Clark-Wilson Policies in ACP: Controlling Information Flow Between Solid Apps

Ellie Forsyth[1], Ross Horne[2]

[1] *Electronic and Electrical Engineering,, University of Strathclyde, Glasgow, UK*
[2] *Computer and Information Science, University of Strathclyde, Glasgow, UK*

Abstract

This paper explains how to avoid certain unintended information flows between apps connected to the same Solid pod. We draw attention to threats faced if security policies for Solid pods omit the identities of clients, resulting in confidential information intended for one app leaking to other apps. We also explain good practice usage of ACP for avoiding such insecure configurations and draw parallels with the famous Clark-Wilson policy model for enterprise security. We propose that trusted apps enforcing security policy models should be developed so that pod owners need not be policy experts to operate secure pods.

1. Introduction

The Solid protocol [1, 2] is a draft Web specification that aims to standardise how storage, called *Pods*, interact with apps. A key intention is that personal data processing, as governed by GDPR for instance, becomes more transparent. A normal usage pattern for Solid is that a user – the owner of a Pod – allows multiple apps to connect to the same Pod while also logging in to each app using the same identity. A consequence of this is that the same identity of a user is used to access resources stored in a Pod via different apps. Importantly, not all apps are run by mutually trusted organisations; indeed, in the worst case one app may be compromised, and hence measures must be in place to ensure that information does not leak from one app to another.

This paper builds on an observation in related work [3] that security policies, permitted by Solid, that administer access control solely based on the identity of the user are not secure. In particular, we analyse under what conditions the policy of a Solid Pod can be configured such that data remains secure even when multiple applications have access to the Pod, as described above. We will analyse the problem in terms of information flows that are permitted by policies defined using two draft access control mechanisms: Web Access Control (WAC) [4, 5] and Access Control Policy (ACP) [6].

WAC was proposed in the early days of the Solid project, as a vocabulary for access control rules specifying when users, and groups, identified using URIs known as WebIDs [7], can access a given resource in a Pod. These access permissions are stored within an Access Control List (ACL) document which is referred to by a Pod when access to a

SoSy'24: Privacy @ the Solid Symposium, May 2-3, 2024, Leuven, Belgium

ellie.forsyth.2020@uni.strath.ac.uk (E. Forsyth); ross.horne@strath.ac.uk (R. Horne)

resource is requested. Beyond Solid, WAC is used by the W3C to specify access control for some of their own sites. On the other hand, ACP was introduced more recently to set more specific permissions in the Solid project than WAC. In particular, ACP can set the *client* (the Solid app) and *issuer* (identity providers).

This work we explain pitfalls related to designing security policies using WAC and ACP. We (1) explain threat models and attack vectors relevant to security policies for Solid; (2) demonstrate key attack vectors that exist in Solid pods that employ WAC using the standard implementation offered by the Community Solid Server; (3) devise measures to address attack vectors, leveraging ACP, and explain why they are appropriate for the developers of WAC and ACP; (4) connect the innovation to established security policies, notably Clark-Wilson, which goes beyond objects (data) and subjects (users) to include processes much like the role of apps in Solid. The paper also explains that security apps could be developed to ensure that Pods are configured with policies that adhere to security policy models.

2. ACP in relation to the Clark-Wilson Security Policy Model

In this section, we explain how a key difference between WAC and ACP can be explained in terms of the famous Clark-Wilson security policy model that was designed for managing the integrity of data used by enterprise processes [8]. The section then goes on to explain some attack vectors that arrise if certain features appearing in ACP that also appear in the Clark-Wilson security policy model are not used correctly.

2.1. ACP compared to WAC

The primary distinction between policies designed using Access Control Policy (ACP) rather than Web Access Control (WAC) is their ability to manage access to resources with respect to clients and issuers, not only users or groups of users. To understand this see the WAC policy in Fig. 1, which we will see does not fully prevent undesirable information flows between apps due to its restricted policy framework, which solely allows the specification of agents using `acl:agent`. This particular authorisation rule indicates that an agent identified by a WebID can read and write, to any resource in the namespace indicated by `acl:default`.

```
@prefix acl: <http://www.w3.org/ns/auth/acl#>

<#exampleOfWAC>
  a acl:Authorisation;
  acl:agent <https://solidweb.me/Ellie-s-Pod/profile/card#me>;
  acl:default <https://solidweb.me/Ellie-s-Pod/Resource1/>;
  acl:mode acl:Read, acl:Write.
```

Figure 1: WAC Policy example

In contrast, ACP provides more context information, enabling the precise specification of agents through `acp:agent`, apps via `acp:client`, and Identity Providers (IDPs) using `acp:issuer`, depicted in Figure 2. The rest of the paper explains the security implications of these features for Solid.

```
@prefix acl: <http://www.w3.org/ns/auth/acl#>
@prefix acp: <http://www.w3.org/ns/solid/acp#>

<#exampleOfACP_1>
  a acp:AccessControlResource;
  acp:resource <https://solidweb.me/Ellie-s-Pod/Resource1/>;
  acp:accessControl <#ownerAccess1>;
  acp:memberAccessControl <#ownerAccess1>.

<#ownerAccess1>
  a acp:AccessControl;
  acp:apply [
    a acp:Policy;
    acp:allow acl:Read, acl:Write;
    acp:allOf [
        a acp:Matcher;
        acp:client <https://solidweb.me/ClientPod/app1/clientid.jsonld>;
        acp:agent <https://solidweb.me/Ellie-s-Pod/profile/card#me>;
        acp:issuer <https://solidweb.me/> ] ].
```

Figure 2: ACP Policy example restricting *resource1* such that the subject can only access the pod via *app1*. A single trusted issuer is also named.

2.2. Clark-Wilson in ACP

The Clark-Wilson model and `acp:client` in ACP have the common feature that the access of information via specified clients—the apps accessing resources within a Pod—can be restricted. Within the Clark-Wilson model, a fundamental component sometimes called the *process* (or *Transformation Procedure* c.f. Clark & Wilson) acts as an intermediary between the users and the resources, as with the applications in the Solid framework.

In Clark-Wilson access is governed by "permissions" that specify the combinations of agent, client and resource for which access is permitted. Thus, in ensuring that only authorised clients are granted access, `acp:client` is aligned with the Clark-Wilson security policy model. ACP does further extend Clark-Wilson, for instance, by allowing also the issuer which vouches for the identity of the user to be specified in the policy.

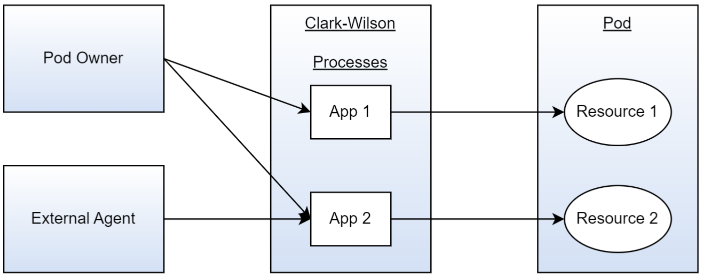

Figure 3: Schematic example of an ACP policy aligning with the Clark-Wilson security policy model.

Illustrated in Fig. 3 is a conceptual representation of an example scenario for a Solid Pod, describing access control mechanisms available to Pod Owners. This framework empowers Pod Owners to meticulously tailor access privileges, allocating permissions to specific resources through designated applications. Furthermore, Pod Owners can permit more limited access to other agents, thereby enabling controlled interaction with designated resources through authorised applications. In Fig. 3 a Pod Owner limits access to a resource *resource1* when *app1* is used and likewise for *resource2* and *app2*. In addition, they have granted more restricted access to an external agent to access *resource2* only through *app2*.

The ACP policy governing *resource1* in Fig. 3 was already provided in Fig. 2. That ACP policy can be extended with the policy in governing *resource2* provided in Fig. 4. This ACP policy governs how two agents can access a resource referred to as *resource2* in Fig. 3, as described above. This is a realistic scenario since it may be the case that the owner wishes to share more widely information in *app2* while having the reassurance that information in resources accessed from *app1* does not flow to *app2*.

```
<#exampleOfACP_2>
  a acp:AccessControlResource;
  acp:resource <https://solidweb.me/Ellie-s-Pod/Resource2/>;
  acp:accessControl <#ownerAccess2>, <#externalAgent>;
  acp:memberAccessControl <#ownerAccess2>, <#externalAgent>.

<#ownerAccess2>
  a acp:AccessControl;
  acp:apply [
    a acp:Policy;
    acp:allow acl:Read, acl:Write;
    acp:allOf [
           a acp:Matcher;
           acp:client <https://solidweb.me/ClientPod/app2/clientid.jsonld>;
           acp:agent <https://solidweb.me/Ellie-s-Pod/profile/card#me>;
           acp:issuer <https://solidweb.me/> ] ].

<#externalAgent>
  a acp:AccessControl;
  acp:apply [
    a acp:Policy;
    acp:allow acl:Read;
    acp:allOf [
           a acp:Matcher;
           acp:client <https://solidweb.me/ClientPod/app2/clientid.jsonld>;
           acp:agent <https://solidweb.me/EF-Pod/profile/card#me>;
           acp:issuer <https://solidweb.me/> ] ].
```

Figure 4: ACP Policies with `acp:client` preventing an unwanted information flow between *app2* and *app1* from Fig. 2.

The policy in Fig. 3 inspired by Clark-Wilson we have just shown can be implemented in ACP, but is not supported by WAC. While ACP gives us a vocabulary for defining policies, its secure usage is contingent upon the proper configuration of ACP policies, particularly with respect to `acp:client` directives, as we elaborate on next.

2.3. Attack vectors when `acp:client` is omitted

We explain an attack vector enabled when WAC is used or an ACP policy is configured insecurely. This proves that policies must stipulate `acp:client` unless data is sanitised.

Suppose a user intends for two resources to be exclusively accessed by two different apps, as depicted in Figure 3. Failure to specify that a client (app) should exclusively access resource X results in permitting both apps access to X. The steps that a compromised client may take to access resources not intended for them are illustrated by the message sequence chart (MSC) depicted in Figure 5. The MSC not only clearly displays the attack vector but also the interactions a user can have with their Pod. We assume (1) that an honest app has created a resource. An app compromised by the attacker then requests access to a resource within a user's Pod via an honest IdP (2). The IdP, subsequently, issues an access token named a DPoP token and this is sent back to the requesting app (3). The app can now spend this DPoP token to access the user's Pod resource (4); where the request is checked by the authorisation server for the pod. The authorisation server will verify the app by checking the resource ACP policy, but (5) since the policy does not state the client (app), access is granted to the app. Therefore, (6) a compromised app is accessing a resource within a Pod maliciously.

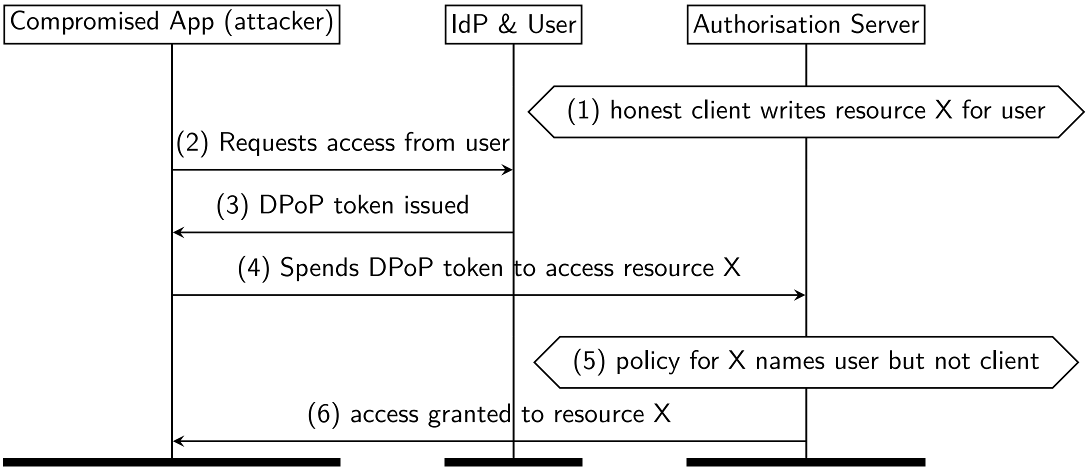
Compromised App (attacker)
IdP & User
Authorisation Server
(1) honest client writes resource X for user
(2) Requests access from user
(3) DPoP token issued
(4) Spends DPoP token to access resource X
(5) policy for X names user but not client
(6) access granted to resource X

Figure 5: An attack vector valid when the client is not named in the policy.

Using `acp:client` the attack vector in Figure 5 is preventable. Since the ACP policy is able to state within the permissions which app can access a given resource within a Pod, the attack vector would be blocked by the authorisation server at step (5). The use of `acp:client` in policies is of course contingent upon client IDs being associated with Solid applications. Consequently, apps with dynamic client IDs may only be entrusted with sanitised data that may flow freely between apps.

2.4. Technicalities of client identity

The effective use of `acp:client` is, of course, contingent upon the existence of a usable client ID to specify within an ACP policy. Many Solid apps currently generate their client IDs using Dynamic Client Registration (DCR), a protocol where an app is registered with the authorisation server by sending an HTTP POST request to the client registration endpoint [9]. Subsequently, the authorisation server assigns a opaque random client ID for the given session. The use of DCR in many apps poses challenge for setting appropriate ACP policies, as the client ID is generated after the security policy has been established. Consequently, the acp:client predicate cannot reference a client ID generated using DCR. Therefore we recommend that apps are converted from using DCR client ID to a usable, stable one. The process for doing this varies from app to app.

It may be possible to have a security policy model where DCR clients can access certain data. However, it should be known that such data should be sanitised in the sense that it is fit for consumption by any app in the ecosystem that a user connecting to the pod may use. As we have seen previously in this paper, information without a policy that identifies clients, may flow freely through the system. This potentially means that private personal data should not be handled by such apps, or there is a risk of violating regulation such as GDPR due to exposure to data breaches involving a single compromised app in the ecosystem. Since personal data is a primary use-case for Solid, one must ask carefully the question of what counts as sanitised data.

3. Beyond Clark-Wilson: attacks when `acp:issuer` omitted

Having explained the importance of `acp:client`, we now explain another attack vector made possible by omitting `acp:issuer` from ACP policies. By specifying authorised Identity Providers (IdPs), the `acp:issuer` predicate restricts authentication and access privileges solely to designated IdPs trusted by the pod. Without stipulating `acp:issuer` threat actors could exploit any other compromised IdPs listed in a WebID to issue fraudulent tokens, thereby gaining unauthorised access to Pod resources without the knowledge or consent of owner of the WebID.

Consider the MSC diagram in Figure 6. In this attack vector, a user does not need to request access nor does an honest client (app). The user has however listed a compromised IdP in their WebID that would not be acceptable for accessing the pod. Here we see that a compromised IdP can simply generate a DPoP token themselves and spend it to access a resource within the Pod, with neither the user's knowledge nor the use of any app listed in the ACP policy. Once again, the authorisation server checks the ACP policy and, assuming the issuer (IDP) has not been stated within the policy beforehand, access is given to the resource.

To block this attack vector, the predicate `acp:issuer` can be added into the policy to prevent this attack vector from occurring with an arbitrary IdP. This way a pod can list explicitly which IdPs it trusts sufficiently to serve as a root of trust, for access to the pod. That is, the IdP is trusted not to exploit it position and also to put in place adequate measures to avoid becoming compromised, e.g., through cyber attacks on the

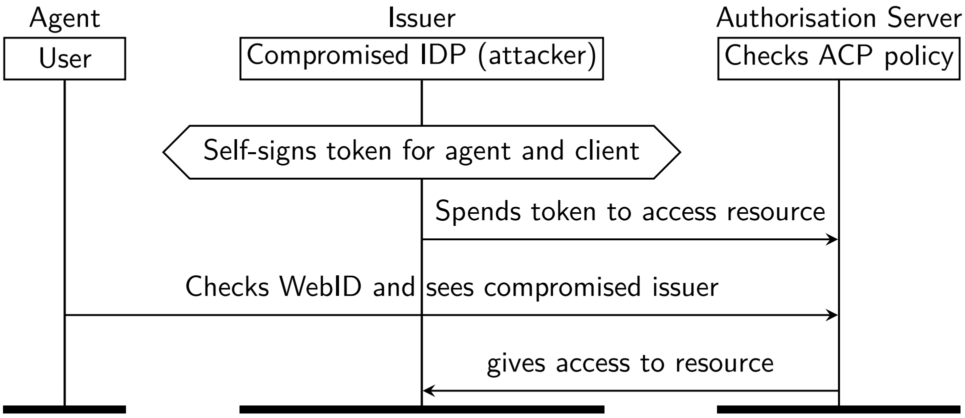
Agent
User
Issuer
Compromised IDP (attacker)
Authorisation Server
Checks ACP policy
Self-signs token for agent and client
Spends token to access resource
Checks WebID and sees compromised issuer
gives access to resource

Figure 6: Attack vector if compromised issuer listed in WebID of user and authorisation server does not restrict to trusted issuers. Vector allows attacker to access resources intended for a given user and client.

IdP itself. Note that issuers were not part of the Clark-Wilson security policy model.

4. Towards trusted security policy apps & Conclusion

We have seen, given the attack vectors in Figures 5 and 6, that using ACP is in itself not sufficient to ensure that the policy set for a Pod is guaranteeing reasonable confidentiality and integrity properties. Pod owners really need to set a policy that adheres to an appropriate security policy model, and that requires expertise that almost certainly the pod owner will not possess and quite possibly nor will most app developers. As a way to address this challenge, we develop a prototype security app as a proof of concept for a special highly-trusted app that sets a security policy on behalf of the pod owner.[1] The security app must have the privileges to control the policy, by setting a policy with the access mode `acl:Control` for any part of the pod that the app is intended to manage and naming explicitly the security app using `acp:client`. Indeed, it makes sense that when the pod is created the security app that is permitted to manage the policy of the pod is coupled with the pod by setting an appropriate policy. An organisation hosting such a security app must therefore be trusted above all other apps used – perhaps the pod provider that is anyway entrusted to properly manage the data in the pod, or an organisation trusted for its security practices.

Vulnerabilities discussed in Section 2 resulting from the absence of `acp:client` in ACP policies are critical since they may result in confidential personal data leaking to compromised apps. This paper also points out that this is a problem as old as enterprise computer security by drawing parallels with the Clark-Wilson security policy model.

[1] The app is provided in a repository: https://github.com/eforsyth1/SecureSolidApp

For a more recent example of a security policy model where more than the user it taken into account, the Android security policy model is sensitive to the user, platform, and developer as well as the app when determining whether access is permitted [10]. Rich policies are inevitable when many organisations are responsible for a decentralised ecosystem. For example, in the case of Solid, we in addition pointed out, in Section 3, that the IdP has disproportionate power to access resources, if compromised, and hence we recommend that policies are made aware of which IdPs should be trusted.

A key challenge is that pod owners do not have the relevant expertise to manually set and check their own policies. Priority future work that this paper exposes is the need to agree on a precisely defined security policy model for Solid. A security policy model goes beyond simply using ACP in the sense that it should enforce correct usage of ACP to adhere to confidentiality or integrity goals. An example of such a goal is to avoid unintended information flows between apps controlled by different organisations, as discussed in this work in a particular instance. Drawing inspiration from Clark-Wilson is just a starting point towards this aim.

References

[1] A. Sambra, et al., Solid: A platform for decentralized social applications based on linked data, 2016. MIT CSAIL & Qatar Computing Research Institute.

[2] S. Capadisli, T. Berners-Lee, R. Verborgh, K. Kjernsmo, Solid protocol, 2023. URL: https://solidproject.org/ED/protocol.

[3] C. Esposito, R. Horne, L. Robaldo, B. Buelens, E. Goesaert, Assessing the Solid protocol in relation to security and privacy obligations, Information 14 (2023) 411.

[4] O. Sacco, A. Passant, S. Decker, An access control framework for the web of data, in: IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, 2011, pp. 456–463. doi:10.1109/TrustCom.2011.59.

[5] S. Capadisli, T. Berners-Lee, Web Access Control, 2022. URL: https://solidproject.org/TR/wac.

[6] M. Bosquet, Access control policy (ACP), 2022. URL: https://solid.github.io/authorization-panel/acp-specification/.

[7] S. Tramp, H. Story, A. Sambra, P. Frischmuth, M. Martin, S. Auer, Extending the WebID protocol with access delegation, in: Proceedings of the Third International Workshop on Consuming Linked Data (COLD2012), CEUR-WS. org, 2012.

[8] D. D. Clark, D. R. Wilson, A comparison of commercial and military computer security policies, in: IEEE Symposium on Security and Privacy, 1987, pp. 184–184. doi:10.1109/SP.1987.10001.

[9] J. Richer, M. Jones, J. Bradley, M. Machulak, P. Hunt, Oauth 2.0 dynamic client registration protocol, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7591.

[10] R. Mayrhofer, J. V. Stoep, C. Brubaker, N. Kralevich, The Android platform security model, ACM Trans. Priv. Secur. 24 (2021). doi:10.1145/3448609.