Dynamic Analysis .. What is it and how to defeat it?!

Dynamic analysis is an important issue today as the number of malware is increasing every year. For example, in the year 2008 Symantec got more than 4000 new unknown sample per day! and MacAfee got about 12,300 per day!. This emphasized the need for automated tools that can scan the submitted samples and try detecting malicious software among them.

In this article I'll try to discuss some of the most frequently used techniques of dynamic analysis with emphasis on how to overcome them.

Before I talk about dynamic analysis let's just say what static analysis is. Static analysis is about analyzing the malware without executing it. The sample will not run but its structure will be examined and searched for known signatures. However, static analysis is very weak against obfuscated or packed malware. Unless the packing routine is well known, the analysis component cannot unpack the target malware for analysis. Also the static analysis can be cheated using indirect jumps and self-modifying code.

The following are some of dynamic analysis methods that are mainly used by AV products and/or sandboxes:

- **API hooking**: the analysis component hooks an API such as CreateFileA, so that when the malware tries to open a file, the component can intercept its call to the function and determine the parameters, that is, it traces the behavior of the malware through intercepting its calls. API hooking can do "post" and "pre" processing of the API call, in other words, the analysis component intercepts the call to the API and examines the parameters then calls the API and gets the returned result for further examination and handles the result back to the malware to continue its execution. This procedure is mostly transparent to the malware as nothing is changed in the parameter or returned results. See figure 1.
    - **Defense:** a malware should be executed in a higher level than the analysis component. If the component is run in the user level, malware should be in kernel level. If the component runs in kernel mode, the malware should exhibit some of rootkits techniques to hide its presence. Also many AV systems opt to monitor Windows APIs but not Native Windows APIs. With such systems, malware can evade the monitoring if using only native windows APIs.

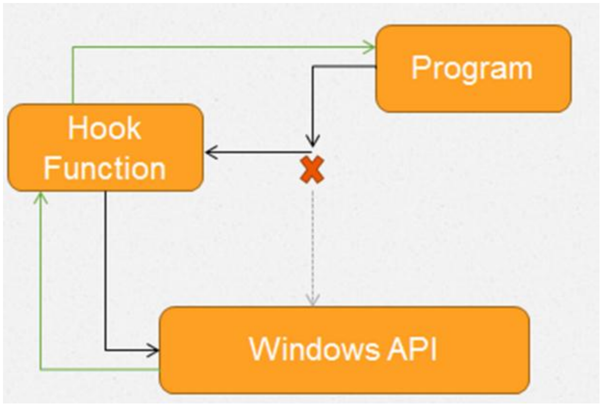

Fig. 1. API hooking

- **Taint Analysis**: Taint analysis is about highlighting "interesting" data in the malware, mark it and trace it along the program execution. Taint analysis mainly consists of three methods:
    - **Direct Data Dependencies**: where the taint label or mark is propagated for direct assignment or arithmetic operations that involve a tainted value. See figure 2.
        - **Defense**: It's pretty easy to find workarounds to assign registers to values without direct assignment or arithmetic instructions. Such methods are to use stack to pass the values through PUSH and POP, or writing the value in a memory location and then reading it back if the AV system doesn't support address tainting, etc. AV systems differ on how tight they monitor instructions.

```
//tainted x

a = x           ; mov eax, x

a = a + x       ; add eax, x
```

Fig. 2. Direct data dependencies example

    - **Address Dependencies**: In this method, a memory address can also be tainted if for example it is written to by an instruction that has a tainted value as one of its arguments. If a tainted value is used as a pointer or as index of an array, the target address is also tainted. See figure 3
        - **Defense**: Data and address dependencies mainly consider single instruction to decide if the operands should be tainted or not. Both data and address dependencies can be evaded using control flow operation to set other variable. See figure 4.

```
//tainted pointer x e.g., jump table
a = x[10]               ; mov eax, [x + 10]

//tainted value y e.g., translation table
b = c[y]                ; mov eax, [c+y]
```

Fig. 3. Address dependencies examples

```
//tainted x
if (x == 0) {
v = 0;
} if (x == 1) {
v = 1;
} …
```

Fig. 4. Setting values using control flow

- **Instruction tracing**: AVs can try to trace the execution of the instruction through setting the trap flag EFLAG which raise debug exception that is handled by the AV and allows it to examine the instruction and its parameter.
    - **Defense**: Anti-debugging techniques that are commonly used by malware can be effectively applied in this case. One of these techniques is to read the EFLAG and detect if it is set or not.
- **CPU & Memory Emulation**: This is the most commonly used by AV products installed at the user end. The AV emulates the commonly used instructions and emulates its execution, it also emulates memory operations. This technique is effective in unpacking malware and obtaining the unpacked body and then applying signature matching over the body.
    - **Defense**:
        - As anti-debugging, there are several anti-emulation techniques that are widely used by malware to detect the emulation environment. Running instructions that are no likely to be emulated is one of these techniques. Also in some cases, there are certain versions of CPU which are known by bugs when executing certain sequence of instructions, emulators will behave different in such situations.
        - Another very useful technique is using the internet to request some data. In that case the emulators will be exposed to the malware as the data will never be delivered. One drawback of this technique is that it is slow and also it can be overcome by network simulation.
        - Logic bomb is a type of malware that doesn't show malicious behavior unless a condition takes place, such as certain date or time or specific user input. Logic bombs can greatly evade emulators as there is very small probability that the condition occurs during the scan.
        - Slow execution of a malware can greatly help it to escape detection by AV. Typical AV system gives few seconds for the samples to execute and unpack to determine if it is malicious or not. If a malware slows down its execution, that would make the AV give up and terminate the scan process. It is highly recommended to not to use RDTSC register for time counting as some AV detect that and patch it so the malware can continue execution.
        - Timing difference is one of the methods that are used to detect the presence of emulator. The time it takes for an emulator to execute an instruction is different than the real CPU, so the malware program can detect this difference and abort execution.
        - A malware program can also check the presence of common external applications on user systems. If such applications are not present, then it is highly likely an analysis environment.

- **Network Simulation**: Some dynamic analysis engines have a network simulator to deceive the malware and make it feels that it can connect to the internet. Generally, those engines either support emulation to basic network service such as DNS and SMTP but with no real data, or allow the malware to have limited connection to the outside world but in this case all the communications is tightly monitored and filtered. But either case they mainly can allow the malware to request data but not send to outside world.
    - **Defense**: Due to this limited allowed outbound connection, a malware can use like three-way handshake connection with an external website. If it fails, then the malware can detect it is inside an emulator.

Here is a table that summarizes all the above:

AV Technique	**Malware fight back**
API hook in user mode	Run in Kernel mode
API hook in kernel mode	Use rootkit techniques
Direct data and address taint analysis	Use control flow statements for information flow
Instruction trace	Use anti-debugging techniques Read EFLAG
CPU & memory emulation	- Anti-emulation techniques - Use data from the internet - Logic bomb behavior - Delayed execution
Network simulation	Get data or files from the internet

Dynamic analysis techniques are diverse and advanced. In this article I tried to shed some lights on how they work on different levels and how a malware writer can fight them back. I hope soon I write a more in-details article about sandboxes that are a widely used by AV companies to classify new unknown samples.

Written By:
M0SA, February 13, 2012
m0sa.vx@gmail.com

References:

- “A Survey on Automate Dynamic Malware Analysis Techniques and Tools”, Manuel Egele et al, ACM Computing Survey.
- “Taint Analysis”, Edgar Barbosa, H2HC 2009
- And many others