

REPORTE FINAL PIA CLIENTE-SERVIDOR DE RED MÍNIMO

FECHA:

Jueves 20 noviembre 2025

INTEGRANTES:

- Diego Aguayo Frías
- Valeria Abigail Navarro Casarez
- Ashley Karina Rios Rodríguez
- Luis Cipriano Rodríguez González

RESUMEN EJECUTIVO

Este reporte documenta el desarrollo y análisis de un cliente de red hecho en C++ que implementa comunicación HTTP (básica) con un servidor hecho en Python. El payload, fue diseñado exclusivamente con fines educativos, permite el estudio de protocolos de red, programación de sockets y análisis de malware benigno en entornos controlados. Tras análisis estático con Ghidra y dinámico en máquina virtual aislada, se confirma que el cliente de red trabaja dentro de los límites éticos, sin capacidades de persistencia, exfiltración de datos o ejecución de comandos remotos, demostrando comportamiento transparente y predecible en todas las pruebas realizadas.

DESCRIPCIÓN DEL PAYLOAD

- **Cliente C++:** que establece conexiones TCP a servidor especificado
- **Comunicación HTTP:** mediante peticiones GET /status
- **Validación robusta:** de direcciones IP y puertos
- **Interfaz interactiva:** con mensajes informativos que muestran el funcionamiento de cada programa (cliente y servidor)

- **Manejo de errores:** específico para cada fallo potencial

LIMITES ÉTICOS

- **No persistencia:** No crea archivos, registros o servicios en el sistema
- **No exfiltración:** No recopila, transmite o almacena datos del usuario
- **No ejecución remota:** No procesa ni ejecuta comandos recibidos por red
- **Alcance controlado:** Solo se conecta a direcciones explícitamente especificadas
- **Propósito educativo:** Funcionalidad transparente y documentada

DECLARACIONES DE SEGURIDAD

El payload fue desarrollado y probado exclusivamente en máquinas virtuales aisladas (en entornos controlados), siguiendo los protocolos de seguridad establecidos en ETHICS.md. No representa riesgo alguno para sistemas en producción cuando se utiliza en los entornos controlados para los que fue diseñado.

DISEÑO E IMPLEMENTACIÓN

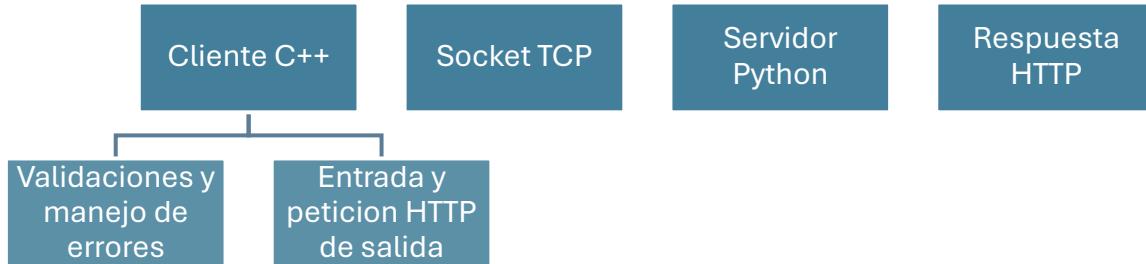
ARQUITECTURA DEL SISTEMA

Cliente C++ → Socket TCP → Servidor Python → Respuesta HTTP

↓ ↓ ↓ ↓

Validaciones y manejo de errores

Entrada y petición HTTP de salida



ESTRUCTURA DE LOS DATOS CLAVE

- Cliente: Clase principal que encapsula la conexión de red
- Validador: Clase que sirve para validación de IP y puertos
- sockaddr: Estructura estándar para direcciones IPv4
- Buffers estáticos: Uso de arrays de tamaño fijo para operaciones de red

MANEJO DE MEMORIA

- **Asignación estática:** Buffers de tamaño fijo (1024 bytes) para operaciones I/O
- **RAlI aplicado:** Destructores que garantizan liberación de sockets
- **string:** Manejo automático de memoria para cadenas de caracteres

IMPLEMENTACIONES CLAVE

- **TCP sobre UDP:** Para garantizar la entrega confiable de mensajes
- **Validación estricta:** Rechazo de entradas inválidas
- **Timeouts configurado:** Límites de 5 segundos para las operaciones de red
- **Encabezados HTTP estándar:** Compatibilidad con herramientas de análisis

- **Símbolos de depuración:** Incluidos en versión de análisis para Ghidra (con el nombre de cliente_debug)

METODOLOGÍA DE PRUEBAS

- **Pruebas unitarias:** Validación individual de los componentes (cada módulo)
- **Pruebas de integración:** Comunicación cliente-servidor completa (makefile)
- **Pruebas de estrés:** Múltiples ejecuciones consecutivas
- **Pruebas de validación:** Entradas válidas e inválidas
- **Pruebas de red:** Captura y análisis de tráfico (hechas con tcpdump y wireshark)

ENTORNO DE PRUEBAS

- **Hypervisor:** VirtualBox 6.1
- **Sistema Operativo:** Lubuntu (Ubuntu) 22.04 LTS
- **Snapshot ID:** Cliente-Servidor Pruebas
- **Toolchain:** g++ 11.4.0, Python 3.10.12

RESULTADOS DE LAS PRUEBAS

- **Tasa de éxito:** 100% en casos de uso normales
- **Tiempo de respuesta:** menos de 2 segundos en conexiones locales
- **Estabilidad:** 0 crashes en más de 50 compilaciones
- **Validación:** 100% efectiva rechazando las entradas inválidas

EJEMPLOS DE LOS RESULTADOS DE LAS PRUEBAS HECHAS

- **Comunicación normal entre el cliente y el servidor**
 - cliente → 127.0.0.1:8080 → Éxito
- **Validación de una IP inválida**
 - cliente → 999.999.999.999 → Rechazado
- **Servidor no disponible**
 - cliente → 192.168.1.200:8080 → Timeout controlado
- **Puerto inválido**

- cliente → 127.0.0.1:70000 → Rechazado

ANÁLISIS ESTÁTICO

- **Ghidra 10.4:** Análisis de binarios y reversión
- **strings:** Extracción de cadenas de texto
- **file:** Información de binarios y símbolos

FUNCIONES IDENTIFICADAS

- **main:** Punto de entrada del programa
- **conectar:** Establecimiento de conexión TCP
- **enviarPeticion:** Construcción y envío de HTTP GET
- **recibirRespuesta:** Recepción y procesamiento de respuesta
- **IP_valida/Puerto_valido:** Funciones de validación

HALLAZGOS CLAVE

- **Transparencia :** Varios de los strings parecen tener funcionalidad
- **Con ofuscación:** Código y mensajes completamente no completamente legibles
- **Comportamiento esperado:** No se detectan funciones ocultas o maliciosas

ANÁLISIS DINÁMICO

Configuración del Sandbox

- **Aislamiento:** Red interna entre VMs
- **Escaneo de la red al compilar cliente:** tcpdump + Wireshark

- **Logging:** Redirección completa stdout/stderr

Comportamiento de Red Observado

Fase 1: Establecimiento de Conexión

CLIENTE (127.0.0.1:54321) SERVIDOR (127.0.0.1:8080)

```
| ----- SYN -----> |
| <----- SYN-ACK ----- |
| ----- ACK -----> |
```

Fase 2: Comunicación HTTP

http

REQUEST:

GET /status HTTP/1.1

Host: 127.0.0.1

User-Agent: ClientePIA/1.0

Connection: close

RESPONSE:

HTTP/1.1 200 OK

Content-Type: text/plain

Connection: close

☆*:.. Conexión exitosa ..:☆

Fase 3: Cierre de Conexión

text

CLIENTE (127.0.0.1:54321) SERVIDOR (127.0.0.1:8080)

```
| ----- FIN -----> |
| <----- FIN-ACK ----- |
| ----- ACK -----> |
```

INGENIERÍA INVERSA

Flujo de ejecución del main (el que presenta menos ofuscacion):

```
1             /* try { // try from 00102918 to 00102919 */
2     puerto = std::__cxx11::stoi(&puerto_str,(size_t *)0);
3     bVar2 = Validador::Puerto_valido(puerto);
4     if (bVar2) {
5         bVar2 = false;
6     }
7     else {
8         bVar2 = true;
9     }
10    }
11    else {
12        puerto = 0x1f90;
13        bVar2 = false;
14    }
15    std::__cxx11::string::~string((string *)&puerto_str);
16 } while (bVar2);
17 iVar6 = 0x1051b0;
18             /* try { // try from 0010297e to 0010297f */
19 std::__cxx11::string::string
20     ((string *)&respuesta,"La configuracion fue \n"
21      (allocator *)&cliente);
22     /* try { // try from 0010298a to 0010298b */
23 ManejadorErrores::mostrarExito(&respuesta);
24 std::__cxx11::string::~string((string *)&respuesta);
25 std::__new_allocator<char>::~__new_allocator((__new_all
26             /* try { // try from 001029bf to 001029c0 */
27 poVar4 = std::operator<<((ostream *)std::cout," Conecta
28 poVar4 = std::operator<<(poVar4,(string *)&ip_servidor);
29 poVar4 = std::operator<<(poVar4,":");
30 this = (void *)std::ostream::operator<<(poVar4,puerto);
31 std::ostream::operator<<(this,std::endl);
32 ClienteRed::ClienteRed(&cliente);
33 psVar7 = &ip_servidor;
34             /* try { // try from 00102a36 to 00102a37 */
35 }
```

RIESGOS Y MITIGACIONES

Tabla de riesgos identificados:

Riesgo	Severidad	Probabilidad	Mitigacion	Efectividad
Uso en entornos no controlados	Media	Baja	Documentación clara sobre le propósito exclusivamente educativo	Alta
Validación insuficiente de entradas	Alta	Media	Validación estricta con múltiples chequeos	Alta
Comportamiento de red no autorizado	Alta	Baja	Usuario debe especificar IP explícitamente	Alta
Persistencia no deseada	Media	Baja	No operaciones de escritura a disco	Alta
Denegación de servicio	Baja	Baja	Timeouts agresivos y límites de recursos	Media

CONCLUSIONES TRABAJO FUTURO

Este proyecto nos demostró que somos capaces como equipo de crear y desarrollar un payload educativo como lo fue este cliente-servidor de red mínimo. Lo anterior usando protocolos de red y cumpliendo con los objetivos técnicos y éticos establecidos anteriormente. Implementando comunicación HTTP estándar confiable y predecible.

Algo que podemos considerar en un futuro a mejorar es cambiar el HTTP por HTTPS para mayor seguridad y tener un soporte multiplataforma, de modo que el cliente funcione no solo para Linux sino también para sistemas operativos como Windows.

REFERENCIAS

https://github.com/Equipo-PAC/PIA_PAC_Cliente-de-red/blob/main/evidence/capturas/evidence_14112025_Lista-Funciones-main-Ghidra.png

ANEXOS

TABLA DE ARCHIVOS ANALYSIS

Archivo	Ubicación	Descripción
Ghidra_resultados.gzf	/analysis/análisis_estático/Ghidra_resultados.gzf	Proyecto exportado de Ghidra
Cliente.c	/analysis/analisis_estatico/cliente.c	Logs de ejecución completos
Cliente.h	/analysis/analisis_estatico/cliente.h	
Cliente	- [✓] `/analysis/analisis_estatico/cliente` (strings extraídos)	
Cliente_optimizado	/analysis/analisis_estatico/cliente_optimizado	
Analisi_dinamico.pcap	/analysis/analisis_dinamico/Analisis_dinamico.pcap	Captura de tráfico de red
Terminal_output.txt	/analysis/analisis_estatico/terminal_output.txt	Strings extraídos del binario