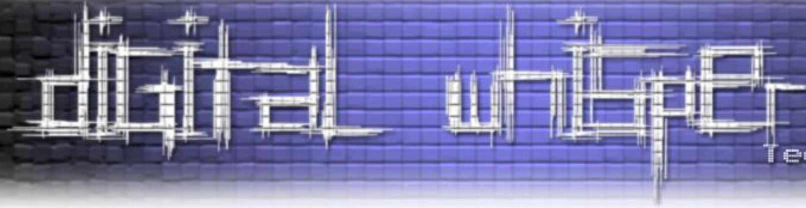


איך בונים Vulnerability Scanner

מאת עילי טרטמן

הקדמה

אתחיל בשאלה, איזה סוגים של Vulnerability scanners קיימים? אם הייתי שואל את השאלה הזו לפני 10 שנים, רוב האנשים היו עונים שיש רק סוג אחד, ומפנים אותי לקנות את הגרסה האחרונה של Nessus.

אפשר לטעון שכלים כמו SAST, DAST ו-SCA נוצרו כבר בסוף שנות ה-2000, אך הם עדיין לא היו נפוצים בקהילה. היום, המצב שונה לגמרי. אם תחפשו בגוגל "Vulnerability Scanner", תוצפו בעשרות פרסומות שמציעות לך את "הגרסה האחרונה והמשופרת" של הסורק שלהם — אבל איזה מהם באמת מתאים לארגון שלך? במאמר הזה ננסה להבין מהם הסוגים השונים שקיימים, איך השחקניות הגדולות מפתחות את הכלים הללו ולבסוף נראה איך לכתוב אחד בעצמינו.

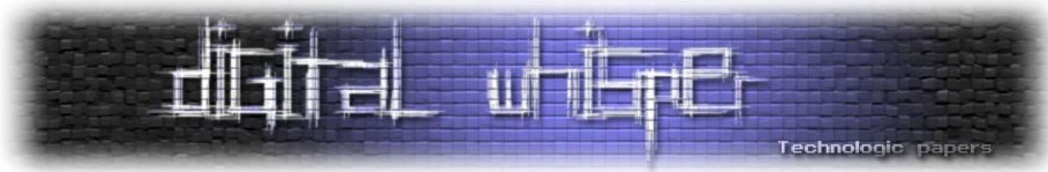
מושגים בסיסים

לפני שנתחיל קצת מושגים כדי שנדבר את אותה השפה.

CVE

CVE הוא קיצור של Common Vulnerabilities and Exposures — ההגדרה הרשמית היא תוכנית לשיתוף מידע על פגיעויות ידועות בציבור בתחום הסייבר. במילים פשוטות CVE == חולשה ברכיב מסויים (תוכנה, מערכת הפעלה, חבילת קוד...) שדווחה ומוכרת בציבור.

התוכנית מנוהלת על-ידי MITRE Corporation והוקמה בשנת 1999. כיום CVE נחשב לסטנדרט התעשייתי לזיהוי וסיווג פגיעויות. עד היום דווחו מעל ל 200 אלף CVEs ובכל יום בממוצע מדווחים מעל 130 חדשים.



CPE

CPE הוא סכמה למתן שמות לתוכנות, חומרות ומערכות הפעלה. הוא מספק מבנה אחיד שמאפשר לכלים ומאגרי מידע להתאים פגיעויות לפלטפורמות ספציפיות. כמו שבטח ניחשתם גם הוא מתחוזק ע"י MITRE ונחשב לסטנדרנט בתעשייה.

דוגמה לאיך CPE נראה:

```
cpe:2.3:a:apache:log4j:2.14.1:*:*:*:*:*:*:*
```

- cpe:2.3 - הגרסה של הסכמה.
- a — מציין Application (יכול להיות גם h עבור Hardware או o עבור Operating System).
- apache - ה-Vendor.
- log4j — המוצר.
- 2.14.1 — הגרסה של המוצר.

השדות הנותרים (מה שמסומן בכוכבית) מייצגים עדכון, מהדורה, שפה ועוד.

למה זה מעניין? כל CVE מועשר בקישורים ל-CPE-ים תואמים ע"י NIST ואיך זה בדיוק עוזר לנו? נגלה בהמשך.

Host Based Vulnerability Scanners

נתחיל בקל, זהו הסוג הראשון והוותיק ביותר, שנוצר בסוף שנות ה-90, וסביר להניח שזה מה שעולה בראש כשחושבים על סורק פגיעויות. כלים אלה מבוססים על agents שיושבים על ה-Workloads שאותם נרצה לסרוק, ושולחים נתונים לשרת עיבוד מרכזי — שבדרך כלל נמצא בתוך רשת היעד ברשתות On-Prem ובחלק מהמקרים גם בענן. השרת מעבד את הנתונים מכל Endpoint ובדרך כלל מפיק סוג של דוח או Dashboard.

דוגמאות בולטות:

- Tenable Nessus / Tenable SC
- Qualys Agent
- Rapid7 InsightVM (Nexpose)

Network Vulnerability Scanners

אלו הם ה"קרובים" של ה-Host based. רוב הכלים המסחריים מסוג Host based מציעים גם גרסת Network.

כלים אלה פועלים ע"י פריסת שרת ברשת היעד או שרת בענן של החברה שמתחבר מרחוק אל ה-workloads (בדרך כלל דרך SSH או WinRM/WMI). השרת מעבד את הנתונים שמתקבלים מה-workloads ומפיק גם פה בדרך כלל סוג של דוח או Dashboard.

בנוסף, רוב הכלים האלו מספקים גם מידע על פורטים פתוחים ומיפוי רשת — נתונים שאינם זמינים בכלים מבוססי-agent. אם אתה CISO של חברה שרוצה לסרוק סביבת on-prem קלאסית, כנראה שתרצה להשתמש בשילוב של שתי הגישות — agent + network scan.

דוגמאות נפוצות:

- Tenable Nessus
- OpenVAS (by Greenbone)
- Qualys

Disclaimer

לפני שנצלול לכל אחד מן הכלים במאמר ולאיך הם עובדים אציין כי המידע שנכתב במאמר מבוסס על הערכה אישית ולא על פי מידע רשמי שנמסר מן החברות המוזכרות במאמר, לפני השימוש ולניתוח רשמי של איך כל כלי עובד יש להיוועץ עם מומחה.

אז איך הם עובדים ?

נבחן את אחד הכלים המובילים בתעשייה - **Tenable Nessus** (או בשמו החדש Tenable SC).

הכלי משתמש בשפת סקריפטים בשם NASL (Nessus Attack Scripting Language) כדי לזהות פגיעויות. מכיוון ש-Nessus היה בעבר קוד פתוח, ניתן עדיין למצוא כמות גדולה של קבצי NASL ב-GitHub.

כל קובץ NASL מכסה לרוב כמה CVEs. ניתן לראות בדוגמה שאין חיבור ישיר ל-CPE (למרות שמדובר ב-plugin שנוצר ע"י הקהילה) בקבצי ה-NASL לרוב מתבצעת בדיקה ישירה של הפגיעות ע"י קוד שכתב חוקר. דוגמה לקובץ NASL:

CVE-2024-30078- / cve_2024_30078_check.nasl

Code Blame 93 lines (81 loc) · 2.91 KB

```
39    # Define the endpoint and command to be executed
40    endpoint = "/check";  # Replace with the actual endpoint
41    command = "your_command_here";  # Replace with the actual command to be executed
42
43    # Function to check vulnerability and execute command
44    function check_vulnerability_and_execute(ip, port, endpoint, command)
45    {
46      # Construct the URL and payload for the vulnerability check
47      url = string("http://", ip, ":", port, endpoint);
48      payload = string(
49        'POST ', endpoint, ' HTTP/1.1\r\n',
50        'Host: ', ip, '\r\n',
51        'Content-Type: application/json\r\n',
52        'Content-Length: 42\r\n',
53        '\r\n',
54        '{"command":"check_vulnerability","cve":"CVE-2024-30078"}'
55      );
56
57      # Send the request and receive the response
58      response = http_send_recv(data:payload, port:port);
59
60      # Check if the response indicates vulnerability
61      if ("\"vulnerable\": true" >< response[2])
62      {
63        security_hole(port);  # Report the vulnerability
64
65        # Construct the payload for command execution
66        payload_command = string(
67          'POST ', endpoint, ' HTTP/1.1\r\n',
68          'Host: ', ip, '\r\n',
69          'Content-Type: application/json\r\n',
70          'Content-Length: ', strlen(command) + 23, '\r\n',
71          '\r\n',
72          '{"command":"', command, '"}'
73        );
74
75        # Send the request to execute the command
76        http_send_recv(data:payload_command, port:port);
77      }
78      else
79      {
80        security_note(port);  # Report that the target is not vulnerable
81      }
82    }
```

[Image from by lvyitian/CVE-2024-30078-, licensed under GNU GPL v3]

נקפוץ לדוגמה נוספת והפעם בקוד פתוח.

OpenSCAP

OpenSCAP הוא פרויקט קוד פתוח המשמש עבור vulnerability scanning, compliance checking & security automation הוא מבוסס על תקן SCAP (Security Content Automation Protocol) שפותח ע"י NIST.

איך OpenSCAP מוצא פגיעויות? הוא מסתמך על הקשר בין CVE ל-CPE שמנוהל ע"י NIST NVD. לדוגמה, אם נרצה לסרוק פגיעויות במערכות מבוססות RHEL, ניגש לכתובת:

https://www.redhat.com/security/data/oval/v2/RHEL9/

ונוריד את הקובץ: **rhel-9.oval.xml.bz2**

על מנת להוציא דו"ח של פגיעויות על המטרה נריץ את הפקודה:

```
oscap oval eval --report vulnerability.html rhel-9.oval.xml#
```

הקובץ מכיל הגדרות בשפת OVAL למציאת CVEs במערכת ההפעלה והחבילות המותקנות אך איך הוא יודע לעשות זה?

נכנס לקישור הבא:

https://security.access.redhat.com/data/metrics/

security.access.redhat.com/data/metrics/

Index of /metrics/

Search

Name	Last Modified
container-name-repos-map.json	Thu, 09 Oct 2025 09:45:08 +0000
cvemap.xml	Tue, 14 Oct 2025 12:27:55 +0000
cvemap.xml.bz2	Tue, 14 Oct 2025 12:28:43 +0000
ds/	-
repository-to-cpe.json	Tue, 14 Oct 2025 00:48:36 +0000
rhsa.rss	Tue, 14 Oct 2025 12:00:45 +0000

ונוריד את הקובץ **repository-to-cpe.json**

קובץ זה מכיל את המיפוי בין חבילות RHEL לבין הפורמט של CPE ומתוחזק ע"י RedHat.

```
{
  "data": {
    "3scale-amp-2-for-rhel-8-ppc64le-debug-rpms": {
      "cpes": [
        "cpe:/a:redhat:3scale:2.13::el8",
        "cpe:/a:redhat:3scale:2.14::el8",
        "cpe:/a:redhat:3scale:2.15::el8",
        "cpe:/a:redhat:3scale:2.16::el8",
        "cpe:/a:redhat:3scale_amp:2.11::el8",
        "cpe:/a:redhat:3scale_amp:2.12::el8",
        "cpe:/a:redhat:3scale_amp:2.8::el8"
      ],
      "repo_relative_urls": [
        "content/dist/layered/rhel8/ppc64le/3scale-amp/2/debug"
      ]
    },
```

כפי שניתן לראות, כל חבילה מתאימה לאחד או יותר CPE-ים. הכלי בודק את החבילות המותקנות על מערכת ההפעלה ומבצע את ההתאמה לרשימה. במידה ויש התאמה היעד פגיע.

RedHat בעצמם מפנים לכלי בדוקומנצטציה הרשמית שלהם:

6.1. Configuration compliance tools in RHEL

You can perform a fully automated compliance audit in Red Hat Enterprise Linux by using the following configuration compliance tools. These tools are based on the Security Content Automation Protocol (SCAP) standard and are designed for automated tailoring of compliance policies.

SCAP Workbench

The `scap-workbench` graphical utility is designed to perform configuration and vulnerability scans on a single local or remote system. You can also use it to generate security reports based on these scans and evaluations.

OpenSCAP

The `OpenSCAP` library, with the accompanying `oscap` command-line utility, is designed to perform configuration and vulnerability scans on a local system, to validate configuration compliance content, and to generate reports and guides based on these scans and evaluations.

> Important
> You can experience memory-consumption problems while using **OpenSCAP**, which can cause stopping the program prematurely and prevent generating any result files. See the OpenSCAP memory-consumption problems Knowledgebase article for details.

DAST

Dynamic Application Security Testing (DAST) הוא סוג של בדיקת "קופסה שחורה" (Black Box), שבוחן אפליקציות רצות — לרוב אפליקציות WEB — מבחוץ. הוא פועל כמו תוקף אמיתי שמנסה לזהות נקודות תורפה באפליקציה.

DAST מסוגל לגלות פגיעויות כמו:

- SQL injection
- Cross-site scripting (XSS)
- Security misconfigurations
- Authentication and session issues

בניגוד לכלים הקודמים שסיקרנו מרבית הפלט שלו לא יהיה בפורמט של CVEs אלא יותר בסגנון הבא:

Top Vulnerabilities

⚠ Critical	SQL Injection	8
◇ High	Cross-site scripting	3
◇ High	Vulnerable package dependencies (high)	1
⌞ Medium	Directory traversal	11
Medium	Vulnerable package dependencies medium	8

דוגמאות לכלים:

- OWASP ZAP
- Burp Suite (Pro)
- Acunetix

אז איך DAST עובד ?

1. Scanning (סריקה) - הכלי סורק את האפליקציה ומזהה HTML attributes, URLs, APIs כדי לאתר נקודות כניסה אפשריות.
2. Attack Simulation (הדמיית תקיפה) - הכלי מחקה התקפות אמיתיות ע"י שליחת בקשות מעובדות לאפליקציה, למשל ניסיון לנצל חולשת XSS.
3. Vulnerability Detection (זיהוי פגיעויות) - לאחר ההדמיה, הכלי מנתח את תגובות האפליקציה כדי לזהות פגיעויות ולדרג את חומרתן.
4. Reporting (דיווח) - מופק דוח עם כל הפגיעויות שנמצאו באפליקציה.

כשמדובר בביצועים ודיוק, קיימים פערים גדולים בין כלים קוד פתוח לכלים מסחריים. המסחריים לרוב מציעים דיוק גבוה יותר, ניתוח מתקדם ודוחות מעמיקים, בעוד הפתוחים מתמקדים בגמישות והתאמה אישית ומציעים כלים הרבה יותר חלשים ופחות "מוכנים מראש".

SAST

Static Application Security Testing (SAST) הוא סוג של בדיקת "קופסה לבנה" (white box), שבוחן את קוד המקור, bytecode או binary code של אפליקציה, מבלי להריץ אותה בפועל. המטרה - לזהות חולשות אבטחה בקוד עצמו.

חלק מסוגי הפגיעויות ש-SAST מזהה:

- SQL Injection
- Cross-Site Scripting (XSS)
- CSRF
- Buffer Overflows
- סיסמאות/מפתחות בקוד (Hardcoded Credentials)
- APIs לא מאובטחים

כלים נפוצים

- SonarQube
- Checkmarx
- Fortify Static Code Analyzer
- Veracode
- CodeQL (by GitHub)

איך SAST עובד ?

נבחן לדוגמה את הכלי Semgrep , שהוא פתרון Open-Source פופולרי (עם גרסה מסחרית זמינה).

```
Findings:

  app.js
     javascript.express.security.audit.express-check-csurf-middleware-usage.express-check-csurf-
     middleware-usage
        A CSRF middleware was not detected in your express application. Ensure you are either using
        one  such as `csurf` or `csrf` (see rule references) and/or you are properly doing CSRF
        validation in your routes with a token or cookies.
        Details: https://sg.run/BxzR

         10┆ var app = express();
```

Semgrep מורכב ממספר רכיבים:

- מנוע עיבוד מרכזי
- שרת שפה (Language Server)
- מסד נתונים המכיל חוקים (Rules Database)

כל פגיעות שהכלי מוצא נובעת מחוק מוגדר מראש (Rule) , שנראה כך:

```
rules:
- id: info-leak-on-non-formated-string
  message: >-
    Use %s, %d, %c... to format your variables, otherwise this could leak information.
  metadata:
    cwe:
    - 'CWE-532: Insertion of Sensitive Information into Log File'
    references:
    - http://nebelwelt.net/files/13PPREW.pdf
    category: security
    technology:
    - c
    confidence: LOW
    owasp:
    - A09:2021 - Security Logging and Monitoring Failures
    subcategory:
    - audit
    likelihood: LOW
    impact: MEDIUM
  languages: [c]
  severity: WARNING
  pattern: printf(argv[$NUM]);
```

החלק המעניין הוא **ה-pattern** שבתחתית:

```
pattern: printf(argv[$NUM]);
```

ה-pattern מעובד ע"י המנוע של הכלי בשילוב ה-Source Code של המפתח על מנת לזהות את הפגיעות באפליקציה.

שיטה זו, של "סריקת תבניות דמויות Regex", נמצאת בשימוש רחב מאוד בקרב כלי SAST. עם זאת, בשנים האחרונות (Large Language Models) **LLMs** מתחילים להיכנס לתמונה ולהציע רמות דיוק והבנה גבוהות בהרבה. גם בגרסה המסחרית של SemGrep כבר משולבות יכולות AI לזיהוי איכותי יותר.

SCA

כלי SCA (Software Composition Analysis) סורקים את ה-Code Base של אפליקציה במטרה לנתח רכיבי קוד פתוח וספריות צד שלישי על מנת לחשוף:

- פגיעויות ידועות (Known Vulnerabilities)
- בעיות רישוי (License Compliance)
- ספריות ישנות או לא מתוחזקות

כלים פופולריים:

- Snyk
- Dependabot (GitHub)
- OWASP Dependency-Check
- Black Duck
- WhiteSource (כיום Mend)

איך SCA עובד?

רוב הכלים מהסוג הזה יוצרים תחילה SBOM (Software Bill of Materials) - רשימה מפורטת של כל הרכיבים, הספריות וה-dependencies שמרכיבים את האפליקציה. לאחר יצירת ה-SBOM, המנוע של הכלי משווה אותו אל מאגר פגיעויות (בדרך כלל NVD או GHSA) ומתריע על CVEs תואמים שנמצאו.

דוגמה להרצת SCA (snyk):

```
Testing /Work/snyk/snyk...

Organization:      team
Package manager:   npm
Target file:       package-lock.json
Project name:      snyk
Open source:       no
Project path:      /Work/snyk/snyk
Local Snyk policy: found
Licenses:          enabled

✔ Tested 455 dependencies for known issues, no vulnerable paths found.

Tip: Detected multiple supported manifests (27), use --all-projects to scan all of them at once.

Next steps:
- Run `snyk monitor` to be notified about new related vulnerabilities.
- Run `snyk test` as part of your CI/test.
```

Cloud Vulnerability Scanners

כלים אלו מיועדים לסרוק משאבים בענן — הן שירותי IaaS (כמו מכונות וירטואליות ו-storage buckets), והן שירותי PaaS ו-SaaS בסביבות ענן שונות. הם עושים זאת ע"י שימוש ב- **cloud APIs** ובמנגנונים נוספים כדי להעריך את רמת הפגיעויות בסביבה.

דוגמאות לכלים פופולריים

- Amazon Inspector
- Wiz
- Orca Security
- Qualys Cloud Platform

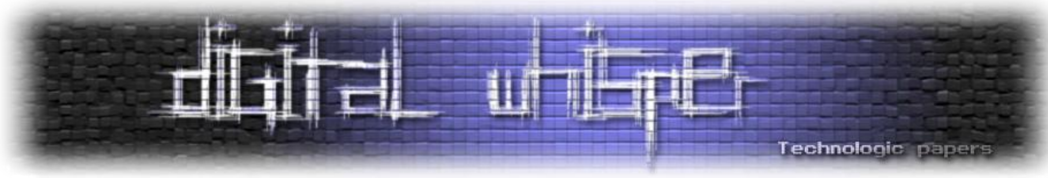
בלבול נפוץ

רבים חושבים שסורקי ענן מבוססים על טכנולוגיה חדשה לגמרי, אך למעשה ברוב המקרים הם משתמשים באותם עקרונות של סריקה כמו הכלים המסורתיים. ההבדל הוא רק במיקום ובסוג הסביבה. בסופו של דבר, גם בענן אנו סורקים את אותם רכיבים: מערכות הפעלה, תוכנות, ספריות וחומרות — רק הפעם הן רצות על תשתית וירטואלית.

אז מה שונה באמת?

במקום להתקין agent על כל workload או להתחבר אליו מרחוק, הכלים החדשים משתמשים ביכולת של הענן לבצע snapshot של ה-workload בזמן ריצה.

לאחר מכן, הסריקה מתבצעת על עותק ה-snapshot מבלי לפגוע בפעילות השוטפת של יעד הסריקה. התהליך של התאמת הפגיעויות נעשה בדיוק כמו בסורקים מבוססי-agent או סורקים מבוססי-רשת — לרוב לפי CPE ↔ CVE.



Container Vulnerability Scanners

כלים אלה מיועדים לזהות פגיעויות בתמונות קונטיינרים (container images) ובסביבות קונטיינרים כמו Docker ו-Kubernetes.

דוגמאות לכלים

- Trivy
- Clair
- Snyk Container

איך הם עובדים?

גם כאן הכל מתחיל ב-SBOM.

מרבית הכלים יוצרים SBOM של הקונטיינר, שמפרט את כל החבילות המותקנות והגרסאות שלהן. לאחר מכן הכלים מתחלקים לשני סוגים:

1. כלים שמבצעים התאמה בין החבילות ל-CPEs הרלוונטיים ולבסוף מאתר את ה-CVEs התואמים על פי הקורלציה שמנוהלת ע"י NIST.
2. כלים שמשתמשים ב-GHSA מאגר פגיעויות שמתוזק ע"י GitHub.

הסוג השני נפוץ יותר בהרבה.

נדגים בעזרת Syft - ו-Grype

נבחן שני כלים פופולריים בקוד פתוח:

1. Syft - יוצר SBOM מ-image של קונטיינר.
2. Grype - מקבל את ה-SBOM שנוצר ע"י Syft, ומשווה אותו למאגרי CVE כדי לזהות פגיעויות תואמות.

נריץ את syft על היעד:

Syft

A CLI tool and Go library for generating a Software Bill of Materials (SBOM) from container images and filesystems. Exceptional for vulnerability detection when used with a scanner like Grype.

Validations passing | go report A+ | release v1.23.1 | Go v1.24.1 | License Apache 2.0 | Discourse Join | Mastodon Follow

```
> syft clashapp/qa-page -o json | jq '.artifacts[] | select(.name == "nginx")'
 ✔ Loaded image
 ✔ Parsed image
 ✔ Cataloged image          [113 packages]
{
  "name": "nginx",
  "version": "1.10.3-1+deb9u3",
  "type": "deb",
  "found-by": [
    "dpkg-cataloger"
  ],
  "locations": [
    {
      "path": "/var/lib/dpkg/status",
      "layer-index": 1
    }
  ],
  "metadata": {
    "package": "nginx",
    "source": "",
    "version": "1.10.3-1+deb9u3"
  }
}
~                                                              10:11:44 AM
py382 >
```

ניתן לראות שהוא חילץ את החבילות המותקנות על היעד ל-SBOM בפורמט JSON.

כעת נטען אותם ל-Grype:

```
{
  "vulnerability": {
    "id": "CVE-2020-11724",
    "severity": "Medium",
    "links": [
      "https://security-tracker.debian.org/tracker/CVE-2020-11724"
    ],
    "cvss-v2": {
      "base-score": 5,
      "vector": "AV:N/AC:L/Au:N/C:N/I:P/A:N"
    }
  },
  "matched-by": {
    "matcher": "dpkg-matcher",
    "search-key": "distro[debian 9] constraint[< 1.10.3-1+deb9u5 (deb)]"
  },
  "artifact": {
    "name": "libnginx-mod-http-xslt-filter",
    "version": "1.10.3-1+deb9u3",
    "type": "deb",
    "found-by": [
      "dpkg-cataloger"
    ],
    "locations": [
      {
        "path": "/var/lib/dpkg/status",
        "layer-index": 1
      }
    ],
    "metadata": {
      "package": "libnginx-mod-http-xslt-filter",
      "source": "nginx",
      "version": "1.10.3-1+deb9u3"
    }
  }
}
```

ויש לנו את זה. הכלי מציע פגיעויות ביעד.

איך לבנות אחד בעצמך

הרגע לו חיכינו הגיע. למדנו על רוב הסוגים הקיימים היום בשוק ועכשיו האמיצים מבינינו ילמדו כיצד לבנות כלי כזה בעצמינו.

אתחיל ואגיד שזו לא משימה קלה. יש סיבה שהחברות שמוכרות את הכלים האלו שוות מאות מילונים וחלקן אפילו מילארדים, אך זה לגמרי אפשרי.

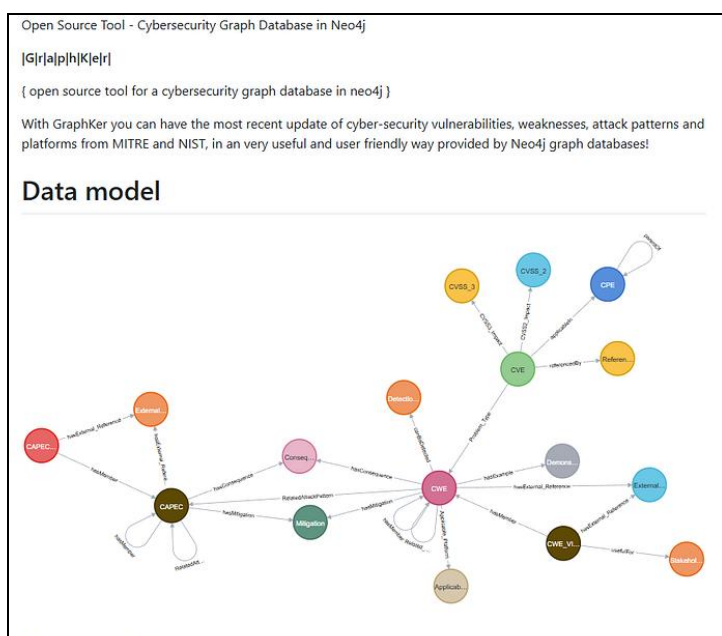
אז נניח שאני הבעלים של סטארטאפ בתחום הסייבר ואני רוצה להוסיף למוצר שלי יכולת של סריקת פגיעויות ב-Windows. איך אעשה את זה? (דרך אגב אחת המשימות הכי מורכבות בתחום תיכף תבינו למה)

שלב 1

ראשית אצטרך Database שיכיל את כלל המידע מה-NVD (CWE,CPE,CVE..), יחד עם הקשרים ביניהם.

במקום להקים את זה מאפס, אפשר להשתמש בפרויקט קוד פתוח מעולה בשם:

https://github.com/amberzovitis/GraphKer



[Image from by https://github.com/amberzovitis/GraphKer]

הכלי בגדול עושה את כל העבודה בשבילנו ומכניס את כל המידע מ-NVD לתוך DB של NEO4J (מסד נתונים גרפי) ומייצר את הקשרים שנצטרך בשביל הסורק שלנו.

שלב 2

שנית אני אכתוב Script קצר שמתחבר מרחוק ומחלץ את כל התוכנות המותקנות על יעד:

```
param(
    [string]$ComputerName,
    [pscredential]$Credential = (Get-Credential)
)

Invoke-Command -ComputerName $ComputerName -Credential $Credential -ScriptBlock {
    $paths = @(
        'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall',
        'HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall'
    )

    foreach ($p in $paths) {
        if (Test-Path $p) {
            Get-ChildItem $p | ForEach-Object {
                $x = Get-ItemProperty $_.PSPath
                if ($x.DisplayName) {
                    $x | Format-List *
                    Write-Host "--------------------------------------"
                }
            }
        }
    }
}
```

אם נסתכל על כל השדות שניתן לחלץ נשים לב שב-Windows שאין תאימות לתקן של NVD (CPE) ללא שימוש ב-Microsoft Defender Vulnerability Managment בניגוד לVendors אחרים (כמו RedHat שראינו לפני כן שמפרסמים את המידע באינטרנט).

כך יראה output של ה-Script:

```
AuthorizedCDFPrefix :
Comments            :
Contact             :
DisplayVersion      :
HelpLink            :
HelpTelephone       :
InstallDate         :
InstallLocation     :
InstallSource       :
ModifyPath          :
Publisher           : Node.js Foundation
Readme              :
Size                :
EstimatedSize       :
UninstallString     :
URLInfoAbout        :
URLUpdateInfo       :
VersionMajor        :
VersionMinor        :
WindowsInstaller    :
Version             :
Language            :
DisplayName         :
PSPath              :
PSParentPath        :
PSChildName         :
PSProvider          :
```

כעת אחרי שחילצנו את את כל התוכנות על היעד נכניס אותם ל-Database:

```
1 MERGE (c:Computer {name: $computerName})
2 MERGE (s:Software {
3     AuthorizedCDFPrefix: $AuthorizedCDFPrefix,
4     Comments: $Comments,
5     Contact: $Contact,
6     DisplayVersion: $DisplayVersion,
7     HelpLink: $HelpLink,
8     HelpTelephone: $HelpTelephone,
9     InstallDate: $InstallDate,
10    InstallLocation: $InstallLocation,
11    InstallSource: $InstallSource,
12    ModifyPath: $ModifyPath,
13    Publisher: $Publisher,
14    Readme: $Readme,
15    Size: $Size,
16    EstimatedSize: $EstimatedSize,
17    UninstallString: $UninstallString,
18    URLInfoAbout: $URLInfoAbout,
19    URLUpdateInfo: $URLUpdateInfo,
20    VersionMajor: $VersionMajor,
21    VersionMinor: $VersionMinor,
22    WindowsInstaller: $WindowsInstaller,
23    Version: $Version,
24    Language: $Language,
25    DisplayName: $DisplayName,
26    PSPath: $PSPath,
27    PSParentPath: $PSParentPath,
28    PSChildName: $PSChildName,
29    PSProvider: $PSProvider
30 })
31 MERGE (c)-[:HAS_SOFTWARE]->(s);
32
```

שלב 3

ועכשיו לחלק הבעייתי, כמו שציינתי קודם Microsoft לא משחררת CPEs ללא שימוש ב- Microsoft Defender Vulnerability Management ואני לא יכול להניח שלכל לקוח שלי יהיה רישוי (במידה ויש לו הכלי במילא יודע לחלץ CVES). אז מה עלי לעשות ? יש כמה גישות להתאים בין ה-outputs של תוכנה ל-CPE הרלוונטי שלו:

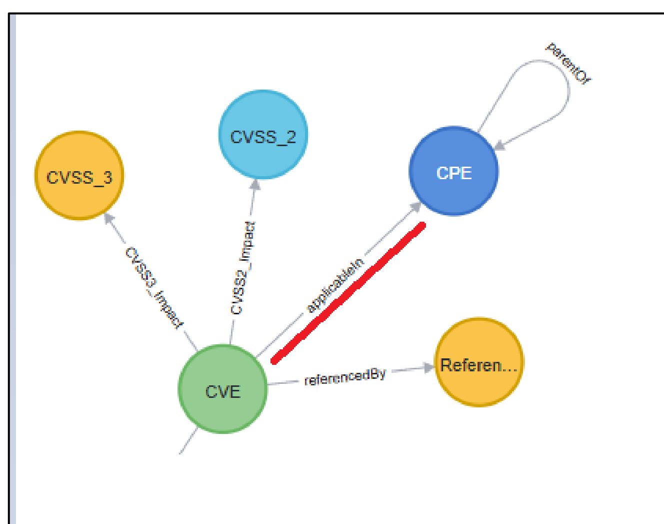
1. להשתמש ב-LLM או vectorized database (כמו Weaviate או Pinecone).
2. להשתמש באלגוריתם דמיון טקסטואלי (כגון Levenshtein distance).
3. לבנות מנוע התאמה מותאם אישית לפי תיעוד ה-CPE של NVD או על סמך ה-Advisories שמשחררת Microsoft.

כן זה לא פשוט, אני מזכיר שוב החברות שעושות את זה שוות מאות מיליוני דולרים. מניסיון אישי ב-Windows קשה להגיע לרמת סמך גבוהה ללא בדיקות שנכתבות ע"י חוקר, לעומת זאת ב-Linux ניתן לעשות שילוב של הטכניקות המוצגות פה בשילוב כלי open-source ולהגיע לרמת סמך גבוהה אפילו ברמה מסחרית.

שלב 4

כעת אחרי שכל התוכנת מהיעד שלנו נמצאת ב-DB ומקושרות ל-CPE הרלוונטי כל שנותר הוא למצוא את ה-CVEs שמקושרים.

למזלינו GRAPHKER כבר טיפל בזה:



[Image from by https://github.com/amberzovitis/GraphKer]

לרוע מזלינו גם פה התהליך לא כלכך פשוט. בתמונה למעלה ניתן לראות קשר ישר בין CVE ל-CPE אך יש חריגות.

אם נקפוץ לדוגמה של CVE-2021-45046 נוכל לראות מקרים כאלו:

Configuration 8 (hide)

cpe:2.3:o:siemens:6bk1602-0aa12-0tp0_firmware:*:*:*:*:*:*:*:* Show Matching CPE(s)	Up to (excluding) 2.7.0
Running on/with	
cpe:2.3:h:siemens:6bk1602-0aa12-0tp0:-:*:*:*:*:*:*:* Show Matching CPE(s)	

משמע ה-CVE רלוונטי אך ורק אם תוכנה מסויימת רצה על מערכת הפעלה ספציפית ויש דוגמאות נוספות ומורכבות אף מזו.

לכן עלינו לכתוב מנוע היודע להתייחס לקשרים המורכבים ולהתריע רק שיש התאמה מלאה.

וזהו סיימנו ! אם הגעתם עד פה ברכות אתם יודעים לבנות Vulnerability scanner ממש כמו הגדולים.

סיכום

לסיכום למדנו על סוגי ה-Vulnerability scanners הקיימים בשוק, איך כל אחד עובד ואיך אפשר לכתוב אחד בעצמינו.

שוק ה-Vulnerability scanners הוא שוק ענק, מהוויתקים והרווחים בתעשיית הסייבר. כל כמה שנים מתווסף שחקן חדש ונפתח נתח שוק שניתן להכנס אליו.

מה הטרנד עכשיו אתם שואלים?

להתריע רק על מה שבאמת מנוצל. הכלים האלו מייצרים עשרות אם לא מאות התרעות על חולשות ופגיעיות באפליקציות של לקוחות.

סטארטאפים כמו Upwind ו-Oligo לדוגמה יודעים להתריע רק על Vulnerabilites שבאמת נטענים לזכרון ובכך לצמצם את ה"רעש" לצוותי ה-Security.

על המחבר

קוראים לי עילי מתעסק בתחום הגנה בענן, בעבר בתחום ה-R&D של כלי הגנה בסייבר. במקביל סטודנט לתואר שני במדמ"ח עם התמחות בבינה מלאכותית.

עבור יצירת קשר תוכלו למצוא אותי:

https://www.linkedin.com/in/ilay-tertman/

מקורות מידע

- https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/security_hardening/scanning-the-system-for-configuration-compliance-and-vulnerabilities_security-hardeningLink 2
- https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/security_hardening/scanning-the-system-for-configuration-compliance-and-vulnerabilities_security-hardening#scanning-the-system-for-vulnerabilities_vulnerability-scanning
- https://www.oligo.security/
- https://www.upwind.io/feed/runtime-context-for-smarter-patch-management-upwind-simplifies-open-source-image-updates
- https://semgrep.dev/
- https://www.zaproxy.org/
- https://github.com/snyk/cli
- https://github.com/anchore/syft
- https://github.com/anchore/grype
- https://github.com/advisories
- https://learn.microsoft.com/en-us/defender-vulnerability-management/tvm-software-inventory
- https://en.wikipedia.org/wiki/Nessus_Attack_Scripting_Language
- https://nvd.nist.gov/products/cpe
- https://www.cve.org/
- https://snyk.io/
- https://www.open-scap.org/tools/openscap-base/
- https://github.com/lvyitian/CVE-2024-30078-
- https://www.redhat.com
- https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/security_hardening/scanning-the-system-for-configuration-compliance-and-vulnerabilities_security-hardening#scanning-the-system-for-vulnerabilities_vulnerability-scanning
- https://github.com/amberzovitis/GraphKer