International Journal of Engineering & Technology, 7 (4.41) (2018) 9-14

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Using Steganography & Cryptography to Hide Data in EXE Files

Nisha P. Shetty[1]*, Nikhil Ranjan[2]

[1]Assistant Professor, Department of Information and Communication Technology, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal 576104 INDIA

[2]Student, Department of Electronics and Communication, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal-576104 INDIA

**Corresponding author E-mail: nisha.pshetty@manipal.edu*

Abstract

The most common term we hear when talking about data security is Cryptography; it comprises of practices and study of techniques for secure communication from one party to another without compromising private data that is to be shared between them. The objective of cryptography in simple words is to hide the plain text/data by modifying it into a cypher text/data such that no third party can read it easily. Another term less commonly known is Steganography; it is the process of hiding/encoding a secret message within an ordinary or dummy data file and the simple extraction of it at its destination.Our project aims to combine the best features of the two worlds. The raw data is first compressed, and then split into bytes for encryption via AES. Once encryption is done the encrypted data is hidden behind an exe file. Additionally a SHA-256 hash of the compressed original data is also included to evaluate file integrity.

Keywords: *Cryptography, AES, Steganography, SHA-256, EXE file format*

1. Introduction

In this information age where data is considered as an asset by corporations and governments, data security is becoming the centre of any digital storage/ network system related discussion or conference. To stop the illegal access or leak of data companies have used multiple techniques like encryption, digital signatures etc. The objective of this paper is to show a method which secures data behind an exe file such that it is undetectable while de-compilation or simple analysis. Not only does it include the best encryption algorithm and standards but also a robust hashing scheme along with steganography which makes it practically very difficult to breach through all of the layers.

Steganography does not make any attempt to alter the data to be sent, hence if an attacker finds out that a transmitted file has secret data within, he/she can easily uncover the underlying data by analysing the file thoroughly. On the other hand Cryptography makes no attempt to hide the data from plain sight and hence gathers a lot of attention to it. This project began as an attempt to combine the best of theses worlds and to use steganography to hide already encrypted data and also use hashing to ensure integrity. This will ensure that any hidden message will not gather attention of people around, and even if it is detected, the attacker will still need to decrypt the data. This provides a 2 layer security by combining Steganography, Cryptography and Hashing for tamper protection, privacy and security.

2. Background Theory

2.1. AES (Advanced Encryption Standard)

AES (Fig. 1) is a subset of the Rijndael cipher developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen. The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data. AES is currently the world's most popular encryption algorithm due to its highly robust encryption method and the use of long keys which make it harder to decrypt by brute-force.

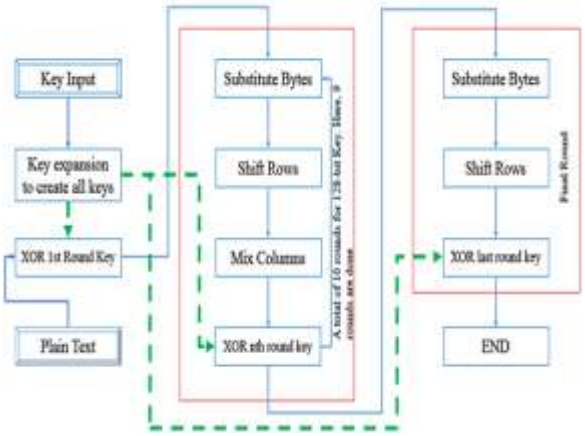

Fig. 1: Block schematic of AES encryption.

2.1. SHA-2 (Secure Hash Algorithm-2)

The SHA-2 (Secure Hash Algorithm-2nd Family) is a group of cryptographic hash functions. A cryptographic hash is like a signature for a file. SHA-256 and SHA-512 belong to this family. SHA-256 algorithm generates an almost-unique, fixed size 256-bit (32-byte) hash and similarly SHA-512 generates a fixed 512-bit (64-byte) hash for any given different data. Hash is a one-way function – it cannot be decrypted back just like CRC and they are also collision-resistant, this means that it is very difficult to find two different inputs will give the same Hash value.

2.3. Literature review

We referred multiple different papers in the domains of cryptography and steganography to find the best and most efficient way to complete our requirements [9][10]. The idea of combining steganography and cryptography is not a very old concept and recently has been grabbing the attention of many researchers and the IT industry.
To study steganography, Davern et al [3] talked about multiple steganography techniques used throughout history and in the computer age, it offers a general idea into the world of steganography. On the same note, Donovan Artz [8] introduces some examples of steganography along with some freely available tools for steganography and Stego-analysis.
Brojo Kishore Mishra compared the advantages and disadvantages of both cryptography and steganography [1]. He has suggested in his paper that the combination of both the methods is plausible and should be looked into. Aos A. Z. et al [2] describe as to why PE files are good for making cover data for steganography as they have variable sizes and this disparity is perfect for hiding data [6]. Thus the two papers mentioned in the paragraph helped in reinforcing our choice of choosing exe files for our project. For the programming aspect, Binal Shah et al give the performance metrics related to encryption and decryption of data using the languages C#, MATLAB and Java [4]. C# proved to be the fastest in performing operations hence, it was chosen for the project.
One of the best and most creative methods for hiding data was presented by El-Khalil et al who presented the "hydan" method for encoding data inside an exe file using functionally-equivalent instructions using a key-dependent process[7] in 2003. This method however limits the amount of information that is possible to hide in the file. M.M.Abdulrazzaq et al [5] used a different method. They developed new file handling methods for the OS and replacing the original ones supplied by the OS itself for hiding data in a PE (Programmable Executable) file. Our method does not require building new file handling methods and it can be done by using the original supplied methods itself.

2.4. EXE file format

Fig.2 shows the important memory blocks inside an EXE file. The memory block at the end of the EXE file (Base of Image Header: IMAGE_NT_HEADERS), is prone to padding of data without disturbing the meta-data of the file except the size [6]. The length of Image header is specified in the COFF header and thus only the required length of data is read if needed, any extra data is ignored. Thus, that specific section makes for a perfect cover for our data.

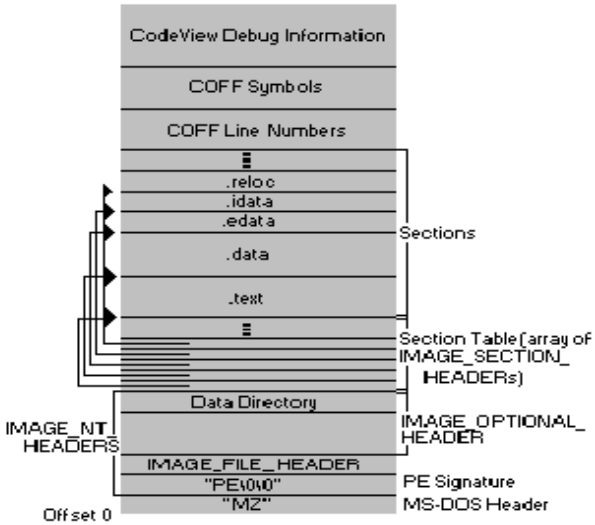

Fig. 2: EXE file format

3. Methodology

The objective here is to first compress, then encrypt the raw data to pad it behind an EXE file. After that, the data has to be extracted and then decrypted. All of this has to be done without changing the meta-data and behaviour of the cover EXE file.

The memory block at the end of the EXE file (Base of Image Header), is prone to padding of data without disturbing the behaviour of execution file and it goes undetected by most anti-virus programs as this section is not scanned by them [5]. Hence, this enables us to add an arbitrary amount of data at the end of the exe file without disturbing anything else and it will go undetected during de-compilation or any other simple analysis.

A simple implementation of the idea is discussed in this paper. The complete project was made on Windows 8.1 running .NET Framework 4.6.1, the language used was C# and the IDE used was Visual Studio 2017 Community Edition. The methodology used to achieve the objectives is shown in Fig. 3. Jet Brains dotPeek was used to analyse the exe files.

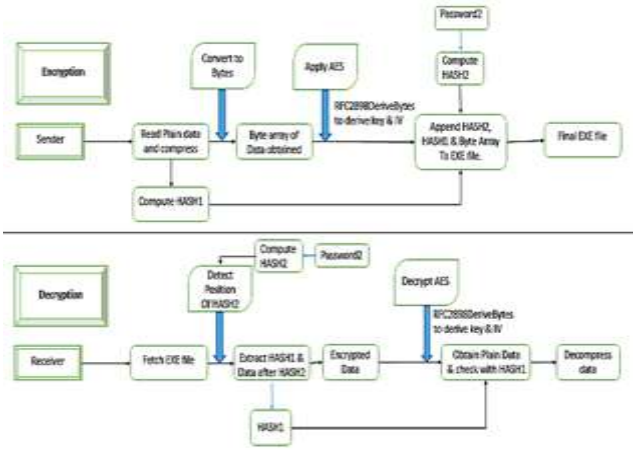

Fig. 3: Flowchart of methodology adopted

Below are the steps required to generate our Final-EXE file which will have appended data, it is a detailed version of the Fig. 3.

1. Select the files to be sent and compress them using ZIP. This helps in reducing footprint of the data and converts the raw data to a known extension i.e. '.zip'.
2. Compute Hash of the above compressed original data, let us call it HASH1. This will help in checking file integrity later. If data is tampered with, the hash value generated by the data would no longer be equal to HASH1 and hence we will be able to detect tampering.
3. Read all bytes of the above ZIP file and encrypt it using AES algorithm.
4. Details about the AES algorithm we used:
   (a) Block size:128;
   (b) Key Size = 128;
   (c) Set Padding Mode to PKCS7;
   (d) Define cipher mode as CBC (Cipher Block Chaining) so that the decryption of a block of ciphertext will depend on all the preceding ciphertext blocks. This will ensure validity of all the blocks.
   (e) Compute a key using RFC2898DeriveBytes (password-based key creator, uses a pseudo-random number generator for this purpose) using Password let us call this "KEY".
   (f) Derive Key and IV for AES-CBC using KEY. The key is used to generate sub-keys and the IV stands for initialization vector which derives the first key.
5. After Encryption save the final encrypted bytes in a .bin file.
6. Take another password which is called as Pattern Password and compute its hash, let us call it HASH2 (use SHA-512/256). This is important because HASH2 will help us to locate our data inside the EXE file.
7. Create a dummy EXE file with a random program.
8. Append to the EXE file: HASH2, HASH1 and the .bin file in that respective order and delete any other overhead data like the .bin file.

The final EXE file is ready to be sent over the network just as an ordinary file. It will be in this format shown in Fig.4.

EXE file	HASH2	HASH1	Data	EOF

Fig. 4: Structure of EXE file with data

To extract the data the following steps have to be taken:

To undo the steganography operation:

1. Read all the bytes of the EXE file and save to byte array for further processing.
2. Find the hash of the Pattern Password which will be HASH2 and search for it in the byte array.
3. Read all the bytes following the location of HASH2. This will give you the byte array which contains the encrypted .bin file and HASH1.
4. After this, we know the size of HASH1 which will be of 32/64bytes (depends on the hashing scheme used by the sender), and the data bytes following it will be the encrypted data. Separating them, we will get HASH1 and the encrypted .bin file.

   For Decrypting:

5. Decrypt the .bin file by using AES and entering the same key as mentioned during encryption.
6. Compute hash of the decrypted data and let us call it HASH3.
7. If HASH1 = HASH3 then the file has been successfully retrieved. However, if HASH1 is not equal to HASH3, try to find the next occurrence of HASH2 in the rare event that the EXE file also contains the same byte pattern as HASH2 somewhere in between it and repeat the process. If not found, that means data has been compromised.
8. Extract the data from the ZIP file.

4. Result Analysis

4.4. Program in Action

This section consists of screenshots which show the program working just as presented in the methodology section.

Home Screen.

Upon opening the program, the user is greeted with the window shown in Fig. 5. It consists of 3 functional buttons:

- Exe_Packer': Used to open a different form that can be used to compress, encrypt and pack data behind EXE file
- 'Exe_Opener': Opens a different form that can be used to decrypt and unpad data from the exe file.
- 'Create dummy exe': Creates a dummy exe file which upon execution will print multiplication tables.

Fig. 5. Home Screen

Create Dummy EXE file.

When the 'create dummy exe file' button is pressed, as shown in Fig. 6, a small exe file of 4KB is created which will be used for appending data. The folder we have chosen to hide behind the exe file contains one .jpg image of size 128KB. Folder contents and size of file is shown in Fig. 6 and Fig.7.

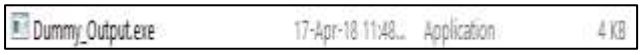

Fig. 6. Original EXE file

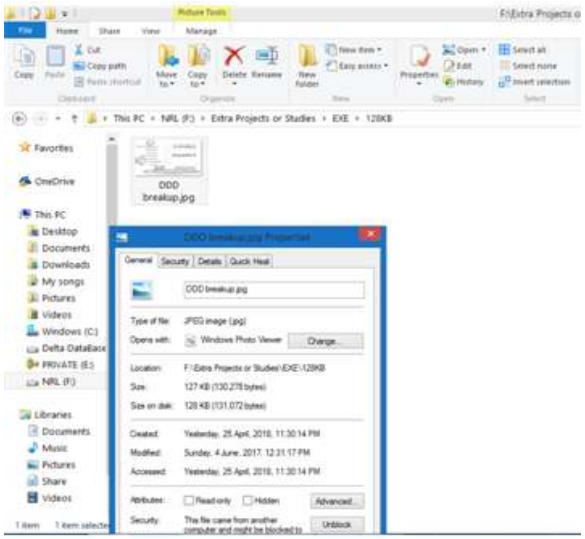

Fig. 7: Sample data

Exe Packer.

Fig. 8 shows EXE Packer taking inputs as folder, encryption algorithm and key to compress and pack data. The final file created after compression and encryption is shown in Fig. 9. The file was first zipped automatically for compression then encrypted to create a .bin file which can be used to dump binary data. Next, we need to pad the encrypted data to the dummy exe file. Fig. 8 again shows the required inputs i.e. location of the dummy exe file and the second password given and the "pack" button being pressed. The program returns a message box saying that the padding was successful. On choosing the dummy EXE file, the encrypted file ('Data.bin') is padded to the dummy exe file. This leaves us with a 108KB+4KB = 112KB EXE file as shown in Fig. 10.

Exe Opener.

Upon execution of the EXE opener form and putting in the required values, we should be able to get back the data. As input to the software we will provide the following:

1. Location of exe file with data
2. Password used for pattern detection
3. Encryption algorithm (in this case fixed i.e. AES)
4. Cipher Key for AES.

Providing this information is sufficient for retrieving back the data and decrypting it as shown in Fig. 11. The resulting output file in .zip format is shown in Fig. 12. When we open the file using WinRAR, we get an unpacked size of 130,278 bytes which is equal to 128KB and the folder contents are the same in Fig. 13 and 7. Hence, data has been successfully retrieved back.

Fig. 8: EXE Packer Encrypting Data

Fig. 9: Compressed & Encrypted binary file.

Fig. 10: Dummy EXE file with data

Fig. 11: EXE Caller Retrieving back the data

Fig. 12. Extracted Data in Compressed form

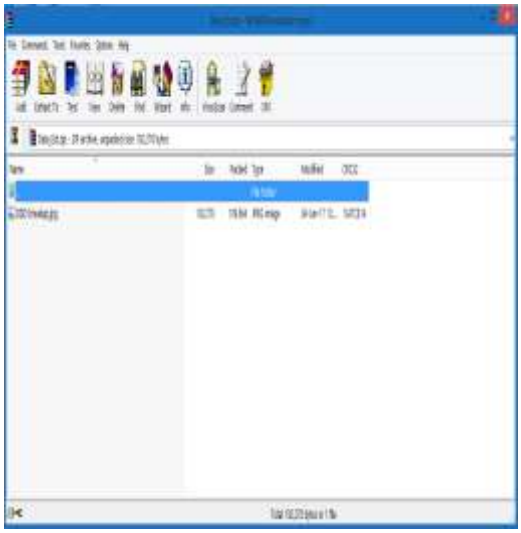

Fig. 13. Final Extracted Data

4.5. File Analysis

The following screenshots in Fig. 14 and Fig. 15 shows the "JetBrainsdotPeek" software retrieving the meta-data and code of the original exe file without data and the modified exe file with data. It is clear that the 108KB appended data is nowhere to be found upon analysis. This is because it is hidden in a section where no analysis is done. Let us have a look at the meta-data and code of the original exe file shown in Fig. 16 and Fig. 17 and compare them for differences. We are trying to look for any differences in meta-data or code of the exe files because if differences are found, this would be a disaster. The objective of the project was to append data 'without' disturbing the exe file.

On comparing metadata of the exe files, we see that the EXE files look the same, this is because the data was appended in a section which is usually never scanned. All the values in the metadata are same for the both the files, however file sizes for both will be different due to padding of data. Looking at the code, we see that both files decompile to give the same code, hence upon execution, the outputs for both the files will be same.

5. Conclusion

5.1. General Conclusion

We can see that such a method could not only ensure a very high data security standard but also the data would be hidden from peering eyes, hence we have successfully combined the best parts of cryptography and steganography. This helps in keeping the confidentiality of critical data that was hidden. Also, format of the data to be hidden or extension is irrelevant as any given file type will be compressed to .zip format before encryption or decryption.

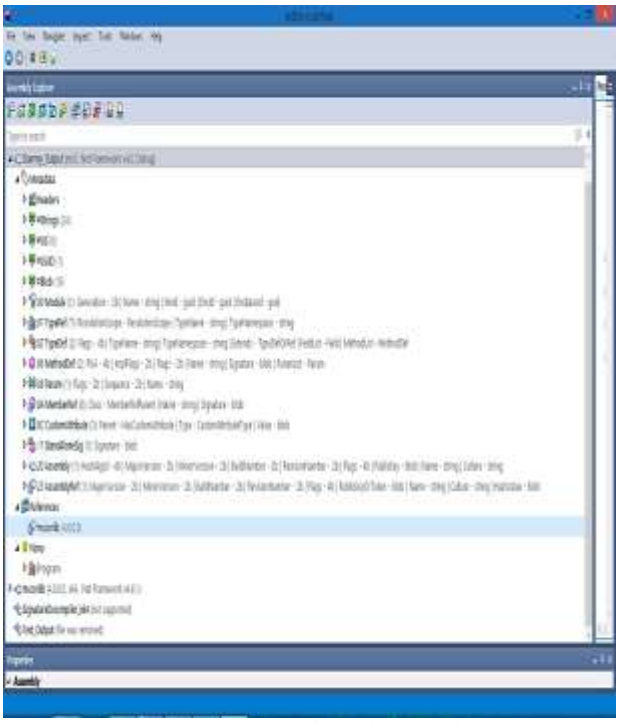

Fig. 14: Metadata of modified EXE file

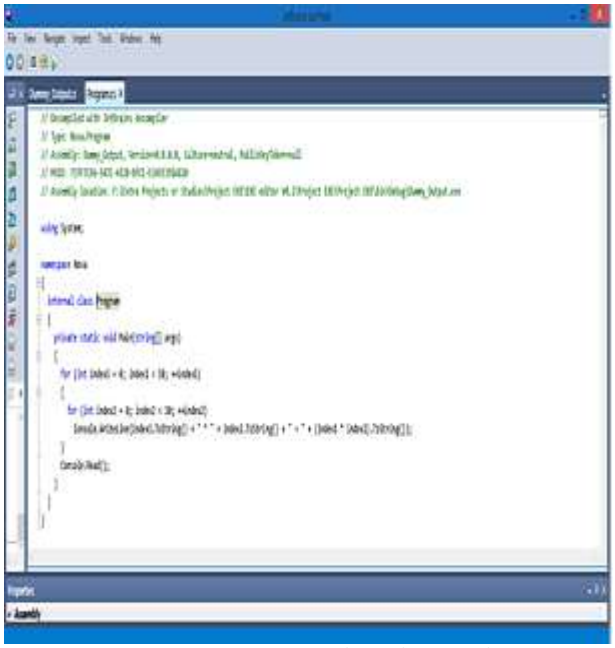

Fig. 15: Decompiled code of modified EXE file

Fig. 16: Metadata of original EXE file

Fig. 17: Decompiled code of original EXE file

5.2. Future Scope

Currently, this method is applicable for only exe files, however to incorporate other file types more research needs to be done to find vulnerabilities in the different file types. Research into multimedia type files like video or audio can also be thought upon as their huge sizes will allow for more data to be hidden inside undetected. Newer encoding methods must be thought upon to achieve said goals.

File signatures are the only one security measure that prevents such type of data addition. File signatures are the last and only line of defence left for stopping unauthorized padding of data. Research into vulnerabilities into file signatures by modifying PE header etc. can extend this method of signed executables/files without compromising the signature of the file itself.

References

[1] Haripriya Rout, Brojo Kishore Mishra,"Pros and Cons of Cryptography, Steganography and Perturbation techniques". IOSR Journal of Electronics and Communication Engineering (IOSR-JECE) e-ISSN: 2278-2834, PP 76-81. ISSN: 2278-8735

[2] A. A. Z., A. W. Naji, S. A. Hameed, F. Othman and B. B. Zaidan, "Approved Undetectable-Antivirus Steganography for Multimedia Information in PE-File," 2009 International Association of Computer Science and Information Technology - Spring Conference, Singapore, 2009, pp. 437-441. doi: 10.1109/IACSIT-SC.2009.103

[3] Davern, P.S, M.G, "Steganography It History and Its Application to Computer Based Data Files", School of Computer Application (SCA), Dublin City University. Working Paper. Studies (WPS), Baghdad, Iraq, 2007.

[4] Binal Shah &ZahirAalam, "Implementation and Performance Evaluation of the AES Algorithm for Data Transmission using Various Programming Languages",Communications on Applied Electronics (CAE) – ISSN : 2394-4714 , Foundation of Computer Science FCS, New York, USA Volume 3– No.4, November 2015.

[5] M.M.Abdulrazzaq, Hilal M. Yousif Al-Bayatti, Moayad A. Fadhil, "A Proposed Technique for Information Hiding in a PE-File", Journal of Advanced Computer Science and Technology Research, Vol.3 No.4, December 2013, 153-162.

[6] Goppit. (2005)," Portable Executable File Format-A Reverse Engineer View" http://www.CodeBreakers-Journal.com. VOL. 2, NO. 3, 2005.

[7] El-Khalil R., Keromytis A.D. (2004) Hydan: Hiding Information in Program Binaries. In: Lopez J., Qing S., Okamoto E. (eds) Information and Communications Security. ICICS 2004. Lecture Notes in Computer Science, vol 3269. Springer, Berlin, Heidelberg

[8] D. Artz, "Digital steganography: hiding data within data," in IEEE Internet Computing, vol. 5, no. 3, pp. 75-80, May-June 2001. doi: 10.1109/4236.935180

[9] Behrouz A. Forouzan,"Cryptography & Network Security". ISBN-13: 978-0-07-0660-46-5, 2007 Indian Edition

[10] William Stallings, "Cryptography& Network Security: Principles & Practice" 5th Edition'. ISBN 13: 978-0-13- 609704-4