Technologic Papers

איך בונים פרימטר בענן

מאת יהב פסטינגר

הקדמה

כיום גופים רבים בוחרים שלא להוציא את המידע הרגיש שלהם למרחב הענן הציבורי מכיוון שבניגוד לרשתות הפנימיות שבהם הפרימטר (האיזור שחוצץ בין הרשת הפנימית לרשת הציבורית) נתון לפיקוח והגבלות רבות, בענן הציבורי הפרימטר אינו מוגדר היטב והסיכוי לטעות קונפיגורציה שעלולה לשבור את הפרימטר גבוהה בהרבה.

ספקיות הענן הבינו את גודל הבעיה וכדי לתת מענה לצורך הזה הכריזה AWS בשנת 2022 על הרעיון של Data Perimeter. הרעיון פשוט - ליצור איזור (בהמשך נבין איך נראה האיזור הזה) שבו יאוחסן המידע הרגיש. המטרה שלנו להחיל את כלל הפוליסות על האיזור הזה בשביל למנוע גישה של תוקף למידע. באמצעות סט של חוקים שמאפשרים רמת גרנולריות גבוהה ביותר נוכל למנוע תרחישים מורכבים שתוקפים עושים בהם שימוש ביום-יום.

החוזק המשמעותי של הקונספט הוא שבניגוד להרבה מכלי ההגנה המוצעים כיום בשוק שעובדים במוד של גילוי ולאו דווקא מניעה, הרעיון פה הוא למנוע את הגישה של התוקף למידע בזמן אמת כך שגם אם הצליח להגיע ליכולות משמעותיות ברשת שלנו, עדיין נוכל למנוע ממנו להדליף את המידע החוצה.

אז מה זה AWS Data Perimeter

הרעיון מדבר על כמה רבדים שונים מהם מורכב הארגון:

1. הפן הרשתי - הרשת ממנה אני עובד ביומיום (זו יכולה להיות רשת פרטית או רשת עם כתובת IP פומבית).
2. הפן הזהותי - מיהי הזהות שמבצעת את הפעולה. לדוגמא, משתמש SSO שמצומד ל-Role מסוים או IAM User. לכל זהות מצומדת סט של הרשאות שמגדיר את הפעולה אותן היא יכולה לבצע.
3. מה הזהות שמצומדת לי. לכל זהות מצומד סט של הרשאות שמגדיר את הפעולות אותן אני יכול לבצע.

4. הפן המשאבי - הענן מורכב מאוסף של שירותים שבכל שירות יש מבחר של משאבים (resources). לדוגמא, תחת השירות S3 שמהווה את שירות האחסון המרכזי ב-AWS ישנו משאב של Bucket שמייצג את יחידת האחסון הבסיסית. אל ה-Bucket נוכל לכתוב קבצים באמצעות סט של פעולות (APIs).

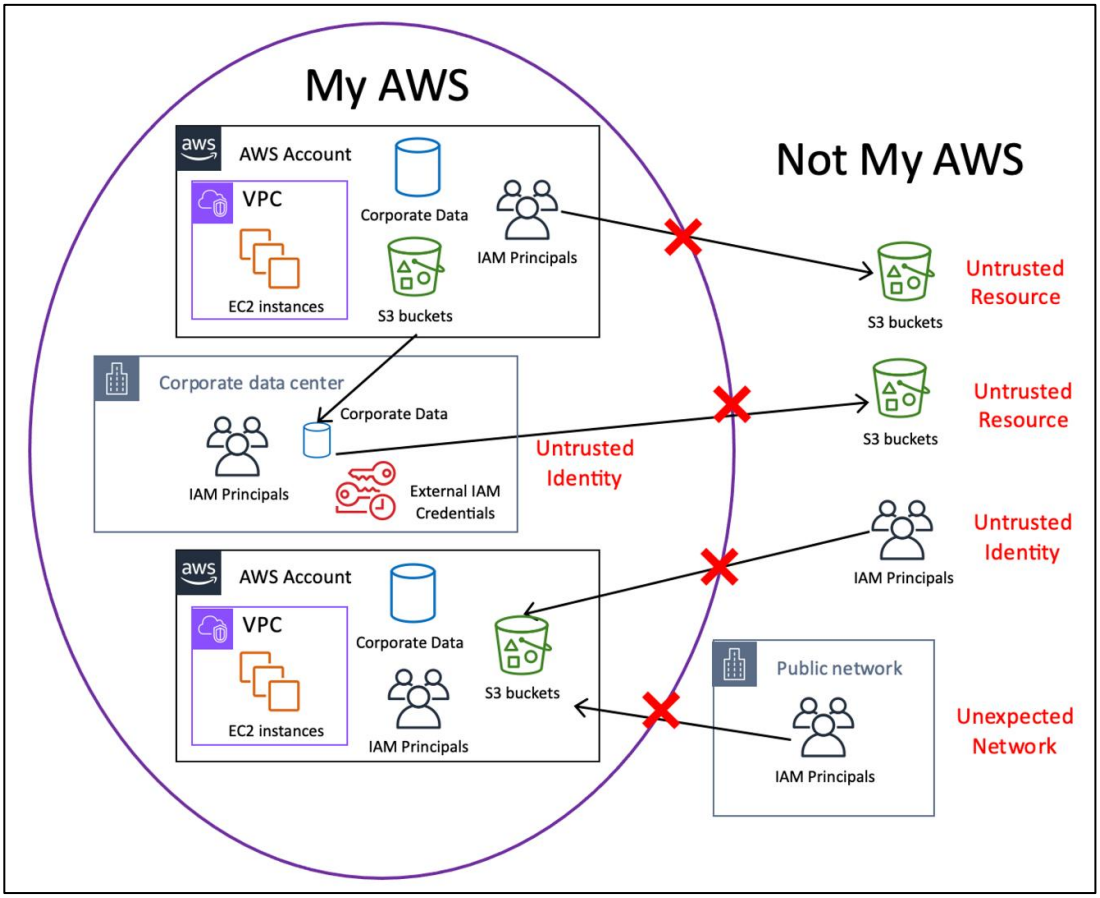
אלה הקומפננטות הבסיסיות שעליהן נרצה להחיל את סט החוקים שלנו. הרעיון פשוט מאוד ומגדיר את החוקים הבאים:

1. בפן הרשתי
   a. מהרשת של הארגון שלי ניתן לבצע פעולות רק באמצעות זהות של הארגון שלי
   b. מהרשת של הארגון שלי ניתן לבצע פעולות רק על משאבים של הארגון שלי
2. בפן הזהותי
   a. רק זהויות של הארגון שלי יכולות לגשת למשאבים של הארגון שלי
   b. זהות ארגונית יכולה לבצע פעולות אך ורק מהרשת שלי
3. בפן המשאבי
   a. ניתן לגשת למשאבים שלי אך ורק באמצעות זהות של הארגון שלי
   b. ניתן לגשת למשאבים שלי אך ורק מהרשת של הארגון שלי

בואו ננסה להבין מהם תרחישי התקיפה הרלוונטיים ולמפות בינם לבין החוק שמתמודד איתם:

1. תוקף שנמצא מחוץ לרשת שלי מנסה לגשת למשאבים שלי באמצעות ה-credentials (הטוקן איתו אני מבצע פעולות בענן) שלו.
   a. במקרה זה חוק 3.b ימנע את הגישה (מכיוון שהתוקף מנסה לגשת למשאב שלי שלא מהרשת של הארגון שלי)
2. תוקף גנב credentials ומנסה לפנות באמצעותם למשאב של הארגון שלי. דוגמא לתרחיש הזה: בשנת 2019 תוקף פרץ לענן של Capital One והשיג credentials לחשבון שבו היה מאוחסן מידע על גבי Bucket, ולאחר מכן ניגש לאותו ה-Bucket מהרשת הפרטית שלו (תוכלו למצוא פה חקירה מפורטת של אירוע התקיפה).
   a. חוק 2.b ימנע את הגישה (מכיוון שהתוקף מבצע פעולה עם הזהות שלי שלא מהרשת של הארגון שלי)
   b. חוק 3.a ימנע את הגישה (מכיוון שהתוקף מנסה לגשת למשאב של הארגון שלי שלא מהרשת של הארגון שלי)
3. תוקף השיג אחיזה ברשת שלי ומנסה לפנות למשאב שלו באמצעות credentials שלו. דוגמא לתרחיש הזה: בשנת 2022 מייקרוסופט זיהתה קבוצת תקיפה איראנית שמשתמשת בשירות OneDrive כשרת C&C (Command and Control) באמצעות credentials שהביאו מהסביבה העננית שלהם (תוכלו למצוא מידע מלא על התרחיש פה).
   a. חוק 1.a ימנע את הגישה (מכיוון שהתוקף מנסה לבצע פעולה מהרשת שלי שלא באמצעות הזהות של הארגון שלי)

ישנם עוד תרחישי תקיפה שנמנעים באמצעות סט החוקים שהגדרנו. ניתן למצוא סיכום של כלל תרחישי התקיפה באיור הבא:



[נלקח מהמקור הבא: https://docs.aws.amazon.com/whitepapers/latest/building-a-data-perimeter-on-aws/building-a-data-perimeter-on-aws.html]

איך זה נראה בפרקטיקה

אז ראינו עד כה איך הדברים נראים בתאוריה, אבל בואו נבין איך ניתן לקחת את זה לפרקטיקה (כלומר איך ניתן לממש את סט החוקים שהגדרנו).

על מנת לעבור לשלב הבא נצטרך להכיר עוד כמה קונספטים שקיימים ב-AWS:

1. SCP (Service Control Policy) - סט של חוקים שחלים על כלל הזהויות בארגון שלי ומאפשרים ניהול של כלל החוקים ממקום ריכוזי אחד (חשבון שנקרא Management). באמצעות המנגנון הזה נוכל להחיל את החוקים שרלוונטיים **בפן הזהותי**.

2. VPC Endpoint Policy - VPC Endpoint מאפשר לבצע פעולות מול שירות מסויים בלי הצורך לצאת לאינטרנט. לדוגמא, במידה ואנחנו רוצים לכתוב ל-Bucket נוכל ליצור VPC Endpoint של שירות S3 ובאמצעותו נוכל לגשת ולכתוב ל-Bucket שלנו. שימו לב שבאמצעות הפיצ'ר המשמעותי הזה למעשה נוכל למנוע גישה לאינטרנט ובכך לצמצם באופן משמעותי את ערוצי הדלף. בעת יצירת VPC Endpoint נוכל לשייך לו פוליסה. באמצעות המנגנון הזה נוכל להחיל את החוקים שרלוונטיים **בפן הרשתי**.
3. Resource Control Policy - ניתן לשייך פוליסה עבור משאב מסויים (Resource Based Policy). Resource Control Policy (פיצ'ר שיצא בשנה האחרונה) מאפשר לנו לנהל את ה-RBP של כלל המשאבים בצורה ריכוזית. אפשר לחשוב עליו כמעין SCP של עולם המשאבים (אם ב-SCP אנחנו אוכפים חוקים על כלל הזהויות אז ב-RCP אנחנו אוכפים חוקים על כלל המשאבים).

כדי להחיל את החוקים AWS פרסמה סט של conditions-keys שיעזרו לנו לכתוב את אותם החוקים:

1. ResourceOrgID - באמצעות התנאי הזה נוכל לאכוף את הגישה אך ורק למשאבים של הארגון שלי.
2. VpceOrgID (יצא ממש לאחרונה!) - באמצעות התנאי הזה נוכל לאכוף כי הגישה למשאבים תתאפשר אך ורק מ-VPC Endpoints של הארגון שלי.
3. PrincipalOrgID - באמצעות התנאי הזה נוכל לאכוף כי הגישה למשאבים תתאפשר אך ורק מזהויות של הארגון שלי.

וזהו! יש לנו עכשיו את כל הכלים כדי לממש Data Perimeter.

כך יראה ה-SCP שלנו:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessNotToMineResource",
      "Effect": "Deny",
      "Action": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceOrgID": "MY-ORGANIZATION-ID"
        }
      }
    },
    {
      "Sid": "DenyAccessNotFromMineEndpoint",
      "Effect": "Deny",
      "Action": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:VpceOrgID": "MY-ORGANIZATION-ID"
        }
      }
    }
  ]
}
```

כך יראה ה-VPC Endpoint Policy שלנו:

```
{
    "Statement": [
        {
            "Sid": "AllowAccessOnlyToMyResourceAndFromMyIdentities",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
              "StringEquals": {
                "aws:ResourceOrgID": "MY-ORGANIZATION-ID",
                "aws:PrincipalOrgID": "MY-ORGANIZATION-ID"
              },
            }
        }
    ]
}
```

כך יראה ה-RCP שלנו:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessNotFromMineNetwork",
      "Effect": "Deny",
      "Action": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:VpceOrgID": "MY-ORGANIZATION-ID"
        }
      }
    },
    {
      "Sid": "DenyAccessNotFromMineIdentity",
      "Effect": "Deny",
      "Action": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgID": "MY-ORGANIZATION-ID"
        }
      }
    }
  ]
}
```

שימו לב שלכל אחד מה-condition keys יש וריאנט שמאפשר להחיל אותו על רמה ארגונית אחרית (Account או Organizational Unit). את כלל ה-condition keys תוכלו למצוא פה.

למה לא הכל מושלם

אז על פניו הכל נראה מושלם ופשוט. הצלחנו בצורה מאוד פשוטה להחיל חוקים שיוצרים לנו פרימטר סגור שממנו לא ניתן להדליף מידע החוצה.

אז למה לא הכל מושלם? ראשית, הקונספט הזה עדיין נמצא בראשיתו, ולכן לא כל הפעולות ולא כל המשאבים תומכים בכל היכולות שהוזכרו פה. לדוגמא:

- לא עבור כל השירותים יש תמיכה ב-VPC Endpoint. לדוגמא, השירות Route53 (השירות שאחראי על תעבורת ה-DNS ב-AWS) לא תומך ב-VPC Endpoint. מצד שני, לרוב שירותי ה-Data יש תמיכה ב-VPC Endpoint, ולכן הדבר האפשרי יהיה לפצל את השירותים שאין להם תמיכה ב-VPC Endpoint לאיזור רשתי נפרד.
- לא כל הפעולות נתמכות באמצעות ה-condition keys. ספציפית, יש רשימה שניתן למצוא בתיעוד של AWS אשר מפרטת את הפעולות שלא נתמכות באמצעות ResourceOrgID:

aws:ResourceOrgID

Use this key to compare the identifier of the organization in AWS Organizations to which the requested resource belongs with the identifier specified in the policy.

- **Availability** – This key is included in the request context only if the account that owns the resource is a member of an organization. This global condition key does not support the following actions:
  - AWS Audit Manager
    - `auditmanager:UpdateAssessmentFrameworkShare`
  - Amazon Detective
    - `detective:AcceptInvitation`
  - AWS Directory Service
    - `ds:AcceptSharedDirectory`
  - Amazon Elastic Block Store – All actions
  - Amazon EC2
    - `ec2:AcceptTransitGatewayPeeringAttachment`
    - `ec2:AcceptVpcEndpointConnections`
    - `ec2:AcceptVpcPeeringConnection`
    - `ec2:CopySnapshot`
    - `ec2:CreateTransitGatewayPeeringAttachment`
    - `ec2:CreateVpcEndpoint`
    - `ec2:CreateVpcPeeringConnection`
    - `ec2:DeleteTransitGatewayPeeringAttachment`
    - `ec2:DeleteVpcPeeringConnection`
    - `ec2:RejectTransitGatewayPeeringAttachment`
    - `ec2:RejectVpcEndpointConnections`
    - `ec2:RejectVpcPeeringConnection`

- עוד סיבה מרכזית מדוע לא הכל מושלם היא יישויות שמתנהגות באופן מוזר/חריג. נסתכל לדוגמא על הפעולה create-job תחת S3. פעולה זאת מאפשרת הרצה של פעולת S3 מול כמה אובייקטים שונים. לדוגמא, לקרוא מידע מכמה אובייקטים או למחוק כמות גדולה של אובייקטים.
  ניתן לראות כי אחד מהפרמטרים של הפעולה הוא role-arn:

```
  create-job
--account-id <value>
[--confirmation-required | --no-confirmation-required]
--operation <value>
--report <value>
[--client-request-token <value>]
[--manifest <value>]
[--description <value>]
--priority <value>
--role-arn <value>
[--tags <value>]
[--manifest-generator <value>]
[--cli-input-json | --cli-input-yaml]
[--generate-cli-skeleton <value>]
[--debug]
[--endpoint-url <value>]
[--no-verify-ssl]
[--no-paginate]
[--output <value>]
[--query <value>]
[--profile <value>]
[--region <value>]
[--version <value>]
[--color <value>]
[--no-sign-request]
[--ca-bundle <value>]
[--cli-read-timeout <value>]
[--cli-connect-timeout <value>]
[--cli-binary-format <value>]
[--no-cli-pager]
[--cli-auto-prompt]
[--no-cli-auto-prompt]
```

הפעולה עובדת בתצורה כזאת שכאשר היא מתחילה לרוץ היא מבצעת AssumeRole (פעולה אשר מאפשרת לקבל הרשאות של זהות אחרת) כדי לעבור ל-role שהעברנו לה בפרמטר. מה הבעיה? לאחר שהיא עוברת ל-role שלנו היא מבצעת את הפעולות מתוך רשת חיצונית ולא מתוך הרשת של הארגון שלנו, ולכן במידה ונחיל על הזהות הזאת את החוק כי ניתן לגשת אך ורק מהרשת הארגונית שלנו הפעולה תיכשל. לכן, נצטרך לשים החרגה ספציפית עבור זהויות מהסוג הזה.

- עוד סיבה היא שישנן פעולות שחורגות מההתנהגות הצפויה ועלולות לשבור את הפרימטר. לדוגמא, ישנה הפעולה put-bucket-notification-configuration. פעולה זו מאפשרת לנו להגדיר לאיפה נרצה לשלוח נוטיפיקציות על שינויים של אובייקטים שמתרחשים ב-Bucket. לדוגמא, נוכל לקבל התראה בכל פעם שנכתב אובייקט ל-Bucket. מה הבעיה? ניתן לבחור לשלוח את ההתראות הללו למשאב חיצוני (לדוגמא, ל-SNS Topic חיצוני), וכך למעשה כלל ההתראות ישלחו אל מחוץ לארגון (או חלילה לסביבה של התוקף). בהתראות הללו נוכל למצוא גם את שם ה-bucket או שם האובייקט, מידע אשר עלול להיות רגיש עבור גופים מסויימים. AWS עשו מאמצים לתעד את ההתנהגויות הלא צפויות הללו, וניתן למצוא את המידע הזה (ועוד הרבה פוליסות אחרות) ברפו הזה.

AWS עדיין משפרת דברים

אז למרות שהקונספט נראה שלם ובשל למעשה הוא עדיין נמצא בראשיתו והספקית נמצאת בעבודה מתמדת על מנת לשפרו ולאפשר בקרה חזקה והדוקה יותר על הפרימטר. כמה פיצ'רים מגניבים שיצאו ממש לאחרונה:

- RCP - המנגנון אשר מאפשר להחיל חוקים על כלל המשאבים בארגון זהו פיצ'ר שיצא בשנה האחרונה. למעשה עד אז היינו צריכים להחיל את החוקים שלנו על כל אחד מהמשאבים בנפרד בלי שתהיה לנו היכולת לנהל את כלל החוקים הללו בצורה ריכוזית (נשמע הזוי נכון?).
- Aws:VpceOrgID - מדובר על condition-key שהגיע אלינו באוגוסט השנה. אם עד היום היינו צריכים להשתמש ב-condition-key בשם aws:SourceVpce שבו יכולנו לציין את ה-VPC Endpoints הספציפיים שאנחנו רוצים לאפשר גישה מהם, כעת עם ה-condition-key החדש אנחנו יכולים לאכוף גישה מכל endpoint שנוצר בתוך הארגון. דרך נוספת לעשות משהו באופן ריכוזי.

AWS-ל יש תיעוד ממש טוב

אז AWS עושה מאמצים כבירים על מנת להנגיש את הקונספט הזה בצורה הנוחה ביותר. כחלק מהמאמצים הללו היא מתחזקת רפו בגיטהאב שבו היא מספקת דוגמאות לפוליסות שיכולות להוות את הבסיס עבור תפיסת ה-Data Perimeter. תוכלו למצוא שם החרגות כלליות שיש צורך לעשות, פירוט על התנהגויות חריגות של פעולות/שירותים מסויימים ועוד המון מידע מועיל למי שמעוניין להקים Data Perimeter בענן.

סיכום

אז אם עד לאחרונה לבנות פרימטר בענן הייתה משימה בלתי-אפשרית, נראה שאנחנו יותר ויותר מתקרבים לעולם שבו נוכל לנהל סיכונים גם על מידע רגיש יותר שיעלה לענן. היום, רוב מערכות הארגונים מתבססות על כלי ניטור שלאו דווקא ימנעו את דלף המידע אלא יאפשרו לנו לגלות זאת לאחר הדליפה. עם הקונספט שראינו פה הדברים יכולים להיראות אחרות - על ידי הגדרה מקדימה של סט חוקים נוכל למנוע מראש תרחישי תקיפה רבים ובכך למנוע מתוקף להוציא את המידע שלנו מחוץ לענן.

עם זאת צריך לומר שתחזוק סט החוקים הזה כיום היא משימה לא פשוטה. הדבר הרבה פעמים דורש להכיר את מנגנוני ההגנה של הספקית לעומק, התנהגויות שונות ומשונות של שירותים שונים והכרה רחבה של תרחישי תקיפה בענן. נראה שהספקיות הולכות לכיוון שבו הן ינגישו לנו יותר ויותר בקלות את הדברים שפעם היה כמעט בלתי-אפשרי לממש, ולכן גם אם כיום נראה לכם שיהיה זה קשה לתחזק את הרכיבים

השונים בפתרון לאורך זמן, אין זה אומר שהדבר לא יהיה קל יותר בעתיד, ולכן שווה לחכות ולראות איזה יכולות נוספות יצאו בהמשך.

במאמר דיברתי ספציפית על ספקית הענן AWS, אבל יכולות דומות קיימות גם בספקיות השונות. לדוגמא, ב-GCP קיים מנגנון בשם VPC-SC שמיועד להתמודד עם חלק מתרחישי התקיפה שהוזכרו כאן. אני מעודד אתכם ללמוד ולהכיר את המנגנון המתאים בהתאם לספקית בה אתם משתמשים ביומיום.

על המחבר

קוראים לי יהב פסטינגר, וכיום אני מתעסק במחקר למטרות הגנה בענן.

עבור יצירת קשר תוכלו למצוא אותי ב-Linkdein.

מקורות מידע

- https://docs.aws.amazon.com/whitepapers/latest/building-a-data-perimeter-on-aws/building-a-data-perimeter-on-aws.html
- https://dl.acm.org/doi/10.1145/3546068
- https://www.microsoft.com/en-us/security/blog/2022/06/02/exposing-polonium-activity-and-infrastructure-targeting-israeli-organizations/
- https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_scps.html
- https://docs.aws.amazon.com/vpc/latest/privatelink/what-is-privatelink.html
- https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_condition-keys.html
- https://github.com/aws-samples/data-perimeter-policy-examples
- https://cloud.google.com/vpc-service-controls/docs/overview