Basic Static Analysis

- **실행 파일에서 여러가지 정보를 얻는 방법 3가지**
  1. 악성 여부를 판단하는 악성 안티 바이러스 도구 활용
  2. 악성코드를 판별하는 해시 사용
  3. 파일의 문자열 함수, 헤더에서 개략적인 정보 수집
- **각 기법을 서로 다른 정보를 수집할 수 있고, 사용하는 기법은 분석 목적에 의존적**
  - 일반적으로 많은 정보를 사용하기 위해서 여러가지 기법을 사용

Anti virus scanning

- 처음으로 악성코드를 분석할때 좋은 시작 단계
  - **이미 알려진 악성 코드 일수도 있음**

→ 하지만 안티 바이러스 도구들이 완벽한건 아님

  - 의심파일의 식별자로 이루어지는 데이터 베이스 및 행위와 패턴에 주로 의존

⇒ **악성코드 제작자가 공격코드를 바꿔서 우회한다면 무용지물**

- 대중적으로 알려지지 않는 악성코드는 스캐너를 우회할 가능성이 높음
- **다양한 안티 바이러스 도구에서 확인을 해보는게 낫다**
  - 서로다른 시그니처와 경험적 기법을 사용하기 때문
- 바이러스 토탈 → 대표적인 안티 바이러스 도구

Hash

- **악성코드를 식별할때 사용하는 널리 알려진 방식**

- 악성코드를 식별하는 고유 해시값을 생성
- 안전한 해시 알고리즘 SHA-1을 사용하기도 하지만 MD5가 해시에 흔히 사용됨
- md5deep을 사용해서 해시를 계산할 수 있음

→ 해시를 이름으로 사용가능

→ 악성코드를 분류할 수 있도록 다른 분석가들과 해시를 공유

→ 이미 식별된 파일인지 아닌지 온라인에서 해시를 검색

String

- 프로그램에서 문자열은 연속된 문자를 의미
- 프로그램이 메세지 출력이나 URL접속을 한다고 하면 관련된 문자열을 사용할 것
- 문자열 검색은 프로그램 기능에 대한 힌트를 얻는 간단한 방법
- Strings 를 이용해서 ASCII나 유니코드 포멧으로 저장된 문자열을 검색 가능

⇒ 마이크로 소프트는 유니코드 문자열 구현을 설명하기 위해서 wide character string 이라는것을 사용 → 유니코드의 표준마다 조금씩 다름 그래서 마이크로 소프트 구현을 참고

Packing & 난독화

- **악성코드 제작자는 파일의 탐지와 분석을 더 어렵게 하기 위해서 패킹과 난독화 기법을 사용**
- 난독화는 제작자가 은폐를 시도한 실행코드
- **패킹은 프로그램이 압축되어 분석을 할 수 없게 만드는 것**

  → 악성코드의 정적 분석을 힘들게 함
- 패킹이나 난독화된 프로그램들은 **대부분 문자열이 거의 없음**

  → 소수의 문자열만 보인다면 패킹이나 난독화의 가능성

⇒ 패킹되고 난독화된 코드는 적어도 Load Library 와 GetProcAccess 함수를 포함하는데 추가함수를 로딩해 접근하는데 사용

Packing file

- 패킹된 프로그램을 실행하면 wrapper 프로그램이 패킹된 프로그램 파일의 압축을 해제 언패킹된 파일을 실행
- 패킹된 프로그램을 분석을 하게되면 Warraper 프로그램만 분석이 가능함
- PEID를 이용해서 패커를 탐지

⇒ peid 도구는 2011년 4월 이후로 개발이 중단

→ **여전히 패커와 컴파일러 탐지에 있어서 가장 좋은 도구**

PE file Format

- 파일 포멧은 프로그램 기능에 대해서 많은 정보를 담고 있음
- **윈도우 실행파일에 객체코드 , DLL이 PE 파일을 사용**

→ PE 파일 포멧은 윈도우 OS 로더가 래핑한 실행 코드를 관리할 수 있는 필수 정보를 담은 데이터 구조

- 윈도우가 로드하는 거의 대부분으 프로그램은 PE파일 포멧을 사용
  - 코드에 관한 정보, 애플리케이션 유형, 필요한 라이브러리 함수, 메모리 공간 요구 사항을 포함한 헤더로 시작 **→ 분석가에게는 매우 큰 가치가 있음**

LNK Library & Function

- 실행 파일에서 가장 유용한 정보 중 하나 import 하는 함수 목록
- import란 실제 다른 프로그램에 저장되어 있지만 외부 프로그램이 사용할 수 있게하는 함수

  일반적인 코드들이 담고 있는 라이브러리

  → 코드 라이브러리는 링크로 메인 실행파일을 연결할 수 있음

  → 정적으로 실행시간에 동적으로 링크가 가능 | **ex) 파이썬 모듈**

- 프로그래머들은 여러 프로그램에서 동작하는 기능들을 재 구현 할 필요 없이 자신의 프로그램에 import해서 사용

⇒ PE 파일 헤더에서 얻을 수 있는 정보는 라이브러리 코드 링크 방법에 의존적

- **악성코드를 이해하는데 라이브러리 코드의 링크 형태를 아는것이 매우 중요**

Static/Runtime/Dynamic LNK

정적링크

- 유닉스와 리눅스에서는 일반적
  - 라이브러리 링크에서는 적게 사용
- 정적으로 연결하면 실행파일로 라이브러리의 모든 코드를 복사 → 실행파일 크기가 증가
- **코드를 분석할때 어려운점은 정적으로 연결된 코드와 실행파일 자체코드를 분간하는 것**

  → PE파일 헤더에서 파일에 링크코드가 존재하는지 여부를 인지하는 정보가 없기 때문
- **런타임 링크는 악성코드의 패킹이나 난독화할 때 자주 사용**
  - 함수가 필요할 때에만 라이브러리에 연결됨

```
일부 마이크로 소프트의 함수들은 프로그래머가
프로그램의 파일헤더에 명시되지 않은 함수들을 임포트하게 해줌

1. LoadLibrary , GetProcAddress
2. LdrGetProcAddress와 LdrLoadDLL도 사용
-> 이것을 사용하면 프로그램에서 시스템 내부의 라이브러리를 모두 호출 가능
-> 이 함수 사용으로는 의심 프로그램이 어떤 함수를 링크 했는지 알 수 없음
```

동적링크

- 악성코드 분석가가 사용하는 제일 흔한 방식
- **라이브러리를 동적으로 연결하면 호스트 운영체제는 프로그램을 로드할때 필요한 라이브러리를 검색**

  → 프로그램이 링크된 라이브러리 함수를 호출하면 라이브러리 내의 함수를 실행

- PE파일 헤더는 로드할 모든 라이브러리 프로그램이 사용할 모든 함수에 대한 정보를 저장
  - **사용된 라이브러리와 호출된 함수 종종 프로그램의 가장 중요한 부분**

- **프로그램이 어떤 작업을 수행하는지 추측이 가능**

```
URLDownLoadToFile 함수를 사용하면
-> 인터넷에서 어떤것을 다운로드 한다는것을 알 수 있음
```

- Dependency Walker를 이용해서 동적링크 함수를 탐색

Import Function

- PE 파일 헤더는 실행 파일이 사용하는 특정 함수의 정보도 포함
- 윈도우 함수명 자체로 실행파일이 무슨 행위를 하는지 정보를 제공함
- MSDN 라이브러리를 이용해서 문서로 제공

Export Function

- DLL과 EXE는 다른 프로그램과 상호 작용 할 수 있도록 해당 함수를 export
- **DLL이 하나 이상의 함수를 구현하여 DLL을 import 하고 재사용 할 수 있도록 export**

  ⇒ DLL은 EXE가 사용하는 기능을 제공하기 위해서 특별히 설계

  그래서 DLL은 익스포트 함수를 보기 쉬움
- EXE는 다른 EXE에 기능을 제공할 용도로 사용되지 않으므로 export 함수는 드물다

⇒ 실행파일 내부에서 export 함수를 발견했다면 유용한 정보들을 얻을 수 있다

- 소프트웨어 제작자는 유용한 정보를 제공하는 형태로 export 함수를 명명
  - **일반적인 방법으로는 MS에서 사용한 이름을 이용하는것**

→ 하지만 프로그래머는 함수를 임의로 명명하는 경우도 존재

→ **따라서 함수 이름만으로 정교한 분석은 한계점이 존재**

⇒ **악성코드가 export를 사용한다면 오해하기 쉬운 이름이나 임의로 함수 이름을 사용하거나 생략 아니면 명백하지 않은 이름을 사용할것**

- Dependency Walker를 이용해서 export 함수를 확인 가능

- **윈도우에서 사용하는 모든 export 함수를 볼 수 있음**