Secure Packer/Loader Communication

Encryption and Compression of Payloads

James Cogswell
Cyber Security Engineering
George Mason University
Fairfax, VA
jcogswe@gmu.edu

***Abstract*—The majority of malicious programs out there are packed. In order to combat such destructive malicious attacks, it is important to first understand the technology that the attackers use. Packers grant hackers the ability to not only execute their malware on a remote target, but also to avoid detection, prevent the ability to reverse engineer after the attack, and be able to reuse the malicious code in future attacks since reverse engineering will likely be ineffective. Two primary components of packers are encryption and compression and are both the primary areas of research for this project. The compression and encryption algorithms along with proper implementation are very important for the successful implementation of the Packer/Loader project. This paper provides explanations for using certain encryption and compression algorithms over others to help the readers understand why their importance and usefulness. This paper discusses other implementations and suggestions for packer programs as well as an evaluation of the assignment and its requirements.**

***Index Terms*—Packer, Loader, Encryption, Compression, Malware, Cyber Security, Linux, Windows, Anti-virus**

I. Overview of Senior Design Project

Lockheed Martin has developed a project that spans across four semesters, and two of which have already passed. The first two semesters were for the completion of Part I of the Packer/Loader project, and Part II is an expansion of the capabilities of its predecessor while retaining all previous functionality. The inherited programs are compatible with Windows OS and Lockheed Martin wishes to expand the compatibility to other mainstream operating systems, particularly Linux.

The requirements for the Packer/Loader programs are to encrypt and compress a portable executable (PE) file format binary executable and send the payload to a remote target machine which will contain a loader program. The packed payload must contain a stub that is not detectable by anti-virus software so that it is allowed to be unpacked by the loader and executed. The loader program is going to unpack the payload and run the malicious code exclusively in memory as a separate thread/process so that incident response forensics efforts become ineffective. Further, execution of programs must be configurable to allow the user to choose between an automatic run and a prompt during decryption and decompression.

Because there is a requirement to expand compatibility to Linux, flags will be added to the packer program to allow the user to choose which operating system they are targeting. It will then become necessary to pack the executable in such a way that it will be compatible and undetectable by the operating system for which it will be sent. The packer program will also allow the user to communicate with loader programs on several different target servers and will therefore act as a command-and-control (C2) server.

Encryption used in the packer will be password-based AES. Source code of the lossless compression format will be included so that the compression format is operating system agnostic. Therefore, the loader programs should theoretically be capable of decompressing the packed executable regardless of what operating system it runs on.

II. Research Area

Research which has been conducted for this paper will be largely based on secure communication between the packer and loader machines as well as encryption and compression to identify the most compatible, efficient, and secure options. The quality of the packer and loader programs as separate components does not matter if they cannot communicate securely and execute a payload without detection. If the packed executable is encrypted using a weak algorithm, then it can be reverse engineered by security researchers. It is possible that the researchers will obtain enough information to effectively detect and prevent the use of the packer and loader in the future, ultimately rendering it useless. If the compression used to pack the executable is incompatible with the target machine's operating system, then the loader will be unable to decompress and run the malicious payload at all. If the communication tunnel is unsecure or uses the wrong protocol for the wrapper, then the payload may not even reach the intended destination at all.

So, while many other aspects of this project can be a bit more technical, this aspect is heavily reliant on research of current technologies. There are countless ways to make the project work, but many elements of the process have been identified and prevention measures have been put into place by anti-virus software vendors to detect these methods. Therefore, it is imperative to seek out unique methods of carrying out such a process so that it is successful every time.

III. Introduction to Compression and Encryption Algorithms

A. Overview of Compression Formats

Execution of the packed PE needs to happen exclusively in memory, which is a factor in considering which compression format to choose. Packers that compress executables in memory on a remote machine are also referred to as software packers or runtime packers, among others. There are several different packers available to programmers that can achieve this goal but many of them are exclusive to one type of OS such as Windows or Linux. Therefore, the best compression choice depends on the requirements of the project and the intended target. Ideally, you would want to choose one that is compatible with more than one operating system.

Another method worth considering is using open-sourced compression, so you have the code to decompress the file upon execution regardless of the operating system on which it is used. Luckily, there is a software packer that is currently under development called the Ultimate Packer for Executables, or UPX. UPX is free, open source, written in C++, and is compatible with several different operating systems [1].

One of the main advantages of UPX over other runtime packers is the fact that the compression used is its superb data compression ratio (defined in Eq. 1) compared to others of its kind.

$$Compression\ Ratio = \frac{Uncompressed\ Size}{Compressed\ Size} \tag{1}$$

The lower the data compression ratio, the better. However, the more a file is compressed, the more resources it will require to decompress it and bring back the original code [2]. In the Lockheed Martin Packer/Loader project, the speed of decompression can be ignored for the most part since the differences in file sizes before and after compression are marginal considering the speed and efficiency of modern computers.

UPX uses the data compression algorithm called UCL which is also free, open source, and written in C++. UCL is a re-implementation of some of the NRV compression algorithms [3]. The UCL algorithm is written to be simple enough to write a decompressor in only a few hundred bytes of code. Additionally, UCL decompression is *very* fast and requires no additional memory.

There are two distinct types of compression algorithms: lossy and lossless. Decompression with lossy algorithms do not restore files to their original form but does reduce the size of the data. Lossy compression also compromises the quality of the data, typically known as irreversible compression, and is primarily used to compress images, audio, and videos and therefore the intention is to avoid using something like this in the project. Decompression with lossless algorithms will preserve the quality of the data and allows the file to be restored to its original form [4]. UCL is a lossless compression algorithm that is ideal for this type of work.

B. Overview of Encryption Algorithms

Along with compression, encryption is the other key component of a packer/loader program. There are countless encryption algorithms to choose from and all of them have their own strengths, weaknesses, and widely varying mathematical operations which dictate their efficiency and security. The two main types of encryption are symmetric and asymmetric encryption.

1) Asymmetric Encryption

Asymmetric encryption algorithms take some input and a unique encryption key for encryption and the decryption requires a different key. This is also known as public key encryption (PKE) and is very commonly used today. Every user has a public key that is known to others and a private key which is only known to them. To make the explanation easier to understand, the industry standard is to refer to the sender as Alice and the recipient as Bob. Typically, if Alice wishes to send Bob an encrypted message, C, she will use an encryption algorithm, E, to encrypt the message, M, by using Bob's public key PU_B as described in Eq. 2.

$$C = E(PU_B, M) \tag{2}$$

When Bob receives the message from Alice, he will decrypt the message using the decryption algorithm, D, using his own private key PR_B as described in Eq. 3.

$$\begin{aligned} M &= D(PR_B, C) \\ &= D[PR_B, E(PU_B, M)] \end{aligned} \tag{3}$$

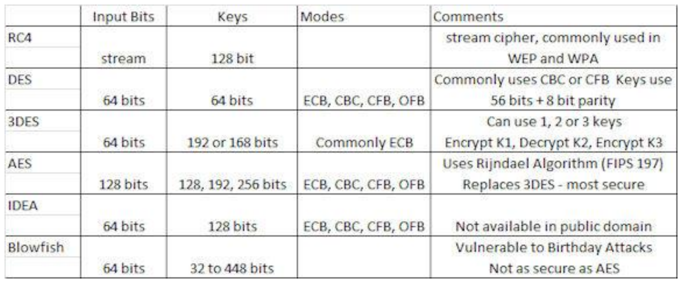
Even if an attacker were to sniff the message and obtain Bob's public key, it is computationally infeasible for them to use this information and to derive Bob's private key. The reason for this is largely out of scope for this paper but essentially relies on the use of prime numbers, of which there are an infinite number. Therefore, it is nearly impossible to derive the correct prime numbers used in the calculation while it is easy for Alice and Bob to use the keys they must encrypt and decrypt messages successfully every time.

2) Symmetric Encryption

Symmetric encryption algorithms use one key for encryption and decryption. Using the same references as before, if Alice intends to send Bob an encrypted message using a symmetric encryption algorithm, she will need to choose a single, unique secret key to encrypt the message and share that key with Bob so he can decrypt the message, as defined in Eq. 4.

$$C = E(K_{Secret}, M) \tag{4}$$

There is no shortage of symmetric encryption algorithms to choose from, but of course some of them are going to be more secure than others. Some of these algorithms require a password of equal length to the message being encrypted while others require certain fixed key lengths. Keys lengths equal to the size of the message itself are often considered very secure but require far too much storage space – since you are effectively doubling the size of the message when you encrypt it – for it to be practical. Key lengths themselves also do not necessarily determine the security of a symmetric encryption algorithm either. For example, a weaker encryption algorithm such as 3DES with a 192-bit key is not even remotely as secure as AES with a 256-bit key and the difference is in the details of the algorithms used in these cryptosystems. Figure 1 [5] details the differences between some of the more common symmetric encryption algorithms.



	Input Bits	Keys	Modes	Comments
RC4	stream	128 bit		stream cipher, commonly used in WEP and WPA
DES	64 bits	64 bits	ECB, CBC, CFB, OFB	Commonly uses CBC or CFB Keys use 56 bits + 8 bit parity
3DES	64 bits	192 or 168 bits	Commonly ECB	Can use 1, 2 or 3 keys Encrypt K1, Decrypt K2, Encrypt K3
AES	128 bits	128, 192, 256 bits	ECB, CBC, CFB, OFB	Uses Rijndael Algorithm (FIPS 197) Replaces 3DES - most secure
IDEA	64 bits	128 bits	ECB, CBC, CFB, OFB	Not available in public domain
Blowfish	64 bits	32 to 448 bits		Vulnerable to Birthday Attacks Not as secure as AES

Figure 1 - Symmetric Encryption Algorithms [5]

3) *Comparing Symmetric and Asymmetric Algorithms*

Asymmetric encryption and public key cryptosystems allow Alice and Bob to exchange keys securely so that they can communicate with each other over an unsecure network. Asymmetric encryption is more secure than symmetric encryption because the components that are computationally infeasible to break (i.e., the private keys) are never shared and the public keys cannot be effectively utilized by an attacker to derive the private key. Public-key cryptography is very widely used today, and its most common use cases are digital signatures, blockchain, and public-key infrastructure (PKI) [6].

Symmetric encryption is mostly used for encrypting data at rest and in banking when encrypting credit card numbers or other PII that is required for transactions [6]. For the most part, the only benefit symmetric encryption has over asymmetric encryption is speed. Symmetric encryption is faster because the keys are generally much smaller than the keys used in asymmetric encryption and because the encryption and decryption of asymmetric algorithms takes some time to complete. The reason symmetric encryption is not used as commonly as asymmetric encryption is primarily due to the fact that sharing the single key that is used for both encryption and decryption is its major security concern. Consider this: if Alice wanted to encrypt a message and send it to Bob using only a symmetric encryption algorithm, she would need to share that key with Bob so he can decrypt the message. How would Alice go about doing this? She could write it down on paper and hand it to him directly, but that is completely unreasonable since Alice and Bob could be thousands of miles away from each other. Well, Alice could just send Bob an email with the encryption key inside but since we are assuming this is the first time they are communicating over an unsecure network, then the secret key could be easily sniffed, and they would have no way of knowing whether the future exchanges of communication between them had been read or altered during transmission.

If one were to choose between symmetric and asymmetric encryption, what it ultimately comes down to is whether or not they will be the only one with the key or if someone else will need to have the key as well. If someone is encrypting their tax information on their computer, they would want to use a symmetric encryption algorithm. If they are sending a message to someone else over an unsecure network, then they would want to implement some kind of public key infrastructure so they can establish secret keys between themselves, all future recipients, and allow the others to establish secret keys with them as well.

4) *Combining Symmetric and Asymmetric Encryption*

As previously stated, asymmetric encryption is slower than symmetric and can result in slowdowns in the system since asymmetric encryption and decryption take time and resources to perform the mathematical calculations. Symmetric encryption would solve the speed problem, but then we face the security issue when attempting to share the secret key. Wouldn't it be nice if Alice and Bob could use asymmetric encryption to establish a secure connection between them, which they could then use to agree on a symmetric key for future communication? Perhaps a bit unsurprising, this exists and is the most common way for two remote hosts to communicate.

An example of combining symmetric and asymmetric encryption is the Diffie-Hellman Key Exchange (DHKE). A detailed description of DHKE is out of scope for this paper but is one of the most common ways to securely agree on a secret key between two parties over an unsecure network. DHKE has been improved since its inception to include more secure algorithms, such as by using Elliptic Curve Cryptography (ECC) to strengthen the algorithm while maintaining efficiency. This is known as Elliptic Curve Diffie-Hellman, or ECDH.

5) *Choosing the Right Encryption*

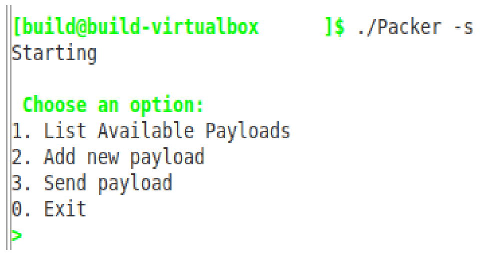
Lockheed Martin requires a password-based encryption algorithm, and the same password should be used for both encryption and decryption of the packed executable; therefore, only symmetric encryption will be used. Choosing a symmetric cryptosystem is ideal in this situation as compared to asymmetric or a combination since the users are the only ones that need to know the password and they will be the ones setting it.

The most secure, uncrackable symmetric encryption algorithm that exists today is the one-time pad (OTP). However, OTP requires a password that is equal in length compared to the file being encrypted which is unreasonable for a practical enterprise solution, or for any primary encryption method for any system for that matter since it effectively doubles the required storage space after encryption. The next best choice is to utilize the OpenSSL library to encrypt the payload with AES-256, which is the industry standard for symmetric encryption today.

IV. Current Status of the Technology

Part I of the Lockheed Martin Packer/Loader project was successful for Windows OS but not for Linux. The packer program ran locally on the Linux machine and is capable of compressing and encrypting the payload using AES and a user-provided password [7]. The loader is running on the remote Windows machine as a service awaiting data transmission from the machine where the packer program is running. Once the packed executable is sent to the remote Windows machine, the loader then decrypts and decompresses the payload and executes the code exclusively in RAM without ever touching disk [7].

The current packer program has a menu that the user can interact with where they will select an option to (1) see a list of available payloads, (2) add a new payload, (3) send the payload, and (4) exit, as shown in Figure 2 [7]**Error! Reference source not found.**. Any new payloads that the user wants to add to the packer would need to be placed into the directory that is defined in the configuration. If a payload is added, then it will be pulled from that directory and placed into the configuration-defined directory where other available payloads exist. When the user chooses to send the payload, it will then be packed (e.g., compressed and encrypted) and sent to the specified target system running the loader process [7].



```
[build@build-virtualbox      ]$ ./Packer -s
Starting

 Choose an option:
1. List Available Payloads
2. Add new payload
3. Send payload
0. Exit
>
```

Figure 2 – Packer Menu [7]

The protocol used for communication is secure file copy, or SCP, which uses the SSH protocol. However, since SCP is added functionality of Linux CLI command *cp*, it lacks support for non-Unix operating systems. To resolve this issue, third party software CYGWIN was used to be able to use SCP on the Windows machine [7].

There are many different packers in use today and they are typically authored by malware authors. The primary objective is to make it difficult to detect the malware and reverse engineer it. Malware authors can use compression that is well-known and widely available, or they can write some of their own. Ideally, ensuring that the anti-virus does not know how to unpack the packed executable is the primary goal. Runtime packers generally include a bit of unpacked code known as a "stub" which includes instructions for unpacking the malware once it reaches the target systems memory [8]. Additionally, the purpose of the packer is to hinder the anti-virus program's ability to use a heuristic approach to detecting the malware since the packer aims to break the signatures that the anti-virus will look for [8].

Research by Jon Oberheide and some of his colleagues at the University of Michigan wrote a Web-based application called PolyPack which can use ten packers and ten anti-malware engines to test the effectiveness of the packers against all ten anti-malware engines. A user just needs to submit a file that contains malware and all ten packers will pack the file. Next, the anti-malware engines scan the packed files. PolyPack reports on the effectiveness of the anti-malware engines and the user can identify which of the ten packers are most effective. One of the more common publicly accessible packers today, named Themida, scored highest over the other nine [9]. Themida was originally created for protecting commercial software from software crackers. Furthermore, Black Hat conducted a survey in 2006 that found that 92 percent of malware programs are packed [10]. So, it can be reasonably asserted that malware packers are still a very relevant topic among cyber defenders and hackers alike.

V. Technology Extension

Packer technology is expected to remain in the toolkit of malware authors. In the past, packers were used simply for compressing executables to save network bandwidth and space in the disks of personal computers which was much more important before than it is now [11]. Today's storage and network technologies completely eliminate the need for packers to compress regular executable files. Even though some companies still use packers to bundle executables with component files in their commercial software deployments [11], there is one primary use for packers today – allowing malware to avoid detection by the anti-virus on the victim's machine.

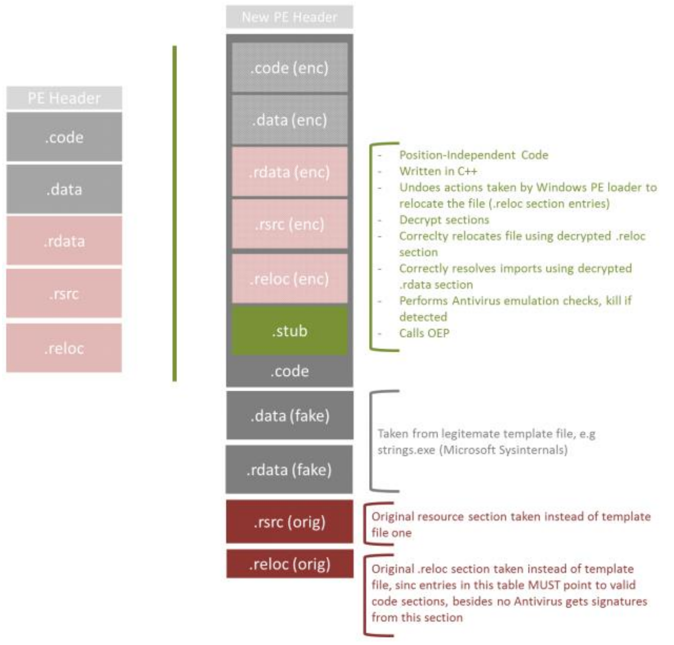

New PE Header
.code (enc)
.data (enc)
.rdata (enc)
.rsrc (enc)
.reloc (enc)
.stub
.code
.data (fake)
.rdata (fake)
.rsrc (orig)
.reloc (orig)
PE Header
.code
.data
.rdata
.rsrc
.reloc
- Position-Independent Code
- Written in C++
- Undoes actions taken by Windows PE loader to relocate the file (.reloc section entries)
- Decrypt sections
- Correclty relocates file using decrypted .reloc section
- Correctly resolves imports using decrypted .rdata section
- Performs Antivirus emulation checks, kill if detected
- Calls OEP
Taken from legitemate template file, e.g strings.exe (Microsoft Sysinternals)
Original resource section taken instead of template file one
Original .reloc section taken instead of template file, sinc entries in this table MUST point to valid code sections, besides no Antivirus gets signatures from this section

Figure 3 – New Packed PE [12]

Just as anti-virus companies attempt to find better ways to detect packed executables containing malware so they can protect their users from these types of attacks, malware authors find new ways to successfully execute the malware while avoiding detection. Arne Swinnen and Alaeddine Mesbahi suggest a new method for writing packers that more effectively avoid detection by anti-virus programs. At a very simplified level, this method requires the use of a legitimate 'template file' into which the malicious code will eventually be injected. The malicious code will be encrypted and added to the template as a resource (i.e., .rsrc) in the header [12]. The header sections must appear legitimate and some of them must be left unencrypted. In order to avoid breaking the template executable, the section data needs to be located in the expected areas of memory [12]. To do this, the packer calculates the location in memory of the template on disk, saves the data encrypted at the location on disk in the new packed executable. The stub is located after the encrypted data and this data will be the new first .text section of the packed PE file [12]. Figure 3 [12] demonstrates this approach.

Again, this is one of the proposed methods for improving packer programs and their ability to avoid detection. This is not a standardized process which is why it is so difficult to defend against. Any malware author can use an infinite number of ways to achieve their goal of executing malware on their target without detection.

VI. Practical Contribution of the Technology

Executing malicious code exclusively in memory on a remote target without detection, prevention, or the ability to reverse engineer it is an incredibly powerful tool in a red teamers arsenal. This is extremely important in the area of cyber security engineering because those in this profession often serve in offensive roles whether they work as penetration testers or even for the government as offensive cyber operators. Additionally, as many people know, the best offense is a good defense. If cyber security engineers have a strong understanding of how runtime packers are engineered and implemented to avoid detection, then they will be much more well-prepared to design new and improved detection methods to prevent future attacks.

Packers grant the ability to inject their malicious code into several different executables used in different attacks over time. Since the goal is not only to avoid detection but also to make it difficult or impossible to reverse engineer, then the malware may have a much longer lifespan since the signatures and exploits will not necessarily be able to be identified and added to the anti-virus databases. Being able to reuse the same malware for different attacks is also a massive benefit since it is not trivial to identify and exploit vulnerabilities in today's systems where security professionals with decades of experience have dedicated their entire careers to preventing attacks and exploitation.

VII. Conclusion

As previously stated, the knowledge of tools and techniques commonly used by black hat hackers is incredibly useful. The ethical considerations for improving the use of malicious software is often tricky but should not always be conflated with illegal activity or malicious intent. It is more important now than ever to ensure that those cyber defenders in the position to defend our infrastructure, assets, personal information, etc. are just as good – if not better – than those who want to carry out attacks against those whom they are responsible for protecting, whether they are public citizens, critical infrastructure, customers of an organization, or the organization itself.

The ultimate goal of packers and loaders are to successfully inject malicious code into an otherwise normal executable file, send it via secure tunnel to a remote machine, decrypt and decompress the malware without being detected, and run the malware exclusively in volatile memory. There is no limit to the number of ways this can be done which works in the benefit of this project just as much as it counts against it. Other members of the research project group are responsible for making it possible to allocate the memory space in the target system and automatically run the packed executable as a separate process once it reaches its destination.

The biggest challenge of this particular portion of the project is to ensure that not only are the best compression and encryption algorithms chosen, but also to use the optimal implementation of them. Static compilation of libraries can be extremely large which would most definitely trigger some alarms in the remote system. Therefore, once working packer and loader programs are developed, it will be imperative to spend a lot of time performing testing to ensure that the packed executables are light weight in order to avoid detection.

VIII. Technology Paper Evaluation

Writing and researching this paper has been instrumental in the preparation for this very challenging task. This project requires learning new skills that will be incredibly useful in the future. Thankfully, this research paper was assigned in the beginning of the senior design project where research is extremely important in facilitating preparedness before the development process begins.

There are a few requirements that should be changed in order to make the time spent researching topics like this more useful in practical applications. First, there are few peer-reviewed academic papers that are written about successful implementation and use of what is generally considered a black hat hacker's tool. Most of the peer-reviewed academic papers were about preventing and detecting malware packers, which is not helpful in the preparation of the role for this project. There are plenty of professional, well researched papers that are not peer-reviewed academic papers that are a whole lot more useful for background research. Also, this is not a standardized process as noted earlier in this writing, so a lot of the research is going to be comprised of individual pieces of information from websites containing manual pages, sites like stackoverflow and GitHub, and YouTube tutorials of someone else's implementation of the technology which can be used to make the whole project possible in a creative, unique implementation.

Another requirement that should be changed is the length requirement. When the length is decided by the writer, many students create higher quality work when they are able to decide how many pages are required to fully explain the required material. If a student decides that one and a half pages is sufficient and the grader feels like the research lacked a lot of content, then the student would lose points for lack of effort. However, if the project is fairly simple and a student can write a two-and-a-half-page paper that contains high-quality research and is written clearly enough for a layman to understand it, then taking a grade hit for that is unjustified. In summary, quality should be valued much higher than quantity.

Lastly, it isn't explicitly stated how important it is to stick to the specific definitions for the required sections and so it was assumed that it is important just to be safe, but not all projects can be easily researched when limited by these guidelines. It might be more effective both for the researcher and for the readers to give a few requirements outlining what the researcher should aim to include and leave these definitions fairly open to interpretation. For example, researching design alternatives is difficult to do with this topic because every packer must be created in a unique way in order to avoid detection, so it really has an infinite number of design alternatives, and none of which can be used by the research group.

Overall, this project is extremely important and useful. As a suggestion, just relax the restrictions and guidelines as much as possible to allow the students to use their own creativity for their research paper. Additionally, it would be enormously beneficial to make this paper a group project. All students are working toward the same thing so this would encourage even more collaboration, keep everyone on the same page in terms of technologies introduction, technology extension, practical contribution, etc. and would result in a longer, more extensive, and heavily researched paper while requiring less overall working hours for each of the students. The group working on this specific project only made first contact with the sponsor in week six or seven, so this project has required a very large chunk of time to complete, and it is not clear that all of the effort which was required was completely necessary, although it is obvious that there is tremendous value in this project overall.

Please let it be known that these suggestions and criticisms are aimed directly at the specific requirements. ABET requirements are what they are, so perhaps these suggestions are meant more for them since they cannot be changed within this course. Overall, the assignment was extremely beneficial to the students, even the way it has been assigned.

IX. Acknowledgment

Research was completed for the Lockheed Martin Packer/Loader project part two which will improve the current project, add new capabilities, and expand compatibility. The sponsor is Allison Elfring, Sr. Software Engineer with Lockheed Martin and the group mentor is Dr. Mohamed Gebril.

X. References

[1] M. Oberhumer, “UPX: the Ultimate Packer for eXecutables - Homepage,” GitHub, Jan. 22, 2020. https://upx.github.io/ (accessed Oct. 25, 2020).

[2] M. Hawthorne, “What is Compression Ratio? definition & meaning,” Technipages, Aug. 16, 2019. https://www.technipages.com/definition/compression-ratio (accessed Oct. 26, 2020).

[3] M. Oberhumer, “UCL data compression library,” oberhumer.com, Jul. 20, 2004. http://www.oberhumer.com/opensource/ucl/ (accessed Oct. 25, 2020).

[4] MKS075, “Difference between Lossy Compression and Lossless Compression,” GeeksforGeeks, May 22, 2019. https://www.geeksforgeeks.org/difference-between-lossy-compression-and-lossless-compression/ (accessed Oct. 26, 2020).

[5] L. Pontarelli, “CCIE Study Notes: Version 4.1 - 2.0 Symmetric Encryption,” Blue Wolf Spirit, Jan. 31, 2017. http://www.bluewolfspirit.com/notes/security-written/version_41_20_symmetric_encryp.html (accessed Oct. 26, 2020).

[6] R. Yackel, “When to Use Symmetric vs. Asymmetric Encryption | Keyfactor - Security Boulevard,” Security Boulevard, Jun. 17, 2020. https://securityboulevard.com/2020/06/when-to-use-symmetric-vs-asymmetric-encryption-keyfactor/ (accessed Oct. 26, 2020).

[7] M. Palmer, C. Bai, H. Rowlette, A. Chapin, and A. Herrera, “Windows Packer/Loader,” George Mason University, 2020. Accessed: Oct. 10, 2020.

[8] R. Abrams, “An Introduction to Packers,” WeLiveSecurity, Oct. 28, 2008. https://www.welivesecurity.com/2008/10/27/an-introduction-to-packers/ (accessed Oct. 24, 2020).

[9] J. Oberheide, M. Bailey, and F. Jahanian. PolyPack: An Automated Online Packing Service for Optimal Antivirus Evasion. In Proceedings of the 3rd USENIX conference on Offensive technologies, August 2009.

[10] T. Brosch and M. Morgenstern. “Runtime packers: The hidden problem?” https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Morgenstern.pdf, 2006.

[11] P. O’Kane, S. Sezer, and K. McLaughlin, “Obfuscation: The Hidden Malware,” IEEE Security Privacy, vol. 9, no. 5, pp. 41–47, Sep. 2011, doi: 10.1109/MSP.2011.98.

[12] A. Swinnen and A. Mesbahi, “One packer to rule them all: Empirical identification, comparison and circumvention of current antivirus detection techniques.,” BlackHat USA, 2014. Available online: https://www.blackhat.com/docs/us-14/materials/us-14-Mesbahi-One-Packer-To-Rule-Them-All-WP.pdf.