Writing Safe Program Code

- 고급 언어는 일반적으로 컴파일 및 링크되어 기계어로 바뀐 후에 target processor에서 직접 실행됨
- 보안 문제 :
  - 올바른 알고리즘 구현
  - 알고리즘에 대한 올바른 기계어 변환
  - 데이터에 대한 유효한 조작

Correct Algorithm Implementation

- 좋은 프로그램을 개발하는 기법에 대한 문제
  - 알고리즘의 문제의 모든 경우를 올바르게 처리하지 못할 수 있음
  - 이러한 프로그램의 결점은 공격 당할 수 있는 bug가 됨
- 많은 TCP / IP 구현에 사용된 initial sequence number가 예측가능함
  - 패킷 식별자/인증자로 sequence number들을 묵시적으로 사용함.
  - 예측 가능한 식별자는 세션 공격을 가능하게 함
- 프로그래머가 테스트/디버깅을 돕기 위해 프로그램에 추가 코드를 의도적으로 포함시키는 경우
  - 종종 코드는 프로그램의 제품 버전에 남아 있음.
    → 정보를 부적절하게 공개할 수 있음
  - 이러한 코드는 보안 검사를 거치지 않고 수행할 수 없는 작업을 수행하도록 허용할 수 있습니다
  - sendmail의 이러한 취약점은 Morris Internet Worm에 의해 악용됨.

Ensuring Machine Language Corresponds to Algorithm

- 알고리즘과 구현된 기계어 간의 일치 문제
  - 대부분의 프로그래머는 이 문제를 무시함
  - 컴파일러 또는 인터프리터가 고급언어 문장을 올바르게 기계어 코드를 생성하거나 실행한다고 가정함
  - 컴파일러 최적화 요구 수준과 관련
- 원본 코드와 기계어 코드를 비교할 필요가 있음
  - 악의적인 컴파일러 작성자는 추가 코드를 삽입할 수도 있음
  - 비교 작업은 매우 느리고 어려운 작업임

Correct Data Interpretation

- 데이터의 올바른 해석에 대한 문제
- 데이터는 컴퓨터에서 binary bit들의 그룹으로 저장됨
  - 대개 byte 단위로 메모리에 저장됨
  - word(16bit) 또는 longword(32bit) 등의 큰 단위로 저장될 수 있음
  - 데이터는 메모리를 직접 접근하거나, CPU 레지스터로 복사한 후 사용함
  - 데이터에 대한 해석은 실행되는 기계어에 의해서 이루어짐
- 변수 데이터에 대한 해석의 제약과 검증을 위해 프로그래밍 언어 별로 다른 기능을 제공함
  - strongly typed languages – more limited, safer
  - other languages – 자유로운 해석 허용
    - 프로그램이 데이터의 해석을 명시적으로 변경하는 것을 허용.
    - (예) C언어의 정수와 메모리 주소(포인터) 간의 쉬운 변환

Correct Use of Memory

- 동적 메모리 할당에 대한 문제
  - 미리 크기를 알지 못하는 데이터를 조작하는 데 사용
  - 필요 시 할당, 사용 후 반환
- 메모리 누수(memory leak)
  - heap에서 사용 가능한 메모리가 지속적으로 감소하여 소진됨
  - 이 때에 비정상 종료될 수 있음
- 많은 예전의 언어(C 언어)들은 동적 메모리 할당에 대한 명시적 지원기능을 제공하지 않음
  - 메모리 할당과 반환을 위해 standard library routines을 사용
  - 동적 할당 메모리가 더 이상 필요하지 않는 시점을 결정하는 것이 어려움
- 현대의 언어(Java, C++)들은 메모리 할당과 반환을 자동적으로 처리함
  - 실행 부담은 있지만, 신뢰성이 더 있음

Race Conditions

- 경쟁 조건
  - 여러 프로세스/쓰레드들이 자원을 통제되지 않은 상태로 메모리 접근을 경쟁하는 상황
- 메모리 접근에 대한 동기화가 없다면,
  - 공유 값을 접근, 사용, 변경하는 것으로 인해서 데이터 값이 손상되거나, 변경한 내용이 손실될 수 있음.
- 해결책
  - 병행코드를 작성할 때에 적절한 동기화 primitive 함수를 올바르게 선택하여 사용함.
- 교착상태(deadlock)
  - 프로세스/쓰레드들이 다른 것이 점유한 자원을 기다림
  - 교착상태 발생 시에, 하나 이상의 프로그램을 종료시킴
  - 공격자가 서비스거부 공격을 위해 교착상태를 유발할 수 있음

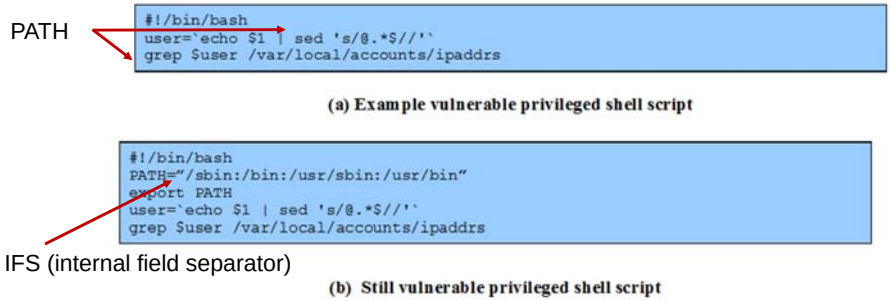
Operating System Interaction

- 프로그램은 운영체제의 제어 하에 있는 시스템에서 실행됨
  - 자원 접근을 조정하고 공유함
  - 실행 환경을 구성함
  - 환경변수와 인수를 포함
- 다중 사용자 개념을 지원함
  - 자원(resources)은 user가 공유하고, 다른 범주의 사용자에게 여러 접근 권한을 부여하는 허가권을 가짐
  - programs은 여러 자원에 대한 접근이 필요함
    - 그렇지만 과도한 수준의 접근은 위험함
  - 여러 프로그램이 common file과 같은 공유 자원을 접근할 때에 문제가 발생할 수 있음

Environment Variables

- 환경 변수 – 부모로부터 상속받은 문자열 값의 집합
  - 실행 중인 프로세스 동작에 영향을 줄 수 있음.
  - 프로세스 생성 시에 메모리에 환경변수가 포함됨
- 환경변수는 프로그램이 언제든지 수정할 수 있음
  - 수정된 값은 children으로 전달됨
- 신뢰할 수 없는 프로그램 입력의 또 하나의 source가 됨
  - 가장 일반적인 사용은 local user가 증가된 권한을 얻고자 시도하는 것임 (예) 관리자 권한 획득 시도

Vulnerable Shell Script Example



- 환경변수 PATH 변경으로 같은 이름의 다른 유틸리티 실행 가능
- 환경변수 IFS – 명령어 행 구분자 변경 가능
  - '='를 IFS로 지정한다면 PATH를 기본값으로 재지정하는 문장이 실행되지 않음


연세대학교

Vulnerable Compiled Programs

- 프로그램은 PATH 환경변수 조작에 취약할 수 있음
  - 중요한 환경변수를 "안전한" 값으로 초기화 해야 함
- 환경변수 LD_LIBRARY_PATH
  - 동적 라이브러리 함수를 찾기 위한 디렉토리 목록을 명시
    - 적절한 동적 라이브러리 함수를 찾는 데 사용
  - 동적 링크 라이브러리는 융통성을 제공하지만 공격에 취약할 수 있음
  - 해결책
    - 정적 링크 라이브러리 함수 사용, 또는
    - 환경 변수의 사용 금지 등.


연세대학교

Use of Least Privilege

- 권한 상승(privilege escalation)
  - 공격자는 결함을 이용하여 더 높은 권한을 가질 수 있음
  - 더 높은 권한을 가진 공격자는 시스템을 변경할 수 있음
- 최소 권한 원칙(principle of least privilege)
  - 자신의 작업을 완수하는 데 필요한 최소 권한으로 프로그램을 실행해야 함.
- 프로그램을 실행할 때 요구되는 적절한 user 및 group privilege를 결정해야 함 – 특히 특권이 있는 프로그램에 대해서
  - 사용자 추가 또는 그룹 권한 부여 등을 결정
- 특권이 있는 프로그램이 필요한 파일과 디렉토리만 수정할 수 있도록 보장해야 함
  - 흔히 발견되는 결함은 모든 파일과 디렉토리에 대한 소유권을 갖는다는 것임.
  - (예) 사용자의 웹 문서 파일 소유권


연세대학교

Root/Administrator Privileges

- root/관리자 권한을 갖는 프로그램이 공격자의 주요 공격대상임
  - 높은 수준의 시스템 접근 및 제어를 제공
  - 시스템 자원을 보호하기 위해 접근을 관리할 필요가 있음
- 이러한 프로그램에서 특권은 대개 시작할 때만 필요함
  - 그 후에는 일반 사용자로 실행할 수 있음
- 좋은 설계는 복잡한 프로그램을 필요한 권한을 가진 더 작은 모듈로 나누어야 함.
  - 구성 요소 간에 높은 수준의 분리(isolation)를 제공
  - 한 구성요소의 보안 허점이 미치는 영향을 감소시킴
  - 테스트 및 검증 용이
- chroot 기능
  - 프로그램이 파일시스템의 일부분만 사용할 수 있도록 root를 변경시킴 → chroot jail
  - 프로그램이 공격 당해도 제한된 파일시스템 영역만 영향 가능


연세대학교

System Calls and Standard Library Functions

- 프로그램은 공통된 작업을 위해 system calls과 standard library 함수를 사용함
  - 프로그래머는 이러한 함수에 대해서 예상대로 동작한다고 가정을 하고 사용함
- 그렇지만, 예상대로 동작하지 않을 수 있음
  - 시스템은 성능을 최적화하는 방식으로 자원 사용을 관리함
    - 공유자원 서비스 요청이 버퍼에 넣어져서
    - 접근 순서를 변경하거나, 서비스 요청이 수정됨
  - 시스템 최적화가 프로그램 목표와 상충될 수 있음
- (예) 안전한 파일 삭제(secure file shredding) 프로그램 설계
  - 파일 링크만 삭제하지 않고 복구되지 않도록 데이터 overwrite
  - write 모드 open – 새로운 block에 쓰기를 할 수 있음.
    - 실제로 기존 데이터의 overwrite가 이루어지지 않을 수 있음
  - write 순서
    - 응용프로그램 buffer → 시스템 buffer → device


연세대학교

Secure File Shredder

```
patterns = [10101010, 01010101, 11001100, 00110011, 00000000, 11111111, … ]
open file for writing
for each pattern
   seek to start of file
   overwrite file contents with pattern
close file
remove file
```

(a) Initial secure file shredding program algorithm

```
patterns = [10101010, 01010101, 11001100, 00110011, 00000000, 11111111, … ]
open file for update
for each pattern
   seek to start of file
   overwrite file contents with pattern
   flush application write buffers
   sync file system write buffers with device
close file
remove file
```

(b) Better secure file shredding program algorithm


연세대학교

Preventing Race Conditions

- 공통의 시스템 자원 접근을 하는 프로그램들은 적절한 synchronization mechanisms이 필요하다.
  - (예) 사용자 메일박스 – 메일 클라이언트, 메일 전송 프로그램
  - 가장 많이 사용하는 방법: 공유 자원에 대해서 lock 사용
- lockfile
  - 프로세스가 공유자원을 접근 권한 획득을 위해서 lockfile을 생성하고 소유함
  - 고려사항
    - 프로그램이 lockfile의 존재를 무시하고 공유자원을 접근하려고 하면 시스템은 이를 막지 못함. (프로그램 책임)
    - 이 방식의 동기화는 공유자원을 사용하는 모든 프로그램이 협력해야 함.
    - 구현 문제 – atomic operation
      - lock file의 존재 검사와 생성이 중단 없이 함께 수행되어야 함


연세대학교

Perl File Locking Example

```
#!/usr/bin/perl
#
$EXCL_LOCK = 2;
$UNLOCK    = 8;
$FILENAME  = "forminfo.dat";

# open data file and acquire exclusive access lock
open (FILE, ">> $FILENAME") || die "Failed to open $FILENAME \n";
flock FILE, $EXCL_LOCK;
… use exclusive access to the forminfo file to save details
# unlock and close file
flock FILE, $UNLOCK;
close(FILE);
```

- 권고 lock – 프로그램에서 필요 시에 사용
  - flock 함수 – advisory lock을 적용하거나 제거함
    flock(fd, operation)
- 필수 lock – 운영체제에서 구현


연세대학교

Safe Temporary Files

- 많은 프로그램들이 temporary file을 사용함
  - 흔히 잘 알려진 공유 시스템 공간에 저장함
  - 파일 이름이 unique하고, 다른 프로세스가 접근하지 않아야 함
- process ID를 사용한 파일 이름 부여
  - unique, but predictable
  - 공격자는 프로그램 검사와 생성 사이에 자신이 추정한 이름의 파일 생성을 시도하고, 그 이름으로 password 파일에 대한 심볼릭 링크를 생성하는 script를 만듬.
  - 특권이 있는 프로그램이 이 임시파일을 삭제한다면 문제가 발생
- 안전하게 temporary file을 생성하고 사용하려면
  추정이 불가능하도록 파일 이름을 random하게 부여해야 함
- 임시 디렉토리는 sticky 허가 비트 설정
  - 임시 파일 소유자와 관리자만 파일 삭제 가능

Temporary File Creation Example

- 임시파일 생성도 atomic 시스템 함수를 이용하는 것이 좋음
  - mkstemp() 함수는 안전
  - tempname(), tmpfile() 등은 조심해서 사용해야 함
- 임시파일 생성 시에 제한적인 파일 생성 플래그를 사용

```
char *filename;
int fd;
do {
  filename = tempnam (NULL, "foo");
  fd = open (filename, O_CREAT | O_EXCL | O_TRUNC | O_RDWR, 0600);
  free (filename);
} while (fd == -1);
```

  - 임시파일이름의 파일이 정상적으로 오픈될 때까지 반복 시도

Other Program Interaction

- 프로그램은 운영체제/표준 라이브러리 이외에
  다른 프로그램의 함수와 서비스를 이용할 수도 있음
  - 상호작용에 주의를 기울이지 않으면 보안 취약점이 발생가능함
    - 전송되는 데이터 크기 및 해석에 대한 가정 확인 필요
  - 사용되는 프로그램이 다양한 사용을 고려하지 않아서 모든 보안 문제를 적절히 다루지 않았을 때에 발생
    - 특히 웹 프로그램에서 많이 발생
    - 예전 프로그램을 이용하는 새로운 프로그램들이 발생 가능한 보안 이슈들을 찾아내어 관리 해야 할 책임이 있음.
- 데이터 기밀성 및 무결성 문제
  - 네트워크 연결은 보안 프로토콜 사용 – IPsec, SSL/TLS, ssh 등
- 상호작용에서 발생한 예외 및 오류를 감지하고 처리하는 것은 보안 측면에서 중요함

Handling Program Output

- 프로그램 출력은
  - 나중에 사용하려고 파일/데이터베이스에 저장하거나,
  - 네트워크로 전송하거나
  - 사용자에게 출력(display)할 수 있음
- 출력 데이터 유형
  - binary – 복잡한 구조체(X 윈도우 메시지, 네트워크 프로토콜) 등
  - text – 문자 집합 인코딩, html 형식 문서 등.
- 프로그램 보안 관점에서 출력이 예상되는 형식 및 해석과 일치해야 하는 것이 중요함
- 프로그램은 허용되는 출력 content가 무엇인지 확인하고, 신뢰할 수 없는 데이터를 필터링하여 유효한 출력만 표시되게 해야 함
  - html markup 제거, 안전한 markup만 허용 등.
- 출력에 사용할 문자 집합을 지정해야 함
  - 웹 문서는 헤더에 명시적 지정이 가능
  - 지정되지 않으면 웹 브라우저는 기본 문자집합을 가정함