The WAC Machine

Decentralised, Provably Secure and Consistent
Web Access Control

SPEC

whack WAC | conceptual framework that, like all maths models, is precise, concise and can be reasoned about unambiguously, so we can all draw the same conclusions.

FORMAL
- unambiguous → only one possible interpretation
- consistent → no contradictions
- machine-checked → proved correct by compiler
- executable → can run as a computer program

(see)

WAC
- enhancement → make unambiguous & consistent
- extensions
  - → multiple eval modes
  - → policy DSL
  - → data sharing → fun/hom encryption

LIB
- web assembly
- digitally signed
- test suite → . test gen
  - statistical prop-based

CERT PROTO
- zero machine →
  - proved correct
  - has DID w/ source code hash → Nix for REPRODUCIBILITY
  - certified by theorem prover
- certification
  - → certified machine challenges machine to certify by running test suite and checking results against itself (b/c certified machine output is correct, well probabilistically speaking)
  - → publish DID w/ source code hash
  - → add DID doc to block chain → REUSE/LEVERAGE open call #1 software

AMBIGUITY

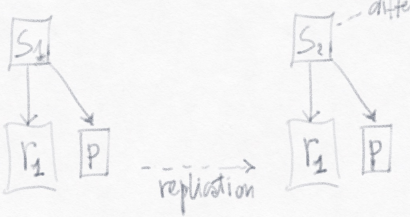
- ambiguous spec (policy inherit)
- two services, each interprets spec in own way
- user migrates data + policy from S_1 to S_2
- security jeopardised

Ref: whoc-voc eval model

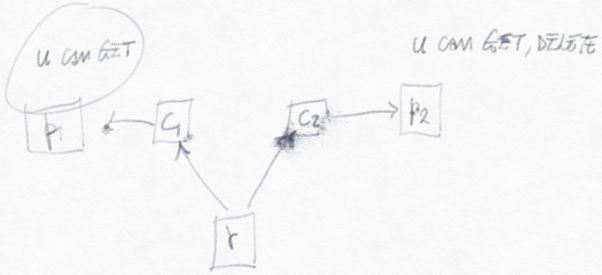
subtle & much more less obvious examples can be built by taking also into account res/policy hierarchies

example

can't come up w/ good example not involving inheritance



this could be an example



DELETE /c1/r
Auth: (u)

S_1 → inherit from C_1 ⇒ deny

S_2 → inherit from C_2 ⇒ allow

(1) - Big corps acquire your data
- You've got NO CONTROL over how they use/share it

(2) CAN INDIVIDUALS REGAIN CONTROL OF THEIR OWN ONLINE DATA?

* SOLID →
- enable true data ownership
- improve privacy
- decentralised apps controlled by individuals rather than large corporations

core spec ↓ WAC

User-Generated Content

https://en.wikipedia.org/wiki/User-generated_content

User-generated content (**UGC**), alternatively known as **user-created content** (**UCC**), is any form of content, such as images, videos, text, testimonials, and audio, that has been posted by users on online platforms such as social media, discussion forums and wikis. It is a product consumers create to disseminate information about online products or the firms that market them.[1][2]

User-generated content is used for a wide range of applications, including problem processing, news, entertainment, customer engagement, advertising, gossip, research and many more. It is an example of the democratization of content production and the flattening of traditional media hierarchies.

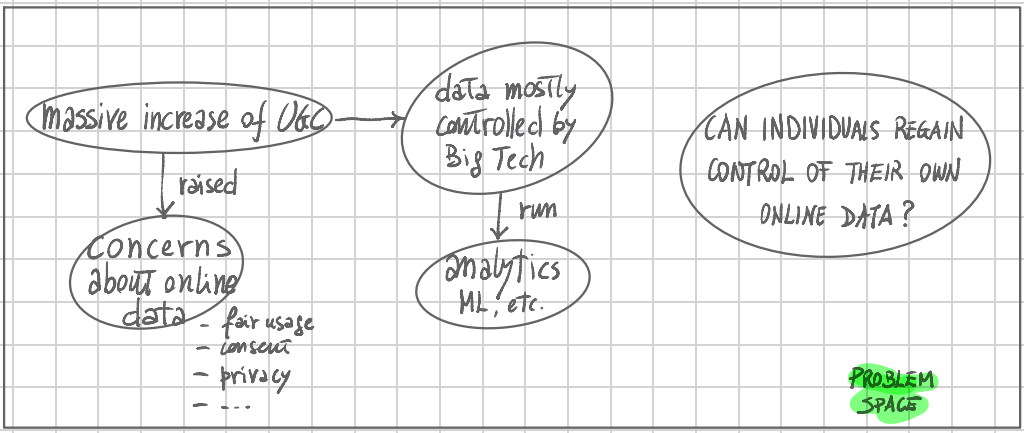
By 2020 businesses are increasingly leveraging User Generated Content to promote their products, as it is seen as a cost effective and authentic way to grow a brand's image and sales. Due to new media and technology affordances, such as low cost and low barriers to entry, the Internet is an easy platform to create and dispense user-generated content,[7] allowing the dissemination of information at a rapid pace in the wake of an event.[8]

The advent of user-generated content marked a shift among media organizations from creating online content to providing facilities for amateurs to publish their own content.[2]

Conversational or two-way media is a key characteristic of so-called Web 2.0 which encourages the publishing of one's own content and commenting on other people's content.

a concentration phenomenon is occurring globally giving the dominance to a few online platforms that become popular for some unique features they provide, most commonly for the added privacy they offer users through disappearing messages or end-to-end encryption (e.g. WhatsApp, Snapchat, Signal, and Telegram), but they have tended to occupy niches and to facilitate the exchanges of information that remain rather invisible to larger audiences.[18]

OUTLINE



BUILD A DECENTRALISED

individuals can trust programs to flawlessly enforce their access control policies

leverage Web standards & decentralized id s/w developed by OC #1

* author policy in lang close to plain Eng
* interactive env to dev & test policies (REPL)
* trusted exec env to enforce policies according to spec
* protocol to extend n/w of trusted envs
* host data/policies on any trusted node (portability)

for this to happen, every node must be proven to do same thing as zero machine

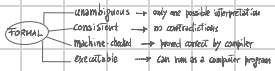
FORMAL SPEC

(1) initial alg model → our understanding
    - functional core
    - ETCS
(2) model validation → on GH w/ UAC's authors
(3) extension → - make more flexible, esp. eval
    - make provisions for our policy eval
    - hooks for fun/hom encryption
(4) executable spec → - encoding in Haskell / Idris
    - machine-checked proof of correctness

FORMAL SPEC

this goes in 3.2 proposed sol.

this goes in 3.1 concept & objectives



The formalisation and extension of the WAC specification will be attained through subsequent refinement phases. In the tradition of functorial semantics[1,2], a preliminary algebraic model will provide a concise, consistent and unambiguous interpretation of WAC in terms of basic structures and structure-preserving transformations in the elementary theory of the categories of sets[3,4]. We will then engage with the WAC authors on GitHub to validate and refine the model. Following that, we will devise a specification extension to make the evaluation model more flexible so as to allow the evaluation of access control policies expressed in terms of predicates on a generic set-i.e., functions from a set X to the Boolean algebra {0, 1}. At the same time, we will investigate another specification extension to accomodate data sharing through functional and homomorphic encryption techniques. Finally, we will encode the formal specification in an advanced programming language (either Haskell[*] or Idris[*]) which we will then leverage to produce a machine-checked proof of correctness of the resulting computer program. The resulting (correct!) executable specification will be submitted to the Solid project for consideration as a future version of WAC.

[3] F. W. Lawvere. An elementary theory of the category of sets. Proceedings of the National Academy of Sciences of the U.S.A., 52:1506–1511, 1964.

[4] Rethinking set theory, Leinster

[1] F. W. Lawvere, *Functorial Semantics of Algebraic Theories*, Proc. Nat. Acad. Sci. ***50*** (1963) pp. 869–872.

[2] Filippo Bonchi, Dusko Pavlovic, Pawel Sobocinski, *Functorial Semantics for Relational Theories*, (arXiv:1711.08699)

```
data Tree a = Node a [Tree a]

path :: (a -> Bool) -> Tree a -> [a]
path p = collect []
  where
   collect xs (Node a ts) | p a       = a:xs
                          | otherwise = concatMap (collect (a:xs)) ts

t = Node 0
      [ Node 1
          [ Node 11 []
          , Node 12 []
          ]
      , Node 2
          [ Node 21 []
          , Node 22 []
          ]
      ]
r = path (==22) t

main :: IO ()
main = do
 print r
```

=> [22, 2, 0]

the path from the root to the node satisfying the predicate p. It can easily be defined recursively on a canonical multi-way tree structure with the help of the standard list processing function concatMap. It is just as easy to prove path correct by induction.

To translate a generic accRes of functor into e.g. Haskell code, observe that such a functor can be defined in terms of the function path that finds ...

Below is the corresponding Haskell code.