considered to be one of the worst Web problems ever, has been patched almost immediately, and as of this writing, 4 months after its disclosure, SSL Pulse [27] still reports 777 popular sites to be vulnerable. A similar story goes for the *HttpOnly* cookie flag, an effective countermeasure with virtually no impact on a Web application, which only sees a 54 % adoption rate among the Alexa top 10,000 sites, 12 years after its introduction.

In order to improve the current state of practice, we give an overview of the most important best practices, which are essential for improving the security of modern Web applications. All of these technologies are widely supported, as can be verified using the helpful *Can I Use* site [7]. While many of the techniques covered below and explained in detail in this book, are applicable for both new and legacy applications; deploying them for legacy applications may be more challenging.

P. De Ryck et al., *Primer on Client-Side Web Security,*
SpringerBriefs in Computer Science, DOI 10.1007/978-3-319-12226-7_10

10.1.1 Secure Communication Channel

The lack of a secure communication channel is an enabling factor for numerous other attacks, such as session hijacking, compromising script inclusions, etc. Therefore, the most important best practice is deploying Web applications over a properly configured Transport Layer Security (TLS) channel, a measure not only useful for new Web applications but also for legacy applications. Several resources offer detailed insights into a proper TLS configuration [28, 29, 33], of which the following attention points are most relevant for Web applications:

- Deploy the latest version of TLS, using good cipher suites that offer perfect forward secrecy.
- Avoid using mixed Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS) content, as HTTP content is easily manipulated on the network.
- Use *Strict Transport Security* [12] for HTTPS-only deployments, to prevent a potentially forged HTTP request from ever leaving the browser.
- Mark all cookies that are used over an HTTPS connection as *Secure*, to prevent cookies from being leaked over (forged) HTTP requests.

A very useful tool for verifying the configuration of a TLS deployment is Qualys' SSL Labs Web site [26], which checks your deployment for common vulnerabilities, insecure ciphers, and misconfiguration.

The technologies listed above are currently available in modern browsers, but several promising technologies are still in active development and are worth keeping track of. The most promising technologies aim to address the forging of TLS certificates, using either public key pinning [9], or by using DNSSEC records to certify the key associated with the certificate (DANE) [13].

10.1.2 Application-level Techniques

Many of the attacks discussed in previous chapters can be mitigated on the application level. These mitigation techniques are often supported by popular Web application frameworks or offered by useful libraries. We identify two classes of techniques: design-level techniques that prevent vulnerabilities by design and locally applicable code-level techniques that actively mitigate specific attacks.

On the design level, an important best practice is the use of a multifactor authentication system. Such systems significantly increase the security of the authentication process, largely mitigating phishing scams, brute-forcing attacks, or the theft of credentials. By using an authentication provider, multifactor authentication can be integrated into your own authentication process, or the whole authentication process can be outsourced. Additionally, several well-known providers allow traditional authentication on trusted machines and only enable multifactor authentication in other

scenarios. A second, related, design-level best practice is to protect sensitive operations with a reauthentication request. This practice prevents a user from performing unintended operations, for example when misdirected in a clickjacking attack.

On the code level, a developer can take several countermeasures to tighten an application's security, effectively mitigating many attack scenarios. We give a brief overview of the most common code-level countermeasures that should be applied in any Web application:

- Context-sensitive sanitization of outputs is crucial in preventing injection vulnerabilities. This should be the first line of defense against cross-site scripting and scriptless injection attacks, potentially supplemented with a strict Content Security Policy (CSP), as discussed in the next section.
- In order to prevent forged requests, token-based approaches are an effective mitigation technique. Every sensitive operation should be authorized by a token, to ensure its authenticity. Additionally, Web applications should reject cross-origin requests when they are unexpected, which can be checked using the `Origin` header.
- By renewing the session identifier after a change in privilege, session fixation attacks can be effectively mitigated, and the scope of session hijacking attacks can be limited.
- Web application developers should be aware that when they include third-party scripts, they implicitly trust the third party to be non-malicious, and remain free of compromise. The risk of a compromise of a third party automatically spreading to your Web application can be reduced by placing the third-party code within the origin of the Web application.

10.1.3 Security Policies

Server-driven, browser-enforced policies inform the browser about the application's behavior, enabling the browser to block any deviating action, which are potentially malicious. As a best practice, we recommend the use of three widely supported policies, discussed below: the *HttpOnly* restriction on cookies, the use of a strict framing policy, and the use of a strict CSP.

Every cookie issued by a Web application, that is not used by JavaScript within the browser, should be flagged as *HttpOnly*. This applies to most cookies issued today, and should especially be true for cookies holding sensitive tokens, such as session identifiers or authentication tokens.

Framing policies restrict the origins that are allowed to frame the application that defines the policy. By doing so, an application can prevent framing by a malicious Web page, which may be trying to misdirect the user, for example using a clickjacking attack. Modern browsers support two framing policies, the *X-Frame-Options* policy [31] and the *frame-ancestors* directive in CSP [36], of which the latter is the more expressive. Restricting the set of origins that is allowed to frame a page may not

always be possible. In that case, the application should restrict framing on all possible pages and ensure that only non-sensitive pages can be framed by any origin. In the near future, the upcoming UI Security specification [20] will offer fine-grained heuristics to determine the legitimacy of the user's interactions.

CSP [36] mainly aims at preventing actions triggered by an attacker who injects content into the application page, of which cross-site scripting is a well-known example. A CSP policy is not meant as a primary defense mechanism against injection attacks but merely aims at restraining an attacker that manages to break through the existing injection defenses. To effectively prevent injection attacks, CSP needs to disable inline scripting, a practice many applications depend on, for example, when defining JavaScript handlers in attributes. Due to this dependency, CSP may be less suited to retrofit to legacy applications but is certainly a viable option for newly developed applications, which can take this into account. Additionally, the upcoming version of CSP [2] will support script nonces, which allow predefined script blocks to be executed, even when placed inline.

10.2 Research-driven Security Technology

Many of the technologies recommended above as a best practice and discussed earlier in this book have resulted from security research. These technologies are an important valorization and dissemination trajectory for research results as they are adopted by mainstream browsers and are essentially deployed on almost all Web-connected machines throughout the world. Finding the right synergy between research results and mainstream is not trivial. Success stories are CSP [35] and Strict Transport Security [16], which went from research proposal to deployment in about a year. On the other side, research results can take several years before being picked up [4] or do not make it at all, as illustrated by the numerous proposals for improving session management [3, 5, 6, 11, 24].

On the other hand, research on currently adopted mechanisms is important to determine the feasibility of certain techniques, especially when deploying them for legacy applications. One example is research on the use of CSP [41], providing insights in the shortcomings of CSP for legacy applications, which in turn drives the next version of the specification [2].

Apart from determining the impact of current security technologies on legacy applications, other research areas are also worth exploring. One ongoing research problem is the integration of potentially untrusted JavaScript into a Web application. Numerous proposals have been made in the past 6 years [1, 15, 19, 21–23, 25, 38, 40], but as of this writing, there is no practical solution ready for deployment. Bringing these valid but often complex proposals towards the modal developer is crucial for ensuring adoption.

Similar to Web technologies and security mechanisms, research is shifting towards the client side. Recent papers aim to detect vulnerable Web sites in the browser [34],

focus on vulnerabilities that only exist at the client side, such as DOM-based cross-site scripting [37] or exploits of new HTML5 APIs [39], and investigate the security of browser extensions [17].

Finally, as TLS becomes more important every day, it receives a significant amount of focus from the research community. Research does not only focus on the cryptographical properties of TLS [8, 10, 30] but also investigates current deployments [14] and proposes countermeasures to prevent attacks such as man-in-the-middle [18]. As TLS currently offers an all-or-nothing solution, cutting out any intermediaries out of the communication channel, interesting research challenges lie in the controlled integration of these intermediaries. Example scenarios are enabling Web caches when using TLS, allowing certain parties to embed content in designated parts of Web pages, and allowing perimeter security solutions to inspect TLS traffic.

10.3 Conclusion

A result from the evolution towards client-enforced security policies is that we now have multiple defensive technologies against specific Web attacks, enabling a defense-in-depth strategy. For example, by deploying several mitigations against cross-site scripting attacks, the harm of a successful cross-site scripting exploit can be severely limited, or even prevented altogether. As the complexity of Web applications grows, legacy systems will need to be protected, such defense-in-depth strategies will become increasingly important.

A final conclusion to draw from this book is that Web security is a continuous race between attackers and defenders, similar to the security of other complex systems. On one hand, we see regular discoveries and disclosures of new attacks, on the other hand, we have a strong research community working on new defenses, as well as security-aware browser vendors incorporating state-of-the-art technologies. Due to this fast pace, it is more important than ever to stay up-to-date with the latest technology, which is precisely the goal of this book.

References

1. Agten, P., Van Acker, S., Brondsema, Y., Phung, P.H., Desmet, L., Piessens, F.: JSand: complete client-side sandboxing of third-party JavaScript without browser modifications. In: Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC), pp. 1–10 (2012)
2. Barth, A., Veditz, D., West, M.: Content security policy level 2. W3C Working Draft (2014)
3. Bortz, A., Barth, A., Czeskis, A.: Origin cookies: session integrity for Web applications. Web 2.0 Security and Privacy (W2SP) (2011)
4. Chen, E.Y., Bau, J., Reis, C., Barth, A., Jackson, C.: App isolation: get the security of multiple browsers with just one. In: Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS), pp. 227–238 (2011)
5. Dacosta, I., Chakradeo, S., Ahamad, M., Traynor, P.: One-time cookies: preventing session hijacking attacks with stateless authentication tokens. ACM Trans. Internet Technol. (TOIT) **12**(1), 31 (2012)

6. De Ryck, P., Desmet, L., Piessens, F., Joosen, W.: Eradicating bearer tokens for session management. W3C/IAB Workshop on Strengthening the Internet Against Pervasive Monitoring (STRINT) (2014)
7. Deveria, A.: Can i use . . . support tables for HTML5, CSS3, etc. http://caniuse.com (2014)
8. Duong, T., Rizzo, J.: BEAST - Here Come The XOR Ninjas. http://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf (2011)
9. Evans, C., Palmer, C., Sleevi, R.: Public Key Pinning Extension for HTTP. IETF Internet Draft (2014)
10. Gluck, Y., Harris, N., Prado, A.: BREACH: reviving the CRIME Attack. http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf (2013)
11. Hallam-Baker, P.: Http integrity header. IETF Internet Draft (2012)
12. Hodges, J., Jackson, C., Barth, A.: HTTP strict transport security (HSTS). RFC Proposed Standard (RFC 6797) (2012)
13. Hoffman, P., Schlyter, J.: The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA. RFC Proposed Standard (RFC 6698) (2012)
14. Huang, L.S., Rice, A., Ellingsen, E., Jackson, C.: Analyzing forged ssl certificates in the wild. In: Proceedings of the 35th IEEE Symposium on Security and Privacy (SP) (2014)
15. Ingram, L., Walfish, M.: Treehouse: Javascript sandboxes to help web developers help themselves. In: Proceedings of the USENIX annual technical conference (ATC) (2012)
16. Jackson, C., Barth, A.: ForceHTTPS: protecting high-security web sites from network attacks. In: Proceedings of the 17th international conference on World Wide Web (WWW), pp. 525–534 (2008)
17. Kapravelos, A., Grier, C., Chachra, N., Kruegel, C., Vigna, G., Paxson, V.: Hulk: eliciting malicious behavior in browser extensions. In: Proceedings of the 23rd USENIX Security Symposium, pp. 641–654 (2014)
18. Karapanos, N., Capkun, S.: On the effective prevention of tls man-in-the-middle attacks in Web applications. In: Proceedings of the 23rd USENIX Security Symposium, pp. 671–686 (2014)
19. Magazinius, J., Phung, P.H., Sands, D.: Safe wrappers and sane policies for self protecting javascript. In: Proceedings of the 15th Nordic Conference on Secure IT Systems (NordSec), pp. 239–255 (2010)
20. Maone, G., Huang, D.L.S., Gondrom, T., Hill, B.: User interface safety directives for content security policy. W3C Last Call Working Draft (2014)
21. Meyerovich, L., Livshits, B.: ConScript: specifying and enforcing fine-grained security policies for Javascript in the browser. In: Proceedings of the 31st IEEE Symposium on Security and Privacy (SP), pp. 481–496 (2010)
22. Mickens, J.: Pivot: fast, synchronous mashup isolation using generator chains. In: Proceedings of the 35th IEEE Symposium on Security and Privacy (SP), pp. 261–275 (2014)
23. Miller, M.S., Samuel, M., Laurie, B., Awad, I., Stay, M.: Caja: safe active content in sanitized javascript. http://google-caja.googlecode.com/files/caja-spec-2008-01-15. pdf (2008)
24. Murdoch, S.J.: Hardened stateless session cookies. Security Protocols XVI, pp. 93–101 (2011)
25. Phung, P.H., Sands, D., Chudnov, A.: Lightweight self-protecting javascript. In: Proceedings of the 4th ACM Symposium on Information, Computer and Communications Security (ASIACCS), pp. 47–60 (2009)
26. Qualys: Qualys SSL labs. https://www.ssllabs.com/ (2014)
27. Qualys: Trustworthy internet movement—ssl pulse. https://www.trustworthyinternet.org/ssl-pulse/ (2014)
28. Ristić, I.: OpenSSL cookbook. Feisty Duck (2013)
29. Ristić, I.: Bulletproof SSL and TLS. Feisty Duck (2014)
30. Rizzo, J., Duong, T.: The CRIME attack. https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit?pli=1#slide=id.g1d134dff_1_222 (2012)
31. Ross, D., Gondrom, T.: HTTP Header Field X-Frame-Options. RFC Informational (RFC 7034) (2013)

32. Schneier, B.: Hearbleed. https://www.schneier.com/blog/archives/2014/04/heartbleed.html (2014)
33. Sheffer, Y., Holz, R., Saint-Andre, P.: Recommendations for Secure Use of TLS and DTLS. IETF Internet Draft (2014)
34. Soska, K., Christin, N.: Automatically detecting vulnerable websites before they turn malicious. In: Proceedings of the 23rd USENIX Security Symposium, pp. 625–640 (2014)
35. Stamm, S., Sterne, B., Markham, G.: Reining in the web with content security policy. In: Proceedings of the 19th international conference on World wide web (WWW), pp. 921–930 (2010)
36. Sterne, B., Barth, A.: Content security policy 1.0. W3C Candidate Recommendation (2012)
37. Stock, B., Lekies, S., Mueller, T., Spiegel, P., Johns, M.: Precise client-side protection against dom-based cross-site scripting. In: Proceedings of the 23rd USENIX Security Symposium, pp. 655–670 (2014)
38. Ter Louw, M., Ganesh, K.T., Venkatakrishnan, V.: AdJail: practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements. In: Proceedings of the 19th USENIX Security Symposium, pp. 371–388 (2010)
39. Tian, Y., Liu, Y.C., Bhosale, A., Huang, L.S., Tague, P., Jackson, C.: All your screens are belong to us: attacks exploiting the html5 screen sharing api. In: Proceedings of the 35th IEEE Symposium on Security and Privacy (SP), pp. 34–48 (2014)
40. Van Acker, S., De Ryck, P., Desmet, L., Piessens, F., Joosen, W.: WebJail: least-privilege integration of third-party components in web mashups. In: Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC), pp. 307–316 (2011)
41. Weinberger, J., Barth, A., Song, D.: Towards client-side html security policies. In: Proceedings of the 6th USENIX Workshop on Hot Topics on Security (HotSec) (2011)