DATA TRANSFER INITIATIVE

User Data Portability Threat Model

A reference model and examination of threats to privacy and harmful content, that can take place in a server-to-server transfer of user data as requested by the user.

Lisa Dusseault

Table of Contents

Introduction **2**
Quick Overview of Data Portability **3**
Requirement Details **5**
1. Person-centered 5
2. One-time 6
3. Known Data Types 7
Reference Architecture of Vetted Services **8**
Source-initiated variant 10
Destination-Initiated Variant 11
Sourcing the threat categories **12**
Using a secure transport helps with threats **14**
Disclosure or loss of privacy 14
Indirect Loss of Privacy 14
Tampering 15
Vetting source and destination help **15**
Non-Repudiation 16
Denial of service (DOS) 17
Elevation of privilege 17
Non-compliance 18
Harmful Content **20**
Bulk Submission 21
Content Harmful to People 23
Content Harmful to Services 24
E2EE and content safety 24
Spoofing **25**
New avenues for phishing attacks 26
Vetting services 27
Permission and Access challenges **27**
Who initiates 29
Post-transfer permission and access work 31
Summary 32

Introduction

Data portability describes functionality for many applications; a user's photos, playlists, tasks, notes, shopping and search history, and social media posts are all types of data the user might want to copy or move to a different service. It's framed as a consumer right in laws (e.g. EU, UK and Canada) and other places. As services build and enhance features to provide this data portability right, we need to make sure these services know to protect users against certain kinds of threats.

In computer science, the ideal is to have a threat model for a well-scoped problem before trying to pick solutions. Data portability is hardly a well-scoped problem, though - not all applications are covered by the same architecture and functionality. In searching for solutions and discussing threats, it's all too easy to get lost in hypotheticals and lose track of how decisions fit together - like trying to solve two jigsaw puzzles at once without committing to fit any pieces together.

What we can do is pick one generic solution, see how many use cases it covers, and consider what threats it is vulnerable to. We need to list some use cases not only to motivate the architecture, but also because some threats depend heavily on use cases. We will need at least some high-level architecture for the solution, to help us see which threats are hypothetical (some attacks can only work with an architecture we're not likely to choose) and which threats are real.

In this paper (originally three blog posts at dtinit.org/blog), common use cases help us pick a useful reference architecture, and the reference architecture allows us to develop a threat model for data portability. The resulting threat model won't apply directly to every architecture or solution proposed, but it will be easier to copy much of the same work for a different architecture. The list of threats provides a useful checklist and guide for scoping standards and development work.

What about other use cases or other architectures? Many readers will be able to think of exceptions and other use cases. That's OK. The reference architecture is not intended to exclude other architectures or prevent extensions that solve additional use cases. It will still give us the ability to compare different use cases and different architectures in the face of these threats.

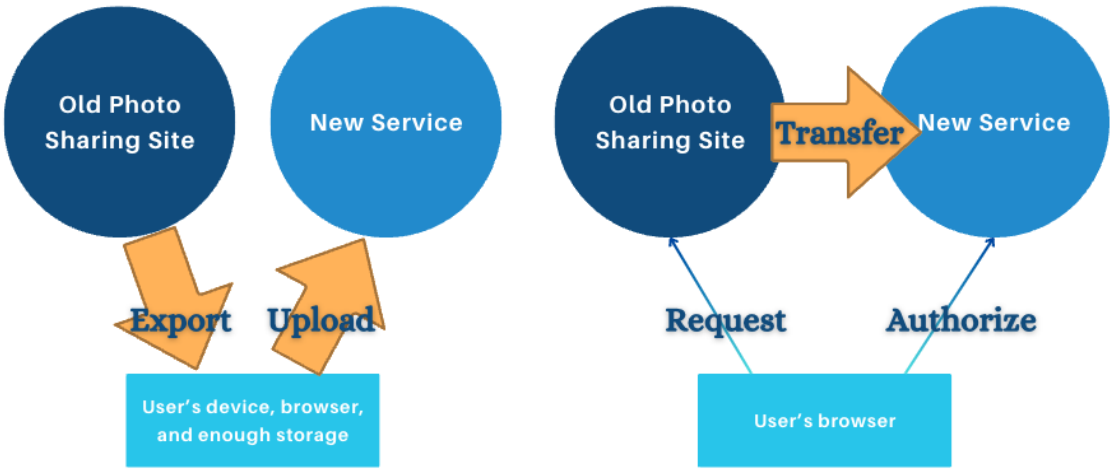
Quick Overview of Data Portability

First I want to make sure we're on the same basic page, before describing the details of a rhinoceros when you think I'm about to describe an elephant.

In one sentence, the core usage model can be summarized:

> *As a person with my data in a source service, I want to move it to a destination service by having the services handle transferring the data and making sure it lands correctly at the destination.*

A key assumption here is that data travels directly between services, as opposed to the Export/Import approach where data is stored on a personal computer or online storage in between services.



Different interactions in Export/Import approach vs. server-to-server transfer

Having services exchange data directly is key to practically exercising a right to data portability in cases where the data is high volume, requiring high bandwidth and storage capabilities, or the data is complicated enough to require agreement between the source and destination on a schema in order to transfer well. Photos (many people with mobile phones have thousands of photos in the cloud) and music playlists (with

the ability to actually play the playlist in a new service) exemplify these two cases pretty well.

The core interactions for server-to-server data portability look something like this:

- I must authorize a source service to access and send some data.
- I must authorize a destination service to accept that data on my behalf.
- After both authorizations are made, direct server-to-server transfer of data can begin.

This high-level description is missing a key decision, however. Does the authorization of the source service get sent to the destination so it can ask for data? Or does the authorization of the destination get sent to the source so that it can start sending? This is a question of who initiates the authorization request and who receives that authorization result.

Source-initiated architecture	**Destination-initiated architecture**
On a data-transfer Web page owned by the source, the user will select a destination (e.g. from a drop-down list) and the source then requests the user to authorize the transfer to the destination. If successful, the destination provides the source with a way to prove that authorization to send data.	On a data-transfer Web page owned by the destination, the user will select a data source (e.g. from a drop-down list) and the destination then requests the user to authorize getting the data from the source. If successful, the source provides the destination with a way to prove that authorization to get data.

As we'll see later, and you can kind of tell from the mirrored language, these actually have a lot in common. Either system would work pretty well for a lot of use cases, and we don't need to decide now.

Requirement Details

Now that we have the basic picture at a very high level, we need to characterize some real use cases in enough detail to make sure that the reference architecture works. In any interesting problem domain, there's always quite a population of simple needs, complicated situations, and problems in between. When we draw a line and say which problems can be solved simply now, we're often saying that the problems outside the line can be tackled later – and often by the same architecture with extra features.

I think the key things that characterize the simpler set of problems is that they are variations on a request from an individual person for a one-time data transfer of a known type of data. Let's unpack these three pieces.

1. Person-centered

The core problem to solve is when a person makes a request involving that person's own data.

When DTI talks about data portability we usually assume consumer and individual use cases. This arises in part from EU and US state regulations that emphasize individual data rights. Note that this centers not only the person but also ***their data***.

The following use cases can be handled later or by different solutions:

- B2B use cases - any time data transfer occurs from business to business, without involvement or request of a user.
- B2B2B use cases, where the entity that wants to transfer data is itself a business. Data that is business oriented (retail data, business finance data, Human Resources data) is subject to different threats.
- Data that does not belong to the user might be transferable but has different privacy characteristics. For example, users might wish to select photos on a stock photo service and copy them somewhere else, but that has different characteristics than transferring my own private photos.

- Community data, even if it was contributed by the user, also has different privacy characteristics. For example, many knitters on ravelry.com have contributed images and pattern information to a knitting pattern database, and those are now a communal resource.

What does remain in the core usage model includes both data *about* the user and data that the user *contributed*. Those often overlap and both often require privacy, so we can conclude that privacy is a requirement of the core usage model and threats to privacy cannot be waved away.

Another facet of person-centered data portability is that we can start with the use cases where the person whose data is being transferred requested the transfer themselves. There are studies about the privacy and security considerations when two services exchange a user's private data (Weiss, 2008), and those considerations are very different when the user didn't request that exchange.

2. One-time

The one-time nature of the simplest data portability use cases is also an important simplification. It's clear that there are some use cases that are satisfied with one-time data transfer, and while real-time ongoing data transfer functionality has fascinating and important uses, it is definitely more complicated. Ongoing transfer requires additional features (like token lifetime limitations, token renewal, canceling, notifications) and those features introduce additional security considerations.

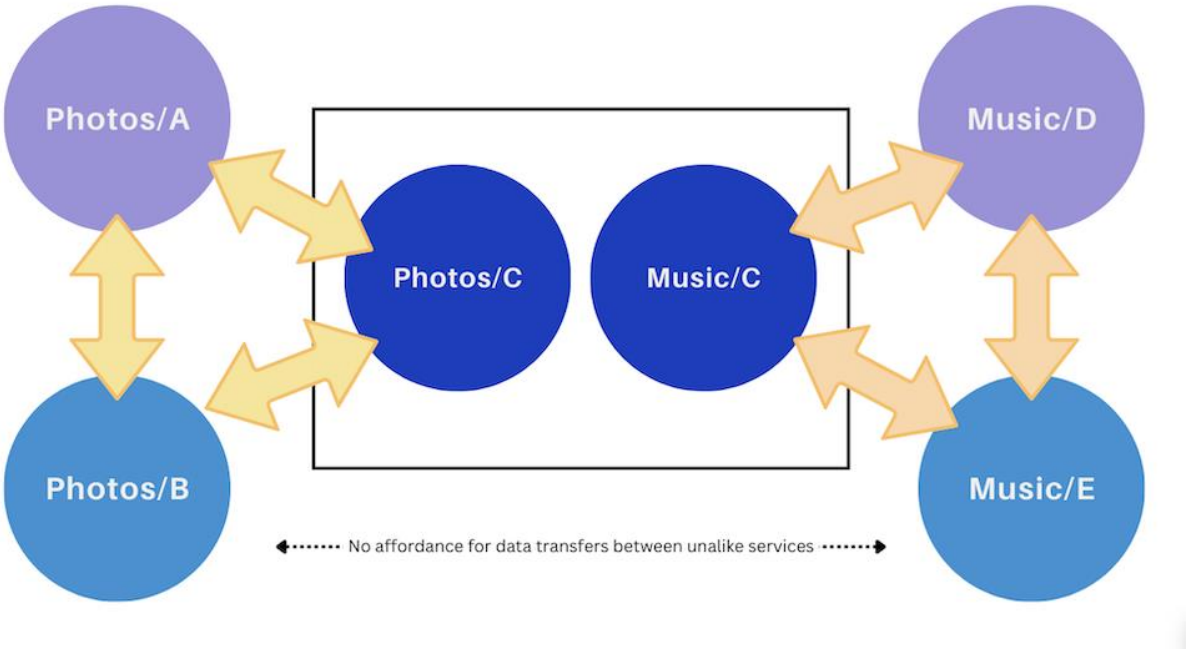
Repeated asynchronous data transfer use cases can also be deferred. As soon as we start thinking about repeated copying we will want to talk about features that are commonly part of backup or replication solutions, as well as token lifetimes and cancellation features as with real-time data transfer.

That's not to say that the user can only request data transfer one time and never gets any other chances! It just means that our core architecture can focus on one transfer only, without relationship to past or future transfer requests – definitely a simplifying assumption.

Finally, we can solve the case where users can be notified about transfer status *_in-band, _meaning* they can learn about the completed transfer in the same place (likely a Web page, but possibly an app) where they kicked it off. Out-of-band notifications may be a complication to the threat model.

3. Known Data Types

One of the most obvious and simple use cases is when a user asks to transfer their personal photo collection from one service to another photo service. The same architecture can probably be used when the user asks to transfer their music playlists from one music service to another music service. If we use the same architecture for both, does that mean we're going to have a photo server start to send photos to music streaming servers, or that a music streaming server starts to send music playlists to newsletter hosting sites? No - that's the difference between *general-purpose* and *universal* solutions. A general-purpose solution can be re-used, without requiring each service to handle the universe of possible data types.



Even though C hosts both music and photos, it only transfers photos to photo services

This will be a simplifying choice to both our architecture and to mitigating threats. Since a photo destination knows to expect photo data, it can check to see if the data is legitimately photos. The destination can reject data that it would not normally host. Nobody is going to require photo services to accept virus code packages and host them where only photos are supposed to be hosted.

We also don't have to worry about the destination receiving a never-ending stream of binary data in a denial-of-service attack, because we allow those services to cut off the stream when data gets too large for the data type in question. If we didn't know the data type, these kinds of security choices would be harder. Maximum sizes for an ActivityPub post, for an image, and for a long-form video, are significantly different numbers. A system for hosting posts with ActivityPub may place performance limits at a very different point than YouTube does.

Reference Architecture of Vetted Services

To solve this core usage model, we can now define a pretty obvious architecture using common Internet standards and components.

1. Both source and destination services have ways of authenticating the user account and starting authenticated sessions, and they will continue to use those.
2. Oauth will be used to check for user authorization. Either the source will initiate transfer and trigger Oauth for the user to authorize the destination, OR the destination will initiate transfer and trigger OAuth for the user to authorize the source.
3. Services sending and receiving personal data will use transport security to encrypt all traffic involved.
4. Hosts will be identified by domain names and domain certificates.
5. Destinations can refuse to accept data that seems to be the wrong type, harmful, exceeds quotas or threatens compliance in any way.

6. Destinations that normally host private data will default to incoming data being private.

These architectural components are all pretty common! Most platforms with private personal content already use most of these tools, and certainly host identification and transport encryption are very common. These services already authenticate users and apply content filtering. Most probably have OAuth somewhere in their service architecture, and if not, it's a mature technology with plenty of libraries available.

To these obvious and well-understood choices, let us add a more surprising one: source servers will vet possible destinations and offer only an allowed list. Destinations may also vet sources and only accept or initiate transfers from their allowed list of sources. (By the way – the choice for source servers to make sure that destinations are acceptable is independent of the choice for authorization to be requested by the source or by the destination. It can work either way).

The choice of vetting other services is a notable architectural choice because it's a component that the services don't already have and it may not be trivial to do. It is a weighty responsibility to assess a service and say that it is OK to exchange data with that service, because of the number of bad things bad people try to do with private data. Luckily, for most data types, there's not too many destinations to consider vetting and listing some top options. If the user wants to move photos, there just aren't a million or even a thousand photo hosting sites that are well-known and reasonable places to host their personal photos. A drop-down list in a Web form, containing a list of vetted photo hosting services, is a reasonable UX choice that covers a lot of user choice, and contains ample opportunity to add new services in the future.

If any of the choices made above feel limiting or disappointing, remember this is a reference architecture. Especially once we've considered the threat model for this reference architecture, we'll be in a good position to add features or compare a more open architecture.

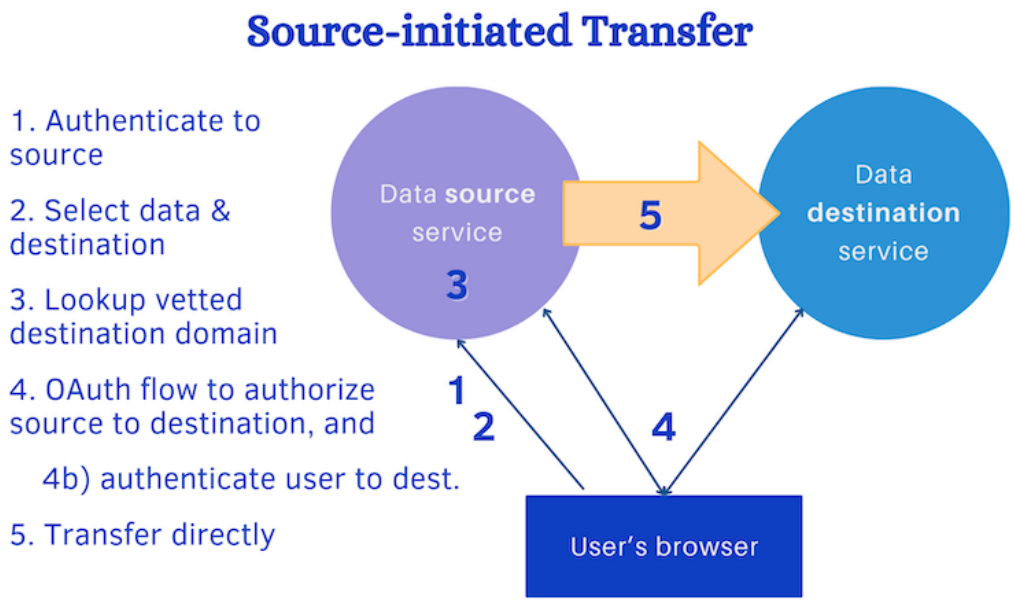
One major decision left open: data transfer could be initiated by the source service holding the data or it could be initiated by the destination requesting access to the data. That leads to two very similar architecture diagrams, and two very mirrored sets

of steps to start a transfer. One architecture is source-initiated, and the other is destination-initiated.

Both alternatives:

- Use OAuth to authorize one server to the other with the user's consent;
- Can authenticate the user to both sides;
- Involve vetting the destination service and (if needed) the source;
- Allow the user to use Web pages to choose data and destination; and
- Use secure transport for all interactions.

Source-initiated variant

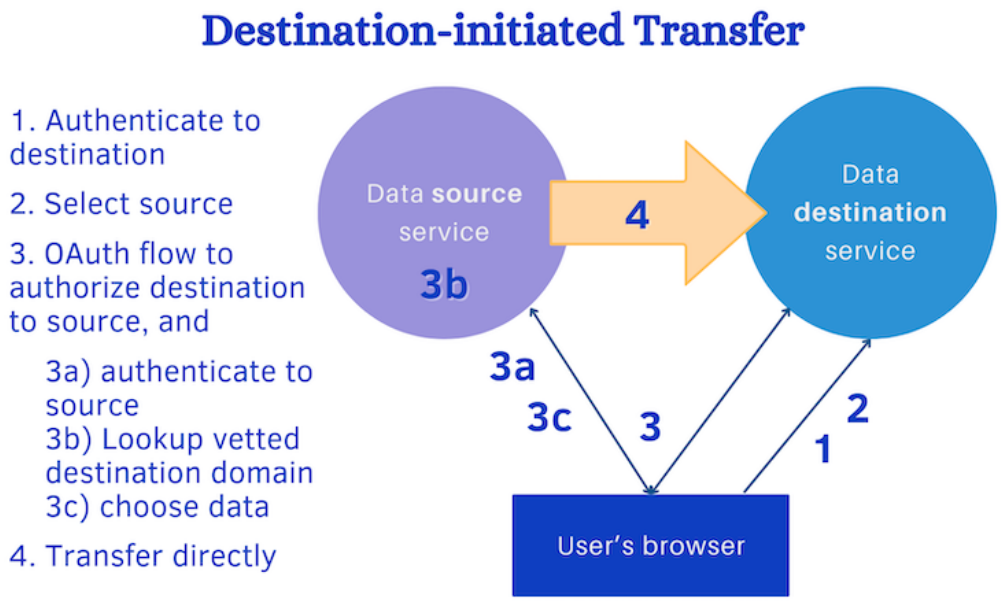


In source-initiated transfer, illustrated above:

1. User authenticates (login, etc) to the source service
2. User sets up a data transfer, including choosing what data and what service to send it to.
3. Source makes sure that there is a secure domain associated with a vetted destination before continuing.

4. a) Source initiates the OAuth flow, resulting in authorizing the two services to each other. b) During this, destination requires authentication (eg. login or session key).
5. Data transfer can now begin using a protocol or API over secure transport. The source may use the destination's API, the destination may use the source's API, or they may use a standard protocol.

Destination-Initiated Variant



In destination-initiated transfer:

1. User authenticates (login, etc) to the destination service.
2. User chooses to load data in from another service.
3. Destination initiates OAuth flow, requesting authorization for access to the user's data. (a) Source requires authentication (login or session key). (b) Source also checks that the destination is in the vetted list. (c) Source also prompts the user to select data to transfer.
4. After complete OAuth flow, data transfer can begin, using any API or protocol over secure transport.

Because these approaches both do the data transfer over secure transport, and both allow the source of data to check the destination's domain against a list of vetted destinations, they both meet the requirements. Actually the destination can add a step to either architecture to do its own vetting of the data source service, if confirming the identity of the source service is necessary in a given use case (see for example data and content threats below).

Sourcing the threat categories

I came up with a list of threats to consider from a number of sources. Most of the threat types or categories in this list can be found in well-known threat frameworks, particularly STRIDE (security) and LINDDUN (privacy). ITU's X.800 and X.805 offer security architectures for system interconnection and for end-to-end communication respectively, with many threat considerations framed more positively as "security services" than as threats.

Security Threats involving possessing data/content aren't always the same as the threats to servers or protocols, so it's also useful to consult NIST's catalog (Appendix E) of data threats and Camille François' Actor-Behavior-Content framework (ABC).

Threat category	Sources / more info
Disclosure of the data being transferred	STRIDE, X.805
Indirect loss of privacy, e.g. linking, identifying, detecting	LINDDUN and NIST
Tampering	STRIDE, X.805
Non-Repudiation	X.800
Denial of Service	STRIDE, X.805
Elevation of Privilege	STRIDE
Non-Compliance	LINDDUN
Harmful Content	ABC
Spoofing and related bad actor activity	STRIDE, X.805
Permission and access control challenges	LINDDUN and other themes that arose

Evaluating each of these threat categories with a high-level architecture to compare against allows for some threat categories to be set aside as mostly solved. Sometimes we can note that more detailed decisions made within the reference architecture can be of further help mitigating threats, and that's useful as a checklist for future design work.

Some threat types are similar enough to problems that data source and destination services already have to handle that we can use those services' existing solutions. That's not to say those solutions are easy (content moderation is very hard!) but they may be routine - the experience and expertise are there to deal with these threats, and

the threats can often be handled the same way that they are dealt with when direct user content contributions are involved.

Using a secure transport helps with threats

If we stipulate that all the interactions in our reference architecture diagrams happen over secure transport, especially the data transfer itself (big arrow in above diagrams), that helps with many threats.

Disclosure or loss of privacy

Secure transport protocols provide great protection from exposure of data in transit. The source and the destination both have access to the data in most cases, but that's the user's intent anyway. Secure transport protocols are mature and well-tested, so we can fairly confidently say that this problem can be solved. Of course, the same precautions that are required whenever secure transport is used must be applied. For example, when TLS is used to secure HTTP traffic, the domain certificates of hosts must be checked using well-known steps.

Indirect Loss of Privacy

Linking and identifying are when privacy and confidentiality expectations are violated not by direct access to private information, but by combining other information. For content transfers, we might consider some of the following things confidential:

- The list of contents (album or directory names, data object names)
- The amount of data transferred
- The date and time data was transferred
- The account name, session tokens, etc.

Because the setup of a data transfer happens over secure transport, the selection or filter provided by the user is going to be protected. Then, because the data transfer

happens in the cloud, we find we have the excellent property that one user's data transfer traffic disappears anonymously in the midst of a lot of other traffic.

Many API endpoint addresses and content addresses must be kept confidential to maintain privacy against indirect breaches. Historically this has caused much loss of privacy often with no user visibility into the situation. Whatever protocol/API is chosen for server-to-server traffic, many addresses, labels and identifiers must be secured as well as the content being transferred. This can include URLs for user accounts, URL parameters with user account information or directory names, and other metadata. All these tokens must be secured within the secure transport and not be visible as unencrypted protocol header or envelope data. This is a known principle in using secure transport and many good solutions are known and commonly used.

Tampering

Tampering is quickly dealt with because transport security not only protects the privacy of data being transferred, it also prevents an attacker from modifying the data. It's important for secure transport to protect the setup also, because we don't want an attacker to tamper with the setup of the data transfer and try to send the data to a different destination or choose a different selection of data.

If I seem to be waving this off too quickly, wait until I discuss bad actors. I'm interpreting tampering for now as having somebody in the path of transferred data (or the information needed to set up a transfer of data) modifying it en route without authorization, and I'm interpreting other problems as lying within other threat categories.

Vetting source and destination help

Many threats are greatly mitigated by vetting the source of data as well as the destination, depending on the use case. Often it must be both! Services acting as data sources can be harmed by sending data to unverified destinations even if the user

requested the transfer. Likewise, services receiving data transfers may be harmed by malicious behavior of unverified sources.

Of course, just verifying a service doesn't guarantee good behavior, but it goes a long way. There's no guarantee at all against services "going rogue" - behaving responsibly from one day to the next and building up access and reputation, until one day the service starts behaving maliciously. In the context of data transfers, at least the user is involved in the process. Not only do the source and destination services vet each other, but also the user is actively choosing to send data. Thus, a cautious user could choose not to share data with a service they consider more risky.

Examples where trusting the data partner not to act too maliciously is important, are found within each threat category below.

Non-Repudiation

We can imagine either the source server or the user repudiating some content:

- The user might deny having posted some content to the source; or
- The source might deny that some content was ever posted or deny the timestamp for the post.

This kind of repudiation is mostly dealt with by deciding that the source and destination can vet each other and communicate over secure transport.

This can be important in some cases! A user moving their social media from one server to another might want to set up their account with a history of posts made elsewhere. A post made in 2020 might be moved to a new server in 2023, and we'd like to know if we're just supposed to take that on faith or if there's any evidence that it really was originally posted in 2020. The DTI Fediverse Miniconf considered issues like these – why some actors might want to make an account appear several years old when it isn't, why that's bad – and discussion can be found in the report.

Non-repudiation turns out to be better solved when direct server-to-server data transfer is used, because the source can believably assert provenance. This is better than when users download unsigned data and upload it to a new destination, where

the destination service has no ability to confirm if the user tampered with their own data.

Denial of service (DOS)

DOS threats are already routine for many sites, with or without data transfer functionality. We want to make sure that data transfer doesn't significantly increase DOS exposure, especially because data portability will result in more data transfer moved more quickly if it's successful. So let's break it down a bit:

- Some parts of transfer setup will probably occur using Web pages and forms, and online services already must know how to protect DOS attacks on those.
- OAuth is a widely deployed protocol with a well-known threat model including DOS attack considerations.
- Let's decide now that it's OK for services to apply rate throttling or storage quotas to incoming data or data requests.
- It helps that the services can vet each other, if one service throttles to such a low rate that it causes performance problems for the service trying to collate and send data, this can be negotiated between parties.

There are some system design details to setup data transfer to happen gracefully and efficiently (without accidental DOS effects). If a data transfer is set up with the scope communicated in advance, then the destination service knows to expect a large transfer, and can set aside resources or throttle the connection appropriately. If the passive party sends throttling or quota limits in advance, the active party can work within those or notify the user quickly. These details would be good to consider when the data transfer details are worked out, but the high-level architecture already looks pretty promising.

Elevation of privilege

Elevation of Privilege is considered a system security threat that occurs when attackers try to do unexpected things, like overflow buffers, to try to gain access to system resources they otherwise wouldn't be able to. With this architecture, user interface components and protocol implementations need to have the same

protections they would in any other part of the servicers' user interfaces. Connections to other services, because they are vetted, are at relatively low risk but still good API safety practices should be followed. These protections are all within the power of each service to provide for themselves.

Non-compliance

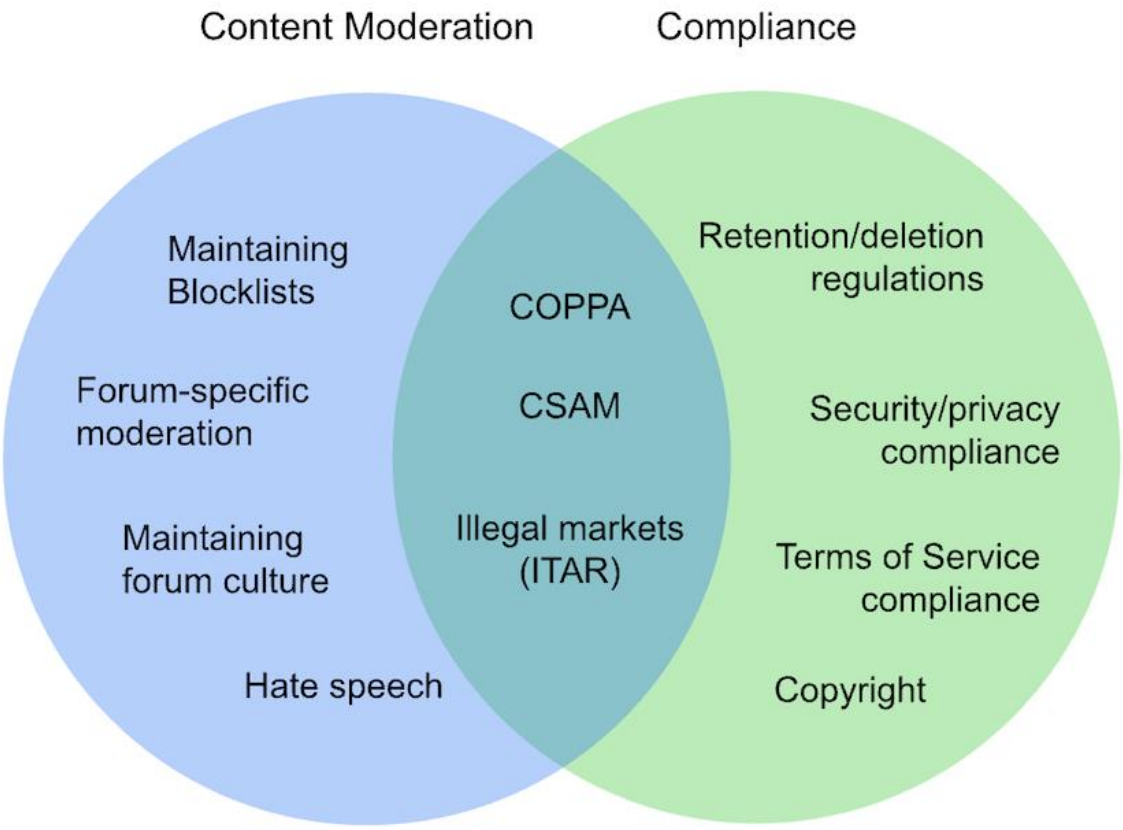
Non-compliance is a threat when a system that tries to comply with regulations is made non-compliant by participating in data transfer. Any company that has a compliance function already puts controls in place to stay compliant no matter what data users request or upload, so much of this is already in the services' control.

Many compliance regulations that apply to companies hosting content are driven by the content itself, its subject matter or representation. However, there are additional regulations and policies that are not about what is in the content but how the company handles it.

Source	Content policy	Content handling policy
From regulations	• Child-appropriate content laws (COPPA) • CSAM regulations • Healthcare data (HIPAA) • Personal data protections (GDPR)	• Limits to where data may be sent - e.g. to companies located in certain states. • Data deletion policies and requirements
From company policies	• Company policies on hate speech	• Company's Terms of Service, Privacy Policy, or customer contracts • Data retention policies

Compliance overlaps with several other threats. For example, a company can experience a breach that violates privacy expectations and the privacy breach may also cause it to be in violation of regulations or its own security policy or terms of service.

For content hosting services, compliance considerations overlap especially with content moderation, but neither topic encompasses the other:



Lines are blurry but some things overlap more than others

The fact that data is transferred directly between vetted services may make it easier for many companies to ensure compliance. It's likely that content that is acceptable on one service is also acceptable on another, for starters. The vetting/trusting process for data senders is more than users go through (users may try to circumvent policies or regulations and post content they're not supposed to) and companies are less likely to get malicious content sent from another service than directly from users.

There are definitely differences between different companies' definitions of acceptable content and compliant behavior, however. Different jurisdictions, regulations applied to certain industries or services and not others, and different terms of service or privacy

policies can all result in a need for the destination to filter content even when it comes from another service.

Most of the work to ensure compliance must be done by each company on their own, for content provided directly from users as well as content provided via other services, so like denial of service and elevation of privilege, we can leave the handling of this threat in implementers' hands, and the high-level architecture need not change. Low-level features might help, for example providing a content manifest before a data transfer begins, or providing metadata that the content source generated in its analysis of whether the data was acceptable to that service.

Harmful Content

Harmful Content is already a threat in any system that hosts user-contributed data. It's especially a problem in bulk, because bulk transfers of data may make it harder or less effective to do content filtering or content moderation, or bypass protections on the more common input channels. Harmful content can be harmful to the service or harmful to other users.

While we seem to discuss the threats of data flowing from sites more often than the threats of data flowing to sites, there are threats to both parties. The W3 privacy principles framework puts emphasis on information flows in both directions for consideration of harm.

Bulk Submission

Content that arrives in bulk can bypass normal channels (whatever channels are most commonly used to get content). For example, the normal channel for Instagram posts is that ***one user submits one post at a time via the Instagram app***; and the normal channel for Instagram comments is that ***one user submits one comment at a time via the app with a notification to the poster***. Normal channels have protections like in-band verifications, notifications (as in the Instagram post/comment example here) and rate limiting.

Many of the common content protection mechanisms are challenging or inappropriate to use in bulk submission, so we'll list some of these, define and explain the challenge.

Mechanism	Explanation of Mechanism	Challenges for Data Transfer
In-band verifications	There can be verification of submitted content before it is finalized. A user submitting an abnormal media type may be asked to confirm by the Web page. A mobile app may warn that some of the content seems risky to privacy or contains terms that might be offensive before allowing the post.	Bulk content upload/transfer does not allow for in-band verifications like this. Confirmations must be built in differently or skipped.
Human Interaction Proofs (e.g. CAPTCHAs)	A special-case of in-band verification is technology that attempts to verify that a human was involved in submitting content or a message. This can be an automated Turing test or a biometric.	Bulk content upload/transfer does not allow for human interaction to be tested at the individual message/post/media level. Bot-generated content can be mixed in with human-generated content.
Third-party Notifications	Notification messages can be triggered by content submissions, often for the purpose of immediate review. Many online content platforms send a notification to a content provider when somebody comments on their content. Many forum services notify a moderator so that new content can be reviewed quickly for appropriateness, for meeting terms of service or community guidelines.	Bulk upload/transfer can allow for notifications, but at too high a volume, are likely to be useless or worse. Each message is less likely to be reviewed carefully if it is a large collection of older posts originally posted elsewhere. The flood of expected notifications can hide an important notification of harmful content.
Rate-limiting	Rate-limiting and notifications together can work very well to stem a flood of bad traffic. Many social "attacks" involve flooding a comment section, online forum or other venue with messages that one-by-one could be dealt with via notifications and removals. Flooding can be limited in many creative ways, from putting a limit on where a new account can post to limiting the number of comments anybody can submit per comment section.	Bulk upload/transfer must by its nature bypass normal rate-limits, so it will be an attractive channel for bad use in flooding attacks.
Quarantine	Sites may quarantine some content submissions; suspicious content may be put in quarantine or all content may go through a step of being visible in a limited way before being published more widely. Email quarantines are frequently used inside corporate and consumer email systems - even if it's not called "Quarantine", a spam folder serves a similar purpose.	It may not be feasible to put all or even only suspicious content into quarantine in bulk upload/transfer situations. Processes to move content out of quarantine may not scale to bulk submissions.

These mechanisms – in-band verifications, human interaction proofs, 3rd-party notifications, rate-limiting and quarantine – cannot be used in bulk transfer situations, or at the very least must be adapted to work a little differently. Thus, content service providers must solve the problems that are solved with these features in a different way for bulk transfer, and that can be a significant effort of new engineering and code/process maintenance. If we consider these mechanisms and challenges while designing data transfer solutions we may be able to help the content platforms replace these mechanisms.

Content Harmful to People

Other researchers and writers have catalogued depressingly the many ways in which content can be harmful to people. Perhaps the best thing to do here is to point to the ABC Framework and the D for Distribution article (which points out that tools that boost distribution can boost harmful content as well as good content). Data portability can certainly be used to boost, change or disguise distribution, so it is worth reading both those articles.

A few examples from our specific use cases, inspired by those sources:

- A user may develop extensive block lists on one social media site but not for another, creating challenges if harmful blocked content is ported to a new site and thus no longer blocked.
- A harasser can use data transfer tools to move their account, again and again and again, to continually evade the block, mute or ignore features of their harassee.
- Disinformation posts can be copied from site to site more easily with data transfer tools. It's all too easy to imagine a disinformation campaign in which people who support a certain (perhaps extreme) position are asked to copy all of one account's content into all of their accounts and sites to disseminate that content.

Content Harmful to Services

There are unique risks in bulk-importing data that is normally not imported - data that is normally observed, derived or inferred. Some examples:

- Search engines build logs of past searches that they can use for contextual clues to improve future results.
- Online stores build purchase history databases, then use those to recommend products
- Music streaming services track listening activity to select more tracks to play, as well as to provide info such as Spotify Wrapped.

When these services use activity history to provide benefits to users, there's a great case to be made to allow the user to bring their own history to a new service rather than start from scratch. However, there are now risks to that new service in importing bulk data. First, the new service needs to be wary of possible tampering that could worsen the service's functionality (such as importing data claiming that a new song was played ten million times). Second, the new service needs to check the bulk data for buffer overflows and embedded code at least as well as data contributed one-by-one.

E2EE and content safety

What if the content is end-to-end encrypted (E2EE)? In this case, the server is by definition not in possession of keys to unencrypt content. This a situation where content moderation is already challenging.

E2EE is not the most common setup when user-approved server-to-server data transfer is requested, but there are still some potential use cases.

- After using an E2EE messaging app for years, using it to exchange everything from family photos to payment requests, a user wishes to transfer their message history to another service. The messages and photos are all stored encrypted on the original service.
- A digital drop-box service used in the legal industry contains encrypted content submitted by a user (the client), and the keys have been distributed in a different

channel to the user's legal representatives. A client later wants to move this data off the expensive legal site but cannot just get rid of it yet.

To the extent that the interoperability of these use cases can be solved, the use of E2E encryption of content at-rest can make transfers somewhat more risky also. The server that originally stored the encrypted data can make few assurances if it does not hold keys to unencrypt the data; thus in these scenarios we can't necessarily rely on the source server doing content filtering/moderation. The destination does not necessarily operate under the same model. A service model at the source wherein harmful content detection is done at the client (because of the E2EE design) would present extra challenges when that content is sent to a new destination with a different model to deal with harmful content.

E2EE use cases present a good opportunity to remind ourselves that system models and user expectations can differ significantly between seemingly similar systems. When a personal messaging app that uses E2EE with client storage interoperates with a different personal messaging app that uses E2EE with server-side storage, or a non-E2EE messaging app, data portability can be difficult and even when achieved technically may be done in a way that users did not expect. Practitioners in the area already find that users can be surprised by the results of a data transfer even within the same system (e.g. merging 2 accounts)!

To summarize, the existing challenges of E2EE content moderation are not significantly increased when data transfer is introduced. In specific concrete cases, we do need to be mindful of the safety assumptions on both sides when E2EE content is transferred. Techniques that may allow some identification of harmful encrypted content, such as homomorphic encryption, ought to work just as well in data transfer situations.

Spoofing

Spoofing is when a data transfer participant tries to convince one or both of the other parties in a data transfer that they are somebody else other than who they are.

Recall our reference system architecture. Since this architecture authenticates the user both to the source and then to the destination service, the overall system is well-protected from bad actors impersonating the user. That's great news, because many threats to services and users come from a small subset of users. It doesn't make the problem of bad actors go away, but it means that the problem has already been solved by robust systems. A user who attempts to evade restrictions, post harmful content, or otherwise behave in ways that the service provider forbids or prevents, can still be identified in these scenarios, and may often be blocked or removed if their behavior is bad enough.

We also need to make sure that users are protected from bad actors impersonating sources or destinations, and this is a little trickier to see.

New avenues for phishing attacks

The architecture we've defined operates on the World Wide Web, and one of the big challenges of the Web is how easy it is to spoof users by pretending to be a legitimate service. For example, an email pretending to be from “photobarrel.com” could actually be from “photcbarrel.com” (with a C where an O should be) and lead to a password challenge; this is an example of both spoofing and phishing. TLS and host certificates do not help here: bad actors can easily obtain certificates too. Data transfer setups may offer new avenues for spoofing, such as:

1. A user sets up content transfer from one publishing server to another.
2. Because their content is visible on one site then updating on another, an attacker can tell that they have real-time and continuous data transfer in place
3. The attacker sends a notification appearing to be the source server sending a regular authentication renewal notification, with links to the attacker site rather than the source site
4. The user attempts to quickly authorize the renewal but authorizes something else instead, or their password is phished.

While there are undoubtedly new avenues like this, spoofing and phishing are already so common that we need mechanisms like two-factor authentication and authentication challenges whenever the stakes are high. Practitioners in the area have

seen verified instances of data exfiltration attempts and successes using data portability tools.

Vetting services

A significant protection we can employ to protect users from these attacks is to limit data transfers to happen between pairs of vetted, known participants. DTI is working on ways to standardize a trust model (recently announced!) so that decisions to trust sources and destinations can be more accountable and transparent.

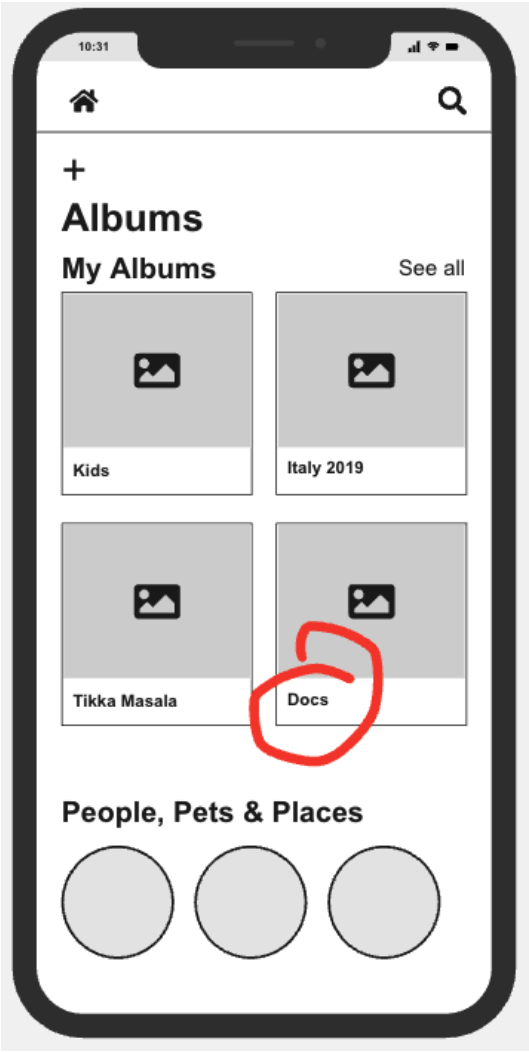
What does vetting data transfer partners solve? It prevents the worst opportunities for spoofing, because a user fooled by a visual URL scam like “photcbarrel.com” will not be able to use that domain name in the data source service without that service being able to tell it's not vetted. It's also possible that the user will only select from a list of available destinations.

Some data transfer use cases can allow unvetted participants more safely, but in the cases where risks are higher, trust between services is important and feasible, and the challenges to scale N-to-N trust can be addressed.

Permission and Access challenges

One of the greatest challenges remaining in doing data transfers with appropriate privacy isn't even caused by bad actors - it lies in the difficulty of translating a vague user intent into detailed software algorithms.

Photos once again provide a simple example of this. I may have many photos on a service, especially if that service collects photos automatically from my devices and conveniently stores them in one place.



The problem: my Docs album has images of vaccine documents, visas, health insurance…

It's all too easy for a photo migration project to include images I don't even think of as photos: that picture I took of my driver's license, or a screenshot of some HR data or scan of financial documents. Ideally the source of a data transfer would be able to check on the situation and help apply appropriate filters. To that end, it would help for the source to know some of the context for the destination, such as whether the transfer will result in publicly accessible content. This is an interesting feature to design for, and very content-dependent.

We also have permission and access challenges with the destination. Many services accepting user data would like to know the user intent when accepting it. You're uploading a photo, would you like it to be private or public? Would you like it to be Creative Commons licensed or copyright protected? Which folders? Which tags? Would you like it to go on your stream? If a new playlist is uploaded, should it be added to the user's playlists? Should music not yet purchased be purchased now? What if quota is exceeded, do you want to pay for more quota now?

Just like any complicated activity can go badly, and the results can be mistakes from the user point of view, complicated data transfers can be done badly. Even with legitimate source, destination and authorized user, no attacks or bad actors in sight, a user can accidentally copy a photo album of docs to a public location. Phrased as an accident, it's not technically a threat – but sometimes bad actors set up victims to make mistakes. A site requesting photos for "cancer research fundraising PR purposes" or another benevolent goal can induce users to share more than they intended. We need legitimate data transfer hosts and destinations to present carefully designed and tested User Experience (UX), and in some cases we need extra information during transfer setup so that the source can learn more about the destination and vice versa.

We can talk about architectures that involve detailed setup on both sides - careful filtering of content on the source side, thoughtful application of privileges and licenses on the destination side, or other choices that these services find important depending on their context. Maybe the solutions will involve more complicated use of OAuth, or more handshakes between the services to make sure both sides have completed checking with the user how they want this to work.

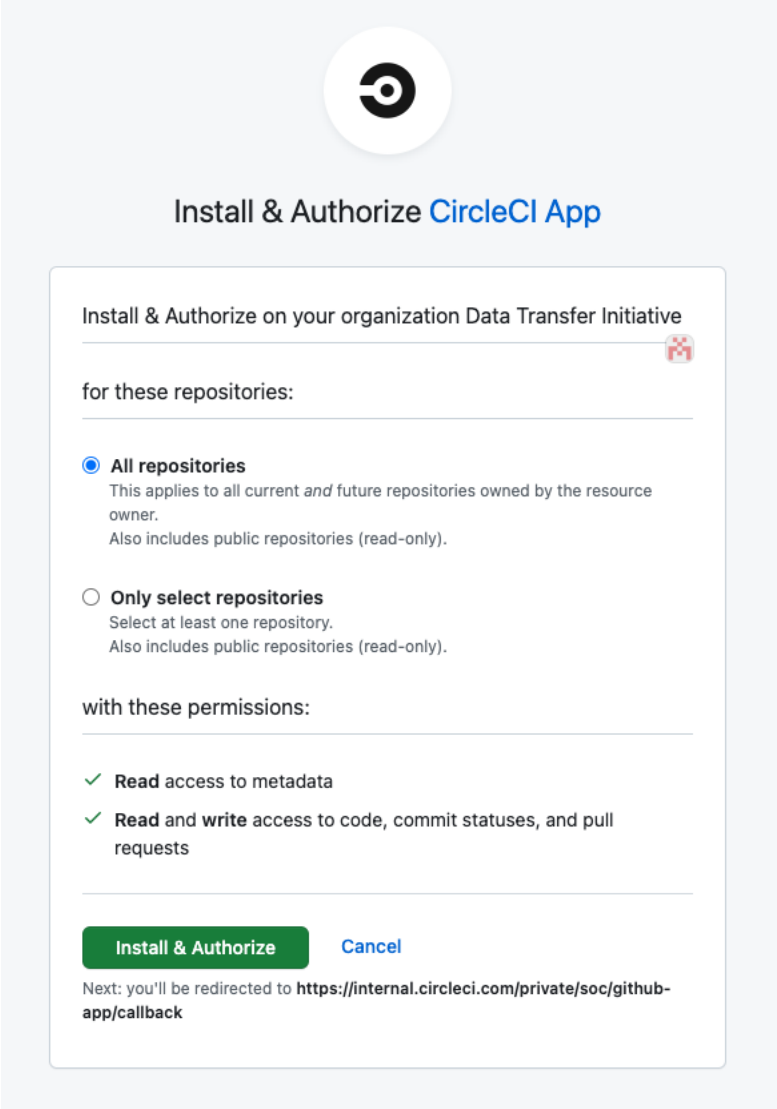
Who initiates

We illustrated the authentication issue with the source-initiated architecture. Is it any better or worse in destination-initiated transfer? When the source service initiates transfer, whether it be in an app or a Web page, the source can supply all kinds of UX, including selections, verifications and filters, to make sure that the *correct limitations are used to generate outgoing data*.

In contrast, in the architecture where the destination initiates transfer, the destination (an app or a Web page) can show all kinds of UX to ensure that when the data arrives, the *correct permissions and licenses are applied to the incoming data*.

Since both parties need to present these decisions in a clear informative UX, we need solutions that allow both.

Our choice of OAuth for the way that users authorize the non-initiating service is key to this dilemma. Not only is it a good interoperable choice for authorizing the user, it also allows the responding service to include any number of decision-making steps in Web pages in the path of authorizing transfer. It doesn't make all the problems easy, but it is a pretty powerful framework.



When CircleCI initiates an OAuth interaction with GitHub, GitHub's interstitial Web page asks about scope and permissions in a clear way

Post-transfer permission and access work

Consider the use case where there are detailed dispositions to be made after a content contribution.

- Flickr has a sophisticated upload UX, with the ability to choose tags and albums for some or all photos in the upload, and to consider which photos should be private, public, limited to family, and/or Creative Commons licensed.

- Wikimedia Commons has a contributor flow that involves long Web forms for *several* of the steps in their flow.

Wikimedia contributor flow: Learn, Upload, Release rights, Describe, Add data, Use

The interesting thing about their UX is that it happens after the data upload is complete – both Flickr and Wikimedia Commons show thumbnails of the images so that users know exactly what they're annotating and are less likely to make mistakes. If we architect data transfer such that it happens out-of-band with the user, it could be more challenging to apply the right privacy choices on the destination. This is another area where data transfer participants will have to be thoughtful and innovative in designing UX that helps users make good choices and fewer mistakes.

Summary

Our analysis of threats against our reference architecture allows us to better understand which threats are going to be more or less challenging to mitigate in data portability solutions.

- Direct loss of privacy, tampering and unauthorized access are easier to mitigate through common encrypted transport approaches.
- Some threats are mostly harmful to the service, such as denial-of-service, elevation of privilege and non-compliance. Services offering data portability features will have to plan ahead to protect their bandwidth, storage, servers and other assets.
- Harmful content is a particularly challenging threat to mitigate, as bulk transfers make many content-oriented safeguards work less well. Services will need to

consider this if harmful content is a relevant threat to a given application or use case.

- Deceptive activity, including phishing and spoofing, may find new approaches to trick people into sharing sensitive data or providing excessive access, possibly taking advantage of complicated permissions and scopes in consent processes. Services will need to do careful UX design and be mindful of what third-parties are using data portability features.

None of these threats are entirely new to any system we've considered. Thus, data portability does not present an intractable security risk to content platforms.

When practitioners are writing specifications for data portability services or standards for data interoperability, additional threat analysis should be done for the specific context and solution. The threats considered here can be a good guide.

We do hope that data portability becomes easier and more possible. Ever-larger user datasets online have made direct downloading a poor solution to migrating between services for photos, videos, email and social media. Trust and interoperability issues make it hard for users to migrate or share their own tasks, notes, shopping and search history and more. The benefits of data portability do incur some risks, but we have prepared better for those risks by laying groundwork for identifying threats and finding solutions or mitigations to those threats.