Distributed Data Usage Control for Web Applications: A Social Network Implementation

Prachi Kumari, Alexander Pretschner*
Karlsruhe Institute of Technology
76131 Karlsruhe, Germany
{kumari,pretschner}@kit.edu

Jonas Peschla, Jens-Michael Kuhn
TU Kaiserslautern
67653 Kaiserslautern, Germany
{j_peschl,j_kuhn}@cs.uni-kl.de

ABSTRACT

Usage control is concerned with how data is used after access to it has been granted. Respective enforcement mechanisms need to be implemented at different layers of abstraction in order to monitor or control data at and across all these layers. We present a usage control enforcement mechanism at the application layer. It is implemented for a common web browser and, as an example, is used to control data in a social network application. With the help of the mechanism, a data owner can, on the grounds of assigned trust values, prevent data from being printed, saved, copied&pasted, etc., after this data has been downloaded by other users.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Security

Keywords

Web based social networking, privacy policies enforcement, sticky policies, data usage control, Mozilla Firefox extension

1. INTRODUCTION

Usage control extends the concept of data protection beyond access control [1, 2]. That is, it influences the actions that can and have to be performed over data after access has been granted. In addition to access control requirements, usage control policies stipulate (i) what the recipient is allowed to do (rights) and (ii) what they must do (duties) with the data. Among other things, usage control requirements and the associated policies are relevant for privacy, the protection of intellectual property and/or secrets, digital rights management, and compliance with regulations such as HIPAA or SOX. The enforcement of usage control policies is particularly challenging in distributed environments where the data provider has no or only limited control over the IT infrastructure of the receiver.

Usage control policies can in general be enforced in two ways. *Detective enforcement* [3] aims at detecting violations of a policy. In case of a violation, usually a compensating, correcting, or notifying action is taken. In contrast, *preventive enforcement* aims at avoiding policy violations.

The subject of this paper is the preventive enforcement of usage control policies. These policies can and should be enforced at different layers of abstraction in the system. Among others, this has, for various policy languages [4–11], been done at the operating system level [12], at the X11 level [13], for Java [14, 15], the .NET CIL [16] and machine languages [17, 18]; at the level of an enterprise service bus [19]; for dedicated applications such as the Internet Explorer [20] or in the context of digital rights management [21–23]. The reason for this variety of enforcement mechanisms is that the *data* that has to be protected comes in different *representations*: as network packets, as attributes in an object, as window content, etc. In principle, all these representations eventually boil down to some representation in memory, but it turns out to be more convenient and simpler to perform protection at higher levels of abstraction. For instance, disabling the print command is easily done at the word processor level; taking screenshots is easily inhibited at the X11 level; prohibiting dissemination via a network is most conveniently performed at the operating system level; etc. The question of how data flows can be detected in-between different layers of abstraction is not the subject of this paper; see Section 5.

Instead, in this paper we present a usage control enforcement mechanism at the application layer, more specifically a web browser application. The context is privacy in web based social networks (WBSN) with use cases that have been taken from data protection requirements derived elsewhere [24]. One example use case that our system can handle is as follows: in a WBSN, Alice has a best friend Bob and an acquaintance Carol as her contacts. Alice wants Bob to be able to visit her profile and copy pictures from her album. Meanwhile, Carol should only be allowed to view Alice's profile and pictures but should not be able to copy her pictures. In today's WBSNs, it is not possible for Alice to enforce any such privacy controls where she can prohibit future usage of rendered data. We hence consider attacks on a user's privacy that emerge from *other users* rather than from the *provider of the WBSN*.

*This work was supported by the German National Science Foundation (DFG) under grant no. PR 1266/1-1.

CODASPY'11, February 21–23, 2011, San Antonio, Texas, USA.

Most of today's popular WBSNs offer access control mechanisms only. In these approaches, the user, in the role of a data provider, has no means to control the usage of the data once the social network's web server has delivered the respective web page. In particular, this means that all data that can be accessed by anyone *can also be used in any way*. We provide a mechanism to prevent this.

Few social networks provide some basic approaches to enforce usage control requirements. For instance, StudiVZ, a popular German social network for students, offers picture galleries where the context menu can be deactivated with JavaScript. This blocks the possibility to save a picture via a click on the right mouse button, or copy the selected content to the clipboard in this way. In general, it is also possible to use a feature of CSS that assigns different styles for each output media. If the user wants to print parts of a page with sensitive content, with this feature it is possible to hide the sensitive content or even the whole page. However, these techniques exhibit room for improvement. Among other things, suppressing the right click on pictures by JavaScript is only possible if JavaScript is enabled at all, and Javascript can be easily deactivated without much effort. Furthermore, this blocking prevents the user from accessing additional functionality in the context menu, which might not be intended to be deactivated.

Existing WBSNs also suffer from standard security problems, including social engineering, cross site scripting, hacking user accounts and breaking into the database or the system. Although these attack methodologies can be used by a sophisticated user to circumvent our mechanism, looking into these problems is outside the scope of this paper.

Problem. In sum, the problem that we tackle is the enforcement of usage control requirements in a web-based social network. We control events[1] such as print, copy&paste, save, etc. on *rendered data* like text and images. We hence consider privacy problems that are a consequence of other users rather than a WBSN provider misusing personal data.

Solution. Our solution is a system with three modules: the client (an extension for the browser of the user who accesses data from other users in a social network and which contains the policy enforcement point), the server (an augmentation of the social network system that generates and ships the policies that will be enforced by the client), and a policy decision point (a component that can be deployed anywhere and that evaluates user requests w.r.t. applicable policies). By means of a security analysis, we show that the policies are indeed enforced under specific assumptions.

Contribution. Our contribution is twofold: firstly, we are not aware of enforcement mechanisms for usage control policies at the application layer, achieved in the context of WBSN applications for a web browser. We present one here. Secondly, by means of the prototype, we provide insights into the limitations of solutions which are specific to one particular layer in the system. By the evaluation of the mechanism we also lay out the assumptions and conditions which need to hold true to provide guarantees in terms of security and effectiveness of the system.

Organization. Section 2 presents related work. Section 3, the core of this paper, introduces the proposed framework and the major terms used and describes the implemented mechanism both at the client and the server side. Section 4 evaluates the enforcement mechanism. The paper concludes by looking into possible refinements and planned future works in Section 5. Appendix A describes the architecture of the client-side enforcement mechanism.

[1]In the remainder of this paper, we will call all user interactions *events* whereas the term *action* denotes the corresponding measure taken by the policy enforcement point, e.g., execution, inhibition, modification.

2. RELATED WORK

Privacy and data protection problems in web based social networks have been outlined by a number of researchers [25–29]. One major issue is that WBSNs encourage users to share personal data without providing sufficient means to prevent the misuse of such data. Privacy settings in WBSNs generally offer only access control. They provide no means to control the usage of rendered data. This leaves users with the choice to publish data and lose complete control over the future use/abuse of it, or to refrain from publishing any data. The possibility of publishing data while maintaining usage control through policies has not been explored yet.

Research for privacy in social networks addresses two broad issues: (i) data protection from WBSN providers and (ii) data protection from other WBSN users. Solutions for the former tend to focus on decentralization of data storage in WBSN, while the latter focus on access control models for secure data sharing among users. Yeung et al. [30] have presented a decentralized WBSN architecture based on a friend-of-a-friend (FOAF) ontology [31]. Every user can upload his FOAF specification and profile data to his own trusted server. The friends/contacts of a user can access his data through his URI. Some other notable contributions in decentralization of data storage include Safebook, PeerSoN and MyNet. Safebook [32, 33] is a peer-to-peer WBSN which leverages trust relationships among users for data sharing. PeerSoN [34] is another such WBSN which uses peer-to-peer infrastructure coupled with encryption to enable users keep control of their data and use the social network offline. MyNet [35, 36] is an application that provides peer-to-peer social networking on mobile phones and other pervasive computing devices. In this paper, we consider WBSN providers as trusted parties and therefore do not look into this aspect of privacy protection.

The other category of solutions, directed towards securing user data against malicious WBSN users, propose access models for secure data sharing. These models are limited to providing differential access control and leave open the usage control aspect of data protection. We are aware of two models which use a similar approach as ours, but for access control. Gollu et al. [37, 38] have proposed Lockr, a model for access control in social networks based on social attestations and access control lists. Data access is granted based on social attestation which is a piece of metadata encapsulating a relationship between two users. However, Lockr does not provide any mechanism to prevent misuse of data once access has been granted. Another WBSN model has been proposed by Carminati et al. [39]. It is a decentralized rule based access control model which enforces access control at the client side. The access control rules are based on a trust model where users can rate their contacts as good friends, best friends, etc. This is similar to our approach for generating usage control policies but the cited model does not take into account the sensitivity of the data items (Section 3.3.1). The major difference, however, is that our system focuses on actually enforcing usage control requirements.

Other than these, the EU-funded research project PrimeLife has two applications for privacy in social networks [40–43]. Scramble! is a Firefox extension that stores encrypted data at the WBSN provider using the OpenPGP infrastructure. Clique is a WBSN that provides fine-grained access control based on *audience segregation*. According to this model, one user can have many *faces* (a profile with particular combination of information) according to different groups of his contacts. The groups represent social circles like family, colleagues etc. This work is not concerned with enforcing usage control requirements after data has been shipped.

Probably closest to our work in terms of usage control enforcement in web clients is that by Egele et al. [20]. They present a dynamic analysis method to identify if sensitive information flows out of the Internet Explorer. While their goal is entirely different from ours, this methodology can be used to identify user actions like copy&paste, save page etc. However, their dynamic analysis method identifies actions as illegitimate based on whether sensitive data flow was initiated by the browser or by its helper objects. It cannot identify an action as illegitimate if it was initiated by the browser, as in case of a usual copy&paste action.

To the best of our knowledge, there is no publicly available work that enforces usage control in WBSNs. However, quite some research has been conducted that focuses on usage control in other application domains and other levels of abstraction in the system [12–23], as explained in Section 1. All this work is specific to certain applications or levels of abstraction. They cannot directly be applied to the case of WBSNs as it requires enforcement of usage control policies in the server and the client applications.

At the server side, we have implemented a policy generation mechanism based on the trust model introduced by Kruk et al [44]. In their work, they have calculated trust to assess the relationship between two identities [45]. For this they introduce a *Friendship Level Metric* to assign a trust value between 0 and 1 to relationships. These trust values are multiplied on the paths between the resource owner and requester and the highest one is chosen. Depending on the *Social Network Access Control List* containing a maximum allowed relationship distance and the minimum trust, it is decided whether or not access is granted.

We use a similar approach, but we estimate exact trust values as we not only decide if data is accessible, but also to which degree it is usable, if access is granted to the requester. We want to stress that our implementation of the usage control mechanism is independent of the trust model used. It can be replaced by any other trust model as long as it can be used to generate usage control policies as we do.

At the client side, we have implemented a policy enforcement point in form of a Mozilla Firefox extension. There are many publicly available Firefox add-ons which provide some kind of privacy and security features, e.g., blocking advertisements and scripts, or hiding the user's IP address [46]. To the best of our knowledge, none of these add-ons can enforce data usage control requirements in the Firefox browser. In the context of privacy, a notable Firefox extension is the Tabulator which uses semantic web techniques to display the structure of web pages [47]. It can be used in the context of WBSNs to show a user how he is linked to other users and all the data displayed on his profile. This information can be used to increase awareness about data proliferation on the web. The extension does not enforce privacy policies.

Thus, to our knowledge, both in the context of WBSN and the web browser, we are the first to present an enforcement mechanism for usage control requirements.

3. USAGE CONTROL FRAMEWORK

3.1 User and System Requirements

Remember Alice's problem from Section 1. Alice wants to differentiate among people directly connected to her; and on this basis, she wants to control usage of her data. One solution to this problem is a policy-based usage control enforcement mechanism. The WBSN should enable Alice to specify usage control policies both for different contacts and for different kinds of data. At the same time, a policy enforcement mechanism for these policies is needed at the client side (web browser). There are hence two relevant sets of requirements, one set of requirements specific to policy generation mechanisms in WBSNs, and a second set for policy enforcement mechanisms in web browsers. (In principle, data can of course be accessed directly via the network, without passing through a browser. Many of today's social networks prohibit direct access via CAPTCHAs which solves the problem; the same is achieved, among other things, by authentication mechanisms of the kind discussed in Section 3.4.)

The user requirements that we implemented in our system have been derived elsewhere [24]. Typically, the user has requirements such as *only best friends can save my profile*, *only good friends can download sensitive pictures from my album* etc. Through such requirements, the owner controls three aspects of data usage: (i) Subject: "Who" (e.g., good friend, best friend) (ii) Object: "What" (e.g., pictures, profile) (iii) Events: "Usage" (e.g., save, download). The user requirements that we consider are given in Table 1.

Table 1: User requirements

Label	User requirements description
UR-1	User must be able to differentiate among contacts at the same degree of connection.
UR-2	User must be able to assign different sensitivities to different personal data.
UR-3	User must be able to define the set of allowed and inhibited actions.
UR-4	User preferences in requirements UR-1, UR-2 and UR-3 must be enforced together at the client side.
UR-5	Enforcement should not be client-specific.

We refine these requirements into a set of statements that, in the current implementation, refer to copy&paste, print, save, and view events, when user data is accessed by another user. Our policies are capable of expressing more complex requirements, including temporal and cardinal requirements [9], but for the sake of simplicity, we restrict ourselves to simple inhibition policies in this paper. The policy decision point (PDP), however, can interpret and monitor the entire expressiveness of the language.

To enforce these requirements, we derived a set of system requirements for the user requirements. These requirements detail the processes of policy generation and enforcement at the server's and the client's sides (Table 2).

While requirement SR-18 is relevant in practice, we are interested in a prototypical implementation and decided to implement our system solely for the Mozilla Firefox browser. This choice is motivated by three main reasons: firstly, it is possible to modify the functionality of the browser without

Table 2: System requirements for policy generation and enforcement

Label	User requirement description
SR-1	Contacts can be assigned to one of the following predefined categories: *best friends, good friends, friends, acquaintances, never met.*
SR-2	Categories in SR-1 are mapped to trust values 1.0, 0.8, 0.6, 0.4 and 0.2 respectively.
SR-3	New categories for contacts can be created by the user.
SR-4	A trust value ranging between 0.0 to 1.0 must be assigned to the new category at the time of creation.
SR-5	Personal data can be assigned one of the sensitivity levels: *private, high sensitive, medium sensitive, low sensitive, not sensitive.*
SR-6	Sensitivity levels in SR-5 are mapped to values 1.0, 0.8, 0.6, 0.4 and 0.2 respectively.
SR-7	Permission classes are *maximum permission, high permission, medium permission, low permission, minimum permission.*
SR-8	The user defines which of the four usage events viz. copy item, save page, print page and view page source are allowed/inhibited for each of the permission classes.[a]
SR-9	The controlled events in SR-8 apply *only* to text, images and the complete page excluding the content rendered by plug-ins.
SR-10	The policy is generated combining the requirements SR-(1-9).
SR-11	The WBSN sends the policy to the web browser on the client side.
SR-12	Before delivering content protected by usage control, the WBSN has to be sure, that there is a browser at the client side, which can enforce usage control and understands the information regarding usage control given by the server.
SR-13	The browser is able to provide authentication guarantees.
SR-14	The browser is able to enforce the received policy.
SR-15	The browser should differentiate between protected and unprotected pages[b] opened at the same time.[c]
SR-16	Policies delivered by WBSN have to be enforced by the browser until revocation.
SR-17	WBSN (and in turn the user) may revoke policies.
SR-18	Policies must not be browser/client specific.[d]

[a] At present, all other events than these four are out of the scope of our application.
[b] A protected page contains sensitive data protected by a usage control policy.
[c] This requirement distinguishes our solution from the Javascript fixes mentioned in the introduction section.
[d] This leaves scope for a more generic solution irrespective of the policy execution point.

modifying its core (add-ons, plug-ins); secondly, it is open source, has good SDK and developer network support and therefore is comparatively better suited for experiments than other browsers; thirdly, it is the second most used browser [48, 49], so offering a solution for it is likely to have more impact than other browsers.

3.2 High Level Interactions

We are now ready to describe the system which implements the system requirements given in Table 2. To translate these requirements to policies, we introduce a policy generation component in the social networking application. As policy generation requires modifying the source code of the social networking application, we could not use one of the existing popular applications like Facebook for our purpose. The instantiation of our social networking application is called SCUTA, an abbreviation for 'Social network with Control of Usage and Trust Assessment.' It is a modification of the PHPizabi social networking platform [50]. The general idea is to use trust ratings for variable usage control. In our example, Alice can rate Bob and Carol for trust. The respective trust values are used to generate usage control policies for Bob and Carol. We would like to re-emphasize that although SCUTA works on trust assessments, it is independent of any particular concrete trust model. We are perfectly aware that trust is a challenging concept and that every single trust model is usually subject to a heated debate (which we do not want to enter in this paper). As a consequence, we simply assume an intuitive notion of trust to be given.

Figure 1 shows an overview sequence diagram of the usage control enforcement process. The policy enforcement point (PEP) resides at the client side in form of a Mozilla Firefox extension. We call it BRUCE, an acronym for 'Browser-side Usage Control Enforcement.' The server, SCUTA, generates policies and delivers both requested data and an associated policy to the client. The policy is then sent by BRUCE to the PDP and stored there. We chose to send the policy via BRUCE instead of sending it directly to PDP because we wanted to reuse the connection between BRUCE and the PDP (which is anyway required for enforcing the policy). In case of a usage event, BRUCE queries the PDP. The PDP allows or denies the attempt based on the policy.

Note that the PDP can be deployed at any of the three locations: the client, the server or a remote point. In our demonstrator, the PDP is run remotely. Moving the PDP out of the application-specific parts of usage control solution has the benefit that multiple PEPs which are application specific parts, can use the same PDP technology. This results in a greater knowledge base about occurring events for the PDP and avoids additional complexity for approaches to usage control at different levels of abstraction (see Section 5). Also, in the local network of a company, resource consumption could be reduced if the client machines of all employees were attached to one single PDP running on a central server. In our example, when Bob or Carol access Alice's data, irrespective of the client they use, they query the same PDP.

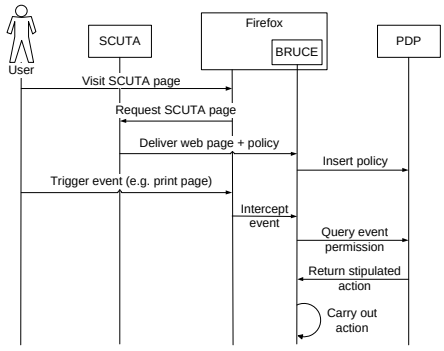

Figure 1: High level sequence diagram of the system

3.3 Policies

3.3.1 Permissions

In our policy generation model, we use three major terms: *trust*, the level of confidence one user has in the other user

for not misusing his data; *sensitivity*, the intuitive degree of a data item being worth protection; and *permission*, the set of permitted user actions for a data item. We define permissions as a function of trust and sensitivity.

Trust. SCUTA provides functionality for a user to cluster his direct contacts into five predefined categories: best friend, good friend, friend, acquaintance and never met. There is a sixth class, not connected, that is automatically assigned by the system and that reflects the fact that the respective user is not in the friends' list of the other user. These six labels of contacts are mapped to trust levels on an equidistant scale of 0 to 1. 0 corresponds to not connected, .6 corresponds to friends, 1 to best friends, etc. Users can also create new categories for the contacts (e.g., colleagues) and assign trust values to them.

Sensitivity. Sensitivity is not global for all personal content of one user, but can differ for each data item. For example, in general an email address is more sensitive than the first or last names. SCUTA allows its users to rate their content according to five predefined sensitivity levels: minimum sensitive, low sensitive, medium sensitive, high sensitive and private. In addition, there is a sixth sensitivity level that implicitly labels all those data items that have not been rated for sensitivity by the user; their sensitivity is considered to be the lowest. Similar to trust assessments, these categories are mapped to equidistant sensitivity levels such that 0 corresponds to not labeled, .8 corresponds to highly sensitive, and 1 corresponds to private content.

Permissions. Trust and sensitivity are then combined to compute permissions $p(t, s) = t \cdot (1 - s)$ where t is the trust rating of the recipient, and s is the data item's sensitivity value. SCUTA maps them to five predefined classes of permissions: minimum, low, medium, high and maximum that correspond to intervals of the same size (e.g., minimum permission iff $0 \leq p(s,t) \leq .2$ and high permission if $.6 < p(s,t) \leq .8$). Before these permission classes are computed, SCUTA verifies the existence of a PEP at the client side (Section 3.4). If the PEP does not exist, SCUTA nonetheless computes the permission class for each data element and delivers only the content for which all usage events are allowed. If there is no such permission class, no content is delivered. This is SCUTA's default behavior in absence of a PEP; it cannot be changed by the user.

Users can define what events are allowed for a particular category of permissions, as exemplified in Table 3.

Table 3: Mapping events to permission classes

Permission class	View item	Copy item	Save page	Print page	View page source
Minimum permission					
Low permission	x				
Medium permission	x	x			
High permission	x	x	x	x	
Maximum permission	x	x	x	x	x

The semantics of our policies is permit-override. This means that by default, everything is permitted. Explicitly stated prohibitions override this default.

3.3.2 Server-Side Policy Generation

Whenever a SCUTA-protected page is downloaded by a BRUCE-enabled Firefox browser (more precisely, in one tab of the browser), a new policy is created and sent to the PDP. Each policy consists of a set of rules, one for each usage event and each permission class unless the default permissions apply. Usage is hence controlled identically for every data item within the same class, be this an image, a piece of text, etc. However, depending on the permission class, there can be different rules for different usage events. This design decision is motivated by efficiency and simplicity considerations—we could as well have chosen to have one rule per data item, resulting in possibly much larger policies.

If a web page is requested, SCUTA computes the permission class for every sensitive data item on the page. We then proceed in two steps.

Firstly, each sensitive data item is embedded into an HTML element which captures the computed permission class as shown in Listing 1. This augmented HTML code is shipped to the client.

```
<div>
 <table>
  <tr>
   <td><strong> E-mail </strong></td>
   <td><span class="lowPermission">
        alice@example.net </span></td>
  </tr>
  <tr>
   <td><img src="../images/Alice.jpeg"
        class="mediumPermission"/></td>
  </tr>
 </table>
</div>
```

Listing 1: HTML code with protection

Secondly, SCUTA generates a policy that associates a permission class with the set of allowed and prohibited actions for each of the permission classes. This is done w.r.t. the user's preferences, as stated in his WBSN profile. For instance, on the grounds of this information, it can be established that a *print* event is disallowed for all elements in the low permission class. We differentiate between data for which access is denied and data for which access is granted but usage is controlled. The idea is that for all data, for which access is denied (*view item* restricted in permission class), no usage control policies need to be generated, as access is a prerequisite for usage.

The policy is assigned an ID, the scope ID, that will be used by BRUCE to associate a specific browser tab (displaying the content governed by the policy) with the computed policy (there can be multiple tabs with different policies). The *scope* of a policy hence refers to one browsing session in one browser tab.

Finally, the HTML code that assigns permissions to data items as well as the policy that assigns permissions to allowed and prohibited usage events is shipped to the client.

3.3.3 Events

BRUCE controls events that correspond to usages of sensitive data items. In the policies, these are abstractly specified as, for instance, "copy" or "print." As an example, see Listing 2. These abstract events abstract from a specific browser technology: while copy&paste event is implemented differently in different browsers (and possibly in different implementations of the same browser), the main functionality is intuitively the same.

Viewed from the Firefox perspective, one abstract event can correspond to multiple internal events. Internal events ("cmd_copy," "context-copyimage-contents," "cmd_print,") are

raised whenever user commands are executed. Usually, different user commands can trigger the same internal event. For instance, a copy to clipboard operation (one internal event) can be invoked from the edit menu, from a context menu, and by pressing the Ctrl-C keyboard shortcut (three different user commands).

While we specify policies at the level of abstract events (see above), our enforcement technology works at the level of internal events. This is explained in Section 3.5.

3.3.4 Concrete syntax: ECA Rules

Technically, policies come as sets of event-condition-action (ECA) rules [51, 52] with one rule per permission class per usage event, unless the default permission applies. The ECA rule describes what *action* has to be performed if a specific *event* is triggered and the *condition* has been evaluated to true. Trigger events are provided as abstract events in the above sense. Remember that they correspond to internal events (and respective user commands) executed on data, such as print, save as, etc., and that they are also used to define permission classes. As mentioned above, in the condition part, our policies can be specified for complex temporal logic conditions, but in this paper, we stick to conditions that always evaluate to true. Moreover, at present, the policy can contain only 'allow' or 'inhibit' in the action part. Other actions, including modification, execution, and delay [51], is the subject of current work.

An example ECA rule is given in Listing 2. It shows an ECA rule for a *copy* event, as defined by `<id>` element of the *triggerEvent* section. As mentioned above, the scope will be used by BRUCE to associate this policy with one specific browser tab. The PDP returns the action *inhibit* if the condition holds true; in this example, if the permission class of the content is either medium or low.

```
<controlMechanism>
        <id>copySelected(4bed490f465e5)</id>
  <description> prevents copying a content, if
      the content is marked by a specific class
  </description>
  <triggerEvent
      xmlns="http://www.master-fp7.eu/event">
    <id>copy</id>
    <parameter name="scope"
        value="4bed490f465e5"/>
  </triggerEvent>
  <condition
      xmlns="http://www.master-fp7.eu/pastOSL">
    <or>
      <XPathEval>
        /triggerEvent/parameter[@name='class']/
            @value='mediumPermission'
      </XPathEval>
      <XPathEval>
        /triggerEvent/parameter[@name='class']/
            @value='lowPermission'
      </XPathEval>
    </or>
  </condition>
  <actions>
    <inhibit/>
  </actions>
</controlMechanism>
```

Listing 2: Exemplary ECA rule

3.4 Policy Deployment

Before the generated policy can be deployed, SCUTA has to verify the existence of BRUCE on the client side. If BRUCE is not installed, the client should not render any usage-protected data at all. The general idea is that server-side applications that support usage control would drive client users to obtain and install the extension because if no usage control enforcement mechanism is in place, access to usage-protected data would simply be denied. Once the existence of BRUCE has been established, SCUTA needs to send control information to BRUCE, including usage control policies and the scope IDs assigned to delivered pages. We implemented this with the help of custom HTTP headers. By doing so, it is also possible to use HTTPS communication without much effort to prevent some attack scenarios (Section 4).

Table 4: Header Field Commands

Header Field	Value	Semantics
x-auth_phrase	an arbitrary string	The extension send the x-auth_phrase value to the URL specified in x-auth_url.
x-auth_url	a valid URL	
x-policy_url	a URL pointing to a policy	The extension fetches the policy and inserts it into its responsible PDP.
x-set_scope	a unique identifier	The receiving tab is associated with the value. Usage commands will be checked for allowance with the scope as one parameter.
x-release_scope	a unique identifier	The scope associations for all tabs to the given value are removed and usage commands are blocked until a new URL is loaded.
x-revoke_policy	one or more identifiers separated by pipe symbols	The extensions informs the PDP about the mechanisms which shall be removed.

To the end of providing some basic authentication without relying on Public Key Infrastructures, we introduced two HTTP headers, *x-auth_phrase* and *x-auth_url* (see Table 4 for a list of header field commands, or HFCs). SCUTA starts by sending an authentication phrase (a nonce) to the client. By using HTTP headers, the server sends an authentication URL which is used by the client to send back the authentication phrase. A normal browser without a BRUCE PEP installed would simply not answer with an authentication response. Therefore, after a certain timespan, SCUTA can safely assume that the client is not equipped with BRUCE.

This very simple type of authentication is not intended to be secure against malicious attacks but only introduced as a placeholder for a more sophisticated solution. Such a solution could be based on the usage of public key cryptography. The above nonce can then be encrypted by SCUTA using the extension's public key. This will make sure that the nonce can only be decrypted by the extension using the private key. The extension then would compute the response to the received nonce and return it encrypted with the SCUTA's public key. Because this authentication mechanism would require the extension to have its own private key, we would of course need to secure it from being leaked or tampered with. For a more detailed security analysis, see Section 4.1.1.

Once the authentication has succeeded, SCUTA sends the header field command *x-policy_url*. BRUCE then fetches the respective policy and the scope ID (header field *x-set_scope*), and deploys the policy on the PDP via an XML-RPC message.

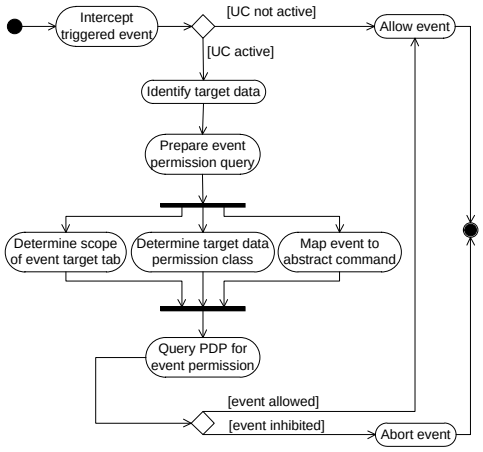

Figure 2: Client-side enforcement–the BRUCE perspective

If policies are to be revoked because a session ends, then SCUTA communicates this to BRUCE via the remaining two header fields, *x-release_scope* and *x-revoke_policy*. BRUCE then instructs the PDP accordingly.

3.5 Client-Side Policy Enforcement

Figure 2 provides a high level description of the usage control enforcement functionality of BRUCE.

Remember that user commands (e.g., Ctrl-C) trigger internal events (e.g., "cmd_copy"). In a nutshell, whenever such an internal event is raised, BRUCE retrieves the corresponding abstract event. This is because policies are specified in terms of abstract events. BRUCE then queries the PDP w.r.t. the abstract event that corresponds to the internal event. Depending on the decision of the PDP, it does or does not execute the code that corresponds to the internal event.

From BRUCE's viewpoint, each usage has three aspects: a scope, a target data item, and the abstract event that the usage is mapped to. Since our policies are based on permission classes (one rule per usage event per permission class unless default permissions apply) rather than on data items, BRUCE does not need to send a reference to the target data item to the PDP. Instead, queries sent to the PDP contain the abstract usage event and the scope as parameters.

The scope of an event corresponds to the scope ID of the policy that was generated for the protected page on which the event was triggered—that is, one specific session in a browser tab. The scope must be sent because there may be multiple tabs with different applicable policies. With this information from a query, the PDP knows which of the active policies applies.

The target of an event is either the entire protected page, a single DOM element or a set of DOM elements on the page (the Document Object Model, DOM, provides a structural representation of HTML documents which represents HTML elements as objects that can be accessed from arbitrary programming languages). This depends on which event the user triggered and whether or not multiple items are, for instance, selected on the page that is to be protected.

The extension sends a query to the PDP for each usage event. If the response is inhibition, then the intercepted event is aborted. The architecture and inner workings of BRUCE are described in Appendix A.

3.6 Example Revisited

Now we demonstrate how SCUTA and BRUCE help Alice grant different usage rights to Bob and Carol. We know that Alice has rated Bob as best friend and Carol as acquaintance. This assigns Bob and Carol the trust ratings of 1 and .4, respectively. Alice has a low sensitive picture with a sensitivity of .4. When Bob and Carol access this picture in Alice's profile, the picture is delivered to Bob's client with medium permission class ($p = 1 * (1 - .4) = .6$). A respective code snippet is shown in Listing 1. Carol's client gets the picture with low permission class ($p = .4 * (1 - .4) = .24$). The definition of actions in permission classes is as shown in Table 3. According to these settings in Alice's profile, Bob is allowed to copy the picture while Carol is only allowed to view it and cannot copy. The generated policy consists of a set of rules like the one in Listing 2. So when Carol tries to copy this picture, BRUCE queries the PDP for the allow/inhibit status of the event 'copy.' The PDP replies with 'inhibit' and the extension blocks the abstract command 'copy.' Thus Alice's picture is protected for copy by Carol.

4. EVALUATION

The goal of our system evaluation is twofold. On the one hand, we want to analyze and demonstrate the ability of our system to withstand deliberate attacks from malicious users. To this end, we explored possible ways in which the security of our system can be compromised by means of attack trees. On the other hand, we want to understand the limitations of our system. These were found out as a result of the security analysis. This helped us finalize the assumptions under which the system can provide guarantees about the enforcement of usage control (Section 4.2), the limitations of the system (Section 4.3) and future work in this direction.

4.1 Security analysis

Our evaluation is strictly limited to the application layer and does not cover vulnerabilities arising from the other layers. For example, the system can be circumvented by accessing usage controlled data directly in the cache folder via an external file browser, by inspecting the main memory with a memory spy, or by taking screenshots from the screen. We will get back to these issues in Section 5. Moreover, we do not consider attacks from WBSN providers since these are considered trustworthy in our context.

The attack tree for performing prohibited events on data is shown in Figure 3. The subtrees with the goals "get access to database" and "hijack user account" show attack scenarios for SCUTA.[2] We do not go into the details of them as they are classical security problems in themselves. However, we mention one standard attack in which resources such as pictures can be directly accessed by typing their URLs in the address bar. Luckily, there is a standard solution to this problem. The server generates a (random) token and

[2] *Login* shows the attack scenario where a malicious user can login as SCUTA administrator

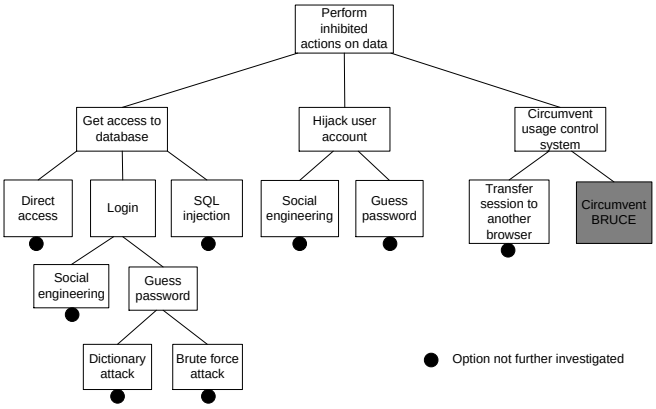

Figure 3: Attack tree for the system

a timestamp and includes them in the URL of the sensitive data item. The token is not bound to any user but is specific to a time interval which is used to invalidate the URL after a given timeout. In other words, before shipping the data item, the web server checks if a permitted time interval has not been exceeded; or it simply deletes the data item after the first download. In Listing 1, we show the unmodified URL of the image for simplicity's sake.

In the remainder of this section, we explore the subtree "circumvent BRUCE" in detail. For analyzing the circumvention of BRUCE, we treated SCUTA and the PDP module as black boxes and performed the security analysis under the assumption that they are secure with respect to the usage control system's specified functionality. Further security analysis covers three aspects of BRUCE: (i) Firefox's extension system; (ii) communication between SCUTA and the usage control extension; and (iii) communication between the PDP module and the usage control extension. The subtree is detailed in Figure 4.

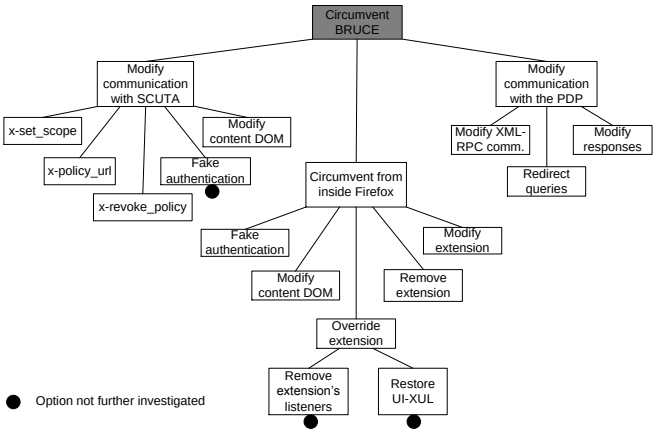

Figure 4: Attack tree for BRUCE

System vulnerabilities can be exploited in all the three submodules. As the server delivers sensitive content as soon as the client seems to be authentic, an attacker only needs to fake the authentication of the extension and leave the rest to the server itself. The extension only processes messages and commands it receives. Whether they were manipulated before they arrived is not detectable. Also, the interception and dropping of messages is not detectable. At the PDP, as it only gets called by the extension and does not initiate any communication with other systems by itself, it is quite easy to overcome it. There is no chance that the PDP module gets to know about modification or omission of policies and permission queries. Also, if the PDP resides at a remote node additional points of attack in the system arise because the attacker could try to break in the traffic routed between the PDP and the client.

4.1.1 Modifying client-server communication

Both the message contents and its headers can be modified between SCUTA and BRUCE. As the provided solution uses information about HTML tags stored in the document, the system could be compromised by their modification. Policies refer to certain attributes of HTML elements to define usage restrictions. If the values of these attributes were changed, the extension would query for permission of commands with parameter values that differ from the original ones used in the policies. Thus the PDP module would allow those commands although they should have been inhibited.

This kind of attack could also be performed from the inside of Firefox by a malicious extension that modifies a web page after rendering, leading to the same result.

Instead of modifying HTML content, an attacker can also change the message headers. The header fields that can be used for this are 'x-policy_url', 'x-revoke_policy' and 'x-set_scope'.

Remember that 'x-policy_url' indicates the URL for the policy that has to be applied to the message's content. The extension will obtain and then insert it into the PDP module in order to make sure that it will be enforced. But as the URL's authenticity and integrity is not checkable, one could either replace the URL by another which points to a policy that grants all usage, or even easier: just delete this header.

Upon inspection of the HTTP messages that the browser receives, one is able to find out which policies were sent to it. Whenever a message with the header field 'x-policy_url' set arrives, the attacker can retrieve the specified policy. By investigating it, he could determine the IDs of the activated policies. This knowledge allows to insert 'x-revoke_policy' header fields with exactly these IDs into messages before they reach the browser. Due to these header fields, the extension would instruct the PDP module to discard the respective policies and from then on usage control for the messages mentioned in the beginning would be broken.

If a message contains a set_scope instruction, this means the server tells the extension to put the contents under usage control. So the simplest way to get around the system would be to remove this header field before the messages arrive at the browser. Although this would be sufficient, changing the value of this header to something that is not a known scope in the PDP module instead would have the same effect. The tab receiving the message would be taken under usage control and triggered commands checked for permission. But, as the PDP module would not know of the modified scope value, it would allow the commands.

Manipulation of messages and headers is a serious threat to the system. As the three involved subsystems do not exchange additional information to validate their intended actions and the integrity of emitted messages, one could easily

compromise the system using a man-in-the-middle attack. The rather simple solution in this case is to use HTTPS instead of HTTP for the communication.

4.1.2 Modifying communication with the PDP

The usage control framework could also be circumvented by modification or redirection of messages exchanged between the extension and the PDP module. Due to the lack of integrity and authenticity checks in the system, such attacks would not be noticed. The XML-RPC interface provided by the PDP module offers methods for policy insertion, command permission checks and revocation of policies. An attacker can modify the request or response to circumvent the mechanism easily. For policy insertion calls, there are several kinds of changes that could be applied, but the easiest would be to replace the given policy with an empty one as this would lead to permission for all queried events. An attacker can also record the identifiers of the specified control mechanisms in the policy, instead of modifying them. These identifiers then can be put into a fake policy revocation call. The attacker can also redirect communication to another target server that supports the same interface as the PDP, but which handles incoming calls according to the attacker's wish.

A man-in-the-middle attack can easily compromise the usage control system just like it can by modifying the client-server communication. However, while the client and server have the possibility to use HTTPS in place of HTTP to exchange their messages, our current PDP module does not support HTTPS for communication (which is not a conceptual problem but a matter of maturity of our implementation). Hence, the unencrypted messages can be modified with no big effort. Therefore encryption and integrity checks should be considered in future work. For example, to discover if policies have been modified, the PDP module could compute a hash over received policies and send it to the originating web application. This way policy integrity could be checked, assuming authenticity of such messages could be assured. Tampered policy revocations could be prevented if the PDP module asked the respective web applications for authenticity of incoming revocation instructions.

4.1.3 Circumvention from within Firefox

Another vulnerability arises from interfering with BRUCE's behavior from within the Firefox browser. This can be done by modifying BRUCE's source code (which is written in Javascript) or deploying a nullifying extension. Some of the exploits are listed below:

- Reset usage command handling by changing back the overridden commands to their original definition;
- Revoke policies directly after they have been sent to the PDP module;
- Modify the scope mapping mechanism, e.g. discard all mappings;
- Alter usage command handling such that no queries are sent to the PDP module;
- Add internal events (and corresponding user commands, see Section 3.3.3) that exactly copy the behavior of usage-controlled internal events—since the name of the internal event is different, usage control requirements will not be enforced for these new events; or
- Fake the extension authentication.

Besides this, providing object orientation via prototyping-based programming is a special feature of Javascript that makes it easy to modify existing objects. This includes the possibility to replace methods by any arbitrary function definition. This complicates shielding the extension's code against malicious injections. Also, attempts to protect the extension against modifications of its source code would be futile as they could just be removed.

The aforementioned change can also be done by extracting the extension from its XPI archive[3], then modifying the corresponding source file and repacking it. After the next start of Firefox, usage control would not work anymore. To counter this, BRUCE's integrity should be checked right before it gets loaded. Another possibility to achieve this attack goal is by installing a malicious extension which can override the methods dynamically. In this case an integrity check at the start may prove useless. Detecting such changes during runtime appears to involve a considerable effort. The threat induced by a possible malicious extension is the bottleneck of the current system. Perhaps it is feasible to check other installed extensions if they modify BRUCE before the browser starts—a subject of future work.

4.2 Assumptions

Based on the evaluation of the system in the previous paragraphs, we draw the set of assumptions under which the system provides usage control according to the stipulated requirements and cannot be circumvented or forced to work in unexpected ways.

Our system guarantees client-side browser-level enforcement of usage control policies specified by the user and generated at the server side (user requirements given in Table 1) under the condition that the following requirements are met:

1. The communication between SCUTA and the Firefox extension is carried out over a secure channel.
2. The PDP module is extended by the capability to communicate over a secure channel.
3. The client must be able to guarantee that the Firefox extension is not modified before and after installation.
4. The client must be able to guarantee that no malicious extension is installed and if installed can be detected while running the extension.
5. A tamper-proof mechanism to authenticate the Firefox extension with SCUTA exists.
6. As every data that is delivered to the client is also stored in cache folders or displayed on a screen at the client side, usage control enforcement mechanisms exist to solve the problem at other levels of abstraction (see Section 5).

To meet the requirements 3 and 4, one could think of a client system which provides usage control on the levels below the application level. Then, the installation package of the Firefox extension could be shipped with a policy attached which implies satisfaction of these requirements.

Under the condition that the aforementioned requirements hold true, we consider our system to be secure at the client side and at the communication links level with respect to the attacker model presented in the security analysis. However, at the server side, attacks like social engineering, cross

[3]XPI is short for Cross-Platform Installer Module. It belongs to Mozilla's XPInstall technology, see https://developer.mozilla.org/en/XPI.

site scripting, hacking user accounts and breaking into the database or the system are classical problems and we cannot provide any guarantees against them in this paper. Looking into such more general security issues is not in the scope of this paper (Section 1).

4.3 Limitations

The developed system enforces usage control for data that is delivered by websites via web pages to the client. It implements the requirements given in Table 2. However, there is one limitation of our implementation in terms of functional completeness: The solution has been implemented only for native data. By native data we mean the content that Firefox can render by itself and this does not include contents handled by plug-ins [53], for example, Flash or PDF.

The limitations of the system, found during the evaluation, can be summarized as follows.

1. The system provides usage control guarantees only for the case of native data.
2. The system provides guarantees only if the assumptions in Section 4.2 are met.
3. The system provides usage control guarantees only for the *single user page* scenario (see below).
4. The system provides usage control guarantees only at the browser's level of abstraction. Data stored in cache folders is not protected; screenshots can also be taken.

In a *single user page* scenario, the page being rendered contains data only about one user, e.g., Alice's homepage. This can be contrasted to *multiple users page* scenario where the rendered page contains data about multiple users e.g. page displaying search results for "Alice" as there can be many users with the name Alice and their basic information like name, city, country are displayed on the webpage in a list. In the evaluation, we also found that implementing this does not seem to be a conceptual problem and is left as future work.

5. CONCLUSIONS AND FUTURE WORK

We have presented and evaluated a framework for enforcing usage control requirements at the application level for the particular case of web browsers. As an example, we have implemented our framework in the context of privacy protection in web based social networks. We use the case of WBSNs only for the demonstration of our proposed mechanism. The usefulness of our mechanism is of course not limited to WBSNs but extends to all the web applications that are accessed using a web browser. Referring to the use case in the introduction, when Alice specifies Bob and Carol as different kinds of contacts, they are assigned a trust level and based on the sensitivity of the data being accessed, Bob and Carol receive data with different policies. So while Bob can copy and print Alice's personal data and pictures, Carol can only view selected parts of the profile and cannot copy or print anything.

In the proposed framework, SCUTA acts as a policy generation point which can generate and render usage control policies in the form of ECA rules. The generated policy is communicated to the client through HTTP header fields. BRUCE provides usage control for Firefox's native data types to SCUTA. Policies that are bound to delivered content are fetched and then enforced by the framework. As it only serves as policy enforcement point, it utilizes an external policy decision point via a fixed XML-RPC interface.

As the evaluation points out, our system guarantees client side data usage control enforcement at the browser level if certain assumptions hold true. This guarantee is with respect to the attacker model presented in the security analysis part of the evaluation. However, additional work has to be done in order to make sure that these assumptions are always met, e.g., for all clients and all content types. In this respect, our solution is not mature yet. To complete the set of manageable kinds of data, e.g., by Flash animations, additional effort has to be put into the examination of embedded data types, which are rendered in Firefox through plug-ins. The heterogeneity among them presents a challenge for a general solution which is independent of particular content types.

The guarantees provided by our mechanism depend on active security mechanisms implemented in other parts of the system. The security analysis investigated several possible types of circumvention. Intuitively, integrating encryption and certification into the system could improve security. Also, exchanging the header field commands by an interface that provides the same capabilities while being less prone to manipulations could contribute to better security. Another line of future work is to extend the implementation of usage control for pages that have data about multiple users.

The aforementioned challenges only mark the beginning of what future work has to cover in order to achieve relevant usage control systems. In this context, issues concerning usage control at different layers of abstraction also need to be addressed. For example, when a browser caches a picture which is protected by usage control at the browser level, in a file on the hard disk, usage control should be in place for the operating system level. Similarly, there must be a way to inhibit screenshots.

From the time when the picture "leaves the browser," the receiving layer should be capable of continuing usage control enforcement. For this we need a data-centric approach for usage control that transcends the traditional approach based on events: the flow of data through the different levels of abstraction has to be detected, and enforcement of data usage must take place at all these levels. We are currently working on such a framework. There already is usage control enforcement with data flow tracking for the operating system [12] and the X11 levels [13], and now we are working towards connecting them with the browser. We have already extended BRUCE to (1) identify protected pictures with files cached on the hard disk and to (2) pass on the information about the protected cached data along with the associated policies to the enforcement mechanism at the operating system level where the cache file can be usage-controlled. In a similar vein, we are working on a connection with the enforcement mechanism at the X11 level to inhibit screenshots.

Finally, we see another research challenge in the context of heterogeneous cyber-physical systems. In this direction we are working on connecting a smart metering system to SCUTA where the social network fetches energy usage data from the smart meter according to policies specified by the smart energy user. In this case, we are interested in looking into the issues of policy conflicts and evolution of obligations across system boundaries. Another point of interest is the delegation of rights and duties in such a system where usage control is not centralized but distributed.

6. REFERENCES

[1] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. *Commun. ACM*, 49(9):39–44, 2006.

[2] J. Park and R. Sandhu. The UCON ABC usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.

[3] D. Povey. Optimistic security: a new access control paradigm. In *Proceedings of the 1999 workshop on New security paradigms*, NSPW '99, pages 40–45. ACM, 2000.

[4] R. Iannella (ed.). Open Digital Rights Language v1.1, 2008. `http://odrl.net/1.1/ODRL-11.pdf`.

[5] Multimedia framework (MPEG-21) – Part 5: Rights Expression Language, 2004. ISO/IEC standard 21000-5:2004.

[6] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise Privacy Authorization Language (EPAL 1.2). IBM Technical Report, 2003. `http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/`.

[7] Open Mobile Alliance. DRM Rights Expression Language V2.1, 2008. `http://www.openmobilealliance.org/Technical/release_program/drm_v2_1.aspx`.

[8] X. Zhang, J. Park, F. Parisi-Presicce, and R. Sandhu. A logical specification for usage control. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 1–10. ACM, 2004.

[9] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A policy language for distributed usage control. In *Proc. ESORICS*, pages 531–546, 2008.

[10] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proc. Workshop on Policies for Distributed Systems and Networks*, pages 18–39, 1995.

[11] W3C. The Platform for Privacy Preferences 1.1 (P3P1.1) Specification, 2005. `http://www.w3.org/TR/2005/WD-P3P11-20050104/`.

[12] M. Harvan and A. Pretschner. State-based Usage Control Enforcement with Data Flow Tracking using System Call Interposition. In *Proc. 3rd Intl. Conf. on Network and System Security*, pages 373–380, 2009.

[13] A. Pretschner, M. Buechler, M. Harvan, C. Schaefer, and T. Walter. Usage control enforcement with data flow tracking for x11. In *Proc. 5th Intl. Workshop on Security and Trust Management*, pages 124–137, 2009.

[14] M. Dam, B. Jacobs, A. Lundblad, and F. Piessens. Security monitor inlining for multithreaded java. In *Proc. ECOOP*, pages pp. 546–569, 2009.

[15] I. Ion, B. Dragovic, and B. Crispo. Extending the Java Virtual Machine to Enforce Fine-Grained Security Policies in Mobile Devices. In *Proc. Annual Computer Security Applications Conference*, pages 233–242. IEEE Computer Society, 2007.

[16] L. Desmet, W. Joosen, F. Massacci, K. Naliuka, P. Philippaerts, F. Piessens, and D. Vanoverberghe. The S3MS.NET Run Time Monitor: Tool Demonstration. *ENTCS*, 253(5):153–159, 2009.

[17] U. Erlingsson and F. Schneider. SASI enforcement of security policies: A retrospective. In *Proc. New Security Paradigms Workshop*, pages 87–95, 1999.

[18] B. Yee, D. Sehr, G. Dardyk, J. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *Proc IEEE Symposium on Security and Privacy*, pages 79–93, 2009.

[19] G. Gheorghe, S. Neuhaus, and B. Crispo. xESB: An Enterprise Service Bus for Access and Usage Control Policy Enforcement. In *Proc. Annual IFIP WG 11.11 International Conference on Trust Management*, 2010.

[20] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song. Dynamic spyware analysis. In *Proceedings of USENIX Annual Technical Conference*, June 2007.

[21] Adobe livecycle rights management es. `http://www.adobe.com/products/livecycle/rightsmanagement/indepth.html`, August 2010.

[22] Microsoft. Windows Rights Management Services. `http://www.microsoft.com/windowsserver2008/en/us/ad-rms-overview.aspx`, 2010.

[23] A. Pretschner, M. Hilty, F. Schutz, C. Schaefer, and T. Walter. Usage control enforcement: Present and future. *Security & Privacy, IEEE*, 6(4):44–53, 2008.

[24] P. Kumari. Requirements analysis for privacy in social networks. 8th International Workshop for Technical, Economic and Legal Aspects of Business Models for Virtual Goods, Namur, 2010.

[25] A. Acquisti and R. Gross. Imagined Communities: Awareness, Information Sharing, and Privacy on the Facebook. In *Privacy Enhancing Technologies Workshop (PET)*, Robinson College, Cambridge, United Kingdom, June 2006.

[26] C. Dwyer and S. Hiltz. Trust and privacy concern within social networking sites: A comparison of Facebook and MySpace. In *Proceedings of the Thirteenth Americas Conference on Information Systems*, Keystone, Colorado, USA, August 2007.

[27] J. Grimmelmann. Facebook and the Social Dynamics of Privacy. *Legal Studies, New York Law School*, 7:33–34, 2008/2009.

[28] L. Edwards and I. Brown. Data Control and Social Networking: Irreconcilable Ideas? In A. Matwyshyn, editor, *Harboring data: Information security, law, and the corporation*. Stanford University Press, 2009.

[29] K. Williams, A. Boyd, S. Densten, R. Chin, D. Diamond, and C. Morgenthaler. Social Networking Privacy Behaviors and Risks. *Seidenberg School of CSIS, Pace University, USA*, 2009.

[30] C. Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-Lee. Decentralization: The future of online social networking. In *W3C Workshop on the Future of Social Networking Position Papers*, 2009.

[31] The friend of a friend (foaf) project. `http://www.foaf-project.org/docs`, September 2010.

[32] A. Cutillo, R. Molva, and T. Strufe. Privacy preserving social networking through decentralization. In *WONS'09: Proceedings of the Sixth international conference on Wireless On-Demand Network Systems and Services*, pages 133–140. IEEE Press, 2009.

[33] Safebook publications. `http://www.safebook.us/publications.html`, September 2010.

[34] Peerson: Privacy-preserving p2p social networks. http://www.peerson.net, September 2010.

[35] D. Kalofonos, Z. Antoniou, F. Reynolds, M. Van-Kleek, J. Strauss, and P. Wisner. MyNet: a Platform for Secure P2P Personal and Social Networking Services. In *6th IEEE International Conference on Pervasive Computing and Communications (PERCOM'08)*, Hong-Kong, China, March 2008.

[36] Z. Antoniou and D. Kalofonos. User-Centered Design of a Secure P2P Personal and Social Networking Platform. In *3rd IASTED International Conference on Human-Computer Interaction (IASTED-HCI'08)*, Innsbruck, Austria, March 2008.

[37] K. Gollu, S. Saroiu, and A. Wolman. A Social Networking-Based Access Control Scheme for Personal Content. In *21st ACM Symposium on Operating Systems Principles (SOSP '07)*, Stevenson, Washington, October 2007.

[38] A. Tootoonchian, K. Gollu, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: Social Access Control for Web 2.0. In *First ACM SIGCOMM Workshop on Online Social Networks (WOSN)*, Seattle, WA, August 2008.

[39] B. Carminati, E. Ferrari, and A. Perego. Enforcing access control in web-based social networks. *ACM Trans. Inf. Syst. Secur.*, 13(1):1–38, 2009.

[40] Website of the Scramble! project. http://www.primelife.eu/results/opensource/39-scramble, September 2010.

[41] Website of the clique project. http://clique.primelife.eu/, September 2010.

[42] F. Beato, M. Kohlweiss, and K. Wouters. Enforcing access control in social networks. In *Proc. HotPets*, 2009.

[43] B. Berg and R. Leenes. Audience segregation in social network sites. In *Proceedings for SocialCom2010/PASSAT2010.IEEE*, pages 1111–1117, 2010.

[44] S. Kruk, S. Grzonkowski, A. Gzella, T. Woroniecki, and H. Choi. D-foaf: Distributed identity management with access rights delegation. In *Proc. Asian Semantic Web Conference 2006*, 2006.

[45] FOAFRealm. Foafrealm project site. http://www.foafrealm.org/, Jul 2010.

[46] Mozilla's add-on repository AMO. https://addons.mozilla.org/, July 2010.

[47] The tabulator extension. http://dig.csail.mit.edu/2007/tab, September 2010.

[48] J. Peschla. Data usage control for a web application: The client. Bachelor's thesis, University of Kaiserslautern, July 2010.

[49] Browser market share. http://marketshare.hitslink.com/browser-market-share.aspx?qprid=0&qptimeframe=M&qpsp=138&qpnp=1, September 2010.

[50] Phpizabi homepage. http://www.phpizabi.net/, September 2010.

[51] A. Pretschner, M. Hilty, D. Basin, C. Schaefer, and T. Walter. Mechanisms for Usage Control. In *Proc. ACM Symposium on Information, Computer & Communication Security*, pages 240–245, 2008.

[52] MASTER consortium. MASTER Deliverable 5.1.1: Security Enforcement Language. http://www.master-fp7.eu/, April 2010.

[53] Mozilla's plugin documentation. https://developer.mozilla.org/en/Plugins, May 2010.

APPENDIX

A. COMPONENTS OF BRUCE

In this section we give a brief overview of the major components of the policy enforcement point. They appear in the order in which they are called throughout usage control setup and enforcement.

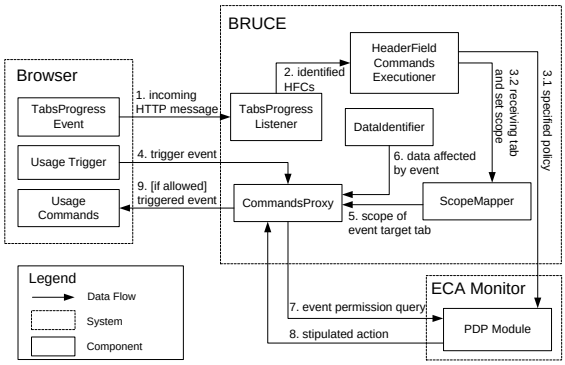

Figure 5: Architecture of BRUCE

TabsProgressListener is the component that is attached as event listener to the browser window so that it gets notified about all web-request and -response related events. This way it examines the HTTP headers of each incoming web-response for HFCs. Identified HFCs are forwarded to *HeaderFieldCommandsExecutioner* which carries them out. Besides, TabsProgressListener ensures that the browser cache cannot be accessed via Firefox itself.

HeaderFieldCommandsExecutioner carries out HFCs received from *TabsProgressListener*. Therefore it either communicates with the WBSN (authentication), the PDP (policy activation and revocation) or instructs *ScopeMapper* to update its mapping (scope assignment or release).

ScopeMapper is the component that maintains the associations between tabs and scopes.

CommandsProxy is the component that intercepts triggered usage events and decides about their progress. To determine whether an intercepted event has to be aborted, *CommandsProxy* queries *ScopeMapper* for the scope of the active tab and retrieves the event target from *DataIdentifier*. When a scope is set, then for each target element it queries the PDP with the parameter values for scope and abstract command. If for all queries the response is approval, the intercepted event is resumed, otherwise it is aborted. If no scope is set for the active tab it is either blocked and the event is aborted or the displayed page is not under usage control and the event is resumed.

DataIdentifier is an encapsulation of DOM element determination facilities. Given a usage related event and the target page it calculates the set of affected elements which is the target of the event as explained in Section 3.5.