Contents

Some definitions

SPV – Software Protection Vendor (packer, protector).
SSV – Software Security Vendor (antivirus).
TSA – TimeStamp Authority server.
Taggant – the structure that contains a digital signature of a file.
CMS – Cryptographic Message Syntax, hereinafter, the structure that contains data (a file hash, a TSA response) and their digital signature.

Notes:

1) the TSA response must meet Standard RFC 3161;
2) the SHA256 algorithm is used for file hashing.

1. The process of creating taggants for the SPV

1) Initialize the taggant library with the TaggantInitializeLibrary function;
2) Within the process of creating a protected file, the SPV must reserve some space in the file where the taggant will be placed. The size of the reserved space must be equal to constant TAGGANTS_REQUIRED_LENGTH from module taggant_types.h;
3) The SPV must go through the complete procedure of file protection. Please note that after the taggant is created, the SPV should no longer modify the protected file. Exceptions are file modifications upon its digital signature (with parameters IMAGE_DIRECTORY_ENTRY_SECURITY of the directory in the optional header changed) and if HASHMAP hashing is used upon taggant creation;
4) The SPV must place the necessary data to the file enter point according to the manual (relative jump JMP 0x8 and 8-byte pointer to the location of taggants in a physical file);
5) Check user license by calling TaggantGetLicenseExpirationDate and optionally notify user about license expiration date;
6) Create a context for file reading handler functions by calling TaggantContextNew;
7) Create a TAGGANTOBJ helper object using the TaggantObjectNew function;
8) Optionally, add a regions that should be covered by hashmap (function TaggantAddHashRegion);
9) Call TaggantComputeHashes to calculate full file hash and hashmap;
10) Fill out packer information structure with help of TaggantPackerInfo function;
11) Receive a response from the TSA server by calling the TaggantPutTimestamp function (optionally);
12) Create a taggant structure by calling TaggantPrepare. Write the taggants into the protected file;
13) Free the helper object TAGGANTOBJ using the TaggantObjectFree function;
14) Free the context by the TaggantContextFree function;
15) Free the taggant library resources using the TaggantFinalizeLibrary function.

* Note: For the protected file that contains a taggant to be correctly processed by anti-viruses (SSV), the SPV must delete the description (and maybe together with the data) of the IMAGE_DIRECTORY_ENTRY_SECURITY directory from the file.

** When protecting several files, you should go through steps 2-13 several times (maybe, with the exception of steps 6 and 14).

1.1 How to check if license information is valid

Before generating the taggant, SPV may check if the user license (license includes ca, spv and user certificates and user private key) is valid. This can be done by calling of the TaggantGetLicenseExpirationDate function. It also returns expiration date of user certificate. SPV may notify user that the certificate is expired (or will expire soon) and ways how to renew it. Please note, certificates renewal process is optional (but recommended), taggant library will be working well with expired certificates too.

1.2 Why and how to use hash map

Instead of the full file hash, taggant library provides and ability to add a hash map to the taggant. Hash map represents a set of regions of the file that are covered by the taggant. Using of hash map is optional but highly recommended. Hash map allows SSV to do a quick check if the taggant is valid for a file (if the file is not modified).
Please note, hashmap should cover only very important parts of the file, for example, file PE header, loader code, data directories.

1.3 Handling overlays in the files with the taggant

Flexibility of the taggant library allows or enable or disable the ability to append protected file with the unsigned overlay data. SPV should determine this behavior while protection.
If SPV wants to enable ability to append overlay data in the protected file, then while calling of the function TaggantComputeHashes, the parameter uFileEnd must contain a value of the current file size.
To disable this ability, uFileEnd should be zero.

2. The process of checking taggants for SSV

1) Initialize the taggant library with the TaggantInitializeLibrary function;
2) Create a context for file reading handler functions by calling TaggantContextNew;
3) Check if the file has a taggant structure and get it using the TaggantGetTaggant function;
4) Create a TAGGANTOBJ helper object using the TaggantObjectNew function;
5) Check the CMS digital signature in the taggant structure (i.e. check whether the CMS is signed with the certificate derived from the IEEE Root certificate or not) by calling the TaggantValidateSignature function. If the function returns an error, deem the taggant structure incorrect;
6) Optionally, check the TSA response contained in the taggant and get the time of file protection using the TaggantGetTimestamp function. If the function returns an error, deem the taggant structure does not contain timestamp;
7) Optionally, check the packer version with help of TaggantPackerInfo function;
8) Check if the hashmap exists in the taggant (by means of the function TaggantGetHashMapDoubles) and validate it using TaggantValidateHashMap function;
9) Depending on the result of #8, validate a full file hash;
10) Retrieve user and SPV certificates from taggants using the TaggantGetInfo function and check if they are not blacklisted;
11) Free the TAGGANTOBJ helper object using the TaggantObjectFree function;
12) Free the context using the TaggantContextFree function;
13) Free the taggant library resources using the TaggantFinalizeLibrary function.

* To check several files, repeat steps 2-11 several times (optionally, with the exception of steps 2 and 12).
** To check taggant with the several root certificates, repeat steps 5-10 with each certificate for each file.

2.1 How to check if the root certificate is valid?

SSV can check if the root CA certificate using the function TaggantCheckCertificate.

3. Function descriptions

3.1 UNSIGNED32 TaggantInitializeLibrary(TAGGANTFUNCTIONS *pFuncs, UNSIGNED64 *puVersion) [SSV + the SPV libs]

The function initializes the library, cryptoalgorithms, and installs its own handler functions to operate the memory. This function must be called prior to calling any other function from the library. When finishing using the taggant library, the TaggantFinalizeLibrary function must be called.

Parameters:

- **pFuncs** – a pointer to the filled TAGGANTFUNCTIONS structure
- **puVersion** – a pointer to a variable of the UNSIGNED64 type that the taggant library version will be written to

The TAGGANTFUNCTIONS structure contains the following elements:

```
typedef struct
{
        int size;
        // Memory callbacks
        void* (__DECLARATION *MemoryAllocCallBack)(size_t);
        void* (__DECLARATION *MemoryReallocCallBack)(void*, size_t);
        void (__DECLARATION *MemoryFreeCallBack)(void*);
} TAGGANTFUNCTIONS, *PTAGGANTFUNCTIONS;
```

- **size** – the size of the TAGGANTFUNCTIONS structure being passed.
- **MemoryAllocCallBack** – allocates a memory buffer of a specified size. Returns the pointer to the allocated buffer. Parameters:
  - **size_t** – the size of the buffer that must be allocated
- **MemoryReallocCallBack** – reallocate a memory buffer. The function returns the pointer to a newly allocated memory buffer. Parameters:
  - **void*** **-** a memory buffer that must be reallocated
  - **size_t** – the size of the necessary buffer
- **MemoryFreeCallBack** – Free the memory buffer. Parameters:
  - **void*** **-** a memory buffer that must be freed

Please note:

- for x86 applications, handler functions use the stdcall method for argument passing. For applications of other platforms, their standard argument passing method is used.
- any elements of the TAGGANTFUNCTIONS structure (as well as the pointer passed to the function itself) can be null (NULL). This will mean that internal memory operation functions (malloc, realloc, free) will be used.
- If any element of the TAGGANTFUNCTIONS structure is NULL, then the values of other elements are ignored, and internal memory operation functions will be used.

3.2 void TaggantFinalizeLibrary() [SSV + the SPV libs]

The function frees all the helper objects created for operating the library. Must be called after operations with the taggant library are finished. To initialize the library, the TaggantInitializeLibrary function must be called.

3.3 UNSIGNED32 TaggantComputeHashes(PTAGGANTCONTEXT pCtx, PTAGGANTOBJ pTaggantObj, PFILEOBJECT hFile, UNSIGNED64 uObjectEnd, UNSIGNED64 uFileEnd, UNSIGNED32 uTaggantSize) [SPV lib]

The function calculates hash for the entire file (including full file hash and hash map).
The TaggantComputeHashes function must be called prior to calling TaggantPrepare for calculating the hash of the file that is being signed.
What does the hash of the entire file include? The hash of the entire file is divided into two structures: a default hash (PE file hash) and an extended hash (a region hash calculated from the physical end of a PE file to the physical end of the file (this region is called overlay)). A default hash includes hash of the entire file excluding the following regions:

1) Checksum field of the optional header of the PE file
2) Digital Signature header (IMAGE_DIRECTORY_ENTRY_SECURITY directory in the DataDirectory) from the optional header of the PE file (except for the Checksum field itself)
3) The taggant structure

A set regions for hash map should be specified (by function TaggantAddHashRegion) prior to calling TaggantComputeHashes.
Results returned:

- **TNOERR** – no errors, the function is successfully done;
- **TINVALIDPEENTRYPOINT** – the enter point of the PE file is not found;
- **TINVALIDTAGGANTOFFSET** – the taggant structure is incorrect (the code at the enter point of the PE file does not correspond to the manual);
- **TFILEACCESSDENIED** – operations with the file returned an error.

Parameters:

- **pCtx** – the context that contains handler functions for operating files created by the TaggantContextNew function;
- **pTaggantObj** – a pointer to the TAGGANTOBJ object;
- **hFile** – a pointer, a file identifier;
- **uObjectEnd** – a pointer to the end of the PE file in the physical file (a physical size of the PE file). This parameter can be calculated by summing PointerToRawData and SizeOfRawData of the last section of PE file, or by calculating the end of SizeOfImage (from the optional header of the PE file) of the physical file relative to its begin;
- **uFileEnd** – a physical size of the file that the hash should be calculated for. This parameter cannot be less than uObjectEnd. Note for the SSV: if the file contains a digital signature (and if

this signature is at the end of the file), the physical size of the file passed to this function must be reduced by the size of this digital signature. This means that after the taggant has been embedded, the file has also been signed with the digital signature that must not interfere with the calculations in the taggant library;

- uTaggantSize – size of the space reversed for taggant in the file.

3.4 PTAGGANTOBJ TaggantObjectNew(PTAGGANT pTaggant) [SSV + the SPV libs]

The function creates an object that will keep all the helper structures and objects required for working with the taggant (CMS, certificates, taggant, data), signing and validating data in the taggant, retrieving data (user and intermediate certificates, file hashes, time). After finishing using the TAGGANTOBJ object, the TaggantObjectFree function must be called to free the memory used.

Results returned:

- a pointer to the TAGGANTOBJ object.

Parameters:

- pTaggant – a pointer to the TAGGANT structure returned by TaggantGetTaggant. Parameter should be used for SSV only, SPV should set this parameter as NULL.

Function does not have parameters.

3.5 void TaggantObjectFree(PTAGGANTOBJ pTaggantObj) [SSV + the SPV libs]

The function frees the object after using TAGGANTOBJ. To initialize the object, the TaggantObjectNew function must be used.

3.6 PTAGGANTCONTEXT TaggantContextNew() [SSV + the SPV libs]

The function creates a context (structure) that contains handler functions for operating files. Later on, the context will have to be passed to the file operation functions. Since the context is a structure, a vendor, from now on, can edit the parameters of this structure and install their own file operation handlers. This function will be useful if there is a simultaneous access to, for example, a file in a real file system (using context #1) and to a file in a memory buffer (using context #2). When finishing using the context, the TaggantContextFree function must be called.

The result returned:

- a pointer to the TAGGANTCONTEXT structure.

The TAGGANTCONTEXT structure contains the following elements:

```
typedef struct
{
        int size;
        /* File IO callbacks */
        size_t (__DECLARATION *FileReadCallBack)(PFILEOBJECT, void*, size_t);
        int (__DECLARATION *FileSeekCallBack)(PFILEOBJECT, UNSIGNED64, int);
        UNSIGNED64 (__DECLARATION *FileTellCallBack)(PFILEOBJECT);
} TAGGANTCONTEXT, *PTAGGANTCONTEXT;
```

- **size** – the size of the TAGGANTCONTEXT structure. Note: if a vendor modifies default handler functions in this structure, they should take care of a version control. In future, when issuing new versions of the taggant library, the contents of this structure and the number of parameters can change. The vendor, based on this value of the structure size – size - should fill the correct number of parameters;
- **FileReadCallBack** – a file read function. The function returns the count of bytes actually read. Parameters:
  - **PFILEOBJECT** – a pointer, a file identifier;
  - **void*** - a pointer to the buffer that the read block from the file will be placed into;
  - **size_t** – the size of the buffer that must be read.
- **FileSeekCallBack** – a function of an internal read pointer offset. The function returns 0 if there are no errors, or any other value if there is any error. Parameters:
  - **PFILEOBJECT** – a pointer, a file identifier;
  - **long** – the number of bytes by which the internal file pointer should be set off;
  - **int** – a setoff type. Can have the following values:
    - SEEK_SET (0), set the pointer relative to the file begin;
    - SEEK_CUR (1) , set the pointer relative to its current position;
    - SEEK_END (2), set the pointer relative to the begin of the file end.
- **FileTellCallBack** – returns the current value of the internal file pointer. Parameters:
  - **PFILEOBJECT** – a pointer, a file identifier.

Please note:

- for x86 applications, handler functions use the stdcall method for argument passing. For applications of other platforms, their standard argument passing method is used;
- any elements of the TAGGANTFUNCTIONS structure can be null (NULL). This will mean that internal file operation functions (fread, ftell, fseek) will be used. In this case, file operation functions as a file identifier (of the PTAGFILEOBJECT type parameter) must receive a pointer to the FILE structure that can be, for example, received upon opening a file with a standard function - fopen.

3.7 void TaggantContextFree(PTAGGANTCONTEXT pTaggantCtx) [SSV + the SPV libs]

Frees the context created using the TaggantContextNew function.

Parameters:

- **pTaggantCtx** – a pointer to the TAGGANTCONTEXT structure.

3.8 UNSIGNED32 TaggantGetTaggant(PTAGGANTCONTEXT pCtx, PFILEOBJECT hFile, TAGGANTCONTAINER eContainer, PTAGGANT *pTaggant) [SSV lib]

The function retrieves the taggant structure from the file. After the taggant structure is retrieved, a TAGGANTOBJ object should be created using the TaggantObjectInitialise function, and the taggant should be validated using the TaggantValidate function.

Parameters:

- **hFile** - a pointer, a file identifier;
- **eContainer** – the type of the file that the taggant is retrieved for, the values supported:
  - TAGGANT_PEFILE – the input file – a Windows PE file;
- **pTaggant** – returns a pointer to the buffer with the taggant structure, it is used. This buffer should be freed by TaggantFreeTaggant function.

Results returned:

- **TNOERR** – no errors;
- **TNOTAGGANTS** – the file does not contain a taggant;
- **TTYPE** – the type of the file passed by the TAGGANTCONTAINER parameter is not supported.

3.9 UNSIGNED32 TaggantValidateSignature(PTAGGANTOBJ pTaggantObj, PTAGGANT pTaggant , PVOID pRootCert) [SSV lib]

The function checks whether the pTaggant structure performs in accordance with the manual (for example, whether the taggant begin and end markers are validated), and whether the CMS is signed in the taggant with the correct digital certificates (the certificates derived from the IEEE Root certificate).

Note: if this function returns a TNOERR result, this does not mean that the file has not been modified, or that the taggant belongs to this file, or that the taggant contains a correct timestamp. After the taggant is validated using the TaggantValidateSignature function, depending on the type of hashes in the taggant structure, the hashes of the file itself are calculated and compared. Further, the file signature date is validated using the TaggantGetTimestamp function. Only if all the validations have returned a positive result, can we conclude that the file contains the correct taggant, i.e. the file has not been modified.

Parameters:

- **pTaggantObj** – a pointer to the TAGGANTOBJ object;
- **pTaggant** – a pointer to the buffer that contains the taggant structure. The taggant structure must be retrieved using the TaggantGetTaggant function.
- **pRootCert** – a pointer to the buffer where the IEEE root certificate is located in the PEM (base64) format.

Results returned:

- **TNOERR** – no errors;
- **TBADKEY** – CMS validation has failed. Perhaps, this is caused by the CMS signed with the certificate derived not from the IEEE Root certificate, or the structure is modified;

- **TLIBNOTINIT** – the library taggant has not been initialized using the TaggantLibraryInitialize function.

3.10 UNSIGNED32 TaggantGetInfo(PTAGGANTOBJ pTaggantObj, ENUMTAGINFO eKey, UNSIGNED32 *pSize, PINFO pInfo) [SSV lib]

Function retrieves various data from the taggant structure (file hashes, SPV and user certificates).

Parameters:

- **pTaggantObj** – a pointer to the TAGGANTOBJ object;
- **eKey** – defines the type of information that should be retrieved from the taggant. Can have the following values:
    - *ETAGGANTBLOB* = 0 – for retrieving taggant blob structure;
    - *ESPVCERT* = 1 – for retrieving an intermediate SPV certificate;
    - *EUSERCERT* = 2 – for retrieving a user certificate;
    - *EFILEEND* = 3 – for retrieving a file end value of Extended file hash;
- **pSize** – this parameter should have the size of the buffer passed by the pInfo parameter, or, if the size of this buffer is insufficient, the function will return the required buffer size in it;
- **pInfo** – a pointer to the buffer where the retrieved information should be placed to. This parameter can be NULL, in this case, the pSize parameter shall contain the size of the required buffer.

Results returned:

- **TNOERR** – no errors;
- **TMEMORY** – the size of the pInfo buffer is not enough for storing the retrieved information;
- **TNOTAGGANTS** – the taggant structure has not been validated by the TaggantValidate function, or the taggant structure was modified or is not valid;
- **TLIBNOTINIT** – the taggant library has not been initialized by the TaggantLibraryInitialize function.

3.11 UNSIGNED32 TaggantPrepare(PTAGGANTOBJ pTaggantObj, PVOID pLicense, PVOID pTaggantOut, UNSIGNED32 *uTaggantReservedSize) [SPV lib]

The function signs data (file hashes, a timestamp response) with a user certificate and creates a taggant structure that should be placed into the file according to the manual. To put timestamp into taggant, the function TaggantPutTimestamp should be called before TaggantPrepare.

Parameters:

- **pTaggantObj** – a pointer to the TAGGANTOBJ object;
- **pLicense** – a pointer to the buffer where user license information is located. The license information should include an intermediate SPV certificate, a user certificate, a user private key located in the same order from the begin of the file and encrypted in the PEM format (base64);
- **pTaggantOut** – a pointer to the buffer where the taggant structure will be placed to;

- **uTaggantReservedSize** – size of pTaggantOut buffer, a space reserved in the file for taggant. If the size is insufficient, this parameter returns approximate necessary size, SPV should reserve enough buffer in protected file and repeat TaggantPrepare.
- The size of this buffer must comply with the value of the TAGGANTS_REQUIRED_LENGTH constant (see the source code).

Results returned:

- **TNOERR** – no errors;
- **TBADKEY** – user license information is incorrect;
- **TLIBNOTINIT** – the taggant library has not been initialized by the TaggantLibraryInitialize function;
- **TINSUFFICIENTBUFFER** – buffer for taggant is insufficient.

3.12 UNSIGNED32 TaggantAddHashRegion(PTAGGANTOBJ pTaggantObj, UNSIGNED64 uOffset, UNSIGNED64 uLength) [SPV lib]

The function adds a region for hashing into the HASHBLOB_HASHMAP structure. Note: the size of a buffer with the pHash pointer that contains the HASHBLOB_HASHMAP structure must be sufficient for adding one more region.

Parameters:

- **pTaggantObj** – a pointer to the TAGGANTOBJ object;
- **uOffset** – the begin of a region in a physical file relative to the begin of the file;
- **uLength** – the length of the region added;

Results returned:

- **TNOERR** – no errors;
- **TENTRIESEXCEED** – number of regions exceeded the maximum allowed;
- **TLIBNOTINIT** – the taggant library has not been initialized by the TaggantLibraryInitialize function.

3.13 UNSIGNED32 TaggantGetTimestamp(PTAGGANTOBJ pTaggantObj, UNSIGNED64 *pTime, PVOID pTSRootCert) [SSV lib]

The function retrieves from the taggant structure the time of signing the file. This function must be called only after the TaggantValidate function has been successfully called, i.e. when the CMS contained in the taggant structure has been validated and found to be correct. Note: if this function returns an error, this means that the file contains an incorrect taggant structure.

Parameters:

- **pTaggantObj** – a pointer to the TAGGANTOBJ object;
- **pTime** – the time when this file was signed in the UNIX format (in seconds relative to the year of 1970). Please note that the type of this parameter is UNSIGNED64, i.e. it contains 8 bytes of information. This is to prevent the issue of the year of 2038 with the standard UNIX time.

- **pTSRootCert** – a pointer to the buffer where the IEEE root certificate (used for timestamp) is located in the PEM (base64) format.

Results returned:

- **TNOERR** – no errors;
- **TLIBNOTINIT** – the taggant library has not been initialized by the TaggantLibraryInitialize function;
- **TNOTIME** – the taggant does not contain the time of the file's signature;
- **TINVALID** – the response from the Timestamp Authority server is incorrect;
- **TERROR** - TAGGANTOBJ does not contain the TAGGANTBLOB structure, i.e. the file's hashes calculated using the TaggantComputeHashes function.

3.14 UNSIGNED32 TaggantPutTimestamp (PTAGGANTOBJ pTaggantObj, const char* pTSUrl, UNSIGNED32 uTimeout) [SPV lib]

The function adds time to the CMS that defines when the file was protected. Note: when this function is called, the library connects to the internet to get a response from the Timestamp Authority server. The SPV must warn the user that the file can only be signed (taggants can be created) with an active connection to the internet that is not blocked by the firewall.

TaggantPutTimestamp must be called prior to calling TaggantPrepare, i.e. before creating the taggant structure.

Parameters:

- **pTaggantObj** – a pointer to the TAGGANTOBJ object;
- **pTSUrl** – the URL of the Timestamp Authority server. This string must contain null-termination (at the end) that defines the end of the string;
- **uTimeout** – the maximum time for which the function will be waiting for the response from the server.

Results returned:

- **TNOERR** – no errors;
- **TLIBNOTINIT** – the taggant library has not been initialized by the TaggantLibraryInitialize function;
- **TNONET** – no connection to the internet;
- **TTIMEOUT** – the timestamp authority server response time has expired.

3.15 UNSIGNED32 TaggantGetLicenseExpirationDate(PVOID pLicense, UNSIGNED64 *pTime) [SPV lib]

The function checks if the user license (license includes spv and user certificates and user private key) is valid (license structure and content are valid) and return expiration date of user certificate.

Parameters:

- **pLicense** – a pointer to the buffer where user license information is located. The license information should include an intermediate CA and SPV certificates, a user certificate, a user private key located in the same order from the begin of the file and encrypted in the PEM format (base64);
- **pTime** – the time when this file was signed in the UNIX format (in seconds relative to the year of 1970). Please note that the type of this parameter is UNSIGNED64, i.e. it contains 8 bytes of information. This is to prevent the issue of the year of 2038 with the standard UNIX time.

Results returned:

- **TNOERR** – no errors;
- **TBADKEY** – user license information is incorrect;
- **TLIBNOTINIT** – the taggant library has not been initialized by the TaggantLibraryInitialize function.

3.16 UNSIGNED32 TaggantCheckCertificate(PVOID pCert) [SSV lib]

The function checks if the certificate located in pCert buffer (in base64 encoded format) is valid.

Parameters:

- **pCert** – a pointer to the buffer where the certificate is located in the PEM (base64) format.

Results returned:

- **TNOERR** – no errors;
- **TINVALID** – the certificate is not valid;
- **TLIBNOTINIT** – the taggant library has not been initialized by the TaggantLibraryInitialize function.

3.17 UNSIGNED32 TaggantValidateDefaultHashes(PTAGGANTCONTEXT pCtx, PTAGGANTOBJ pTaggantObj, PFILEOBJECT hFile, UNSIGNED64 uObjectEnd, UNSIGNED64 uFileEnd) [SSV lib]

The function validates default hash (hash for the entire file, regions computed using TaggantComputeDefaultHashes). Function should be called by SSV to check if the hashes included in the taggant matches real file hashes. This function must be called after validating of taggant/CMS by function TaggantValidateSignature.
Depending on a hash type defined in the taggant (extract it using TaggantGetHashType) functions TaggantValidateDefaultHashes or TaggantValidateHashMap should be called.

Results returned:

- **TNOERR** – no errors, the function is successfully done;
- **TMISMATCH** – hashes do not match;
- **TLIBNOTINIT** – the taggant library has not been initialized by the TaggantLibraryInitialize function;
- **TFILEACCESSDENIED** – operations with the file returned an error.

Parameters:

- **pCtx** – the context that contains handler functions for operating files created by the TaggantContextNew function;
- **pTaggantObj** – a pointer to the TAGGANTOBJ object;
- **uObjectEnd** – a pointer to the end of the PE file in the physical file (a physical size of the PE file). This parameter can be calculated by summing PointerToRawData and SizeOfRawData of the last section of PE file, or by calculating the end of SizeOfImage (from the optional header of the PE file) of the physical file relative to its begin;
- **uFileEnd** – a physical size of the file that the hash should be calculated for. This parameter cannot be less than uObjectEnd. Note for the SSV: if the file contains a digital signature (and if this signature is at the end of the file), the physical size of the file passed to this function must be reduced by the size of this digital signature. This means that after the taggant has been embedded, the file has also been signed with the digital signature that must not interfere with the calculations in the taggant library.

3.18 UNSIGNED32 TaggantValidateHashMap(PTAGGANTCONTEXT pCtx, PTAGGANTOBJ pTaggantObj, PFILEOBJECT hFile) [SPV lib]

The function validates a hash for a region digit which is a hash map. Function should be called by SSV to check if the hashes included in the taggant matches real file hashes. This function must be called after validating of taggant/CMS by function TaggantValidateSignature.
Depending on a hash type defined in the taggant (extract it using TaggantGetHashType) functions TaggantValidateDefaultHashes or TaggantValidateHashMap should be called.

Results returned:

- **TNOERR** – no errors, the function is successfully done;
- **TMISMATCH** – hashes do not match;
- **TLIBNOTINIT** – the taggant library has not been initialized by the TaggantLibraryInitialize function;
- **TFILEACCESSDENIED** – operations with the file returned an error.

Parameters:

- **pCtx** – the context that contains handler functions for operating files created using the TaggantContextNew function;
- **pTaggantObj** – a pointer to the TAGGANTOBJ object;
- **hFile** – a pointer, a file identifier.

3.19 TAGGANTHASHTYPE TaggantGetHashMapDoubles (PTAGGANTOBJ pTaggantObj, PHASHBLOB_HASHMAP_DOUBLE *pDoubles) [SSV lib]

The function extracts information about hash regions included into taggant, if hash map is used. Call this function after validating of the taggant by TaggantValidateSignature or after validating of file hashes by TaggantValidateHashMap.

Parameters:

- **pTaggantObj** – a pointer to the TAGGANTOBJ object;
- **pDoubles** – a pointer to the PHASHBLOB_HASHMAP_DOUBLE array of hash regions.

Function returns number of regions in hashmap.

3.20 PPACKERINFO TaggantPackerInfo(PTAGGANTOBJ pTaggantObj) [SSV + the SPV libs]

The function returns a pointer to the PACKERINFO structure. SPV should get this pointer to fill out packer information, before the taggant is being generated. SSV may get it to read packer information, after the taggant is verified.

Parameters:

- **pTaggantObj** – a pointer to the TAGGANTOBJ object;

Function returns a pointer to PACKERINFO structure.

3.21 void TaggantFreeTaggant(PTAGGANT pTaggant) [SSV lib]

The function frees the TAGGANT object returned by TaggantGetTaggant function.

Parameters:

- **pTaggant** – a pointer to the TAGGANT object. This buffer should be freed by TaggantFreeTaggant function.

4. Types descriptions

Type	Description
#define PFILEOBJECT void*	A pointer that passes the file (for example, a pointer to a structure, a pointer to a class, file handle) to use callback functions to operate files.
#define UNSIGNED8 unsigned char	A one-byte unsigned type.
#define UNSIGNED16 unsigned short	A two-byte unsigned type.
#define SIGNED32 int	A four-byte signed type.
#define UNSIGNED32 unsigned long	A four-byte unsigned type.
#define UNSIGNED64 unsigned long long	An eight-byte unsigned type.
#define TAGGANTS_REQUIRED_LENGTH 0x2000	The size of the buffer that the SPV must reserve in the file. The buffer that the taggant structure will be written to in future.
#define HASH_SHA256_DIGEST_SIZE 32	The size of the structure in bytes, sufficient for storing a SHA256 hash.

typedef enum
{
    TAGGANT_HASBLOB_DEFAULT = 0,
    TAGGANT_HASBLOB_EXTENDED = 1,
    TAGGANT_HASBLOB_HASHMAP = 2
} TAGGANTHASHBLOBTYPE;
	• TAGGANT_HASBLOB_DEFAULT – a hash type - HASHBLOB_DEFAULT
	• TAGGANT_HASBLOB_EXTENDED – a hash type - HASHBLOB_EXTENDED
	• TAGGANT_HASBLOB_HASHMAP – a hash type - HASHBLOB_HASHMAP

typedef struct
{
    UNSIGNED32 PackerId;
    UNSIGNED16 VersionMajor;
    UNSIGNED16 VersionMinor;
    UNSIGNED16 VersionBuild;
    UNSIGNED16 Reserved;
} PACKERINFO, *PPACKERINFO;
	The structure that defines a packer information.

typedef struct
{
    UNSIGNED16 Length;
} EXTRABLOB, *PEXTRABLOB;
	A backup structure. Should not be filled in this specification version.

typedef struct
{
    UNSIGNED16 Length;
    UNSIGNED16 Type;
    UNSIGNED16 Version;
    unsigned char Hash[HASH_SHA256_DIGEST_SIZE];
} HASHBLOB_HEADER, *PHASHBLOB_HEADER;
	The begin of the HASHBLOB structure.
	• Length – the size of the entire HASHBLOB structure;
	• Type – the type of the hash contained in the HASHBLOB structure. This variable can only contain values of the TAGGANTHASHBLOBTYPE type;
	• Version – the version of the HASHBLOB structure, accepts the HASHBLOB_VERSION

	value; • Hash[HASH_SHA256_DIGEST_SIZE] – a hash digest.

typedef struct
{
HASHBLOB_HEADER Header;
} HASHBLOB_DEFAULT, *PHASHBLOB_DEFAULT;
	The structure defines a default hash.

typedef struct
{
HASHBLOB_HEADER Header;
UNSIGNED64 PhysicalEnd;
} HASHBLOB_EXTENDED, *PHASHBLOB_EXTENDED;
	The structure defines an extended hash.
	• PhysicalEnd – an offset relative to the begin of a physical file that defines the end of a physical file.

typedef struct
{
UNSIGNED64 AbsoluteOffset;
UNSIGNED64 Length;
} HASHBLOB_HASHMAP_DOUBLE, *PHASHBLOB_HASHMAP_DOUBLE;
	The structure defines the file region for the region array in the HASHBLOB_HASHMAP structure.
	• AbsoluteOffset – an offset relative to the begin of a physical file that defines the begin of a region;
	• Length – a region length

typedef struct
{
HASHBLOB_HEADER Header;
UNSIGNED16 Entries;
UNSIGNED16 DoublesOffset;
} HASHBLOB_HASHMAP, *PHASHBLOB_HASHMAP;
	The structure defines the type of file hashing - TAGGANT_HMHASH – as a file hash map.
	• Entries – a region count in Doubles
	• DoublesOffset – relative offset of doubles array from beginning of taggant blob structure

typedef struct
{
HASHBLOB_DEFAULT DeafultHash;
HASHBLOB_EXTENDED ExtendedHash;
} HASHBLOB_FULLFILE, *PHASHBLOB_FULLFILE;
	The structure defines the type of file hashing – TAGGANT_FFHASH – as a full file hash.

typedef struct
{
HASHBLOB_FULLFILE FullFile;
HASHBLOB_HASHMAP Hashmap;
} HASHBLOB, *PHASHBLOB;
	The structure defines the type of hashing and file hashes.

typedef struct
{
UNSIGNED16 Length;
UNSIGNED16 Version;
PACKERINFO PackerInfo;
} TAGGANTBLOB_HEADER,
	The TAGGANTBLOB structure header.
	• Length – the TAGGANTBLOB structure length;
	• Version – the version of the TAGGANTBLOB structure, has a TAGGANTBLOB_VERSION value;

*PTAGGANTBLOB_HEADER;	• PackerInfo – a packer identifier structure.

typedef struct
{
    TAGGANTBLOB_HEADER Header;
    HASHBLOB Hash;
    EXTRABLOB Extrablob;
} TAGGANTBLOB, *PTAGGANTBLOB;
	The structure that contains file descriptions and hashes. This structure shall be signed with a user certificate to get a CMS.

typedef struct
{
    UNSIGNED32 MarkerBegin;
    UNSIGNED32 TaggantLength;
    UNSIGNED32 CMSLength;
    UNSIGNED16 Version;
} TAGGANT_HEADER, *PTAGGANT_HEADER;
	The taggant structure header.
	• MarkerBegin – a marker of the TAGGANT_MARKER_BEGIN structure begin;
	• TaggantLength – a physical size of the overall taggant structure;
	• TaggantLength – a physical size of the taggant CMS;
	• Version – a version of the taggant structure, takes a TAGGANT_VERSION value.

typedef struct
{
    EXTRABLOB Extrablob;
    UNSIGNED32 MarkerEnd;
} TAGGANT_FOOTER, *PTAGGANT_FOOTER;
	The structure defines the end of the taggant structure.
	MarkerEnd – a marker of the taggant structure end.

typedef struct
{
    TAGGANT_HEADER Header;
    PVOID CMSBuffer;
    TAGGANT_FOOTER Footer;
} TAGGANT, *PTAGGANT;
	The taggant structure itself.
	CMSBuffer – a buffer containing the signed BLOB taggant structure and a TSA response.

5. Building of taggant library

Before compilation of taggant library the openssl should be installed. In the current project the openssl version 1.0.0g had been used, it can be downloaded from http://www.openssl.org/source/. Compilation manual for openssl for different platforms located inside the archive in the INSTALL.* files (for example, the file INSTALL contains installation steps for UNIX, INSTALL.W32 – installation for Windows 32, INSTALL.MacOS – installation for MacOS etc).

Openssl should be installed in the following folder (if you would like to use automatic building make files for compilation of taggant library):

- c:\openssl – for Windows and contain
  - c:\openssl\include – include files;
  - c:\openssl\lib – lib files;
- /usr/local/ssl – for Unix and contain
  - /usr/local/ssl/include – include files;
  - /usr/local/ssl/lib – lib files;
- msys/local/ssl – for MinGW for Windows and contain
  - msys/local/ssl/include – include files;
  - msys/local/ssl/lib – lib files;
- /opt/local – for Mac OS and contain
  - /opt/local/include – include files
  - /opt/local/lib – lib files.

There are following kinds of taggant library you can compile:

1) SPV_Shared – shared library for SPV;
2) SPV_Static – static library for SPV;
3) SSV_Shared – shared library for SSV;
4) SSV_Static – static library for SSV.

Independent on a platform to which the taggant library is compiling, the directive TAGGANT_LIBRARY should be defined. Depending on a library type, directives SPV_LIBRARY (for SPV library) or SSV_LIBRARY (for SSV library) should be defined too.

5.1 Compilation of taggant library for Win32 using msbuid

Unpack the file Taggant.Win32.zip.

Go to the “Taggant” folder where the file “Taggant.vcproj” is located. If you have Microsoft Visual Studio 2005 or newer installed, you may open Taggant.vcproj project and select necessary configuration in IDE.

To compile shared library for SPV run the following command:

```
msbuild Taggant.vcproj /p:Configuration=SPV_Shared
```

The folder “SPV_Shared” with the library “libspv.dll” should appear after successful compilation.

To compile static library for SPV run the following command:

msbuild Taggant.vcproj /p:Configuration=SPV_Static

The folder “SPV_Static” with the library “libspv.lib” should appear after successful compilation.

To compile shared library for SSV run the following command:

msbuild Taggant.vcproj /p:Configuration=SSV_Shared

The folder “SSV_Shared” with the library “libssv.dll” should appear after successful compilation.

To compile static library for SSV run the following command:

msbuild Taggant.vcproj /p:Configuration=SSV_Static

The folder “SSV_Static” with the library “libssv.lib” should appear after successful compilation.

To recompile the library use the command like:

MSBuild Taggant.vcproj /t:Rebuild /p:Configuration=SPV_Shared

To clean compilation folders use the command:

MSBuild Taggant.vcproj /t:Clean /p:Configuration=SPV_Shared

5.2 Compilation of taggant library for Win32 using nmake

Unpack the file “Taggant.Win32.zip”.

Go to the “Taggant” where the file “Taggant.mak” is located.

To compile shared library for SPV run the following command:

nmake /f Taggant.mak CFG="SPV_Shared"

The folder “SPV_Shared” with the file “libspv.dll” should appear after successful compilation.

To compile static library for SPV run the following command:

nmake /f Taggant.mak CFG="SPV_Static"

The folder “SPV_Static” with the file “libspv.lib” should appear after successful compuilaton.

To compile shared library for SSV run the command:

nmake /f Taggant.mak CFG="SSV_Shared"

The folder “SSV_Shared” with the file “libssv.dll” should appear after successful compilation.

To compile static library for SSV run the command:

nmake /f Taggant.mak CFG="SSV_Static"

The folder “SSV_Static” with the fike “libssv.lib” should appear after successful compuilation.

5.3 Compilation of taggant library for Win32 using MinGW by “make”

Unpack the file “Taggant.Win32.MinGW.zip” to the home (or any other) folder of MinGW.

Using Linux command line emulator MSYS, go to the folder “Taggant”. Depending on a kind of taggant library you would like to compile, enter to the necessary folder: SPV_Shared, SPV_Static, SSV_Shared, SSV_Static.

Run “make” command to compile the project. Necessary library should appear after successful compilation.

5.4 Compilation of taggant library for Linux using “make”

Unpack the file “Taggant.Linux32.zip”.

Using command line, go to “Taggant” folder. Depending on a kind of taggant library you would like to compile, enter to the necessary folder: SPV_Shared, SPV_Static, SSV_Shared, SSV_Static.

Run the “make” command, after successful compilation the necessary library should appear (libspv.so or libspv.a or libssv.so or livssv.a).

5.5 Compilation of taggant library for Mac OS using make

Please note, due to openssl limitation, it does not allow compilation under Mac OS X. If you have Mac OS X installed, then you can update openssl using ports. Check the currently installed openssl version on MacOS system using the command:

```
$ openssl version
```

If you have the openssl version lower than 1.0.0g, it is recommended to update it. To update openssl using ports, do the following:

- Download necessary .dmg file from http://www.macports.org/install.php. Run installer;
- In the terminal run the command:

```
$ sudo port sync; sudo port selfupdate; sudo port install openssl
```

Unpack the file “Taggant.MacOS32.zip”.

Using terminal, go to the “Taggant” folder. Depending on a kind of taggant library you would like to compile, enter to the necessary folder: SPV_Shared, SPV_Static, SSV_Shared, SSV_Static.

Run the “make” command, after successful compilation, the necessary library should appear (libspv.dylib or libspv.a or libssv.dylib or livssv.a).

6 UPX – test packer

6.1 Using

This version of UPX (that allows to add taggant to executable files) has the same command line options as a usual version, but it requires to specify a license file which contains all necessary certificates to create a taggant.

License file should be passed as a command line parameter, after the –u switch, for example:

```
upx.exe -u license.pem myfile.exe
```

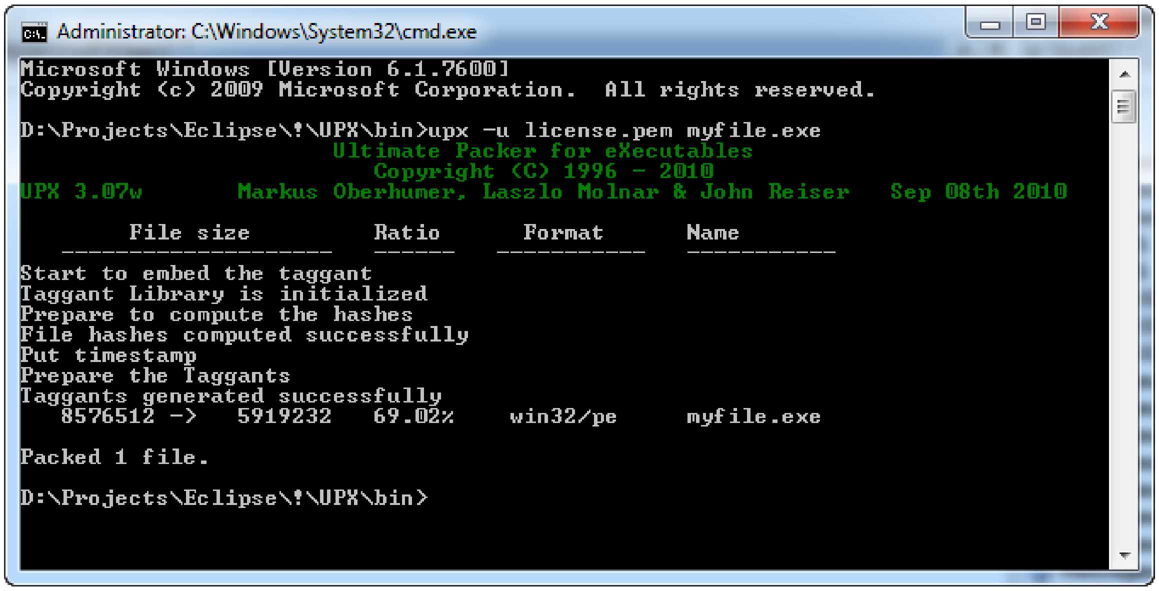
there:

- license.pem – license file;
- myfile.exe – file to be packed;
- optionally, the additional parameter --taggant-hashmap can be specified to compute file hash using hashmap, instead of full file hash which is enabled by default;
- optionally, parameter --taggantsize=8192 allows to set the size of the taggant structure in bytes (by default it is a taggant minimum required size = 0x2000 bytes, maximum – taggant maximum size = 0xFFFF bytes).

See example of the usage on the screenshot below.

Packing log is output to console. If there will appear any error, it will be shown in console and red highlighted.

Please note, taggant library requires active internet connection, otherwise the file will not be packed. Make sure the internet connection is alive and it is not blocked by firewall.



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

D:\Projects\Eclipse\!\UPX\bin>upx -u license.pem myfile.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2010
UPX 3.07w       Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 08th 2010

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
Start to embed the taggant
Taggant Library is initialized
Prepare to compute the hashes
File hashes computed successfully
Put timestamp
Prepare the Taggants
Taggants generated successfully
   8576512 ->   5919232   69.02%    win32/pe     myfile.exe

Packed 1 file.

D:\Projects\Eclipse\!\UPX\bin>
```

6.2 Compilation

For compilation we will use MinGW compiler with help of Linux command line emulation system – MSYS.

To compile UPX we have to install UCL compression library firstly.

Installation of UCL:

1) run command line emulator MSYS;
2) using “cd” command go to UCL folder;
3) run the command “make clean” to delete previously compiled files;
4) run configuration command “./configure”;
5) run “make” command to compile UCL library;
6) install UCL by running the command “make install”.

Compilation of UPX:

1) run command line emulator MSYS;
2) using “cd” command go to UPX folder;
3) run the command “make clean” to delete previously compiled files;
4) run “make” command to start compilation.

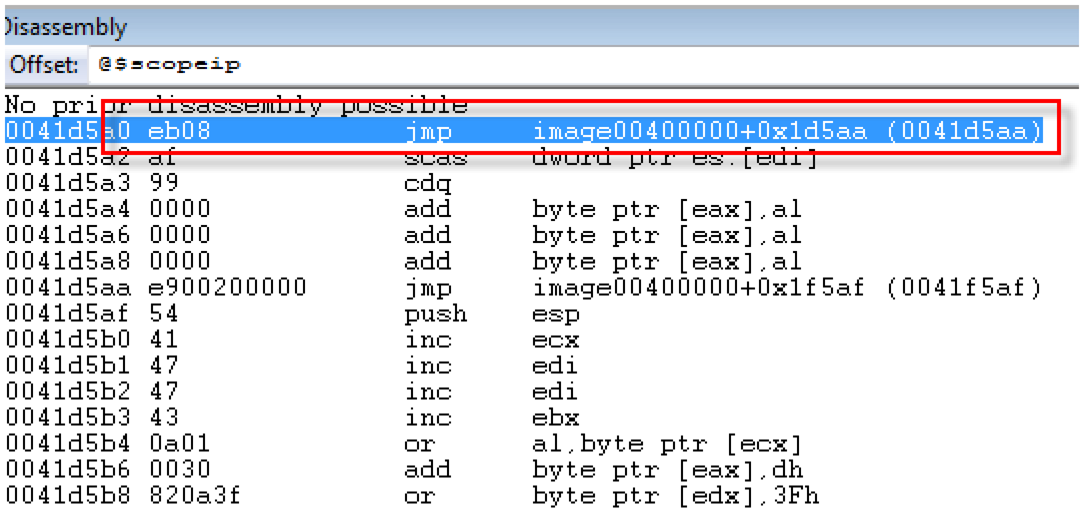
After compilation process finished, the file upx.exe should be created in the UPX folder.

Note, taggant library libspv.dll should exist in the UPX folder while compilation.

6.3 Specification

Where the taggant structure is placed in the PE file?

According to the documentation of library taggant, the PE file entry point must start from short 8-bytes relative jump over the location of the pointer to the taggant structure in the physical file (see screenshot below).



```
No prior disassembly possible
0041d5a0 eb08          jmp     image00400000+0x1d5aa (0041d5aa)
0041d5a2 af            scas    dword ptr es:[edi]
0041d5a3 99            cdq
0041d5a4 0000          add     byte ptr [eax],al
0041d5a6 0000          add     byte ptr [eax],al
0041d5a8 0000          add     byte ptr [eax],al
0041d5aa e900200000    jmp     image00400000+0x1f5af (0041f5af)
0041d5af 54            push    esp
0041d5b0 41            inc     ecx
0041d5b1 47            inc     edi
0041d5b2 47            inc     edi
0041d5b3 43            inc     ebx
0041d5b4 0a01          or      al,byte ptr [ecx]
0041d5b6 0030          add     byte ptr [eax],dh
0041d5b8 820a3f        or      byte ptr [edx],3Fh
```

After 2 bytes from PE file entry point there is 8-bytes pointer to taggant location in physical file.

In our case, after pointer to taggant structure there is a long relative jump over it and the taggant structure itself.

Following structure describes how the data is located starting from PE file entry point:

```
typedef struct
{
        char ep_jmp_code;
        char ep_jmp_size;
        unsigned long long tag_offset;
        char over_tag_jmp_code;
        unsigned long over_tag_jmp_size;
        TAGGANT taggants[taggantsize];
} TAGGANT_EP, *PTAGGANT_EP;
```

there:

- **ep_jmp_code**, **ep_jmp_size** – short relative jump over 8-bytes pointer to taggant structure in physical file;
- **tag_offset** – 8-bytes pointer to taggant structure in physical file;
- **over_tag_jmp_code**, **over_tag_jmp_size** – long relative jump over taggant structure;
- **taggants** – taggant structure.

Common types and functions of taggant library are defined in the taggant_types.h and taggantlib.h files, so they should be included in the project too. Also, the SPV_LIBRARY preprocessor definition should be specified. Main code that creates taggant structure is located in p_w32pe.cpp file. The logic is divided on to two parts:

1) reserve space for taggant structure;
2) creating of taggant structure.

We reserve the space in the file (loader) for taggant using following code:

```
  // Expand loader memory to the TAGGANT_MAX_SIZE
  const unsigned lsize = getLoaderSize() + sizeof(TAGGANT_EP) + taggant_size;
  MemBuffer loader(lsize);
  memcpy(loader + sizeof(TAGGANT_EP) + taggant_size, getLoader(), lsize - sizeof(TAGGANT_EP) -
taggant_size);
  patchPackHeader(loader + sizeof(TAGGANT_EP) + taggant_size, lsize - sizeof(TAGGANT_EP) -
taggant_size);
```

After the file is completely packed, we are creating the taggant.

```
  printf("Start to embed the taggant\n");

  // Check if license file exists
  if (!file_exists(opt->license_name))
  {
        throwCantPack("license file does not exist");
  }

        // loading user license
        InputFile licin;
        licin.sopen(opt->license_name, O_RDONLY | O_BINARY, SH_DENYWR);
        off_t userlic_len = licin.st_size();
```

```
            char* userlic = new char[userlic_len];
            licin.read(userlic, userlic_len);
            licin.closex();

    // Initialize callback functions
    TAGGANTFUNCTIONS fncs;
    memset(&fncs, 0, sizeof(TAGGANTFUNCTIONS));
    fncs.size = sizeof(TAGGANTFUNCTIONS);
    long long unsigned int lib_ver = 0;
    TaggantInitializeLibrary(&fncs, &lib_ver);
    printf("Taggant Library is initialized\n");

    int err = 0;

    // get the physical offset of entry point
    int ep_physical_offset = oh.entry - osection[1].vaddr + osection[1].rawdataptr;

    // check user certificate and return it's expiration date
    UNSIGNED64 exp_date;
    if (TaggantGetLicenseExpirationDate((PVOID)userlic, &exp_date) == TNOERR)
    {
            // print expiration date
            printf("Certificate expiry date: %s", asctime(gmtime((time_t*)&exp_date)));

        // fill the taggant structure
        unsigned char* tagmem = loader;
        PTAGGANT_EP tag = (PTAGGANT_EP)tagmem;
        tag->ep_jmp_code = 0xEB;
        tag->ep_jmp_size = 8;
        tag->tag_offset = ep_physical_offset + sizeof(TAGGANT_EP);
        tag->over_tag_jmp_code = 0xE9;
        tag->over_tag_jmp_size = taggant_size;

        // write the current entry point and taggant information into file
        fo->seek(ep_physical_offset, SEEK_SET);
        fo->write(loader, sizeof(TAGGANT_EP) + taggant_size);

        // Create context
        PTAGGANTCONTEXT pTagCtx = TaggantContextNew();
        if (pTagCtx)
        {
            // Initialize callback functuions
            pTagCtx->FileReadCallBack = (size_t (__DECLARATION *)(PFILEOBJECT, void*,
size_t))&fileio_fread;
            pTagCtx->FileSeekCallBack = (int (__DECLARATION *)(PFILEOBJECT, UNSIGNED64,
int))&fileio_fseek;
            pTagCtx->FileTellCallBack = (UNSIGNED64 (__DECLARATION *)(PFILEOBJECT))&fileio_ftell;

            //compute the taggant and place it to the file
            long long obj_size = osection[2].rawdataptr + osection[2].size;

            fo->seek(0, SEEK_END);
            long long tag_file_size = fo->st_size();

            //initialize taggant object
            PTAGGANTOBJ tagobj = TaggantObjectNew(NULL);
```

```
      if (tagobj)
      {
        printf("Prepare to compute the hashes\n");

        UNSIGNED32 hashres;
        if (opt->taggant_hashmap)
        {
          long long tmp = pe_offset + 4 /* sizeof(pe_signature) */ + 20 /* sizeof(pe_file_header) */ + 64
/* offset to Checksum */;
          // Add region from file begin to CRC value of optional header
          TaggantAddHashRegion(tagobj, 0, tmp);
          tmp += 4;
          // Add region from CRC value of optional header to digital signature header of data
directories
          TaggantAddHashRegion(tagobj, tmp, 60);
          tmp += 60 + 8;
          // Add region from digital signature header to end of file header
          TaggantAddHashRegion(tagobj, tmp, osection[0].rawdataptr - tmp);
          // Add region of loader
          TaggantAddHashRegion(tagobj, lrd_offset, lrd_size);
        }

        hashres = TaggantComputeHashes(pTagCtx, tagobj, (void*)fo, obj_size, tag_file_size,
taggant_size);

        if (hashres == TNOERR)
        {
             printf("File hashes computed successfully\n");

             // set packer information
             PPACKERINFO packer_info = TaggantPackerInfo(tagobj);
             packer_info->PackerId = 1;
             packer_info->VersionMajor = UPX_VERSION_HEX >> 16 & 0xFF;
             packer_info->VersionMinor = UPX_VERSION_HEX >> 8 & 0xFF;
             packer_info->VersionBuild = UPX_VERSION_HEX & 0xFF;

             // try to put timestamp
             printf("Put timestamp\n");
             UNSIGNED32 timestamp_res = TaggantPutTimestamp (tagobj, "http://taggant-
tsa.ieee.org/", 50);
             switch (timestamp_res)
             {
             case TNOERR:
             {
                    printf("Timestamp successfully placed\n");
                    break;
             }
             case TNONET:
             {
                    printf("Can't put timestamp, no connection to the internet\n");
                    break;
             }
             case TTIMEOUT:
             {
                    printf("Can't put timestamp, the timestamp authority server response time has
expired\n");
```

```
                        break;
                    }
                    default:
                    {
                        printf("Can't put timestamp\n");
                        break;
                    }
                    }

                    // generate cms
                        printf("Prepare the Taggants\n");

                        UNSIGNED32 prepareres = TaggantPrepare(tagobj, (PVOID)userlic, (char*)tag +
sizeof(TAGGANT_EP), &taggant_size);
                        if (prepareres == TNOERR)
                        {
                            printf("Taggants generated successfully\n");
                        } else
                        {
                            if (prepareres == TINSUFFICIENTBUFFER)
                            {
                                taggant_size = (taggant_size + 0x1000 - 1) &- 0x1000;
                                printf("Buffer for taggant is insufficient, approximate necessary
size is %d (use --taggantsize=%d command line parameter)\n", (int)taggant_size, (int)taggant_size);
                            }
                            err = 1;
                        }
                } else
                {
                    err = 2;
                }

                    // finalize taggant object
                    TaggantObjectFree(tagobj);
            } else
            {
             err = 3;
            }
            TaggantContextFree(pTagCtx);
        } else
        {
            err = 4;
        }
    } else
    {
        err = 1;
    }

    //
    delete[] userlic;
    TaggantFinalizeLibrary();

    // check for errors
    switch (err)
    {
    case 1:
```

```
{
        throwCantPack("can't prepare taggant, license information is incorrect?");
        break;
}
case 2:
{
        throwCantPack("can't compute file hashes");
        break;
}
case 3:
{
        throwCantPack("can't create taggant object");
        break;
}
case 4:
{
        throwCantPack("can't create taggant context");
        break;
}
}


// write the taggant to output file
fo->seek(ep_physical_offset, SEEK_SET);
fo->write(loader, sizeof(TAGGANT_EP) + taggant_size);
```

7 SSV test program

SSV Test is a test program that allows to find taggant in the file, verify if the taggant is valid, extract certificates information and the taggant timestamp (the time when the file had been packed).

The Root Certificate, which will be used to verify the taggant, should be passed as a first command line parameter to SSV Test program. Please note, root certificate should be in PEM (Base64 encoded) format. The second command line parameter can be passed optionally. It determines path (or file) where the SSV Test program will search files with the taggant. If the second parameter is:

- file, then program will search taggant in this particular file only;
- folder, then program will recursively check all files inside this folder;
- not specified, then program will recursively check all files inside current folder.

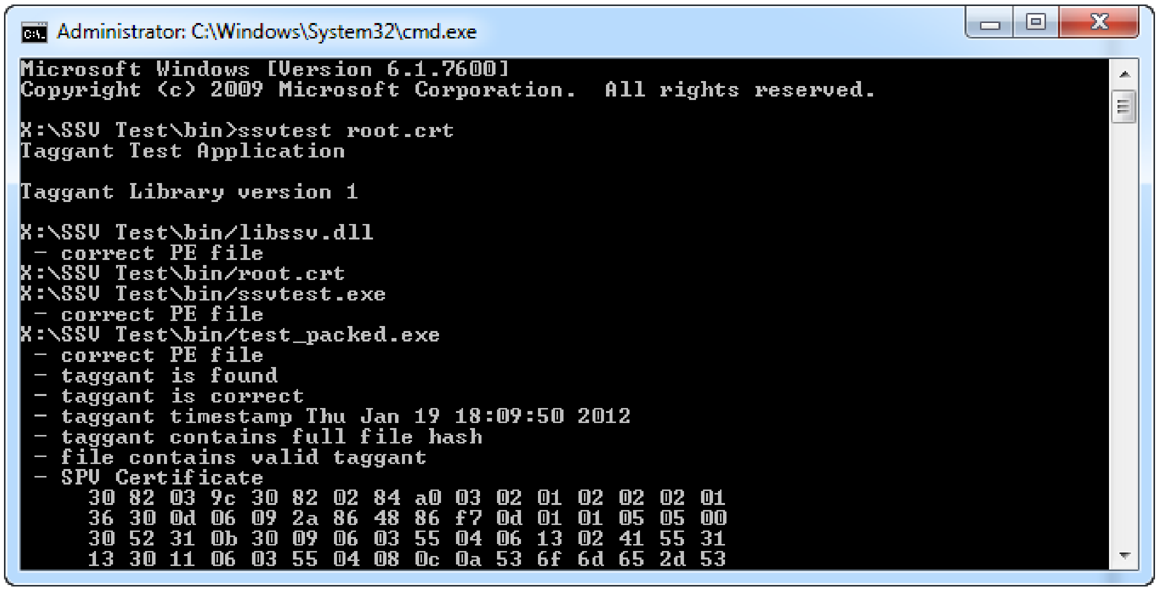
Example:

    ssvtest.exe root.crt tsroot.crt d:

there:

- root.crt – file with the root certificate;
- tsroot.crt – file with the root certificate for timestamp server;
- d: – disk D that will be scanned for the files with the taggant.

The process of files scanning is output to console. If the taggant is found then information about intermediate, user certificates and timestamp will be shown in console also.



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

X:\SSV Test\bin>ssvtest root.crt
Taggant Test Application

Taggant Library version 1

X:\SSV Test\bin/libssv.dll
 - correct PE file
X:\SSV Test\bin/root.crt
X:\SSV Test\bin/ssvtest.exe
 - correct PE file
X:\SSV Test\bin/test_packed.exe
 - correct PE file
 - taggant is found
 - taggant is correct
 - taggant timestamp Thu Jan 19 18:09:50 2012
 - taggant contains full file hash
 - file contains valid taggant
 - SPV Certificate
     30 82 03 9c 30 82 02 84 a0 03 02 01 02 02 02 01
     36 30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00
     30 52 31 0b 30 09 06 03 55 04 06 13 02 41 55 31
     13 30 11 06 03 55 04 08 0c 0a 53 6f 6d 65 2d 53
```