DOI: http://dx.doi.org/10.26483/ijarcs.v9i1.5526

ISSN No. 0976-5697

Volume 9, No. 1, January-February 2018

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

HASHING BASED ENCRYPTION AND ANTI-DEBUGGER SUPPORT FOR PACKING MULTIPLE FILES INTO SINGLE EXECUTABLE

Abhi Gupta
Department of Computer Science
Baba Farid Group of College
Bathinda,India

Dr Meenakshi S Arya
Department of Computer Science
Baba Farid Group of College
Bathinda,India

Abstract : During the recent times, Software in computers use an external layer of protection to provide an extra blanket of security against crackers and reverse engineers. This is done using specialized software commonly termed as packers. Mostly all packers use the approach for packing that is easily unpack-able by crackers; also many unpacking methods are available over the Internet to deal with approach . The paper proposes a novel packer algorithm that uses encryption and an anti-debugging technique to keep away reverse engineers. The proposed algorithm can pack multiple software at the same time. The experimental results demonstrate the effective packing ability of packer on the object files.

Keywords: Information security; Multiple Packing; Hash key encryption; Process Writing; Reverse Engineering; Anti-debugging; Anti-reverse

1. INTRODUCTION

The final computation by any computer program is performed using executables (.exe) files, which are capable of achieving a specific goal as per the instructions specified within them. Though the system software may immensely depending upon the architecture however at lower level, all software is a set of instructions only. Any alterations made to the executable file can give a different result irrespective of other factors. A potential attack on the system can read sensitive data from the executables and change the same to get access to or alter it's functionality by replacing the executable code with another code[1]. This in turn can allow the attacker to access any file on the computer and send it on to remote location through internet. The executables can be altered in many ways [2]:

Program Analyzers: Programs analyze at file level. Executables can be modified at binary level in secondary storages. At binary level they are just hex values There exist many programs and tools[3] (such as CFF_Explorer and HEX-WORKSHOP) which can convert these values into meaningful information and even a novice cracker can modify this information thereby causing potential harm to the system software.

Debugging: If an application allows debugging, then attackers can execute the application in debugging environment, where they can put breakpoints and study the application by executing it step by step. The changes can also be made during the run time. Various debugging tools are: windbg, ollydbg [4], IDA Pro.

Memory dumping: Using tools (Lord-PE, olly-dbg memory dump plug-in) process memory contents can be written into a file. Later these can be analyzed for meaningful information.

Data directories analyze: By analyzing the data directories study the behavior of an executable can be studied and understood which is inclusive of the imported functions and TLS. A lot of information can be found in these directories.

To solve above problems there is a solution named packer [5], it gives extra layer of security without affecting the internal functionality of an application. Basically security is achieved through encryption, but packer does another job also that is optional, it names compression. Before executing the first instruction related to an application, decryption and decompression must to be done at the execution time. However, as compression is just related to size of executable so it can be treated as optional. In this paper a technique of packing by using hash key based encryption is proposed. The technique is able to pack multiple applications at same time. In the proposed algorithm, during the execution time, a separate process space of different applications is created that result in isolation between them. A flag variable is used to keep track of the operation of whether the files are being packed or unpacked. Once the file is packed, the flag variable is changed to represent unpack. The paper is organized as follows: Section II throws light on the background and the related work done in the field, Section III describes the overall process design and the algorithm, the system evaluation and results have been performed in Section IV whereas Section V summarizes the work and also presents the future scope of the proposed technique.

2. BACKGROUND AND RELATED WORK

A. PE file format

Windows executable file particularly follows a format which should satisfy many constraints, then only windows loader is able to load it. An executable in windows environment is not just stand alone application like plain assembly instructions, there may be lot of information that may be required by that executable. That information is present in different section of that executable. A window NT

based application has 9 sections. But some applications may also have only two sections, and some may have more. But the important thing is information in these sections [6].

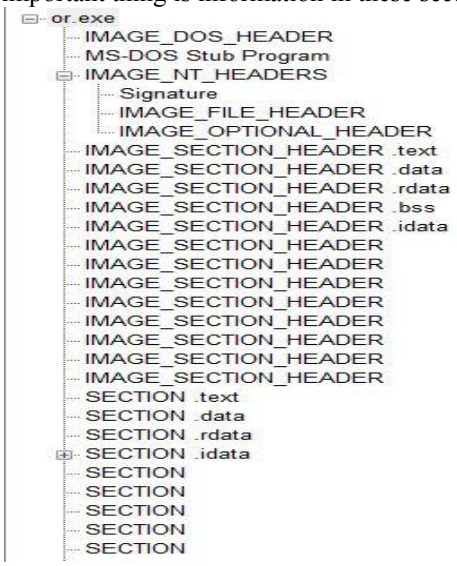

Fig.1 PE file format[14]

Many files in windows environment follow this format like: DLL, COM, SYS, and OCX.

B. Windows loader

The loading process just starts to begin when we double click an executable. It has the following steps:

I. Read the first page table of memory that contains: DOS headers, PE headers, and section headers.

II. Determine the memory requirements in address space, if it is not available, then allocate another memory reign of page.

III. Place all the sections required into the address space as mentioned in the section headers.

IV. If the image is not loaded at its preferred base address then relocate using relocation tables.

V. Walk through all the dlls required by executable. If any dll is not loaded into the memory, loads that dll into its share memory.

VI. Resolve all imports of import address table.

VII. After creating initial heap and stack create main thread of process and start the process.

C. Packer

Packer is a tool which encrypts the executable and adds extra code into it, and according to its needs it also modifies the headers of executable file. It makes the arrangements like first stub (packer code) is executed which decrypts all the packed executable part, and then imports are fixed. After that actual code related to that executable is executed. It just does one side of encryption and put the code in executable that do another side. This technique can easily evade any static checking based on signature. So without understanding no one can modify the content of executable.

D. PEB

Process environment block contain process related information it can be found at 30h of the segment register FS. PEB is allocated by operating system itself.

E. Related work

In [5] the details about internal structure of windows based executable have been discussed. This paper provides a lot of information required to develop the executable under windows environment, and it also describes the things that we have to tackle while altering an executable. It also describes the kind of tasks that a window loader program does on an executable before it begins the execution of the same. It elaborates on the role of different sections and various information fields of executable file. To create packer we have to understand all these fields. And this paper clearly describe every perspective of executable file under windows environment.

Fanglu Guo & Peter Ferrie & Tzi-cker Chiueh [7] presents just in time unpacking method. In the proposed work, let the unpacking algorithm is let to work for itself. The algorithm just checks the stack pointer and dirty page execution. So with debugger's attachment some selective packed applications can unpack manually. Most viruses come packed with their binary image. So due to this signature based checking of any AV program can easily be evaded. But to run any packed executable they must been unpacked at memory or secondary memory then they can fed to operating system. Once the unpacked algorithm does that, the method keeps checking the memory written by application once it detected that the unpacking is done. Then it calls the AV routine for signature based scanning. At this time it can work fine. The proposed algorithm just inserts breakpoint at the end of unpacked code in application. Once control reaches at that breakpoint it means the unpacked algorithm has done its job. Then another common way is to make store dump of memory that have the unpacked image after some fix (imports, Address entry point, etc). The dump file is able to run directly, so it shows a way how many packers security can be evaded by using just in time technique of unpacking.

Yu San-Chao & Li Yi Chao [8] presents an unpacking technique which is remove code obfuscation from executable. It is named AG-Un-packer. It decides when the program decrypts itself in memory completely by tracing long jump or intersectional jump. The algorithm is able to find imports address table in packed executable. All this is done by stack trace and forensic tracing. All calls to API's go through these import address tables. This technique can deal with the known and unknown packers. The concept of PUSHAD (push all general purpose registers in stack) and POPAD (pop all general purposes register from stack) is used. After the POPAD instruction there is JUMP instruction. When all come in the PUSHAD, POPAD, and jump sequence it means that is stub code. Breakpoint can be inserted at that end of code before jump. The proposed technique captures these intersectional jumps by using exception handling and is capable of monitoring the end of unpacked code in many packers. This is common for most packers who use this approach. 81% of packers use this technique for unpacking the packed executable.

Sungwon Han & Keungi Lee & Sangjin Lee [9] present signature based detection method of forensics in PE files. The approach uses signature based technique to find the type of packer or encryption method, as to detect signature of malware there is need to detect the signature of packer. The algorithm implements both ways of detection (signature based detection and entropy based detection). There are

some limitations in signature based detection as it can't detect new encryption methods or change in existing encryption methods. And entropy based detection have there own problems. But in this paper detection with entropy statistic of entry point is provided. Many third party PE identifiers are based on this technique.

Lee Ling Chuan & Chan Lee Yee [10] presents the technique to detect computer worm. At each stage the worm changes its code and makes it difficult to detect using static analyze as each time the signature of the executable will be different from the earlier. But at execution time it again comes with its original form. It unpacked itself first to achieve its goal. The technique is used to compare code of worm with different stages and when it performs its execution before it goes through its unpacking algorithm following the extraction of malicious worm. The algorithm also puts breakpoint analyzes the code for its signature.

Tao Ban & Ryoichi Lsawa & Shanqing Guo [11] present technique to identify the packer on malware. They use p-spectrum induced linear support vector machine to implement automated packer identification with good accuracy and scalability. This method helps to improve the scanning efficiency of anti-virus program by using the same approach of unpacking the executable into an empty section, fix their imports and then jump to original address of entry point. Finally the signature of unpacked code are matched with the signatures provide in the database for known packers. By identifying the signatures, they provide a lot of information regarding the packed executable like compiler version, compression ratio, imports names, etc.

Ang Li & Yue Zhang [12] proposed a technique to pack an executable based on tokens. This retains the confidentiality of protected PE file with the security token. They introduced new techniques of anti debugging and anti dumping of memory segment. These things are essential to unpack the executable using its algorithm. But the results show that the algorithm has a higher time complexity as compared to traditional methods.

Xabier Ugarte-Pedrero & David Balzarotti & Igor Santos [13] presented information about how runtime packers work. They are mostly used by malware writers to obfuscate their code. It creates a main hurdle to anti-virus static analyze, and any signature based detection can easily be evaded by introducing packing algorithm in executable. In their study they also show different ways to unpack a packed binary image of executable. In the run time they used a virtual environment for execution of an executable. All the control lies with the other process which is able to monitor the executable and give support for executing (debugger, emulator). The paper also throws light on different types of complexities faced normally while designing a simple packer.

3.OVERALL DESIGN

The proposed algorithm can inject multiple executable files into itself. Operating system does not allow editing the file that is mapped with process, and it only allows share read operation on that file. So to achieve this, a child process is introduced that does the job for main process. Some arguments need to pass from main process to child process. This is achieved through memory level writing. But to write memory of another process first we must create memory in that process which is not shared by any other operation. To do that both main and child processes must use same signature for reaching at the same memory location. Child process is created in suspended mode where EAX register in its context always has address of its entry point; by this we can reach the image base of child process. Once main process is done, then it just resumes the main thread of child process and gets terminated, and frees the mapped file. Then the child process begins its execution. All object files are either present in any of section of main file (depend on either padding space can hold file or not) or they are placed at the end of main file. Each file just follows an offset value, which points at the end of the injected file.

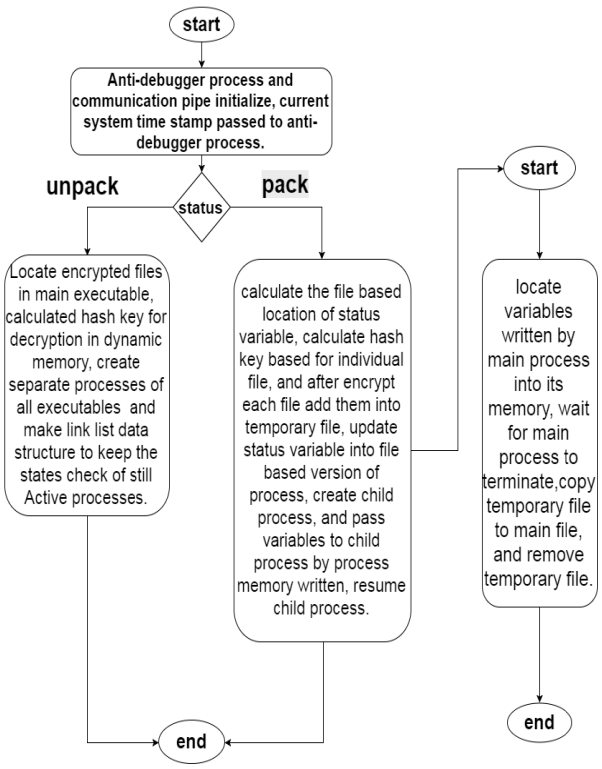

Fig.2. Flow chart of overall design

Proposed algorithm consists of 3 parts an Anti-debugger, un-packer and Packer. This algorithm will either run into un-packer mode or packer mode which is decided by status flag. But anti-debugger module will run in both situations.

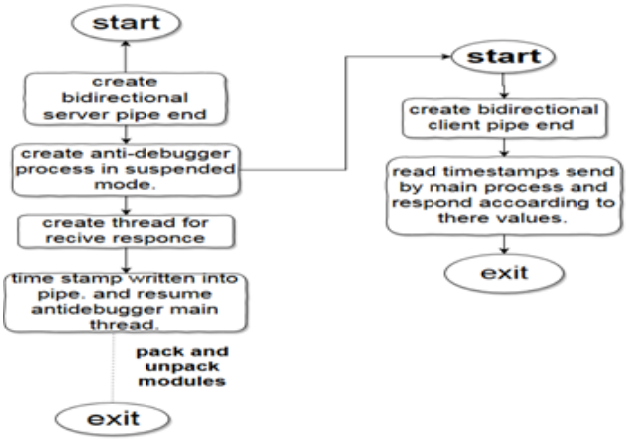

Fig. 3. Flow chart of anti-debugging module

Main process creates child process in anti-debugger module which always checks the presence of any known or unknown debugger or any cracker presence by the current status arguments passed by main process to it. At any stage child process can halt the execution of main process. It also provide an efficient solution for PUSHAD signature scanning at debugging time.After that the main process applies the encryption algorithm on object files and puts all result into a temporary file. As running processes mapped file cannot be edited, to do so we must wait until the main process is terminated and releases its mapped file. Before terminating itself the main file creates a child process and writes information about main process handler, source name, and destination name in it. It resumes the main thread of the child process which was created in suspended mode. After that child process starts its work. The flow chart of main process is shown as in fig.4.

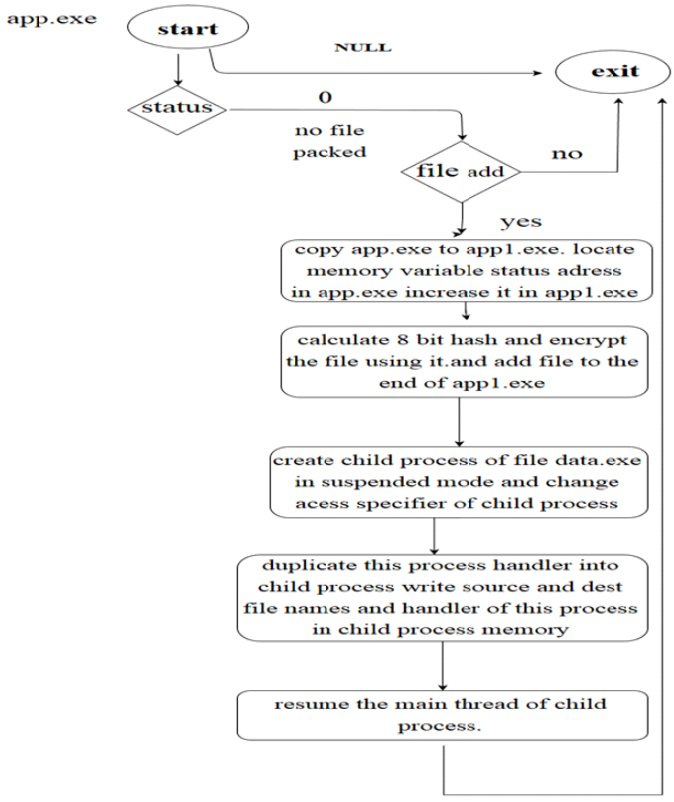

Fig.4. Flow chart of packing

The main objective of the main process is to find the file based location of a memory variable STATUS, and to make a copy of itself by reading its mapped file in secondary storage, opening object files locations according to files sizes, determining their location in the main file's copy, encrypting using hash key that is calculated based on file size and is unique for each file and finally putting these files into their appropriate locations. After that main process creates child process which is also pre compiled code. After creating executable environment the main thread of child process is resumed. In fig.5 Flow chart of child process shown. This is responsible of deleting main file after copy it from temporary file.

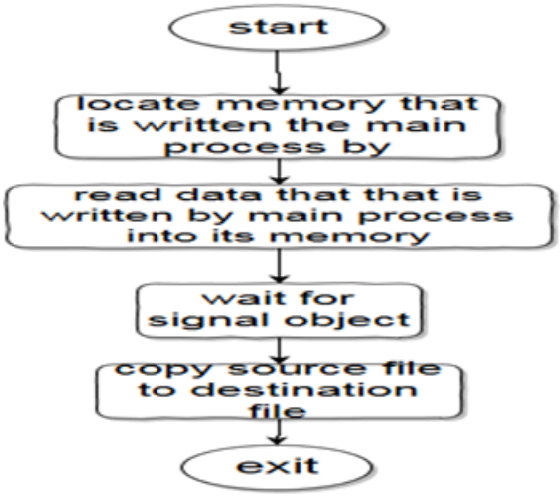

Fig.5. Flow chart of child process

Child process initially calculates the memory region where main process writes the arguments that are required for its operation. After finding that memory it reads the contents written and initializes its variables accordant to arguments. Then semaphore or wait for signal object can use. That API blocked that process until it receives the information regarding main process dethatched, which means main processes mapped file is free now for write operation. Then it just performs the copy operation between source to destination. After that it just removes the source application. Fig.6 shows how main file looks after all this. More file will be added to main file it increase space in main file.

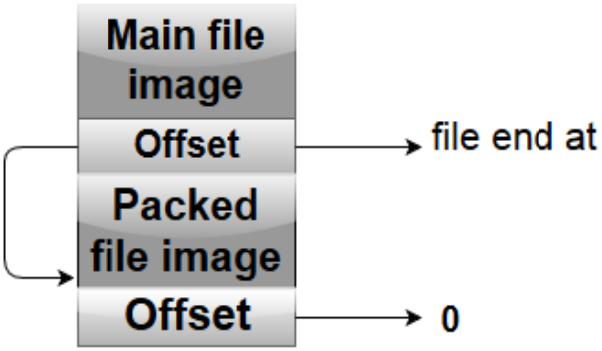

Fig.6. File representation

All files in the main file are encrypted using hash key value. So at this point all information regarding object files are secure. Here offset is the offset value that points the end of file that is going to begin at next 4 bytes. Each time that offset is used to calculate the size of file, and when offset is zero that means end of file. In fig.7 there is the flow chart of unpacking. If the status variable contains value then that indicate at un-packer algorithm. This algorithm just unpacked the files just by doing reverse operation. First it creates its global variable writable which is only readable by default. This gives an access to write n numbers of file names for creating processes for them. Then process of finding packed files is beginning. This is done via analyze each

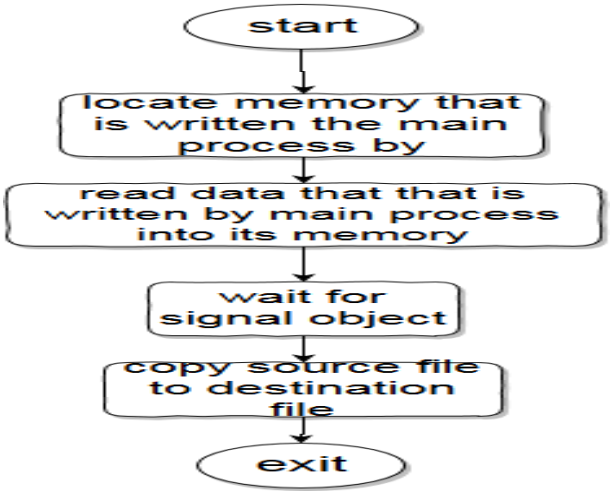

Fig.7. Flowchart of unpacking and executing module

Section table of main file, and if there is no file signature in any section then it must be at the end of main file. Using file offset we reach at the end of file where first file ends. Then in loop statement the followings steps are done:

1. Use offset value to calculate size of file. Allocate memory in dynamic allocator according to file size.
2. Calculate 8 bit hash key. And unpacked the file into dynamic memory.
3. File data is written to file by using string in global memory, and after that string is increased. Then current handler regarding file is closed.
4. Process data is initialized after that process is created.
5. Allocator memory is deallocated for reuse.

4. SYSTEM EVALUATION AND RESULTS

Portable executable in windows environment are not based on just one concept, which is not like just start execution whatever feeds to loader. There are lots of things to be done before the actual execution of portable executable. And these things are not same for all executables but there sequence is same implying that it just depends on the nature of executable. Like: - .DLL files, .COM files, .EXE files, etc. do not differ just in their extensions but in a lot of other different ways as well. Some executables export their functionality to other executable while some import some code from other DLL. While some executables are compiled with TLS set value, others need specific resources (GUI Texture). Some need debugger support and some use heavily use of operating system exceptional handler to achieve its goal. The loader is however capable of handling all these by itself. Windows loader creates the environment for execution of particular file and sets up all the prerequisites needed before the execution of file. When the environment is fully ready, then actual control is transferred to the first instruction of the executable. Hence, when an executable is packed then it needs to be either unpacked so as to facilitate the window loader to setup environment accordingly else the entire job of loading the application will have to be done by the application itself before jumping to the actual code of executable file. When all things get resolved, then only the executable can run.

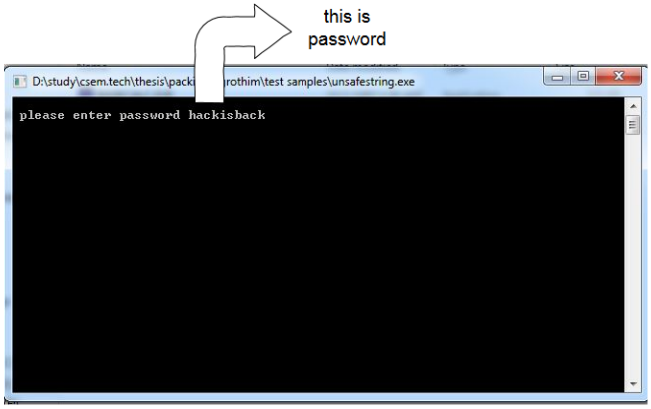

Fig.8. Output of Unsafestring.exe without packing

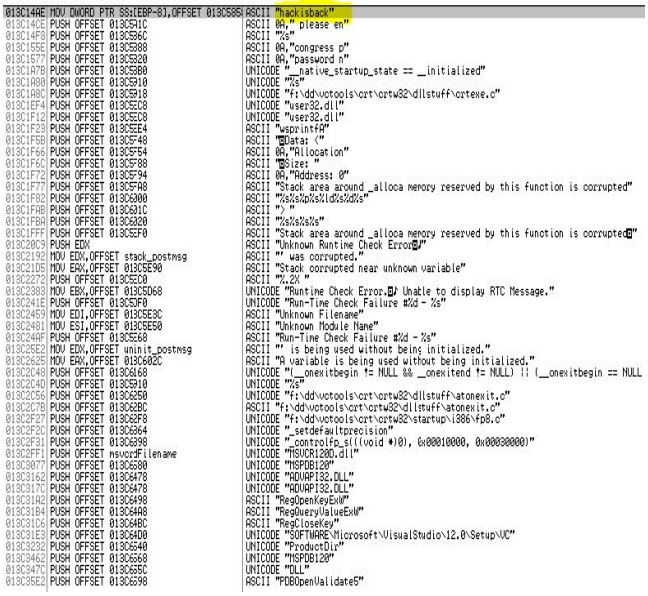

Fig.9. String references in Unsafe String.exe without packing[15]

Fig.8 depicts an application which just compares the string with user input string implying that string is saved somewhere in file itself. The saved string can be seen in Fig.9.

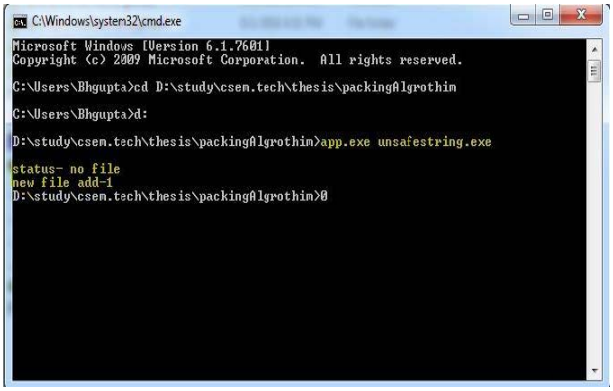

Fig.10. Packing of unsafestring.exe

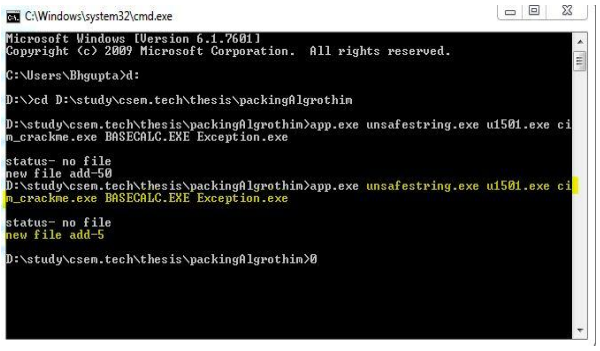

Fig.11. String references in Unsafe String.exe after packing[15]

All the contents of the file are changed as can be seen in Fig.11. After applying the proposed packing, all strings contents of file have changed. Now the executable can easily bypass any kind of signature based detection and crackers can't understand that. They don't represent any useful information now.

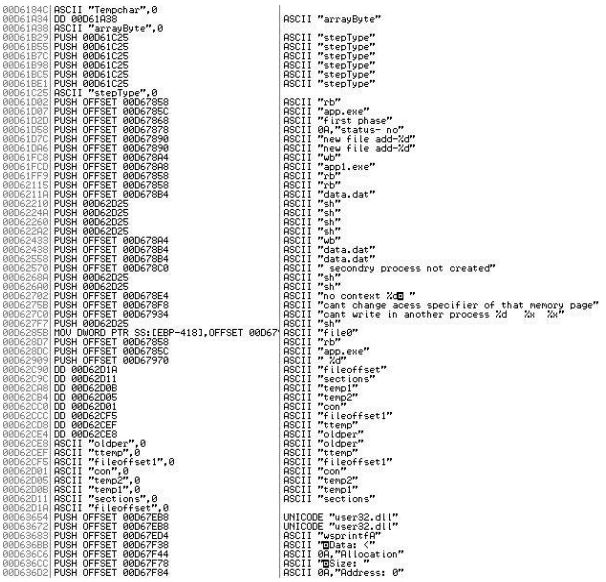

Fig.12. Multiple executable packing

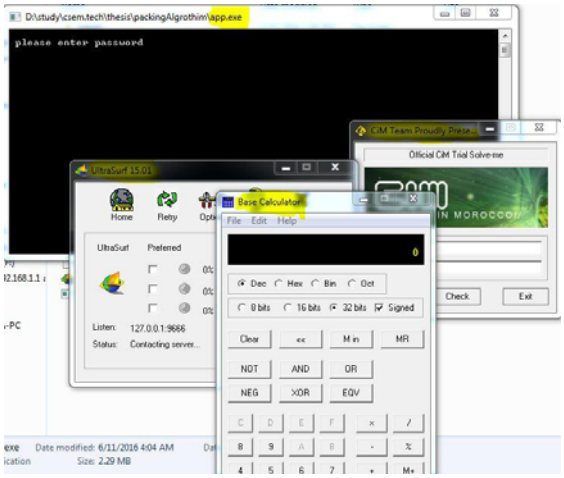

Fig.13. Execution of packed files

In another experiment 4 executable files are passed as input to the algorithm, which converts all of them into one single executable file. When that single file is executed, then all the files packed within are executed as depicted in Fig.13.

5. CONCLUSION AND FUTURE WORK

In this paper , a novel encryption algorithm based on hash key is proposed. The algorithm has been implemented on various window based x86 & x64 executables. It injects the executable files into itself. Hashing has always been used for verifying the data content but in the proposed technique, it has been used for encryption as size of the image is the only thing that is not changed in whole image. The technique has been implemented on many window based executables and experimental results show that this technique is capable of packing up to 2 GB executable files. The proposed technique assures packing of any kind of executable file. The experimental results show the execution of multiple packed files also. Proposed technique used different process space for different executables by letting the operating system create abstraction between all executables which are packed. Future scope of the research includes:

- Protection function for anti-debugger (timestamp compare, checking of debugger process, ENDIAN order for anti memory dump)
- Encryption with wider byte rage for generating key like 128 bit.
- Direct process level writing of unpacked image.
- Multiple threads can used to improve speed of execution.
- Multiple timestamps for anti-debugger.
- Extend the file capability more then 2 Gb by using mapped files instead of heap

6. ACKNOWLEDGMENT

First and foremost, I would like to express my gratitude to the Almighty God, for giving me the ability to think, research, and investigate. As we say in Hindi that "it is the teacher who brings you close to God" so a heartfelt thanks to my advisor, Dr Meenakshi S Arya,for their valuable supervision and support in every step taken in the preparation of this research. It was indeed an honor to learn under them, then I am grateful and obliged to all faculties of Computer Science & engineering department for molding me at correct time so that I can have a touch at fins destination. Then I would like to thanks all my friends for the moral support and encouragement, finally and most importantly, I would like to thank my parents, for their unconditional love, encouragement and support.

REFERENCES

[1] Philip O'Kane, Sakir Sezer, and Keiran McLaughlin. "Obfuscation: The hidden malware.",IEEE Journal of Security & Privacy,Vol.9.5,pp.41-47,sept-oct 2011.

[2] Li, Ang, et al. "A token strengthened encryption packer to prevent reverse engineering PE files." Estimation, Detection and Information Fusion (ICEDIF),International Conference on. IEEE, jan 2015.

[3] Hexrays Software "Hex-editor"[online] Avaliable: https://www.hexrays.com/products/ida/support/download.shtml.

[4] Oleh Yuschuk Software "Ollydbg"[online] Available: http://www.dc214.org/notes/rev_eng/Docs/OllyDbg%20Shortcuts.pdf.
[5] Microsoft Corporation, "Microsoft portable executable and common object file format specification revision 6. 0", 1999.
[6] Pietrek, Matt. "Peering inside the PE: a tour of the win32 (R) portable executable file format." Microsoft Systems Journal-US Edition, pp. 15-38, 1994.
[7] Fanglu Guo, Peter Ferrie, and Tzi-Cker Chiueh. "A study of the packer problem and its solutions." Springer Journal of Recent Advances in Intrusion Detection,Vol. 5230, pp. 98-115, 2008.
[8] San-Chao Yu, Yi-Chao Li, "A unpacking and reconstruction system-AGUnpacker" International Symposium on Computer Network and Multimedia Technology, International Conference on. IEEE,pp. 1-4, 2009.
[9] S. Han, K. Lee and S. Lee,"Packed PE file detection for malware forensics." IEEE Journal on Computer Science and its Applications, pp. 1-7, 2009.
[10] Lee Ling Chuan, Chan Lee Yee, Mahamod Ismail and Kasmiran Jumari, "Automating uncompressing and static analysis of Conficker worm." 9th Malaysia International Conference on. IEEE, pp. 193-198, 2009.
[11] T. Ban, R. Isawa, S. Guo, D. Inoue and K. Nakao, "Efficient malware packer identification using support vector machines with spectrum kernel."Eighth Asia Joint Conference on Information Security,pp. 69-76, 2013.
[12] Ang Li, Yue Zhang, Junxing Zhang and Gang Zhu, "A token strengthened encryption packer to prevent reverse engineering PE files", International Conference on Estimation, Detection and Information Fusion, pp. 307-312, 2015.
[13] X. Ugarte-Pedrero, D. Balzarotti, I. Santos and P. G. Bringas,"SoK: Deep packer inspection: A longitudinal study of the complexity of run-time packers", IEEE Symposium on Security and Privacy, pp. 659-673, 2015.
[14] Ntcore Software "Cff-Explore"[online] Available: http://www.ntcore.com/exsuite.php.
[15] Wayne J. Radburn Software "Peview"[online] Available: http://wjradburn.com/software/PEview.zip