Practical Application of open Source Frameworks to Achieve Anti-Virus Avoidance

Ignus Swart CSIR,
Pretoria, South Africa
1

Abstract

A common aim of malware creators is to have the ability to spread their software undetected through various networks until the required goal is completed. In response to this, anti-virus vendors have implemented various strategies to detect viruses as they attempt to execute and propagate from one target to the next. Some of the anti-virus vendors claim to achieve impressive success rates as high as 98.7% that indicates the problem of spreading viruses and malware is well taken care of. Yet, despite the impressive detection rates, a proliferation of open source tools, frameworks and utilities are being introduced that claim to have the ability to avoid anti-virus detection. As an example, the very popular Metasploit framework has several encoders available that can alter the virus signature in such a way that it will avoid the anti-virus engine and allow the malicious code to be executed. This approach has been implemented and simplified in the Social Engineering Toolkit (SET) as part of a menu driven approach that is accessible to people with a relatively low skill level. The SET framework, implemented in Metasploit, is only one such framework and several more specialised open source tools exist, that does not only focus on encoding but on other common anti-virus avoidance techniques such as binary editing, packing and encryption. Open source packages such as UPX compress the data in the selected virus executable to such an extent that it will most likely completely circumvent the anti-virus and similarly so for a program that is encrypted with a common encryption product such as TrueCrypt. Should the anti-virus still detect the offending executable after either packing or encryption a combination of the two applications might yield superior results.

The aim of this paper is to experiment on a common executable that is classified as malware e.g. the meterpreter module of Metasploit, and make use of the various open source frameworks and utilities to document the techniques and success rate of anti-virus avoidance. By presenting the results of this research, it will contribute to the understanding of security personnel / researchers on what can be achieved with open source frameworks and how to better protect against the virus threat.

Paper Relevance: While great strides have been made in anti virus detection it is not nearly perfect and many open source tools can be used to effectively hide even old executables flagged as malicious. The question is how difficult is it to use these tools and how effective are they?

Keywords: Antivirus avoidance, open source packer, protector, binder

1. Introduction

The current worldwide strategy for dealing with malicious software has not really changed much in the past five years and significant challenges still remain. (McAfee 2011) in their third quarter threat report states that well over 70 million malware types has been identified and the complexity of the malware is increasing. Malware detection happens if a sufficient number of machines on the internet have been infected and a sample of the malware reached one of the antivirus vendors. Detailed analysis is performed and an identification signature is determined for the new malware that is then added to the vendors antivirus database. The new detection signature reaches the clients when a software update is performed and the malware infection starts to recede until it is finally almost negligibly encountered in the wild. After this, the malware creators restart the creation process to maintain the foothold they have over currently infected machines that does not yet have the newly created Anti Virus (AV) signatures and to maintain the ability to infect new machines.

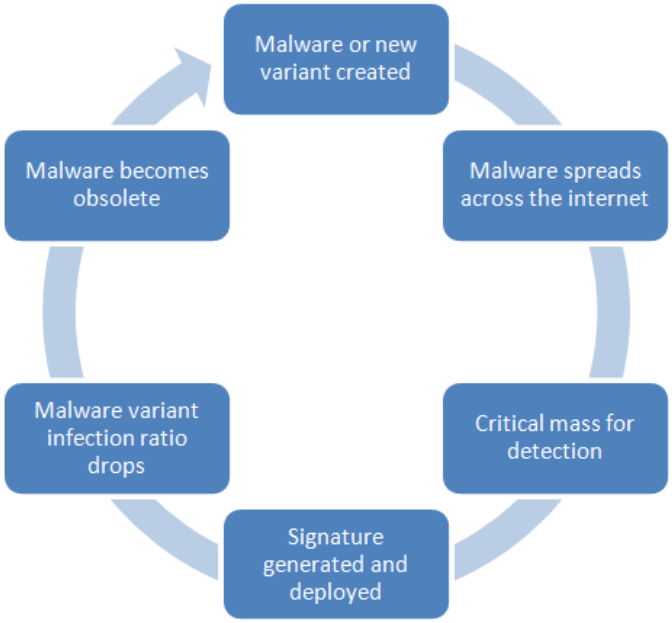

Figure 1: Current malware detection and perpetuation cycle

Research conducted by (Kanich et al. 2011) found that although on average 95.5% of all PC users had an antivirus installed on their system, only 42.9% on average has kept the installation up to date with the latest virus definitions. This leaves a huge potential for malware perpetuation even though antivirus companies has already found an effective detection mechanism. In a report by (Verizon 2011) they stated that only 1% of victims of a successful attack that they investigated was notified by their antivirus of the attack on their system.

Table 1: Antivirus installation and updated percentage breakdown

	A.V Installed (%)	Up to date (%)
US	98.7	22.8
India	92.7	68.7
Other	95.2	37.3
Average	95.5	42.9

It should be noted that although the malware detection cycle has remained the same, antivirus vendors has made significant inroads to simplify and speed up the submission of potential new malware. Additionally tremendous effort has been put in place to automate the detection and classification of malware by automated reverse engineering since the sheer number and complexity of malware would make manual inspection nearly impossible. (Debrey & Patel 2010) in a publication estimate that between 79%-92% of all malware employs packers to increase the difficulty of reverse engineering the malware binary, effectively eliminating the option of only employing humans in the analysis process.

This paper will examine several open source antivirus avoidance tools and frameworks that require little or no programming experience to gauge their effectiveness at avoiding antivirus detection. Section two will describe techniques that antivirus vendors use to detect malware while section three will describe methods used by malware creators to hide malware. In section four the various selected tools and frameworks are described along with the experimentation procedure with the results and conclusion presented in section five and six respectfully.

2. Antivirus methodologies used for detection

According to (Daoud et al. 2008) anti-virus detection strategies fall predominantly into two methods namely static and dynamic analysis. The main distinction between static and dynamic analysis are that static detection does not actively execute the code but employs several scanning techniques to detect a potential threat. According to (Brand 2011) some examples of static analysis are control flow graphs, dataflow analysis, string extraction and target architecture determination. Dynamic analysis on the other hand executes the code and monitors it for known suspicious behaviour such as opening an executable file in read and write mode, or attempting to write to the boot sector. Both methods have significant problems as of late and so far no easy solution has been found.

Static analysis can only be effective if the suspicious binary can be matched to a already identified signature and with the introduction of tools such as packers and cryptors, this analysis can be severely hampered. Packers are already widely used as previously noted and (Guo et al. 2008) explains that currently not all packers can be detected and unpacked by antivirus vendors. Packers such as Ultimate Packer for eXecutables (UPX) is already widely know and can be easily unpacked, but in the case of proprietary tools such as Themida it might take up to six months to write a unpacker program that will work effectively.

Dynamic analysis allows the executable to execute in a virtual environment where the full calling sequence, file requests and input output operations are mapped. In theory it should present a clear picture of what the malware's intended purpose is regardless of the encryption or packing features used to defeat static analysis. The significant problem with dynamic analysis according to (Nataraj et al. 2011) however, is the time that is required for the dynamic analysis and the fact that avoidance techniques such as virtual machine detection and debugger attachment detection is also available to malware creators. This will not only prolong the dynamic analysis to an inconvenience, but as the author mentioned could possibly take 254 year of machine time to test against the full 2010 Symantec malware library. Various researchers are attempting to address the shortcomings of the antivirus engine by either employing multi core cloud processing or moving to other architectures perhaps better suited for the task. (Vasiliadis & Ioannidis 2010) presented a novel idea to achieve an exponential increase in performance capability by moving the antivirus engine's processing to the graphics card of the computer. In related work (da Silva dos Santos Silva 2009) proposed a distributed antivirus that will spread the processing of dynamic scanning across various cpu's spread all across the local area network of the organization. Both solutions are attempts to increase the available computing power to the antivirus while still maintaining a enjoyable user experience and

while it might not solve all the problems, it could give dynamic analysis a greater chance of identifying malware.

3. Antivirus avoidance methodologies

As discussed previously almost all anti-virus engines rely on a type of signature in the existing executable to accurately identify a potential malicious threat. Due to this relationship between the fingerprint and identification several methods have become available to alter or even hide the offending fingerprint. (Brand 2011) mentions that well over 80 different antivirus avoidance techniques exist but no one antivirus tool exist to negate them all.

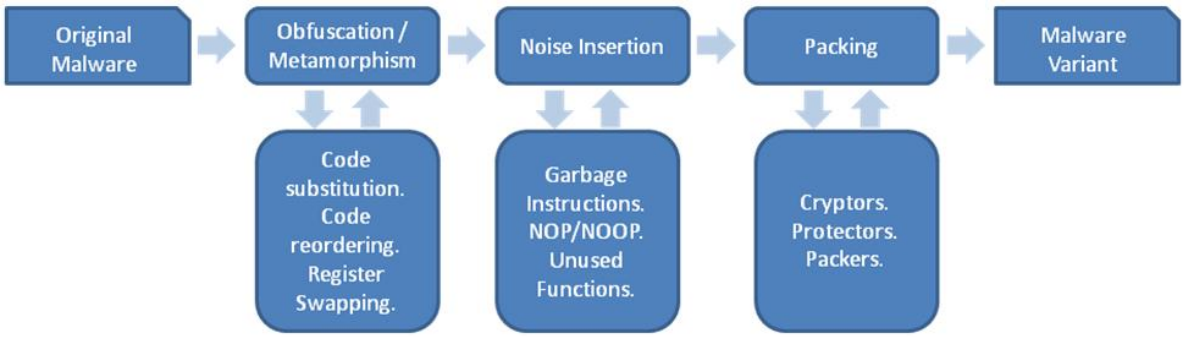

Figure 2: Malware antivirus avoidance process

(Ollman 2009) states that Crypters, Protectors, Packers, Binders are some of the methods malware creators use to avoid detection. (Patel 2011) adds obfuscation, register swopping, junk insertion, subroutine transposition as extras wile (Tzermias et al. 2011) adds encryption, UTF-encoding and shellcode encoding. (Davis 2011) brings polymorphism and metamorphism to the collection that will allow code to generate dynamic signatures every time it is executed. The various techniques can be combined and re-combined with other techniques to produce a unique signature of the malware that no antivirus vendor has seen yet. A summary of some of the more prominent techniques are discussed below:

- **Crypters** - Are described as programs that add encryption to the code base of the malicious program to defeat the static analysis of the antivirus and prevent the detection of suspicious code markers by disassemblers such as IDA Pro. This is useful for getting the virus transmitted past network security boundaries but problematic on the target machine since the executable would still need to be decrypted before running.

- **Protectors** - Are programs that specifically alter the code in such a way to remove any information from the code that could be useful in debugging attempts. Should a debugging attempt be detected the protector could even have the ability to execute a different set of instructions to hide the true intentions of the malicious application or if a sandbox is detected, attempt to break out of the sandbox.

- **Packers** - According to the author are designed to reduce the size of the target executable to facilitate proliferation of the executable but it has the added benefit of reducing the surface area available to the antivirus vendors. With the addition of polymorphic encoding the packer not only reduces the size of the malicious executable but also constantly alters the signature of the executable. (Guo et al. 2008) explains that this particular technique further increases

the size of the antivirus signature if the vendor is unable to unpack the malicious code decreasing overall system performance.

- **Binders** - Are described as tools that allow the embedding of malware content into files commonly searched for, so effectively creating one executable that contains multiple executables. An example of this is notepad.exe, office applications or even a pirated version of Windows that will perform the requested action but additionally install malware that was combined in the executable.

- **Noise insertion** – Refers to the ability to add various types of instructions to the existing code base that effectively does nothing to alter the application's behaviour but changes its code signature to something completely different. Examples of noise insertion in a application could be a NOP insertion or a function call with no real meaning such as Sleep(0), "mov eax, eax" or "mul 0x1, ebx".

When comparing the efficiency of the various common antivirus avoidance techniques (Hu 2011) states that packers, cryptors and protectors are some of the best avoidance techniques currently available. This efficiency is due to the fact that it prevents reverse engineering to such an extent that both static and dynamic analysis of the code becomes relatively useless. A summary of the effectiveness of the various types of avoidance techniques can be found in Table 2 that was compiled by (Ollman 2009).

Table 2: Effectiveness of antivirus avoidance techniques

File Checksums	RegEx Signatures	File Heuristics	Behavioura
	✓✓✓	✓✓✓	✓✓
	✓✓✓	✓✓✓	✓
	✓✓✓	✓✓	✓
	✓✓✓	✓	
	✓✓✓	✓✓✓	✓✓✓

✓✓✓	✓✓✓	✓✓✓
✓✓✓	✓✓✓	✓✓✓
✓✓✓	✓✓	✓

4. Open source tools / frameworks to evaluate

Open source examples were chosen to allow full examination of the results and the experiment was divided into two categories. One part where only the malware executable is used and the other where the source of the malware is also available for processing. A further consideration was that the majority of malware is designed to establish a connection from an infected computer to a remote host and the software evaluated would need to perform a similar task. As such, to replicate the compiled executable scenario would require an open source product that establishes remote connections, and is normally flagged as malware by antivirus vendors. Netcat fits the description perfectly as it is both open source and constantly flagged by antivirus vendors as either malicious software or a hacking tool. For the second part of the experiment Metasploit was chosen since it not only has the ability to create remote connections but all the source is available and multiple encoders are also available. The meterpreter module is also constantly flagged as either malicious software, Swrort.A, EPACK.Gen2, Rozena.A or Swort-C malware variant and thus fits the specified criteria. The baseline meterpreter module is detected 62% of the time making it an ideal candidate to simulate malware antivirus avoidance.

Not all antivirus avoidance techniques are always applicable in all situations and it is worth mentioning the differences between the various types. If the source code of a piece of malware is available all the types of antivirus avoidance techniques are available. In the event that only the compiled binary is available, all is not lost as a range of the antivirus avoidance techniques are still applicable but it does somewhat limit the options. Further explanation follows in the tool discussion below.

- **Ultimate Packer for eXecutables** – (UPX 2011) has been available since 1996 is still one of the leading open source software packers. An updated version 3.08 has been released on 12 December 2011 and adds support for BSD systems.

- **PEScrambler** – According to (Nick 2011) is a tool that will obfuscate Win32 binaries by relocating portions of code and inserting anti-disassembly protection. Development has stagnated as of late and the project seems to be at a standstill

- **Social Engineering Toolkit (SET)** – Is an exceptional penetration testing framework that combines MSFEncode and MSFPayload with automation to such an extent that creating software executables, literally becomes menu driven. Additional parts of the Metasploit framework are also incorporated into the SET framework such as Shellcodeexec.exe that is a lightweight executable with the ability to interpret shellcode and execute it in memory.

- **MSFVenom** – Compiled into the Metasploit framework the MSFVenom executable is the combination of MSFPayload and MSFEncode into one executable. The advantages are faster encoding times and a less complicated command line environment. Several encoders are present in the Metasploit framework, ranging from polymorphic encoders to simplistic XOR encoders.

- **Shaddam Source** – An open source protector that highlights the potential pitfalls available when making use of open source and freeware projects. The project does have some merit but hidden inside the code is various ways to not only encode and hide the selected executable but also to compromise the attackers computer.

- **Yoda** – A protector project that had its last update in 2006 and while that may sound old, it still has the ability to be problematic for certain anti-virus vendors. Full source is included in the download for anyone to investigate the intricacies of how the protector functions.

5. Experiment Results when uploaded to VirusTotal

The results of the experimentation with open source software on the Netcat executable is depicted in Figure 3. The baseline measurement is when the Netcat executable is uploaded to Virustotal without any modifications and resulted in a 62% detection rate from the 43 participating virus vendors. Packing the executable with UPX made absolutely no difference and the detection rate remained at a constant 62%. Making use of the PEScrambler tool resulted in a 44% detection rate, an improvement of 18% or 8 less antivirus vendors that will flag Netcat as malicious software. Shaddam's protector managed a best of 41% detection rate but included a backdoor that wanted to infect the testing machine. Yoda's protector achieved a 55% detection rate and was outperformed by manual editing from a five year old tutorial by (Team 5150, 2006) on how to hide Netcat from antivirus software. The manual editing involved changing four INT 3 instructions to NOP's to remove the antivirus signature from the file and was surprisingly effective to this day.

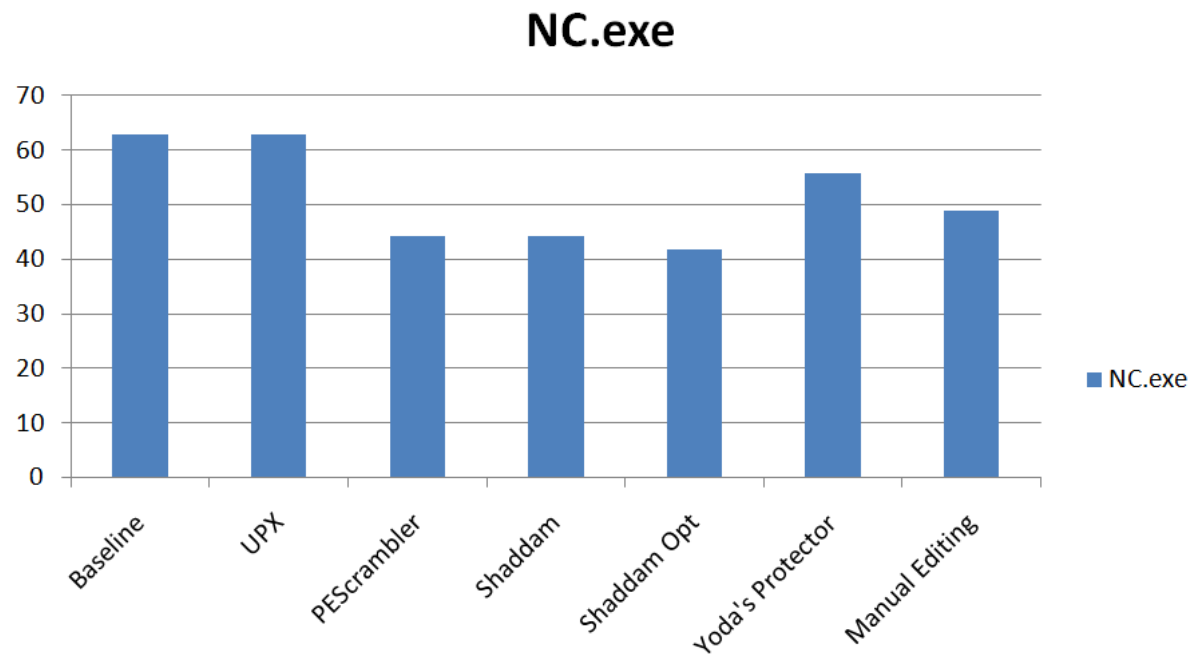

Figure 3: NC.exe VirusTotal Detection Rate

The Social Engineering Toolkit combined in the Metasploit Framework performed most admirably and achieved impressive results. The first option was to establish a meterpreter reverse connection by embedding it in a Win32 template executable and it was detected by 55% of the antivirus vendors participating on Virustotal. Secondly the SET framework option to encode a meterpreter module to Shellcode was employed and it achieved an extraordinarily low detection rate of only 2%. The problem with this type of antivirus avoidance is that it is more or less limited to being run from either a CD/DVD or USB drive and that there are two parts to the executable namely the Shellcode interpreter

and the Shellcode. One of the options that defeated all antivirus detection was the 64 bit Windows Shell Reverse connection and it was as simple to create as starting the SET framework, selecting option 4 (Create a payload and listener) and then selecting option 6 (Windows Shell Reverse TCP X64).

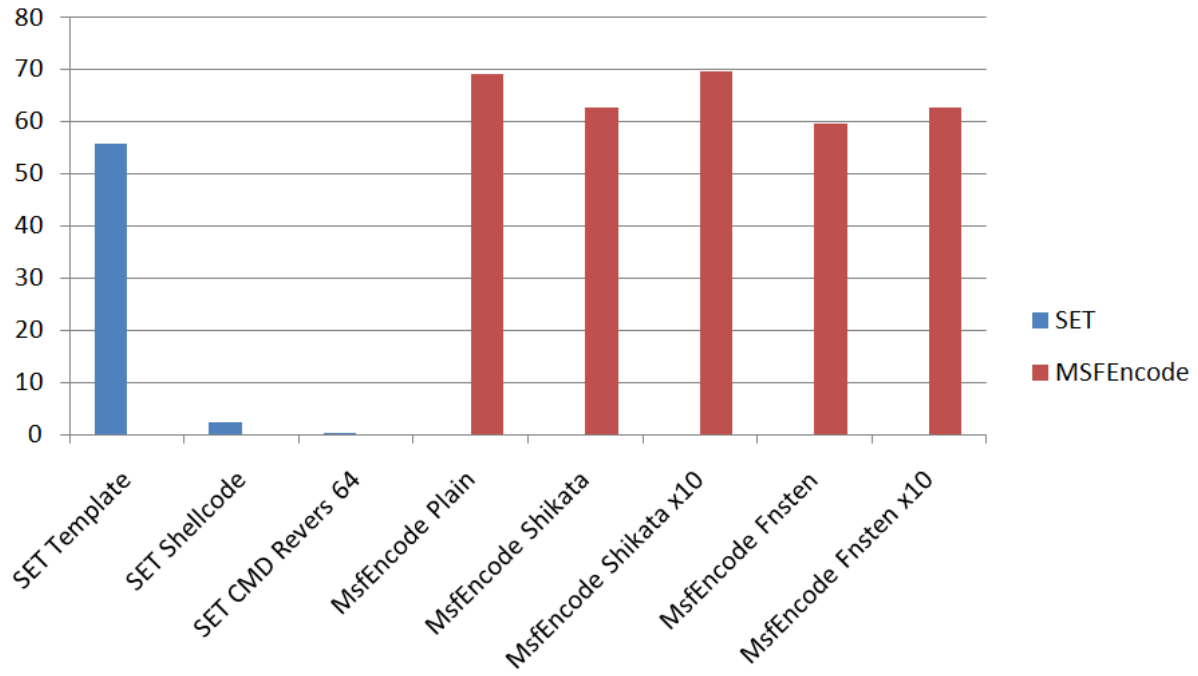

Figure 4: Social Engineering Toolkit Detection Rate

For the MSFEncode example a normal meterpreter shell was encoded with the various encoding schemes available in the Metasploit framework. The results are interesting due to the fact that with no encoding a 69% detection rate was obtained that dropped to 62% with a polymorphic encoder encoded once. If the encoder was run ten times, the detection ratio climbed back to the same level as when no encoding was applied. The results were similar for the XOR encoder where the once off encoding led to a lower detection score than when the encoder was set to an iteration of ten times. As a side note the polymorphic encoder did not perform significantly better than the normal XOR encoder but a possible reason for this could be the small size of the executable.

It should be noted that no manual editing was performed on the generated executables and if this was employed, a further drop in the detection rate could have been achieved. As an additional test a custom python module was written to open a socket on a target machine and connect to a remote machine. The lines of code required to perform this task in a Python module is less than 20 lines and similarly a custom receiver on the attacker machine is also coded in less than 20 lines. Once deployed on the target machine, no antivirus alerted the user to the newly created connection and Virustotal also gives the executable a clean bill of health with a 0% detection rate. With this very simple network communication enabled, an attacker can already perform various functions such as information gathering, Shellcode deployment and privilege escalation.

6. Conclusion

From the results obtained it was clear that antivirus software is effective against a multitude of threats but custom malware will in all likelihood not be detected. This could lead one to believe that malware created to target the whole internet indiscriminately will eventually be uploaded to a antivirus vendor and classified as malware to be detected from there on out. However malware that target specific

organisations will have a much lower chance of being detected and thus a much higher chance of performing its malicious intentions for a longer period. Assessing the skill level of a malware creator is quite problematic, but from the examples investigated it is clear that an advanced degree is not required, especially if the tools such as the SET and Metasploit frameworks are considered. The software used are all opens source applications, and the code examples are freely available from the internet for anyone with a modicum of programming knowledge to follow. The conclusion is thus that with open source tools, it is quite possible to create malware that will completely avoid antivirus detection or at least avoid the antivirus solution deployed by the selected target organisation. If an attacker is determined enough to keep on trying it is all but a certainty that eventually antivirus avoidance will be achieved and organisations should keep this in mind when planning their security.

7. Bibliography

Brand, M. (2011) "*Analysis of avoidance techniques of malicious software*", [Online], Available at: http://www.ruxcon.org.au/2011-talks/analysis-avoidance-techniques-of-malicious-software/ (Accessed on 22/01/2012).

da Silva dos Santos Silva, C. (2009) "*RAVE - Final Thesis Report*", [Online], Available at: http://docs.di.fc.ul.pt/jspui/bitstream/10455/3289/1/RAVE_-_Final_Thesis_Report_-_Carlos_Silva.pdf (Accessed on 14/12/2011).

Daoud, E.A., Jebril, I.H. & Zaqaibeh, B. (2008) "*Computer Virus Strategies and Detection Methods*", International journal of open problems in computer science and mathematics, Vol 1, No. 2, pp.29-36.

Davis, M. (2011) "*Getting Dirty with GCC*", [Online], Available at: http://www.ruxcon.org.au/assets/Presentations/2011-2/ruxcon2011.pdf (Accessed on 06/01/2012).

Debrey, S. & Patel, J. (2010) Reverse Engineering Self-Modifying Code: Unpacker Extraction. In *17th Working Conference on Reverse Engineering*. Beverly, MA, 2010. IEEE, pp 131-140.

Guo, F., Ferrie, P. & Chiueh, T. (2008) A Study of the Packer Problem and Its Solutions. In *Proceedings of the 11th international symposium of recent Advances in Intrusion Detection*. Massachusetts, 2008. Springer-Verlag Berlin, pp 98-115.

Hu, X. (2011) *"Large-Scale malware Analysis, Detection, and Signature Generation",* [Online], Available at: http://kabru.eecs.umich.edu/papers/thesis/thesis_0902_01PM.pdf (Accessed on 12/12/2011).

Kanich, C., Checkoway, S. & Mowery, K. (2011) Putting out a HIT: Crowdsourcing Malware Installs. In *Proceedings of the 5th USENIX Workshop on Offensive Technologies*. San Fransico, 2011, pp 6-13.

McAfee. (2011) "*McAfee Quaterly Threat Report*", [Online], Available at: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q3-2011.pdf (Accessed on 16/01/2012).

Nataraj, L., Yegneswaran, V., Porras, P. & Zhang, J. (2011) A comparitive assessment of malware classification using binary texture analysis and dynamic analysis. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. Chicago, 2011. ACM New York, pp 21-30.

Nick, H. (2011) "*PEScrambler",* [Online], Available at: http://code.google.com/p/pescrambler/source/checkout (Accessed on 28/12/2011).

Ollman, G. (2009) *"Damballa Serial Evasion Tactics*", [Online], Available at: http://www.damballa.com/downloads/r_pubs/WP_SerialVariantEvasionTactics.pdf (Accessed on 11/12/2011).

Patel, M. (2011) "*Similarity Tests for Metamorphic Virus Detection*", [Online], Available at: http://scholarworks.sjsu.edu/etd_projects/175 (Accessed on 18/01/2012).

Team 5150. (2006) *"Taking back netcat*", [Online], Available at: http://team5150.com/~random/apps/netcat/Taking_Back_Netcat.pdf (Accessed on 11/12/2011).

Tzermias, T., Sykiotakis, G., Polychronakis, M. & Markatos, E.P. (2011) Combining static and dynamic analysis for the detection of malicious document. In *Proceedings of the European Workshop on System Security (EuroSec)*. Salzburg, Austria, 2011, pp 1-4.

UPX. (2011) *"Ultimate Packer for eXecutables",* [Online], Available at: http://upx.sourceforge.net/ (Accessed on 16/01/2012).

Vasiliadis, G. & Ioannidis, S. (2010) GrAVity: A Massively Parallel Antivirus Engine. In *Proceedings of the 13th International Symposium*. Ottowa, Canada, 2010. RAID, pp 79-96.

Verizon. (2011) "*Verizon Data Breach Investigations*". [Online] Available at: http://www.verizonbusiness.com/resources/reports/rp_data-breach-investigations-report-2011_en_xg.pdf (Accessed on 12/01/2012).