

Anti-Virus Engine 설계 1 편

By Maxoverpro[Max](장상근)

maxoverpro@paran.com

<http://www.maxoverpro.org>

이 Anti-Virus Engine 설계 1편에서는 Anti-Virus Engine 설계에 있어 알아야할 사항에 대해서 알아 볼 것이다.

1. 서론.

이 문서는 AV Engine을 이루는 요소들을 포함하여 Anti-Virus Engine을 개발하는 방법에 대한 내용을 다루도록 하겠으며, AV-Engine의 인터페이스, 실시간 감시, 파일 시스템 드라이버와 다른 응용소프트웨어들을 진단하기 위한 Plug-in기능에 대해서 다룬다.

2. 본론.

Anti-Virus Engine을 만들기 위해 아래와 같은 것들을 정하는 것이 우선이다.

① Anti-Virus Engine 설계 시 반드시 고려해야 할 것들.

■ 플랫폼.

AV-Engine 설계에 있어 플랫폼을 정하는 것은 매우 중요한 일로서 다음과 같은 것들이 설계시 고려되어야 한다.

첫째, CPU의 구조(x86, Alpha CPU Etc...) 이다. CPU마다 제 각기 명령어 처리방식이 달라 AV-Engine이 작동되지 않거나 작동한다 하더라도 해당 CPU를 제대로 지원하지 못한다면 그 또한 문제가 된다.

현재까지 많이 쓰이고 있는 CPU(X86,32Bit)가 될 수 있고 지금 바뀌어지고 있는 CPU(x86,64Bit)가 기반이 될 수 있다. 또한 사용용도에 따라 서버급에 설치될 경우 Alpha CPU나 Sparc CPU를 고려해야하고, 휴대폰이나 다른 전자장비를 대상으로 설계할 경우도 생각해야 한다.

둘째, 운영체제이다. 어떤 운영체제를 선택하느냐에 따라 **AV-Engine**의 유명도가 결정되기도 한다. 서버급을 목표로 삼는다면 **SUN OS** 나 **Linux**가 목표가 될 수 있으며, 일반 컴퓨터 사용자들이 많이 사용하고 있는 **MS**의 **WINDOWS**를 고려해야 한다. 어쩌면 이 부분에서 괜찮은 인터페이스와 좋은 **AV-Engine**을 설계한다면 상업화 할 수도 있다.

■ 프로그래밍 언어.

프로그래밍 언어는 현재 다양한 편으로 예전의 **DOS**시절과 달리 **C** 나 **Assembler**만 있지 않고, **C#** ,**Java**등 다양한 언어가 존재하고 있다. 하지만 **AV-Engine**를 설계시 어떤 플랫폼이든 간에 최적화가 되어야 하므로, 어느 플랫폼도 지원하고 시스템에도 최적화 하기 쉬운 **C**나 **C++**를 기반으로 하여 **AV-Engine**이 설계되어지고 있다. 하지만 **C++**은 객체지향 언어에 기반을 두고 있기 때문에 **C**언어 보다는 같은 프로그램을 제작함에 있어 파일이 크거나 느린 문제가 존재하기도 하고, **C**를 선택해도 플랫폼에서 처리하는 데이터 타입에 따라 어떤 문제가 발생할 수도 있다. 되도록이면 **AV-Engine**을 설계하면서 표준화된 방법으로 **AV-Engine**을 설계하는 것이 좋다.

■ 파일 액세스.

AV-Engine 설계 시 매우 중요한 부분으로 운영체제와 그 운영체제의 파일 시스템과는 독립적으로 설계된 가상의 파일시스템 계층에서 작동되어야만 빠른 파일 스캐닝을 가져올 수 있으며 메모리 인터페이스도 이 부분의 한 요소이다. 그리고 **Scan-Engine**은 오염되지 않은 메모리에 할당된 곳에서 작동되어야 한다.

■ 필요로 되는 것들의 모듈화.

모듈화는 최근 소프트웨어 개발 추세로 모듈화를 통해 깔끔한 인터페이스와 각각 만들어진 프로그램들을 한 프로그램으로 만들 수 있는 장점을 가져온다.

이 점에서 실제 **AV-Engine** 기능 설계 시 모듈화 시켜야 부분 중 **Engine Update** 및 관리 프로그램등이 모듈화가 필요한 대표적인 부분이다.

② 실제 **AV-Engine** 에 필요한 기능.

■ **AV-Engine Framework**

AV-Engine Framework는 **AV-Engine**의 자체라고 할 만큼 핵심적인 기술이라고 할 수 있다. **Framework** 설계를 할 때는 확장성을 고려하여 **AV-Engine**에 탑재할 다양한 컴퍼넌트를 지원할 수 있어야 하며, 새로운 파일 포맷을 지원하는 스캐너들도 장착 가능한 구조로 디자인 되어야 한다. 그리고 **AV-Engine Framework**를 보호하기 위해 디지털 증명서 처리나 라이선스를 통해 **AV-Engine Framework**를 보호할 수 있어야 한다.

■ 가상 파일 시스템

Anti-Virus Engine 설계시 반드시 고려해야 할 것들에서 알아보았듯이 운영체제나 파일 시스템에 독립되어 사용해야 한다.

예를 들어 C언어의 파일 처리 함수들이 사용되며 파일 포인터 구조를 이용해 파일 스캐닝 모듈 설계 시 이용된다.

■ 특정 파일 스캐너

Anti-Virus Engine이 파일들을 스캐닝중에 다양한 파일을 만날 수 있다.

예를 들면, 윈도우의 PE 구조가 있겠으며 엑셀파일인 XLS라는 확장자를 가진 파일들일 수도 있겠고, HWP라는 아래한글 파일 일수도 있다. 만약 특정 프로그램에서만 열리는 파일들에 악성코드가 숨겨져 있다면, **AV-Engine** 자체가 탐지하기는 힘들 것이다. 이점에서 특정 파일을 열어 악성코드 검사를 해주는 스캐너가 필요로 되는데 이것을 특정 파일 스캐너라 하겠다.

■ 메모리 스캐너

메모리 스캐너는 메모리에 상주하는 악성코드를 찾아내는 역할을 한다.

■ 파일 압축 해제기

현재 다양한 압축 기술들이 존재하고 있다. 많이 사용되고 있는 ZIP, RAR이 있고 한국에서 많이 쓰이는 ALZ이란 압축 기술이 존재하고 있다. 이런 압축된 파일에 악성코드가 감염되었을 경우 잠재적인 위험이 있기 때문에 파일 압축 해제기로 압축을 해제하고 **Anti-Virus Engine**에 의해서 스캐닝을 가능하게 만들어 주는 역할을 한다.

■ 치료 기능.

Anti-Virus Engine Framework에 이어 **Anti-Virus**의 신뢰성을 가질 수 있는 중요한 부분이 악성코드에 대한 치료부분이다.

이 부분에서는 악성코드를 어떻게 제거해야 하는지에 대한 판단을 해야 하는데 레지스트리에 존재하는 악성코드의 제거나 악성코드 파일을 삭제를 할 것인지 결정을 하게 된다. 이 중에서도 시스템 파일이 악성코드에 감염이 되었을 경우 치료에 대한 신중한 결정과 정확한 치료가 요구되어 진다. 이런 악성코드 치료기능들은 **Anti-Virus Engine**의 성능과 신뢰도에 매우 큰 영향을 미친다.

■ 엔진 업데이트

하루에 수많은 악성코드들이 만들어 지고 있는 것이 현실이고, 아무리 좋은 백신이라도 알려지지 않은 악성코드를 치료하기란 쉽지 않다. 이런 알려지지 않은 악성코드를 치료하기 위해 **Heuristic Anti-Virus** 기술이 있긴 하지만 정확하게 이 것은 이런 악성코드이다. 라고 정의 내리지는 못하고 치료에 있어서도 완벽한 치료는 한계가 있다.

이런 점에서 엔진 업데이트는 매우 중요하다. 업데이트 방법은 **Anti-Virus Engine**의 구성 컴퍼넌트와 실행파일이 될 수 있고, 악성코드의 패턴을 가지고 있는 데이터베이스 파일이 될 수 있다.

현재 많이 사용하고 있는 업데이트 방법은 인터넷을 통한 업데이트를 많이 하고 있다. 업데이트 도중에는 **Anti-Virus Software**를 보호하기 위해 디지털 증명서나 라이선스를 통해 업데이트되는 도중에 악성코드에 의한 **Anti-Virus**가 공격당하지 않도록 해야 한다.

3. 결론.

Anti-Virus Engine 설계에 대한 기본적인 지식을 이 문서에서 다뤘지만, 그렇게 쉽게 만들 수 없는 것이 **Anti-Virus Engine** 이고, 신뢰성과 성능 향상을 위한 오랜 시간 동안 여러 환경에서 테스트 해보아야만 하며 많은 경험과 투자를 해야 제대로 된 **Anti-Virus Engine**을 만들 수 있다.

다음 장에서는 간단하게 실제 **Anti-Virus Engine**를 구현해보도록 하겠다.