This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE CCNC 2010 proceedings

R2M: a Reputation Model for Mashups

Vincent Toubiana, Vincent Verdot, Gerard Burnside and Eric Joubert
Alcatel-Lucent Bell Labs
Application Domain, Hybrid Communications dpt.
Centre de Villarceaux, Route de Villejust
91620 Nozay, France
Email: *firstname.lastname*@alcatel-lucent.com
Telephone: (+33) 1 3077 2784

***Abstract*—The Web 2.0 has changed the Internet landscape, users are no longer only consumers but now also producers of content. The increasing number of personal data published on Web Service Providers fathered a new kind of applications: the mashups. These third-party applications access users' information through service providers' APIs via secure authorization protocols such as OAuth. But these protocols rely on the users who must blindly grant access to each mashup, with no idea beforehand about its trustworthiness.**

We propose a Reputation Model for Mashups to address this issue. The R2M solution monitors mashups' calls on the Web Service Providers' APIs, detects suspicious activities, and finally reports to the user to collect his feedback in order to collaboratively build the mashup's reputation. We describe an implementation of R2M on the Bell Labs' service *Dundal.com* to prove its feasibility in a real use case. From this experimentation, we plan to collect user experience to improve the R2M key mechanisms and refine the reputation computation.

***Index Terms*—Reputation, trust mechanisms, API authorization, OAuth, mashups.**

I. Introduction

Web 2.0's user-friendly principles: ease of use, creation and sharing, allow any user to actively participate in the development of Internet. Indeed, nowadays users are no longer only consumers as they used to be but also producers of content, interacting with each other through Web Service Providers (WSP). This new model enables unlimited possibilities, but raises along some new issues...

Photos, videos, applications, blogs, personal information, contact addresses, etc., anything can be published on the Internet with just a few mouse clicks. The success of Web 2.0 tightly relies on this richness of content provided by end-users as long as they trust this model, but for how much longer? Users deliver a lot of sensitive data to various Web Service Providers but they also really care about their privacy. Understanding this 2.0's paradox is crucial; basically users agree on exposing personal data only if they can control what, when and where it will be published.

Web 2.0 also fathered a new type of application: mashups. Mashups are Web applications which combine data and services from various sources. The richness of Web 2.0's content naturally leverages the growth of such applications, easily deployed, they can be found on many blogs and websites. However, as mashups are based on various content belonging to users, an explicit authorization is required to access the service interface (or API). This new need of third-party access authorization was addressed by several API authorization protocols which allow the user to "control" his privacy.

But, as efficient as these protocols are, they still rely on the user who decides to grant (or not) an access to a querying third-party, typically a mashup application. Indeed, the user has the *responsibility* to allow a Web application to access an API on his personal content. In this final and crucial step in the access authorization process, the user is alone, with no information, no comments, no guidelines, facing a usually barely known and potentially malicious third-party application.

To support the user's decision whether to grant or not to a third-party access to personal content, we propose an API Reputation Model (R2M). Improving the existing API authorization protocols, R2M brings a set of lightweight mechanisms which monitors the access and collects the users' feedback. Thus, it offers a better vision of how user's data is handled and a valuable indication on third-parties' trustworthiness.

The remaining of the paper will be organized as follows. In section II we will present the principles of secure API authorization mechanisms, then we will mention relevant existing solutions and discuss about the related open issues. In section III, we will introduce the main concept of our solution R2M, and describe its key mechanisms. Then in section IV we will present an implementation that we are currently experimenting, with screenshots and examples. We will also discuss about possible improvements and security considerations of R2M. Finally, we will conclude with future works and perspectives.

II. State of the art

In this section we present how a third-party, typically a *mashup* application (also known as *consumer* [6]), can access *users*' personal data available on a *service provider*; those are the three main actors of this scenario. We will describe the main existing mechanisms and protocols that enable a secure access on APIs and finally we will discuss about the related open issues.

A. The "Straw-Man" Approach

To access user's data stored by a Web Service Provider that does not provide APIs nor implements a secure authorization protocol, a mashup must get the owner's login name and

978-1-4244-5176-0/10/$26.00 ©2010 IEEE

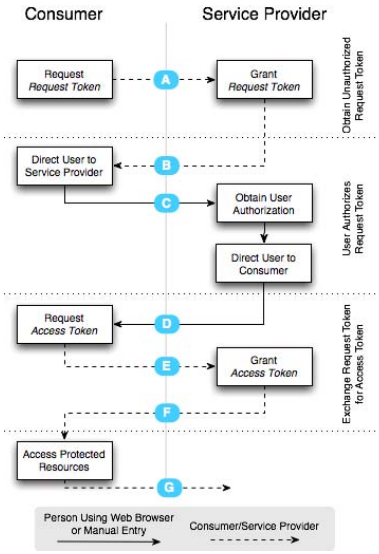

Fig. 1. OAuth authorization flow.

password. This was referred to as the *straw-man* approach in [1]. Afterwards the mashup can log into the Web Service Provider website on behalf of the user and access his personal content using various techniques: *screen-scraping* [2] (*e.g.* using HTML IFRAMEs) or simply by downloading identified resources (*e.g.* an xml data file).

However, providing the credentials to a third-party confers it an unrestricted access (in time and scope) to the user's data hosted by the corresponding WSP. Besides, the user account becomes totally exposed to the mashup's author who could misuse the collected personal information. Finally, users could be impersonated by any attacker compromising the mashup's database.

B. API Authorization Protocols

Web Service Providers expose APIs to enable newly deployed mashups to easily integrate their features. With the success of Web 2.0 and the growth of User-Generated Content (UGC) [3], these APIs are now, for the most part, implying users' personal data which should be handled while respecting the owner's privacy. However the basic straw-man approaches (cf. II-A) were clearly missing a secure way to access these public interfaces and provide a real control to the user such as the ability to grant *and* revoke an authorization.

So several API Authorization Protocols were proposed to address this concern. Basically, to grant access to a third-party application, these protocols redirect the user to a service provider login page. Then the user manually authorizes the requesting mashup to access his information through the secure service provider's APIs. This simple mechanism prevents the disclosure of the user's credentials while guaranteeing that he actually accepted that the mashup accesses his data. In the following sections we intend to present the main existing solutions and their differences, then we will explain why they are not fully satisfactory to this day.

1) AuthSub: AuthSub allows third-party application developers to securely access data from users' Google services [4]. The first time a mashup accesses a user's data, it is redirected to a *Google consent page* which asks for the user's granting and displays various information: third-party's URL and name but also the list of services it requests access to. Once the mashup is successfully authorized, it gets an *authentication token* which allows a single access to the services or can be exchanged for an unlimited access (or until revocation by the user).

2) OpenAuth: OpenAuth, proposed by AOL, provides restricted and full data access mechanisms [5]. Indeed, when the user authorizes a third-party, he can choose to grant access for the mashup either temporarily or permanently. The former option requires the user's consent whenever the mashup needs to access to his data. On one hand, this solution confers a strict and secure access to the user, on the other hand it is not suited for frequent mashups' accesses (and obviously for mashups that run permanently without a User Interface).

3) OAuth: The most recent and popular, OAuth [6] is an open-source initiative combining the strengths of major API authorization protocols, inheriting features from OpenAuth and AuthSub described above (cf. II-B1 and II-B2).

To access a user's data, a mashup must first get a *request token* from the service provider, see Figure 1 (A,B). Then it redirects the user to the service provider consent page where he actually authorizes (or not) the third-party application (C). Once the access is granted, the *request token* is authorized by the service provider (D). Unlike AuthSub, an *authorized request token* cannot be directly used by the mashup to call the API but must be exchanged for an *access token* (E,F) finally used by the application to get access to user's information (G).

C. Open Issues

All these protocols offer a secure and reliable access control on the Web Service Providers' APIs. Authenticated by the service provider, the third-party applications also need to be explicitly authorized by the owner of the data they want to access. Nevertheless, several issues are not addressed, especially regarding the relationship between the user and the consumer.

Indeed, the mechanisms described above are rather efficient but they all rely on the user's assessment of the third-party trustworthiness. The problem is that none of these protocols provide such information, leaving the user facing an application he barely knows requesting access to his personal data. Service providers are aware of this concern but the only answer so far has been a warning message that you may find, for instance, on the Google's consent page, advising to only authorize trustworthy mashups... see Figure 2.

The second main issue is the lifetime of the authorization. Current mechanisms only provide two access modes: temporary and permanent. None of these approaches are satisfactory. The former is clearly inconvenient, not suitable to the growing number of mashups. The latter is prone to privacy attacks, the user has no idea on how the third-party uses the authorized

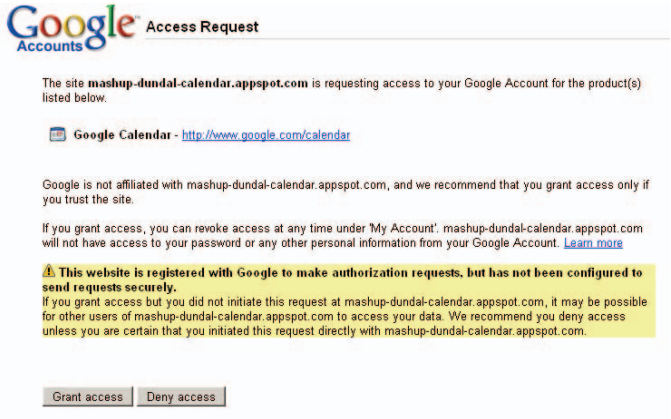

Fig. 2. Google's consent page.

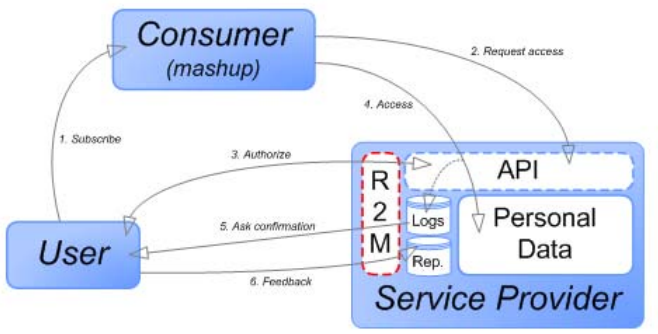

Fig. 3. R2M functional overview.

access: what is going on with his and others' personal data? A typical example of applications where a user would want to know more about the use of his personal data are FaceBook applications.

III. Reputation-Based Approach

In this section we will introduce our solution, the Reputation Model for Mashups (R2M) which intends to improve the existing API authorization protocols by addressing the issues identified in the previous section (cf. II-C). First we will introduce the main principle, then we will describe its key mechanisms.

A. Principle

Although *mashup* is a pure Web 2.0 concept, related security mechanisms and specially the API authorization protocols described above (cf. II-B) do not take advantage of the users' feedback. Unlike e-commerce platforms which integrate reputation mechanisms to ease the interactions between unknown sellers and skeptic buyers, Web Service Providers do not act as trusted intermediary between users and mashups.

However, users' input is a key of Web 2.0 success, widely exploited by websites and applications: folksonomy, User-Generated Contents, wikis and blogs; comments and ratings are applied to almost anything: movies, softwares, games, restaurants, ... The involvement of end-users in the Web makes them more confident in their relationship with unknown third-parties (WSP or consumers). We believe that this concept should be applied to complete existing security mechanisms.

The Reputation Model for Mashups, presented in this paper, addresses the authorization protocols issues outlined in section II-C by evaluating third-parties reputation according to the users' input. Deployed by the Web Service Provider, R2M logs every API call and gets users' experience feedback. According to the users' ratings on the mashup trustworthiness, we compute a reputation score. This value is then displayed on the Web Service Provider consent page to help users in deciding whether or not to grant access to a mashup. The model consists in a set of mechanisms to monitor the third-party applications, display statistics to the users and get their feedbacks (cf. Figure 3 for a functional overview).

B. Key Mechanisms

R2M is independent from the API authorization protocol implementation and do not alter its process. So it could be easily and seamlessly deployed over existing Web Service Provider APIs. As OAuth is the most popular authorization protocol at the time of writing, we will describe the R2M key mechanisms over it (they would be similar for any other protocol).

1) Authorization: This is the first step in the authorization process, the user is asked by the Web Service Provider to grant or not personal data access to a third-party application. As mentioned above, R2M does not alter the authorization process. During this phase, the only change is the display of additional information about the third-party on the WSP's consent page. The extra information will be the mashup reputation along with a warning message describing the potential threats and possibly users' comments or statistics on previous API accesses. Thus the user has a full knowledge of the risk level and the potential consequences of his decision: Instead of blindly trusting a mashup, a user can inspect, thanks to other people's statistics, which part of the data is read and/or modified by this mashup.

2) Stateful logging: Once the consumer is authorized, every API call on the user's data is logged by R2M. As all requests must be signed, the API authorization protocol identifies and authenticates the mashup so R2M can easily maintain logs for each consumer. This information is then used to generate access statistics, inform the user or detect misconduct.

As a mashup may alter a user's data integrity, the R2M stateful logging mechanism keeps track of any changes to allow the user to revert unwanted operations. Thus, the user is able to adjust his trust toward a third-party (revocation, negative rating or comment) according to its actions while keeping his data safe.

3) Feedback: User's feedback is the key concept of R2M. After a time period (or a number of uses) defined by the Web Service Provider, an authorization confirmation is sent to the user (probably by email). In this message, R2M informs the

user about the mashup activities on his data through relevant statistics outlining potentially unwanted/abnormal operations. To be even more specific, the user should be able to get the whole logs, recent comments and ratings and a description of API documented by the service provider. Last but not least, the message requests the user to give his feedback about the mashup (most likely on the WSP website) to confirm the access authorization.

On the WSP's confirmation page, the user has to rate and comment the mashup behavior according to his experience and the provided statistics. A satisfied user will confirm the authorization access, whereas an unsatisfied one will immediately revoke it to prevent the mashup from accessing his data. In case the user does not provide any feedback, it is up to the Web Service Provider to decide by default either to revoke the mashup authorization or to assume the user is barely satisfied.

4) Reputation computing: Mashup reputation is computed by R2M from the users' inputs and published on the Web Service Provider's consent page. The collected feedbacks are stored in a local database associated to the WSP. The third-party ratings provided by the users are then processed to obtain a mean and reliable value by preventing any attempt to falsify the reputation score (*e.g.* fake users, robots, etc). The use of local or centralized reputation database will be discussed later in section V.

Note that the evaluation of a mashup reputation requires a significant number of ratings to provide reliable information to the end-users. Indeed, in case of a new third-party application, the reputation score is not yet established, so R2M cannot guide the user during the authorization process. Nevertheless, the stateful logging mechanism (cf. III-B2) allows the user to revert any unwanted writing operations realized by a malicious mashup on his personal data; such a behaviour would dramatically decrease its reputation for the next users, thus limiting the possible harm to only a handful of early adopters.

5) Suspicious behavior detection: According to R2M, a mashup with a low reputation score has very little chance to be authorized to access to personal data. However, a formerly nicely reputed mashup could also change its behavior, for example due to an internal policy change or possibly to an attacker compromising the application. In such a case, as the reputation score is a mean value, it will decrease slowly and will not trigger a notification to the users about the ongoing threat in a timely enough manner.

To address this specific concern, R2M also monitors the API calls: type of interaction, frequency, etc. Thus R2M allows the service provider to detect unusual and suspicious activities. Then the WSP can warn its customers when a third-party that used to only read data suddenly starts modifying data or when the number of requests is growing abnormally.

As smart as R2M is, the user remains the best judge to consider if an activity is correct or not regarding his personal data. That is why R2M also detects suspicious behaviors based on the recent users' ratings. Indeed, if the recent ratings associated to a mashup are not corresponding to its current reputation score (typically lower), R2M alerts the WSP that will immediately inform users who previously granted access to this third-party.

6) Revocation: Although it is highly recommended, users hardly care (or simply forget) about revoking outdated authorization access. However, each access granted is a potential risk so it is important to minimize the number of currently valid authorizations by revoking obsolete access. Therefore, when an access remains unused for a while, R2M detects it and the service provider can ask to the user if he wants to revoke it. Moreover, the revocation process which is likely to take place on a Web Service Provider's page is another opportunity to collect the user's feedback (cf. III-B3) about the previously authorized third-party.

IV. Implementation

To experiment our Reputation Model for Mashups in real conditions, we deployed it on www.dundal.com, a service providing a communication hyperlink infrastructure and which exposes APIs via OAuth [7]. *Dundal.com* is a Bell Labs' experimentation [8] so it was straightforward to integrate our solution into it. This experimenting deployment allowed us to evaluate the feasibility of our solution over a real use case and identify issues and possible improvements.

A. Databases

Web Service Providers have to manage a huge amount of information. For example, if we consider *Dundal.com* implementing R2M over OAuth, the following data are stored (here in database tables). This scheme could be extended to any WSP and over any API authorization protocol.

- *Core business data*. Information specific to the WSP process, *e.g.* in *Dundal.com* they are the communication hyperlink containers and their properties or the users' profiles (*e.g.* name, email, etc).
- *Users' personal data*. Typically the sensitive information that must be protected, *e.g.* in *Dundal.com* they are the communication hyperlinks themselves.
- *Security data*. Information related to the various security mechanisms: users' and consumers' credentials, API authorization data (identifiers, granted accesses and scopes), etc.
- *Reputation* (R2M specific). Users' feedbacks about the consumers: comments, ratings, last confirmation date, initial mashup's score, etc. Users and consumers are uniquely identified and bound via an authorization identifier (*e.g. access token* in OAuth).
- *Logs* (R2M specific). The detailed and exhaustive history of API calls (method and paramaters), mentioning the requesting consumer and the targeted user's data. The original state of altered information may also be stored to allow to revert undesired operations. Because this history can become extensive with mashups that run offline and periodically, a simple mechanism can be implemented to move these pieces of information to another (offline) database periodically in order to avoid clogging the runtime database (current implementation simply drops

Fig. 4. R2M-enhanced consent page.

this information after the user has given its feedback on the mashup, see IV-C).

B. Web Service Provider's Consent Page

Figure 4 shows the integration of R2M in the Web Service Provider's consent page. The most significant changes here are the mashup reputation score along with a descriptive warning message and the number of voters, and a link to display users' comments (some statistics may also be proposed).

Mashup's reputation informs about the users' satisfaction regarding this third-party's trustworthiness *for this service*. Currently, the reputation corresponds to the average rating. We choose a simple five level rating system, ranging from *Dangerous* to *Trustworthy*. To be more specific and guide even better the users when authorizing a third-party, the rating is translated into clear text. This also allows the Web Service Provider to inform his customer regarding the potential risks.

C. Feedback Request

After a short time period defined by the Web Service Provider (*e.g.* currently fours days on *Dundal.com*, although in order to avoid being predictable we could instead add some randomness into the time period so that malicious mashups can not implement a well-behaved implementation until the user has given his feedback and then *misbehave*), the user is requested by e-mail (easily retrieved from WSP's database) to evaluate the mashup and to confirm the recently granted authorization. This confirmation message contains specific information to prevent phishing attacks: user's name, third-party's name description, date of authorization, etc. The message also presents a summary of the mashup's calls on the API, the involved user's data and the potentially suspicious activities (if any).

This information must encourage the user to give his feedback; however if the message is ignored, either a neutral score is automatically attributed or the authorization is revoked (*Dundal.com* implements the former option). Note that the user's feedback is collected on a specific WSP's page but it could also be parsed from the e-mail reply. In both cases, R2M assures that each user's review on the mashup is counted as only one vote (he also can still revoke the authorization at anytime).

Fig. 5. Email sent to request a review.

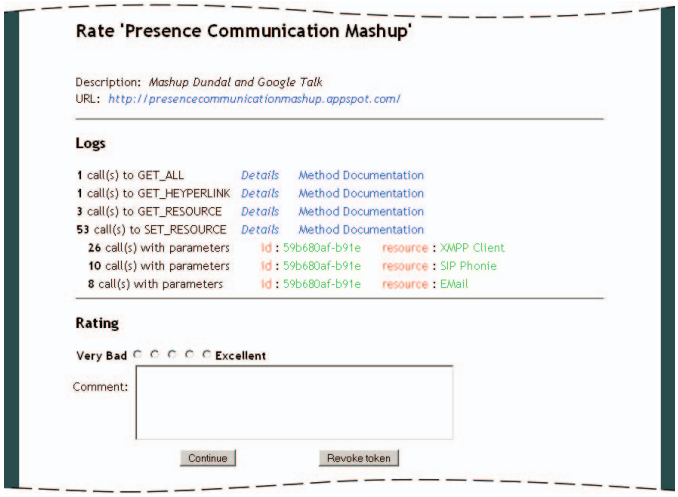

Fig. 6. Mashup rating page.

D. Rating Page

The rating page actually gets the user's inputs. As shown in Figure 6, the mashup name and description (provided by the consumer itself) are displayed. A summary of API calls is also presented with the possibility for the user to get access to the whole detailed logs. The number of calls for each method is indicated, R2M outlining those which exceed a normal behavior (based on average values over all users). As API names may not be explicit enough for end-users, a clear description can be provided by the WSP. So the user can grade the third-party's trustworthiness according to his experience and possibly add a comment, to finally confirm the authorization or revoke it.

V. Discussion

Some discussions on improvements and open issues related to the Reputation Model for Mashups.

A. Local vs. Centralized

In our implementation, we assumed that every Web Service Provider deploys and maintains its own R2M. However a

centralized approach could also be considered, implementing R2M as a new service which collects the reputation scores from all the WSP. The main advantage of this approach is the fast convergence of mashups' reputation, indeed the feedbacks will no longer be limited to a small population of users but to all of them. Then, the detection of third-party misbehavior would be even faster.

Nevertheless, we preferred a local implementation of R2M as it allows to easily get access to various information about the user (name, e-mail, id), the consumer (name, description, id) and the service itself (API description, authorizations); a centralized (*i.e.* non-local) approach would require yet another set of APIs to WSPs... Moreover, we also believe that the behavior of a third-party may differ from one service to another, thus dangerous activities on a small subset of data could be hidden by a nice reputation on most other services; so a per-service reputation is actually more accurate and relevant for the user.

B. Security Considerations

A reputation-based mechanism is intrinsically prone to various attacks. For example, the *Bad Mouthing* attack [9] that consists in users illegitimately rating a third-party negatively to artificially decrease its reputation, discouraging users to trust the mashup. It is necessary to assure that users are real and can only vote once.

Reputations systems are also vulnerable to *Sybil* attacks [10] when a malicious mashup forges additional identities to keep an acceptable reputation score anytime the previous one goes too low. Service providers can use the third-party URL as identifier or a certification authority to guarantee a unique identity.

These attacks also threatens reputation mechanisms implemented in e-commerce, peer-to-peer and ad hoc networks, so that they were widely addressed in many research works [11] [12]. These countermeasures could be adapted to R2M.

VI. Conclusion

The increasing number of mashups in the Web 2.0 landscape handling users' data through Web Service Providers' APIs requires new security considerations. We proposed to improve the existing API authorization protocols with a Reputation Model for Mashups providing a better vision on third-party activities. Thus, requests on the WSP's APIs are monitored to inform the users about mashups' trustworthiness and to detect potentially suspicious behaviors. Thanks to R2M, users hold all the cards to serenely decide to grant or deny an access authorization to barely known third-parties.

We implemented our solution to the Bell Labs' experimentation *Dundal.com*, a service that proposes APIs to manage communication hyperlinks [8]. This successful integration proved the feasibility of such an approach in combination with OAuth in a real use case. In the future, it will allow us to get the user experience on R2M to improve its key mechanisms and refine the reputation computing process.

References

[1] Ragib Hasan, Marianne Winslett, Richard Conlan, Brian Slesinsky, , and Nandakumar Ramani. Please permit me: Stateless delegated authorization in mashups. In *Proceedings of the 2008 Annual Computer Security Applications Conference*, Anaheim, California, December 2008. IEEE Computer Society Press.

[2] Alfredo Alba, Varun Bhagwan, and Tyrone Grandison. Accessing the deep web: when good ideas go bad. In *OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, pages 815–818, New York, NY, USA, 2008. ACM.

[3] Wikipedia. User-generated content. http://en.wikipedia.org/wiki/User_generated_content.

[4] Google. Google Account Authentication. http://code.google.com/apis/accounts/docs/OAuth.html, 2007.

[5] AOL. OpenAuth. http://dev.aol.com/api/openauth, 2008.

[6] OAuth Core Workgroup. OAuth Core 1.0 specification. http://oauth.net/core/1.0/, December 2007.

[7] Bell Labs. Communication Mashups. http://communication-mashups.com, 2009.

[8] Bell Labs. DunDal. http://www.dundal.com, 2008.

[9] Chrysanthos Dellarocas. Mechanisms for coping with unfair ratings and discriminatory behavior in online reputation reporting systems. In *ICIS '00: Proceedings of the twenty first international conference on Information systems*, pages 520–525, Atlanta, GA, USA, 2000. Association for Information Systems.

[10] John R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[11] David Zage Kevin Hoffman and Cristina Nita-Rotaru. A survey of attack and defense techniques for reputation systems. Technical report, Purdue University, 2007.

[12] Audun Josang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, 2007.