

Cybercrime Malware Assignment 1

Malware Obfuscation



MARCH 19, 2019

ADAM MILLER

Malware, which can be any type of malicious code [4], and detectors or anti-virus tools are in a continual arms race. With malware developing ever more advanced and sophisticated obfuscation techniques and detectors researching more complex detection mechanisms to identify the malware. This arms race has been ongoing for decades and traditionally detectors have relied on large databases of known signatures [3], or hashes, of the malware. However, malware often uses a range of techniques to change this signature from one infection (or generation) to the next. This makes it more challenging for detectors and malware analysts to identify the behaviour of malware in a timely manner, or even to identify the code as being malicious at all. We are going to look at some obfuscation techniques and describe how detection can be carried out for those techniques. Some methods can be highly complex, such as the malware being interwoven into a targeted host file, while others are simpler like changing the packer used. In all cases it makes detection more time consuming and resource intensive to isolate and identify the malware signature. To compound the woes of signature-based detector's, this method is not effective against new malware using unknown vulnerabilities (think zero days). This relentless development of new malware variants has made signature-based detection less effective. However, with the reduction in the effectiveness of signatures, behavioural, heuristic and sandbox-based detection have been developed. To understand how all this works we first need to understand how the different types of malware and how they obfuscate themselves.

4 Categories of malware

Due to the diverse methods malware uses to obfuscate itself, it is necessary to categorize them and there are four main types of obfuscated malware; Encrypted, Oligomorphic, Polymorphic and Metamorphic. Let's go through these now.

Encrypted malware

There are two types of encrypted malware, malware using encryption and malware using packers. With encryption the malware uses encryption to conceal itself from detection. This type of malware is usually composed of the decryptor and its encrypted main body.[1] This method is effective for two reasons, firstly by encrypting the malicious code it executes the malware cannot identify the payloads signature; secondly by changing the encryption key it uses the signature of the encrypted code itself changes.[4] To ensure the malware remains obfuscated throughout multiple generations, and in order to avoid its encrypted signature from being static - and thus identifiable by signature based detectors; Every time the malware is run it can generate and uses a new encryption key to keep its signature unique. For best effect this new key should be generated in a random and unpredictable manner. However, the decryptor portion of the malware cannot be encrypted as it needs to be executed and it retains a static signature. Due to this detection methods that focus on the malware decryptor signature are usually successful. [1]

Packers [3]

Packers are usually legitimate tools to decrease the size of an application while it is stored or transported, like compressing documents but in a way that still lets the application be executed. Even small changes to the underlying application can drastically change the signature of the resulting packed executable. There are multiple packing applications and research on which packers are most effective for evading detectors. One example of this is *"Jon Oberheide and his colleagues at the University of Michigan wrote PolyPack, a Web-based application that supports 10 packers and 10 malware detection engines (like virus total)"*[3]. This research and similar applications can help malware authors identify which packer would be best for their malware to avoid detection.

One-way packed malware can be detected by having a database of all possible signatures a packed malware can produce. This is very inefficient, and a better option is to use what is called "Entropy Analysis"[3] to identify the packed malware. This can detect packed files but cannot detect the packer used, which can cause difficulties for deeper analysis. PHAD, PE-Probe and MRC all use Entropy analysis. Without unpacking the file, it can be difficult to know if it's malware or a legitimate application, especially as we need to identify the right packer to unpack the file. This can be difficult, packers are commonly used to spread malware.

Oligomorphic and Polymorphic

Malware that can mutate their decryptor's from one generation to the next have been designed to fix the shortcomings of purely encrypted malware. The first example of this was the oligomorphic malware which was able to change its decryptor. [1] However oligomorphic malware was initially very limited in the maximum number of decryptor versions it could produce, allowing the signatures of all possibilities to eventually be calculated. This catalogue of signatures allowed detectors to identify all variants of the malware.

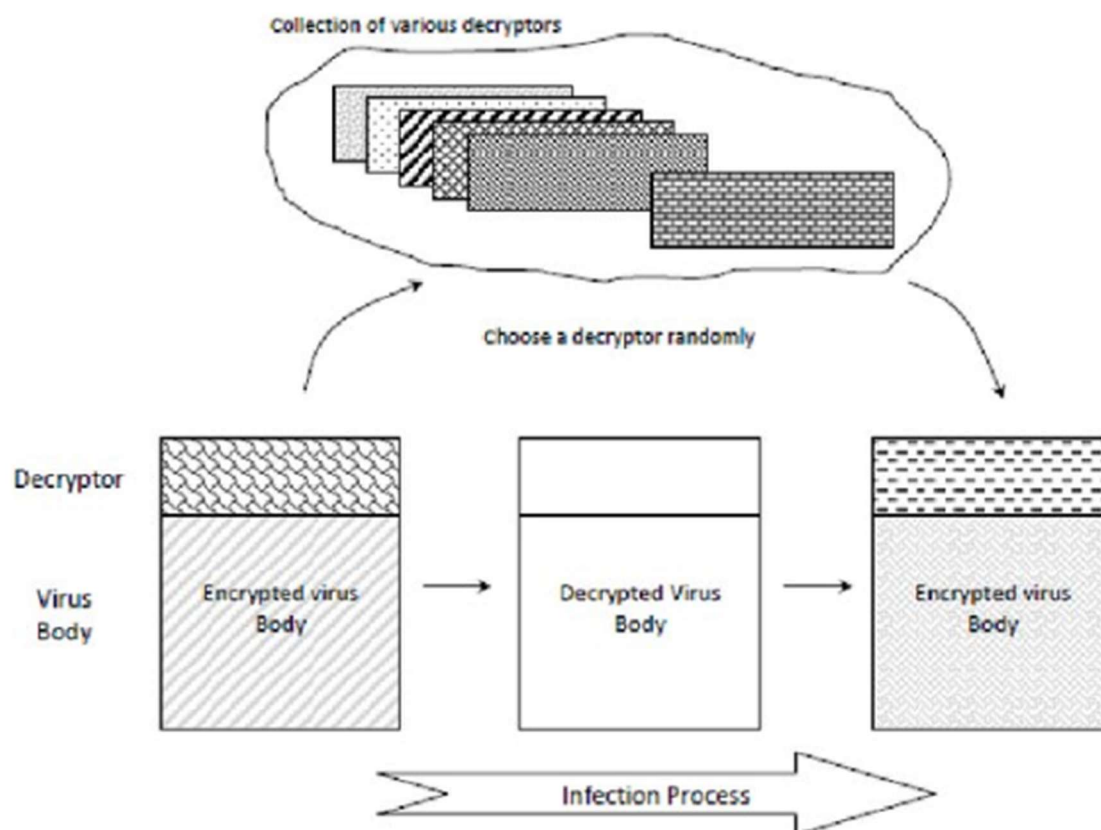


Figure 1

Polymorphic malware is an encryption method that mutates its static binary code. [3] It was developed to attempt to take the ideas of Oligomorphic malware and further improve them by being able to generate an incalculable number of potential decryptor variants so that no single signature sequence will match all possible variants of this malware. It achieves this by using several very cool obfuscation methods we will talk about later including dead code insertion, register reassignment, Code Transposition and Instruction

Substitution [2]. Each time the code is run it mutates itself by using a different key. To make things even more challenging for malware analysts, there are many tools out there such as The Mutation engine that automates the process; allowing regular, non-obfuscated malware to be converted into polymorphic malware.

To detect these types of malware the detectors make use of tools like sandboxing. With sandboxing the detector executes the malware in a secure emulator. We then execute the malware and wait for its constant body (the payload) to be decrypted in RAM after execution and try to match a signature. [1] This works as the polymorphic engine does not significantly change the native opcode that runs in memory. [3] another way to detect polymorphic malware is by using Neural Pattern Recognition, which has shown a high detection rate, based on a small sample set. [3]

Malware obfuscation is a fast-paced arms race that continuously results in more dangerous malware that is harder to detect. Malware authors attempt to counter sandboxed execution by creating malware that detects when it is running in a virtualised environment and not decrypt its payload. Other malware authors create malware that may wait for some event that does not usually occur when executed in a sandbox, before decrypting its payload. Detectors are improving all the time and are incorporating features to defeat this type of malware with advanced techniques. [1] The decrypted code is essentially the same in each case, thus RAM/memory-based signature detection is possible. Block hashing can also be effective in identifying memory-based remnants.

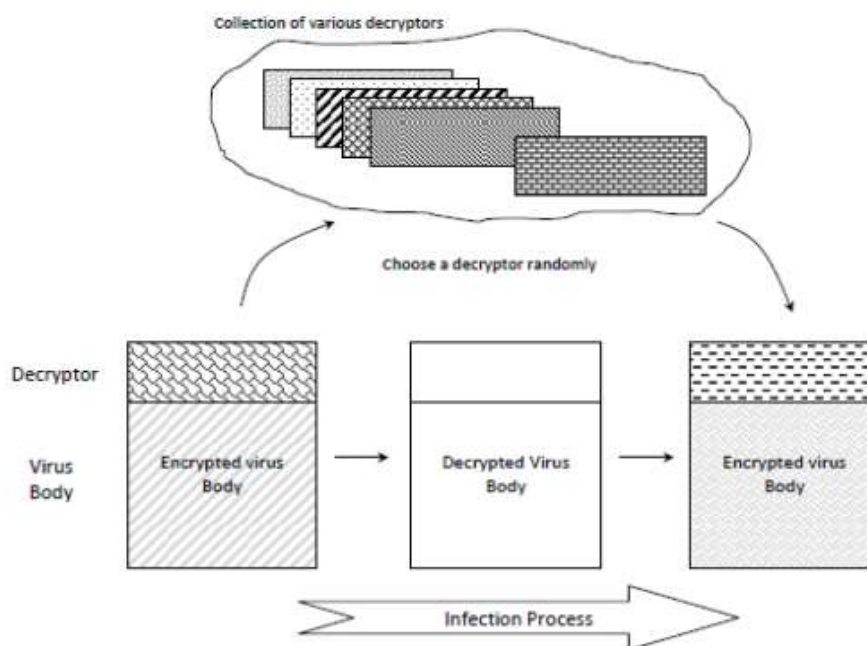


Figure 2

Metamorphic malware

With the previous class of malware, we discussed how the decryptor was changed with each generation of the malware to avoid detection. Metamorphic malware takes this approach and builds on it by incorporating multiple obfuscation techniques into its payload rather than, or as well as, its decryptor. This way it may not need to use encryption or packing and still can be difficult to detect due to its ever-changing signature. It can maintain its behaviour without ever needing to repeat the same set of native opcodes in

memory. [3] It needs to be able to recognize, parse and mutate its own body whenever it propagates. [1]

There are two types of metamorphic malware, open-world and close-world. Open-world, as shown in the Conficker Worm, leverages a command and control structure - with the malware connecting to its controlling master server to download updates and functionality after the initial infection. Closed-world malware from each generation to the next uses self-mutating code via a binary transformer which modifies the binary code itself to avoid detection. [3]. Win32/Apparition was the first example to demonstrate these techniques. [3] The methods used to achieve this level of obfuscation are discussed below.

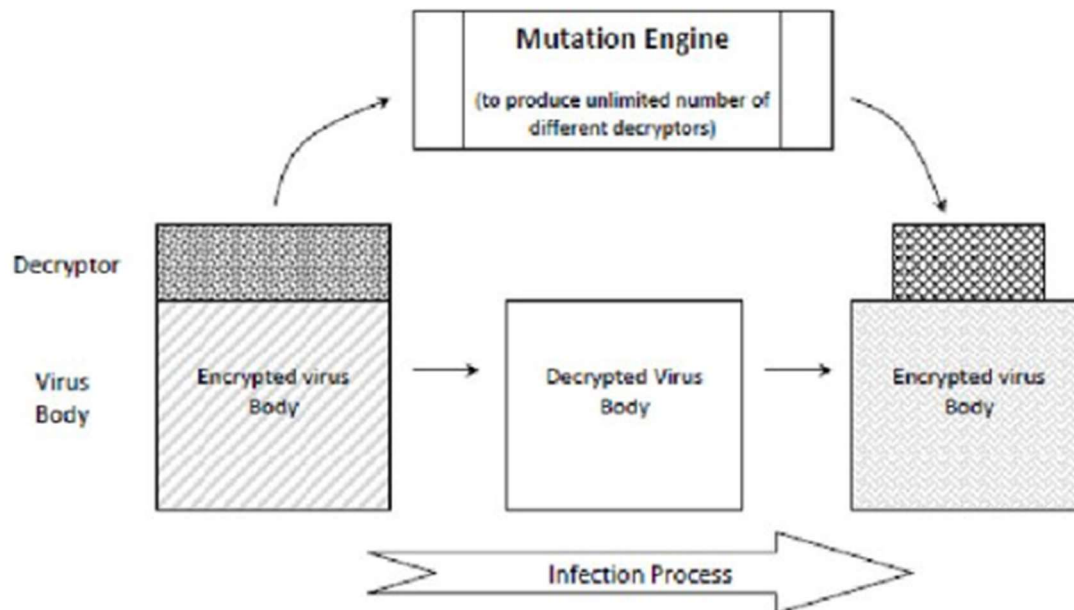


Figure 3

Obfuscation techniques

Polymorphic and Metamorphic malware take advantage of several techniques to obfuscate their code. We are going to go through several methods now.

Garbage/Dead Code Insertion; Dead code insertion pads out the code in some way with garbage, to change the files signature. This garbage could be randomly generated strings; or it could be new instruction sets that don't do anything, or just don't change the malicious operation of the code. NOP or CLC instructions can be used to fill out the code no operation instructions. Using Push and Pop operations on registers is another way. These garbage insertions can be defeated by modern detectors which identify the garbage, such as operations that do nothing, and then deletes it from the code before analysing and comparing the malwares signature. [8]

Register Reassignment/Swapping; In assembly, all programs work from a limited set of instructions and have a limited set of memory space for storing and fetching values. These memory spaces are known as CPU Registers. The number of registers a CPU has can vary. i386, for example, has 4 main registers; EAX, EBX, ECX and EDX. Malware can take advantage of these multiple registers for obfuscation. By switching the registers called

and used the malware can change its code, such as from EAX to EBX and vice versa, from generation to generation while keeping the behaviour the same. [5][1]

Changing flow control/Subroutine Reordering; By changing the order of the program's subroutines malware can produce an exponential number of potential variations. This involves changing jumps in the assembly code and reordering the call sequence by adding subroutines.[3] By changing the order of these jumps, and the order in which different functions are called - combined with other obfuscation methods, such as dead code function insertions, we not only change the signature and make it difficult for automated detectors to identify the malware, but we also increase the challenge of identifying what the malware does through static analysis. Block hashing and heuristic analysis can be the best ways for detectors to identify malware of this type.

Code/Instruction Substitution; Malware, like all code, is made up of a sequence of functions. With most programming languages there are multiple functions that can carry out the same behaviour. In x86, for example, XOR can be replaced by SUB and MOV can be replaced with PUSH. [1] This change results in a new generation of malware, with its own signature that is difficult for detectors to pick up on, even when detecting the instruction set used by the malware. Heuristic and behavioural detection are best placed to identify malware using this form of obfuscation. [8]


Code Transposition; Code transposition is reordering the code in a way the does not impact functionality. This can be through shuffling the order of the instructions and then calling them when needed in the main body. with unconditional branching statements or jumps [1]. The original malware can still be recovered by removing those statements and jumps. This obfuscation, because the malware is so complex, can be difficult and time consuming to both create it, and to detect it. Block hashing is one way to detect this form of malware, where the detectors hash segments, or blocks, of the malicious code are hashed and then checked by an algorithm for similarities with known malware.

Code Integration/Insertion; This is one of the most difficult malware obfuscation techniques to both implement and to detection or analyse. it involves the malware inserting it code within a legitimate program. It does this by decompiling the target executables into manageable objects and inserting itself in between those objects and finally reassembling the entire executable. Once reassembled we see the new generation of the malware. This changes the target programs signature and makes the malware difficult to detect. The best way to detect this malware is by keeping a database of legitimate/white-listed applications and their corresponding baseline signature and treat any applications that deviate from this baseline as malicious. Block hashing and heuristics detection can also be used.

Fileless malware; A new trend in malware obfuscation that has come to the fore over the past 2 years is fileless malware. This obfuscation technique has the malware forgo having a copy of itself stored on the target machines HDD or SSD completely and lives entirely in the RAM. Detectors can have a hard time detecting the malicious function, especially if it combines some armouring techniques, such as relying on external events before acting maliciously, and even when it is detected it can be difficult for to analyse as once the machine is shut down the malware is gone. A live image of the ram is needed to analyse it.

Let's try to obfuscate some malware!

Now we know how the theory of obfuscation, and the different methods we can employ, let's put this theory into practice. We are taking the sample malware from Das Malwerk <http://dasmalwerk.eu> we have chosen *Filename: 25786c51-414b-11e8-a472-80e65024849a.file* as we will obfuscate. This malware has a hash of 36E79238CF645F38FA9CE671A850CC3E29338B65 with a detection rate of 50 / 63 engines picking it up.

50 engines detected this file			
		SHA-256: dd65ca16d3c84bdc6ed04c78c35aap4ad8d145fc9517eac596d51a73bc88bd5e File name: 17b55c7-414b-11e8-b18b-80e65024849a.file File size: 839.5 KB Last analysis: 2019-03-01 08:18:07 UTC Community score: -13	
50 / 63			
Detection	Details	Relations	Behavior
Acrionis	suspicious	Ad-Aware	Generic.MSIL.PasswordStealer.A.60AE...
AhnLab-V3	Trojan.Win32.Orcusnot.C1515795	ALYac	Generic.MSIL.PasswordStealer.A.60AE...
Antiy-AVL	Trojan.Win32.AGeneric	Avast	Generic.MSIL.PasswordStealer.A.60AE...
Avast	Win32.RATX.gen (Tij)	AVG	Win32.RATX.gen (Tij)
Avira	HEUR/AGEN.1018237	BitDefender	Generic.MSIL.PasswordStealer.A.60AE...
CAT-QuickHeal	Trojan.Generic	ClamAV	Win.Packed.Razy.6847695-0
Comodo	Malware.khpi2802a8refi	CrowdStrike Falcon	win/malicious_confidence_100% (W)
Cybereason	maliciousc287	Cyren	W32/Trojan.SWgeniEldoradi
DrWeb	Trojan.Downloader21:65412	Emsisoft	Backdoor.Croos (A)
Endgame	malicious (high confidence)	eScan	Generic.MSIL.PasswordStealer.A.60AE...
ESET-NOD32	a variant of MSIL/Agent.AGU	Fortinet	MSIL/EpyPSWAIQtr
GData	MSIL.Backdoor.Croos.A	Ikarus	Trojan.Rat.Drus
Jiangmin	Trojan.Generic.cacmm	K7AntiVirus	Trojan (004dcf4d1)
K7GW	Trojan (004dcf4d1)	Kaspersky	HEUR.Trojan.Win32.Generic
Malwarebytes	Backdoor.Droos.Generic	MAX	malware (ai score=100)
McAfee	GenericRXAB.TP15B13E1BCC26F	McAfee-GW-Edition	BehavesLike.Win32.Generic.cc
Microsoft	Trojan.Win32/Skeeyah.Adrin	NANO-Antivirus	Trojan.Win32.Skeeyah.xjxjg
Palo Alto Networks	generic.mil	Panda	Tij/CIA
Qihoo-360	Win32/Trojan.RIT	SentinelOne	static engine - malicious
Sophos AV	Mal/Generic-S	Sophos ML	heuristic
SUPERAntiSpyware	Trojan.Agent/Gon.Injector	Symantec	Trojan.Samurai
TACHYON	Backdoor/W32.DN.Agent.859648	Tencent	Win32.Trojan.Generic.SvHj
Trapmine	malicious (high score)	YSA32	T5cape.Trojan.MSIL
ViRobot	Dropper.S.Agent.859648	Webroot	W32.Trojan.Gon
Yandex	Trojan.Agent/uhvEMZvryd	ZoneAlarm	HEUR.Trojan.Win32.Generic
AegisLab	Clean	Alibaba	Clean
Avast Mobile Security	Clean	Babable	Clean
Baidu	Clean	CMC	Clean
eGambit	Clean	F-Secure	Clean
Kingsoft	Clean	TheHacker	Clean
TotalDefense	Clean	Trustlook	Clean
Zoner	Clean	Symantec Mobile Insight	Unable to process file type

Initial output from Virus Total

Here we can see 50 engines detect our malware, so we are going to now try a few ways to reduce the detection rate. From static analysis we could identify that this file is written

in .NET. By using a .NET packer, Netshrink, we get the new hash of; 87755627F18616749F257524152B1C60F036C6EF when checking this hash in VirusTotal, success! It does not exist.



Oops, I know nothing about this item.

Hi there, my name is Win32.Helpware.VT... certain antivirus labs also call me W32.eHeur.BadNews.GAFE, I guess it is because every time I appear they get very upset. It looks like you found a hole in my malware net...

File "87755627F18616749F257524152B1C60F036C6EF" not found

Virus Total doesn't have the files hash value

This is good, but next let's upload the file to virus total. For the hash to be detected the hash must be in the VirusTotal hash database, by uploading the malware we can check

25 engines detected this file

SHA-256

e06e13a3cb7b711bc23ed23a239343ef95f2a2a2d6050a1de1db156c382f89

File name

packed.exe

File size

654.5 KB

Last analysis

2019-03-07 22:01:24 UTC

25 / 66

Detection

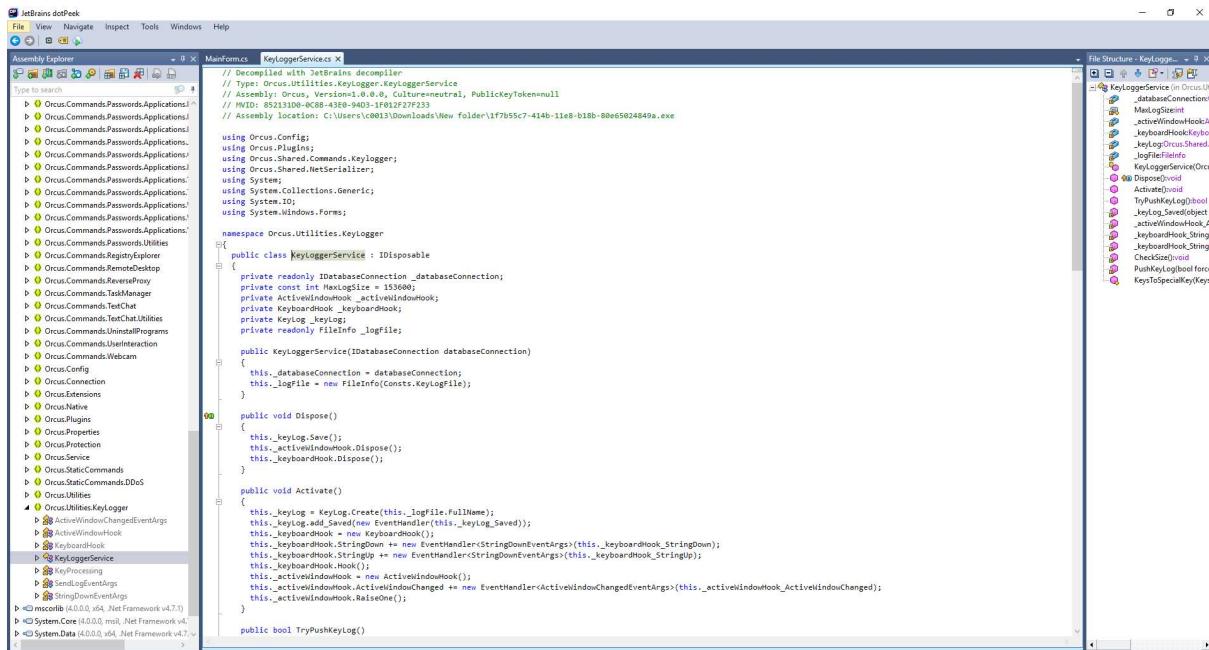
Details

Community

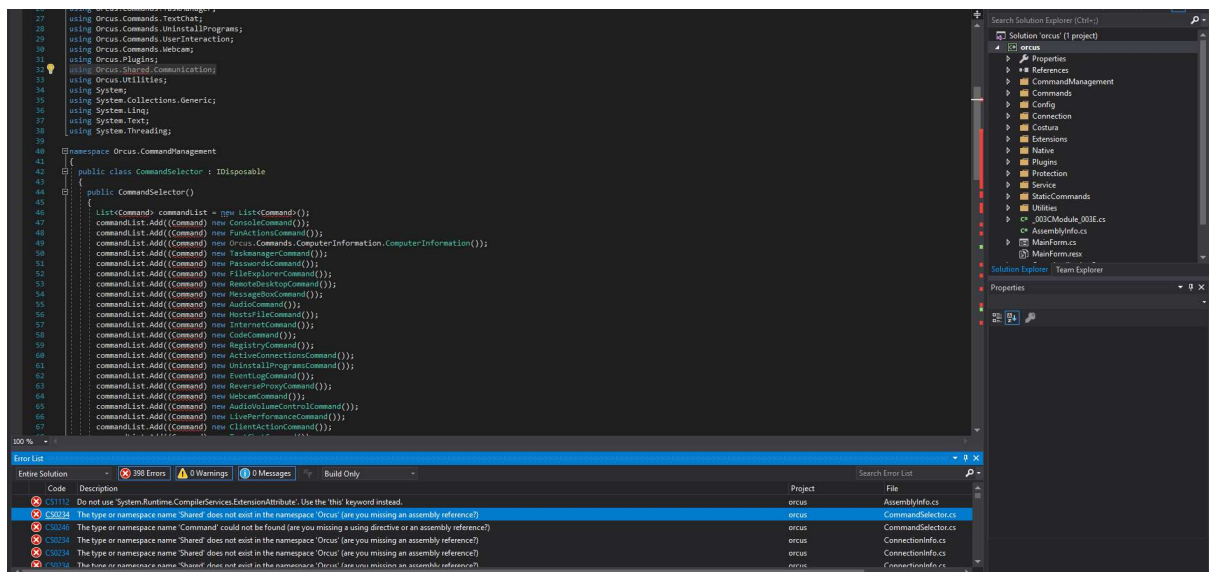
Acronis	suspicious	Ad-Aware	Gen.Variant.MSIL.Perseus.180506
ALYac	Gen.Variant.MSIL.Perseus.180506	Arcabit	Trojan.MSIL.Perseus.D2C1.1.A
Avira	TR/Dropper.MSIL.Gen	BitDefender	Gen.Variant.MSIL.Perseus.180506
CAT-QuickHeal	Trojan.Yakbook.MSIL.Z24	CrowdStrike Falcon	win/malicious.confidence_80% (D)
Cybereason	malicious.678417	Cylance	Unsafe
Cyren	W32/MSIL.Injector.KKGen.ElDorado	DrWeb	Trojan.Downloader.2.E.65412
Emsisoft	Backdoor.Dryux (A)	Endgame	malicious (high confidence)
eScan	Gen.Variant.MSIL.Perseus.180506	F-Secure	Trojan.TR/Dropper.MSIL.Gen
GData	Gen.Variant.MSIL.Perseus.180506	Jiangmin	Trojan.MSIL.kido
Malwarebytes	Backdoor.Dryux.Genetic	MAX	malware (aj score=87)
McAfee-GW-Edition	BehavesLike.Win32.Generic.jc	Sophos ML	heuristic
SUPERAntiSpyware	Trojan.Agent/Gen.Injector	Symantec	MLAttributed.HighConfidence
Trapsmine	malicious.ghmtacon	AegisLab	Clean
AhnLab-V3	Clean	Alibaba	Clean
Antiy-AVL	Clean	Avast	Clean
Avast Mobile Security	Clean	AVG	Clean
Bababie	Clean	Baidu	Clean
Bkav	Clean	ClamAV	Clean
CMC	Clean	Comodo	Clean
eGambit	Clean	ESET-NOD32	Clean
Fortinet	Clean	Ikarus	Clean
K7AntiVirus	Clean	K7GW	Clean
Kaspersky	Clean	Kingsoft	Clean
McAfee	Clean	Microsoft	Clean
NANO-Antivirus	Clean	Palo Alto Networks	Clean
Panda	Clean	Qihoo-360	Clean
Rising	Clean	SentinelOne	Clean
Sophos AV	Clean	TACHYON	Clean
Tencent	Clean	TheHacker	Clean
Trustlook	Clean	VBA32	Clean
VIPRE	Clean	ViRobot	Clean
Webroot	Clean	Yandex	Clean
ZoneAlarm	Clean	Zoner	Clean
F-Prot	Timed out	TrendMicro	Timed out
TrendMicro-HouseCall	Timed out	Zillya	Timed out
Symantec Mobile Insight	Unable to process file type		

So just by changing the packer we can reduce the detection rate from 50 to 25! The detectors that did identify the malware we can see their comments like "Behaves Like",

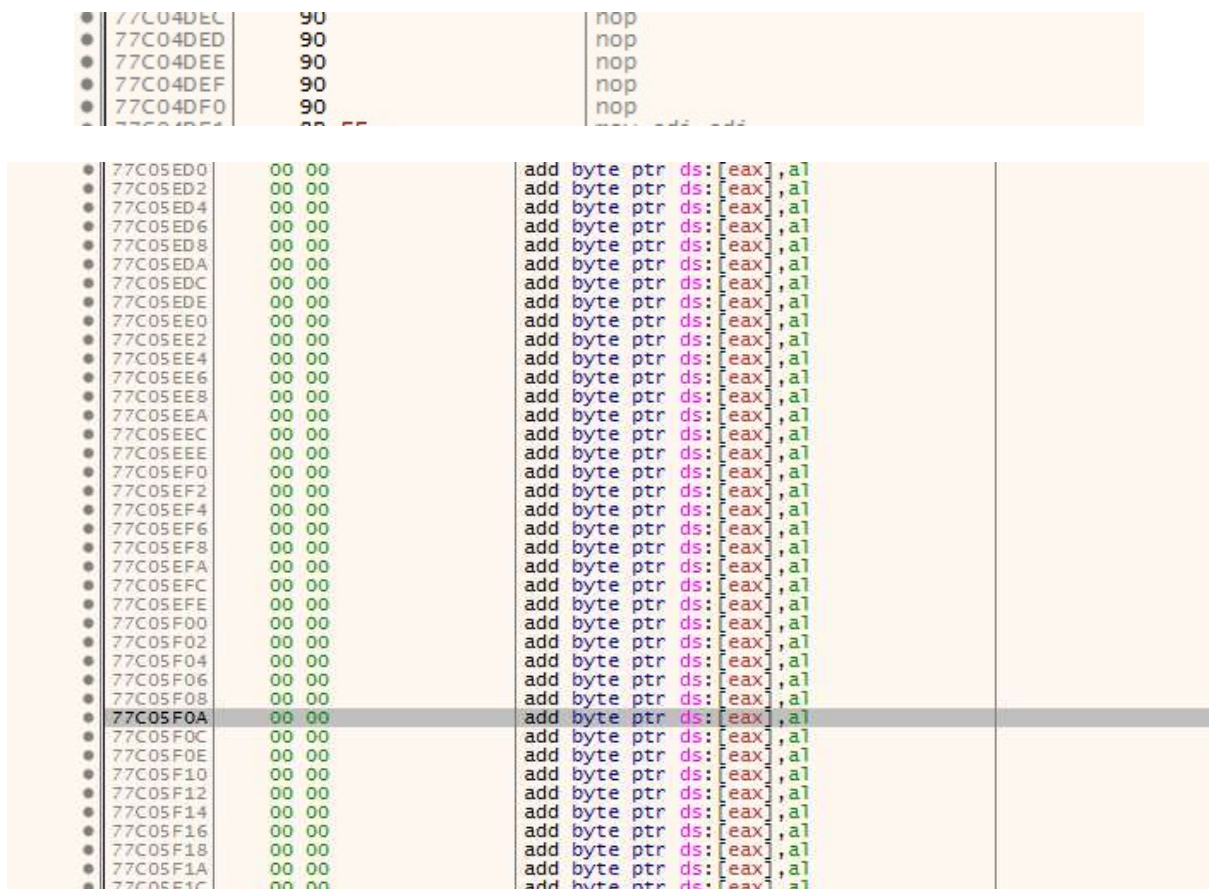
"Heuristic", and "Suspicious" this suggests that some form of dynamic analysis was used to identify the file as malicious. Let's try now to play with the source code. We will decompile this .net application with dotPEEK. This gives us the source code in an exported visual basic file. Opening this in VB Studio we can see the complexity of the malware we selected. First we are going to add a function that will add two numbers, then recompile and get a hash, then compare results.



Opening the file in Visual studio then we see we cannot compile it again. DotPEEK seems to have decompiled it with errors such as "`base.\u002Ector();`" instead of "`base.ctor();`;" 261 errors different errors to fix in all. With this fixed and compiling successfully we have a full understanding of what this malware – Orcus does. Complete with allowing partial Remote Code Execution, setting up FTP servers, allowing DDOS, stealing password and logging keystrokes, we must proceed with the utmost caution. Like a big game hunter about to take out his first sealion. Unfortunately after fixing these 261 errors we get an additional 400 errors, such as "The type or namespace name 'Shared' does not exist in the namespace 'Orcus' (are you missing an assembly reference?) -`using Orcus.Shared.Communication;`" which is beyond our understanding of computer programming. This could be the result of the decompile not catching all of the source code.



Instead of a decompiler let's try using a debugger to walk through the assembly and see if there are some changes we can make at that level. We can see the malware author has done extensive obfuscation already. We saw this when investigating the source code above where we found functions that did nothing. Here at assembly level we see dead code insertion via `nop` and padding at the base of the file;



One thing we could do for an easy demonstration is change the register from the padding at the end from `EAX` to `EBX` but let us try something more challenging. One thing I am

worried about is damaging the code functionality so I am going to replace some of the NOP commands with push EAX; pop EAX which should serve the same function, this demonstrates dead code insertion and instruction substitution. This was done for multiple pairs of NOP commands found.

77B3107A	53	push ebx
77B3107B	5B	pop ebx
77B3107C	50	push eax
77B3107D	58	pop eax

After this small change we have a hash of; 5AED9A880DB19E1EC35E8A63C09EEF45EC50A2C7 lets see if this, before packing it, makes a difference to out detection rate. As expected this file hash has nothing found on Virus Total. When uploading the file itself we get 42/70 detection rate. This is somewhat better than the initial 50/68 we got initially.

42 / 70			Last analysis: 2019-03-12 21:15:08 UTC	
Detection	Details	Community		
Acronis	suspicious	Ad-Aware	Generic.MSIL.PasswordStealer.A-4267...	
AhnLab-V3	Trojan:Win32.Orcuson.C151E795	ALYac	Generic.MSIL.PasswordStealer.A-4267...	
Arcabit	Generic.MSIL.PasswordStealer.A-4267...	Avast	Win32.RATX-gen [Tij]	
AVG	Win32.RATX-gen [Tij]	Avira	HEUR:AGEN.1016231	
BitDefender	Generic.MSIL.PasswordStealer.A-4267...	ClamAV	Win.Packed.Roxy-6847895-0	
CrowdStrike Falcon	win/malicious_confidence_100% (D)	Cybereason	malicious44478	
Cylance	Unsafe	DrWeb	Trojan.Downloader21.65412	
Emsisoft	Backdoor.Orcus (A)	Endgame	malicious (high confidence)	
eScan	Generic.MSIL.PasswordStealer.A-4267...	ESET-NOD32	a variant of MSIL/Agent.ACU	
F-Secure	Heuristic.HEUR/AGEN.1016231	Fortinet	MSIL.SpyBSWAVQ3ur	
GData	MSIL.Backdoor.Orcus.A	Ikarus	Trojan-Rac.Orcus	
Jiangmin	Trojan.Generic.Cacemy	K7AntiVirus	Trojan (004dcf4d1)	
K7GW	Trojan (004dcf4d1)	Kaspersky	HEUR:Trojan.Win32.Generic	
Malwarebytes	Backdoor.Orcus.Generic	MAX	malware (ai score=85)	
McAfee	Backdoor.FD1E109CA297C4A47	McAfee-GW-Edition	BehaviorLike.Win32.Generic.cs	
Microsoft	Trojan/Win32/Fuery.Cid1	Qihoo-360	HEUR/QVM03.0.CQ27.Malware.Gen	
Rising	Backdoor.Panorbi11.6637 (CLASSIC)	SentinelOne	DF1 - Malicious PE	
Sophos AV	Mal/Generic.S	Sophos ML	heuristic	
SUPERAntiSpyware	Trojan.Agent/Gen-Injector	Symantec	Trojan.Sorcurat	
TACHYON	Backdoor/Win32.DM.Agent.A/99648	Trapsmine	malicious/highmlscore	
VBA32	TSoope.Trojan.MSIL	ZoneAlarm	HEUR:Trojan/Win32.Generic	
AvastLab	Clean	Alibaba	Clean	
Antiy-AVL	Clean	Avast Mobile Security	Clean	
Bababab	Clean	Baidu	Clean	
BitDefender	Clean	CAT-QuickHeal	Clean	
CMC	Clean	Comodo	Clean	
Cyren	Clean	eGambit	Clean	
F-Prot	Clean	Kingssoft	Clean	
NANO-Antivirus	Clean	Palo Alto Networks	Clean	
Panda	Clean	Tencent	Clean	
TheHacker	Clean	TotalDefense	Clean	
TrendMicro	Clean	TrendMicro-HouseCall	Clean	
Trustlook	Clean	ViRobot	Clean	
Webroot	Clean	Yandex	Clean	
Zillya	Clean	Zoner	Clean	
Symantec Mobile Insight	Clean	Unable to process this type		

If we pack this malware after the changes we get a hash of 2AB951E7904EBBF355954C5501E6D5EE356120AF and this hash still has no matches on Virus Total. Interestingly when we upload this file to Virus Total we get 32/68 detections. Which is higher than our initial packed file. This could be due to the frequency of uploads we have done and the time the detectors have had to analyse our files.

 32 engines detected this file SHA-256: 1d8d2e3100a18bac98ca874e013a012wed793cbe4e0520690067cafe0d0ff4 File name: 1f7b2537-414b-11e6-b10b-80e00340404e_dump-packed.exe File size: 854 KB Last analysis: 2016-03-12 21:29:47 UTC		32 / 68	
Detection	Details	Community	
Acronis	Acronis	Ad-Aware	GenVariant.MSL.Pernous.180...
ALYac	GenVariant.MSL.Pernous.180...	Antalbit	Trojan.MSL.Pernous.QDC11A
Avira	TR/Chropan.MSL.Gen	BitDefender	GenVariant.MSL.Pernous.180...
CAT-Quickheal	Trojan.Yakbars.MSL.CC4	CrowdStrike Falcon	www.malicious.com/denise... (3)
Cybereason	malicious.418e18	Cylance	Unreals
Cyren	W32/MSIL.Injector.RK.gen!EL...	DrWeb	Trojan.Downloader.DT.B5412
Emisoft	Backdoor.Chous (A)	Endgame	malicious (high confidence)
eScan	GenVariant.MSL.Pernous.180...	ESET-NOD32	a variant of MSIL/Kryptik.Q2V
F-Protec	W32/MSIL.Injector.RK.gen!EL...	F-Secure	Trojan.TR.Chropan.MSL.Gen
GData	GenVariant.MSL.Pernous.180...	Jiangmin	Trojan.MSL.Jack
Kaspersky	HEUR:Trojan.MSL.Crypt.gen	Malwarebytes	Backdoor.Chous.Genens
MAX	malware (ai score=0.2)	McAfee-GW-Edition	Behrendt.Ike.Vm32.Camato.p...
Panda	Trj/GedSak.A	Qihoo-360	HEUR/CYND1.DC007A/move...
SentinelOne	DF1-Suspicious.FE	Sophos ML	heuristic
SUPERAntiSpyware	Trojan.Agent.GenInspector	Symantec	ML.Antibate.HighConfidence
Tapestone	heuristic.highconfidence	ZoneAlarm	HEUR/Trojan.MSL.Crypt.gen
AvastLab	Clean	AhnLab-V3	Clean
Avira	Clean	Antiy-AVL	Clean
Avast	Clean	Avast Mobile Security	Clean
AvG	Clean	Babable	Clean
Baidu	Clean	Bkav	Clean
CMC	Clean	Comodo	Clean
eGambit	Clean	Fortinet	Clean
Ikarus	Clean	KTAntiVirus	Clean
K7GW	Clean	Kingsoft	Clean
McAfee	Clean	Microsoft	Clean
NANO-Antivirus	Clean	Palo Alto Networks	Clean
Rising	Clean	Sophos AV	Clean
SACHYON	Clean	Tencent	Clean
TheHacker	Clean	TrendMicro	Clean
TrendMicro-HouseCall	Clean	Trustlook	Clean
VBA32	Clean	VillRobot	Clean
Webroot	Clean	Yandex	Clean
Zillya	Clean	Znver	Clean
ClamAV	Download	TotalDefense	Download
Symantec Mobile Insight	Download the program like this		

As one final test lets try register swapping at the end of file padding to see if there is any difference. We will also change 1 registry used for an actual instruction. As this is a complex code and to avoid breaking it we will change the registry used at the beginning of the binary.

77831010	72 00	jb htd11.77831012	jmp \$0	x87r3 0
77831012	63 18	arpl word ptr ds:[ebx],bx		x87r4 0
77831014	00 03	add byte ptr ds:[ebx],al		x87r5 0
77831016	88 5E 0C	mov ebx,dword ptr ds:[esi+C]		x87r6 0
77831019	39 F8	cmp ebx,edi		x87r7 0
7783101B	0F 85 SE C2 09 00	jne htd11.778C027F		
77831021	64 A1 18 00 00 00	mov eax,dword ptr ds:[18]		x87Tag
77831027	8B 40 30	mov eax,dword ptr ds:[eax+30]		x87TW_0
7783102A	56	push esi		x87TW_2
7783102C	57	push edi		
77C05E96	00 03	add byte ptr ds:[ebx],al		
77C05E98	00 03	add byte ptr ds:[ebx],al		
77C05E9A	00 03	add byte ptr ds:[ebx],al		
77C05E9C	00 03	add byte ptr ds:[ebx],al		
77C05E9E	00 03	add byte ptr ds:[ebx],al		
77C05EA0	00 03	add byte ptr ds:[ebx],al		
77C05EA2	00 03	add byte ptr ds:[ebx],al		
77C05EA4	00 03	add byte ptr ds:[ebx],al		
77C05EA6	00 03	add byte ptr ds:[ebx],al		
77C05EA8	00 03	add byte ptr ds:[ebx],al		
77C05EAA	00 03	add byte ptr ds:[ebx],al		
77C05EAC	00 03	add byte ptr ds:[ebx],al		
77C05EAE	00 03	add byte ptr ds:[ebx],al		
77C05EB0	00 03	add byte ptr ds:[ebx],al		
77C05EB2	00 00	add byte ptr ds:[eax],al		

With this process complete and the resulting file dumped into an exe, we pack it with NetShrink again and have a hash of 5AED9A880DB19E1EC35E8A63C09EEF45EC50A2C7. The result here is unexpected with 42 detectors identifying the malware;

42 engines detected this file

SHA-256: 1137d802e570e9f31ef33458bc971ca7d30130e2c902777163d19479b31eb731e
File name: 1f7b55c7-474b-11e6-b10b-000e5d34040a_xlump.exe
File size: 838.5 KB
Last analysis: 2019-03-12 21:19:08 UTC

42 / 70

Detection Details Community

Acronis	⚠️ Suspicious	Ad-Aware	⚠️ Generic.MSL.PasswordSteal...
AhnLab-V3	⚠️ Trojan/Worm32-Cryptolok.E131...	ALYac	⚠️ Generic.MSL.PasswordSteal...
Avast	⚠️ Generic.MSL.PasswordSteal...	Avast	⚠️ Win32-DATX-gen [Trj]
AVG	⚠️ Win32-DATX-gen [Trj]	Avira	⚠️ HEUR:AGEN.1016211
BitDefender	⚠️ Generic.MSL.PasswordSteal...	ClamAV	⚠️ Win.Packit.Hypoc-E047855-0
CrowdStrike Falcon	⚠️ win/malicious_confidence_1... [D]	Cyberason	⚠️ malicious-46478
Cylance	⚠️ Untrac	DrWeb	⚠️ Trojan.Droptrojan2.055412
Emisoft	⚠️ Backdoor.Chrus [A]	Endgame	⚠️ malicious (high confidence)
eScan	⚠️ Generic.MSL.PasswordSteal...	ESET-NOD32	⚠️ a variant of MSIL (Agent.A55)
E-Secure	⚠️ Heuristic:HEUR/AGEN.1016211	Fortinet	⚠️ MSIL/spy/MSVAVQ-1
QData	⚠️ MSIL.Backdoor.Chrus.A	Ikarus	⚠️ Trojan-Dat.Chrus
Jiangmin	⚠️ Trojan.Generic.scmem	ITAntivirus	⚠️ Trojan (DD4b14d1)
K7GW	⚠️ Trojan (DD4b14d1)	Kaspersky	⚠️ HEUR:Trojan.Win32.Generic
Malwarebytes	⚠️ Backdoor.Chrus.Generic	MAX	⚠️ malware (score=85)
McAfee	⚠️ Backdoor-TDRESC42RTGAM47	McAfee-GW-Editon	⚠️ Backdoor.Worm32.Generic.0...
Microsoft	⚠️ Trojan:Win32/VaryCdr	Qihoo-360	⚠️ HEUR:QVM360.CC237/Malwa...
Rising	⚠️ Backdoor.Pontederf.6037 [CLASSIC]	SentinelOne	⚠️ DTI - Malicious FF
Sophos AV	⚠️ Mal/Generic-S	Sophos ML	⚠️ heuristic
SUPERAntiSpyware	⚠️ Trojan.Agent/Gem-Threat	Symantec	⚠️ Trojan.Somusak
TACHYON	⚠️ Backdoor/W32.DN-Agent.832648	Trojanminer	⚠️ malicious-high-malware
VBA32	⚠️ Trojan.Trojan.MSL	ZoneAlarm	⚠️ HEUR:Trojan.Win32.Generic
AviraLab	✓ Clean	Alibaba	✓ Clean
Antiy-AVL	✓ Clean	Avast Mobile Security	✓ Clean
Bababie	✓ Clean	Baidu	✓ Clean
Bkav	✓ Clean	CAT-QuickHeal	✓ Clean
CMC	✓ Clean	Comodo	✓ Clean
Cyren	✓ Clean	eGardist	✓ Clean
F-Prot	✓ Clean	Kingsoft	✓ Clean
NANO-Antivirus	✓ Clean	Palo Alto Networks	✓ Clean
Panda	✓ Clean	Tencent	✓ Clean
ThreatHacker	✓ Clean	TotalDefense	✓ Clean
TrendMicro	✓ Clean	TrendMicro-HouseCall	✓ Clean
Trustlook	✓ Clean	VillRobot	✓ Clean
Webroot	✓ Clean	Vandex	✓ Clean
Zillya	✓ Clean	Zoner	✓ Clean

Symantec Mobile Insight

Click here to process the logs

When we upload the file, itself we get the same result.

Three conclusions that we could draw from this are; VirusTotal and its detectors are learning from our uploads each time we obfuscate the malware to become more accurate at detecting its malicious nature. The packer we used, NetShrink could be relatively obscure and the detectors had to spend time analysing it(in this case over a 2 week period). Finally it could be the obfuscation methods we used towards the end, where we focused on changing small segments of the code in the debugger were insufficient to fool the detectors – in this case it is highly possible block hashing was used. While our obfuscation efforts gave us mixed results we were able to go through several obfuscation

methods and see how obfuscation can be detected. Going through the different obfuscation methods we were able to identify multiple ways to avoid detection from signature and code analysis detection. We were also able to review the source code of this malware, Orcus to identify what its purpose was. This knowledge will provide us with a framework to build upon going forward.

https://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_1_of_2

https://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_2_of_2

- [1] [Malware Obfuscation Techniques: A Brief Survey](#) (2010) - Ilsun You, Kangbin Yim
 - [2] [Obfuscation: The Hidden Malware](#) (2011) - Philip Okane, Sakir Sezer, Kieran Mclaughlin
 - [3] Malware Obfuscation Measuring via Evolutionary Similarity (2009) – Jian Li, Jun Xu, Ming Xu, HengLi Zhao, Ning Zheng
 - [4] Static Detection of Malware (2018) - Chapter 7 - [Olav Lysne](#)
 - [5] <https://sensorstechforum.com/advanced-obfuscation-techniques-malware/>
 - [6] https://www.ncsc.gov.uk/content/files/protected_files/guidance_files/Code-obfuscation.pdf
 - [7] A complete dynamic malware analysis (2016) – Dr. Amit Kumar Bindal, Navroop Kaur
 - [8] A constraint-driven approach for dynamic malware detection (2016) - Mario Luca Bernardi, Marta Cimitile, Francesco Mercaldo, Damiano Distanto,
- Figures 1, 2 & 3: Camouflage In Malware: From Encryption To Metamorphism (2012) ; Babak Bashari Rad, Maslin Masrom, Suhaimi Ibrahim



	Declaration Form TO BE COMPLETED IN FULL	
---	--	--

Student ID	B00123233
Student Name	Adam Miller
Assessment Title	Malware Assignment 2
Module Code	H6012
Module Name	CYBERCRIME MALWARE (MSc.)
Lecturer	IAN SHIEL

A SIGNED AND SCANNED COPY OF THIS FORM MUST ACCOMPANY ALL SUBMISSIONS FOR ASSESSMENT WITHIN THE BODY OF THE DOCUMENT.

STUDENTS SHOULD KEEP A COPY OF ALL WORK SUBMITTED.

Plagiarism: the unacknowledged inclusion of another person's writings, ideas or works, in any formally presented work (including essays, examinations, projects, laboratory reports or presentations). The penalties associated with plagiarism are designed to impose sanctions that reflect the seriousness of the University's commitment to academic integrity. Ensure that you have read and understand the University's Plagiarism Policy and Procedures.

Declaration of Authorship

I declare that the work contained in this submission is my own work and has not been taken from the work of others. Any sources cited have been acknowledged within the text of this submission. I have read and understood the policy regarding plagiarism in the Technical University of Dublin – Blanchardstown campus.



Signed..... Date 19/03/2019.....