A User-level Authentication Scheme to Mitigate Web Session-Based Vulnerabilities

Bastian Braun[1], Stefan Kucher[1], Martin Johns[2], and Joachim Posegga[1]

[1] Institute of IT-Security and Security Law (ISL)
University of Passau, Germany
{bb,jp}@sec.uni-passau.de, stefan.kucher@web.de
[2] SAP Research
martin.johns@sap.com

Abstract. After the initial login, web browsers authenticate to web applications by sending the session credentials with every request. Several attacks exist which exploit conceptual deficiencies of this scheme, e.g. Cross-Site Request Forgery, Session Hijacking, Session Fixation, and Clickjacking. We analyze these attacks and identify their common root causes in the browser authentication scheme and the missing user context. These root causes allow the attacker to mislead the browser and misuse the user's session context. Based on this result, we present a user authentication scheme that prohibits the exploitation of the analyzed vulnerabilities. Our mechanism works by binding image data to individual sessions and requiring submission of this data along with security-critical HTTP requests. This way, an attacker's exploitation chances are limited to a theoretically arbitrary low probability to guess the correct session image.

1 Introduction

In order to provide personalized services in the World Wide Web, remote applications must identify and authenticate their users. Upon signing up, users generally choose a username and a password that can be used as a shared secret to establish future sessions. After the authentication of the user, the web application assigns a unique temporary token to the user. This token is stored in the browser and subsequently used by the browser and the application to tell this user and others apart. Several attacks target the browser or the token to hijack established sessions. Clickjacking and Cross-Site Request Forgery mislead the victim's browser to send requests that are determined by the attacker. Session Hijacking and Session Fixation aim at sharing the token with the attacker.

In this paper, we introduce a method to authenticate security-sensitive operations. Our approach, named *Session Imagination*, can be applied to existing web applications and mitigates the above mentioned attacks. Specifically, we apply the two steps of identification and authentication to established sessions. After login, the user is equipped with a shared secret that is not stored in his browser. The former universal token then serves as the identification that is complemented by the shared secret as the authentication for security critical operations. The shared secret can not be stolen by an attacker, and the browser can not be lured into misusing the secret.

Our contribution is twofold: we identify the above mentioned attacks' common root causes and provide an applicable solution that implements the well-known and approved concept of identification and authentication to web sessions. This solution remedies basic deficiencies of current web session implementations. We give details about the authentication scheme and its implementation, evaluate the approach, and show that the protection goals are achieved.

In the next section, we explain our solution's background in more detail. We give an overview of authentication in the web and how the focused attacks work. Then, we explore the attacks' common root causes. In Sec. 3, the actual solution, Session Imagination, is presented. We shed light on the user authentication scheme and provide evaluation of the scheme in terms of overhead, usability, and applicability. Then, we show that Session Imagination raises the bar for all four attacks though it can not yet completely prevent all of them. Sec. 4 presents related work before we conclude in Sec. 5.

2 Background

In this section, we explain details of authentication, particularly in web applications, and attacks on authentication and authenticated sessions. We show that these attacks share some common root causes.

2.1 Authentication

Authentication can happen based on something one knows, something one holds, who one is, what one does, or where one is. The most widespread approach is based on knowledge. A shared secret is established between the authenticating system and the legitimate user. The user provides this secret together with his identification (i.e. the claim who he is) in order to authenticate. The security of this approach lies in the fact that the shared piece of information remains secret. Thus, an entity which can provide the secret must be legitimate. Usually, the identification is called 'username' and the shared secret is the 'password' or 'PIN'.

2.2 Authentication tracking in the web

HTTP was designed to be a stateless protocol. Therefore, web applications have to implement their own session tracking on the application layer. For this purpose, session identifiers (SIDs) are used. Every HTTP request that carries the same SID is recognized by the application to belong to the same user session. In general, authentication tracking in web applications is tied to the application's session tracking, i.e., after a successful logon the user's session is marked to be authenticated on the server-side. In consequence, the SID becomes the user's de-facto authentication credential as long as the session is valid.

2.3 Web Session-based Attacks

The vast majority of all existing web applications utilize HTTP cookies for SID transport. This means that the SID, i.e., the user's credential, is locally stored by the browser and automatically added to all HTTP requests which match the cookie's domain value. Several attacks are known that exploit this mechanism.

Session Hijacking: Session Hijacking denotes a class of attacks that strive to take over an existing session. As pointed out in Sec. 2.2, the session token allows access to individualised services. Thus, an attacker aims at knowing the SID. A promising variant is called *Session Hijacking via Cross-Site Scripting (XSS)*. The attacker first performs a XSS attack to steal the user's session ID and finally obtains access to the web application's internal area in his victim's name. The XSS attack is executed with maliciously injected JavaScript code that reads the stored cookies and transmits them to the attacker's site.

Session Fixation: Session Fixation attacks are similar to Session Hijacking attacks. A Session Fixation attacker places an unauthenticated token at the victim's browser, waits until it gets authenticated (i.e. the user logs in), and finally hijacks the session with the known token [8]. The first step, placing the cookie at the victim's browser, can be taken by several approaches [19]. A web application is vulnerable to Session Fixation attacks if it does not renew SIDs after user authentication. So, the attacker can reuse the known SID after the victim logged in.

Cross-Site Request Forgery (CSRF): An attacker can even perform some actions on behalf of a victim without ever getting any knowledge of the SID. He inserts a crafted link into some website that makes the browser send a request to the target web application. For example, the attacker might put `http://www.yourbank.com/transfer.php?from=your_acc&to=my_acc` into an image (`<img>`) tag on any website. Upon visiting this website, the user's browser tries to retrieve a picture and sends the crafted request to the banking website. The attack is successful if the victim is logged into his account at 'yourbank' at the same time. His browser will attach the SID cookie to the request and legitimate the money transfer.

Clickjacking: A Clickjacking attack [15] exploits the fact that the users' perception may vary from the page rendering outcome by the browser. In this attack scenario, the victim is a user that has an account at the target web application. To perform an attack, the attacker prepares a web page that makes the user perform actions on the target web application.

Technically, there are several ways the attacker can take [13]. First, he can load the target web application in a transparent *integrated frame (iframe)* [21] and place it as an additional layer in front of the visible underlying page while at the same time luring the victim into clicking on a particular area by design of his own page. The attacker can make the user perform arbitrary actions as long as these are invokable by mouse clicks. Second, the attacker can include a button from the target web application in his own context. Therefore, he crafts an iframe that contains just the respective button as the only visible part of the target web page. This way, he changes the context of the triggered functionality.

The victim might suppose to invoke a completely different action at a different web application.

In summary, Clickjacking attacks rely on a gap between a session context as it is perceived by the victim and the actual technical session context in the browser.

2.4 The Attacks' Root Causes

The previously described attacks share common root causes:

First, session authentication means authentication performed by the browser. For the user's perception, only one authentication step happens, namely the login where he provides his username and password. The rest of the session handling is transparent to the user. As explained above, HTTP does not have a session feature and, thus, session handling has to be implemented using session identifiers on the application layer. This fallback solution provides authentication of the browser with every request instead of authentication of the user as it would be required. The following example illustrates this fact: One person logs into his account on a web page, then leaves his computer to have a coffee. Every other person could now interact with the web application on behalf of the user logged in because the browser will do transparent authentication. So, as long as the browser maintains the session ID, all requests are authenticated. The same person accesses a terminal next to the coffee maker. He visits the same web application but he will not be able to access his account without another login though he already authenticated towards this web application.

Second, on the opposite side, the server can not distinguish different contexts of a request. On the server side, incoming requests generated by a JavaScript command, an image tag, or the click of a user respectively are all alike. The requests do not contain evidence that they are intended by the user. The server can not decide whether the user is aware of the action that is caused by a request.

To sum up, the common root causes of Session Hijacking, CSRF, Clickjacking, and Session Fixation are in fact *browser authentication* instead of *user authentication* along with the server's unability to determine a request's initiation context.

Browser-level and User-level Authentication: The authentication of HTTP requests can be divided into two classes: browser-level and user-level authentication.

Browser-level authentication is the current practice in web applications, meaning that after the user provided his credentials for login, the authentication token is cached and subsequent requests are implicitly applied by automatically sending the authentication token. In this case, the browser performs authentication on behalf of the user because the user logged in to the personalised service. Examples of implicit, e.g. browser-level, authentication are the above mentioned cookies, client-side SSL authentication, HTTP authentication (basic and digest), and authentication based on the client's IP address.

The other principle is user-level authentication. In this case, another authentication step for a user's requests is added. We require the user's explicit consent to a user-level authentication step such that this step can not be taken by the browser only but additional action by and knowledge of the user is required. Examples of explicit, user-level authentication are re-entering the username and password and passcodes received as text messages.

We identified two attack vectors emerging from browser-level authentication. CSRF and Clickjacking attacks make the browser send a request and authenticate on behalf of the user even though the authenticated user does not acknowledge. This problem is known as the 'confused deputy problem' [3]. The browser stores all secret information that is needed to authenticate the requests. The underlying assumption becomes evident in the attack scenarios: All requests are supposed to be only initiated by deliberate user clicks or by the browser that fetches regular content. This assumption stems from the early days of the World Wide Web where web applications were not personalized. The addition of web sessions and cookies turned this established assumption to a security risk. The web application can not decide whether the user deliberately initiated the requests. Uncommon request sequences may indicate CSRF attacks, Clickjacking attacks simulate regular user sessions and are harder to detect.

While CSRF and Clickjacking are based on requests initiated by the victim's browser and without his consent, there is another attack vector that exploits the fact that browser-stored information can be easily transferred. Session Hijacking and Session Fixation attacks strive to impersonate the user from different machines towards the web application. Both attacks share the same goal, namely the attacker and the victim share the same SID and are thus indistinguishable from the web application's point of view.

Both attack vectors are based on the same conceptual deficiency: Due to browser-level authentication, no user input is needed to supply evidence that the authenticated user intends the requested action. On the opposite, request authentication including user interaction prevents the attack vectors and remedies the conceptual deficiency.

3 Session Imagination

To mitigate the vulnerabilities described in Sec. 2.3, we implemented a new approach for user-level authentication, named *Session Imagination*. Thereby, we focused on overcoming the vulnerabilities' root causes (see Sec. 2.4). In this section, we will describe our solution that aims at mitigating CSRF, Clickjacking, Session Hijacking, and Session Fixation attacks. Session Imagination separates identification and authentication in web sessions and relies on visual authentication tokens which can be easily remembered and recognized by the user while the authentication token is not stored in the browser.

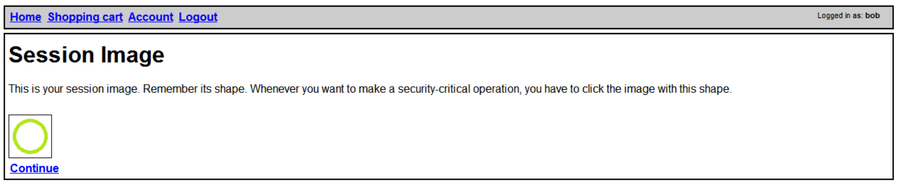
We model the attacker to be a regular web participant. He can send messages, access web applications and set up his own web sites. However, he does neither control the other user's machine or platform nor those of the web application nor the communication infrastructure between them.

3.1 Protection Goals

Our goal is to protect a web application and its users against the attacks described in Sec. 2.3. Protection means that an attacker's chances to reach his goals are limited to an upper bound of probability. The actual upper bound may be configurable. The attacker must not be able to increase this probability. For the sake of completeness, we must say that we aim at securing authentication tracking and do not consider an attacker who owns the login credentials. For example, a phishing attacker gaining knowledge of username and password can still use a protected web application in the victim's name.

3.2 The User-level Authentication Scheme

Session Imagination uses images as per-session user-level authentication tokens. That means that every user is assigned an image upon login. This image is displayed once immediately after login (cf. Fig. 1).



Home Shopping cart Account Logout

Logged in as: bob

Session Image

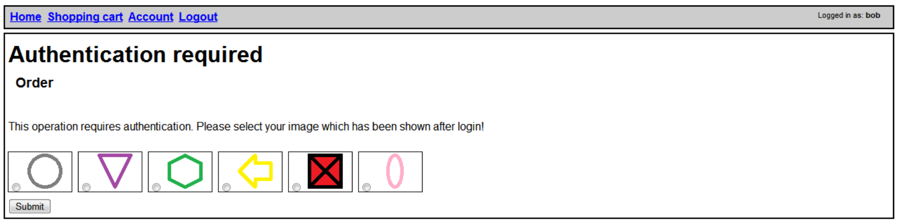
This is your session image. Remember its shape. Whenever you want to make a security-critical operation, you have to click the image with this shape.

Continue

Fig. 1. A fresh session image is given immediately after login. This image has to be remembered throughout the session and identified among a set of images to legitimate security-critical requests.

It is then used together with a conventional session ID in a cookie to authenticate security-critical requests. For example, in an online shop, a set of critical actions is defined, e.g. sending an order or changing the shipping address. Upon requesting such an action, the user has to choose the right image among a given set before the action is executed (cf. Fig. 2). In our example implementation, we used circles, triangles, hexagons, arrows, squares, and ellipses as images. One could also use more usual images like animals, shoes, or hats. We call this intermediate step the 'challenge'. A brief overview of Session Imagination steps is given in Fig. 3.

For every new challenge, the images' shape is slightly varied. That does not affect the user's ability to distinguish the right image from the others but makes simple image recognition, e.g. by automatic hashing, harder. As an example, consider the images in Fig. 4 which represent the same six "classes" as those given in Fig. 2. Differences between two images of the same class can occur in terms of orientation (where appropriate), line color, and fill color. If pictures of animals or items serve as session secret, similar classes can be used. Users are expected to be able to distinguish cats from dogs etc.



Home Shopping cart Account Logout

Logged in as: bob

Authentication required

Order

This operation requires authentication. Please select your image which has been shown after login!

Submit

Fig. 2. Before a critical operation is executed, the respective request has to be authenticated. Therefore, the user has to identify the correct session image.

The next point of image recognisability is the *Uniform Resource Identifier (URI)*. The provided images could be identified by their file names, e.g. `circle1.png`. Given that, an attacker can conclude the image shape from the name which can be stolen by a XSS attack in conjunction with the session cookie. So, we implemented random names for all provided images. The names are re-generated with every response. They serve as one-time passwords that the user does not have to remember because he can identify the correct password by the corresponding image which is valid for the whole session.

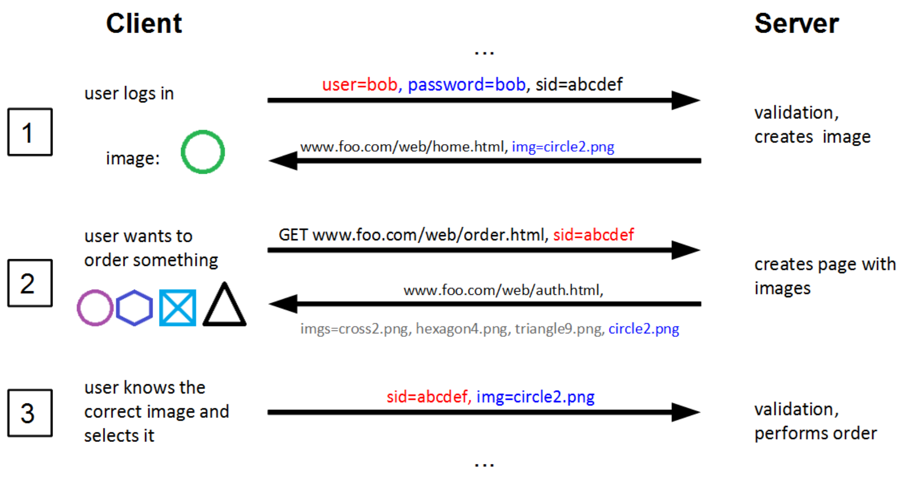
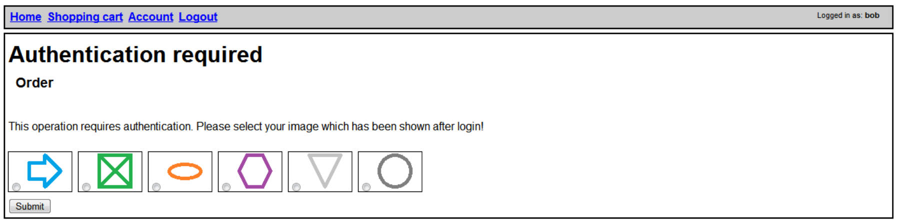

Client
Server
...
1
user logs in
user=bob, password=bob, sid=abcdef
validation,
creates image
image:
www.foo.com/web/home.html, img=circle2.png
2
user wants to order something
GET www.foo.com/web/order.html, sid=abcdef
creates page with images
www.foo.com/web/auth.html,
imgs=cross2.png, hexagon4.png, triangle9.png, circle2.png
3
user knows the correct image and selects it
sid=abcdef, img=circle2.png
validation,
performs order
...

Fig. 3. An overview of authentication steps related to Session Imagination. We used descriptive file names for the pictures for the sake of clarity.

In the run of a XSS attack, the attacker could record the user's click and use a canvas element [20] to prepare an exact copy of the session image. The attacker can choose size 0 x 0 to avoid that the attack is detected by the victim. Next, the canvas is serialized and transmitted to the attacker's domain, e.g. by a hidden form or as a GET parameter. As a countermeasure, the images are integrated as iframes [21] from a different subdomain than the actual web page. The *Same-Origin Policy (SOP)* [16] prevents that the attacker's payload injected in the web page can read the image data.

Finally, the order of images must change with every challenge to avoid recognisability by position, e.g. "always the left most image". In particular, Clickjacking attacks are much easier if the sequence of images is predictable. The examples given in Fig. 2 and Fig. 4 illustrate how the images are re-arranged. The classes that are used in both examples are the same which is necessary to prevent intersection attacks. Otherwise, the attacker could prompt several challenges and compute the intersection. The remaining set must contain the correct class because the user must always be able to choose the correct image. This way, the attacker could reduce the number of candidates with every new challenge.



Authentication required

Order

This operation requires authentication. Please select your image which has been shown after login!

Fig. 4. The actual shapes of the session images vary. This does not lower identifiability by users but prohibits image recognition by hashing.

Session Imagination implements a user-level authentication scheme where the browser is not able to authenticate high-security requests transparently. The conventional separation of identification and authentication is restored. The SID in the cookie serves as a temporal identification while the correct image is the authentication. As we pointed out in Sec. 2.4, a user-level authentication scheme prevents all attacks under consideration.

3.3 Evaluation

The performance evaluation of Session Imagination can be restricted to the measurement of the additional steps required for the authentication of security-critical actions. The restriction to security-critical actions limits the overhead. In our prototype implementation, we considered an online shop as a use case. Putting items to the cart was possible without additional efforts while checking out and changing account information was classified as security-critical. So, for an average shopping trip, only one additional step is necessary.

Next, we come back to the protection goals named in Sec. 3.1. We will show that Session Imagination is able to overcome all of the respective vulnerabilities and, thus, meet the goals. This is achieved by the introduction of identification and authentication for requests to overcome the conceptual deficiency of SID-based authentication.

Session Hijacking and Session Fixation: Session Hijacking and Session Fixation attacks both aim to steal the established session context. Session Imagination does not prevent stealing or setting the session ID. So, we consider the case that the attacker already owns the correct SID. Then, he can act on behalf of his victim unless he faces a challenge where his only chance is guessing the right image. A Session Hijacking attack that makes use of XSS does not increase the attacker's probability. The payload can not access the images because they are served as iframes from a different domain. The right image is not stored on the victim's machine such that the attacker can not steal or set the right image in the same way as the respective cookie.

CSRF: A CSRF attacker can make the victim send a request for a security-critical operation. Though the attacker can generally not read back the application's response, he might know the application and can thus predict the form of the next request. This would be the answer to the challenge. At this point, the attacker not only has to guess the right image among the given ones but he has to guess the right image name which is a dynamic and random string of variable length. This is due to the fact that this string is used as a response parameter to decide whether the user clicked the right image and the attacker can not read the webpage to learn the provided names. In this scenario, the attacker's chances are lower than guessing the right image among the provided ones.

Clickjacking: A Clickjacking attack prohibits the victim's context awareness which is crucial for passing the challenges.

If the attack starts before user authentication, the attacker would have to include the target web application's user login while pretending to log in on the attacker's site. Moreover, the attacker would have to make the victim provide his credentials of the target web application. We consider this to be infeasible.

If the victim is already logged in at the target web application, the attack must fail because the attacker would have to make the victim deliberately click on the session image of another web application. This task can be rendered impossible if the session images contain their web application's context, like the company's logo. If the attacker overlays the images with his own images to hide the context, the attack fails because the attacker can not link the user's session image with the respective attacker image. So, the user ends up clicking an arbitrary image which is equal to guessing the image.

To sum up, in all scenarios, the attacker can not increase his chance higher than the probability to guess the correct image.

Relation to Picture-based Authentication Approaches based on password pictures differ in major aspects from our approach. First, we implement a secret on a session basis. The user thus does not have to remember another persistent password in the form of picture categories. Case studies on the long-term memorability of graphical passwords do not apply to our approach. Moreover, an attacker gaining knowledge of the user's session image can not use this after the user logs out and in again in our approach. Second, the user is not free to choose the picture. This fact avoids that the attacker can take advantage of familiarity with the victim to guess the correct image (e.g. the respective user loves cats).

Relation to CAPTCHAs A CAPTCHA [1] denotes a "Completely Automated Public Turing test to tell Computers and Humans Apart". It is meant to provide a task that can be easily solved by a human but is hard to solve for a computer. However, a CAPTCHA contains all information that is needed to solve the task while the task consists in extracting this information. This would allow an attacker to hijack a session after stealing the session ID.

Relation to Other User-level Authentication Schemes The most widespread approach to make sure that the user is willing to perform the particular action is to require username and password entry again. This, however, is less secure compared to Session Imagination. First, the credentials entry form can be easily spoofed by an attacker (XSS, phishing) which makes the victim provide his confidential login data to the attacker. Second, username and password can be easily stored in the browser. This makes the browser again the storage point of all information needed to hijack sessions. The other common approach is to enter passcodes that have been received via text message. This approach has similar security properties as Session Imagination, e.g. guessing is still theoretically possible and an attacker owning the victim's platform will still succeed. However, this procedure induces additional cost and requires an additional device with GSM connectivity. Due to this fact, mobile and smartphones are excluded from accessing the respective web application. Session Imagination does not require GSM availability and can be used with a single device.

Usability In order to assess the usability of Session Imagination, we conducted a survey. Therefore, we set up an online shop equipped with Session Imagination. 40 users had to provide the correct session image to check out and enter or change the shipping address. We found that 95% of them have never forgotten the correct session image. Next, we asked the test people whether they prefer another password entry (17, 5%), passcodes via SMS (22, 5%), or Session Imagination (47, 5%). The remaining 12, 5% do not like any of these. Nevertheless, 92, 5% would accept additional effort if this protects them from fraud. 47, 5% consider 2-5 challenges acceptable in the course of an online shopping trip where 45% tolerate only 1 challenge. Overall, we can say that a vast majority of all testers accept Session Imagination challenges and prefer this procedure to the alternative approaches.

Decreasing the attacker's chances In our prototype implementation, we presented six images to the user, i.e. an attacker has a chance of 16.67% to guess the right image. More images can reduce the attacker's chances and increase security. As an alternative, a big picture could be presented where the user has to click a certain area to authenticate. The security level then depends on the number of areas. Further, aligned style sheets allow the provider to include many pictures while only some of them are visible to the user. This allows to increase security without lowering usability.

4 Related Work

There is related work in many areas. Session Hijacking prevention either prohibits script access to session cookies [12, 4] or the execution of unauthorized script code [11]. Session Fixation protection strives to renew the SID after authentication [8, 19, 5]. Server-side CSRF protection validates the transmitted referer [2] or request-specific nonces [7]. Client-side approaches strip off authentication information from suspicious requests [6, 17]. Clickjacking [14, 15] attacks can be partially thwarted by HTTP headers [9, 10].

However, all these approaches target only one of the attacks respectively, e.g. they protect against CSRF attacks but can not thwart Clickjacking or combinations of CSRF and XSS [7]. Moreover, some of the standard defenses turned out to not provide the aimed protection level [22, 18].

To the best of our knowledge, there is no web-based approach with the same protection as Session Imagination.

5 Conclusion

In this paper, we have thoroughly examined fundamental deficiencies in today's web session management. We have explained how sessions are established and handled between client and server. We have identified the common root causes of four widespread vulnerabilities, Session Hijacking with Cross-Site Scripting, Session Fixation, Cross-Site Request Forgery, and Clickjacking. The root causes have been found in the use of browser-level authentication schemes and the missing user context on server-side.

Based on these insights, we have proposed a user-level authentication scheme, named Session Imagination. It makes use of images as session-based secrets that are shared between the user and the web application. We have shown its effectiveness in the sense that it mitigates the above mentioned vulnerabilities. The attacker's chances can be expressed as the probability to guess the correct session image. At the same time, this probability can be set by design to an arbitrary low value by providing a considerable number of images. The limit depends on the actual design of the user interface. We have shown its usability in a survey which confirms advantages in terms of user friendliness, universal applicability, cost, and security over the two state-of-the-art approaches. Session Imagination is applicable with reasonable efforts to new and existing web applications. It is technology-independent and does not create new requirements on the client-side.

In sum, we provide a solution that does not tamper with the symptoms of some vulnerability but resolves the underlying problem of web session-based deficiencies. In the course of this, we achieved the mitigation of at least four vulnerabilites that are exploited in practice.

Acknowledgments. We thank Daniel Schreckling for important advices on the prototype and fruitful discussions on open attack vectors. This work was in parts supported by the EU Project WebSand (FP7-256964), `https://www.websand.eu`. The support is gratefully acknowledged.

References

1. Luis Von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: using hard AI problems for security. In *EUROCRYPT'03*, pages 294–311, 2003.
2. Adam Barth, Collin Jackson, and John C. Mitchell. Robust Defenses for Cross-Site Request Forgery. In *CCS'09*, 2009.
3. Norm Hardy. The Confused Deputy: (or why capabilities might have been invented). *SIGOPS Oper. Syst. Rev.*, 22:36–38, October 1988.
4. Martin Johns. SessionSafe: Implementing XSS Immune Session Handling. In *ESORICS 2006*. Springer, September 2006.
5. Martin Johns, Bastian Braun, Michael Schrank, and Joachim Posegga. Reliable Protection Against Session Fixation Attacks. In *Proceedings of ACM SAC*, 2011.
6. Martin Johns and Justus Winter. RequestRodeo: Client Side Protection against Session Riding. In *OWASP Europe 2006*, May 2006.
7. Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Preventing cross site request forgery attacks. In *Proceedings of Securecomm 2006*, 2006.
8. Mitja Kolsek. Session Fixation Vulnerability in Web-based Applications. Whitepaper, Acros Security, `http://www.acrossecurity.com/papers/session_fixation.pdf`, December 2002.
9. Microsoft. X-Frame-Options. [online], `http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx`, (05/20/11).
10. Mozilla. X-Frame-Options response header. [online], `https://developer.mozilla.org/en/the_x-frame-options_response_header`, (05/20/11).
11. Mozilla. Csp (content security policy). Mozilla Developer Network, `https://developer.mozilla.org/en/Security/CSP`, March 2009.
12. MSDN. Mitigating Cross-site Scripting With HTTP-only Cookies. [online], `http://msdn.microsoft.com/en-us/library/ms533046(VS.85).aspx`, (06/08/12).
13. Marcus Niemietz. UI Redressing: Attacks and Countermeasures Revisited. In *in CONFidence 2011*, 2011.
14. Robert Hansen. Clickjacking. [online], `http://ha.ckers.org/blog/20080915/clickjacking/`, (05/20/11).
15. Robert Hansen and Jeremiah Grossman. Clickjacking. [online], `http://www.sectheory.com/clickjacking.htm`, (05/20/11).
16. Jesse Ruderman. The Same Origin Policy. [online], `https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript`, (06/08/12), August 2001.
17. Philippe De Ryck, Lieven Desmet, Thomas Heyman, Frank Piessens, and Wouter Joosen. CsFire: Transparent Client-Side Mitigation of Malicious Cross-Domain Requests. In *ESSoS '10*, pages 18–34, 2010.
18. Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites. In *Proceedings of W2SP 2010*, 2010.
19. Michael Schrank, Bastian Braun, Martin Johns, and Joachim Posegga. Session Fixation - the Forgotten Vulnerability? In *Proceedings of GI Sicherheit 2010*, 2010.
20. W3C. HTML5 - The canvas element. [online], `http://www.w3.org/TR/html5/the-canvas-element.html`, (09/24/11).
21. W3C. HTML5 - The iframe element. [online], `http://www.w3.org/TR/html5/the-iframe-element.html#the-iframe-element`, (08/29/11).
22. Yuchen Zhou and David Evans. Why Aren't HTTP-only Cookies More Widely Deployed? In *Proceedings of W2SP '10*, 2010.