Hidden in Plain Sight

Large-Scale Exposure of Dangling Commits on Major Git Platforms

Kumar **Ashwin**

REDHUNT LABS
DISCOVER. ATTACK. REPEAT.

- Leads Research & Consulting at RedHunt Labs
- Dabbles in **Web**, **Cloud** & **Supply Chain Security**
- Speaker/Trainer: **BlackHat, x33fcon, c0c0n, etc.**

@0xCardinal on social platforms

https://kumarashwin.com

What's on the Menu?

What are Dangling Commits?

Why is this a Security Risk?

Analyzing the Impact

Large-Scale Data Insights

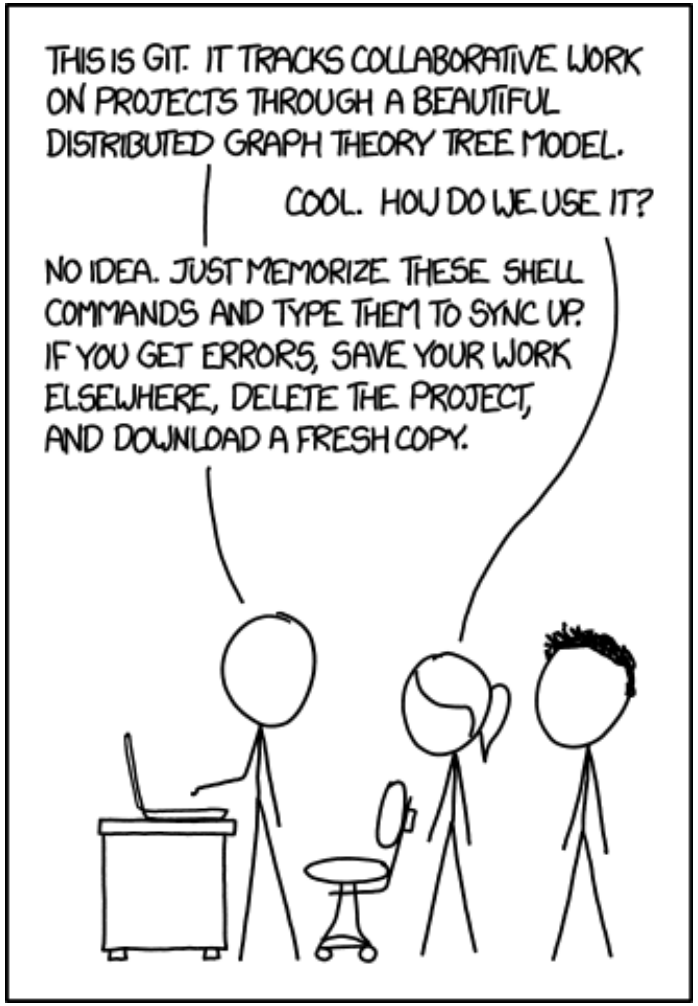
Fixing & Preventing the Issue

Ending Notes

You know git, right?

& git commits?

https://xkcd.com/1296/



How many of you have used git?

How many of you have made commits to a git repository?

How many of you have memorized the git commands?

How many of you have pushed sensitive data to a git repository?

https://xkcd.com/1597/


WHOOPSY DAISY...

How to Delete Git Commits? Consensus is…

1 2 Next

5666

Careful: `git reset --hard` **WILL DELETE YOUR WORKING DIRECTORY CHANGES**. Be sure to **stash any local changes you want to keep** before running this command.

Assuming you are sitting on that commit, then this command will wack it...

```
git reset --hard HEAD~1
```

The `HEAD~1` means the commit before head.

Or, you could look at the output of `git log`, find the commit id of the commit you want to back up to, and then do this:

```
git reset --hard <sha1-commit-id>
```

If you already pushed it, you will need to do a force push to get rid of it...

```
git push origin HEAD --force
```

https://stackoverflow.com/questions/1338728/how-do-i-delete-a-commit-from-a-branch

Deleting a Commit from a Remote Repository

Deleting a commit from a remote repository can be dangerous, as it always requires rewriting the Git history of the target branch. Proceed with caution and communicate with your other collaborators prior to running any destructive commands.

Deleting the most recent commit

If you've pushed a commit to a remote repository and want to remove it:

1. **First delete the commit locally by running:**

   ```
   git reset --hard HEAD~1
   ```

   This command deletes the most recent commit from your local repository, and **discards the changes** from that commit.

2. **Force push to remote:**

   ```
   git push origin <branch-name> --force
   ```

   Replace `<branch-name>` with your current branch. This overwrites the remote branch with your local branch,

https://graphite.dev/guides/how-to-delete-a-git-commit

If you'd like to delete the commits up to a specific commit, run `<git log>` into the command line to find the specific commit id, and then run the following command:

```
git reset --hard <sha1-commit-id>
```

This command discards all working tree changes and moves HEAD to the commit chosen.

Deleting commits already pushed

Alternatively, if your changes are **already pushed**, simply run the following command:

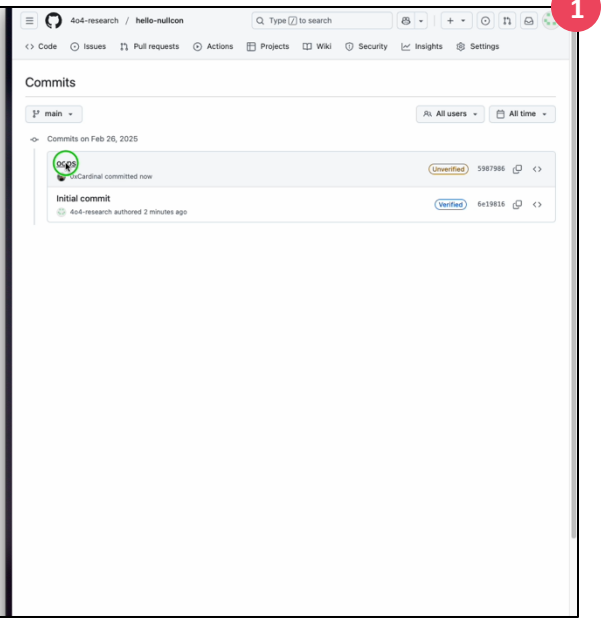
```
git push origin HEAD --force
```

https://articles.assembla.com/en/articles/2941346-how-to-delete-commits-from-a-branch-in-git

What if I told you that deleted commits don’t always disappear?

Don’t believe me, look for yourself…

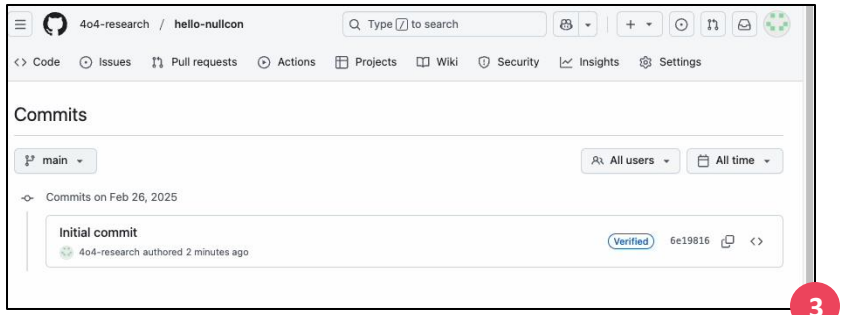
```
→  hello-nullcon git:(main) ✗ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will b
e committed)
        secret.gif

nothing added to commit but untracked files present
(use "git add" to track)
→  hello-nullcon git:(main) ✗ git add .
→  hello-nullcon git:(main) ✗ git commit -am "oops"
[main 5987986] oops
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 secret.gif
→  hello-nullcon git:(main) git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 37.57 KiB | 18.78 MiB/s
, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To research:4o4-research/hello-nullcon.git
   6e19816..5987986  main -> main
→  hello-nullcon git:(main)
```



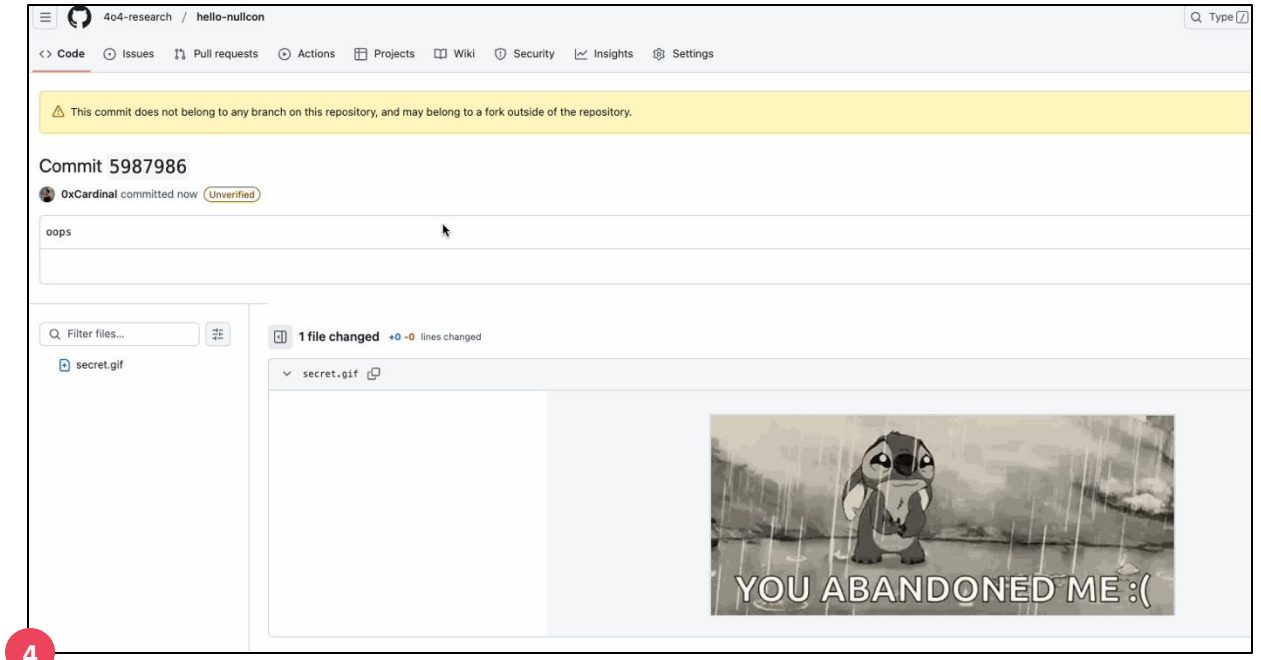
1

```
→  hello-nullcon git:(main) git reset HEAD~1 --hard
HEAD is now at 6e19816 Initial commit
→  hello-nullcon git:(main) git push --force
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To research:4o4-research/hello-nullcon.git
 + 5987986...6e19816 main -> main (forced update)
→  hello-nullcon git:(main)
```

2



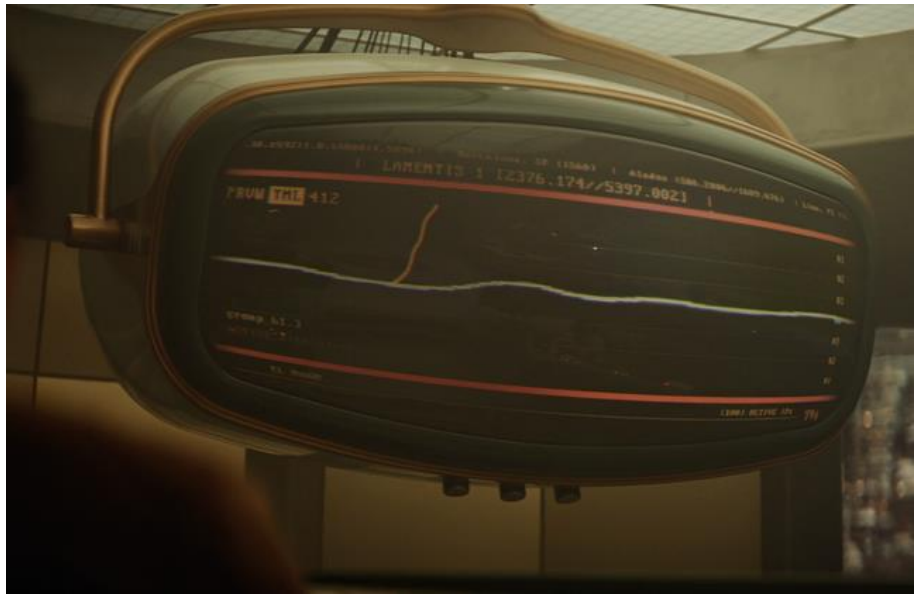
3

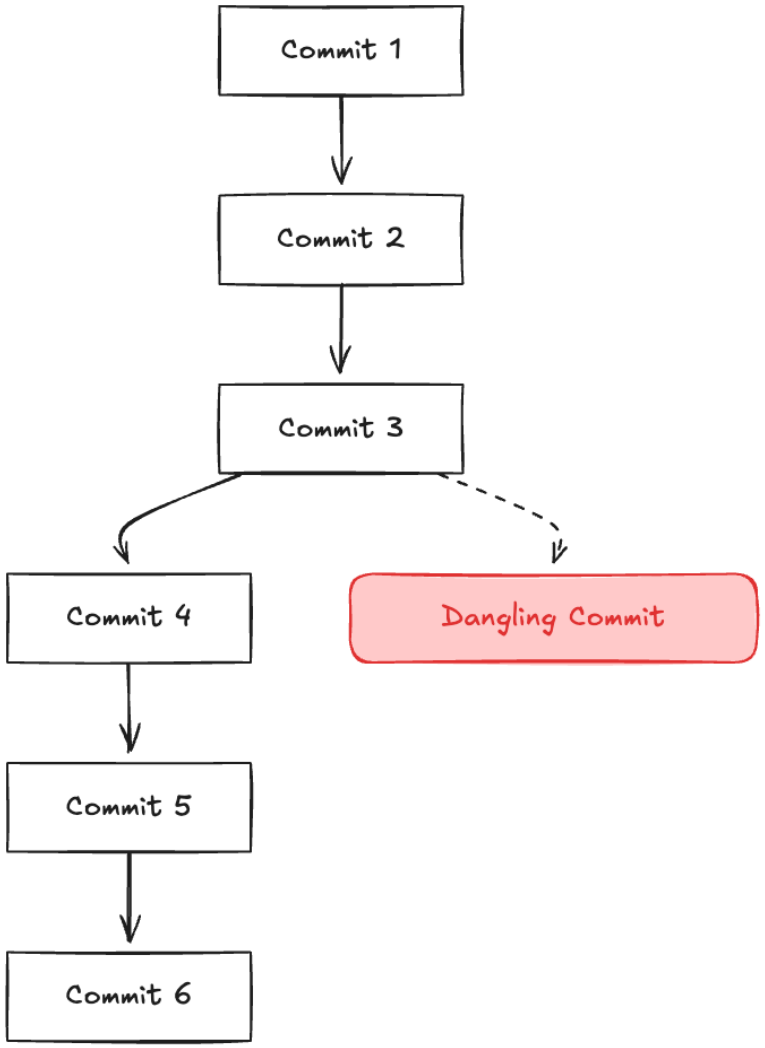


4

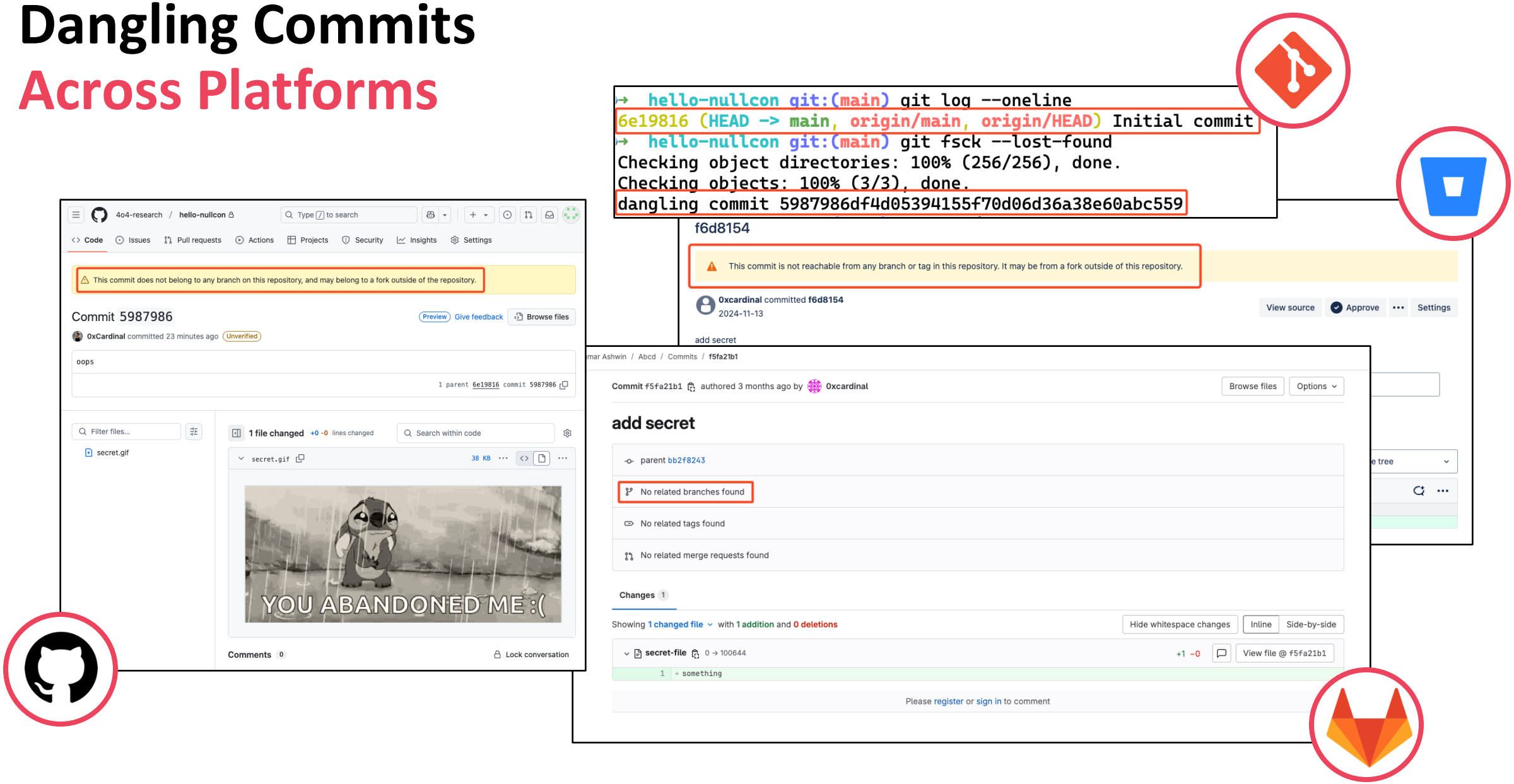
What is a Dangling Commit?

- A commit that exists but is **not connected to any branch**.
- Git doesn't delete it immediately—it keeps it… just in case.





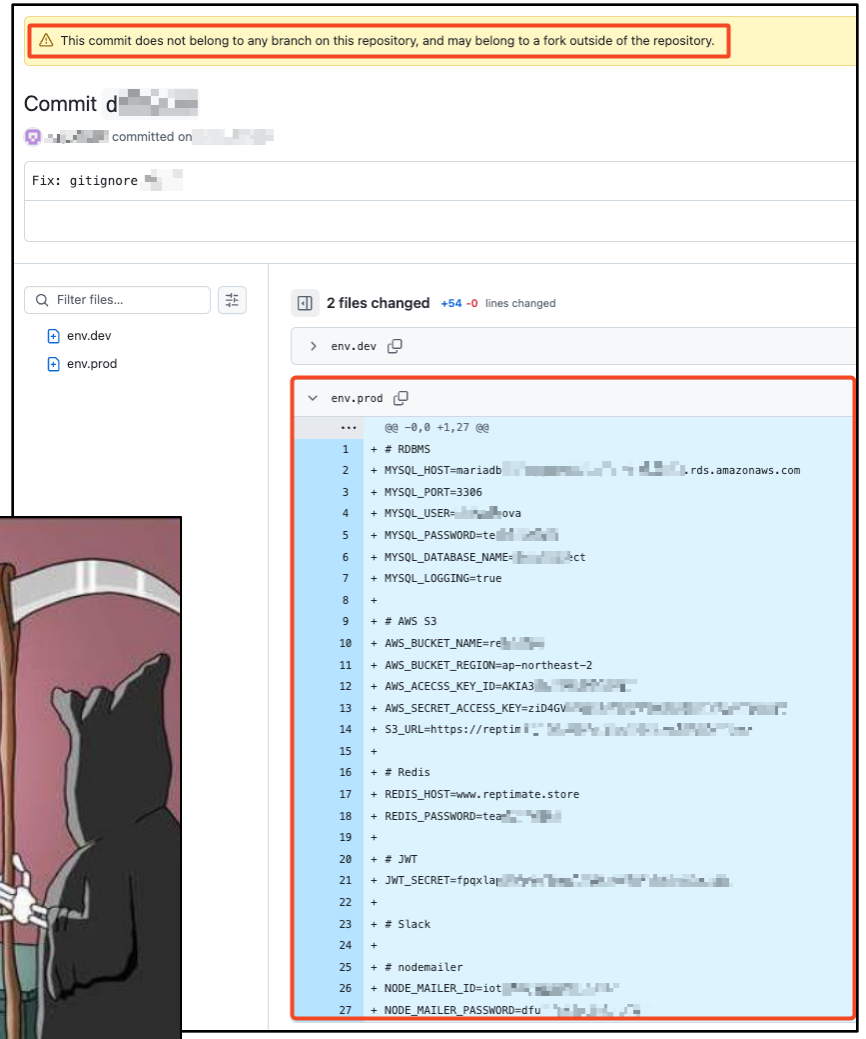
Dangling Commits
Across Platforms



Why Is a Dangling Commit an Issue?

- When you remove something, you don't want it there...
- Maybe a mistake, or just cleaning things up.
- Or… you're trying to hide something? 😉

But Git doesn't forget – and dangling commits can be a **gold mine** of secrets.



We reached out to Git platforms about this issue.

They have no plans to fix it but shared guidance on manual removal.

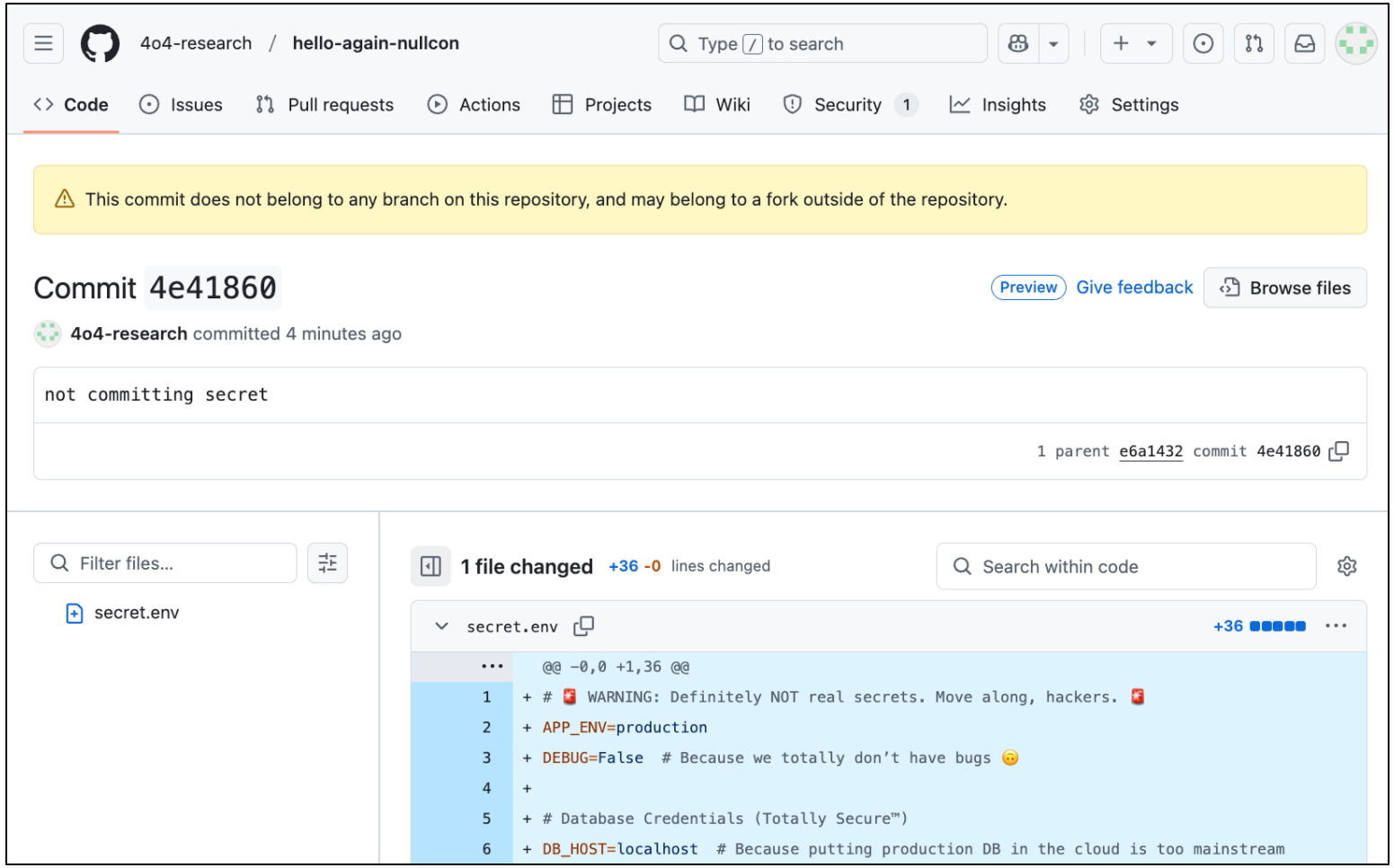
https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository

Cool, we have a solution now, let's try that!

```
→  hello-again-nullcon git:(main) git-filter-repo --sensitive-data-removal --invert-paths --path secret.en
v --force
NOTICE: Fetching all refs from origin to make sure we rewrite
        all history that may reference the sensitive data, via
      git fetch -q --prune --update-head-ok --refmap "" origin +refs/*:refs/*
Parsed 2 commits
New history written in 2.80 seconds; now repacking/cleaning...
You rewrote 1 (of 2) commits.

NOTE: First Changed Commit(s) is/are:
  4e4186057f53818fa8fe3c3a1ca9573a1bc4ca79
NOTE: LFS object orphaning not checked (LFS not in use)

Repacking your repo and cleaning out old unneeded objects
HEAD is now at e6a1432 Initial Commit
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Completely finished after 2.90 seconds.
```

```
→  hello-again-nullcon git:(main) git push --force --mirror origin

Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 234 bytes | 234.00 KiB/s, done.
Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
To research:4o4-research/hello-again-nullcon.git
 + 4e41860...e6a1432 main -> main (forced update)
```

P.S. - **BFG Repo-Cleaner** can also be used.


4o4-research / hello-again-nullcon
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security 1
Insights
Settings
This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.
Commit 4e41860
Preview
Give feedback
Browse files
4o4-research committed 4 minutes ago
not committing secret
1 parent e6a1432 commit 4e41860
Filter files...
secret.env
1 file changed +36 -0 lines changed
Search within code
secret.env
+36
@@ -0,0 +1,36 @@
+ # 🚨 WARNING: Definitely NOT real secrets. Move along, hackers. 🚨
+ APP_ENV=production
+ DEBUG=False # Because we totally don't have bugs 🙃
+
+ # Database Credentials (Totally Secure™)
+ DB_HOST=localhost # Because putting production DB in the cloud is too mainstream


HOPE YOU ENJOYED MY PRESENTATION
THANK YOU FOR LISTENING

How do we remove the dangling commit then?

✅ **Option 1: Contact GitHub Support**

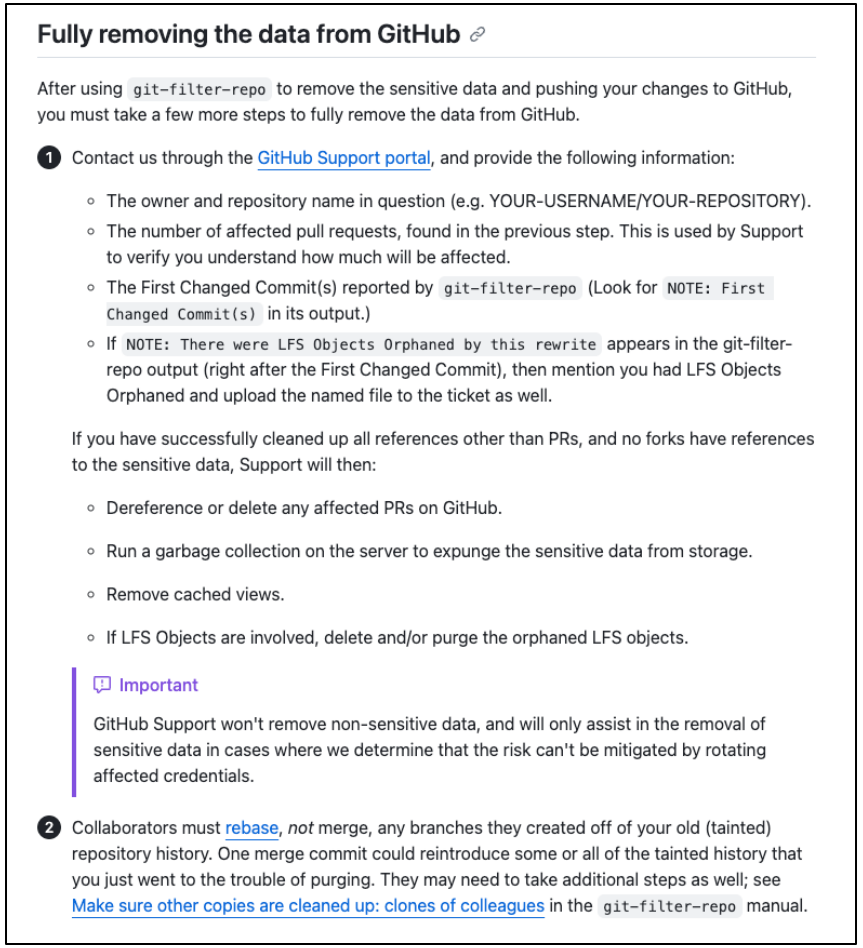
💬 Submit a request and hope for the best!

✅ **Option 2: Delete the Repository**

🗑️ If it's gone, it's gone… right? (Maybe… 🙃)

✅ **Option 3: Make the Repository Private**

🔒 Out of sight, out of mind?



Fully removing the data from GitHub

After using `git-filter-repo` to remove the sensitive data and pushing your changes to GitHub, you must take a few more steps to fully remove the data from GitHub.

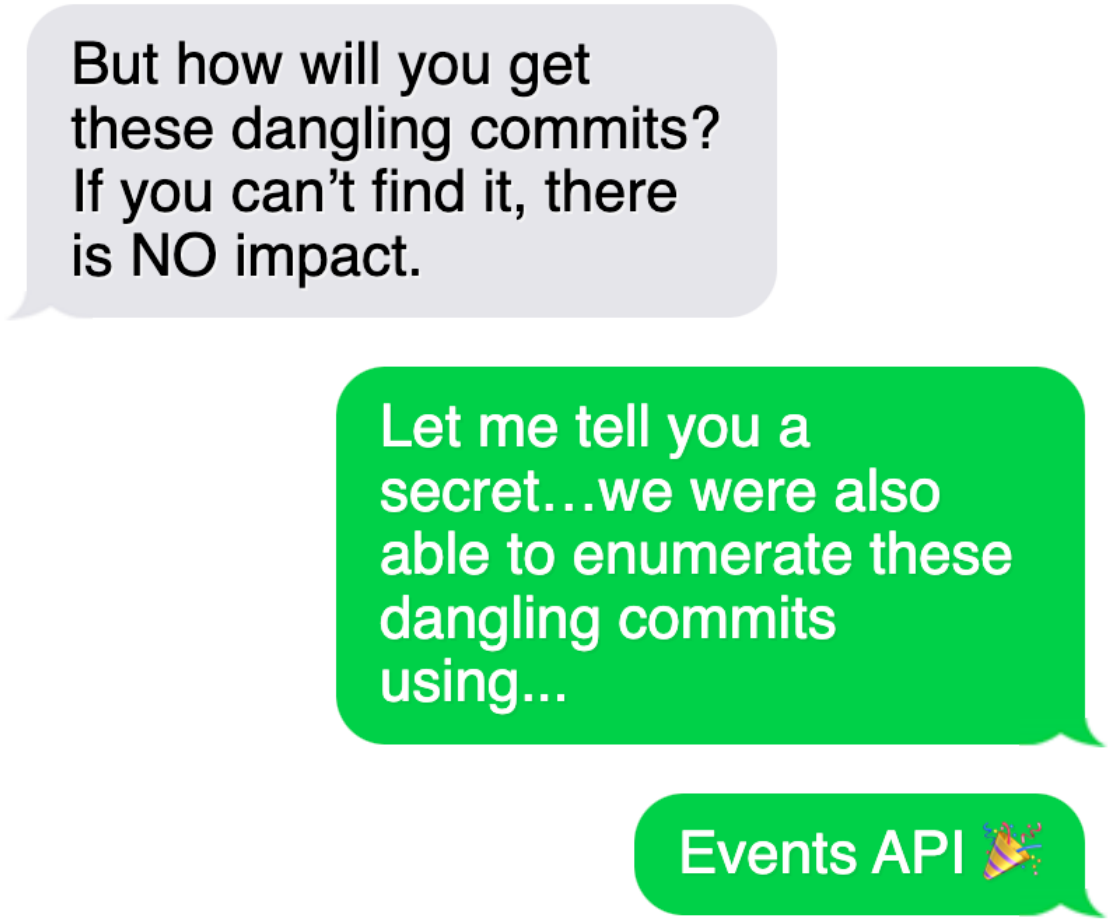
1. Contact us through the GitHub Support portal, and provide the following information:
   - The owner and repository name in question (e.g. YOUR-USERNAME/YOUR-REPOSITORY).
   - The number of affected pull requests, found in the previous step. This is used by Support to verify you understand how much will be affected.
   - The First Changed Commit(s) reported by `git-filter-repo` (Look for `NOTE: First Changed Commit(s)` in its output.)
   - If `NOTE: There were LFS Objects Orphaned by this rewrite` appears in the git-filter-repo output (right after the First Changed Commit), then mention you had LFS Objects Orphaned and upload the named file to the ticket as well.

   If you have successfully cleaned up all references other than PRs, and no forks have references to the sensitive data, Support will then:

   - Dereference or delete any affected PRs on GitHub.
   - Run a garbage collection on the server to expunge the sensitive data from storage.
   - Remove cached views.
   - If LFS Objects are involved, delete and/or purge the orphaned LFS objects.

   > Important
   >
   > GitHub Support won't remove non-sensitive data, and will only assist in the removal of sensitive data in cases where we determine that the risk can't be mitigated by rotating affected credentials.

2. Collaborators must rebase, *not* merge, any branches they created off of your old (tainted) repository history. One merge commit could reintroduce some or all of the tainted history that you just went to the trouble of purging. They may need to take additional steps as well; see Make sure other copies are cleaned up: clones of colleagues in the `git-filter-repo` manual.

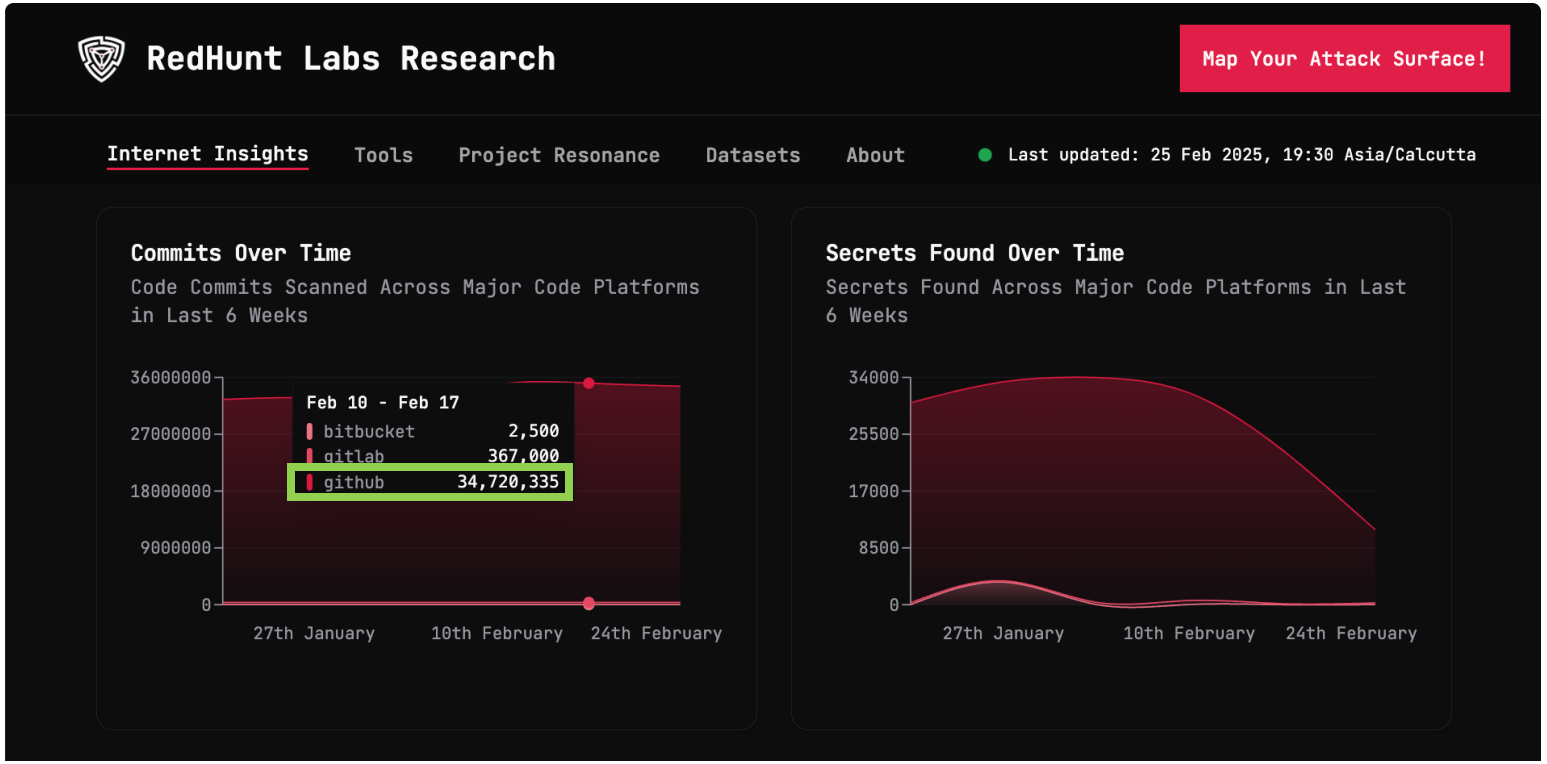

But how will you get these dangling commits? If you can't find it, there is NO impact.
Let me tell you a secret…we were also able to enumerate these dangling commits using...
Events API 🎉

Why GitHub? Understanding Our Dataset

~ 5 years

Sample Data Timeline

All insights are drawn specifically from **GitHub commit events**.



https://research.redhuntlabs.com

GitHub Events


```
{
  "id": "46979116854",
  "type": "PushEvent",
  "actor": {
    "id": 188773812,
    "login": "4o4-research",
    "display_login": "4o4-research",
    "gravatar_id": "",
    "url": "https://api.github.com/users/4o4-research",
    "avatar_url": "https://avatars.githubusercontent.com/u/188773812?"
  },
  "repo": {
    "id": 939329639,
    "name": "4o4-research/hello-again-nullcon",
    "url": "https://api.github.com/repos/4o4-research/hello-again-nullcon"
  },
  "payload": {
    "repository_id": 939329639,
    "push_id": 22869189109,
    "size": 0,
    "distinct_size": 0,
    "ref": "refs/heads/main",
    "head": "69cca5b86242799d3dbcfc53bc7ea5a9a2b9154a",
    "before": "a8c2afe8e86147366c120be2a01fd3a1334e5635",
    "commits": []
  },
  "public": true,
  "created_at": "2025-02-26T12:28:47Z"
},
```


https://api.github.com/repos/4o4-research/hello-again-nullcon/events

The dangling commit IDs can be enumerated from the GitHub Events API –

- **Repository Events**
- **User Events**
- **Public Events**

Let's Talk Patterns – Key Observations

- **Force push** is often used to hide these events.
- **Push events with no commits** are a strong indicator.
- But… not all matching patterns are dangling commits!

- Such an event occurs during multiple git actions -
  - Force Push with No New Commits
  - Branch Deletion
  - Tag Creation or Deletion
  - Empty Push
  - Repository Initialization with No Files
  - Protected Branch with Push Rejected
  - Merge a Pull Request?

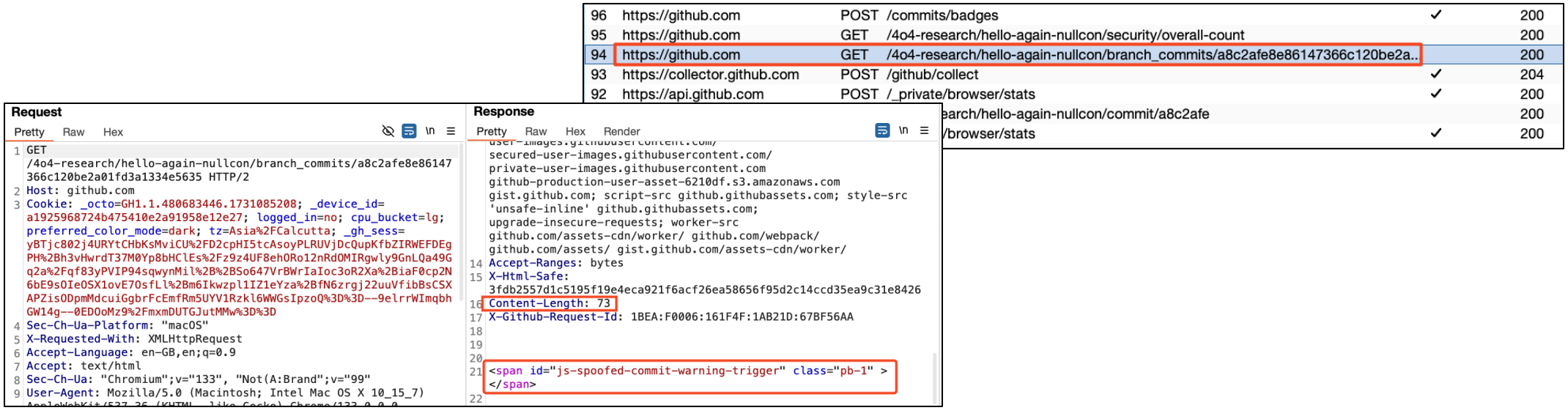

```
{
  "id": "98765432101",
  "type": "PushEvent",
  "actor": {
    "id": 56789012,
    "login": "cyber_ninja",
    "display_login": "cyber_ninja",
    "gravatar_id": "",
    "url": "https://api.github.com/users/cyber_ninja",
    "avatar_url": "https://avatars.githubusercontent.com/u/56789012?"
  },
  "repo": {
    "id": 1122334455,
    "name": "cyber_ninja/stealth_project",
    "url": "https://api.github.com/repos/cyber_ninja/stealth_project"
  },
  "payload": {
    "repository_id": 1122334455,
    "push_id": 87654321098,
    "size": 0,
    "distinct_size": 0,
    "ref": "refs/heads/main",
    "head": "abcdef1234567890abcdef1234567890abcdef12",
    "before": "123456abcdef7890123456abcdef7890123456ab",
    "commits": []
  },
  "public": true,
  "created_at": "2025-02-26T10:30:45Z"
}
```

How Do We Filter the Noise?

- We analysed GitHub's **branch_commits** endpoint.

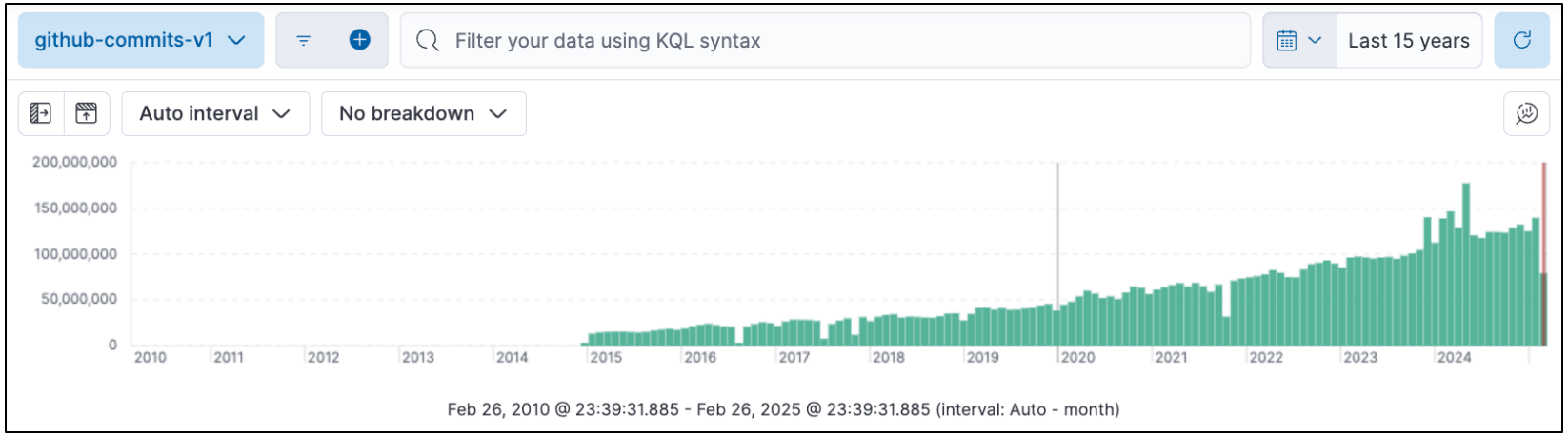
  https://github.com/USER/REPO/branch_commits/COMMIT_ID

- **If a commit isn't linked to any branch → It's dangling!**
- This is the same method GitHub uses internally.

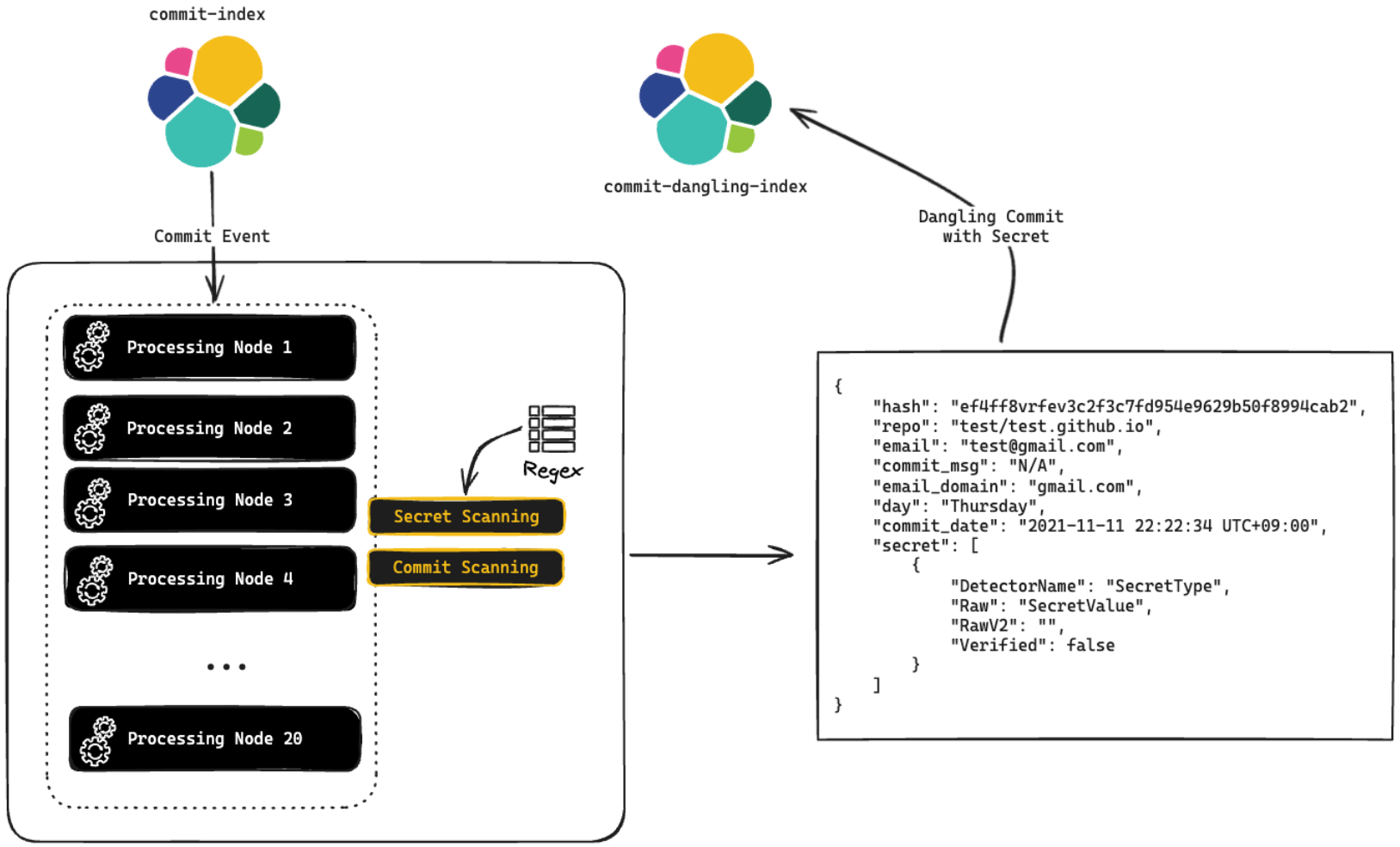


The Missing Ingredient: Lots of GitHub Events

- GitHub **only retains event history for 30 days**
- To **truly measure the impact**, we need **long-term data**
- We needed **more than just 30 days**… we needed **1500+ days** as our sample data.
- At RedHunt Labs, we have maintained a **historical dataset of commits spanning 10+ years** and this data became invaluable in uncovering the real scope of dangling commits.

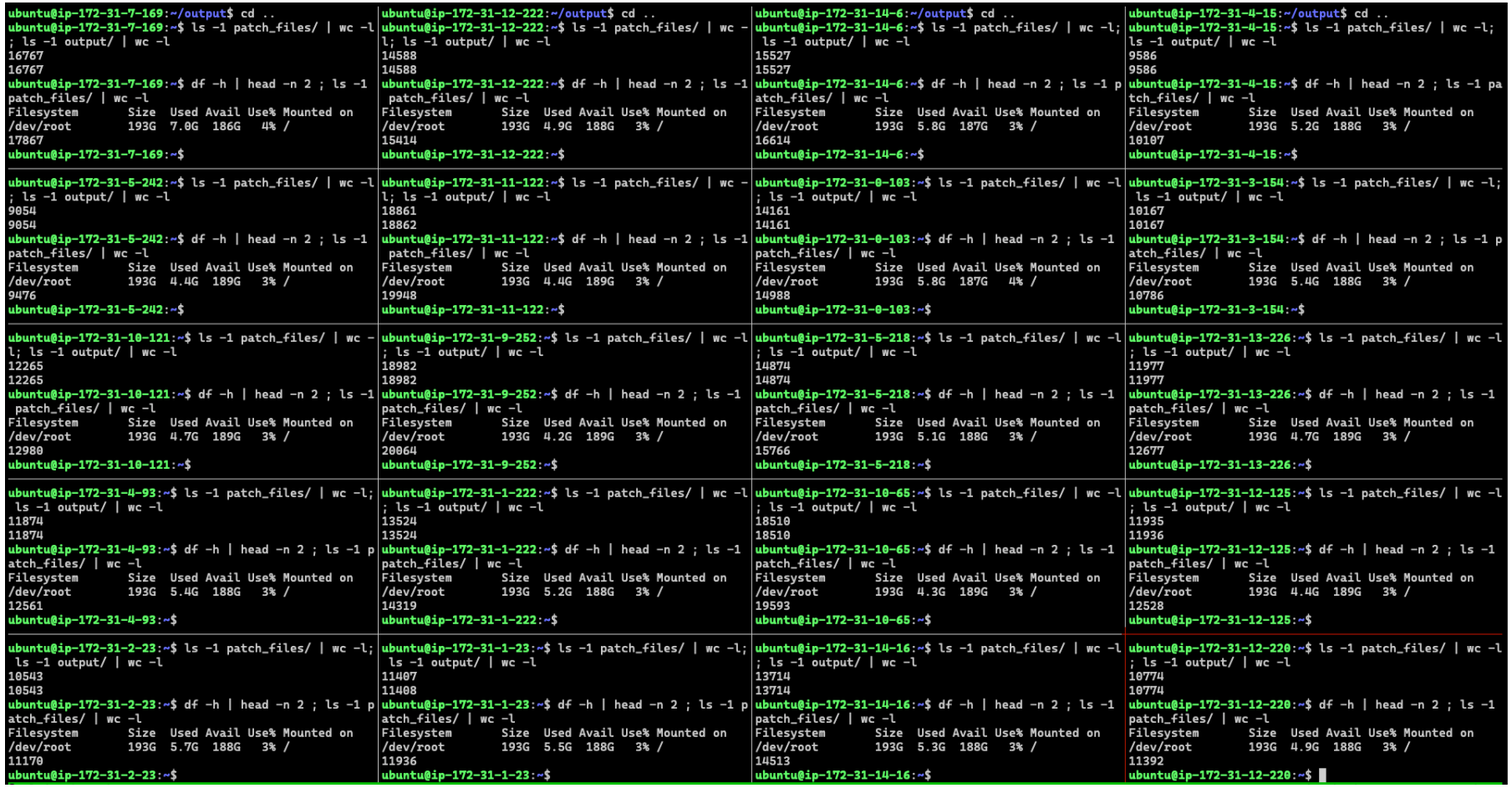


Some Engineering …



REDHUNT LABS
DISCOVER. ATTACK. REPEAT.

… and some Jugaad



These were the ingredients chosen to create the perfect dangling commit detection at scale!

We uncovered the real impact of dangling commits! 

~ 5,400,000,000

GitHub Commits

~ 66,000,000

Dangling GitHub Commits

~ 550,000

Secrets Found in Dangling Commits

???

What **day of the week** sees the **most dangling commits**?

Friday

What **day of the week** sees the **most dangling commits**?

Top 10 values of day	% Secrets in Dangling Commits
Friday	22%
Monday	20%
Wednesday	18%
Tuesday	15%
Thursday	15%
Sunday	6%
Saturday	5%

snyk.io

naver.com

qq.com

checkmarx.com

163.com

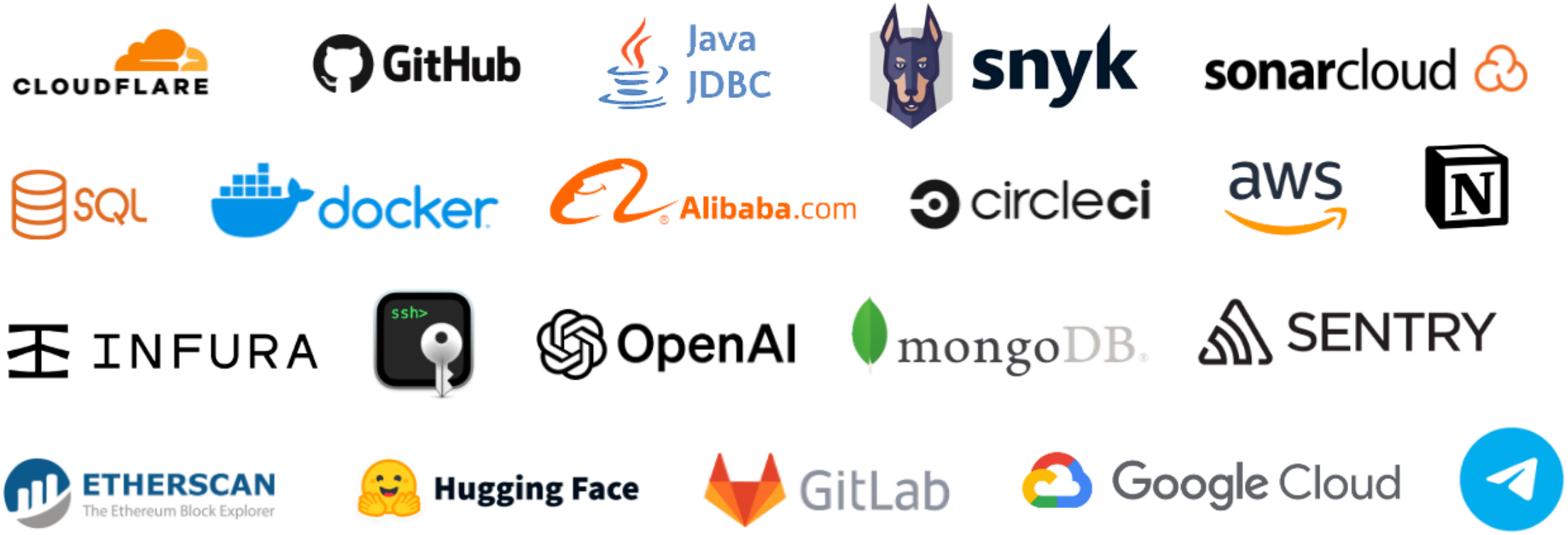
redhat.com

Top Companies Responsible for the Most Dangling Commits

```
git commit -am “Saving release notes”
git commit -am “---”
git commit -am “N/A”
git commit -am “release: 4.0.0”
git commit -am “auto pull”
git commit -am “added chatgpt data”
git commit -am “test”
```

Top Commit Messages for Dangling Commits

Top Types of Secrets Found



The State of Dangling Commits & Secrets Exposure

~ 5,400,000,000

GitHub Commits

~ 66,000,000

Dangling GitHub Commits

~ 550,000

Secrets Found in Dangling Commits

Friday

What **day of the week** sees the **most dangling commits**?

```
git commit -am "Saving release notes"
git commit -am "---"
git commit -am "N/A"
git commit -am "release: 4.0.0"
git commit -am "auto pull"
git commit -am "added chatgpt data"
git commit -am "test"
```

Top Commit Messages for Dangling Commits

snyk.io
naver.com
qq.com
checkmarx.com
163.com
redhat.com

Top Companies Responsible for the Most Dangling Commits

Top **Types of Secrets** Found

CLOUDFLARE, GitHub, Java JDBC, snyk, sonarcloud, SQL, docker, Alibaba.com, circleci, aws, Notion, INFURA, ssh, OpenAI, mongoDB., SENTRY, ETHERSCAN The Ethereum Block Explorer, Hugging Face, GitLab, Google Cloud


For sure, this is not an issue, right?
RIGHT?
imgflip.com

How Do We Avoid Creating Dangling Commits?

🔧**Technical Solution**

Host Your Own Git Management System

- Provides full control over commit policies.
- Costly and complex to scale.
- Risk persists outside your controlled repositories – public, forks, repositories.

Automated Detection & Monitoring

- Use detection logic & scanners to flag issues.
- Monitor commit history for force-pushes and deletions.

Immediate Secret Rotation

- If a secret key is pushed, **rotate it immediately** to prevent exposure.

How Do We Avoid Creating Dangling Commits?

👥 Human Factors & Awareness

Developer Awareness & Training

- Educate developers on the risks of dangling commits.
- Implement secure commit practices in workflows.

Encouraging Secure Commit Hygiene

- Adopt best practices in both **open-source** and **enterprise** projects.
- Enforce pre-commit hooks, signed commits, and branch protection policies.

Thanks!

For any queries, reach out to **research@redhuntlabs.com**

@0xCardinal on social platforms https://kumarashwin.com