Requirements for preventing logic flaws in the authentication procedure of web applications

Youssou Ndiaye, Olivier Barais, Arnaud Blouin, Ahmed Bouabdallah, Nicolas Aillery

Requirements for preventing logic flaws in the authentication procedure of web applications

Youssou Ndiaye
Orange labs Rennes, Univ Rennes, Inria, CNRS
Rennes, France
firstname.lastname@irisa.fr

Olivier Barais
Univ Rennes, Inria, CNRS
Rennes, France
firstname.lastname@irisa.fr

Arnaud Blouin
Univ Rennes, Inria, CNRS
Rennes, France
firstname.lastname@irisa.fr

Ahmed Bouabdallah
IMT Atlantique, IRISA, UBL
F-35576 Cesson Sévigné, France
firstname.lastname@imt-atlantique.fr

Nicolas Aillery
Orange labs Rennes
Rennes, France
firstname.lastname@orange.com

ABSTRACT

Ensuring the security is one of the most daunting challenges that web applications are facing nowadays. Authentication and authorization are two main security fields that web applications must consider to be protected against unauthorized accesses. Various approaches that detect well-known vulnerabilities and flaws exist. However, these approaches mainly focus on detecting input validation flaws. Another kind of flaws that affect web applications are logic flaws, but they lack of considerations.

This paper proposes an approach that helps to considering logic flaws in the context of web applications. The goal of the proposal is to strengthen the authentication procedure of web applications and thus enforce the security early in the design phase. We conducted an empirical study in nine well-known web-based applications to demonstrate that logic flaws may put at risk the authentication procedure. The results showed that logic flaws may be either caused by security issues or usability issues. To overcome such flaws, we provide ten relevant requirements that should be followed in the design of an authentication procedure.

KEYWORDS

Web-based authentication, human behavior, logic flaws, security requirements, usability requirements.

ACM Reference Format:
Youssou Ndiaye, Olivier Barais, Arnaud Blouin, Ahmed Bouabdallah, and Nicolas Aillery. 2019. Requirements for preventing logic flaws in the authentication procedure of web applications. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 9 pages.

1 INTRODUCTION

Authentication and authorization of web applications are continuously targeted by attackers to get unauthorized accesses. Efforts are made in detecting and finding input validation flaws [6, 11, 23]. In contrast, less attention is paid to logic flaws. The main difference between logic flaws and input validation flaws is their exploits. To exploit an input validation flaw the attackers mainly leverage on coding mistakes, such as a lack of input variables control, to then inject malicious code (*e.g.,* SQL injection [30], cross-site scripting [17]). A logic flaw exploit requires the attacker to find out a defect in the way the application makes decisions. We extended the OWASP Foundation's definition of a logic flaw [13] to the context of the authentication of the web-based applications, and provide the following definition:

> *Logic flaws are ways of using the legitimate authentication processing flow of a web-based application in a way that it results with negative impacts either on the users or the organization.*

Considering logic flaws during both the design and the maintenance of authentication procedures of web applications is challenging. For an existing web application, finding logic flaws are costly and cumbersome for developers since these flaws are inherently linked to the application business logic. Developers thus need high level characterization of the logic flaws developers are looking for. For a web application under development, developers need guidance during the design of the authentication procedures to prevent logic flaws.

The proposed approach described in this paper focuses on the design of the user's authentication procedure. The goal is to enforce the security of the design of the authentication procedure by considering human factors and design errors. This paper makes two complementary contributions. First, we conducted an empirical study that shows that logic flaws may put at risk the authentication procedure of a web application. The study highlights that human behaviors (*e.g.,* lost phone, security unawareness) and usability issues (*e.g.,* active session, auto-completed forms) are as important to consider as the security issues [10, 31]. Second, we provide a set of security and usability requirements designers can use during the early design of the end-user authentication procedure to prevent logic flaws. The proposed requirements can be used during different development steps. They can be used to assess the reliability of a web application as we did during the empirical study. For a web application under development, developers can use the requirements to consider and prevent logic flaws during the design of the authentication procedure.

The paper is organized as follows. Section 2 presents the motivating examples from two case-studies. Section 3 presents the empirical study that we conducted to highlight how logic flaws put at risk the authentication procedure of web applications. Section 4 describes

SAC '19, April 8–12, 2019, Limassol, Cyprus
Author Version. 2019.

the approach and the set of requirements we propose. Finally, Section 5 discusses the related works and Section 6 concludes the paper with future works and perspectives.

2 MOTIVATING EXAMPLES

2.1 Skype

Skype[1] is a well-known messaging application powered by Microsoft. In 2012, the blog *pixus-ru* published a combination of six expected design behaviors that led to an exploit. The flaws allowed changing the password of any Skype account by using the corresponding email address [24].

The design errors. The following design errors are identified.

#1 The application allowed a user to create an account by providing a valid email address. The application, however, did not verify that the given email address exists and belongs to the current user.

#2 The application allowed binding multiple accounts to the same email address.

The exploit. Because of the design error #1, an attacker could create a new Skype account based on the email address of an existing user to then initiate a password reset. Then, because the account's binding described in the design error #2, the application asked the attacker for the name of the Skype account to reset. Finally, the attacker could change the initial user's password and grabbed full access to the targeted Skype account.

The design errors described above are, in fact, expected behaviors of the application. The attacker only leverages on the application business logic to impersonate a legitimate user (*e.g.,* changing the password).

2.2 Mobile Connect

Mobile Connect is a digital authentication solution powered by the Group Special Mobile Association (GSMA) [16]. The standard is implemented by mobile operators in order to stand as Identity Provider (IdP) for Service Provider (SP), banking, and governmental services.

The registration phase requires a valid mobile phone number provided by the mobile operator. This registration phase is initiated by the user who provides the phone number to the *Mobile Connect* web site. Afterward, an identity verification is done by the mobile operator and a confirmation code is sent to the user's mobile phone to validate his/her registration and create a 4-digit pin code.

The authentication phase requires two steps. 1) The user enters his/her phone number to the requested application. 2) The user receives a push up notification to his/her mobile phone and validates the authentication by entering the pin code.

The recovery/reset of the pin code is initiated on the *Mobile Connect* web application. 1) The user provides his/her phone number. 2) A push up message with an OTP (One Time Password) [18] is sent to the user's mobile phone. 3) The user fills the OTP on the mobile connect web application, resolves a Human Interaction Proofs (HIPs) challenge [5], and creates a new pin code.

The design errors and exploits. We identified two main design errors.

[1]https://www.skype.com

#1. Anyone can request for an authentication by giving a valid phone number to the *Mobile Connect* web-based service. So, a legitimate entity can inadvertently validate a remote authentication request.

#2. The authentication process is weakened by the recovery process. To reset a pin code, the *Mobile Connect* web-based service requests for a valid phone number and then sends a push-up notification to the same user's mobile phone to create a new pin code. If a malicious user gets access to a legitimate user's phone (*e.g.,* lost or stolen phone), this malicious user can request for a pin code reset and create a new one since he no additional security challenge (*e.g.,* answering a secret question before changing the new pin code).

Human behavior (*e.g.,* lost or stolen phone) is the root cause of this exploit. This, however, may be limited if the specification has provided additional security mechanisms such as a secret question known by the legitimate user.

2.3 Conclusion

The two aforementioned exploits are related to logic flaws. In *Skype* the flaw is due to the weakness of the recovery phase combined with an *insufficient identity verification*. In *Mobile Connect* the exploit is linked to a *weak recovery process* eased by a user-substitution. They both target the legitimate user, have impacts to the entity's authentication procedure, and have caused irreversible damages (*i.e.,* Impersonation and Identity spoofing). The approach we propose provides requirements to help in overcoming logic flaws in the user authentication procedures of web applications.

3 EMPIRICAL STUDY

This experiment is motivated by the following research question:

Do logic flaws put at risk the authentication procedure of web applications?

A positive answer of this question justifies the necessity to take into account logic flaws in the design of the authentication procedure, which is the main goal of the proposed approach. To do so, we analyzed nine well-known applications to find logic flaws according to a given set of classes of logic flaws (Table 1).

Finding logic flaws is a complex task as such flaws are intertwined in business logic of the web application under study. The Table 1 summarizes classes of logic flaws that we identified from the literature [3, 7, 8, 13] and extended to the user's authentication context. In this section, we report an empirical study in which we use these logic flaws to assess the selected applications. We first introduce the subjects of the experiment: nine major web applications.

Amazon (http://www.amazon.fr) and *Showroom Privée* (https://www.showroomprivee.fr) are widely-used e-commerce websites. They manage user identities and credit card storages.

Blablacar (https://www.blablacar.fr) and *Leboncoin* (https://www.leboncoin.fr) are respectively carpooling and sales platforms between users. They also manage user identities and credit card storages.

Ameli (http://www.ameli.fr) is the French social security official website. *Yahoo* is a well-known email provider. *Paylib* (http://www.paylib.fr) and *Mobile Connect* (https://mobileconnect.orange.fr/selfcare/#/welcome) are authentication protocols respectively used

Class of logic flaws	Description
Registration	
Improper Identity management	The application failed to properly manage credential for a given identity.
Insufficient identity verification	The application does not sufficiently verify and ensure that the given identity information "objectively" exists and pertains to the claimant.
Recovery / Reset deadlock	The way the registration is powered does not provide sufficient information to lately permit recovery or reset credentials.
Weak path (security level)	The registration phase contains multiple paths with different security level.
Authentication	
Improper control of interaction frequency	The authentication mechanism does not support or improperly limits the number of attempts to get authenticated.
Insufficient entity authentication and verification	The authentication mechanism does not guaranty the entity's authenticity and non-repudiation.
Weak path (security level)	The authentication mechanism supports multiple paths with different security level.
Recovery/Reset	
Weak recovery process	The registration mechanism suffers from a weak credential recovery.
Insufficient entity authentication	The application does not sufficiently authenticate the current entity while recovering or reseting a credential.

Table 1: Classes of web authentication logic flaws

in electronic payments and as identity providers for French mobile operators.

3.1 Experimental protocol

To run the experiment, we created temporary *Yahoo* and *Facebook* identities to get registered in the corresponding applications. For each application, we manually assess the three phases of the authentication procedure (*cf.* Section 4.1) as following:

1. We manually use the nominal behavior of the application to find logic flaws during each phase.
2. We classify the discovered flaws according to the provided classes of logic flaws.
3. We exploit the flaws. Then, we characterize the exploit according to the entries points, the targets and the consequences (*cf.* Section 4.6. And, we finally select the requirements that are relevant to tackle the flaws.

The mark "✓" means that the corresponding class of flaws is identified in the application. The absence of "mark" states that, we do not identified the corresponding class of logic flaw. The results of the experiment is presented in Section 3.2 and the outcomes are detailed in Section 3.3.

3.2 Results

The results of the experiment are grouped per authentication's phases. For each phase, we give a brief description of the identified flaws and describe the way the flaws are exploited in the corresponding applications.

3.2.1 Registration phase.

Improper Identity Management. The *Improper Identity Management* class of flaws gathers all the logic flaws related to the legitimate user identity management, such as issuing a credential to the wrong identity. This is directly related to the registration phase. The applications *Blablacar* and *Showroom Privee* support both a separate IdP and an email-based registration phase. If an entity uses the IdP to get registered and then creates another account with the same email address, this entity is no longer able to get authenticated unless it performs a credential recovery. In fact, two accounts are bounded to the same identity but are not managed sufficiently. The *Skype* application also suffers from that flaw (*cf.* Section 2.1).

Insufficient identity verification. An insufficient identity verification flaw occurs when the design does not sufficiently provide a mechanism to verify the identity of an entity. In the context of the web applications, the registration phase should ensure that the asserted identity (*e.g.,* email address) objectively exists and belongs to the claimant (*i.e.,* entity that claims the assertion). For instance, the applications *Skype*, *Blablacar*, and *Showroom Privée*) are affected by this flaw as they register the entities without a "sufficient" email verification (i.e., make sure that the email undoubtedly belongs to the subscriber) .

Recovery / Reset deadlock. A recovery deadlock flaws occurs when the registration phase does not provide sufficient information to allow the legitimate entity to perform a future credential reset or recovery. This usually happens when the application uses a separate IdP that provides a valid token to register an entity [19, 29]. The reset process will fail if it is based on the current entity's password as the application no longer has access to the entity's original password.

This has a usability issue in *Showroom Privée* and *Blablacar* by preventing legitimate entities from resetting their own password.

3.2.2 Authentication phase.

Improper control of interaction frequency. Controlling the frequency of user interactions in the authentication process avoids

Applications / Logic flaws	Skype	Blablacar	Ameli	Showroom Privée	Leboncoin	Amazon	Yahoo	Paylib	Mobile Connect
Registration phase									
Improper Identity management	✓	✓		✓					
Insufficient identity verification	✓	✓		✓					
Recovery, Reset deadlock		✓		✓					
Authentication phase									
Improper control of interaction frequency		✓	✓	✓	✓		✓		✓
Insufficient entity authentication		✓		✓	✓	✓	✓	✓	
Weak path (security level)		✓		✓	✓	✓	✓		✓
Recovery/Reset									
Insufficient entity authentication	✓	✓		✓	✓		✓		✓
Weak recovery, reset process	✓	✓		✓	✓		✓		✓

Table 2: The results of nine assessed applications

attacks such as brute force ones (*e.g.,* Captcha, limited number of attempts). Its misuse however, may lead to logic flaws. Six of the nine applications are vulnerable to an *Improper Control of Interaction Frequency*. For example with *Ameli*, a malicious entity may deny the access for fifteen minutes to a legitimate entity after three wrong attempts in the login form. This may be automated and carefully performed to target specific entities based on these gender, age, or localities (this is specific to Ameli because of the social welfare number format).

The other defected applications (*e.g., Blablacar*, *Leboncoin*) did not fix a limited number of attempts in their login form and are thus vulnerable to brute force attacks [2].

Insufficient entity authentication. Web applications support strong authentication processes to protect legitimate entities against unauthorized access. Nevertheless, human behavior are often unconsidered [1, 31]. For instance, six of the nine tested applications did not provide means to sufficiently authenticate the active entity: they assume that the current authenticated entity is able to perform sensitive actions such as password reset. So, an attacker, who gets access to a device where an authenticated session is still active, can easily impersonate the legitimate entity. For example, in *Blablacar*, *Leboncoin*, and *Mobile Connect*, the password reset and the update of the personal identity are performed without any re-authentication of the current user. This does not limit the damage that may cause a user substitution.

Weak path (security level). The class of weak path flaws gathers any logic process that intentionally supports a path that weakens the security level of the expected path. A path is a pre-defined set of actions that achieves a goal. Path equivalence is relevant when multiple paths lead to the same goal (*e.g.,* multiple authentication mechanisms). So that, if a minimal security level is fixed by the design, all other paths should at least achieve that minimal security level. For instance, in *Mobile Connect*, an attacker who has access to an entity’s mobile phone (*i.e.,* stolen or hacked), may request for a pin code reset, and afterward, get authenticated in lieu of the legitimate user (*cf.* Section 2.2). The Blablacar and Leboncoin applications are vulnerable to that flaw because of the recovery process that sends a reset link to entities email address without any additional security mechanism.

3.2.3 Recovery, Reset phase.

Weak recovery, reset process. A weak recovery process is a recovery that did not sufficiently authenticate the requested entity (*e.g.,* suffering from an *insufficient identity verification* flaw) or that supports a security mechanism weaker than the authentication one (*i.e.,* existence of a *a weak path*). For instance, six of the nine tested applications are vulnerable to that class of flaws by sending back a reset link by email without any additional security verification (*e.g.,* secret question, One Time Password).

Insufficient entity authentication. Web applications tend to consider that the current authenticated entity is the legitimate entity and allow her to perform sensitive actions (*e.g.,* a reset password). We consider this design as a logic flaw since a malicious entity that gains access to a service (*e.g.,* a mobile phone stealing) may reset the credential of the legitimate entity in sensitive services such as mailing and banking services. An *insufficient entity authentication* during a password reset may be the origin of a *weak recovery process*. Six of the nine assessed applications suffer from a *weak recovery process* by allowing the current connected entity to reset the password without any additional security mechanism to ensure and re-enforce the entity’s authenticity.

3.3 Outcomes

The outcomes of the experiment highlight three key points:

1. The experiment demonstrates that the above classes of logic flaws can put at risk the authentication procedure (*e.g.,* weak path, insufficient entity authentication, weak recover/reset).
2. The experiment shows that the authentication design should take into account the human habits to improve the security level: for instance, six of the nine applications are vulnerable to *an insufficient entity authentication* during the reset phase.

3. The experiment highlights that the classes of logic flaws can be combined to put at risk the corresponding application (*cf.* Sections 2.1 and 2.2).

This motivates the need to consider logic flaws in the design of the authentication procedure. Thus, we provide the approach described in the next section to help to overcome such flaws.

4 APPROACH

Designing a convenient authentication method requires subsequent security backgrounds. The web application designers rely on secure and reliable authentication methods [19, 29]. However, these authentication procedures mainly deal with identified security issues such as input validation vulnerabilities (*i.e.,* SQL Injection [30], XSS [17]). They also lack at considering logic flaws. Since exploits of logic flaws can permit a malicious entity to circumvent the legitimate user's authentication procedure. The proposed approach gives a convenient definition of the web application user authentication procedure. The approach then provides a set of usability and security requirements to prevent logic flaws in the early design phase.

4.1 Background

We adapted the NIST electronic authentication guideline [4] and the ITU Entity authentication framework [21] to the context of web applications and define the user authentication procedure as a composition of three phases: the registration phase, the authentication phase, and the recovery/reset phase.

4.1.1 Registration phase. The registration phase is the process where an entity requests for a specific credential. A credential is a set of data presented as evidence of a claimed identity and/or entitlements (*e.g.,* login and password). It can be issued by fulfilling an application form (*e.g.,* self-claimed identity in web applications). A trusted third-party application such as CSP (*i.e., Credential Service Provider*) or an IdP can request for an entity registration. In both cases, an identity verification should be performed to prove the identity of the entity. The entity represents the claimant. It can be a human or a device. The term *entity* is used in this the paper to both refer to humans or devices.

4.1.2 Authentication phase. The authentication phase consists of an entity providing its credentials to attest its identity to a relying party. A relying party represents an actor (*e.g.,* a web application) that relies on an identity assertion (*e.g.,* token, login/password). We assume that the authentication protocol is well implemented and reliable. The proposed approach focuses on logic flaws and threats related to the claimant's behavior and the design errors of the business process of the authentication procedure.

4.1.3 Recovery, Reset phase. Recovery phase stands for an entity requesting for retrieving a lost or revoked credential. It is generally followed by a credential reset for sake of security issues. Independently, a credential reset can be performed at any time an entity is authenticated. We are concerned about design errors and human habits that may lead to logic flaws. So, technical vulnerabilities or flaws related to the authentication protocols, methods, and credential storage are not taken into account in this work. From the above definition, we provide the set of requirements grouped by phases as highlighted in Table 3.

Tags	Requirements
Registration phase	
#1	Credential uniqueness assurance
#2	Identity verification assurance
#3	Credential recovery/reset providing assurance
#4a	Path equivalence assurance
Authentication phase	
#5	Interaction frequency limit assurance
#4b	Path equivalence assurance
#6a	Entity authentication assurance
#7	Active session limitation assurance
Recovery/Reset phase	
#6b	Entity authentication assurance
#8	Deadlock avoidance assurance

Table 3: The ten requirements grouped by phases

4.2 Registration design requirements

Four of the ten requirements we provide focus on the registration phase. Three of them are related to security issues (*i.e.,* requirements #1, #2, and #4a) and one is related to usability issue (*i.e.,* requirement #3).

4.2.1 Credential uniqueness assurance (#1). The credential uniqueness assurance is not mandatory to the registration phase but highly recommended. It aims at easing the identity management by issuing one credential per identity. Otherwise, in case the registration phase, by design, allows binding multiple credentials to the same identity, it shall provide a mechanism to correctly manage the credential issuance.

4.2.2 Identity verification assurance (#2). Web applications mostly rely on asserted identity from a claimant. Ensuring the authenticity of the given identity is vital for identity-based service (*i.e.,* social network, e-commerce, banking). The registration phase shall ensure that the provided identity "objectively" exists and is verified correctly. This means that the entity that is performing the registration is the same entity that the given identity is entitled. For instance, if the registration requires an email, the process shall verify that the provided email exists and belongs to the entity that has provided the email.

4.2.3 Credential recovery/reset providing assurance (#3). Web applications commonly use identity federations with a separate IdP to ease the enrollment of a user. Afterward, some rely on the given information from the IdP and link the relevant data to a local entity's account, while others do not handle local entity's account. This usability requirement comes to enforce the procedure by allowing the entity to give sufficient information for future credential reset and recovery (*e.g.,* recovery by e-mail, secret question). The underlying authentication protocols (*e.g.,* Oauth 2.0 [19], OpenID [29]) do not transmit the entity's original password to the relying party application. So, the recovery or reset credential will fail if it is based on the entity's password.

4.2.4 Path equivalence assurance (#4a). Path equivalence means that every path that leads to a registration should at least match the minimum required security level. In case the registration is delegated to a separate IdP, the value of the asserted security level (*i.e.,* Token) from the IdP shall at least reach the same level that the required security level. For example, if the system requires a level 2 (*e.g.,* NIST Recommendation-based [4]) for its identity proofing, the IdP should be at least level 2 or higher. Otherwise, the registration is weakened by the path that supports a lower security level.

4.3 Authentication design requirements

The four following requirements focus on security and usability issues.

4.3.1 Interaction frequency limit assurance (#5). Limiting the number of attempts in a login form is one of the most security design used against brute forcing attacks. A malicious entity, however, may abuse that functionality to perform attacks that may affect the user or the organization. This requirement comes to ensure that the authentication process supports a mechanism that "properly" limits the number or frequency that an entity has to claim its identity. This means that such a limitation should neither lead to security issues (*e.g.,* brute force attacks [2]) nor to usability issues (*e.g.,* deny of service of the legitimate user).

4.3.2 Path equivalence assurance (#4b). Web applications may use different means to authenticate an entity (*e.g.,* multi-factor authentication, biometrics). Path equivalence means that the security of the authentication process is the minimum security level of the provided means [34]. So that, designers shall fix the required security level and ensure that all other existing paths support at least that minimum security level. Otherwise, a weaker authentication path may exist. It is important to notice that the *Path Equivalence Assurance* involves the recovery/reset phase. For instance, a recovery phase may push down the authentication phase security level when the identity verification is not well performed (*cf.* requirement #2 in Section 4.2.2).

4.3.3 Entity authentication assurance (#6a). This requirement gathers all processes that support a mechanism that does not sufficiently authenticate the legitimate entity (*e.g.,* session hijacking, dead URL, cookie stealing). This is also relevant for strong authentication. Remote devices such as a mobile phone can be used as second authentication factor and allows the validation of the authentication procedure remotely. This requirement ensures that the authentication process supports a mechanism that guarantees the authenticity and non-repudiation of the entity during a process that requires an authentication. For instance, when an authentication can be initiated in a web application and remotely validated with a mobile application, the non-repudiation of the entity is engaged (*e.g.,* the mobile is something it has). The process, however, does not guaranty that the one that initiated the transaction is the one that validated it (*e.g.,* the authenticity of the entity is not guaranteed).

4.3.4 Active session limitation assurance (#7). To ease the entity's authentication, web applications leverage on cookies to keep the legitimate entity's session active. In an ubiquitous environment, this may be considered as a logic flaw. In fact portable devices (*e.g.,* laptop, mobile phone) are likely to be lost or stolen. So, they expose the legitimate entity to a possible logic exploit. This requirement enforces the authentication phase by means to limit the lifetime of an active session. For instance, a long active session combining with a misuse of an *entity authentication* (*e.g.,* cookie stealing) enhances the range of possible exploits that attackers can leverage [3].

4.4 Recovery/Reset design requirements

This design phase gathers two requirements. One for security issues that aims at enforcing the legitimate entity's authenticity, and one related to usability issues.

4.4.1 Entity authentication assurance (#6b). This requirement comes to enforce the requirement #6a in Section 4.3.3. Although, the legitimate entity is well authenticated during the authentication phase, sensitive actions such as identity modification shall require an instant authentication. This requirement ensures that, to perform sensitive action such as password reset, the current entity should be instantly re-authenticated to rise the security level and ensure the current entity's authenticity and non-repudiation.

4.4.2 Deadlock avoidance assurance (#8). This requirement is complementary to the requirement presented in Section 4.2.3. It comes to ensure that no blocking states are supported by the reset or the recovery phase. In other word, this means that the legitimate user should be always able to update or recover its own credential.

4.5 Discussion

The ten aforementioned requirements allow designers to avoid and/or anticipate logic flaws that may circumvent the authentication procedure. We notice that not all the requirements are related to security issues: usability issues are also taken into account (*e.g.,* requirement #1, #3 and #8 (Sections 4.2.1, 4.2.3, and 4.4.2). Also, several requirements target the same issues, *i.e.,* the requirements #4a and #4b (Sections 4.2.4 and 4.3.2), and the requirements #6a and #6b (Sections 4.3.3 and 4.4.1). These requirements, however, are applied in different phases. For instance, the *Entity authentication assurance* in the authentication phase aims at ensuring that the claimant is well authenticated and non-repudiation is guaranteed, while in the reset phase, it aims at providing an additional security mechanism to confirm the current entity's authenticity.

The provided set of requirements and the class of logic flaws are not in a direct one-to-one relationship. While some requirements directly target specific class of logic flaws, *e.g.,* requirement #5 (*cf.* Section 4.3.1) for the *Improper control of Interaction Frequency* class of logic flaws, other classes of logic flaws may involve more than one requirement to be avoided. For example *a weak recovery process* is exploited by combining an *Insufficient entity authentication* with a *weak path* (*cf.* Section 2.1, so involved several requirements (*cf.* Section 2.2).

4.6 Using the proposed approach

Because eliciting security requirements to a given application requires security background. The proposed approach provides a set of requirements that designers can follow to prevent future logic flaws that may affect their applications. In addition in considering the classes of logic flaws summarized in Table 1, the designers with security expert's helps, may figure out the related logic flaw threats according to their context and apply the relevant requirements [27, 32, 33].

To a running application, the proposed approach can be used to assess the authentication procedure and improve its reliability. To do so, we briefly adapt the Jiwnani *et al.*'s taxonomy [22] to characterize the logic flaw threat model as following:

- **Entries points.** During which authentication phases the exploit is used?
- **The targets.** Does the exploit target the organization and/or the user?
- **The consequences.** What are the effects of the exploit? Usability and/or security?

From the above characterization of the exploit, a tester can select the corresponding requirements to mitigate it.

Use case of a running application. We use the exploit described in the motivating example (see Section 2.1) to show how the approach can be used in a running application that contains logic flaws.
First, the exploit can be characterized as follows:

- The entries points are the registration and the reset phases. The attacker creates an account with a false identity and then requests for a reset credential (the email address is the key identity attribute).
- The target is the legitimate user account. The attacker changes the legitimate user password ((i.e., the asset).
- The consequences are mainly security issues. The attacker impersonates the legitimate user and steals his/her digital identity. We can also consider a minor usability consequence which is a deny of service of the legitimate user (*e.g.,* the user has no longer access to his/her Skype account).

According to this description, the following requirements should be relevant to provide means to prevent the exploit.

- In the registration phase.

  The registration phase should provide means to limit to one, the number of accounts per user or ensure that conflicting accounts are well managed (*i.e.,* requirement #1 in Section 4.2.1)

  The registration phase should support a mechanism that ensure that the claimant's identity (i.e. the email) exists and is well verified too (sending a confirmation link by email : *i.e.,* requirement #2 in Section 4.2.2).
- In the reset phase.

  Before allowing the current user to reset the password, the Skype application should implement a mechanism to re-authenticate the user (i.e., asking a secret question for example : requirement #6b in Section 4.4.1).

4.7 Threats to validity

The experiment we conducted shows that logic flaws may have negative impacts in the authentication procedure of web applications. In Section 4.6 we described how the proposed approach can be applied in a running application and/or in an application under development. Table 4 highlights the effectiveness of the approach and shows how the affected applications could be improved with the set of the provided requirements. To each affected application, we show the relevant requirements that could help to improve the security and/or the usability.

Logic flaws are dictated by the application's business logic. Consequently, they differ from one context to another. So, while this approach endeavors to avoid them, it do not ensure their absence from a context to another one.

We do not claim the exhaustiveness of the proposed approach. Both the set of requirements and the classes of logic flaws can be extended. The goal is to show, from a given context, that taking into account logic flaws may help to strengthen the authentication procedure. Also, we do not provide means to implementing the requirements, as we are providing information security requirements, it is to the security mechanism to do so.

5 RELATED WORK

5.1 Input Validation flaws

Many efforts have been made to detect vulnerabilities in web applications. However, most of them focus on detecting and testing well-known vulnerabilities related to input validation such as Cross-site Scripting [20, 23] and SQL Injection [30]. The work described in this paper is complementary to their research works by focusing on logic flaws.

5.2 Logic flaws taxonomies

To the best of our knowledge, the OWASP foundation provides the largest classification of existing vulnerabilities and flaws in the context of application security [14]. For each identified vulnerability or flaw, it gives relevant countermeasures. Nevertheless, because of the difficulty to be categorized, minor attentions are paid to logic flaws. While OWASP is general to any kind of applications, Common Weakness Enumeration [7] and the Web Application Security Consortium [35] are respectively more related to software engineering and web application security issues. Efforts are also made to warn security designer about logic flaws. Grossman *et al.* [15] present seven logical flaws that may put an application at risks. They highlight the WASC 24 classes of vulnerabilities [35] where logic flaws are categorized into three categories, *e.g.,* authentication, authorization, logic attacks. While this paper is deeply inspired by the above works, we do not limit the proposal to demonstrate the need to pay more attention to logic flaws. We go straightforward into the logic flaws issues and provide tangible means to tackle them.

5.3 Logic flaws in testing web-based applications

Pelegrino *et al.* [28] provide a black-box approach to detect logic flaws based on the legitimate user's interactions with the corresponding application. Cova *et al.* [6] provide Swaddler that aims at detecting logic flaws that may lead the applications to what they call workflow violation attacks. The approach leveraged on the application expected behavior and then monitors state variables at runtime looking for deviations from the normal behavior. Block [25] is similar to Swaddler, but focuses on authentication bypass. Di Lucca *et al.* [9] consider the browser interactions and provide a state-based testing approach to detect inconsistencies in web applications. As opposite to all these works, our approach does not automatically detect existing logic flaws. Instead, we focus on avoiding design flaws that may lead to logic flaws in the early design phase. Although, the

Applications	Requirements	Descriptions
Skype	#1, #2 and #6b	The application could limit to one the number of account per user and verify that the given email address exists and belongs to the claimant. During the password reset, the process could ask for additional security challenges such as a secret question before sending back the reset link.
Blablacar	All requirements	For the federated identity, the application could support a local account per user to better handle identity and allows future identity credential. Then it could properly limit the user attempts and ensure the claimant identity for the email-based registration. At least, additional security mechanisms such as a shared secret question could be supported to improve the recovery and reset credentials.
Ameli	#5	The application could properly control the interaction frequency by performing a challenge response or a HIP challenge instead of automatically denying access to the legitimate user.
Showroom Privée	All requirements	The same as in Blablacar since they support the same authentication mechanism.
Leboncoin	#5, #6b, and #7	The application could properly limit the number of user attempts and fix a limited time of active session. Also, to reset the password the user could be re-authenticated.
Amazon	#7, and #6b	The application could fix a limited active session. Also, it could re-authenticate the current user for sensitive actions such as identity update, as it does for the password reset.
Yahoo	#5, #6b, and #7	The same as in Leboncoin.
Paylib	#6a	Instead of remotely validating a transaction that may pay confusion to whom initiated it, the validation may be confirmed for example by the legitimate user in the merchant platform with a shared secret. This re-enforce the non-repudiation property.
Mobile Connect	#5,#6a, and #6b	Identical to Paylib, plus properly limiting the number of user's attempts to provide a pin code. The recovery may implement additional security challenges to overcome the user-substitution issues.

Table 4: How the assessed applications could be improved with the proposed approach

above works can be jointly used with our proposal. The detected exploits using these tools can be characterized with our taxonomy and then leveraged on the set of requirements to implement the relevant countermeasures.

5.4 Usability and security requirements

The Human-Computer Interaction community provides a large corpus of works to reduce the gap between usability and security. They consider human as the weakest link that should be taken into account while building better security policies [1, 31, 36]. To take into account the human habits in ubiquitous environments, Mazheliss *et al.* [26] provide a framework for detecting user-substitution in the context of mobile applications. These works can be jointly used with our approach to protect legitimate users from attacks resulting to user-substitution. The usability requirements that we provide inspire thinking and warn designers about usability sake that may involve logic flaws.

Starting with Sindre *et al.* approach [32], several works are dealt on eliciting security requirement [12, 27, 33]. While these works focusing on how to elicit relevant requirements to a given context, our approach leveraged on them, and provides predefined requirements based on existing threats (i.e., logic flaws) to a given context (i.e., web authentication procedure).

6 CONCLUSION AND FUTURE WORKS

This paper introduces a novel approach to enforce the security of the authentication procedure in the early design phase. The approach focuses on design errors and human factors that lead to logic flaws. We introduce a definition of the authentication procedure that is relevant in the context of web applications.

The approach defines ten security and usability requirements grouped by phases. The requirements are designed to take into account both design errors and human factors. We provide a set of logic flaws classes and conduct an empirical study to demonstrate that logic flaws may put at risk the authentication procedure of web applications. The approach can be used jointly with existing recommendations such as the NIST Authentication guidelines [4], the entity authentication assurance framework [21] and/or the security requirement approaches [12, 27, 33].

In our future work, we will provide a formal description of the web authentication procedure based on the three aforementioned phases. This will avoid ambiguities in the interpretation of the specifications, and will additionally allow designers to systematically verify the fulfillments of the requirements to an authentication procedure.

REFERENCES

[1] Anne Adams and Martina Angela Sasse. 1999. Users are not the enemy. *Commun. ACM* 42, 12 (1999), 40–46.

[2] Carlisle Adams, Guy-Vincent Jourdan, Jean-Pierre Levac, and François Prevost. 2010. Lightweight protection against brute force login attacks on web applications. In *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*. IEEE, 181–188.

[3] Iván Arce, Kathleen Clark-Fisher, Neil Daswani, Jim DelGrosso, Danny Dhillon, Christoph Kern, Tadayoshi Kohno, Carl Landwehr, Gary McGraw, Brook Schoenfield, et al. 2014. Avoiding the top 10 software security design flaws. *Technical report, IEEE Computer Societys Center for Secure Design (CSD)* (2014).

[4] William E Burr, Donna F Dodson, William T Polk, et al. 2004. *Electronic authentication guideline*. Citeseer.

[5] Kumar Chellapilla, Kevin Larson, Patrice Simard, and Mary Czerwinski. 2005. Designing human friendly human interaction proofs (HIPs). In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 711–720.

[6] Marco Cova, Davide Balzarotti, Viktoria Felmetsger, and Giovanni Vigna. 2007. Swaddler: An approach for the anomaly-based detection of state violations in web applications. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 63–86.

[7] Common Weakness Enumeration (CWE). [n. d.]. A community-developed List of Software Weakness Types. https://cwe.mitre.org/index.html

[8] Rachna Dhamija and Lisa Dusseault. 2008. The seven flaws of identity management: Usability and security challenges. *IEEE Security & Privacy* 6, 2 (2008).

[9] Giuseppe A Di Lucca and Massimiliano Di Penta. 2003. Considering browser interaction in web application testing. In *Web Site Evolution, 2003. Theme: Architecture. Proceedings. Fifth IEEE International Workshop on*. IEEE, 74–81.

[10] P.S. Dowland D.Katsabas, S.M. Furnell. [n. d.]. HCI principles to promote usable security. ([n. d.]).

[11] Sebastian Elbaum, Gregg Rothermel, Srikanth Karre, and Marc Fisher II. 2005. Leveraging user-session data to support web application testing. *IEEE Transactions on Software Engineering* 31, 3 (2005), 187–202.

[12] M. Jose Escalona and Nora Koch. 2004. Requirements engineering for web applications-a comparative study. *J. Web Eng.* 2, 3 (2004), 193–212.

[13] OWASP Foundation. [n. d.]. Business Logic Security Cheat Sheet. https://www.owasp.org/index.php/Business_Logic_Security_Cheat_Sheet

[14] OWASP Foundation. [n. d.]. Vulnerability. https://www.owasp.org/index.php/Category:Vulnerability

[15] Jeremiah Grossman. 2007. Seven business logic flaws that put your website at risk. *WhiteHat Security, October* (2007).

[16] GSMA. [n. d.]. Introducing mobile connect - the new standard in digital authentication. https://www.gsma.com/identity/mobile-connect

[17] Shashank Gupta and Brij Bhooshan Gupta. 2017. Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management* 8, 1 (2017), 512–530.

[18] Neil Haller. 1995. The S/KEY one-time password system. (1995).

[19] Dick Hardt. 2012. *The OAuth 2.0 authorization framework*. Technical Report.

[20] Vinay M Igure and Ronald D Williams. 2008. Taxonomies of attacks and vulnerabilities in computer systems. *IEEE Communications Surveys & Tutorials* 10, 1 (2008).

[21] International Communication Union (ITU). 2012-09-07. Entity Authentication assurance framework. http://handle.itu.int/11.1002/1000/11608

[22] Kanta Jiwnani and Marvin Zelkowitz. 2004. Susceptibility matrix: A new aid to software auditing. *IEEE Security & Privacy* 2, 2 (2004), 16–21.

[23] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. 2006. *Pixy: A static analysis tool for detecting web application vulnerabilities (short paper)*. IEEE.

[24] Ivan Koldaev. [n. d.]. Hack any skype account in 6 easy steps. http://pixus-ru.blogspot.com/2012/11/hack-any-skype-account-in-6-easy-steps.html

[25] Xiaowei Li and Yuan Xue. 2011. BLOCK: a black-box approach for detection of state violation attacks towards web applications. In *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 247–256.

[26] Oleksiy Mazhelis and Seppo Puuronen. 2007. A framework for behavior-based detection of user substitution in a mobile context. *computers & security* 26, 2 (2007), 154–176.

[27] JD Meier. 2006. Web application security engineering. *IEEE Security & Privacy* 4, 4 (2006), 16–24.

[28] Giancarlo Pellegrino and Davide Balzarotti. 2014. Toward Black-Box Detection of Logic Flaws in Web Applications.. In *NDSS*.

[29] David Recordon and Drummond Reed. 2006. OpenID 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*. ACM, 11–16.

[30] Amirmohammad Sadeghian, Mazdak Zamani, and Azizah Abd Manaf. 2013. A taxonomy of SQL injection detection and prevention techniques. In *Informatics and Creative Multimedia (ICICM), 2013 International Conference on*. IEEE, 53–56.

[31] Martina Angela Sasse, Sacha Brostoff, and Dirk Weirich. 2001. Transforming the 'weakest link' a human/computer interaction approach to usable and effective security. *BT technology journal* 19, 3 (2001), 122–131.

[32] Guttorm Sindre and Andreas L Opdahl. 2005. Eliciting security requirements with misuse cases. *Requirements engineering* 10, 1 (2005), 34–44.

[33] Inger Anne Tondel, Martin Gilje Jaatun, and Per Hakon Meland. 2008. Security requirements for the rest of us: A survey. *IEEE software* 25, 1 (2008).

[34] Anna Vapen and Nahid Shahmehri. 2010. Security Levels for Web Authentication Using Mobile Phones. In *Privacy and Identity Management for Life - 6th IFIP WG 9.2, 9.6/11.7, 11.4, 11.6/PrimeLife International Summer School, Helsingborg, Sweden, August 2-6, 2010, Revised Selected Papers*. 130–143. https://doi.org/10.1007/978-3-642-20769-3_11

[35] Web Application Security Consortium (WASC). [n. d.]. Threat Classification. http://projects.webappsec.org/w/page/13246978/Threat%20Classification

[36] Ka-Ping Yee. 2002. User interaction design for secure systems. In *International Conference on Information and Communications Security*. Springer, 278–290.