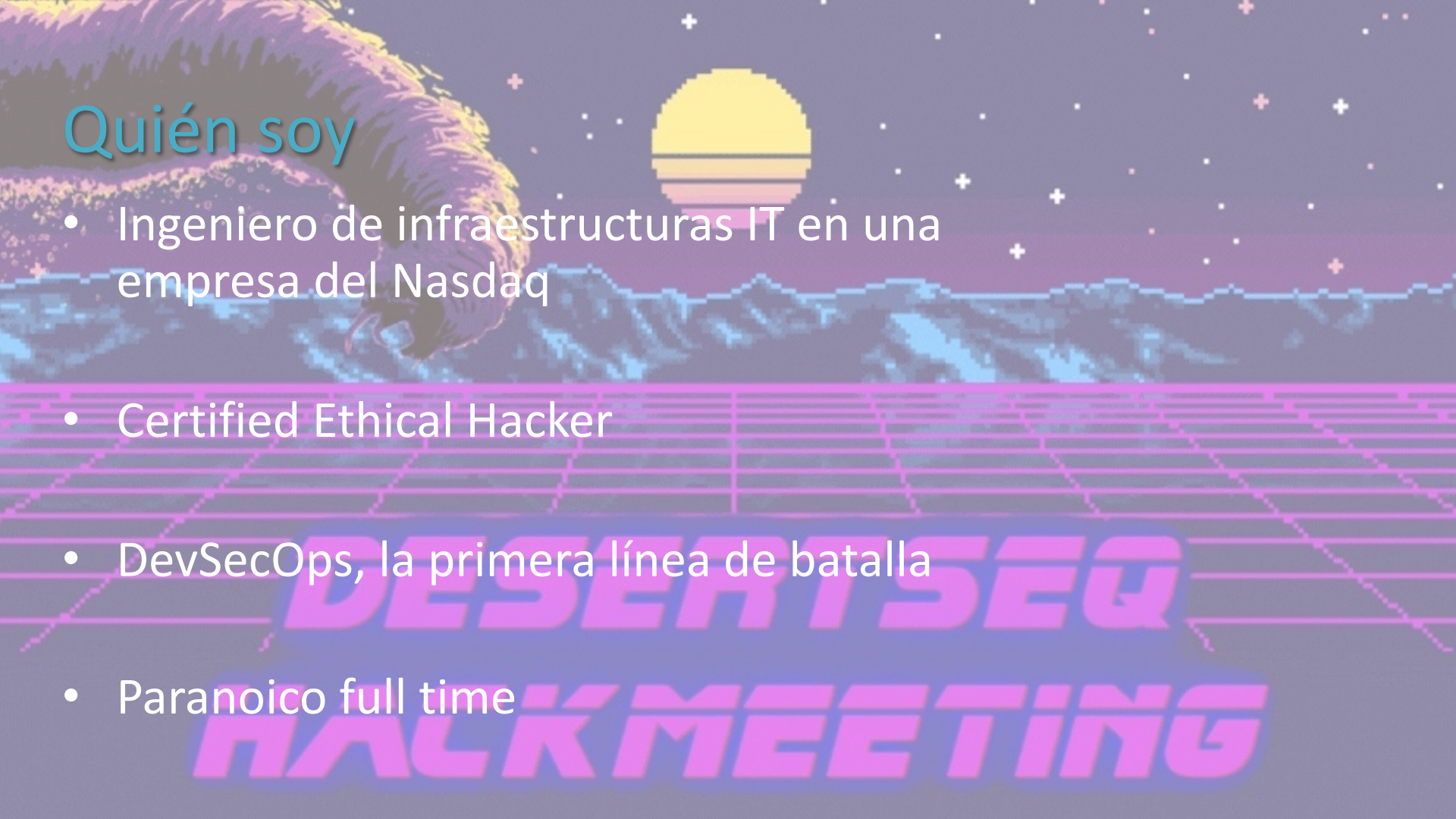
Introducción al análisis estático de código con herramientas Open Source

Jorge Louzao

Quién soy

- Ingeniero de infraestructuras IT en una empresa del Nasdaq
- Certified Ethical Hacker
- DevSecOps, la primera línea de batalla
- Paranoico full time



Análisis Estático de Código

- Security Development Lifecycle define una serie de prácticas para mejorar la seguridad y el cumplimiento de requisitos de las aplicaciones.
- Esto es una práctica común en grandes empresas de software que debería aplicarse en empresas de cualquier tamaño, adaptando la metodología los recursos de los que se disponga.
- En las charlas que he dado en los últimos tiempo sobre SDL he descubierto que los alumnos saben de que se trata porque sus profesores se lo han mostrado pero no son conscientes de la importancia real que esto tiene.

Análisis Estático de Código

- Ante la escalada de ciberataques para comprometer organizaciones, usuarios y datos SDL debería ser de obligada aplicación en
  - Aplicaciones implementadas en un entorno empresarial
  - Aplicaciones que procesan información de identificación personal (PII) u otro tipo de información confidencial
  - Aplicaciones que se comunican frecuentemente a través de Internet u otras redes

Análisis Estático de Código

- Las 5 áreas en las que se divide la metodología SDL son:
  - Formación, directivas y capacidades organizativas
  - Requisitos y diseño
  - Implementación
  - Comprobación
  - Lanzamiento y respuesta

Análisis Estático de Código

- La fase de Implementación se divide en tres procedimientos:
  - Definición de herramientas aprobadas y comprobaciones de seguridad asociadas, opciones de compilado, versiones, etc.
  - Eliminación de funciones y API no seguras, especialmente en código heredado.
  - Y finalmente el análisis estático del código fuente.

Análisis Estático de Código

- El análisis estático de código busca
  - Asegurar que se aplican las directivas de codificación segura.
  - Que no se usan funciones inseguras en el código y ofrecer una alternativa segura.
  - Detección de vulnerabilidades.
  - Examinar funciones críticas como, por ejemplo, las criptográficas.

- El análisis estático de código no substituye la revisión manual del software en busca de fallos de programación y vulnerabilidades.

Análisis Estático de Código

- En cuanto a los puntos débiles
  - Exceso de falsos negativos y la dificultad de configurar los analizadores para ocultarlos.
  - Algunos mensajes de error son un poco incomprensibles.
  - Como encajar el analizador en el flujo de trabajo.
- Con tiempo y entrenamiento esto se supera y obtenemos un beneficio de encontrar errores en una fase temprana.

Análisis Estático de Código

- Aunque esta fase está pensada para formar parte del proceso de integración continua disponemos de herramientas para ayudarnos en la fase de desarrollo.
- Existen decenas de herramientas de pago y open source, en esta charla pretendo centrarme en las Open Source, aunque algunas hayan sido desarrolladas por empresas que hasta fechas más recientes no han destacado precisamente por su implicación en el movimiento del Software Libre

Análisis Estático de Código

- Para obtener información en tiempo real durante el Desarrollo de la aplicación tenemos herramientas como DevSkim, un plugin para Visual Studio y su hermano libre Visual Studio Code, también soporta otros editores como Sublime Text
- https://github.com/Microsoft/DevSkim

Análisis Estático de Código

- DevSkim funciona mediante expresiones regulares y puede ampliarse para ser usado sobre cualquier lenguaje de programación.
- Soporta C/C++, Java, C#, JavaScript y PHP entre otros lenguajes de programación.
- https://github.com/Microsoft/DevSkim/wiki/Writing-Rules

Análisis Estático de Código

```c
#include <stdio.h>

int main()
{
    char str[16];
    gets(str);
    return 0;
}
```

```c
    char str[16];
    gets(str);
    return 0;
```

```c
#include <stdio.h>

int main()
{
    char str[16];
    gets(str);
}
```

DevSkim: Change to fgets

DevSkim: Change to gets_s (Recommended for VC++)

```c
    char str[16];
    fgets(str, <size of str>, stdin);
    return 0;
```

Análisis Estático de Código

- Para Python una de mis herramientas favoritas es Bandit.
- Permite el uso de perfiles para buscar solo determinados tipos de vulnerabilidades
- https://github.com/PyCQA/bandit

Análisis Estático de Código

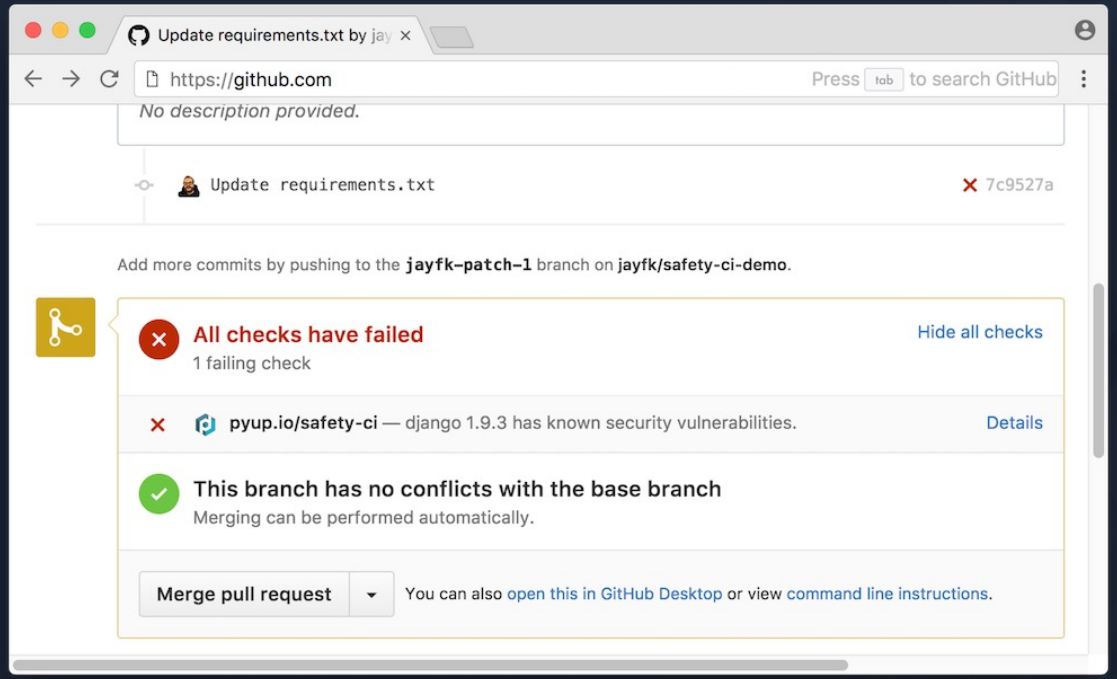
```
root@SkullCanyon:/home# bandit -n 3 -lll /usr/share/sqlmap/sqlmap.py
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 2.7.17
Run started:2020-04-12 14:02:08.458109

Test results:
        No issues identified.

Code scanned:
        Total lines of code: 430
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
Files skipped (0):
root@SkullCanyon:/home#
```

Análisis Estático de Código

- Safety para Python, no es exactamente una herramienta SAST, pero es muy útil para encontrar dependencias con problemas de seguridad
- https://pyup.io/safety/

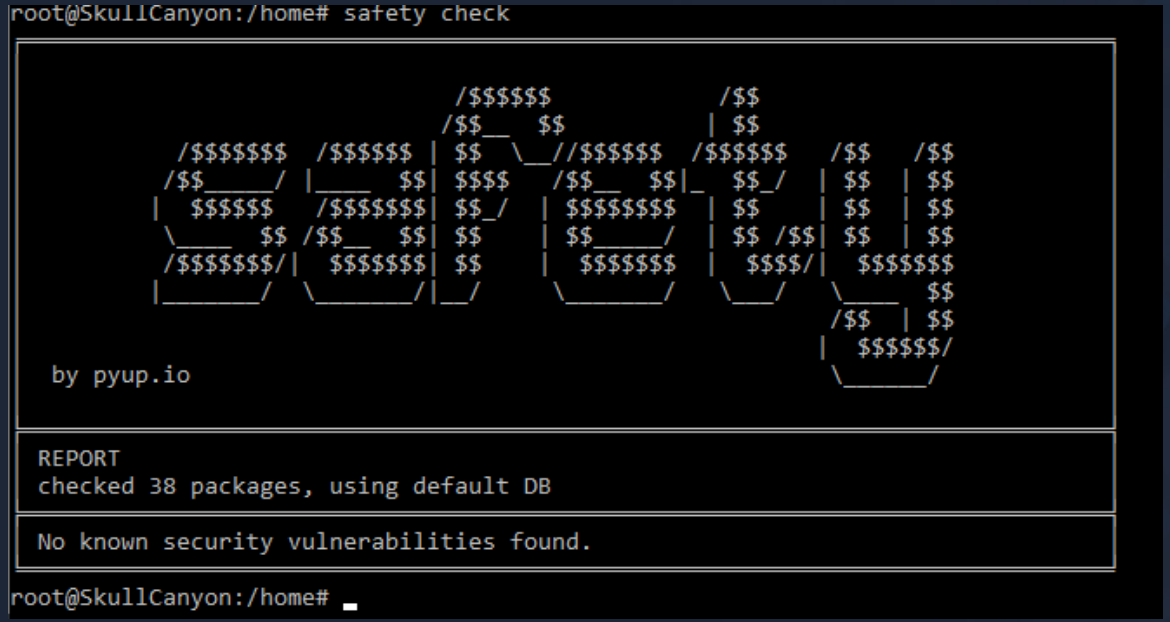
Análisis Estático de Código

- Safety se integra con Github



Análisis Estático de Código

- Y lo podemos ejecutar en nuestro entorno



Análisis Estático de Código

- Para JavaScript tenemos una serie de reglas para NodeSecurity que se integran con Github vía pull requests
- https://github.com/nodesecurity/eslint-plugin-security

Análisis Estático de Código

- Como complemento para JavaScript está la herramienta OAST Retire que incluso tiene una versión en plugin para navegadores bastante útil.
- https://github.com/RetireJS/retire.js

Análisis Estático de Código

Retire.js ☑ Enabled ☐ Show unknown

bootstrap	4.1.1	Found in https://c0r0n4con.com/assets/js/bootstrap.min.js Vulnerability info: High 28236 XSS in data-template, data-content and data-title properties of tooltip/popover CVE-2019-8331 [1] Medium 20184 XSS in data-target property of scrollspy CVE-2018-14041 [1] Medium 20184 XSS in collapse data-parent attribute CVE-2018-14040 [1] Medium 20184 XSS in data-container property of tooltip CVE-2018-14042 [1]
jquery	2.1.4	Found in https://c0r0n4con.com/assets/js/jquery-min.js Vulnerability info: Medium 2432 3rd party CORS request may execute CVE-2015-9251 [1] [2] [3] [4] Medium CVE-2015-9251 11974 parseHTML() executes scripts in event handlers [1] [2] [3] Low CVE-2019-11358 jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution [1] [2] [3]
swfobject	2.3.20130521	Found in https://player.twitch.tv/js/video.d43cebed4a7254ab8bf6.js
swfobject	2.3.20130521	Found in https://player.twitch.tv/js/video.d43cebed4a7254ab8bf6.js
swfobject	2.3.20130521	Found in https://player.twitch.tv/js/video.d43cebed4a7254ab8bf6.js
swfobject	2.3.20130521	Found in https://player.twitch.tv/js/video.d43cebed4a7254ab8bf6.js

Análisis Estático de Código

- Para C/C++ CPPCheck.
- Está disponible para Windows, Linux y Mac a través de brew
- http://cppcheck.sourceforge.net/

Análisis Estático de Código

- CPPCheck detecta:
  - Dead pointers
  - Division by zero
  - Integer overflows
  - Invalid bit shift operands
  - Invalid conversions
  - Invalid usage of STL
  - Memory management
  - Null pointer dereferences
  - Out of bounds checking
  - Uninitialized variables
  - Writing const data

Análisis Estático de Código

- Otra herramienta para C/C++ es Flawfinder, que además se puede integrar en Github
- https://github.com/david-a-wheeler/flawfinder

Análisis Estático de Código

```
root@SkullCanyon:~/nmap# flawfinder ./
Flawfinder version 2.0.11, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
Warning: Skipping directory with initial dot ./.git
Examining ./FPEngine.cc
Examining ./FPEngine.h
Examining ./FPModel.cc
Examining ./FPModel.h
Examining ./FingerPrintResults.cc
Examining
Examining
Examining
Examining
Examining
Examining
Examining
Examining

ANALYSIS SUMMARY:

Hits = 2801
Lines analyzed = 364368 in approximately 10.31 seconds (35341 lines/second)
Physical Source Lines of Code (SLOC) = 224387
Hits@level = [0] 1033 [1] 680 [2] 1822 [3]  64 [4] 228 [5]   7
Hits@level+ = [0+] 3834 [1+] 2801 [2+] 2121 [3+] 299 [4+] 235 [5+]   7
Hits/KSLOC@level+ = [0+] 17.0866 [1+] 12.4829 [2+] 9.45242 [3+] 1.33252 [4+] 1.0473 [5+] 0.0311961
Dot directories skipped = 1 (--followdotdir overrides)
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
See 'Secure Programming HOWTO'
(https://dwheeler.com/secure-programs) for more information.
root@SkullCanyon:~/nmap#
```

Análisis Estático de Código

```
FINAL RESULTS:

./libpcap/lbl/os-sunos4.h:129:  [5] (race) readlink:
  This accepts filename arguments; if an attacker can move those files or
  change the link content, a race condition results. Also, it does not
  terminate with ASCII NUL. (CWE-362, CWE-20). Reconsider approach.
./libpcap/pcap-linux.c:607:  [5] (race) readlink:
  This accepts filename arguments; if an attacker can move those files or
  change the link content, a race condition results. Also, it does not
  terminate with ASCII NUL. (CWE-362, CWE-20). Reconsider approach.
./libz/contrib/untgz/untgz.c:32:  [5] (race) chmod:
  This accepts filename arguments; if an attacker can move those files, a
  race condition results. (CWE-362). Use fchmod( ) instead.
./libz/contrib/untgz/untgz.c:277:  [5] (race) chmod:
  This accepts filename arguments; if an attacker can move those files, a
  race condition results. (CWE-362). Use fchmod( ) instead.
./nbase/nbase_misc.c:829:  [5] (race) readlink:
  This accepts filename arguments; if an attacker can move those files or
  change the link content, a race condition results. Also, it does not
  terminate with ASCII NUL. (CWE-362, CWE-20). Reconsider approach.
./nping/ProbeMode.cc:1752:  [5] (buffer) strncat:
  Easily used incorrectly (e.g., incorrectly computing the correct maximum
  size to add) [MS-banned] (CWE-120). Consider strcat_s, strlcat, snprintf,
  or automatically resizing strings. Risk is high; the length parameter
  appears to be a constant, instead of computing the number of characters
  left.
./nping/ProbeMode.cc:1780:  [5] (buffer) strncat:
  Easily used incorrectly (e.g., incorrectly computing the correct maximum
  size to add) [MS-banned] (CWE-120). Consider strcat_s, strlcat, snprintf,
  or automatically resizing strings. Risk is high; the length parameter
  appears to be a constant, instead of computing the number of characters
  left.

ANALYSIS SUMMARY:

Hits = 7
Lines analyzed = 364368 in approximately 6.12 seconds (59580 lines/second)
Physical Source Lines of Code (SLOC) = 224387
Hits@level = [0] 1033 [1] 680 [2] 1822 [3]  64 [4] 228 [5]   7
Hits@level+ = [0+] 3834 [1+] 2801 [2+] 2121 [3+] 299 [4+] 235 [5+]   7
Hits/KSLOC@level+ = [0+] 17.0866 [1+] 12.4829 [2+] 9.45242 [3+] 1.33252 [4+] 1.0473 [5+] 0.0311961
Dot directories skipped = 1 (--followdotdir overrides)
Minimum risk level = 5
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
See 'Secure Programming HOWTO'
(https://dwheeler.com/secure-programs) for more information.
```

Análisis Estático de Código

- Para Java destacan SonarSource Community y Find Security Bugs
- https://www.sonarsource.com/plans-and-pricing/community/
- https://find-sec-bugs.github.io/

Análisis Estático de Código

- SonarSource Community soporta más lenguajes de programación, como PHP, C#, Go, Flex, Ruby, JavaScript, VB.NET entre otros.
- Para CI funciona con Jenkins, Bamboo y Azure DevOps.
- Se integra con IntelliJ, Visual Studio Code, Eclipse y Visual Studio.

Análisis Estático de Código

- Find Security Bugs está especializado en auditar aplicaciones web desarrolladas en Java.
- Se integra con Jenkins y SonerQube, además de los IDE Eclipse, IntelliJ, NetBeans y Android Studio
- Integración como tarea en Ant y Maven
- Cubre el OWASP Top 10 y CWE de Mitre

Análisis Estático de Código

- Otro tipo de analizador, el que busca copyrights y licencias de las librerías que empleamos en nuestro software.
- https://github.com/nexB/scancode-toolkit/
- https://owasp.org/www-project-dependency-check/

Análisis Estático de Código

- Dependency check de OWASP se integra con Jenkins, SonarQube además de tener versión para ejecutar desde la línea de comandos, como tarea de Ant o plugin de Maven.

Análisis Estático de Código

- Fossology comprueba el cumplimiento con las licencias open source.
- https://github.com/fossology/fossology

Análisis Estático de Código

- Whitesource, no es software libre, pero es mi herramienta favorita para encontrar vulnerabilidades en las bibliotecas open source empleadas en nuestras aplicaciones.
- Se puede descargar la versión gratuita de Whitesource Checker aquí:
- https://www.whitesourcesoftware.com/vulnerability_checker

Análisis Estático de Código

- Whitesource está disponible gratuitamente en Github para analizar nuestros repositorios en busca de vulnerabilidades en componentes open source.
- https://github.com/marketplace/whitesource-bolt
- También en Azure DevOps
- https://marketplace.visualstudio.com/items?itemName=whitesource.ws-bolt

Análisis Estático de Código

- Otras herramientas que actualmente estoy probando:
  - PMD https://pmd.github.io/
  - TFSec https://github.com/liamg/tfsec

Análisis Estático de Código

- PMD es otro analizador estático de código que soporta lenguajes comunes y otros más especializados como Apex y Visualforce, Modelica, PLSQL o Apache Velocity
- También incluye detector de copy&paste para encontrar código duplicado en nuestros fuentes

Análisis Estático de Código

- TFSec es un analizador especializado en plantillas de Terraform
- Encuentra información sensible
- Violaciones de las recomendaciones de mejores prácticas de AWS, Azure y Google Cloud

Análisis Estático de Código

```
tfsec ./example

4 potential problems detected:

Problem 1

  [AWS006] Resource 'aws_security_group_rule.my-rule' defines a fully open ingress security group rule.
  /Users/liamg/example/main.tf:4

      1 |
      2 | resource "aws_security_group_rule" "my-rule" {
      3 |     type        = "ingress"
      4 |     cidr_blocks = ["0.0.0.0/0"]
      5 | }
      6 |
      7 | resource "aws_alb_listener" "my-alb-listener"{

Problem 2

  [AWS004] Resource 'aws_alb_listener.my-alb-listener' uses plain HTTP instead of HTTPS.
  /Users/liamg/example/main.tf:9

      6 |
      7 | resource "aws_alb_listener" "my-alb-listener"{
      8 |     port     = "80"
      9 |     protocol = "HTTP"
     10 | }
     11 |
     12 | resource "aws_db_security_group" "my-group" {
```

Análisis Estático de Código

- Una lista más larga de aplicaciones libres y de pago:
- https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

FIN

Me podéis encontrar en:
https://masto.louzao.network/@louzao
https://louzao.network