제 10 장.
소프트웨어 보안

컴퓨터시스템보안

최태영

목차

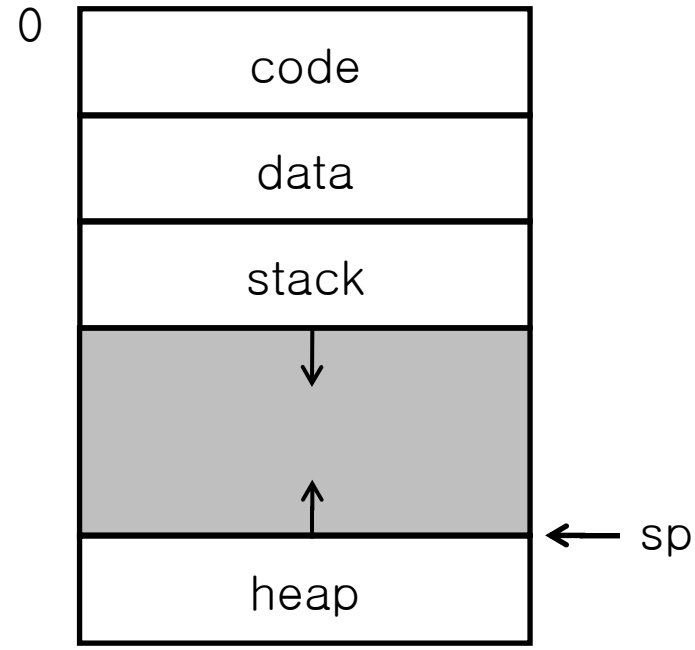
- 소프트웨어 결함
  - Buffer overflow
  - Defense against buffer overflow
  - Incomplete mediation
- Malware
  - Types of malwares
  - Typical malwares
  - Malware detection

소프트웨어 결함

- 현대 소프트웨어는 용량이 매우 큼
  - Linux 소스 코드 : 백만 라인 이상
  - Windows XP 소스 코드 : 4천만 라인 이상
  - 개발단계에서 발견하지 못한 결함이 있을 가능성이 높음
  - 코드 천 라인마다 버그가 1개 있다면
    - 각 운영체제는 최소 1,000개 이상의 버그가 있음
  - 이 버그들은 수정될 수 있는 버그들임
    - 하지만, 패치가 나오기까지는 존재함
  - 대표적 버그들 : buffer overflow, incomplete mediation
- 결함을 공격 대상으로 악의적인 공격이 진행됨

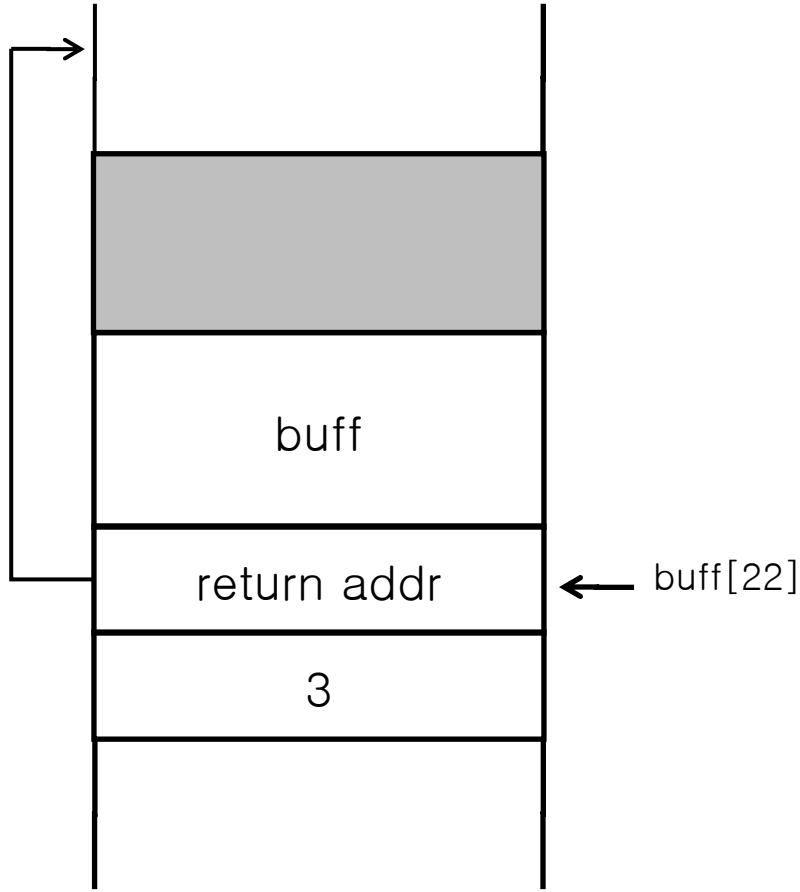
Buffer Overflow

- 받은 데이터는 buffer라는 메모리에 저장
- 받은 데이터의 양이 buffer의 크기보다 크면 buffer 이외의 영역에 데이터가 쓰여짐
- 예) 오른 편의 그림은 일반적인 프로그램의 메모리 할당 형태임
  - Buffer는 data, stack, 또는 heap 영역에 있게 됨.



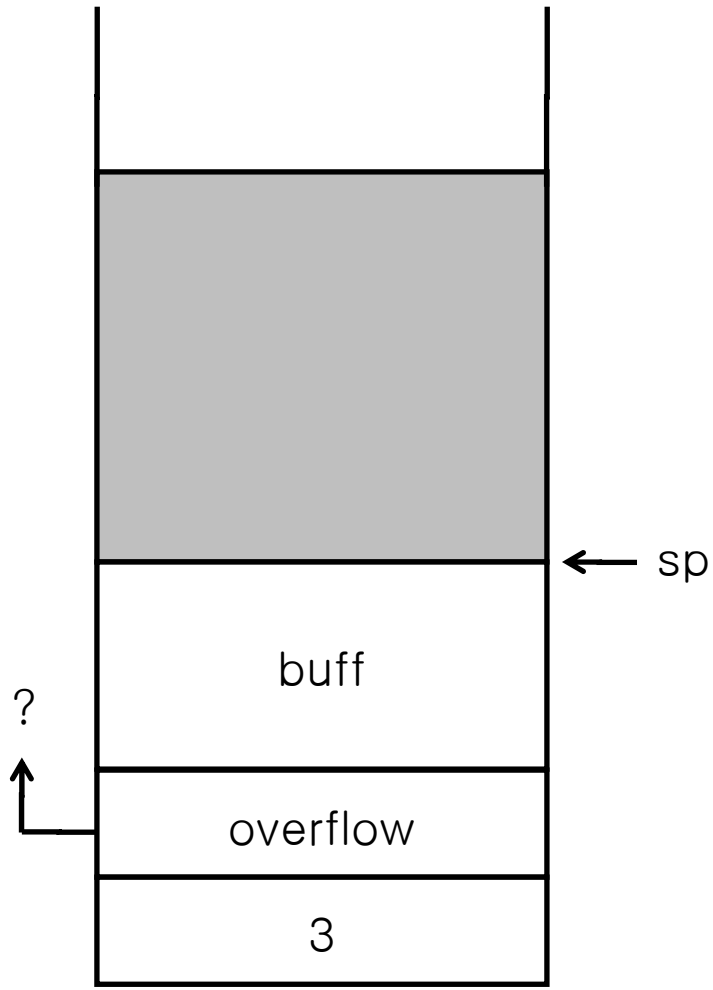
함수 호출시의 메모리 형태

```
void foo(int a)
{
   char buff[20];

   buff[22] = 0;
   return;
}

int main(void)
{
   foo(3);
}
```



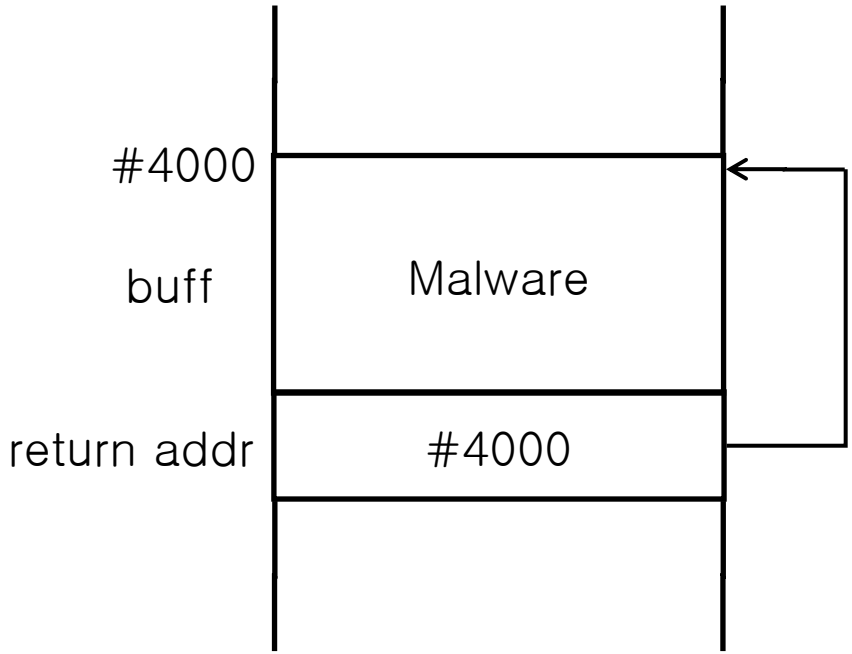
Buffer overflow로 인한 오염

- **`buff[22] = 0;`** 명령어를 수행함에 따라 `return address`의 일부가 `0`으로 변경됨
- 함수 수행이 끝나면 `Program Counter (PC)`는 잘못된 곳으로 이동하여 명령을 수행



Buffer overflow를 이용한 공격

- Malware를 buff에 집어 넣음
  - 메시지 전송 등을 통해
- Buffer overflow를 이용하여 stack내 return addr의 값을 buff 주소로 변경
- 디스크 포멧, 패스워드 파일 전송 등의 명령어 수행

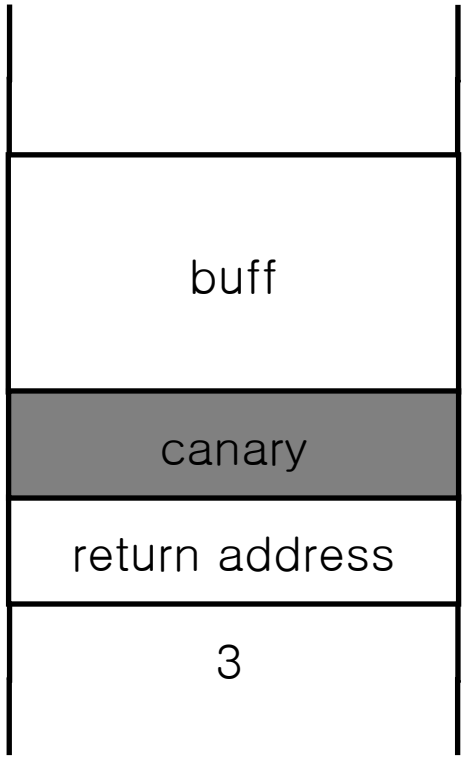


Defense against Buffer Overflow

- 포인터나 배열의 범위를 검사하는 라이브러리 사용
- 개선된 프로그래밍 언어 사용
  - C나 C++은 포인터 사용으로 buffer overflow 유발
  - Java, D, Ada, Lisp 등의 프로그래밍 언어들은 배열의 범위 체크를 실시간으로 수행
  - 대부분의 인터프리터 언어에서 범위 체크를 수행
- 범위 체크는 buffer overflow같은 에러를 줄이지만 실행 시간을 늘이는 오버헤드를 발생시킴
- 카나리아
- 실행 영역 보호

카나리아

- 광산에서 유독가스의 분출을 빨리 인식하기 위한 방법으로 canary의 생존 여부를 이용한 것에서 유래
- Return address와 지역 변수 사이에 삽입한 특정 값
- Buffer overflow 공격을 막기 위한 방법
- Buffer overflow가 발생하면 return address를 변경하는 과정에서 canary값도 변경할 가능성이 많음
- 주기적으로 canary들을 검사하여 변경되었을 경우 buffer overflow 공격으로 판단



변형된 카나리아

- Terminator canaries
  - 문자열이 끝나는 시점인 NULL 문자 다음에 카나리아를 삽입하는 방식
- Random canaries
  - Buffer overflow 공격들 중에는 자신의 내용을 덮어 쓴 후에 카나리아를 덮어 쓰는 방법을 사용하기도 함
  - 이를 막기 위해 시스템 시작 시에 random한 값을 canary로 설정함
- Swap out 방식의 canary
  - 공격자는 random canary를 읽어서 덮어 쓰는 공격을 하기도 함
  - Canary가 있는 페이지를 강제로 swap out한 뒤 그 페이지에 대한 page fault가 발생하면 공격으로 간주

실행 영역 보호

- 컴퓨터 하드웨어에서 각 페이지마다 실행, 읽기, 쓰기 여부를 설정하는 비트가 제공되는 경우
- 한 페이지의 NX 비트가 1로 셋 되어 있으면 그 페이지는 실행할 수 없는 페이지가 됨
  - PC가 NX 비트가 셋 되어 있는 페이지로 이동하면 트랩이 걸리도록 설정함
- 장점 : 소프트웨어 오버헤드가 발생하지 않음
- 단점 : 하드웨어에서 NX bit을 제공해야 함
  - 컴파일러도 NX bit을 염두에 두고 있어야 함

Incomplete mediation

- 입력으로 들어오는 값들의 범위와 길이를 충분히 검사하지 않는 것
- Buffer overflow
  - 입력의 크기가 buffer의 크기 이내인지 확인하지 않아서 발생하는 문제
- 조작된 web query
  - http://www.buyall.com/order/custom=100&goodid=1001&count=1&price=10000
    - 100번 고객이 1001번 상품을 1개 구입하고 그 가격이 10,000원임
  - http://www.buyall.com/order/custom=100&goodid=1001&count=10&price=10000
    - 고객의 query의 내용을 조작하여 보낸 경우

불완전 중재에 대한 대처

- 예외 처리 코드 : 프로그램내의 모든 입력 값에 예외 처리 코드를 포함시킴
- 상호 검증 방식 : 2명 이상이 서로 작성자가 놓친 예외 처리 부분이나 프로그램의 오류를 검토하고 찾아내는 방식
- 제 3 자 검토 : 다른 팀의 전문가에게 프로그램을 검토하고 오류를 지적하도록 요청
  - 일반 검토
  - 개략 검토
  - 면밀 검토
- 테스트 : 실제 수행하여 오류를 검출
  - 모듈 테스트
  - 컴포넌트 테스트
  - 단위 테스트 : 몇 개의 컴포넌트를 조합한 하나의 단위에 대한 테스트
  - 통합 테스트
  - (테스트용 데이터 생성도 중요한 연구분야들 중의 하나임)

불완전 중재에 대한 대처 II

- 목적에 따른 테스트
  - 기능 테스트 : 의도한 기능이 수행되는가를 테스트
  - 성능 테스트 : 요구사항만큼의 수치가 나오는 지를 테스트
  - 인수 테스트 : 고객을 참여시켜 고객 관점에서 수행하는 테스트
  - 설치 테스트 : 제품을 설치하는 과정에서 진행하는 테스트
  - 복귀 테스트 : 시스템을 패치 등으로 변경한 뒤 진행하는 테스트

테스트별 결함 발견 비율

- C. P. Pfleeger와 S. L. Pfleeger의 소개
- 184,000라인으로 이루어진 프로그램 코드에 대한 테스트

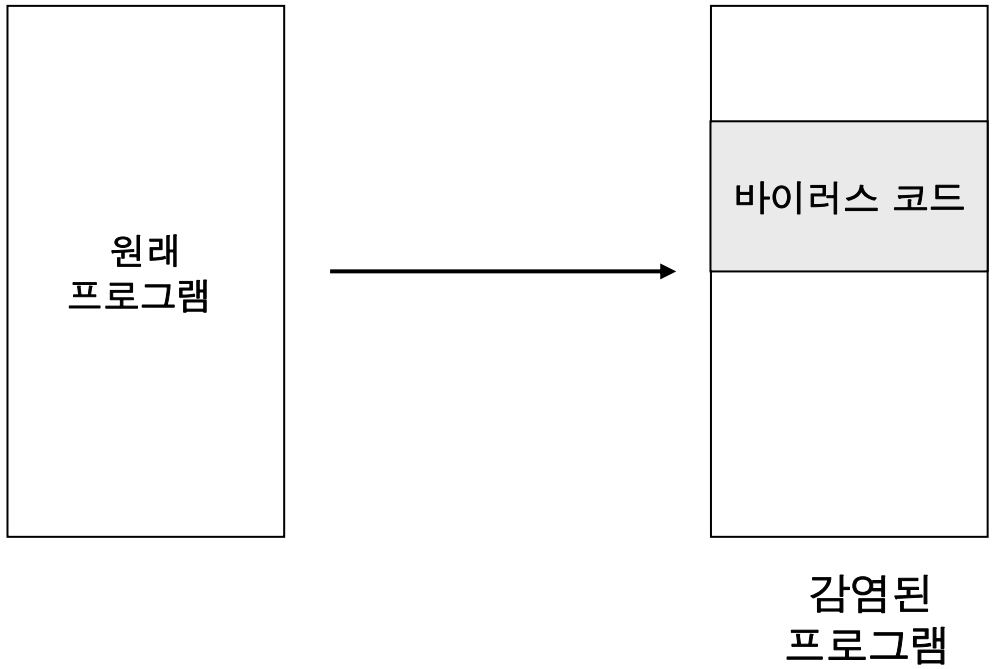
테스트 종류	결함 발견 비율
시스템 설계 검토	17.3%
컴포넌트 설계 검토	19.1%
코드 분석	15.1%
통합 테스트	29.4%
시스템 및 반환 테스트	16.6%

Malwares

- 특징에 따른 분류
  - Virus
  - Worm
  - Trojan horses
  - Backdoors
- 두 가지 이상의 특징을 동시에 포함하는 경우가 많다.

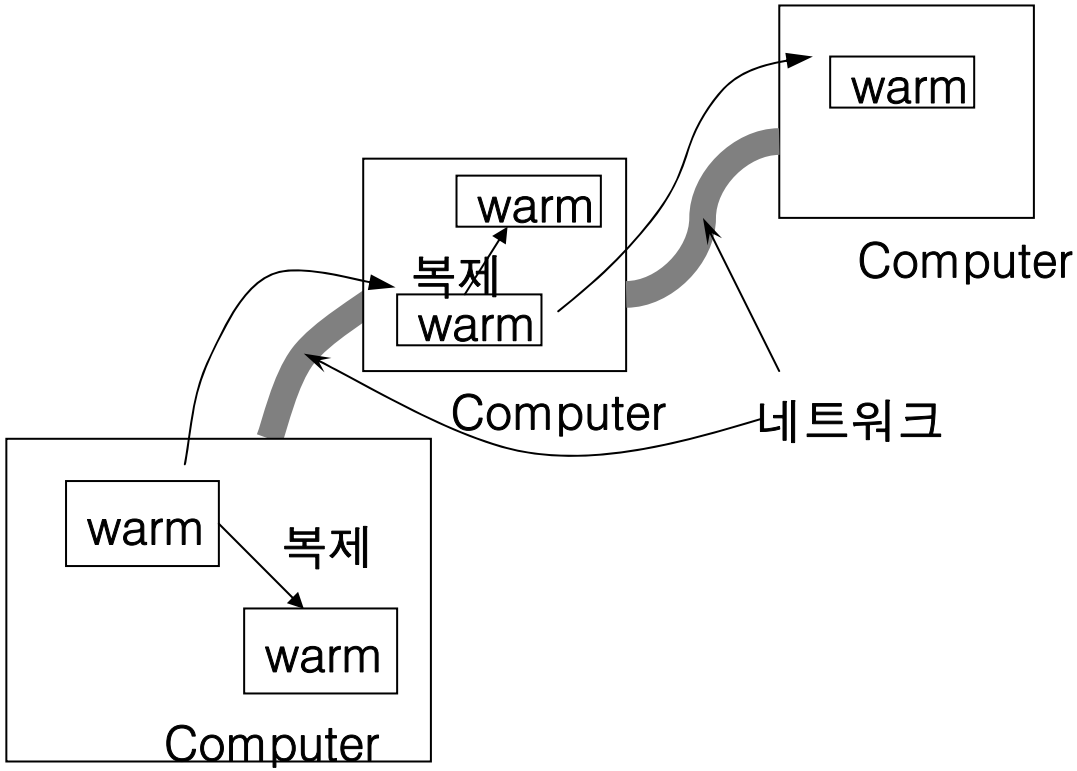
Virus

- 일반 프로그램에 삽입된 악의적인 코드, 다른 프로그램에게 자신의 코드를 삽입시킴



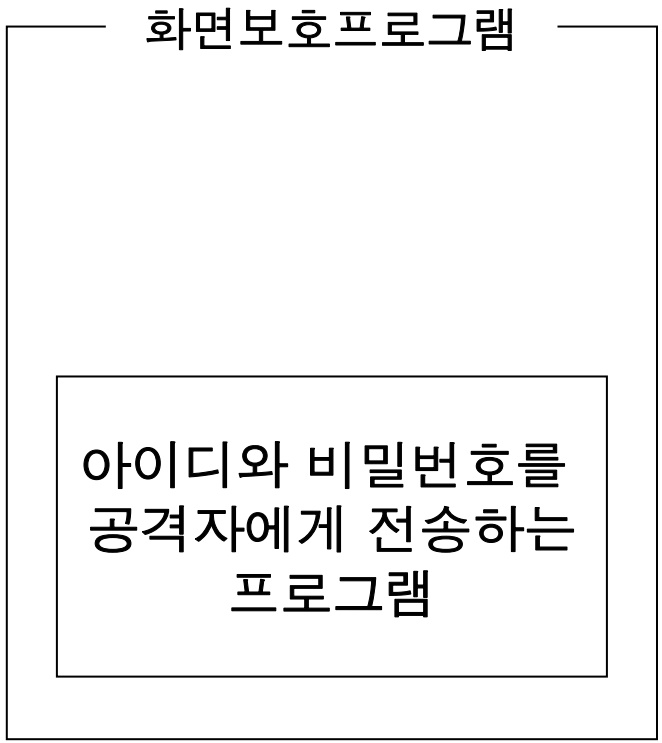
Worm

- 자기 자신의 현재 시스템이나 인터넷으로 연결된 다른 시스템으로 복제시킴



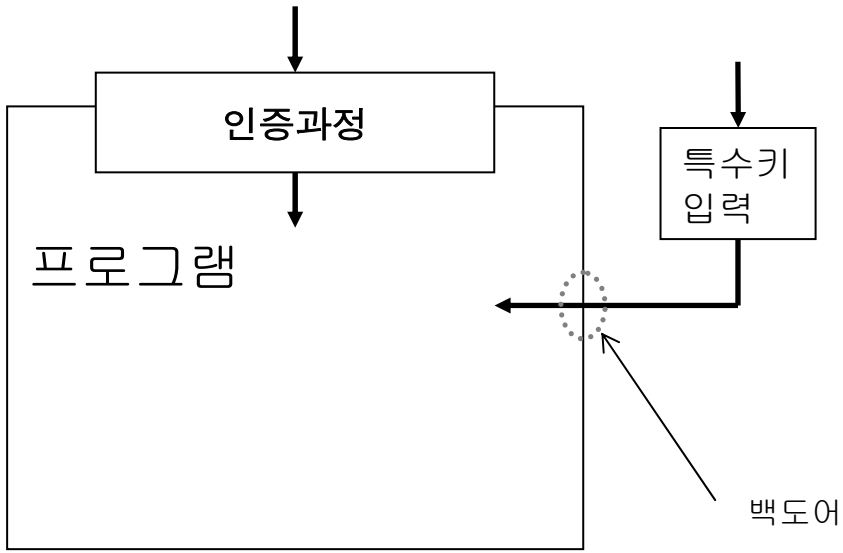
Trojan horses

- 정상적으로 보이는 프로그램 내에 악의적인 코드가 숨어 있는 형태



Backdoors

- 특정한 순서로 입력을 함으로써 인증 절차를 건너뛸 수 있도록 만든 코드
- Trapdoors으로도 불림



멀웨어의 위치

- 기존 파일의 일부를 자신의 코드로 덮어씀
- Boot sector에 자신의 코드 일부분을 저장함
  - 감염된 프로그램이 boot sector에 쓰기를 할 권한이 있는 경우
  - 그 프로그램이 실행될 때 malware code를 boot sector에 덮어 씀
  - 그 code는 interrupt handler를 변경하여 운영체제가 malware의 존재를 알지 못하도록 함
- 메모리 상에 존재
  - Malware에 감염된 프로그램이 실행되는 경우

대표적인 멀웨어들

- Brain virus
- Morris worm
- Code red
- SQL slammer

Brain Virus

- 1986년도에 등장 (멀웨어의 위험성이 인식되지 않던 시기)
- 특별히 해로운 작업을 수행하지는 않았음
- Boot sector와 파일에 숨어 실행되는 파일을 감염시킴
- 디스크 입출력 시마다 boot sector를 검사하여 자신의 코드가 없는 경우 다시 감염시킴

Morris Worm

- 1988년에 등장하여 많은 시스템들을 무력화시킴 (자원 고갈)
- 다양한 접근 방법을 사용 : 비밀번호 추측 온라인 공격, fingerd daemon에 대한 buffer overflow 공격, sandmail의 trapdoor 이용
  - Fingerd나 sandmail의 patch가 발표되었지만 적용되지 않은 시스템이 많았음
- Bootstrap loader (100줄 이하의 C 코드)를 공격대상에게 암호화 전송
- 자신을 잘 감춤 : 컴파일 / 실행한 후 전송된 소스코드를 삭제, process name / number를 수시로 바꿈

Code Red

- 2001년도에 등장
- 15시간 동안 250,000여 시스템을 감염
- 소멸되기 전까지 약 750,000개의 시스템 감염
- Microsoft IIS server에 대한 buffer overflow 공격
- 특정 기간에 미국 백악관 홈페이지에 대해 Distributed Denial of Service (DDoS) 발생

SQL Slammer

- 2004년도에 등장
- 10분동안 약 250,000개의 시스템을 감염
- 그 당시 방화벽의 필터링 대상이 아니었음
  - 크기가 376 bytes
- Microsoft SQL server를 공격하여 DDos 공격을 감행

Malware Detection

- Virus signature
  - 컴퓨터 바이러스의 특정 비트열 : 실행코드나 데이터
  - 정상적인 프로그램이 동일한 비트열을 가지면?
    - Virus signature가 128 bits라면 이러한 문제가 생길 확률은 2^{-128}임
  - Virus signature는 Virus signature DB에 저장
    - (+) Virus signature DB를 충실하게 갱신해줌으로써 많은 공격을 탐지할 수 있음
    - (-) 너무 커진 DB는 효율성을 떨어뜨림
    - (-) 약간만 변경한 malware라도 탐지하지 못함

Modification detection

- 프로그램 파일의 해시 (또는 전자서명)을 저장
- 주기적으로 프로그램 파일과 그 해시를 비교
- 장점
  - 컴퓨터 바이러스 관련 데이터베이스를 유지할 필요가 없음
- 단점
  - 해시 생성 및 비교의 오버헤드가 높음
  - 시스템 파일의 경우 정상적인 환경에서도 변경될 가능성이 높음
    - 변경 가능성이 희박한 파일들만 관리