PROGRAM OBFUSCATION

PROGRAM OBFUSCATION:

In the previous lecture we had briefly been told about Program Obfuscation. We take a deeper look at Obfuscation and its techniques.

INTRODUCTION:

What is Program Obfuscation?

Program Obfuscation means taking a program and 'modifying' it so that the modified program has the same behavior (giving us the same results) as the original program did. But its code otherwise bears a different resemblance to the original code.
Looking different means that, any "adversary" who has access to the program code or the binary should not be able to figure out the internal working of the program.

** The reason for Program Obfuscation is to prevent or make hard the process of reverse engineering.

Why would someone want to reengineer code?

- Security analysts observe the behavior of a software to see if it performs malicious activities or not. If the software under analysis is indeed a malware, the malware author would desire that its analysis be difficult or the code be convoluted enough to make the "adversary" believe it to be non-malicious.
- Sometimes security analysts observe behavior of malware to understand its behavior and control flow. In such cases the authors of such malware desire that the analysis not reveal the correct functioning of the code.
- Adversary software companies wouldn't mind if they could 'understand' the functioning of software made by a competitor and build better software or sell the same software by a different name.

By analysis we mean the following:
A binary has been disassembled and the disassembled code is under analysis to get an understanding of the functioning of the program. OR
A static analysis of the program code is being performed to get an understanding of the functioning of the program.
Program Obfuscation can be thus performed at various levels.

Games are another example where the creators of the game wouldn't want someone to gain an unfair advantage by modifying the code.
Java Scripts are being deployed widely to be run locally on machines through web browsers these days by proprietary software developers. Developers would obfuscate the source code to make it hard for competitors to understand the behavior.

But if someone spends good enough time on such obfuscated code, it won't be long before he/she figures out the behavior of the target program. So obfuscation is not a fool proof way to realize Intellectual Property Protection.

Now, obfuscation is not only restricted to help malware developers to help prevent analyzing their malware code as noted above.
More general applications do exist.

GENERAL OBFUSCATION APPLICATIONS

- Intellectual Property Protection
  -There are proprietary algorithms developed by developers as a part of an application which they would not like competitors to gain access to. They would like the code to be as intangible as possible.
- Watermarking
  -Embed some kind of "mark" in proprietary digital data or code. IF "something" (code or data) is stolen (or used in an undesirable way) this could be figured out.
  -How this works is the fact that the stolen data will be useless as the "mark" and data are tied together in a way that the data is useful only if the "mark" exists in a way it is supposed to.
  -It is also called Steganography.
  -If an "adversary" figures how and where this "mark" is stored and more importantly the connection between the data and the mark, he could remove the watermark and use the code/data the way he wants.
- Tamper Resistance
  -Say you buy an e-book and Acrobat sells software to read it. Acrobat not only needs to make sure that it has some mechanism to ensure that the e-book is not copied and distributed but also that it is read only using the software it sold.
  -If acrobat sells a dedicated device in which such an e-book is stored, it is easier for Acrobat to ensure that there is no way you get this data out and use it.
  -But if it is software, then it becomes harder. Acrobat will 'encode' data in some way that only the "special software" can read it. Maybe along with the data, there is some information which specifies the display it uses, and the user who is intended to use it and so on. Non-encoded (or with no such other information with it), the data is useless.
  If the "adversary" can figure out the association between and the data and the meta-information, maybe he can modify it and use it the way he wants.

Technically speaking, for all the above applications, if an "adversary" spends enough time on the application, he can subvert the protection mechanisms. IF the data uses any kind of encryption, then the key to decrypt the data will be stored inside the software. Figuring out what the key is then is just a matter of time, thus defeating the whole mechanism.

What are Proprietary software developers doing about it?

Developers are developing technologies that make the "subverting" difficult enough that it is simply not worth trying to subvert. If an adversary is unable to figure out what encryption algorithm is used where the key is stored and in what format then the obfuscation technique is indeed worth using.

Ways to make it tough for subversion of obfuscation?

Maybe the key for encryption is a combination of hardware-device-identifiers of some devices on the system and so on. The adversary will have a tough time figuring out which devices they are combined with figuring out the subset of the identifiers which are being used to make the key. These could be ways in which vendors/authors make it hard to subvert obfuscation mechanisms.

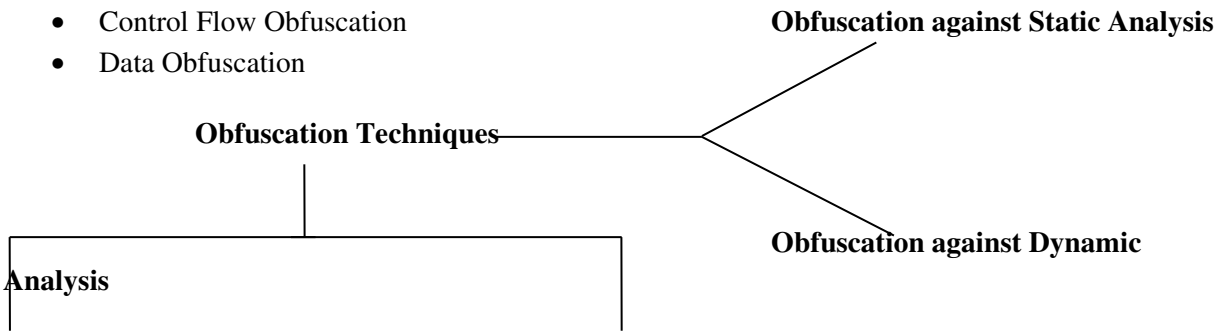
We now take a look at various Obfuscation techniques.

OBFUSCATION TECHNIQUES

Program obfuscation is a general research area. Research has been going on for the last 10 years.
Initially nobody realized the need for it.
General perception: Binary code is hard to figure out by itself.
This perception changed after Java came in. Java byte codes were found easy to reverse engineer which forced people to start thinking of ways to make the process of reverse engineering (understanding the functionality of binaries) harder. Another motivation was copyright protection of music and videos. (DRM is an outcome of this motivation)

Types of Obfuscation Techniques:

- Control Flow Obfuscation
- Data Obfuscation



Control Flow Obfuscation
-moving code around to make code intangible

Data Obfuscation
- making it hard to understand the changes data undergoes.

***** Control Flow Obfuscation (CFO): *****

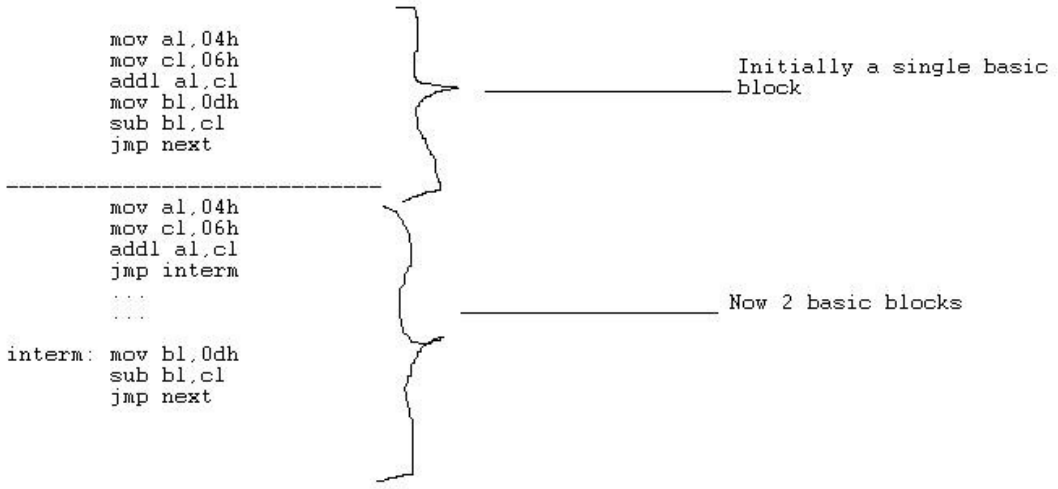
Changing the control flow of the program can be termed as Control Flow Obfuscation.

Renaming Functions is a type of CFO. This may have no relevance in some form of executables. When binaries are made, they are stripped of all function names and symbol information. This works well at a higher level.

Obfuscation can also involve splitting or merging blocks of code. A Basic block can be broken into multiple basic blocks.

Basic block is a sequence of instructions that has one entry point (the first instruction of the block) and one point of exit (the last instruction of the block) with no branching or jump instructions in the middle.

Splitting basic blocks can involve something like this:



The basic blocks can then be re ordered any which way (as long as the functionality of the program is preserved).

Merging/Splitting Loops: The same can be done with loops also.
--Say, a single loop with 2 statements could be turned into 2 loops with one statement each. This can be difficult but it does somewhat make code intangible to the adversary. It is always possible to achieve something like this. The two new statements will now be dependent on each other in some way.
--Similarly, multiple loops can be merged into one single loop.

The idea again is to make code as intangible as possible.

Merging/Splitting Procedures: Just like loops the same can be done with procedures.

A single procedure can be broken into multiple procedures.
Say, a procedure A can be broken into procedures B and C. Thus (when A is called) B will be called followed by a call to C.

Merging of 2 or more procedures into one procedure is possible.

Calling 2 procedures when only one is to be called is another possibility. There could be cases when calling an extra procedure C when only Procedure B is called will not hurt the functionality of the program in any way.

Obfuscation can be done at any level. All techniques are applicable at all levels. But there are some methods of obfuscation which are possible at a lower level which are not possible at higher levels.

Obfuscation at Lower Level vs Obfuscation at Higher Level
At a lower level, you could eliminate all procedures, have no CALLs and instead have only JMPs. Then during disassembly analysis becomes difficult with observer seeing RETs with no CALLs.

Procedures could be either inlined or outlined.
-- Inlining a procedure means that instead of calling a procedure, the procedure is taken and physically inserted at the point where the CALL to the procedure exists.
-- Outlining means that a sequence of instructions which could be grouped as performing functionality, could be taken a made into a procedure and a CALL or a JMP is inserted instead.

Once this kind of splitting or combining is performed it could be reordered to achieve further obfuscation.

Add DEAD CODE in the original code. Dead code means unreachable code i.e. the set of code will never be executed but is there to make the code intangible. Of course there is the need to make it NOT obvious that a particular code is dead code. For instance by putting such code after a JMP would make it obvious that it will never be executed. Desirable thing would be to insert conditions to execute the dead code and also making sure that these conditions are never satisfied so the dead code is not executed.

--Defeating human analysis, defeating techniques which take low level instructions and combine them to give a higher level view of what is going on are the main challenges.
--An important technique which is needed to be avoided is pattern matching which observes a sequence of low level instructions and decides that the pattern translates to some higher level program structure.

> An important thing to note is these obfuscation techniques should be automatic. Questions might arise as to where then these obfuscation techniques are implemented. If it is done at the higher level, then the compiler might optimize them! So it makes sense that these techniques be integrated with the compiler at the right place. Some obfuscation techniques (Data Obfuscation) could actually be done at a higher level as the compiler is unlikely to figure out this and optimize thereafter.

Each of the individual techniques mentioned above are simple as a standalone technique. The real advantage would be when they are combined and then applied iteratively in any particular order. This would be something close to true obfuscation.

*****Data Obfuscation: (DO) *****

This involves obfuscating data instead of control flow.

What is the motivation?
An adversary could also try to figure out the behavior of the program by looking at the transitions that variables/data goes through. The idea is to make it difficult for the adversary to understand these transitions.

Techniques are:
Renaming variables:

This bears no significance at the lower level as data at lower levels are identified and referenced by registers and memory location addresses

Splitting/Aggregating variables:

- Taking individual variables and putting them into a structure or an array. Arrays and structures denote uniformity or represent something as a whole. By putting unrelated variables into a structure or an array an adversary might misinterpret them to be something else.
- 2 variables represent a single variable. Say variable A is now represented by variables B and C as B-C. When B and C are kept separately, it becomes harder to realize that they represent one single variable.

What about obfuscated data interacting with unobfuscated code?

Global variables are not shared between modules. It is rarely found. Say Shared Objects (SOs). Routines are shared and not global variables. Routine names are exported or imported. Internal data representation is the local affair of a program.

Moving variables: from static area to heap or local variables so that it is harder to differentiate local variables from global variables and dynamically allocated variables.

Other techniques:

--Cloning variables
--Inserting junk elements into arrays.
--Encrypting data values.
Decrypt the data when it is needed to be used and then encrypt again. OR use some encryption algorithm or encoding mechanisms that are property preserving. (Order preserving encoding would involve adding a constant offset to a variable and performing arithmetic on it without subtracting the offset.)
--Introducing additional levels of indirection (ALIASING).
Instead of a static variable we have a pointer variable which is initialized. Aliasing could be introduced. Aliasing is a hard analysis problem.
--Introducing memory errors:
Accessing single variables as array members.

Say:
…
int x[10]; -------- > (1)
int y;
int z;
z = z +1; ------- -> (2)
…
Statement 2 is easy to understand.
A memory error here could be referencing z as x[11].
(2) can now be written as:

x[11] = x[11] + 1;

Statements like these are memory errors BUT if permitted, clearly make it confusing for an adversary to analyze the transitions a variable goes through. Moreover if someone is working with binaries it is difficult to figure out the extent/bounds of arrays.

--Adding/Removing Function parameters

Useless parameters could be added and maybe the function performs a lot of operations on these useless parameters. But the actual operation could only have a few lines of code.

As can be noted, the above techniques assume that an adversary has access to the binary (which can be disassembled) or the source code itself.
So we can categorize these techniques as **"Obfuscation against Static Analysis"**

Let us look at obfuscation from a different perspective.

Obfuscation against Dynamic Analysis

The Obfuscation Techniques which we've seen above would most probably work well against static analysis.
However it would not work well against dynamic analysis.

Say, that a program is run in a virtual environment and the adversary observes the system calls which are made by the program. This is the only understanding of the program which an adversary gets in such case. Obfuscation wouldn't help in such analysis.

But, a dynamic analysis does have a limited scope as far as understanding algorithms goes (by simply looking at its inputs and outputs).

Anti virtualization techniques were discussed in brief last class. These techniques are employed by and large by Malware which try and detect if they are being tested/analyzed in a virtual environment. (One such way is by running many I/O operations and non I/O operations and then observing their ratio.)

I/O operations are really slow in VM environment as compared to a non-VM environment, while CPU operation performances are almost the same.

What is the motivation behind Dynamic Analysis?

Intuitive feeling is that Static Analysis would be a real tough task especially since such analyses are automatic and the mechanism can definitely be subverted.
These analyses techniques are merely defenses/solutions developed after newer and newer Obfuscation Techniques were discovered. These analyses may not even completely detect all obfuscation as newer & more complex techniques are developed.
A better compliment would be to see how a software interacts with the System concerned by monitoring System Calls.

Such analyses are performed in a Virtual Environment generally. But Malware, as mentioned could detect this and avoid exhibiting its malicious behavior or it may not be that devious as in, the subject could be a BOT waiting for a command from its BOT MASTER.

A good thing to do could be to combine Static and Dynamic analysis. When unable to figure out what is happening by Dynamic Analysis, use Static Analysis.
Example:
--In Dynamic Analysis, you see that certain paths are not being exercised but want to see the dynamic behavior of those paths. You might need to change a variable value for that. It could however be that there is another variable that needs to be changed too! (…according to the logic of the program). If not then something else could happen (crash maybe).
--This correlation can be figured by means of static analysis. Thus we see the combined use of Static and Dynamic Analysis.
--Limited Static Analysis with Dynamic techniques is thus a good combination.

Summarizing Malware:

* There is plenty of motivation for attackers to remain stealthy. They use techniques like Obfuscation and Anti-virtualization.
* Malware (and their writers) are adaptive. It exhibits evasive techniques to circumvent checks and analysis to achieve its objective and not be discovered.
* Note that, as mentioned above, the analysis techniques are "generally" built after realizing that there are certain peculiar behaviors exhibited by Malware and so on.
  Thus these are reactive mechanisms, whereas Malware behavior could be termed as active.
* Of course if there is a unique detection mechanism which has been developed which is unknown to a large population maybe it will detect & analyze the Malware behavior.
* There is a difference between defenses against Exploits and defenses against Malware. For Malware the need to assume a very strong adversary model is desirable.
  - In exploit detection, it is assumed that program being exploited is a trusted program. Even after plugging in defenses in the program, the program does not actively try and subvert the defense mechanism.
  - A Malware on the other hand tries actively to subvert any defense mechanisms setup against it, to achieve its objective.

* The defense mechanism setup against the Malware should be self protecting.
  If the Malware finds out a way (loophole or a vulnerability in the defense) to disable or subvert the defense mechanisms, then it doesn't make sense.
* This of course is hard to achieve, owing to various factors including System Design, underlying OS and the Architecture. People may be forced to run certain programs with certain privileges. There could be a Malware in it which installs a Rootkit in the system which makes things tougher for the analysts.
* Complete Mediation is another issue.
  Checking if Malware does only 'x', 'y' or 'z', the Malware could do task 'a' and evade the mechanism.
  Thus every action taken by the Malware should be detected and out under the scanner.
* Multi step Attacks (Stepping Stones) is another characteristic of Malware.
  Malware may not achieve its objective in one step, but rather perform a series of steps which leads to its objective in a not so obvious manner.

This is one of the classic techniques for detection evasion. Behavior based detection techniques look for certain behavioral patterns and block it if found.

Defense against Malware is a current problem being faced by all. It is definitely a topic of intense research.

Summarizing other WORRIES:

Insider Attacks:

* One class of attacks is the kind when somebody from outside tries to exploit a vulnerability in a system.
* Another class would be owing to social engineering, the attacker gets a piece of code running into a system, making it now vulnerable
* A third class would be when there are people inside who are "malicious". This of course is a major issue in Financial Institutions in the form of Insider Tradings.

  People are dealing with Insider Attacks through Policies and Law Enforcements.
* A way of achieving this would be to first detect one. This is achieved by means of extensive logging and auditing or access control mechanisms.
* Access control systems tend to be rigid, so they are uncommon to see.
* The reason behind avoiding access control is : There could be exceptions which need to bypass an access control.
* Insider attacks differ from organization to organization.
* Say for a Software developing Organization, one of its developers embeds vulnerabilities in the code. Potential for damage increases manifold as the same software is used in many (thousands of) computers
* It becomes hard to pin point and moreover prove criminal intent. Law Enforcement as a deterrent works little.
* This is an instance of Cyber Warfare or Cyber Terrorism.

This concludes the topic of Malware and Obfuscation techniques.