Patterns in S3 Data Access

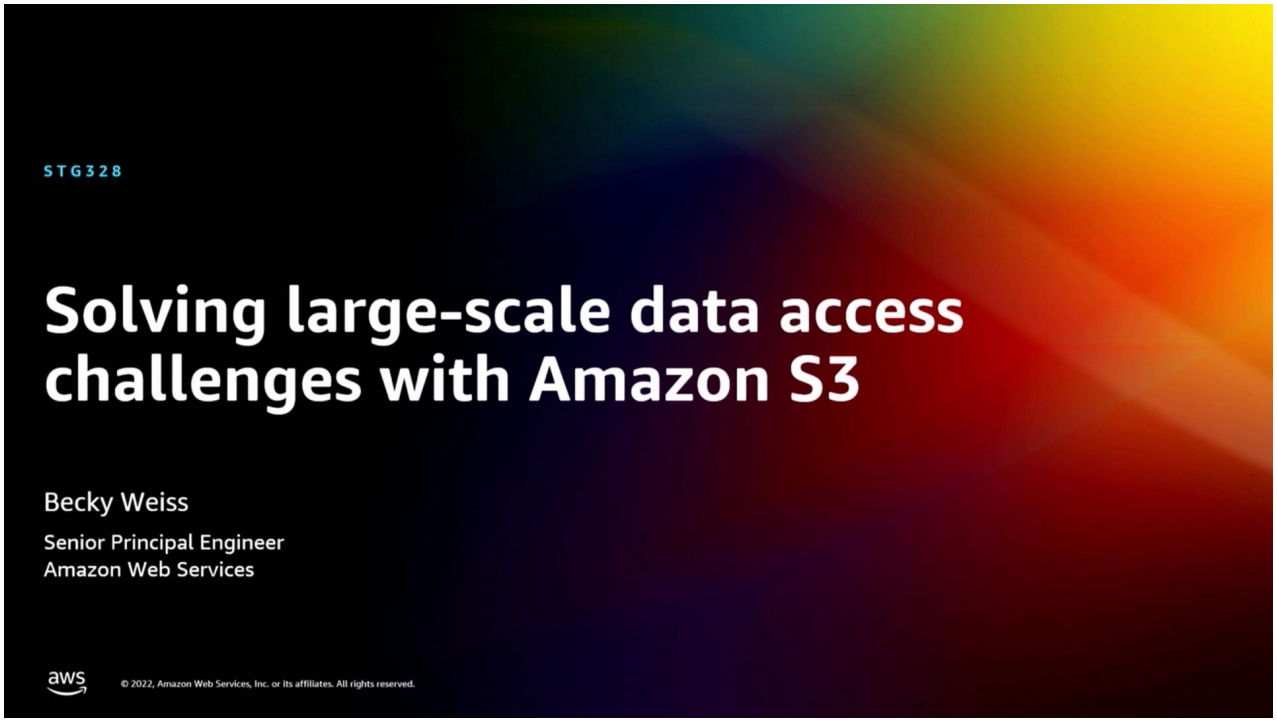
Protecting and enhancing access to data banks, lakes, and bases

Josh Snyder @ fwd:CloudSec, 2023

Synopsis

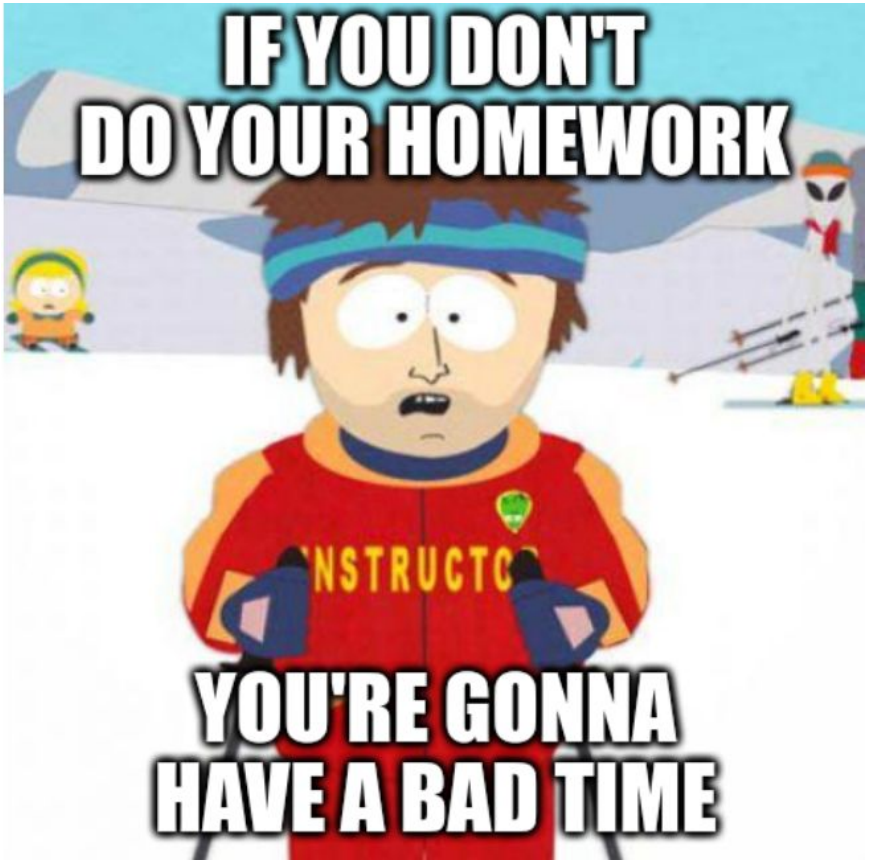
1. Preamble: AWS request signing
2. Problem: Complex-pattern data in S3 (e.g. data lakes)
3. Common solutions
4. Just-in-time access

Go watch this: https://www.youtube.com/watch?v=BOz2_hgoob4



Disclaimer!

Do not go implement anything from this talk without first watching that talk!



Preamble: AWS request signing

Motivation: Josh's Photo Sharing Service

- Users (millions) upload their photos (billions)
- Photos are 1MB up to 1GB
- Photos are private by default
  - original uploader can always see their photo
- Photos can be added to albums (1->many relationship)
- Albums can be shared with other users (1->many relationship)
- Access to an album can be revoked at any time
- No photo needs to be super-popular (no CDN)

Signed URLs

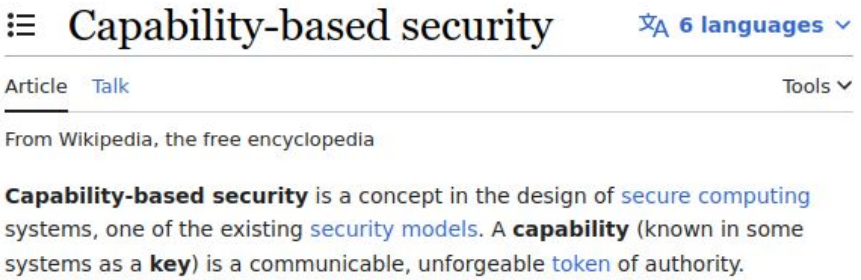
https://artifacts-911be34e61abfc8d.s3.us-east-1.amazonaws.com
/serverless/certs/dev/1608193965988-2020-12-17T08%3A32%3A45.988Z/certs.zip?
response-content-disposition=attachment
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Date=20230607T083045Z
&X-Amz-SignedHeaders=host
&X-Amz-Expires=300
&X-Amz-Credential=ASIA3LROMZGCSJI7XRXE%2F20230607%2Fus-east-1%2Fs3%2Faws4_request
&X-Amz-Signature=57121f2315ae365434e3b70b54e2e72d3b4a9bf93fa0e864c3a960da4457d9c4
&X-Amz-Security-Token=IQoJb3JpZ2luX2VjEBEaCXVzLXdlc3QtMSJHMEUCIAQWLf5nbq4T4PlfFoCc26X1aDgnLw%2BtR9flpwL9
VFc5AiEAyFwpzVUB7%2BftMY%2BXVqzy3LZ2k%2FOMmaUs7H9bgZW%2FdSwqygMIWhACGgw3ODA3MDc2MTMwNjEiDJMAz
soSh3PhmuvJrCqnA0QsG0upWxiYAGjfFflqVXEL14cW8slX8qRFoeUUYvUUpAV2RoZACSI%2BJ8MgbYYmDpTDVe0Uc3FxFrFloy
EA3pDuzXlVFnN9x9tdw5p2QQms1MCKkdQpY6QoNvnD8z3vh4oK7lT0WFMcto2Hw6LCJIlruL2FfYgnGvN5bFBZzcgBTp2xZCUi0b
ExKdZcXZLpkE7Lu%2Bq4IAooox1IuqNVxdw00lGgeODpauSznSXLadQrIv4CC1Gw%2BMYuRMtfsB4P%2F0Rz5wpysfLSUcRsKRf
wxhh8jV9%2FtD4ECHtIrADGhvMVdmcYBtgFAhG9YWjAy78MnfyjmyoanHmdsMQKRXpsqQI%2Fzh%2BH%2FJX%2FnirVyGbmMH
czfvemRV8WA3OP2gc98qhLkgspGAy%2FowpWdnXctHHL3uIKDh3bq%2FX6hn6VwcL8lShztu67J1375Tu%2B4BdcBAM2yn8Ccm
KwyYlmLgFKNMVp%2BJk30V4zicbH9Ug9RAbvQNb52aLNFsp%2FIQ11OgDgvSYDqR9Y3cKW%2F3HZLxgxtsyerqZbJG87dp8X1
MzBKFxs6PKZGV91Fl1%2FITDy%2F4CkBjqUAm8M60CyphsIo2CF7i3sDKE%2FwBiEPKqKf%2Frrj8mDTwnCAAp3eS0OjD2ph8Q
3NDto0QiAeCh8DlYXgsNpjACeFANnqHCgIdB2%2BunKXps9%2FZg7mdaoVyEfUn2X874QtZ%2B62kus9%2Ff8%2BkroF92lip%2B
77%2BANKJ6DE%2BiETcf7F21D%2FRosmDfMfZLBeclJanxzdN5jlLk64e6kkJA0O77LC1hpzVotmKACNAfwJJCFNUsNPkYjMeKzF

Signed URLs expire (good!)

```
−<Error>
    <Code>AccessDenied</Code>
    <Message>Request has expired</Message>
    <X-Amz-Expires>300</X-Amz-Expires>
    <Expires>2023-06-07T08:35:45Z</Expires>
    <ServerTime>2023-06-07T08:54:53Z</ServerTime>
    <RequestId>WSAXPXJKFTQFYDWP</RequestId>
   −<HostId>
      +EGS5rRm7enuhyfw0/Osnq2t+JZslwVAt7fpaW4pwjpH6WepthJR2UJVv8dySiOEVuc4nhKNDwE=
    </HostId>
  </Error>
```

What are signed URLs?

- Portable **capabilities** (antique term, from the ~1970s and 80s)
- Cryptographically authenticated!
- Useful for **browsers**
- Not useful for API-based clients



Capability-based security

6 languages

Article Talk Tools

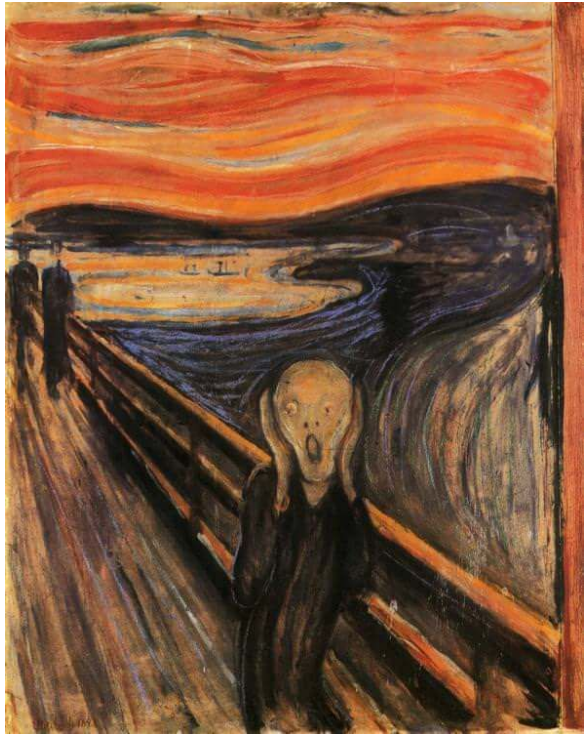
From Wikipedia, the free encyclopedia

Capability-based security is a concept in the design of secure computing systems, one of the existing security models. A **capability** (known in some systems as a **key**) is a communicable, unforgeable token of authority.

Portable Capabilities!?!

- the whole point of capabilities is to flow through a system
- this freaks security engineers out (pictured at right)
- docs used to say this (below); it was removed sometime late September–early October 2022

> ⚠ **Important**
>
> If you make a request in which all parameters are included in the query string, the resulting URL represents an AWS action that is already authenticated. Therefore, treat the resulting URL with as much caution as you would treat your actual credentials. We recommend you specify a short expiration time for the request with the `X-Amz-Expires` parameter.



Making Signed URLs safer

- Options:
  - IP-binding
  - source-VPC binding
  - cookie-binding

https://github.com/hashbrowncipher/safer-signed-urls

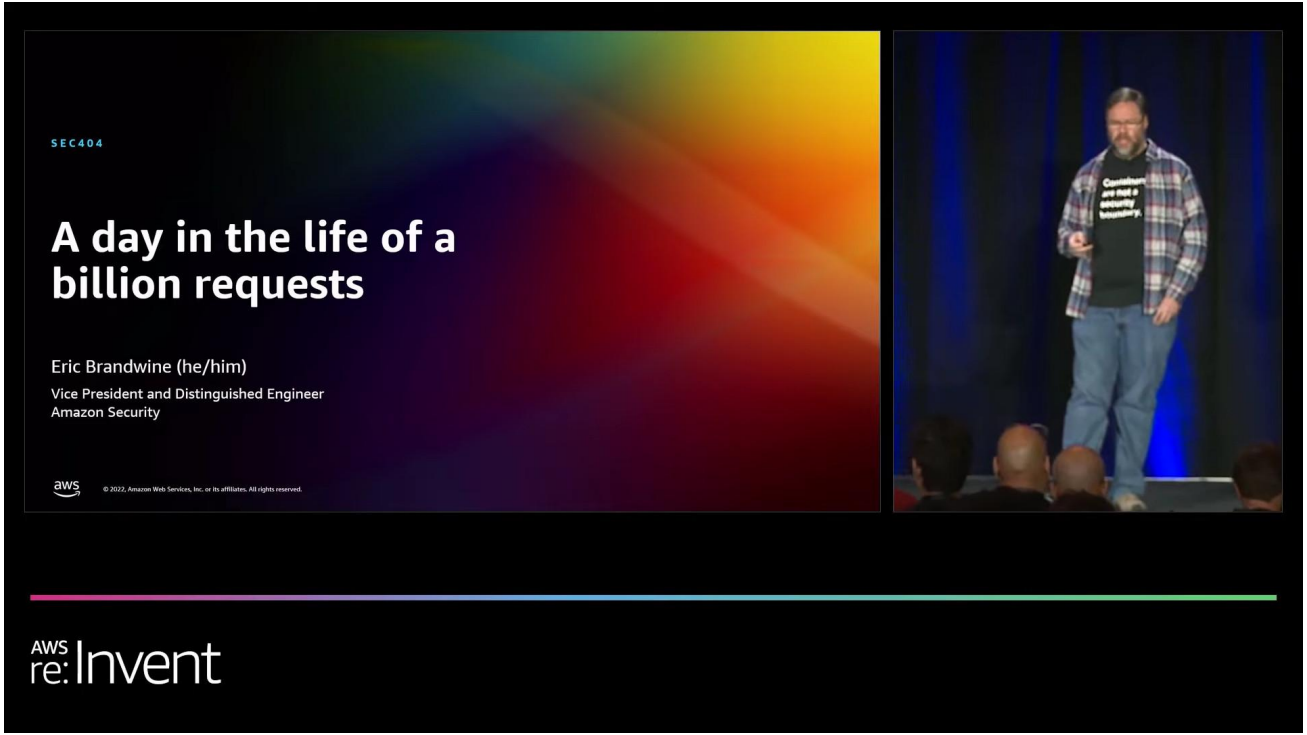
Every request in AWS is a portable capability!

Cryptography of AWS Request Signing
(dangerously oversimplified)

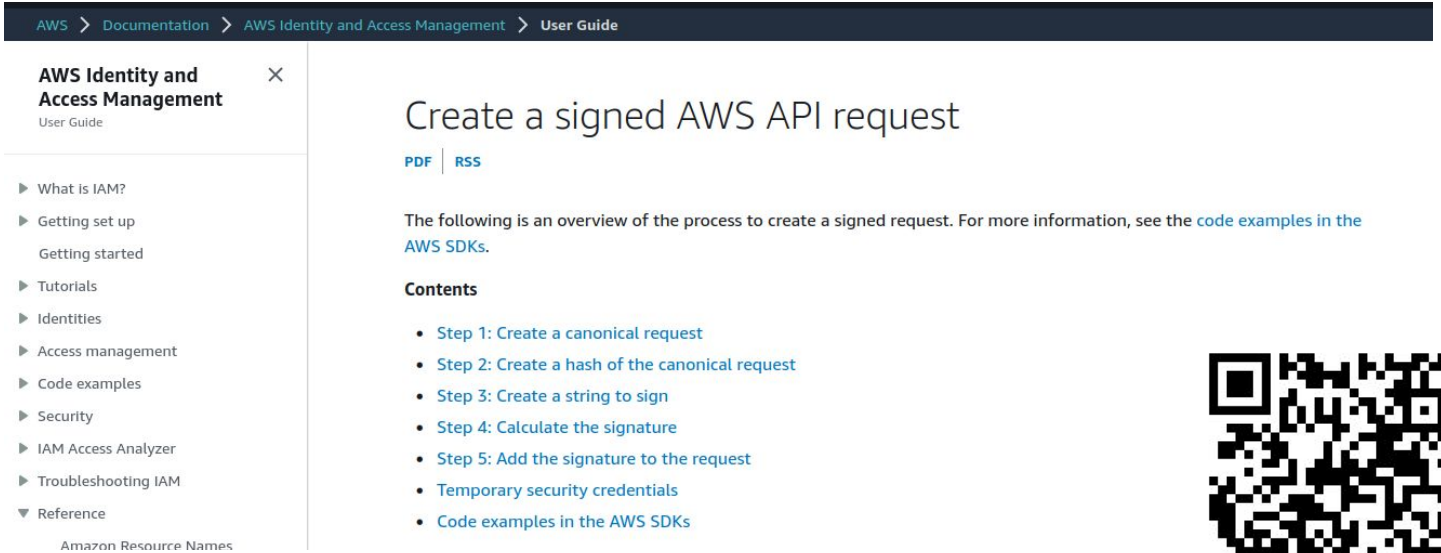
1. Authentication
   a. I send: `HASH(AWS_SECRET_ACCESS_KEY || My Request) || My Request`
   b. Amazon computes: `HASH(AWS_SECRET_ACCESS_KEY || My Request)`
   c. If not equal: send `HTTP 403 Forbidden`
2. Authorization
3. Do the thing
4. Send the result

```
−<Error>
  <Code>SignatureDoesNotMatch</Code>
 −<Message>
    The request signature we calculated does not match the signature you provided. Check your key and signing method.
  </Message>
  <AWSAccessKeyId>ASIA3LROMZGC26I527LK</AWSAccessKeyId>
 −<StringToSign>
    AWS4-HMAC-SHA256 20230610T065831Z 20230610/us-east-1/s3/aws4_request 228c6c2be41bf9d75febd83550ded844c46c601fb5a38c490d8d21054c9881d0
  </StringToSign>
 −<SignatureProvided>
    f4d8f21d2d038442f8186da1870c4885abf39a1bc1429e84abd25a7c335082a0
  </SignatureProvided>
 −<StringToSignBytes>
    41 57 53 34 2d 48 4d 41 43 2d 53 48 41 32 35 36 0a 32 30 32 33 30 36 31 30 54 30 36 35 38 33 31 5a 0a 32 30 32 33 30 36 31 30 2f 75 73 2d 65 61 73 74 2d 31 2f 73 33 2f 61 77 73 34 5f 72 65 71 75 65 73 74 0a 32 32 38 63 36
    63 32 62 65 34 31 62 66 39 64 37 35 66 65 62 64 38 33 35 35 30 64 65 64 38 34 34 63 34 36 63 36 30 31 66 62 35 61 33 38 63 34 39 30 64 38 64 32 31 30 35 34 63 39 38 38 31 64 30
  </StringToSignBytes>
 −<CanonicalRequest>
    GET /serverless/certs/dev/1608193965988-2020-12-17T08%3A32%3A45.988Z/certs.zip X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=ASIA3LROMZGC26I527LK%2F20230610%2Fus-
    east-1%2Fs3%2Faws4_request&X-Amz-Date=20230610T065831Z&X-Amz-Expires=300&X-Amz-Security-
    Token=IQoJb3JpZ2luX2VjEFcaCXVzLXdlc3QtMSJGMEQCIEKnfXqi7vMkH12snXDlJC7n6y3Q3CgJfwzm2rKbLbZtAiAzMYI5jAdOtTKxJuiq0Eoeh8Ju%2B08NtzvsyW%2BZE82PXSrTAwig%2F%2F%2F%2F%2F%2F%2F%2F%2F
    %2F%2F8BEAIaDDc4MDcwNzYxMzA2MSIMvvsswdjwRJrPKgKyKqcD0hkIibCBNVdmilDhHvhvWvBa1%2FC2T7d8ACr7eBSLgBflydOPwtGKbXwE4u7qSuO6v3NpYEskNITFZvy0uy8BjcD5Bn2zzOu6cFg%2FH8VHcqUWEZz2PUcBCQ
    %3D%3D&X-Amz-SignedHeaders=host&response-content-disposition=attachment host:artifacts-911be34e61abfc8d.s3.us-east-1.amazonaws.com host UNSIGNED-PAYLOAD
  </CanonicalRequest>
 −<CanonicalRequestBytes>
    47 45 54 0a 2f 73 65 72 76 65 72 6c 65 73 73 2f 63 65 72 74 73 2f 64 65 76 2f 31 36 30 38 31 39 33 39 36 35 39 38 38 2d 32 30 32 30 2d 31 32 2d 31 37 54 30 38 25 33 41 33 32 25 33 41 34 35 2e 39 38 38 5a 2f 63 65 72 74 73
    2e 7a 69 70 0a 58 2d 41 6d 7a 2d 41 6c 67 6f 72 69 74 68 6d 3d 41 57 53 34 2d 48 4d 41 43 2d 53 48 41 32 35 36 26 58 2d 41 6d 7a 2d 43 72 65 64 65 6e 74 69 61 6c 3d 41 53 49 41 33 4c 52 4f 4d 5a 47 43 32 36 49 35 32 37 4c
    4b 25 32 46 32 30 32 33 30 36 31 30 25 32 46 75 73 2d 65 61 73 74 2d 31 25 32 46 73 33 25 32 46 61 77 73 34 5f 72 65 71 75 65 73 74 26 58 2d 41 6d 7a 2d 44 61 74 65 3d 32 30 32 33 30 36 31 30 54 30 36 35 38 33 31 5a 26
    58 2d 41 6d 7a 2d 45 78 70 69 72 65 73 3d 33 30 30 26 58 2d 41 6d 7a 2d 53 65 63 75 72 69 74 79 2d 54 6f 6b 65 6e 3d 49 51 6f 4a 62 33 4a 70 5a 32 6c 75 58 32 56 6a 45 46 63 61 43 58 56 7a 4c 58 64 6c 63 33 51 74 4d 53 4a
    47 4d 45 51 43 49 45 4b 6e 66 58 71 69 37 76 4d 6b 48 31 32 73 6e 58 44 6c 4a 43 37 6e 36 79 33 51 33 43 67 4a 66 77 7a 6d 32 72 4b 62 4c 62 5a 74 41 69 41 7a 4d 59 49 35 6a 41 64 4f 74 54 4b 78 4a 75 69 71 30 45 6f 65 68
    38 4a 75 25 32 42 30 38 4e 74 7a 76 73 79 57 25 32 42 5a 45 38 32 50 58 53 72 54 41 77 69 67 25 32 46 25 32 46 25 32 46 25 32 46 25 32 46 25 32 46 25 32 46 25 32 46 25 32 46 25 32 46 25 32 46 25 32 46 38 42 45 41 49 61 44 44 63 34 4d 44
    63 77 4e 7a 59 78 4d 7a 41 32 4d 53 49 4d 76 76 73 73 77 64 6a 77 52 4a 72 50 4b 67 4b 79 4b 71 63 44 30 68 6b 49 69 62 43 42 4e 56 64 6d 69 6c 44 68 48 76 68 76 57 76 42 61 31 25 32 46 43 32 54 37 64 38 41 43 72 37 65
    42 53 4c 67 42 66 6c 79 64 4f 50 77 74 47 4b 62 58 77 45 34 75 37 71 53 75 4f 36 76 33 4e 70 59 45 73 6b 4e 49 54 46 5a 76 79 30 75 79 38 42 6a 63 44 35 42 6e 32 7a 7a 4f 75 36 63 46 67 25 32 46 48 38 56 48 63 71 55 57 45
    5a 7a 32 50 55 63 42 43 51 76 69 52 6b 63 63 76 53 6b 6d 42 31 75 50 46 70 6a 33 6e 63 34 4a 4b 6e 47 5a 58 43 64 61 72 4a 6d 62 25 32 42 70 63 34 48 32 6d 48 51 7a 30 25 32 42 36 74 70 78 4a 4d 32 7a 25 32 42 47 70 48 70
    58 7a 56 7a 73 4b 4f 6a 4e 4a 32 71 75 4c 25 32 46 74 55 68 52 68 36 30 65 6c 64 78 36 25 32 42 5a 38 59 46 70 31 63 4c 47 61 6b 53 31 4c 63 33 37 4a 72 61 61 66 66 4a 75 44 70 45 35 4d 6c 53 35 36 25 32 46 79 44 79 48 35
    69 4e 70 49 34 6c 25 32 42 47 5a 38 6f 43 69 71 4c 62 4d 50 4a 77 31 64 41 4b 38 37 51 55 58 61 65 68 30 63 78 6f 73 57 70 4c 32 42 25 32 46 46 5a 42 64 33 47 41 25 32 42 6a 6b 33 62 53 43 59 4a 77 41 42 63 34 4f 36 4f 76
    74 77 56 71 68 39 42 61 45 52 54 53 4d 78 33 44 68 45 71 30 6e 48 58 4b 42 4b 45 5a 6f 36 25 32 42 51 6a 68 69 25 32 46 6b 43 69 6a 5a 56 5a 45 65 51 70 75 42 47 71 51 50 25 32 42 48 54 64 49 62 34 47 49 45 25 32 46 31 49
    51 44 37 54 61 25 32 42 4e 46 4b 7a 65 25 32 42 59 6b 67 4f 72 71 79 74 41 4f 53 35 4f 35 32 36 42 52 4c 35 56 6c 38 4b 76 5a 34 65 64 37 72 63 4d 46 35 55 6b 25 32 42 34 45 51 47 48 56 68 77 44 37 6f 4c 31 78 53 6a 63 33
    76 4b 54 31 56 47 4c 73 6e 25 32 46 74 4c 25 32 42 62 6a 45 6e 4d 35 54 70 65 39 49 4c 69 38 71 55 57 6e 54 4b 45 6f 76 69 25 32 46 58 68 6a 37 55 77 7a 25 32 46 74 33 70 61 71 5a 56 78 63 6b 57 43 58 25 32 46 55 34 34 36
    78 43 30 61 64 6a 70 70 41 70 4a 73 52 65 46 43 57 63 31 55 35 77 32 74 64 44 59 55 51 6a 64 51 32 42 51 37 56 79 58 51 42 55 7a 52 43 68 4a 65 63 30 73 32 25 32 42 4d 50 61 39 6b 4b 51 47 4f 70 55 43 57 78 59 39 44 6e 75
    4d 7a 6f 46 6f 62 68 77 79 72 63 58 71 4f 56 6d 4f 4a 72 68 74 76 42 25 32 46 78 68 55 48 4c 6b 42 33 53 34 65 56 38 42 36 76 58 66 39 33 43 6d 58 56 56 68 53 6c 36 7a 4f 5a 45 70 53 34 53 59 4a 58 73 25 32 42 4b 6c 70
    25 32 42 6a 64 35 71 6e 75 57 4c 6e 49 6f 56 55 25 32 42 39 52 47 77 37 37 51 42 72 58 71 4d 47 38 39 78 43 53 44 71 62 6c 33 46 6c 64 70 45 57 62 31 64 77 33 41 42 6e 44 4f 54 69 71 63 6d 50 6d 69 69 34 35 49 50 61 4e 37
```

For more on SigV4: https://www.youtube.com/watch?v=tPr1AgGkvc4



https://docs.aws.amazon.com/IAM/latest/UserGuide/create-signed-request.html



Hopefully the non-AWS users haven't left yet

This talk *should be** compatible with any service that uses SigV4 authentication, including:

- Google Cloud Storage
- Cloudflare R2
- Backblaze B2
- <your favorite here>

Problem: complex-pattern data in S3

Motivation: Josh's Photo Sharing Service

- Users (**millions**) upload their photos (billions)
- Photos are 1MB up to 1GB
- Photos are private by default
  - original uploader can always see their photo
- Photos can be added to albums (**1->many relationship**)
- Albums can be shared with other users (**1->many relationship**)
- Access to an album can be granted and revoked **at any time**
- No photo needs to be super-popular (no CDN)

What *isn't* complex-pattern data?

- Lots of data ("billions of photos")
- Prefix-based layout by uploader ID
  - Very simple if no sharing is needed
  - `/<user 1>/<photo 1>.jpg`

What is complex-pattern data? (in other words)

- There are too many of them! (users, relationships)
- IAM doesn't know about them!
- Policy changes too fast!

Pivoting to data lakes

	Photo Sharing	**Data Lake**
Who?	End-users	Jobs (seconds -> days)
What code?	Browsers	Arbitrary code
Authentication	username/password	SSO portal
Cookie	HTTP cookie	X.509 certificate (or) JWT
Access method	Browser (HTTP GET)	AWS SDK
Authentication to S3	Query string	Authorization header

Similarities

- The callers run code we do not control
  - We cannot grant them unfettered access to the bucket
- We cannot reshape the data to fit the access problem (e.g. into nice prefixes)
  - and even if we could, shifting PBs of data doesn't sound fun
- We have a database of grants, relating callers to objects
  - the database can change rapidly
- There aren't enough IAM roles to model our callers

How do I secure my data lake?
(and some slightly easier problems, too)

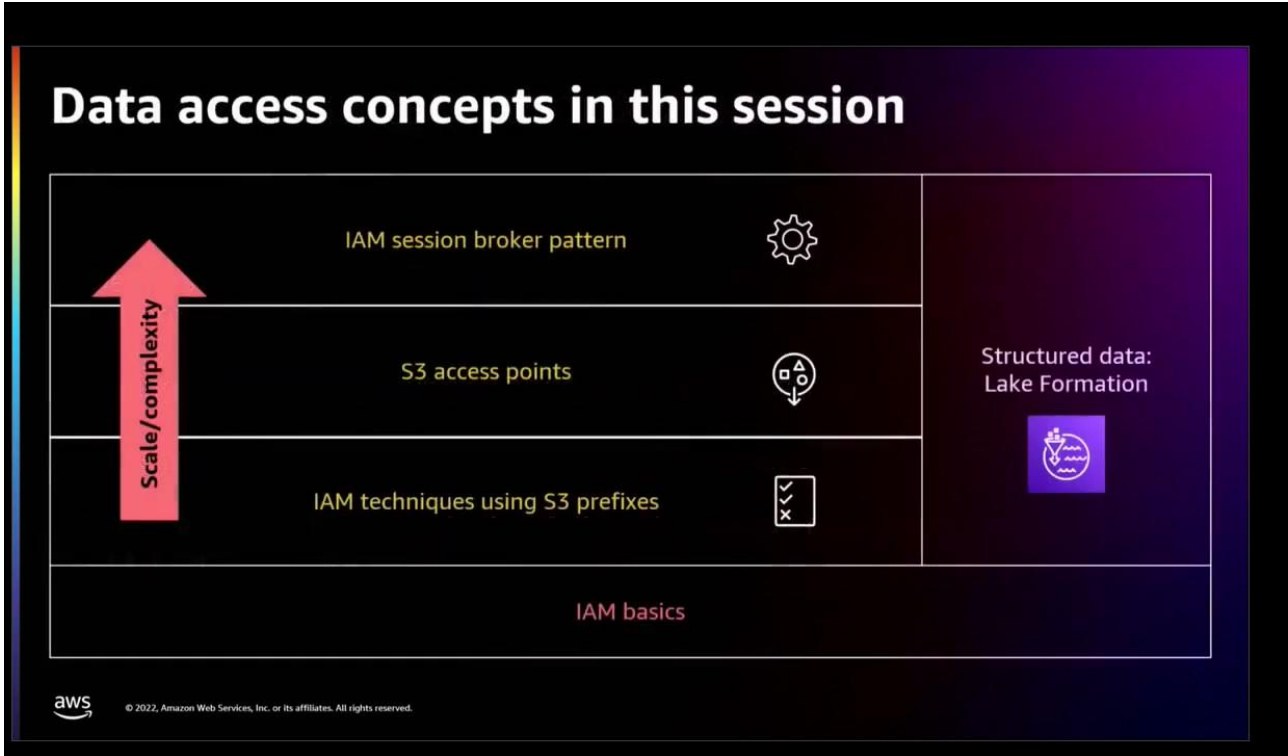
Common solutions
(consider these first)

Overview

- Plain IAM (Weiss @ 5:12)
  - Tagged IAM principals (Weiss @ 19:42)
  - Permissions Boundaries
- Access Points (Weiss @ 25:02)
- STS Session Broker (Weiss @ 40:28)

Overview

See Weiss @ 3:35



"Plain" IAM + S3 Bucket Policy

- IAM is a slow-moving control plane service
- very low rate limit (< 1 mutation / second)
- about 1000 roles, unless you go by account (Weiss @ 19:20)
    - inline policy: max 10 kB
    - attached policy: 6 kB (* 20 policies)
- S3 Bucket Policy (20 kB limit)
    - fits about 30 prefixes (Weiss @ 15:25)

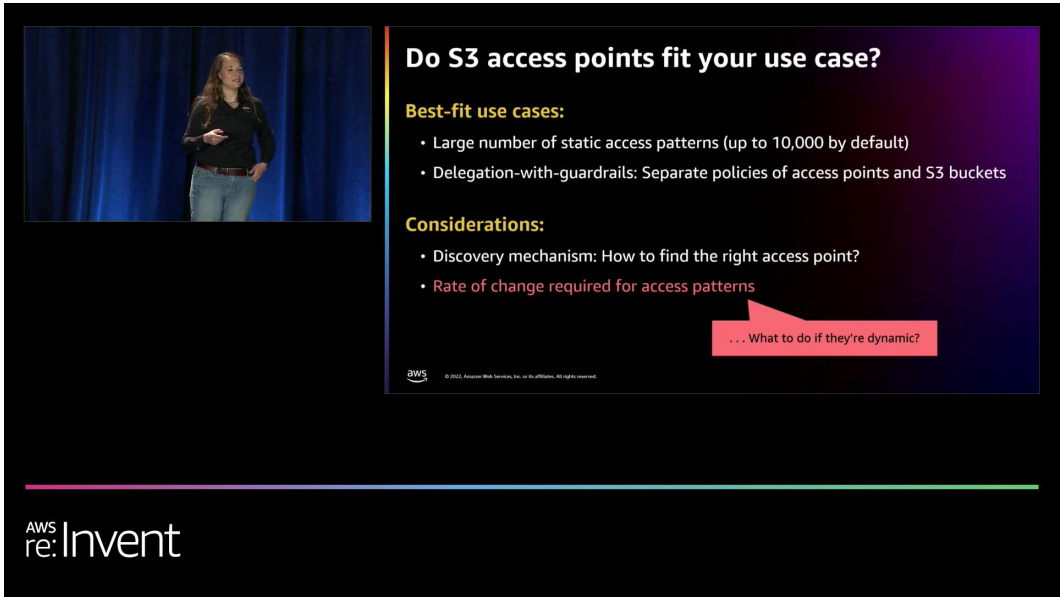
Permissions Boundaries

- Gives an IAM role a way to create new IAM roles with fewer permissions
- Good for a group of services under common ownership
  - example: every Elasticsearch cluster owned by the ES team
- Useful for delegation
  - you delegate a set of permissions to the ES team
  - the ES team creates and manages policies under that boundary, without review by security team
  - the roles can be used directly as IAM instance profiles

Create an S3 Access Point per caller

Weiss @ 34:55

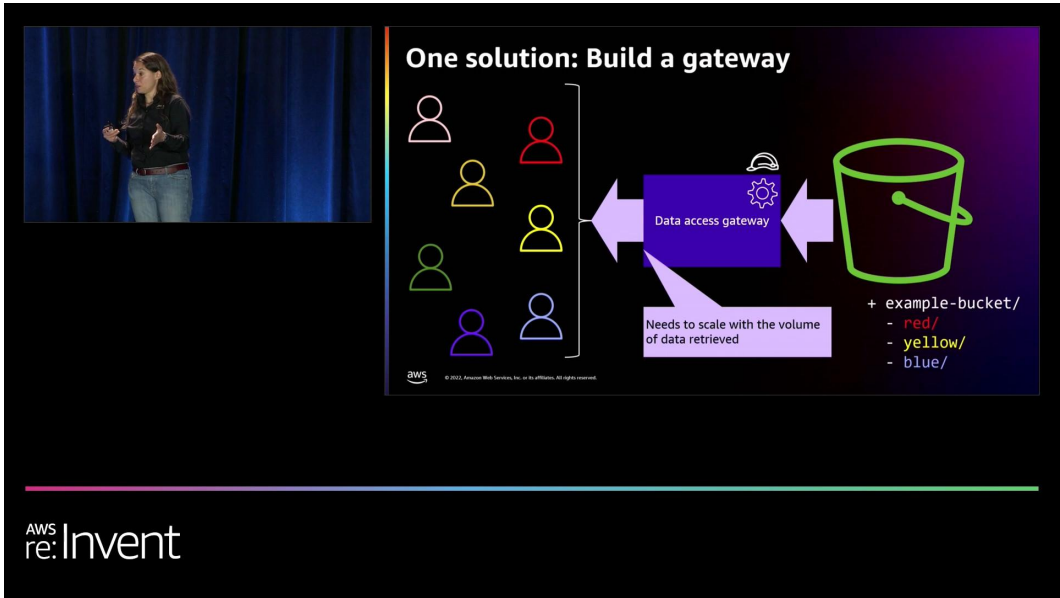
- Multiply out your bucket policy
  - by 10,000 per account-region
- Also useful for delegation
- Doesn't fit dynamic authorization



Let's build a proxy!

Weiss @ 36:05

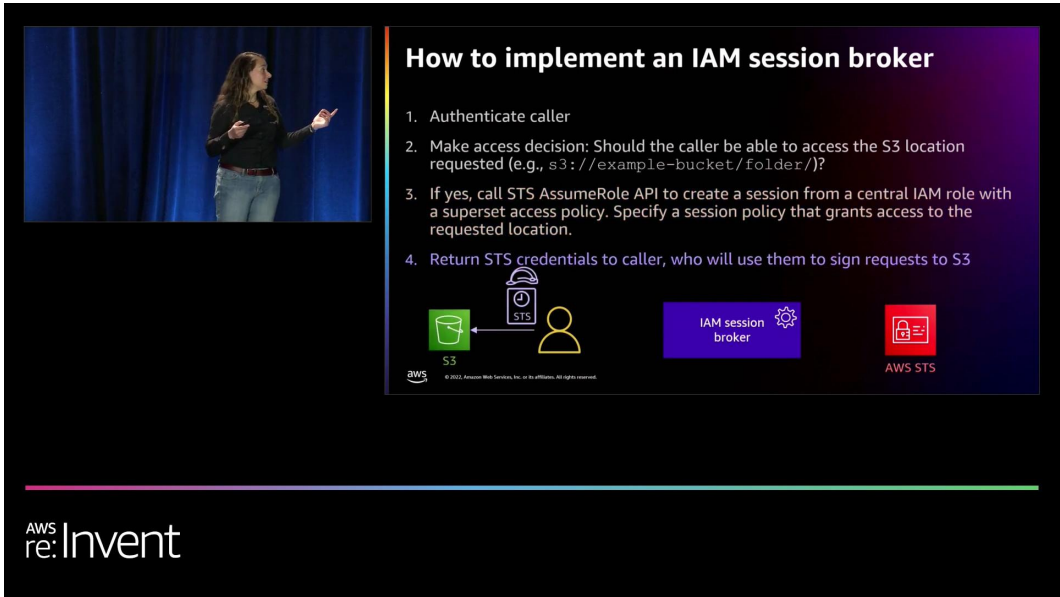
- You can do anything!
  - Build your own authorization layer from scratch
- Spoiler alert: scalability
- S3 can do terabits / second
- Don't do it!



STS Session Broker

Weiss @ 40:28

- sts:AssumeRole creates temporary credentials.
  - can pass `Policy` and `PolicyArn` to downscope the granted role.
- STS is a data plane service



STS Session Broker (drawbacks)

- Limit of 2048 characters in session policy
  - AWS_SESSION_TOKEN
- Account-wide ratelimit
  - Availability concern

Just-in-time authorization with signing

A basic signer

github.com/hashbrowncipher/fwdcloudsec-signers

```
$ curl -s -u josh:password \
  -d '{
       "url":"https://s3.amazonaws.com/permanent/mykey",
       "method": "GET",
       "headers": {}
  }' \
  http://127.0.0.1:8000 | jq "."
{
  "url": "https://permanent-quoic7ui7jhvtjt6.s3.us-west-2.amazonaws.com/josh/mykey",
  "headers": {
       "x-amz-content-sha256": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
       "x-amz-expected-bucket-owner": "111111111111",
       "X-Amz-Date": "20230610T124405Z",
       "Authorization": "AWS4-HMAC-SHA256
Credential=AKIA3LROMZGCV47J47W6/20230610/us-west-2/s3/aws4_request,
SignedHeaders=host;x-amz-content-sha256;x-amz-date;x-amz-expected-bucket-owner,
Signature=7415160f39168fdb4fa47f67702c578ea3bcc97471743ff69933d518103fe473"
  }
}
```

A basic signer

github.com/hashbrowncipher/fwdcloudsec-signers

```
$ curl -s -u josh:password \
  -d '{
       "url":"https://s3.amazonaws.com/permanent/mykey",
       "method": "GET",
       "headers": {}
  }' \
  http://127.0.0.1:8000 | jq "."
{
  "url": "https://permanent-quoic7ui7jhvtjt6.s3.us-west-2.amazonaws.com/josh/mykey",
  "headers": {
       "x-amz-content-sha256": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
       "x-amz-expected-bucket-owner": "111111111111",
       "X-Amz-Date": "20230610T124405Z",
       "Authorization": "AWS4-HMAC-SHA256
Credential=AKIA3LROMZGCV47J47W6/20230610/us-west-2/s3/aws4_request,
SignedHeaders=host;x-amz-content-sha256;x-amz-date;x-amz-expected-bucket-owner,
Signature=7415160f39168fdb4fa47f67702c578ea3bcc97471743ff69933d518103fe473"
  }
}
```

Home directories

```
$ export USER=josh
$ echo $USER > myname; aws s3 cp myname s3://permanent/myname
upload: ./myname to s3://permanent/myname
$ export USER=alice
$ echo $USER > myname; aws s3 cp myname s3://permanent/myname
upload: ./myname to s3://permanent/myname
$ AWS_SKIP_SIGNER=1 aws s3 ls --recursive permanent-quoic7ui7jhvtjt6
2023-06-03 06:05:29          6 alice/myname
2023-06-03 06:05:20          5 josh/myname
$ USER=alice aws s3 cp s3://permanent/myname -
alice
$ USER=josh aws s3 cp s3://permanent/myname -
josh
```

awscli plugin

Use our own credentials (e.g. JWT or x.509) to talk to signer

Use awscli's existing event hooks feature

```python
import os, requests
from botocore.auth import AUTH_TYPE_MAPS, BaseSigner
from requests.auth import HTTPBasicAuth

def make_signing_request(request):
    basic = HTTPBasicAuth(os.environ["USER"], "password")
    resp = requests.post("http://localhost:8000", json=request, auth=basic)
    return resp.json()

class ExternalSigner(BaseSigner):
    REQUIRES_REGION = False

    def __init__(self, credentials):
        pass

    def add_auth(self, request):
        ea_request = dict(
            method=request.method,
            url=request.url,
            params=request.params,
            headers=dict(request.headers),
        )

        response = make_signing_request(ea_request)
        request.url = response["url"]
        request.headers = response["headers"]

def choose_signer(*args, **kwargs):
    return "mysigner"

def awscli_initialize(event_hooks):
    if os.environ.get("AWS_SKIP_SIGNER", "0") != "0":
        return

    AUTH_TYPE_MAPS["mysigner"] = ExternalSigner
    event_hooks.register("choose-signer.s3", choose_signer)
```

Can I change responses?

- Like all things IAM, we can only affect requests, not responses
- Without a proxy, we can't change a response

Home directories: handling ListObjects(V2)

```
$ USER=josh aws s3 ls s3://permanent
2023-06-03 06:05:20          5 myname
$ USER=alice aws s3 ls s3://permanent
2023-06-03 06:05:29          6 myname
```

```
C: Hey 127.0.0.1:8000, could you sign this
ListObjectsV2 call please?
S: Sure! Your signed request is:
  GET /list?...
  Host: 127.0.0.1:8000
  Authorization: Basic am9zaDpwYXNzd29yZA==
C: <sends that request>
S: Here's a ListObjectsV2 response that
makes it look like your home directory is
the only one in the bucket.
```

Can I change responses?

- Like all things IAM, we can only affect requests, not responses
- Without a proxy, we can't change a response
- BUT! We can selectively proxy by rewriting requests:
  - to ourselves (e.g. previous slide)
  - to an S3 object lambda access point
  - to any other HTTP(S) service

How do I integrate this with my data lake?

Assume we're just signing, not modifying the requested bucket or key.

1. (in the caller) add the signer as a shim into your AWS client library
2. (in the signer) authenticate the caller
3. work backwards from key requested to data entity (e.g. table or column)
4. determine whether the caller should have access.
5. sign or reject the request

Does this work with other AWS services?

e.g. DynamoDB

Yes!

…but you might prefer to just use a proxy

Does it multipart?

Yes, most definitely

And you can cache!

Can I run the signer as a lambda?

Yep!

Do I have to feel comfortable running the signer as a service in prod?

Yes!

Should I worry about latency?

Three components:

- Network
- Authorization
  - Caching!
- Signing

Scenario: My bucket is in the "wrong" account

Migration Solution:

1. Reconfigure the app to send all requests to the signer.
2. The signer routes writes into the new bucket.
3. For reads, perform a HEAD on the new bucket
    a. If present, sign a request for the new bucket
    b. If absent, sign a request for the old bucket
4. Copy the data in the background

Features that S3 doesn't have, but signers can simulate

- Atomic rename of one file
- Atomic rename of entire directories
- Symlinking (i.e. alias the most recent version of a blob)
- Hard linking (i.e. instant copy)
- Transparent sharding of "hot" keys
- Time travel queries
- Customized data positioning (for regulatory or lifecycle reason)

Thank you!