UNIVERSITÀ DI PAVIA

Carbanak: A Technical and Economic Analysis of the APT

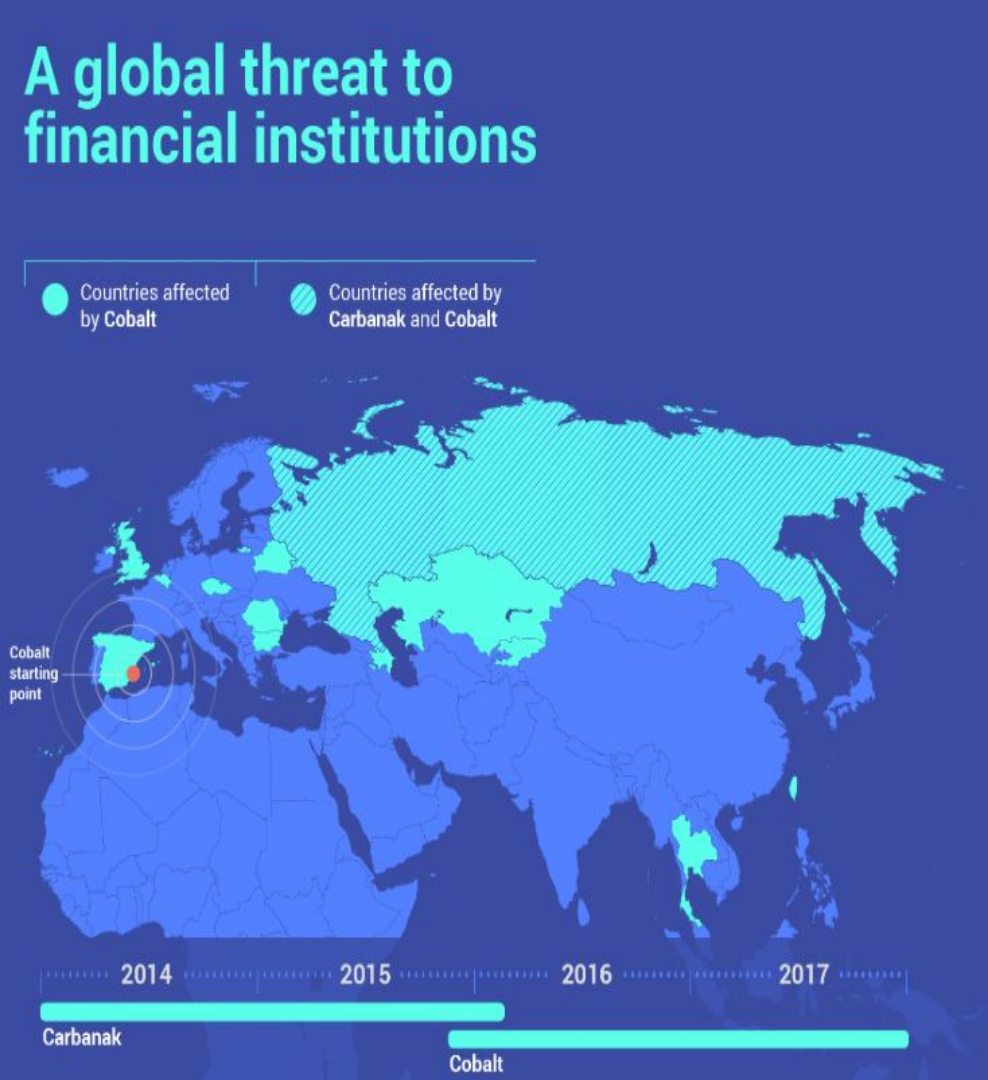
Team 4: Olli Ilomäki, Navid Jalili and Sergiu Francisc

ollipaavojuhan.ilomaeki01@universitadipavia.it navid.jalili01@universitadipavia.it sergiuandrei.francisc01@universitadipavia.it

University of Pavia – Department of Electrical, Computer and Biomedical Engineering

What's Carbanak

- **APT** (Advanced Persistent Threat)
  - Acting as an insider
- Malware
  - Trojan used from 2013 to steal money from bank accounts
- Financial effect:
  - Over 1 Billion dollars stolen over time

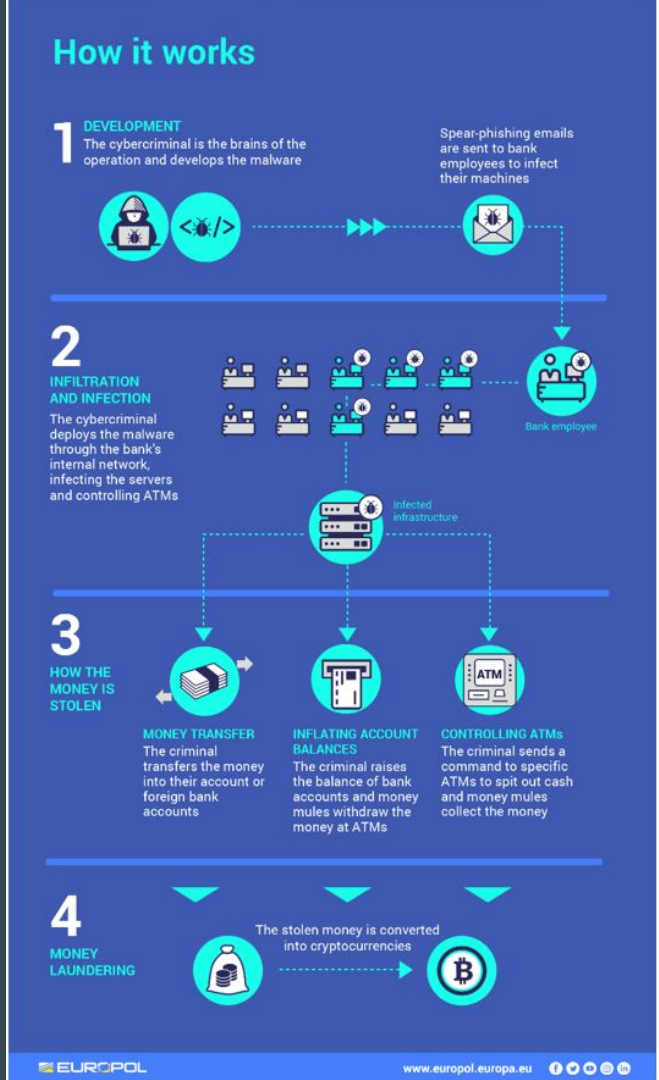


Business Impact

- Headlines
  - Hundreds of banks affected
  - Lost huge amount of cash in dollars
- Behind the headlines are the most crucial pain
  - Indirect costs: **Reputation, trust, operative costs, delays, insurance**
- Clients who have money are ***concerned*** and ***finding alternatives***

What actually happened

- Carbanak acting as a middle-man ***(BPC)***
  - Access through phishing emails
  - not recognizable
  - Learning processes and credentials to use
  - Balance pumping before withdrawal or bank transfer
    - ***Remote Access Tool***
- The whole financial system hacked
- Business process left with zero trust to credentials and actions build for transactions

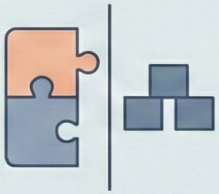


Cyber Kill Chain by Lockheed Martin

- Effective for APT's just like Carbanak
- Main point
  - Identification and prevention of cyber intrusions activity
  - You can interrupt the attack in any stage

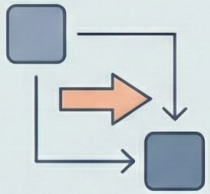
TECHNICAL DEEP DIVE

FEATURES



DYNAMIC API RESOLUTION VS STATIC API IMPORT

COMMAND & CONTROL COMMUNICATION



LATERAL MOVEMENT (LIVING OFF THE LAND)

PERSISTENCE

Features

A Type of Banking Trojan

Derived from the Carberp source code, expanding on its core capabilities

Named Carbanak by Kaspersky Team since it is based on Carberp and the name of the configuration file is “anak.cfg”. (Feb 2015)

Multi-language toolset, using various languages and scripts across different attack stages

Functions as a remote backdoor

Engineered for long-term monitoring and data theft within targeted internal systems

Dynamic API Resolution vs Static API Import

Static API Import

A normal import statement in many programming languages—such as C, Java, C#, etc.—that lets you use classes or functions from a package without writing the full package name every time.

```c
#include <stdio.h>
#include <gsl/gsl_sf_gamma.h>

int main()
{
    double result = gsl_sf_gamma(5.0); // Calculates (5-1)! = 24
    printf("Result: %f\n", result);
    printf("\n");
    return 0;
}
```

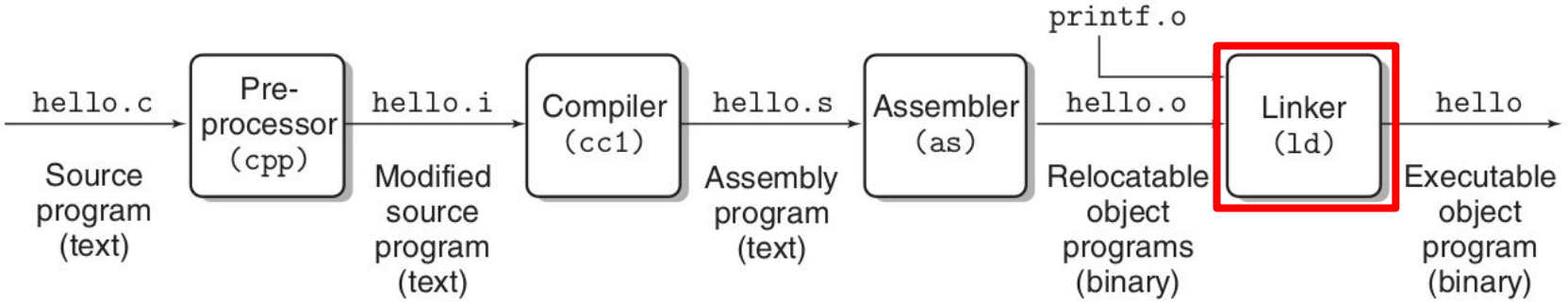


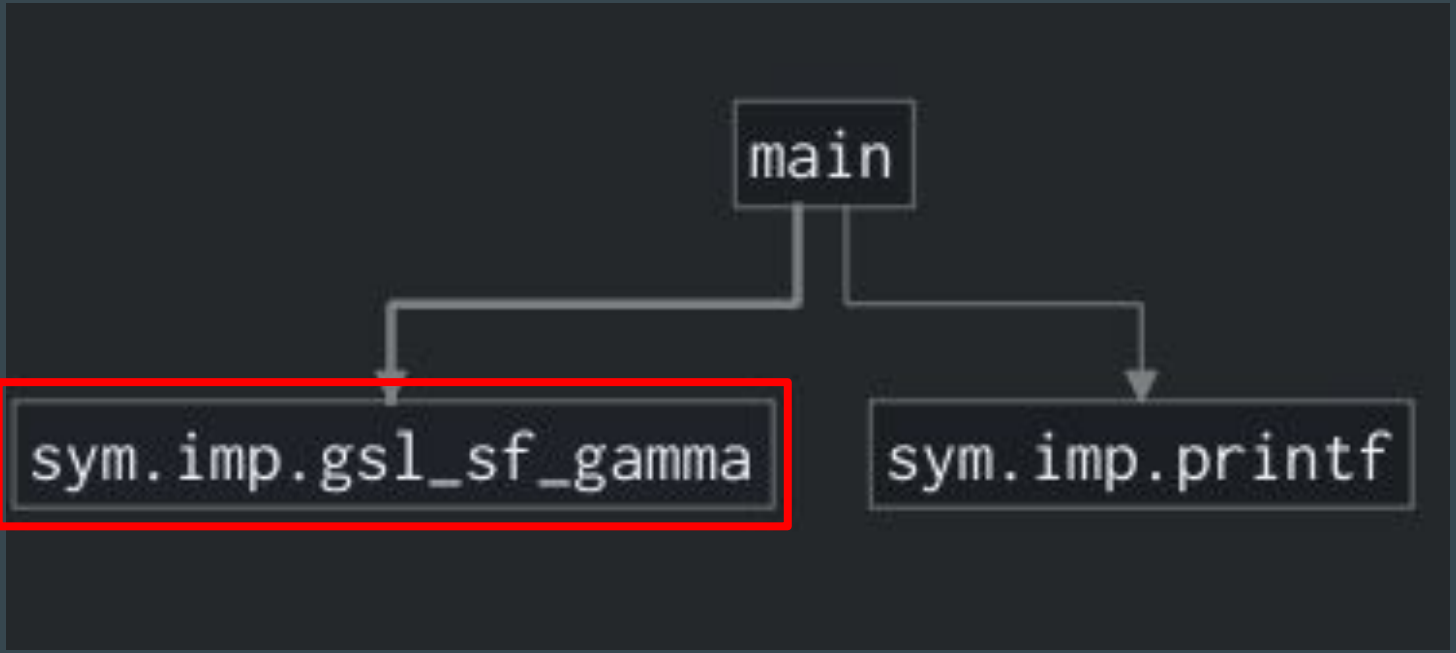
Figure 1.3 **The compilation system.**

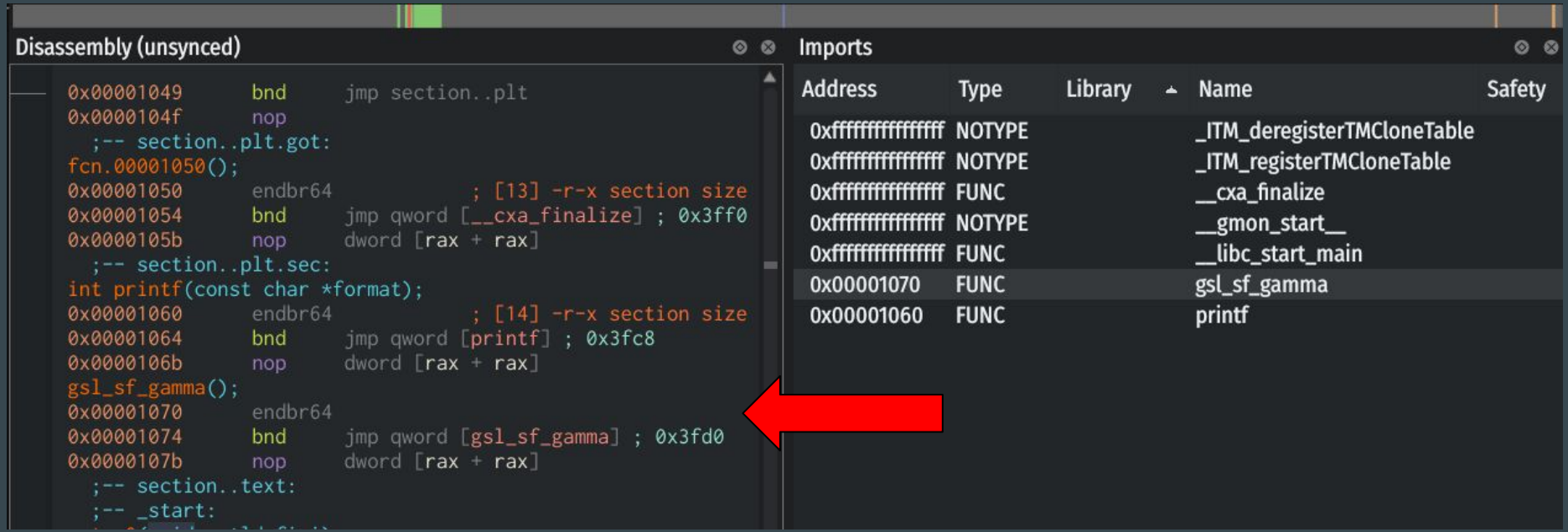
```
linux> gcc -o hello hello.c
```

The linker takes your compiled object file (main.o), which has unresolved calls (like to gsl_sf_gamma or printf), and merges it with the machine code from external library object files, such as printf.o.

The executable is now self-contained, as the function code has been imported statically during the linking phase.

Global Callgraph




Disassembly (unsynced)
0x00001049 bnd jmp section..plt
0x0000104f nop
;-- section..plt.got:
fcn.00001050();
0x00001050 endbr64 ; [13] -r-x section size
0x00001054 bnd jmp qword [__cxa_finalize] ; 0x3ff0
0x0000105b nop dword [rax + rax]
;-- section..plt.sec:
int printf(const char *format);
0x00001060 endbr64 ; [14] -r-x section size
0x00001064 bnd jmp qword [printf] ; 0x3fc8
0x0000106b nop dword [rax + rax]
gsl_sf_gamma();
0x00001070 endbr64
0x00001074 bnd jmp qword [gsl_sf_gamma] ; 0x3fd0
0x0000107b nop dword [rax + rax]
;-- section..text:
;-- _start:
Imports
Address Type Library Name Safety
0xffffffffffffffff NOTYPE _ITM_deregisterTMCloneTable
0xffffffffffffffff NOTYPE _ITM_registerTMCloneTable
0xffffffffffffffff FUNC __cxa_finalize
0xffffffffffffffff NOTYPE __gmon_start__
0xffffffffffffffff FUNC __libc_start_main
0x00001070 FUNC gsl_sf_gamma
0x00001060 FUNC printf

Dynamic API Resolution

A technique where Windows API function addresses are resolved at runtime, instead of being imported and declared upfront when the program is compiled or loaded.

Next, let’s take a look at how these two approaches differ.

Static API Import

APIs are declared upfront via linking

Imports are visible in the PE header / IAT

Easily detectable by AV

Dynamic API Resolution

APIs are looked up in-memory at runtime

No API names visible in the executable

Harder to detect with static analysis

Hey system, I'll need VirtualProtect and WriteProcessMemory, here's the list in advance.

I'll decide what I need while I'm running, and quietly grab those functions when the time comes.

/botep/core/source/winapi.cpp

```
const char* namesDll[] =
{
    _CT_("kernel32.dll"), //KERNEL32 = 0
    _CT_("user32.dll"), //USER32 = 1
    _CT_("ntdll.dll"), //NTDLL = 2
    _CT_("shlwapi.dll"), //SHLWAPI = 3
    _CT_("iphlpapi.dll"), //IPHLPAPI = 4
    _CT_("urlmon.dll"), //URLMON = 5
    _CT_("ws2_32.dll"), //WS2_32 = 6
    _CT_("crypt32.dll"), //CRYPT32 = 7
    _CT_("shell32.dll"), //SHELL32 = 8
    _CT_("advapi32.dll"), //ADVAPI32 = 9
    _CT_("gdiplus.dll"), //GDIPLUS = 10
```

```
#ifdef ON_CODE_STRING
    #define _CT_(X) ("BS" X "ES")  // Wraps with "BS" and "ES" markers
```

DLL names are obfuscated at compile-time with "BS" and "ES" markers

this is only part of libraries used by the Carbanak trojan bot

/botep/core/source/api_func_hash.h

```
#define hashExitProcess 0x07d96c33
#define hashGetComputerNameA 0x0614e3b1
#define hashGetKeyboardState 0x0951c1b5
#define hashGetCurrentThreadId 0x02d3ab94
#define hashGetKeyboardLayout 0x059e7754
#define hashToAsciiEx 0x0899a6f8
#define hashGetCommandLineA 0x0c348a51
#define hashGetCurrentProcess 0x0b601193
#define hashSleep 0x005a2bc0
fine hashLoadLibraryA 0x0aadf0f1
fine hashLoadLibraryW 0x0aadf0c7
fine hashFreeLibrary 0x02b40339
fine hashGetProcAddress 0x0b3c1d03
fine hashOpenSCManagerA 0x0284af31
fine hashOpenSCManagerW 0x0284af27
fine hashOpenServiceA 0x09fa3a01
efine hashOpenServiceW 0x09fa3a37
```

/botep/core/source/misc.cpp

```
1  uint CalcHash( const byte* ptr, int c_ptr )
2  {
3      uint hash = 0;
4      if( ptr && c_ptr > 0 )
5      {
6          for( int i = 0; i < c_ptr; i++, ptr++ )
7          {
8              hash = (hash << 4) + *ptr;
9              unsigned t;
10             if( (t = hash & 0xf0000000) != 0)
11                 hash = ((hash ^ (t >> 24)) & (0x0fffffff));
12         }
13     }
14     return hash;
15 }
```

CARBANAK uses PJW string hash function to locate Windows API functions without disclosing their names.

Key Concepts

PEB (Process Environment Block)

A structure in each process containing information about loaded modules (DLLs).

Can be accessed via the fs:[0x30] (x86) or gs:[0x60] (x64) register.

PEB_LDR_DATA & LDR_MODULE

These structures help enumerate loaded modules in a process.

Export Address Table (EAT)

A section in a DLL that holds function names and addresses.

Import Address Table (IAT)

When a program explicitly links against a DLL, a.

Process Environment Block (PEB) - (winternl.h)

C++ Copy

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  BYTE                          Reserved2[1];
  PVOID                         Reserved3[2];
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  PVOID                         Reserved5;
  ULONG                         Reserved6;
  PVOID                         Reserved7;
  ULONG                         Reserved8;
  ULONG                         AtlThunkSListPtr32;
  PVOID                         Reserved9[45];
  BYTE                          Reserved10[96];
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
  BYTE                          Reserved11[128];
  PVOID                         Reserved12[1];
  ULONG                         SessionId;
} PEB, *PPEB;
```

How to Manually Resolve API Addresses

1. Walk the PEB to locate loaded DLLs.
2. Find kernel32.dll (or any needed DLL).
3. Parse the EAT of the DLL.
4. Find the function address by matching its hash to the stored hash.

/botep/core/source/winapi.cpp

```
PPEB GetPEB()
{
#ifdef _WIN64
    return (PPEB)__readgsqword(0x60);
#else
    PPEB PEB;
    __asm
    {
        mov eax, FS:[0x30]
        mov [PEB], eax
    }
    return PEB;
#endif
}
```

Gets the PEB address

```
PPEB Peb = GetPEB();

    PPEB_LDR_DATA LdrData = Peb->Ldr;
    PLIST_ENTRY Head = &LdrData->ModuleListLoadOrder;
    PLIST_ENTRY Entry = Head->Flink;
```

Iterates over LdrData to find kernel32.dll

```
while ( Entry != Head )
    {
        PLDR_DATA_TABLE_ENTRY LdrData = CONTAINING_REC(
```

```
__declspec(dllexport) ULONG_PTR WINAPI ReflectiveLoader(LPVOID lpParameter)
{
    // the functions we need
    typeLoadLibraryA pLoadLibraryA = N
    typeGetProcAddress pGetProcAddress
    typeVirtualAlloc pVirtualAlloc = N
    typeNtFlushInstructionCache pNtFlu

    USHORT usCounter;

    // the initial location of this im
    ULONG_PTR uiLibraryAddress;
    // the kernels base address and la
    ULONG_PTR uiBaseAddress;
```

Once kernel32.dll is found, it manually parses its EAT to find the addresses of LoadLibraryA and GetProcAddress using their pre-calculated hashes

/botep/core/source/winapi.cpp

```cpp
HMODULE GetDllBase( uint dllHash )
{
    PPEB Peb = GetPEB();

    PPEB_LDR_DATA LdrData = Peb->Ldr;
    PLIST_ENTRY Head = &LdrData->ModuleListLoadOrder;
    PLIST_ENTRY Entry = Head->Flink;

    while ( Entry != Head )
    {
        PLDR_DATA_TABLE_ENTRY LdrData = CONTAINING_RECORD( Entry, LDR_DATA_TABLE_ENTRY, InLoadOrderModuleList );

        //□□□□□□□□□□□□ wchar_t □ char
        char name[64];
        if( LdrData->BaseDllName.Length < sizeof(name) - 1 )
        {
            int i = 0; //□ □□□□□ □□□ □□□□□ □□□□□ □□□□□ □□□
            while( LdrData->BaseDllName.Buffer[i] && i < sizeof(name) - 1 )
            {
                name[i] = (char)LdrData->BaseDllName.Buffer[i];
                i++;
            }
            name[i] = 0;
            Str::Upper(name);
            uint hash = Str::Hash( name, i );
            if( dllHash == hash )
            {
                return (HMODULE)LdrData->DllBase;
            }
        }
        Entry = Entry->Flink;
    }
    return nullptr;
}
```

Malware avoids calling GetProcAddress() directly since security tools monitor it.

It works by manually parsing the export table of a loaded DLL (like kernel32.dll)

Command & Control Communication

Command & Control Communication

Carbanak communicates with its C2 infrastructure using two primary methods: a Custom Binary Protocol and a Pseudo-HTTP Protocol.

Pseudo-HTTP Protocol

Messages are delimited with the ‘|’ character.

A message starts with a host ID composed by concatenating a hash value generated from the computer’s hostname and MAC address to a string likely used as a campaign code. Once the message has been formatted, it is sandwiched between an additional two fields of randomly generated strings of upper and lower case alphabet characters.

```
xVjMTzzJthISUXrHhrSEtaUvnMlC|myserih0cf3f75290c7e5905|lNCjyGFgGCAexwNgv
```

Command Polling Message

```
qIPhfIFGUXDQXDMbgnlFmpZU|myserih0cf3f75290c7e5905|data=listprocess|process=svc
host.exe|idprocess=4294967295|gmGKiJcHQzYLTje
```

Command Response Message

Messages are encrypted using Microsoft's implementation of RC2 in CBC mode with PKCS#5 padding. The encrypted message is then Base64 encoded, replacing all the '/' and '+' characters with the '.' and '-' characters, respectively.

Custom Binary Protocol

If a message is larger than 150 bytes, it is compressed with an unidentified algorithm. If a message is larger than 4096 bytes, it is broken into compressed chunks.

```
typedef struct binaryProtocolMsg{
    uint8_t cmdId;
    uint8_t flag;
    uint32_t messageLength;
    uint16_t chunkLength;
    uint16_t chunkIndex;
    uint8_t chunkFlag; //compressed, more chunks coming
    uint32_t unknown;
    uint8_t hdrXORKey1[4]; //random, unused by some versions
    uint8_t hdrXORKey2[5]; //random, unused by some versions
    uint8_t chunkData[chunkLength];
}
```

Lateral Movement (Living off the Land)

Lateral Movement (Living off the Land - LotL)

Lateral movement is a stage in a cyberattack where the threat actor pivots from the initially compromised system to other systems within the internal network.

Living off the Land (LotL) is a method of attack where the adversaries use legitimate, pre-installed software, system tools, and administrative utilities that are already present on the target operating system (e.g., Windows or Linux).

In Carbanak intrusions against banks, attackers often manipulated Active Directory and financial applications directly through legitimate tools. For instance, they used remote desktop utilities and PowerShell scripts to alter account balances or initiate fraudulent transfers.

Types of LOTL Attacks

Binary Planting: Replacing a legitimate DLL file with a malicious one

Registry Run Keys: Insertion of malware into the registry key that contains instructions to execute a specific program at startup

Fileless Malware: Usage of scripting languages such as PowerShell or Windows Management Instrumentation (WMI) to execute code directly in memory without any interaction with the file-system.

Persistence

Persistence

To maintain persistence, the group create new services open on a new tab. They also add programs to a startup folder that can be referenced with a registry run key.

Persistence

The backdoor drops the following copies of itself into the affected system:

%System%\Com\svchost.exe

It creates the following folders:

%All Users Profile%\Application Data\Mozilla

It injects codes into the following process:

svchost.exe

Persistence

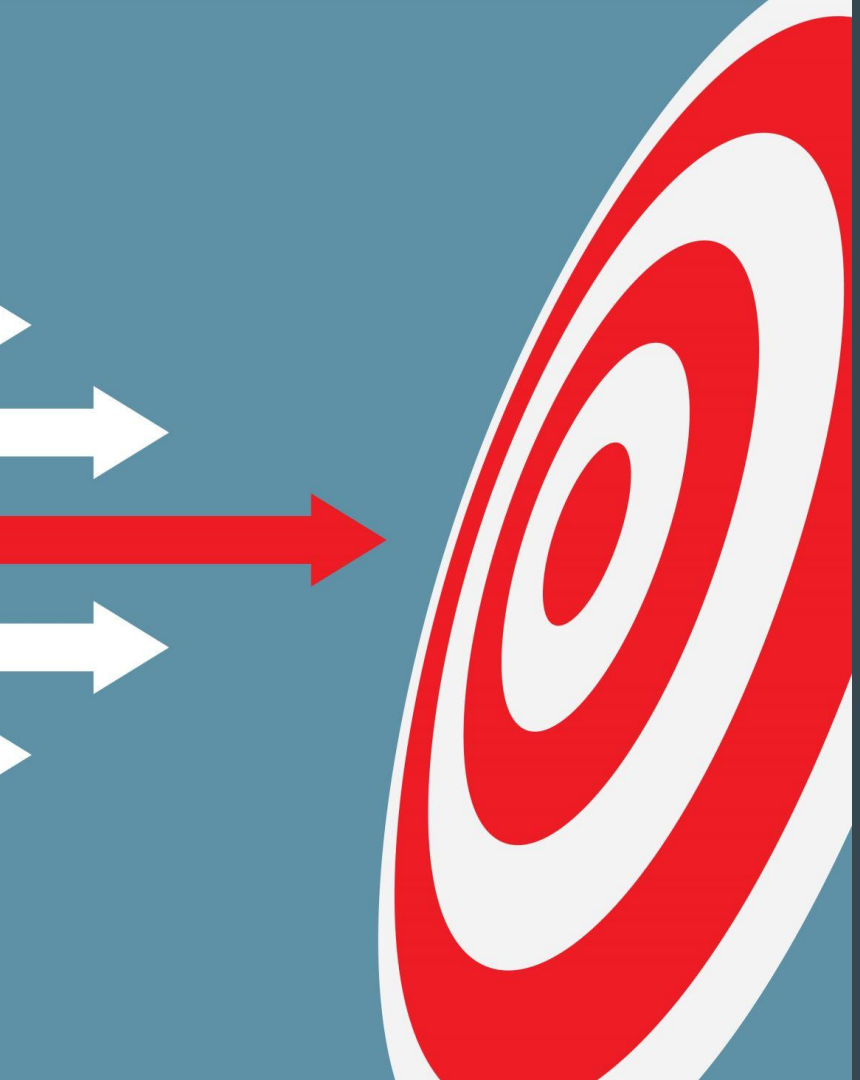
This backdoor registers itself as a system service to ensure its automatic execution at every system startup by adding the following registry entries:

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\

Services\{random service name}Sys

ImagePath = "%System%\Com\svchost.exe"

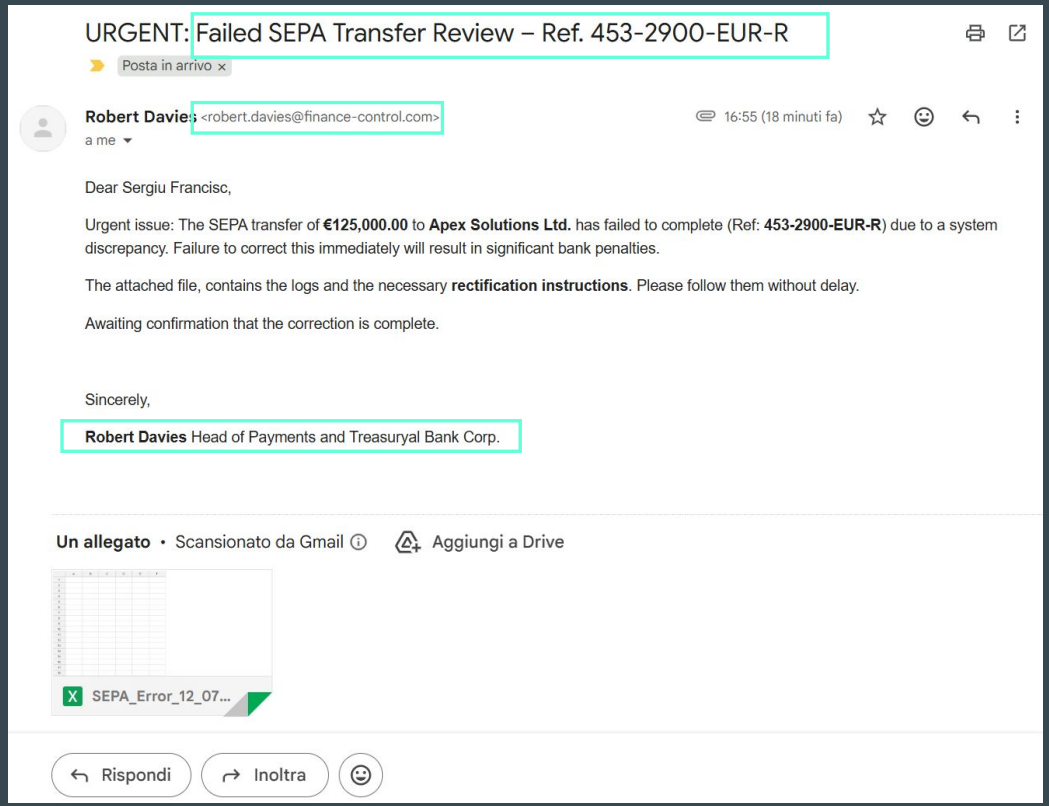
Vulnerabilities involved



The Initial Access Vector: Deconstructing Spear Phishing

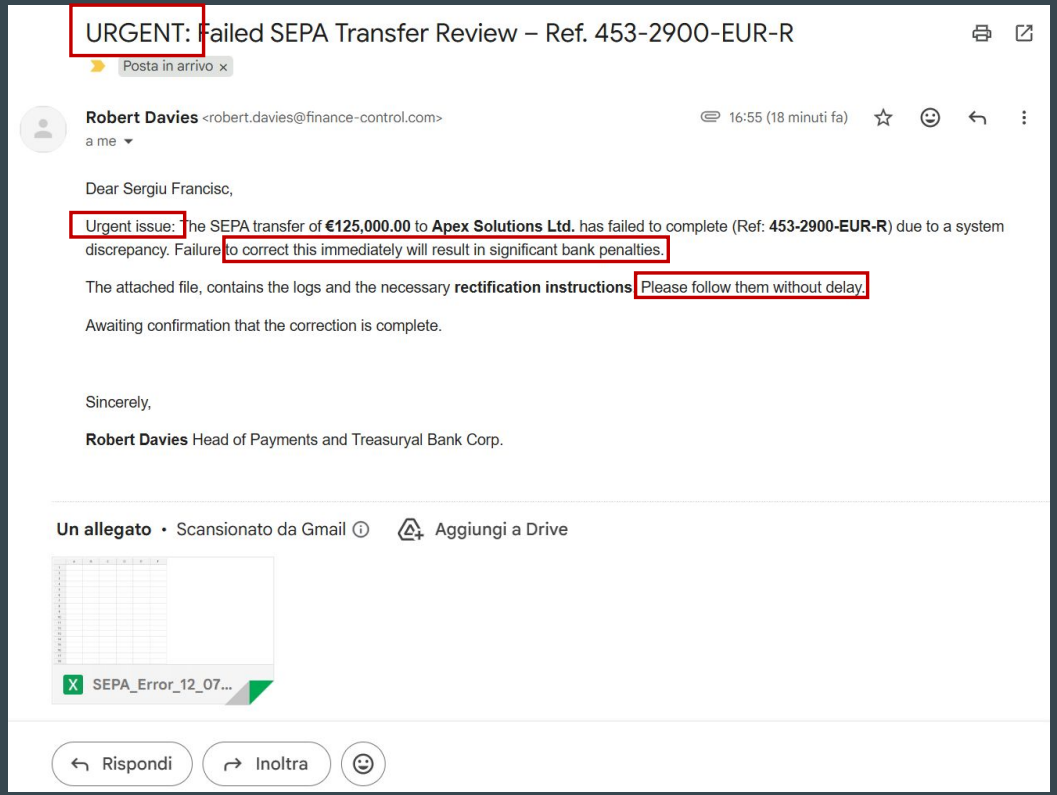
- Perfect Target
- Protocol Engineering
- Payload

Phishing mail example



- Hierarchy and Organizational Trust

Phishing mail example



URGENT: Failed SEPA Transfer Review – Ref. 453-2900-EUR-R

Posta in arrivo ×

Robert Davies <robert.davies@finance-control.com>
a me

16:55 (18 minuti fa)

Dear Sergiu Francisc,

Urgent issue: The SEPA transfer of **€125,000.00** to **Apex Solutions Ltd.** has failed to complete (Ref: **453-2900-EUR-R**) due to a system discrepancy. Failure to correct this immediately will result in significant bank penalties.

The attached file, contains the logs and the necessary **rectification instructions** Please follow them without delay.

Awaiting confirmation that the correction is complete.

Sincerely,

Robert Davies Head of Payments and Treasuryal Bank Corp.

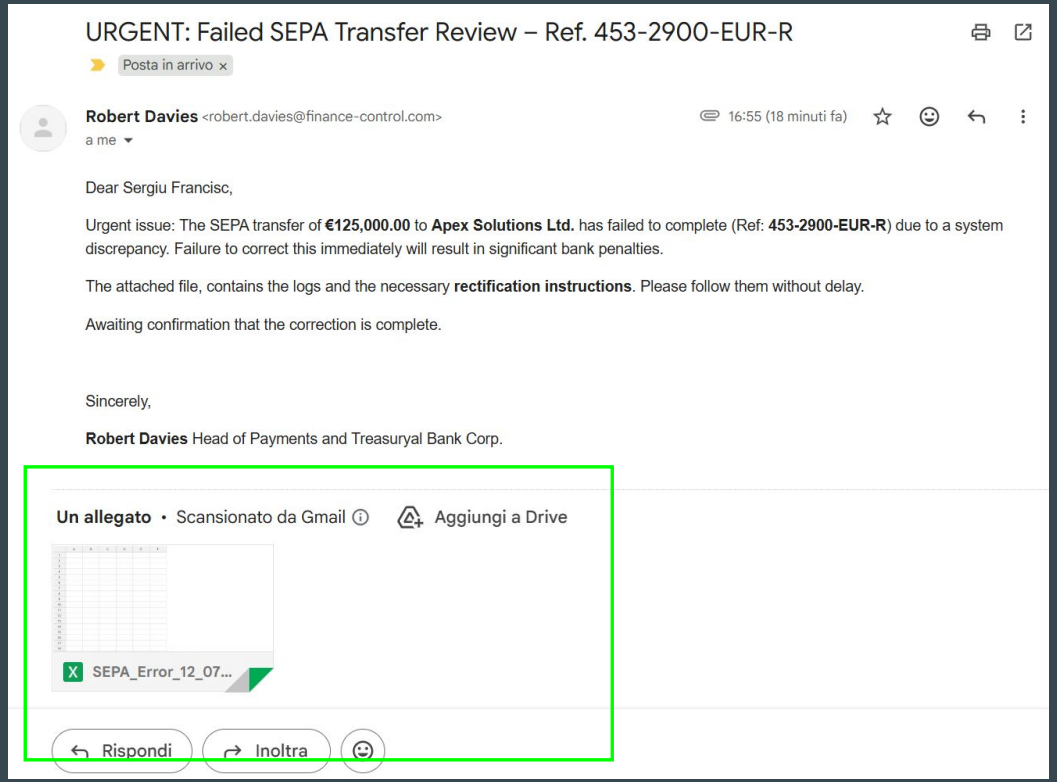
Un allegato • Scansionato da Gmail ⓘ Aggiungi a Drive

SEPA_Error_12_07...

Rispondi Inoltra

- Panic
- Professional Obligation

Phishing mail example

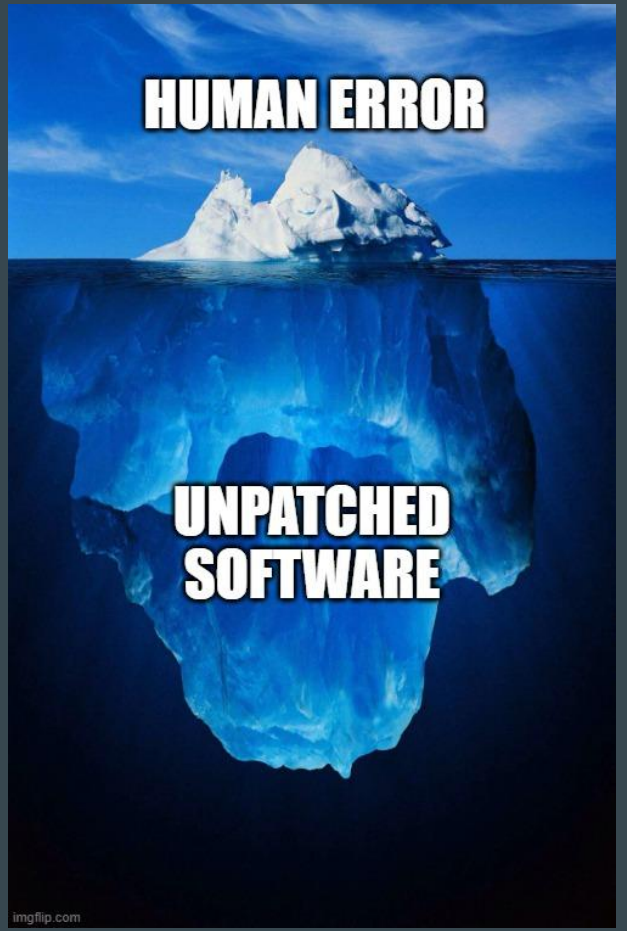


Conclusion:
The primary exploited weakness was Human Vulnerability and the Lack of Enforcement of Defined Protocols.

- The vector: Excel document

Exploited Vulnerabilities in Microsoft Office

- **CVE-2012-0158**
  - RCE via malicious RTF (ActiveX control vulnerability)
- **CVE-2013-3906**
  - RCE triggered by malformed TIFF in Office files
- **CVE-2014-1761**
  - RCE via crafted OLE objects in Word documents



CVE-2012-0158

NAME:
- CVE-2012-0158

DESCRIPTION:
- A remote, unauthenticated attacker could execute arbitrary code, cause a denial of service, or gain unauthorized access to your files or system.

DATE PATCH:
- Patched by Microsoft in April 2012 (Security Bulletin MS12-027).

CVSS SCORE 3.x:
- CVSS: 8.8 (High)

CVE-2013-3906

NAME:
- CVE-2013-3906

DESCRIPTION:
- Microsoft Graphics Component Memory Corruption Vulnerability: Microsoft Graphics Component contains a memory corruption vulnerability which can allow for remote code execution.

DATE PATCH:
- Patched by Microsoft in November 2013 (Security Bulletin MS13-096)

CVSS SCORE 3.x:
- CVSS: 7.8 (High)

CVE-2014-1761

NAME:
- CVE-2014-1761

DESCRIPTION:
- Microsoft Word Memory Corruption Vulnerability: Microsoft Word contains a memory corruption vulnerability which when exploited could allow for remote code execution.

DATE PATCH:
- April 2014 (Security Bulletin MS14-017)

CVSS SCORE 3.x:
- CVSS: 7.8 (Critical)

The Real Issue: Missing System Updates

- **Not the vulnerabilities themselves, but the lack of patching**
- In this case, the exploited vulnerabilities were **not new or unknown**: **Microsoft had already released official patches** months (or even years) before the attack.

We cannot afford to have excellent defenses against future threats (zero-days) if, at the same time, we leave the backdoors open due to errors we could have fixed years ago.

References

1. https://attack.mitre.org/groups/G0008/
2. https://cloud.google.com/blog/topics/threat-intelligence/behind-the-carbanak-backdoor/
3. https://www.group-ib.com/resources/research-hub/anunak-apt/
4. https://www.hwgsababa.com/en/carbanak-the-cyber-theft-of-the-century/
5. https://websec.net/blog/classic-malware-carbanak-639b6860dece7abbe6b48b8a
6. https://websec.net/blog/classic-malware-carbanak-part-2-639e8a8ddece7abbe6b579f2
7. https://github.com/0x25bit/Updated-Carbanak-Source-with-Plugins
8. https://securelist.com/the-great-bank-robbery-the-carbanak-apt/68732/
9. https://rootfu.in/bypassing-amsi-with-dynamic-api-resolution-in-powershell/
10. https://erdalozkaya.com/carbanak-source-code-analysis/
11. https://securitymaven.medium.com/api-hashing-why-malware-loves-and-you-should-care-77c5135d9aaa
12. https://cloud.google.com/blog/topics/threat-intelligence/carbanak-week-part-one-a-rare-occurrence/
13. https://github.com/RamadhanAmizudin/malware/tree/master/Carberp
14. https://www.kiteworks.com/risk-compliance-glossary/living-off-the-land-attacks/
15. https://www.rapid7.com/fundamentals/living-off-the-land-attack/
16. https://www.trendmicro.com/en_us/research/21/d/carbanak-and-fin7-attack-techniques.html
17. https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/bkdr_carbanak.b
18. https://www.europol.europa.eu/media-press/newsroom/news/mastermind-behind-eur-1-billion-cyber-bank-robbery-arrested-in-spain
19. https://www.securityweek.com/carbanak-hackers-use-shims-process-injection-persistence/
20. Randal E. Bryant , David Richard O'Hallaron - Computer Systems: A Programmer's Perspective, Global Edition 3rd Edition
21. de Smidt, G., & Botzen, W. (2018). Perceptions of Corporate Cyber Risks and Insurance Decision-Making. Geneva papers on risk and insurance. Issues and practice, 43(2), 239-274. https://doi.org/10.1057/s41288-018-0082-7
22. https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08064518/Carbanak_APT_eng.pdf
23. https://www.cve.org/CVERecord?id=CVE-2012-0158
24. https://www.cve.org/CVERecord?id=CVE-2013-3906
25. https://www.cve.org/CVERecord?id=CVE-2014-1761