발 간 등 록 번 호
11-1311000-000330-10 부록

전자정부 SW 개발 · 운영자를 위한

C 시큐어코딩 가이드

2012. 9.

행정자치부

제·개정이력(Revision History)

순번	제·개정일	변경내용	제·개정 기관	비고
1	2011년 6월	[제정] C 시큐어코딩 가이드 개발	행정자치부(KISA)	1판
2	2011년 9월	[개정] 보안약점 내용 및 예제코드 수정	행정자치부(KISA)	2판
2	2012년 9월	[개정] SW 개발보안 가이드 개정('12.6월)에 따른 보안약점 항목 및 내용 변경	행정자치부(KISA)	3판

Contents

<표 1> 보안약점 유형별 C 언어 관련 보안약점 목록

유형	항목	보안약점 명칭	CWE
입력 데이터 검증 및 표현	1	SQL 삽입	89
	2	자원 삽입	99
	3	크로스사이트스크립트	79
	4	운영체제 명령어 삽입	78
	5	LDAP 삽입	90
	6	디렉터리 경로 조작	22(23,36)
	7	정수 오버플로우	190
	8	보호 메커니즘을 우회할 수 있는 입력값 변조	807
	9	스택에 할당된 버퍼 오버플로우	121
	10	힙에 할당된 버퍼 오버플로우	122
	11	LDAP 처리	90
	12	시스템 또는 구성 설정의 외부 제어	15
	13	프로세스 제어	114
	14	버퍼 시작 지점 이전에 쓰기	124
	15	범위 초과해서 읽기	125
	16	검사되지 않은 배열 인덱싱	129
	17	널 종료 문제	170
	18	의도하지 않은 부호 확장	194
	19	무부호 정수를 부호 정수로 타입 변환 오류	196
보안기능	1	부적절한 인가	285
	2	중요한 자원에 대한 잘못된 권한허용	732
	3	취약한 암호화 알고리즘의 사용	327
	4	사용자 중요정보 평문(또는 전송)	311
	5	하드코드된 패스워드	259
	6	충분하지 못한 키 길이 사용	310
	7	적절하지 않은 난수 값의 사용	330
	8	패스워드 평문 저장	256
	9	하드코드된 암호화키	321
	10	주석문 안에 포함된 패스워드 등 시스템 주요정보	615
	11	솔트 없이 일방향 해쉬 함수 사용	759
	12	하드코드된 사용자 계정	255
	13	잘못된 권한 부여	266
	14	최소 권한 적용 위배	272
	15	취약한 암호화: 적절하지 못한 RSA 패딩	325
	16	취약한 암호화 해쉬함수: 하드코드된 솔트	326
	17	같은 포트번호로의 다중 연결	605
시간 및 상태	1	경쟁 조건: 검사시점과 사용시점(TOCTOU)	367
	2	제대로 제어되지 않은 재귀	674
	3	심볼릭명이 정확한 대상에 매핑되어 있지 않음	386

유형	항목	보안약점 명칭	CWE
에러 처리	1	오류 메시지 통한 정보 노출	209
	2	오류상황 대응 부재	390
	3	적절하지 않은 예외처리	754
코드 오류	1	널(Null)포인터 역참조	476
	2	부적절한 자원 해제	404
	3	부호 정수를 무부호 정수로 타입 변환 오류	195
	4	정수를 문자로 변환	398
	5	스택 변수 주소 리턴	562
	6	매크로의 잘못된 사용	730
	7	코드정확성: 스택 주소 해제	730
	8	코드 정확성: 스레드 조기 종료	730
	9	무한 자원 할당	770
캡슐화	1	제거되지 않고 남은 디버그 코드	489
	2	시스템 데이터 정보노출	497
API 오용	1	DNS lookup에 의존한 보안결정	247
	2	위험하다고 알려진 함수 사용	242
	3	작업 디렉터리 변경 없는 chroot Jail 생성	243
	4	오용:문자열 관리	251
	5	다중 스레드 프로그램에서 getlogin() 사용	558

제1장 개요

제1절 배경

'**소프트웨어(SW) 개발보안**'은 SW 개발과정에서 개발자 실수, 논리적 오류 등으로 인해 SW에 내포될 수 있는 보안취약점(vulnerability)의 원인, 즉 보안약점(weakness)을 최소화하는 한편, 사이버 보안위협에 대응할 수 있는 안전한 SW를 개발하기 위한 일련의 보안활동을 의미한다. 광의적 의미로는 SW 개발생명주기(SDLC, Software Development Lifecyle)의 각 단계별로 요구되는 보안활동을 모두 포함하며, 협의적 의미로는 SW 개발과정 중 소스코드 구현단계에서 보안약점을 배제하기 위한 '**시큐어코딩(secure coding)**'을 의미한다.

SW 개발보안의 중요성을 인식한 미국의 경우, 국토안보부(DHS)를 중심으로 시큐어코딩을 포함한 SW 개발 전과정(설계·구현·시험 등)에 대한 보안활동 연구를 활발히 진행하고 있으며, 이는 2011년 11월에 발표한 "안전한 사이버 미래를 위한 청사진(blueprint for a secure cyber future)"에도 나타나 있다. 또한, DHS는 국립표준기술연구소(NIST)를 통해 각 단계별 보안활동 및 절차를 표준화하여 연방정부에서 정보시스템 구축·운영 시 참고하도록 하고 있다.

국내의 경우, 2009년부터 전자정부서비스 개발단계에서 SW 보안약점을 진단하여 제거하는 SW 개발보안(시큐어코딩) 관련 연구를 진행하면서, 2012년까지 전자정부지원사업 등을 대상으로 SW 보안약점 시범진단을 수행하였다. 이러한 SW 보안약점 제거·조치 성과에 따라 **'행정기관 및 공공기관 정보시스템 구축·운영 지침'이 개정·고시**됨으로써 전자정부서비스 개발시 적용토록 의무화 되었다.

본 가이드는 정보시스템 구축 시 많이 사용되는 개발언어 중 하나인 **C 기반의 시큐어코딩 기법을 예제 위주로 제시**함으로써, 개발 실무에 활용도를 높이고자 작성되었다.

제2절 가이드 목적 및 구성

목적

· '행정기관 및 공공기관 정보시스템 구축·운영 지침'에 따라 전자정부 관련 정보화사업 수행시, **안전한 SW 개발을 위한 C 기반 시큐어코딩 기법 제시**

※ 신규로 개발되거나 유지보수로 변경되는 소스코드에 적용

활용

· (개발자) C 기반 시큐어코딩을 적용한 **SW 개발시 참조**

· (발주자) SW 보안약점 진단·제거 **요구사항 도출시 참조**

· (기타) C 기반 보안약점 및 대응기법 등 **전반에 대한 이해**

구성

· C 기반으로 정보시스템 개발시, 고려해야할 **보안약점(58개) 설명**과 보안대책 이해를 위한 **코딩 예제(Bad/Good)를 제시**

※ '정보시스템 구축·운영 지침'에서 제시한 43개外 C 언어 특성을 추가 반영

유형	내용
입력데이터 검증 및 표현	프로그램 입력값에 대한 검증 누락 또는 부적절한 검증, 데이터의 잘못된 형식지정으로 인해 발생할 수 있는 보안약점 ※ SQL 삽입, 자원 삽입 등 19개
보안기능	보안기능(인증, 접근제어, 기밀성, 암호화, 권한관리 등)을 적절하지 않게 구현시 발생할 수 있는 보안약점 ※ 부적절한 인가, 중요한 자원에 대한 잘못된 권한허용 등 17개
시간 및 상태	동시 또는 거의 동시 수행을 지원하는 병렬 시스템, 하나 이상의 프로세스가 동작하는 환경에서 시간 및 상태를 부적절하게 관리하여 발생할 수 있는 보안약점 ※ 경쟁조건, 제대로 제어되지 않는 재귀 등 3건
에러처리	에러를 처리하지 않거나, 불충분하게 처리하여 에러정보에 중요정보(시스템 등)가 포함될 때 발생할 수 있는 보안약점 ※ 오류 메시지 통한 정보 노출, 오류상황 대응 부재 등 3개
코드오류	타입변환 오류, 자원(메모리 등)의 부적절한 반환 등과 같이 개발자가 범할 수 있는 코딩오류로 인해 유발되는 보안약점 ※ 널(Null)포인터 역참조, 부적절한 자원 해제 등 9개
캡슐화	중요한 데이터 또는 기능성을 불충분하게 캡슐화하였을 때, 인가되지 않는 사용자에게 데이터 누출이 가능해지는 보안약점 ※ 제거되지 않고 남은 디버그 코드, 시스템 데이터 정보노출 등 2개
API 오용	의도된 사용에 반하는 방법으로 API를 사용하거나, 보안에 취약한 API를 사용하여 발생할 수 있는 보안약점 ※ DNS Lookup에 의존한 보안결정, 위험하다고 알려진 함수 사용 등 5건

제2장 시큐어코딩 가이드

제1절 입력데이터 검증 및 표현

사용자의 입력을 검증하지 않고 그대로 받아들여 사용하면 많은 보안위협에 노출된다. 해당 보안취약점을 예방하기 위해서는 유효한 입력데이터만 허용할 수 있도록 코딩하는 것이 좋으며, 부득이한 경우 입력값을 검증하여 검증된 데이터만 허용하도록 코딩하여 취약점을 제거해야 한다.

1. SQL 삽입(Improper Neutralization of Special Elements used in an SQL Command, SQL Injection)

가. 정의

데이터베이스(DB)와 연동된 웹 어플리케이션에서 입력된 데이터에 대한 유효성 검증을 하지 않을 경우, 공격자가 입력 폼 및 URL 입력란에 SQL 문을 삽입하여 DB로부터 정보를 열람하거나 조작할 수 있는 보안약점을 말한다. 취약한 웹 어플리케이션에서는 사용자로부터 입력된 값을 필터링 과정없이 넘겨받아 동적 쿼리(Dynamic Query)[1]를 생성한다. 이는 개발자가 의도하지 않은 쿼리가 생성되어 정보유출에 악용될 수 있다.

나. 안전한 코딩기법

- 외부 입력이나 외부 변수로부터 받은 값이 직접 SQL 함수의 인자로 전달되거나, 문자열 복사를 통하여 전달되는 것은 위험하다. 그러므로 인자화된 질의문을 사용하여야 한다.
- 외부 입력값을 그대로 사용해야 하는 환경이라면, 입력받은 값을 필터링을 통해 처리한 후 사용해야 한다. 필터링은 SQL 문에서 사용하는 단어 사용 금지, 특수문자 사용금지, 길이 제한의 기준을 적용한다.

다. 예제

외부 입력이 SQL 질의어에 어떠한 처리도 없이 삽입되었다. **name' OR 'a'='a** 와 같은 문자열을 입력할 경우 WHERE 절이 항상 참이 된다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdlib.h>
2:  #include <sql.h>
3:  void Sql_process(SQLHSTMT sqlh)
4:  {
5:    char *query = getenv("query_string");
6:    SQLExecDirect(sqlh, query, SQL_NTS);
7:  }
```

1) 동적쿼리(Dynamic Query) : DB에서 실시간으로 받는 쿼리. parameterized statement가 동적쿼리가 됨

인자화된 질의를 사용함으로써 질의 구조의 변경을 방지할 수 있다.

■ 안전한 코드의 예 - C

```
1:  #include <sql.h>
2:  void Sql_process(SQLHSTMT sqlh)
3:  {
4:    char *query_items = "SELECT * FROM items";
5:    SQLExecDirect(sqlh, query_items, SQL_NTS);
6:  }
```

다음의 예에서는 **queryStr**로부터 받아온 외부 입력에서 **user_id**와 **password**에 해당하는 값을 잘라온 후 SQL문의 인자값으로 그대로 사용하였다. **user_id**에 해당하는 입력값으로 name' OR 'a'='a'--와 같은 문자열을 주면 WHERE 절이 항상 참이 되어 **user_id**에 상관없이 인증부분을 통과할 수 있게 된다.

*(**SELECT * FROM members WHERE username ='name' OR 'a'='a'-- AND password = ''**)*

■ 안전하지 않은 코드의 예 - C

```
1:  static SQLHSTMT statmentHandle;
2:  const char * GetParameter(const char * queryString, const char * key);
3:  static const char * GET_USER_INFO_CMD = "get_user_info";
4:  static const char * USER_ID_PARAM = "user_id";
5:  static const char * PASSWORD_PARAM = "password";
6:  static const int MAX_QUERY_LENGTH = 256;
7:  const int EQUAL = 0;
8:  int main(void)
9:  {
10:   SQLCHAR * queryStr;
11:   queryStr = getenv("QUERY_STRING");
12:   if (queryStr == NULL)
13:   {
14:     // Error 처리루틴
15:     ...
16:   }
17:   // 입력값을 가져온다.
18:   char * command = GetParameter(queryStr, "command");
19:   if (strcmp(command, GET_USER_INFO_CMD) == EQUAL)
20:   {
21:     // userId 와 password값을 가져온다.
22:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
23:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
24:
25:     char query[MAX_QUERY_LENGTH];
26:     sprintf(query, "SELECT * FROM members WHERE username= '%s' AND password =
```

모든 user_Id와 password가 참이 된다

```
    '%s'", userId, password);
27:     SQLExecDirect(statmentHandle, query, SQL_NTS);
28:   }
29:   return 0;
30: }
```

다음의 예제는 http request로부터 추출한 사용자 ID와 암호를 **makeSecureString**함수를 이용하여 위험한 SQL구문 생성에 쓰일 수 있는 단어들과 특수문자를 제거하였다. 또한 공격 구문을 사용자 ID와 암호에 추가하기 힘들도록, 사용자 ID와 암호의 길이를 제한하였다.

static const int로 선언한 **MAX_USER_ID_LENGTH**와 **MAX_PASSWORD_LENGTH**에 설정해 놓은 값만큼의 문자를 제외한 뒷부분은 제거된다. 두 개의 상수값을 조정함으로써 제한 강도를 조절할 수 있다. 그 후에 **regexec** 함수를 통해 정규식에 매칭된 위험한 단어와 문자들을 찾아낸 후, 그 부분을 건너뛰어 뒤쪽의 스트링을 앞쪽의 스트링과 연결하는 방식으로 안전한 입력값을 생성한다.

필터링은 정규식(regular expression)을 사용하였으며, **"[^[:alnum:]]|select|delete|update|insert|create|alter|drop"** 설정을 통해 알파벳과 숫자외의 특수문자들과 select, delete등의 인젝션에 사용할 가능성이 높은 단어 7가지를 필터링 대상으로 설정하였다. 필터링할 단어의 종류를 늘리는 강도 조절은 정규식에 단어를 추가하는 것으로 조정이 가능하다. 보다 정밀한 방어를 위해서는 악용 가능성이 있는 SQL procedure명이나 SQL 명령어들을 필터링할 정규식에 포함시키면 된다.(블랙리스트 개념으로 작동한다)

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  static const char * GET_USER_INFO_CMD = "get_user_info";
3:  static const char * USER_ID_PARAM = "user_id";
4:  static const char * PASSWORD_PARAM = "password";
5:  static const int MAX_QUERY_LENGTH = 256;
6:  static const int MAX_USER_ID_LENGTH = 8;
7:  static const int MAX_PASSWORD_LENGTH = 16;
8:  const char * makeSecureString(const char *str, int maxLength);
9:  const int EQUAL = 0;
10: int main(void)
11: {
12:   // 필터링에 사용할 정규식을 설정한다.
13:   int reti = regcomp(&unsecurePattern, "[^[:alnum:]]|select|delete|update|in-
    sert|create|alter|drop", REG_ICASE | REG_EXTENDED);
14:
15:   // 정규식 설정이 실패했을 경우 나머지 처리를 하지 않고 강제 종료한다.
16:   if (reti)
17:   {
18:     fprintf(stderr, "Could not compile regex\n");
```

```
19:       exit(1);
20:     }
21:
22:     SQLCHAR * queryStr;
23:     queryStr = getenv("QUERY_STRING");
24:     if (queryStr == NULL)
25:     {
26:       // 입력값이 null일 경우 Error 처리한다.
27:       ...
28:     }
29:     char * command = GetParameter(queryStr, "command");
30:     if (strcmp(command, GET_USER_INFO_CMD) == EQUAL)
31:     {
32:       // 각 입력값을 쪼갠 후 makeSecureString함수로 필터링을 거쳐 검증한다.
33:       const char * userId = GetParameter(queryStr, USER_ID_PARAM);
34:       userId = makeSecureString(userId, MAX_USER_ID_LENGTH);
35:       const char * password = GetParameter(queryStr, PASSWORD_PARAM);
36:       password = makeSecureString(password, MAX_PASSWORD_LENGTH);
37:
38:       char query[MAX_QUERY_LENGTH];
39:       sprintf(query, "SELECT * FROM members WHERE username= '%s' AND password =
    '%s'", userId, password);
40:       SQLExecDirect(statmentHandle, query, SQL_NTS);
41:
42:       free(userId);
43:       free(password);
44:     }
45:     regfree(&unsecurePattern);
46:
47:     return EXIT_SUCCESS;
48:   }
49:
50:   // 입력값을 필터링하여 검증하는 루틴이다.
51:   const char * makeSecureString(const char *str, int maxLength)
52:   {
53:     char * buffer = (char *) malloc(maxLength + 1);
54:     char * originalStr = (char *)malloc(maxLength + 1);
55:     strncpy(originalStr, str, maxLength);
56:     originalStr[maxLength] = NULL;
57:     regmatch_t mt;
58:     const char * currentPos = originalStr;
59:     //정규식에 매칭되는 부분이 있으면 그 부분은 건너뛰는 형태로 문자열을 바꾼다.
60:     while (regexec(&unsecurePattern, currentPos, 1, &mt, REG_NOTBOL) == 0)
61:     {
62:       strncat(buffer, currentPos, mt.rm_so);
```

```
63:         currentPos += mt.rm_eo;
64:     }
65:     strcat(buffer, currentPos);
66:     free(originalStr);
67:     return buffer;
68: }
```

라. 참고문헌

[1] CWE-89 Improper Neutralization of Special Elements used in an SQL Command(SQL Injection), http://cwe.mitre.org/data/definitions/89.html

[2] 2010 OWASP Top 10 - A1 Injection, https://www.owasp.org/index.php/Top_10_2010-A1

[3] 2011 SANS Top 25 RANK 1 (CWE-89), http://cwe.mitre.org/top25/

2. 자원 삽입(Improper Control of Resource Identifiers, Resource Injection)

가. 정의

외부 입력값을 검증하지 않고 시스템 자원(resource)에 대한 식별자로 사용하는 경우, 공격자는 입력값 조작을 통해 시스템이 보호하는 자원에 임의로 접근하거나 수정할 수 있다.

나. 안전한 코딩기법

외부의 입력을 자원(파일, 소켓의 포트 등) 식별자로 사용하는 경우, 적절한 검증과정을 수행하도록 하거나 사전에 정의된 적합한 리스트에서 선택되도록 작성한다. 외부의 입력이 파일명인 경우에는 경로순회(directory traversal)를 수행할 수 있는 문자를 제거한다.

다. 예제

다음의 예제에서 소켓을 생성하는데 있어 포트번호를 외부 입력인 getenv("rPort")를 사용하였다. 공격자가 포트 번호를 마음대로 하여 소켓을 생성할 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  int main()
2:  {
3:     char* rPort = getenv("rPort");
4:     struct sockaddr_in serv_addr;
5:     int sockfd = 0;
6:     ...
7:     serv_addr.sin_port = htons(atoi(rPort));
8:     if (connect(sockfd, &serv_addr ,sizeof(serv_addr)) < 0) {
9:        exit(1);
10:    }
11:    return 0;
12: }
```

다음의 예제와 같이 외부 입력값에 따라 미리 설정된 포트를 선택하도록 함으로써 공격자가 임의의 포트를 사용하지 못하도록 한다.

■ 안전한 코드의 예 - C

```
1:  int main()
2:  {
3:     char* rPort = getenv("rPort");
4:     struct sockaddr_in serv_addr;
5:     int sockfd = 0;
6:     int port = 0;
7:     ...
8:     if(strcmp(rPort,"") < 0)
9:     {
```

```
10:         printf("bad input");
11:     }
12:     port = atoi(rPort);
13:     if( port < 3000 || port > 3003)
14:     {
15:     port = 3000;
16:     }
17:       serv_addr.sin_port = htons(atoi(rPort));
18:     if (connect(sockfd, &serv_addr ,sizeof(serv_addr)) < 0) {
19:       exit(1);
20:     }
21:     return 0;
22: }
```

다음의 예제에서는 접속할 ip와 포트번호를 외부 입력으로 받아 소켓으로 서버에 접속하고 있다. 하지만 특정한 서버의 주소 외의 값이 입력되면 그대로 프로그램이 종료되도록 설정되어 있다.

■ 안전한 코드의 예 - C

```
1:  int main(int argc, char **argv)
2:  {
3:      int serv_sock;
4:      struct sockaddr_in serv_addr;
5:
6:      serv_sock = socket(PF_INET, SOCK_STREAM, 0);
7:      memset(&serv_addr, 0, sizeof(serv_addr));
8:      serv_addr.sin_family = AF_INET;
9:
10:     if(!strcmp(argv[0],"127.0.0.1")) return;
11:     serv_addr.sin_addr.s_addr=inet_addr(argv[0]);
12:     serv_addr.sin_port = htons(atoi(argv[1]));
13:
14:     if(bind(serv_sock, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) == -1)
15:         error_handling("bind() error");
16: ...
```

라. 참고문헌

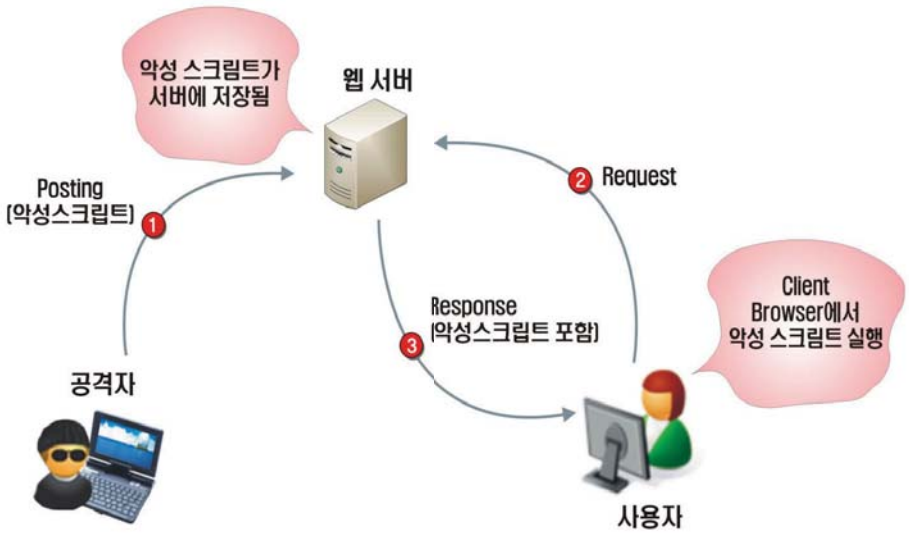
[1] CWE-99 Improper Control of Resource Identifiers (Resource Injection), http://cwe.mitre.org/data/definitions/99.html

[2] 2010 OWASP Top 10 - A4 Insecure Direct Object Reference https://www.owasp.org/index.php/Top_10_2010-A4

3. 크로스사이트 스크립트(Cross-Site Scripting, XSS)

가. 정의

웹 페이지에 악의적인 스크립트를 포함시켜 사용자 측에서 실행되게 유도할 수 있다. 예를 들어 <그림 2-1>과 같이 검증되지 않은 외부 입력이 동적 웹 페이지 생성에 사용될 경우, 전송된 동적 웹 페이지를 열람하는 접속자의 권한으로 부적절한 스크립트가 수행되어 정보유출 등의 공격을 유발할 수 있다.



<그림 2-1> 크로스사이트 스크립트

나. 안전한 코딩기법

- 일반적인 경우에는 사용자가 문자열에 스크립트를 삽입하여 실행하는 것을 막기 위해 사용자가 입력한 문자열에서 <, >, &, “,” 등을 replace등의 문자 변환 함수나 메소드를 사용하여 &lt, &gt, &amp, &quot로 치환한다.
- HTML 태그를 허용하는 게시판에서는 지원하는 HTML 태그의 리스트(White List)를 선정한 후, 해당 태그만 허용하는 방식을 적용한다.

다. 예제

다음의 예제는 CGI를 이용하여 작성한 게시판 프로그램의 일부로서 제목, 작성자, 내용 등의 외부 입력을 inparam 값으로 특별한 입력 검증과정 없이 데이터베이스에 저장하고, 제목 필드에 입력된 값을 출력한다. 만약 악의적인 공격자가 inparam 값에 다음과 같은 스크립트를 삽입할 경우 크로스사이트 스크립트 공격에 노출될 수 있다.

(예 : <script>alert("Cross-Site Scripting Test")</script>)

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <time.h>
4:  #include <sys/timeb.h>
5:  #define LIST "phoneres.html"
6:  #define MAX_ENTRIES 10000
7:    .... ....
8:    .... ....
9:    //http Request Header 정보 읽어 옴
10:  cl = atoi(getenv("CONTENT_LENGTH"));
11:  for(i=0;cl && (!feof(stdin));i++)
12:    {
13:       //http 컨텐츠를 입력필드별로 끊어서 받아들임
14:       inparam[i].f_val = (char *)fmakeword(stdin,'&',&cl);
15:       plustospace(inparam[i].f_val);
16:       unescape_url(inparam[i].f_val);
17:       inparam[i].f_name = (char *)makeword(inparam[i].f_val,'=');
18:    }
19:   .... ....
20:   .... ....
21:  //HTTP Request에 대한 Response 파일을 생성
22:  fp = fopen(LIST, "w+");
23:  cntn = ftell(fp);
24:  fseek(fp, (long) (cntn+2), 0);
25:  fscanf(fp, "%d", &totn);
26:  fseek(fp, 0, 0);
27:  fprintf(fp, "<!%8d>%c", totn+1, 10);
28:  fseek(fp, 0, 2);
29:  .... ....
30:  .... ....
31:  // 결과화면 인터페이스 생성
32:  fprintf(fp, " <HTML><HEAD><TITLE>결과화면</TITLE></HEAD>");
33:  .... ....
34:  .... ....
35:  //생성한 파일에 검증되지 않은 출력 내용 출력
36:  // 본 예제에서 inparam[0]의 경우에는 제목 필드의 입력값이다.
37:  fprintf(fp, "<FONT SIZE=+1><STRONG>%s</STRONG></FONT>", inparam[0].f_val);
38:  fclose(fp);
39:  .... ....
40:  .... ....
```

다음의 예제는 게시판에서 지원하는 HTML 태그의 리스트(White List)를 선정한 후, 해당 태그만 허용하는 방식과 사용자가 입력한 "<", "/" 등 HTML에서 스크립트 생성에 사용되는 모든 문자열을 변경 및 필터링 함으로써 악의적인 스크립트 수행의 위험성을 줄일 수 있다.

■ 안전한 코드의 예 - C

배열로 지정

```
1:  #include <stdlib.h>
2:  #include <string.h>
3:  #include <ctype.h>
4:  //게시판에서 허용하고자 하는 HTML 태그 리스트(White List) 정의
5:  static unsigned char *allowed_formatters[] = {
6:    "b", "big", "blink", "i", "s", "small", "strike", "sub", "sup", "tt", "u",
7:    "abbr", "acronym", "cite", "code", "del", "dfn", "em", "ins", "kbd", "samp",
8:    "strong", "var", "dir", "li", "dl", "dd", "dt", "menu", "ol", "ul", "hr",
9:    "br", "p", "h1", "h2", "h3", "h4", "h5", "h6", "center", "bdo", "blockquote",
10:   "nobr", "plaintext", "pre", "q", "spacer",
11:   "a"
12: };
13:
14: #define SKIP_WHITESPACE(p) while (isspace(*p)) p++
15:
16: //요청 들어온 사이트 링크에 대한 유효성 검증
17: static int spc_is_valid_link(const char *input) {
18:   static const char *href="href";
19:   static const char *http = "http://";
20:   int  quoted_string = 0, seen_whitespace = 0;
21:
22:   if (!isspace(*input)) return 0;
23:   SKIP_WHITESPACE(input);
24:   if (strncasecmp(href, input, strlen(href))) return 0;
25:   input += strlen(href);
26:   SKIP_WHITESPACE(input);
27:   if (*input++ != '=') return 0;
28:   SKIP_WHITESPACE(input);
29:   if (*input == '"') {
30:     quoted_string = 1;
31:     input++;
32:   }
33:   if (strncasecmp(http, input, strlen(http))) return 0;
34:   for (input += strlen(http);  *input && *input != '>';  input++) {
35:     switch (*input) {
36:       case '.': case '/': case '-': case '_':
37:         break;
38:       case '"':
39:         if (!quoted_string) return 0;
40:         SKIP_WHITESPACE(input);
41:         if (*input != '>') return 0;
42:         return 1;
43:       default:
```

```
44:             if (isspace(*input)) {
45:               if (seen_whitespace && !quoted_string) return 0;
46:               SKIP_WHITESPACE(input);
47:               seen_whitespace = 1;
48:               break;
49:             }
50:             if (!isalnum(*input)) return 0;
51:             break;
52:         }
53:     }
54:     return (*input && !quoted_string);
55: }
56: //HTML 연결 문자열에 대한 구성 확인
57: static int spc_allow_tag(const char *input) {
58:     int  i;
59:     char *tmp;
60:
61:     if (*input == 'a')
62:         return spc_is_valid_link(input + 1);
63:     if (*input == '/') {
64:         input++;
65:         SKIP_WHITESPACE(input);
66:     }
67:     for (i = 0;  i < sizeof(allowed_formatters);  i++) {
68:         if (strncasecmp(allowed_formatters[i], input, strlen(allowed_formatters[i])))
69:             continue;
70:         else {
71:             tmp = input + strlen(allowed_formatters[i]);
72:             SKIP_WHITESPACE(tmp);
73:             if (*input == '>') return 1;
74:         }
75:     }
76:     return 0;
77: }
78: // HTML 스크립트 생성에 사용되는 문자열 변경
79: char *spc_escape_html(const char *input) {
80:     char          *output, *ptr;
81:     size_t        outputlen = 0;
82:     const char *c;
83:
84:     if (!(output = ptr = (char *)malloc(outputlen + 1))) return 0;
85:     for (c = input;  *c;  c++) {
86:         switch (*c) {
87:             case '<':
88:                 if (!spc_allow_tag(c + 1)) {
```

```
89:                 *ptr++ = '&';  *ptr++ = 'l';  *ptr++ = 't';  *ptr++ = ';';
90:                 break;
91:               } else {
92:                 do {
93:                   *ptr++ = *c;
94:                 } while (*++c != '>');
95:                 *ptr++ = '>';
96:                 break;
97:               }
98:             case '>':
99:               *ptr++ = '&';  *ptr++ = 'g';  *ptr++ = 't';  *ptr++ = ';';
100:              break;
101:            case '&':
102:              *ptr++ = '&';  *ptr++ = 'a';  *ptr++ = 'm';  *ptr++ = 'p';
103:              *ptr++ = ';';
104:              break;
105:            case '"':
106:              *ptr++ = '&';  *ptr++ = 'q';  *ptr++ = 'u';  *ptr++ = 'o';
107:              *ptr++ = 't';  *ptr++ = 't';
108:              break;
109:            default:
110:              *ptr++ = *c;
111:              break;
112:        }
113:    }
114:    *ptr = 0;
115:    return output;
116: }
```

라. 참고문헌

[1] CWE-79 Improper Neutralization of Input During Web Page Generation(Cross-site Scripting), http://cwe.mitre.org/data/definitions/79.html

[2] 2010 OWASP Top 10 - A2 Cross-Site Scripting(XSS), https://www.owasp.org/index.php/Top_10_2010-A2

[3] 2011 SANS Top 25 - RANK 4 (CWE-79), http://cwe.mitre.org/top25/

4. 운영체제 명령어 삽입(OS Command Injection)

가. 정의

외부 입력이 시스템 명령어 실행 인수로 적절한 처리 없이 사용되면 위험하다. 일반적으로 명령줄 인수나 스트림 입력 등 외부 입력을 사용하여 시스템 명령어를 생성하는 프로그램이 많이 있다. 하지만 이러한 경우 외부 입력 문자열은 신뢰할 수 없기 때문에 적절한 처리를 해주지 않을 경우, 공격자가 원하는 명령어 실행이 가능하게 된다.

나. 안전한 코딩기법

- 외부 입력이 직접 또는 문자열 복사를 통하여 system() 함수로 직접 전달되는 것은 위험하다. 미리 적절한 후보 명령어 리스트를 만들고 이중에서 선택하도록 하거나, 위험한 문자열의 존재 여부를 검사하는 과정을 수행해야 한다.

다. 예제

다음의 예제는 외부에서 파일명을 받아 해당 파일을 보여주는 프로그램이다. 이 프로그램을 **catWrapper**라고 했을 때 공격자가 **Story.txt; ls**를 인자로 전달하면 **Story.txt** 파일의 내용을 보여 준 후 현재 작업 디렉터리 목록을 출력한다. 현재 작업 디렉터리를 보여주는 것도 특권을 가진 상태에서 위험하지만, 다른 치명적인 명령을 위 프로그램을 통해 실행시킬 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  ……
2:    fgets(arg,80,stdin);
3:    commandLength = strlen(cat) + strlen(arg) + 1;
4:    command = (char *) malloc(commandLength);
5:    strncpy(command, cat, commandLength);
6:    strncat(command, argv[1], (commandLength - strlen(cat)) );
7:    system(command);
8:    return 0;
9:  ……
```

운영체제 명령어 실행 시에는 다음과 같이 **strpbrk()** 함수를 사용하여 위험한 문자가 있을 경우 명령어를 수행하지 않도록 해야 한다.

■ 안전한 코드의 예 - C

```
1:  ……
2:  fgets(arg,80,stdin);
3:  if (strpbrk(arg,";\"'."))
4:  {
5:    exit(1);
6:  }
7:  commandLength = strlen(cat) + strlen(arg) + 1;
```

```
8:   if(commandLength < 20)
9:   {
10:     command = (char *) malloc(commandLength);
11:  }
12:  ……
```

다음의 예제는 DB로 하여금 정해진 Batch명령을 연속적으로 실행하고 있다. 외부에서 받은 **backupType**을 그대로 Batch명령의 인자값으로 사용하고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const string BACKUP_DB_COMMAND    = "backup_db";
10: const string BACKUP_TYPE_PARAM     = "backup_type";
11: set<string> allowedBackupType;
12:
13: const string BACKUP_BATCH   = "c:\\util\\rmanDB.bat";
14: const string CLEANUP_BATCH = "c:\\utl\\cleanup.bat";
15:
16: string GetRequestParameter(const string & parameterName);
17: string GetUserHomeDir(const string & userId);
18:
19: int main(void)
20: {
21:     allowedBackupType.insert("All");
22:     allowedBackupType.insert("UpdatedItemOnly");
23:     allowedBackupType.insert("UpdateItemInAMonth");
24:
25:     string command   = GetRequestParameter(COMMAND_PARAM);
26:
27:     if(command == BACKUP_DB_COMMAND)
28:     {
29:             string backupType = GetRequestParameter(BACKUP_TYPE_PARAM);
30:
31:             string backupCommand= "cmd.exe /K \"" + BACKUP_BATCH + " " +
    allowedBackupType.get(backupType) + "&&" + CLEANUP_BATCH + "\"";
32:
33:             system(backupCommand.c_str());
34:             ...
35:     }
```

```
36:     ...
37: }
```

입력받은 **backupType**을 **allowedBackupType**의 데이터와 비교하여 허용되는 입력인 경우에만 동작이 실행되도록 하였다.

■ 안전한 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const string BACKUP_DB_COMMAND              = "backup_db";
10: const string BACKUP_TYPE_PARAM              = "backup_type";
11: set<string> allowedBackupType;
12:
13: const string BACKUP_BATCH   = "c:\\util\\rmanDB.bat";
14: const string CLEANUP_BATCH = "c:\\utl\\cleanup.bat";
15:
16: string GetRequestParameter(const string & parameterName);
17: string GetUserHomeDir(const string & userId);
18:
19: int main(void)
20: {
21:     allowedBackupType.insert("All");
22:     allowedBackupType.insert("UpdatedItemOnly");
23:     allowedBackupType.insert("UpdateItemInAMonth");
24:
25:     string command   = GetRequestParameter(COMMAND_PARAM);
26:
27:     if(command == BACKUP_DB_COMMAND)
28:     {
29:         string backupType = GetRequestParameter(BACKUP_TYPE_PARAM);
30:
31:         if(allowedBackupType.count(backupType) == 0)
32:         {
33:             // Error 처리
34:             return;
35:         }
36:         string backupCommand= "cmd.exe /K \"" + BACKUP_BATCH + " " +
    allowedBackupType.get(backupType) + "&&" + CLEANUP_BATCH + "\"";
37:
38:         system(backupCommand.c_str());
39:         ...
```

```
40:     }
41:     ...
42: }
```

라. 참고문헌

[1] CWE-78 Improper Neutralization of Special Elements used in an OS Command(OS Command Injection), http://cwe.mitre.org/data/definitions/78.html

[2] 2010 OWASP Top 10 - A1 Injection, https://www.owasp.org/index.php/Top_10_2010-A1

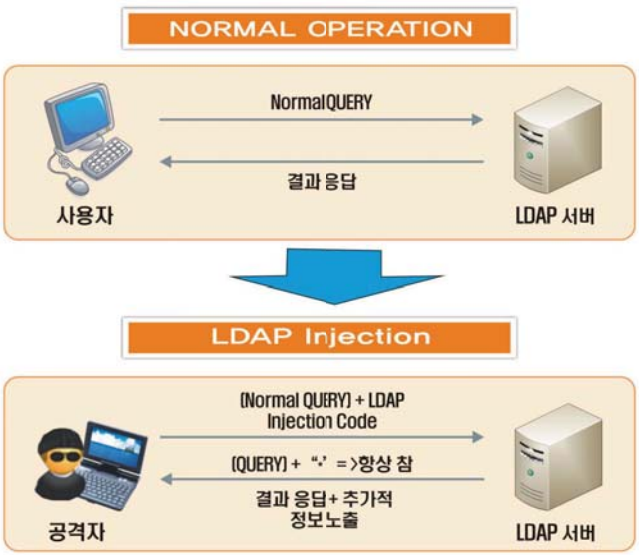
[3] 2011 SANS Top 25 - RANK 2 (CWE-78), http://cwe.mitre.org/top25/

5. LDAP 삽입(Improper Neutralization of Special Elements used in an LDAP Query, LDAP Injection)

가. 정의

공격자가 외부 입력을 통해서 의도하지 않은 LDAP 명령어를 수행할 수 있다. 즉, 웹 응용프로그램이 사용자가 제공한 입력을 올바르게 처리하지 못하면, 공격자가 LDAP 명령문의 구성을 바꿀 수 있다. 이로 인해 프로세스가 명령을 실행한 컴포넌트와 동일한 권한(authentication)을 가지고 동작하게 된다.

LDAP 쿼리문이나 결과에 외부 입력이 부분적으로 적절한 처리없이 사용되면, LDAP 쿼리문이 실행될 때 공격자는 LDAP 쿼리문의 내용을 마음대로 변경할 수 있다.



<그림 2-2> LDAP 삽입

나. 안전한 코딩기법

- 위험 문자에 대한 검사 없이 외부 입력을 LDAP 질의어 생성에 사용하면 안된다.
- DN과 필터에 사용되는 사용자 입력값에는 특수문자가 포함되지 않도록 특수문자를 제거한다. 특수문자를 사용해야 하는 경우 특수문자(DN에 사용되는 특수문자는 '\', 필터에 사용되는 특수문자(=, +, <, >, #, ; \ 등)에 대해서는 실행명령이 아닌 일반문자로 인식되도록 처리한다.

다. 예제

다음의 코드를 보면 외부 문자열이 getenv() 함수를 통해 LDAP 함수에 필터 문자열로 전달되었다. 대개의 경우는 사람 이름이 입력되어 한 사람에 대한 정보만 획득 가능하기 때문에 문제가 없지만 '|'와 같은 문자열을 사용하면 다른 사용자에 대한 정보를 얻을 수 있게 된다.

■ 안전하지 않은 코드의 예 - C

```
1:  int main()
2:  {
3:    char* filter = getenv("filter_string");
4:    int rc;
5:    LDAP *ld = NULL;
6:    LDAPMessage* result;
7:    rc = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter, NULL, 0,
    NULL, NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
8:    return 0;
9:  }
```

다음의 코드는 필터값을 외부 입력값이 아닌 특정한 값을 사용하기 때문에 안전하다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <ldap.h>
3:  #define FIND_DN ""
4:  int main()
5:  {
6:    char* filter = "(manager=admin)";
7:    int rc;
8:    LDAP *ld = NULL;
9:    LDAPMessage* result;
10:   rc = ldap_search_ext_s( ld, FIND_DN, LDAP_SCOPE_BASE, filter, NULL, 0, NULL,
    NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
11:   return 0;
12: }
```

라. 참고문헌

[1] CWE-90 Improper Neutralization of Special Elements used in an LDAP Query(LDAP Injection), http://cwe.mitre.org/data/definitions/90.html

[2] 2010 OWASP Top 10 - A1 - Injection https://www.owasp.org/index.php/Top_10_2010-A1-Injection

6. 디렉터리 경로 조작(Path Traversal)

디렉터리 경로 조작은 크게 상대, 절대 디렉터리 경로 조작으로 나눌 수 있다. 우선 상대 디렉터리 경로 조작에 대한 취약점을 살펴본 후 절대 디렉터리 경로 조작에 대한 설명하겠다.

6-1. 상대 디렉터리 경로 조작(Relative Path Traversal)

가. 정의

외부 입력을 통하여 디렉터리 경로 문자열을 생성하는 경우, 결과 디렉터리가 제한된 디렉터리여야 할 때, 악의적인 외부 입력을 제대로 변환시키지 않으면 예상 밖 영역의 경로 문자열이 생성될 수 있다.

이 취약점을 이용하여 제한된 디렉터리 영역 밖을 공격자가 접근할 수 있게 된다. 가장 많이 쓰이는 것이 ".."을 사용하여 공격하는 것이다. 대부분의 시스템에서 ".."은 상위 디렉터리를 의미하기 때문에 제한된 디렉터리의 상위를 접근할 수 있는 경로를 생성하게 되는 것이다.

나. 안전한 코딩기법

- 파일 접근 함수의 파일 경로 인수의 일부를 외부 입력으로 부터 조합하여 사용하지 않는 것이 바람직하다.

다. 예제

reportName이 파일 이름일 경우에는 지정 디렉터리 **/home/www/tmp**에서 파일 처리 함수를 실행하지만, **reportName**이 **../../../etc/passwd**와 같이 입력되면 상위 디렉터리에 있는 파일에 접근하게 된다.

안전하지 않은 코드의 예 - C

```
1:  void f()
2:  {
3:      char* rName = getenv("reportName");
4:      char buf[30];
5:      strncpy(buf, "/home/www/tmp/", 30);
6:      strncat(buf, rName, 30);
7:      unlink(buf);
8:  }
```

외부 입력을 가지고 파일경로를 조합하여 파일시스템에 접근하는 경로를 만들지 말아야 한다.

■ 안전한 코드의 예 - C

```
1:  void f()
2:  {
3:    char buf[30];
4:    strncpy(buf, "/home/www/tmp/", 30);
5:    strncat(buf, "report", 30);
6:    unlink(buf);
7:  }
```

다음의 예제에서는 외부에서 입력받은 파일명을 그대로 사용하여 파일을 열고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:
5:  using namespace std;
6:
7:  const static string COMMAND_PARAM   = "command";
8:
9:  const static string    READ_DOCUMENT_COMMAND= "read_document";
10: const static string    USER_ID_PARAM    = "user_id";
11: const static string    FILE_NAME_PARAM= "file_name";
12: const static int BUFFER_SIZE    = 256;
13:
14: string GetRequestParameter(const string & parameterName);
15:
16: int main(void)
17: {
18:    string command   = GetRequestParameter(COMMAND_PARAM);
19:
20:    if(command == READ_DOCUMENT_COMMAND)
21:    {
22:          string userId = GetRequestParameter(USER_ID_PARAM);
23:          string fileName = GetRequestParameter(FILE_NAME_PARAM);
24:
25:          ifstream inFile(fileName.c_str());
26:          if(inFile.is_open() == false)
27:          {
28:                // Error 처리
29:          }
30:
31:          char buffer[BUFFER_SIZE];
32:          inFile.read(buffer, BUFFER_SIZE);
```

```
33:             ...
34:     }
35:     ...
36: }
```

다음의 예제에서는 입력받은 외부 입력에 의한 파일 경로를 절대 경로로 변환한 후 해당 위치의 파일을 열고 있다. 또한 **userHomePath**의 길이만큼을 변환된 절대 경로 앞쪽을 잘라서 서로 비교함으로써 공격코드에 의한 파일 경로의 변조가 없었는지 확인하고 있다.

■ 안전한 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:
5:
6:  using namespace std;
7:
8:  const static string COMMAND_PARAM   = "command";
9:
10: const static string    READ_DOCUMENT_COMMAND= "read_document";
11: const static string    USER_ID_PARAM    = "user_id";
12: const static string    FILE_NAME_PARAM= "file_name";
13: const static int BUFFER_SIZE    = 256;
14:
15:
16: string GetRequestParameter(const string & parameterName);
17: string DecodeURLEncodedString(const string & urlEncodedString);
18: string ConvertToAbsolutePath(const string & urlEncodedString);
19:
20: int main(void)
21: {
22:     string command   = GetRequestParameter(COMMAND_PARAM);
23:
24:     if(command == READ_DOCUMENT_COMMAND)
25:     {
26:             string userId = GetRequestParameter(USER_ID_PARAM);
27:             string fileName = GetRequestParameter(FILE_NAME_PARAM);
28:
29:             fileName    = DecodeURLEncodedString(fileName);
30:
31:             string userHomePath    = GetUserHomeDir(userId);
32:             string filePath= ConvertToAbsolutePath(userHomePath + fileName);
1:
2:              if(userHomePath != filePath.substr(0, userHomePath.length()))
3:              {
```

```
4:                  // Error 처리
5:                  return;
6:              }
33:             ifstream inFile(filePath.c_str());
34:             if(inFile.is_open() == false)
35:             {
36:                 // Error 처리
37:             }
38:             char buffer[BUFFER_SIZE];
39:             inFile.read(buffer, BUFFER_SIZE);
40:             ...
41:     }
42:     ...
43: }
```

마. 참고문헌

[1] CWE-23 Relative Path Traversal, http://cwe.mitre.org/data/definitions/23.html

[2] CWE-22 Absolute Path Traversal, http://cwe.mitre.org/data/definitions/36.html

[3] 2010 OWASP Top 10 - A4 Insecure Direct Object Reference, https://www.owasp.org/index.php/Top_10_2010-A4

[4] 2011 SANS Top 25 - Rank 13 (CWE-22), http://cwe.mitre.org/top25/

6-2. 절대 디렉터리 경로 조작(Absolute Path Traversal)

가. 정의

외부 입력이 파일 시스템을 조작하는 경로를 직접 제어할 수 있거나 영향을 끼치면 위험하다. 이 취약점은 공격자가 응용프로그램에 치명적인 시스템 파일 또는 일반 파일을 접근하거나 변경 가능하도록 한다.

다음 두 조건이 만족되면 공격은 성공한다. (1) 공격자가 파일 시스템 조작에 사용되는 경로를 결정한다. (2) 그 파일 조작을 통해서 공격자가 용납할 수 없는 권한을 획득한다. 예를 들어, 공격자가 설정에 관계된 파일을 변경할 수 있거나 실행을 시킬 수 있다.

나. 안전한 코딩기법

- 파일 접근 함수의 파일 경로 인수로 외부 입력을 직접 사용하지 않는 것이 바람직하다.

다. 예제

reportName 파일 이름이 외부 입력에 의해 직접 결정되어 파일을 접근하게 되어 외부로부터 임의의 파일에 대하여 파일 작업 (예제의 경우 삭제 작업)을 수행하게 된다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <unistd.h>
4:  void absolute()
5:  {
6:    /* 외부입력값으로 파일 경로가 설정되고 있다 */
7:    char* rName = getenv("reportName");
8:    unlink(rName);
9:  }
```

외부 입력을 가지고 직접 파일시스템을 접근하는 경로를 만들지 않아야 한다.

안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <unistd.h>
4:  void absolute()
5:  {
6:    /* 외부입력값으로 파일 경로를 설정하지 않아야한다 */
7:    unlink("/home/www/tmp/report");
8:  }
```

다음의 예제에서는 외부에서 입력받은 값을 토대로 **styleFileName**값을 생성하고 그 문자열을 그대로 사용하여 파일의 전체 경로를 생성한 후 파일에 접근하고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:
6:  using namespace std;
7:
8:  const static string COMMAND_PARAM   = "command";
9:
10: const static string APPLY_STYPLE_COMMAND     = "apply_style";
11: const static string USER_ID_PARAM            = "user_id";
12: const static string STYLE_FILE_NAME_PARAM    = "styple_file_name";
13: const static int BUFFER_SIZE    = 256;
14:
15: string GetRequestParameter(const string & parameterName);
16: string GetUserHomeDir(const string & userId);
17:
18: int main(void)
19: {
20:     string command   = GetRequestParameter(COMMAND_PARAM);
21:
22:     if(command == APPLY_STYPLE_COMMAND)
23:     {
24:             string userId = GetRequestParameter(USER_ID_PARAM);
25:             string styleFileName = GetRequestParameter(STYLE_FILE_NAME_PARAM);
26:
27:             string userHomePath    = GetUserHomeDir(userId);
28:             string styleFilePath      = ConvertToAbsolutePath(userHomePath + styleFileName);
29:
30:             ifstream inFile(styleFilePath.c_str());
31:             if(inFile.is_open() == false)
32:             {
33:                     // Error 처리
34:             }
35:             char buffer[BUFFER_SIZE];
36:             inFile.read(buffer, BUFFER_SIZE);
37:             ...
38:     }
39:     ...
40: }
```

다음의 예제에서는 입력값으로 들어온 **styleName**을 토대로 미리 정해놓은 데이터 테이블(**styleFileName**)과 비교하여 미리 정해둔 데이터를 최종적으로 얻게 만든다.(그 외의 값들은 전부 default값으로 처리된다.) 그 후 얻어진 값을 가지고 파일의 경로를 생성하여 파일에 접근한다.

■ 안전한 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:
6:  using namespace std;
7:
8:  const static string COMMAND_PARAM   = "command";
9:
10: const static string APPLY_STYPLE_COMMAND = "apply_style";
11: const static string USER_ID_PARAM = "user_id";
12: const static string STYLE_NAME_PARAM= "styple_name";
13: const static int BUFFER_SIZE    = 256;
14:
15: static map<string, string>          styleFileNames;
16:
17:
18: string GetRequestParameter(const string & parameterName);
19: string GetUserHomeDir(const string & userId);
20:
21: int main(void)
22: {
23:     styleFileNames["Normal"]      = "NormalStyle.cfg";
24:     styleFileNames["Classic"]     = "ClassicStyle_1.cfg";
25:     styleFileNames["Gothic"]      = "ClassicStyle_2.cfg";
26:
27:     string command   = GetRequestParameter(COMMAND_PARAM);
28:
29:     if(command == APPLY_STYPLE_COMMAND)
30:     {
31:             string userId = GetRequestParameter(USER_ID_PARAM);
32:             string styleName = GetRequestParameter(STYLE_NAME_PARAM);
33:
34:             string userHomePath    = GetUserHomeDir(userId);
35:             string      styleFilePath=      ConvertToAbsolutePath(userHomePath      +      style-
    FileNames[styleName]);
36:
37:             ifstream inFile(styleFilePath.c_str());
38:             if(inFile.is_open() == false)
```

```
39:             {
40:                     // Error 처리
41:             }
42:             char buffer[BUFFER_SIZE];
43:             inFile.read(buffer, BUFFER_SIZE);
44:             ...
45:     }
46:     ...
47: }
```

라. 참고문헌

[1] CWE-36 Absolute Path Traversal, http://cwe.mitre.org/data/definitions/36.html

[2] CWE-22 Absolute Path Traversal, http://cwe.mitre.org/data/definitions/36.html

[3] 2010 OWASP Top 10 - A4 Insecure Direct Object Reference, https://www.owasp.org/index.php/Top_10_2010-A4

[4] 2011 SANS Top 25 – RANK 13 (CWE-22), http://cwe.mitre.org/top25/

7. 정수 오버플로우(Integer Overflow or Wraparound)

가. 정의

정수 연산 시에 정수형 변수가 취할 수 있는 범위를 초과할 때 발생하는 경우이다. 정수형 변수의 오버플로우는 정수값이 증가하면서 가장 큰 값보다 더 커져서 실제 비트 패턴으로 저장되는 값은 아주 작은 수이거나 음수로 될 수 있다. 이러한 상황을 계산 후에 검사하지 않고 진행하게 된다면 이 값이 순환문의 조건이나 메모리 할당, 메모리 복사 등에 쓰이거나 이 값에 근거해서 보안 관련 결정을 하는 경우 보안취약점이 발생될 수 있다.

나. 안전한 코딩기법

- Signed 정수 타입의 경우, 값이 오버플로우가 발생하게 되면 양수가 음수로 음수가 양수로 변환되면서 잘못된 값을 전달될 수 있다. 이를 위해, 메모리 할당 함수에서는 할당량을 표시하는 파라미터를 unsigned 타입으로 전달해야 잘못된 값이 사용되는 것을 방지할 수 있다.

다. 예제

Signed int로 선언된 변수를 **malloc()** 함수의 파라미터로 사용하는 경우, 이전 처리 과정에서 오버플로우가 발생하여 음수가 입력이 되어도 큰 양수로 잘못 해석하여 진행될 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdlib.h>
2:  void* intAlloc(int size, int reserve)
3:  {
4:     void *rptr;
5:     size += reserve;
6:     rptr = malloc(size * sizeof(int));
7:     if (rptr == NULL)
8:        exit(1);
9:     return rptr;
10:  }
```

malloc() 함수의 파라미터로 사용되는 변수를 **unsigned** 로 사용해서 유효한 값의 범위를 넓혀 integer overflow 가능성을 낮춘다.

■ 안전한 코드의 예 - C

```
1:  #include <stdlib.h>
2:  void* intAlloc(int size, int reserve)
3:  {
4:     void *rptr;
5:     unsigned s;
6:     size += reserve;
```

```
7:      s=size* sizeof(int);
8:      if(s<0)
9:      return NULL;
10:     rptr = malloc(s);
11:     if (rptr == NULL)
12:     exit(1);
13:     return rptr;
14:   }
```

외부로부터 입력받은 dataIndex와 fieldIndex를 크기 계산 없이 바로 배열의 인덱스로 사용하고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const static char * COMMAND_PARAM  = "command";
10: const static char * GET_DATA_COMMAND        = "get_data";
11: const static char * DATA_INDEX_PARAM        = "data_index";
12: const static char * FIELD_INDEX_PARAM= "field_index";
13: const static char * VALUE_PARAM      = "value";
14:
15: static int DATA_SIZE = 256;
16:
17: int main(void)
18: {
19:     char * queryStr;
20:     queryStr = getenv("QUERY_STRING");
21:     if (queryStr == NULL)
22:     {
23:             // Error 처리
24:             ...
25:     }
26:     string command  = GetRequestParameter(queryStr, COMMAND_PARAM);
27:
28:     if(command.compare(GET_DATA_COMMAND) == 0)
29:     {
30:             int dataIndex = atoi(GetRequestParameter(queryStr, DATA_INDEX_PARAM));
31:             int fieldIndex = atoi(GetRequestParameter(queryStr, FIELD_INDEX_PARAM));
32:             int valueIndex = atoi(GetRequestParameter(queryStr, VALUE_PARAM));
```

```
33:
34:             int index    = dataIndex * DATA_SIZE + fieldIndex;
35:             ...
36:     }
37:     ...
38: }
```

인덱스로 사용되는 값은 음수값 여부의 체크 뿐 아니라 배열의 전체 크기 한계를 넘지 않는지도 충분히 검증한 후 사용되어야 한다.

■ 안전한 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const static char * COMMAND_PARAM  = "command";
10: const static char * GET_DATA_COMMAND = "get_data";
11: const static char * DATA_INDEX_PARAM= "data_index";
12: const static char * FIELD_INDEX_PARAM= "field_index";
13: const static char * VALUE_PARAM      = "value";
14:
15: static int DATA_SIZE = 256;
16:
17: int main(void)
18: {
19:     char * queryStr;
20:     queryStr = getenv("QUERY_STRING");
21:     if (queryStr == NULL)
22:     {
23:         // Error 처리
24:         ...
25:     }
26:     string command  = GetRequestParameter(queryStr, COMMAND_PARAM);
27:
28:     if(command.compare(GET_DATA_COMMAND) == 0)
29:     {
30:         int dataIndex = atoi(GetRequestParameter(queryStr, DATA_INDEX_PARAM));
31:         int fieldIndex = atoi(GetRequestParameter(queryStr, FIELD_INDEX_PARAM));
32:         int valueIndex = atoi(GetRequestParameter(queryStr, VALUE_PARAM));
33:
34:         if((dataIndex < 0) || (fieldIndex < 0))
```

```
35:             {
36:                     // Error 처리
37:                     return;
38:             }
39:             if((dataIndex < 0) || (dataIndex > Integer.MAX_VALUE / DATA_SIZE))
40:             {
41:                     // Error 처리
42:                     return;
43:             }
44:              if((fieldIndex < 0) || (filedIndex > (Integer.MAX_VALUE - (dataIndex *
    DATA_SIZE))))
45:             {
46:                     // Error 처리
47:                     return;
48:             }
49:            int index    = dataIndex * DATA_SIZE + fieldIndex;
50:            ...
51:     }
52:     ...
53: }
```

라. 참고문헌

[1] CWE-190 Integer Overflow or Wraparound, http://cwe.mitre.org/data/definitions/190.html

[2] CWE-78 Improper Neutralization of Special Elements used in an OS Command (OS Command Injection), http://cwe.mitre.org/data/definitions/78.html

[3] 2011 SANS Top 25 - RANK 2 (CWE-78), http://cwe.mitre.org/top25/

8. 보호 메커니즘을 우회할 수 있는 입력값 변조(Reliance on Untrusted Inputs in a Security Decision)

가. 정의

응용프로그램이 외부 입력값에 대한 신뢰를 전제로 보호메커니즘을 사용하는 경우 공격자가 입력값을 조작할 수 있다면 보호메커니즘을 우회할 수 있게 된다. 개발자들이 흔히 쿠키, 환경변수 또는 히든필드와 같은 입력값이 조작될 수 없다고 가정하지만 공격자는 다양한 방법을 통해 이러한 입력값들을 변경할 수 있고 조작된 내용은 탐지되지 않을 수 있다.

인증이나 인가와 같은 보안결정이 이런 입력값(쿠키, 환경변수, 히든필드 등)에 기반으로 수행되는 경우 공격자는 이런 입력값을 조작하여 응용프로그램의 보안을 우회할 수 있으므로 충분한 암호화, 무결성 체크 또는 다른 메커니즘이 없는 경우 외부 사용자에 의한 입력값을 신뢰해서는 안 된다.

나. 안전한 코딩기법

- 상태정보나 민감한 데이터 특히 사용자 세션정보와 같은 중요한 정보는 서버에 저장하고 보안확인 절차도 서버에서 실행한다.
- 보안설계관점에서 신뢰할 수 없는 입력값이 어플리케이션 내부로 들어올 수 있는 지점과 보안결정에 사용되는 입력값을 식별하고 제공되는 입력값에 의존할 필요가 없는 구조로 변경할 수 있는지 검토한다.

다. 예제

안전하지 않은 코드의 예 - C

```
1:  #include "std_testcase.h"
2:  #include "cgic.h"
3:  #include <wchar.h>
4:  #ifdef _WIN32
5:  # include <winsock2.h>
6:  # include <windows.h>
7:  # include <direct.h>
8:  # pragma comment(lib, "ws2_32") /* include ws2_32.lib when linking */
9:  # define CLOSE_SOCKET closesocket
10: # define PATH_SZ 100
11: #else /* NOT _WIN32 */
12: # define INVALID_SOCKET -1
13: # define SOCKET_ERROR -1
14: # define CLOSE_SOCKET close
15: # define SOCKET int
16: # define PATH_SZ PATH_MAX
```

```
17:  #endif
18:
19:  #define TCP_PORT 27015
20:
21:
22:
23:
24:
25:  #ifndef OMITBAD
26:
27:  void     KRD_114_Reliance_on_Untrusted_Inputs_in_a_Security_Decision__char_connect_sock-
     et_cookie_strncmp_0101_bad()
28:  {
29:      char * data;
30:      char data_buf[100] = "trustedvalue";
31:      data = data_buf;
32:      {
33:  #ifdef _WIN32
34:          WSADATA wsa_data;
35:          int wsa_data_init = 0;
36:  #endif
37:          int recv_rv;
38:          struct sockaddr_in s_in;
39:          char *replace;
40:          SOCKET connect_socket = INVALID_SOCKET;
41:          size_t data_len = strlen(data);
42:          do
43:          {
44:  #ifdef _WIN32
45:              if (WSAStartup(MAKEWORD(2,2), &wsa_data) != NO_ERROR) break;
46:              wsa_data_init = 1;
47:  #endif
48:              connect_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
49:              if (connect_socket == INVALID_SOCKET) break;
50:              memset(&s_in, 0, sizeof(s_in));
51:              s_in.sin_family = AF_INET;
52:              s_in.sin_addr.s_addr = inet_addr("127.0.0.1");
53:              s_in.sin_port = htons(TCP_PORT);
54:              if (connect(connect_socket, (struct sockaddr*)&s_in, sizeof(s_in)) ==
     SOCKET_ERROR) break;
55:              /* Abort on error or the connection was closed, make sure to recv one
56:               * less char than is in the recv_buf in order to append a terminator */
57:              recv_rv = recv(connect_socket, (char *)data+data_len, (int)(100-data_len-1), 0);
58:              if (recv_rv == SOCKET_ERROR || recv_rv == 0) break;
59:              /* Append null terminator */
```

```
60:                 data[recv_rv] = '\0';
61:                 /* Eliminate CRLF */
62:                 replace = strchr(data, '\r');
63:                 if (replace) *replace = '\0';
64:                 replace = strchr(data, '\n');
65:                 if (replace) *replace = '\0';
66:             }
67:             while (0);
68:             if (connect_socket != INVALID_SOCKET) CLOSE_SOCKET(connect_socket);
69:     #ifdef _WIN32
70:             if (wsa_data_init) WSACleanup();
71:     #endif
72:         }
73:
74:         {
75:             char * data = "trustedvalue";
76:
77:             char    **array, **arrayStep;
78:             char    cname[1024], cvalue[1024];
79:             fprintf(cgiOut, "Cookies Submitted On This Call, With Values "
80:                     "(Many Browsers NEVER Submit Cookies):<p>\n");
81:             if (cgiCookies(&array) != cgiFormSuccess) {
82:                 return;
83:             }
84:             arrayStep = array;
85:             fprintf(cgiOut, "<table border=1>\n");
86:             fprintf(cgiOut, "<tr><th>Cookie<th>Value</tr>\n");
87:             while (*arrayStep) {
88:                 char value[1024];
89:                 fprintf(cgiOut, "<tr>");
90:                 fprintf(cgiOut, "<td>");
91:                 cgiHtmlEscape(*arrayStep);
92:                 fprintf(cgiOut, "<td>");
93:                 cgiCookieString(*arrayStep, value, sizeof(value));
94:                 cgiHtmlEscape(value);
95:                 fprintf(cgiOut, "\n");
96:                 arrayStep++;
97:             }
98:             fprintf(cgiOut, "</table>\n");
99:             cgiFormString("cname", cname, sizeof(cname));
100:            cgiFormString("cvalue", cvalue, sizeof(cvalue));
101:
102:            /* FLAW */
103:            if (strlen(cname) && !strncmp(cvalue, data, sizeof(data)))
104:            {
```

```
105:                fprintf(cgiOut, "New Cookie Set On This Call:<p>\n");
106:                fprintf(cgiOut, "Name: ");
107:                cgiHtmlEscape(cname);
108:                fprintf(cgiOut, "Value: ");
109:                cgiHtmlEscape(cvalue);
110:                fprintf(cgiOut, "<p>\n");
111:                fprintf(cgiOut, "If your browser accepts cookies "
112:                        "(many do not), this new cookie should appear "
113:                        "in the above list the next time the form is "
114:                        "submitted.<p>\n");
115:            }
116:            else
117:            {
118:                fprintf(cgiOut, "Untrusted values<p>\n");
119:            }
120:
121:            cgiStringArrayFree(array);
122:        }
123:
124: }
125:
126: #endif /* OMITBAD */
127:
128:
129: /* Below is the main(). It is only used when building this testcase on
130:    its own for testing or for building a binary to use in testing binary
131:    analysis tools. It is not used when compiling all the testcases as one
132:    application, which is how source code analysis tools are tested. */
133:
134: #ifdef INCLUDEMAIN
135:
136: int main(int argc, char * argv[])
137: {
138:     /* seed randomness */
139:     srand( (unsigned)time(NULL) );
140: #ifndef OMITBAD
141:     printLine("Calling bad()...");
142:     KRD_114_Reliance_on_Untrusted_Inputs_in_a_Security_Decision__char_connect_sock-
    et_cookie_strncmp_0101_bad();
143:     printLine("Finished bad()");
144: #endif /* OMITBAD */
145:     return 0;
146: }
147:
148: #endif
```

사용자 권한, 인증여부 등 보안결정에 사용하는 사용자 입력값을 사용하지 않고 내부 세션값을 활용한다.

■ 안전한 코드의 예 - C

```
1:  #include "std_testcase.h"
2:  #include "cgic.h"
3:  #include <wchar.h>
4:  #ifdef _WIN32
5:  # include <winsock2.h>
6:  # include <windows.h>
7:  # include <direct.h>
8:  # pragma comment(lib, "ws2_32") /* include ws2_32.lib when linking */
9:  # define CLOSE_SOCKET closesocket
10: # define PATH_SZ 100
11: #else /* NOT _WIN32 */
12: # define INVALID_SOCKET -1
13: # define SOCKET_ERROR -1
14: # define CLOSE_SOCKET close
15: # define SOCKET int
16: # define PATH_SZ PATH_MAX
17: #endif
18:
19: #define TCP_PORT 27015
20:
21:
22:
23:
24:
25:
26: #ifndef OMITGOOD
27:
28: /* goodG2B uses the GoodSource with the BadSink */
29: static void good1()
30: {
31:     char * data;
32:     char data_buf[100] = "trustedvalue";
33:     data = data_buf;
34:     /* FIX: Benign input preventing command injection */
35:     strcat(data, "trustedvalue");
36:
37:     {
38:         char * data = "trustedvalue";
39:
40:         char    **array, **arrayStep;
41:         char    cname[1024], cvalue[1024];
```

```
42:             fprintf(cgiOut, "Cookies Submitted On This Call, With Values "
43:                     "(Many Browsers NEVER Submit Cookies):<p>\n");
44:             if (cgiCookies(&array) != cgiFormSuccess) {
45:                 return;
46:             }
47:             arrayStep = array;
48:             fprintf(cgiOut, "<table border=1>\n");
49:             fprintf(cgiOut, "<tr><th>Cookie<th>Value</tr>\n");
50:             while (*arrayStep) {
51:                 char value[1024];
52:                 fprintf(cgiOut, "<tr>");
53:                 fprintf(cgiOut, "<td>");
54:                 cgiHtmlEscape(*arrayStep);
55:                 fprintf(cgiOut, "<td>");
56:                 cgiCookieString(*arrayStep, value, sizeof(value));
57:                 cgiHtmlEscape(value);
58:                 fprintf(cgiOut, "\n");
59:                 arrayStep++;
60:             }
61:             fprintf(cgiOut, "</table>\n");
62:             cgiFormString("cname", cname, sizeof(cname));
63:             cgiFormString("cvalue", cvalue, sizeof(cvalue));
64:
65:             /* FLAW */
66:             if (strlen(cname) && !strncmp(cvalue, data, sizeof(data)))
67:             {
68:                 fprintf(cgiOut, "New Cookie Set On This Call:<p>\n");
69:                 fprintf(cgiOut, "Name: ");
70:                 cgiHtmlEscape(cname);
71:                 fprintf(cgiOut, "Value: ");
72:                 cgiHtmlEscape(cvalue);
73:                 fprintf(cgiOut, "<p>\n");
74:                 fprintf(cgiOut, "If your browser accepts cookies "
75:                         "(many do not), this new cookie should appear "
76:                         "in the above list the next time the form is "
77:                         "submitted.<p>\n");
78:             }
79:             else
80:             {
81:                 fprintf(cgiOut, "Untrusted values<p>\n");
82:             }
83:
84:             cgiStringArrayFree(array);
85:         }
86:
```

```
87:  }
88:
89:  void      KRD_114_Reliance_on_Untrusted_Inputs_in_a_Security_Decision__char_connect_sock-
     et_cookie_strncmp_0101_good()
90:  {
91:      good1();
92:  }
93:
94:  #endif /* OMITGOOD */
95:
96:  /* Below is the main(). It is only used when building this testcase on
97:     its own for testing or for building a binary to use in testing binary
98:     analysis tools. It is not used when compiling all the testcases as one
99:     application, which is how source code analysis tools are tested. */
100:
101: #ifdef INCLUDEMAIN
102:
103: int main(int argc, char * argv[])
104: {
105:     /* seed randomness */
106:     srand( (unsigned)time(NULL) );
107: #ifndef OMITGOOD
108:     printLine("Calling good()...");
109:     KRD_114_Reliance_on_Untrusted_Inputs_in_a_Security_Decision__char_connect_sock-
     et_cookie_strncmp_0101_good();
110:     printLine("Finished good()");
111: #endif /* OMITGOOD */
112:     return 0;
113: }
114:
115: #endif
```

라. 참고문헌

[1] CWE-807 Reliance on Untrusted Inputs in a Security Decision, http://cwe.mitre.org/data/definitions/807.html

9. 스택에 할당된 버퍼 오버플로우(Stack-based Buffer Overflow)

가. 정의

스택에 할당되는 버퍼들 (즉, 지역변수로 선언된 변수 또는 함수의 인자로 넘어온 변수들)이 문자열 계산 등에 의해서 정의된 버퍼의 한계치를 넘는 경우 버퍼 오버플로우가 발생한다.

나. 안전한 코딩기법

- 프로그램이 버퍼가 저장할 수 있는 것보다 많은 데이터를 입력하지 않는다.
- 프로그램이 버퍼 경계 밖의 메모리 영역을 참조하지 않는다.
- 프로그램이 사용할 메모리를 적절하게 계산하여 로직에서 에러를 발생시키지 않도록 한다.
- 입력에 대해서 경계 검사(bounds checking)를 한다.
- strcpy()와 같이 버퍼 오버플로우에 취약한 함수를 사용하지 않는다.

다. 예제

매개변수로 받은 문자열 크기가 지역버퍼에 복사가 되는지 확인하지 않고, **strcpy()** 함수를 이용하여 데이터를 복사한다. 공격자가 스트링 매개변수의 내용에 영향을 줄 수 있다면 버퍼 오버플로우를 발생시킬 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  void manipulate_string(char* string)
2:  {
3:    char buf[24];
4:    strcpy(buf, string);
5:    ……
6:  }
```

매개변수로 받은 변수(**string**)와 복사하려는 **buf**의 크기를 비교한다. **strncpy()** 함수를 사용하여 복사하기 위한 **buf**의 크기를 제한하고, **buf**의 마지막 문자는 '\0'을 가지게 한다.

■ 안전한 코드의 예 - C

```
1:  void manipulate_string(char* string)
2:  {
3:    char buf[24];
4:    /* 복사하려는 buf와 길이 비교를 한다. */
5:    if (strlen(string < sizeof(buf))
6:      strncpy(buf, string, sizeof(buf)-1);
7:    /* 문자열은 반드시 null로 종료되어야 함 */
8:    buf[sizeof(buf)-1] = '\0';
9:    ……
10: }
```

다음의 예문은 외부의 입력 **queryStr**로부터 문자열을 받아와 **strtok**함수로 잘라서 배열에 입력하는데 공격자가 스트링 길이를 충분히 길게 하여 token의 개수를 **MAX_DATA_BUFFER_LENGTH** 보다 크게 만들면 버퍼 오버플로우를 발생시킬 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #include <regex.h>
5:
6:  #include <sqltypes.h>
7:  #include <sql.h>
8:
9:  static SQLHSTMT statmentHandle;
10:
11: const char * GetParameter(const char * queryString, const char * key);
12:
13: static const char * UPDATE_DATA = "update_data";
14: static const char * DATA_PARAM = "data";
15: static const int MAX_DATA_BUFFER_LENGTH = 256;
16:
17: const int EQUAL = 0;
18: int main(void)
19: {
20:     char * queryStr = getenv("QUERY_STRING");
21:     if (queryStr == NULL)
22:     {
23:             // Error 처리
24:             ...
25:     }
26:
27:     char * command = GetParameter(queryStr, "command");
28:     if (strcmp(command, UPDATE_DATA) == EQUAL)
29:     {
30:             int dataBuffer[MAX_DATA_BUFFER_LENGTH];
31:             int dataIndex = 0;
32:             const char * data = GetParameter(queryStr, DATA_PARAM);
33:
34:             char * token  = strtok(data, " ");
35:             whle(token != NULL)
36:             {
37:                     dataBuffer[dataIndex]        = atoi(token);
38:                     token = strtok(NULL, " ");
39:                     dataIndex++;
40:             }
```

```
41:             ...
42:     }
43:     ...
44:     return EXIT_SUCCESS;
45: }
```

배열의 인덱스 값을 **MAX_DATA_BUFFER_LENGTH**와 비교하여 오버플로우 조건이 되면 더 이상 배열에 값을 입력하지 않도록 하였다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #include <regex.h>
5:
6:  #include <sqltypes.h>
7:  #include <sql.h>
8:
9:  static SQLHSTMT statmentHandle;
10:
11: const char * GetParameter(const char * queryString, const char * key);
12:
13: static const char * UPDATE_DATA = "update_data";
14: static const char * DATA_PARAM = "data";
15: static const int MAX_DATA_BUFFER_LENGTH = 256;
16:
17: const int EQUAL = 0;
18: int main(void)
19: {
20:     char * queryStr = getenv("QUERY_STRING");
21:     if (queryStr == NULL)
22:     {
23:             // Error 처리
24:             ...
25:     }
26:     char * command = GetParameter(queryStr, "command");
27:     if (strcmp(command, UPDATE_DATA) == EQUAL)
28:     {
29:             int dataBuffer[MAX_DATA_BUFFER_LENGTH];
30:             int dataIndex = 0;
31:             const char * data = GetParameter(queryStr, DATA_PARAM);
32:
33:             char * token  = strtok(data, " ");
34:             whle(token != NULL)
35:             {
```

```
36:                     if(dataIndex >= MAX_DATA_BUFFER_LENGTH)         break;
37:                     dataBuffer[dataIndex]          = atoi(token);
38:                     token = strtok(NULL, " ");
39:                     dataIndex++;
40:             }
41:             ...
42:     }
43:     ...
44:     return EXIT_SUCCESS;
45: }
```

라. 참고문헌

[1] CWE-121 Stack-based Buffer Overflow, http://cwe.mitre.org/data/definitions/121.html

10. 힙에 할당된 버퍼 오버플로우(Heap-based Buffer Overflow)

가. 정의

힙에 할당되는 버퍼들 (예를 들어, malloc() 함수를 통해 할당된 버퍼들)에 문자열 등이 저장되어 질 때, 최초 정의된 힙의 메모리 사이즈를 초과하여 문자열 등이 저장되는 경우 버퍼 오버플로우가 발생한다.

나. 안전한 코딩기법

- 프로그램이 버퍼가 저장할 수 있는 것보다 많은 데이터를 입력하지 않는다.
- 프로그램이 버퍼 경계 밖의 메모리 영역을 참조하지 않는다.
- 프로그램이 사용할 메모리를 적절하게 계산하여 로직에서 에러를 발생시키지 않도록 한다.
- 입력에 대해서 경계 검사(bounds checking)를 한다.
- strlcpy()와 같이 버퍼 오버플로우 위험이 없는 함수를 사용한다.

다. 예제

특정 크기의 메모리를 할당한 후, **strcpy()** 함수를 이용하여 사용자 입력값을 그대로 메모리에 복사함으로써 버퍼 오버플로우가 발생할 수 있다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #define BUFSIZE 10
5:  int main(char **argv)
6:  {
7:     char *dest = NULL;
8:     dest = (char *)malloc(BUFSIZE);
9:     ……
10:    strcpy(dest, argv[1]);
11:    ……
12:    free(dest);
13:    return 0;
14: }
```

입력된 문자열에 대하여 길이를 제한하는 **strlcpy()** 함수를 사용해서 값을 저장한다. (**strlcpy()** 함수는 문자 배열의 마지막 부분에 자동적으로 Null을 채워준다.)

안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #define BUFSIZE 10
```

```
5:  int main(int argc, char **argv)
6:  {
7:      char *dest = NULL;
8:      dest = (char *)malloc(BUFSIZE);
9:      ……
10:     strlcpy(dest, argv[1], BUFSIZE);
11:     ……
12:     free(dest);
13:     return 0;
14: }
```

다음의 예제는 외부입력 queryStr에서 문자열을 받아와 strtok함수로 자르고, malloc으로 할당한 heap공간에 입력하는데, 공격자가 스트링 길이를 충분히 길게 하여 token의 개수를 MAX_DATA_BUFFER_LENGTH보다 많게 만들면 버퍼 오버플로우를 발생시킬 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #include <regex.h>
5:
6:  #include <sqltypes.h>
7:  #include <sql.h>
8:
9:  static SQLHSTMT statmentHandle;
10:
11: const char * GetParameter(const char * queryString, const char * key);
12:
13: static const char * UPDATE_DATA = "update_data";
14: static const char * DATA_PARAM = "data";
15: static const int MAX_DATA_BUFFER_LENGTH = 256;
16:
17: const int EQUAL = 0;
18: int main(void)
19: {
20:     char * queryStr = getenv("QUERY_STRING");
21:     if (queryStr == NULL)
22:     {
23:             // Error 처리
24:             ...
25:     }
26:
27:     char * command = GetParameter(queryStr, "command");
28:     if (strcmp(command, UPDATE_DATA) == EQUAL)
29:     {
```

```
30:             int * dataBuffer= (int *)malloc(sizeof(int) * MAX_DATA_BUFFER_LENGTH);
31:             int dataIndex = 0;
32:             const char * data = GetParameter(queryStr, DATA_PARAM);
33:
34:             char * token  = strtok(data, " ");
35:             whle(token != NULL)
36:             {
37:                     dataBuffer[dataIndex]         = atoi(token);
38:                     token = strtok(NULL, " ");
39:                     dataIndex++;
40:             }
41:             ...
42:             free(dataBuffer);
43:     }
44:     ...
45:     return EXIT_SUCCESS;
46: }
```

heap메모리의 인덱스 영역을 MAX_DATA_BUFFER_LENGTH와 비교하여 오버플로우 조건이 되면 더 이상 메모리에 값을 입력하지 않도록 하였다.

■ 안전한 코드의 예 - C

```
7:  #include <stdio.h>
8:  #include <stdlib.h>
9:  #include <string.h>
10: #include <regex.h>
11:
12: #include <sqltypes.h>
13: #include <sql.h>
14:
15: static SQLHSTMT statmentHandle;
16:
17: const char * GetParameter(const char * queryString, const char * key);
18:
19: static const char * UPDATE_DATA = "update_data";
20: static const char * DATA_PARAM = "data";
21: static const int MAX_DATA_BUFFER_LENGTH = 256;
22:
23: const int EQUAL = 0;
24: int main(void)
25: {
26:     char * queryStr = getenv("QUERY_STRING");
27:     if (queryStr == NULL)
28:     {
29:             // Error 처리
```

```
30:             ...
31:     }
32:
33:     char * command = GetParameter(queryStr, "command");
34:     if (strcmp(command, UPDATE_DATA) == EQUAL)
35:     {
36:             int * dataBuffer= (int *)malloc(sizeof(int) * MAX_DATA_BUFFER_LENGTH);
37:             int dataIndex = 0;
38:             const char * data = GetParameter(queryStr, DATA_PARAM);
39:
40:             char * token  = strtok(data, " ");
41:             whle(token != NULL)
42:             {
43:                     if(dataIndex >= MAX_DATA_BUFFER_LENGTH) break;
44:                     dataBuffer[dataIndex]         = atoi(token);
45:                     token = strtok(NULL, " ");
46:                     dataIndex++;
47:             }
48:             ...
49:             free(dataBuffer);
50:     }
51:     ...
52:     return EXIT_SUCCESS;
53: }
```

라. 참고문헌

[1] CWE-122 Heap-based Buffer Overflow, http://cwe.mitre.org/data/definitions/122.html

11. LDAP 처리(Improper Neutralization of Special Elements used in an LDAP Query, LDAP Injection)

가. 정의

LDAP 질의문이나 결과로 외부 입력이 적절한 처리 없이 사용되면 LDAP 질의문이 실행될 때 공격자는 LDAP 질의문의 내용을 마음대로 변경할 수 있다. 특히 여기서는 외부 입력이 LDAP 질의문 자체에 영향을 주는 경우를 뜻한다.

나. 안전한 코딩기법

- 외부의 입력이 직접 루트 디렉터리 접근에 사용되지 않도록 한다.
- 사용할 경우 적절한 접근제어를 수행하여야 한다.

다. 예제

다음의 예제에서 **manager**로 입력받은 외부 문자열이 **snprintf()** 함수를 사용하여 LDAP 함수에 베이스 문자열로 전달되었다. 이 경우 임의의 루트 디렉터리에 접근할 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <ldap.h>
3:  #define MAX 256
4:  int LDAPbind(LDAP *ld, FILE *file, char *password, char *dn)
5:  {
6:    char base[MAX];
7:    char manager[MAX-10];
8:    int rc;
9:    LDAPMessage* result;
10:   fgets(manager,sizeof(manager),file);
11:   snprintf(base, sizeof(base), "(user=%s)", manager);
12:   ldap_simple_bind_s(ld, manager, password);
13:   rc = ldap_search_ext_s( ld, base, LDAP_SCOPE_BASE, "manager=m1", NULL,
      0, NULL,          NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
14:   return rc;
15: }
```

상수로 베이스를 설정하여 위험성을 제거하였다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <ldap.h>
3:  #define MAX 256
4:  int LDAPbind(LDAP *ld, char *username, char *password, char *dn)
5:  {
```

```
6:      int rc;
7:      LDAPMessage* result;
8:      ldap_simple_bind_s(ld, username, password);
9:      rc=ldap_search_ext_s(ld,"ou=NewHires", LDAP_SCOPE_BASE, "manager=m1",
        NULL, 0, NULL, NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
10:     return rc;
11:   }
```

라. 참고문헌

[1] CWE-90 Improper Neutralization of Special Elements used in an LDAP Query (LDAP Injection), http://cwe.mitre.org/data/definitions/90.html

12. 시스템 또는 구성 설정의 외부 제어(External Control of System or Cofiguration Setting)

가. 정의

외부에서 시스템 설정 또는 구성 요소를 제어할 수 있다. 이로 인하여 서비스가 중단될 수 있으며, 예상치 못한 결과가 발생하거나 악용될 가능성이 있다.

나. 안전한 코딩기법

- sethostid() 함수의 인수로 외부 입력을 직접 사용하지 않는 것이 바람직하다.

다. 예제

다음의 예제는 명령줄 인수를 사용하여 현재 시스템의 호스트 ID를 변경하도록 하였다. 물론 **sethostid()** 함수가 제대로 호출되기 위해서는 권한이 필요하지만 여타 다른 기법을 사용하면 특권 없는 사용자가 프로그램을 실행시킬 수도 있다. 악의적인 공격자가 호스트 ID를 마음대로 변경할 수 있다면 네트워크상에서 시스템이 이상하게 식별될 수 있어 다른 동작을 할 수 있다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdlib.h>
2:  #include <unistd.h>
3:  main(int argc, char *argv[])
4:  {
5:      sethostid(atol(argv[1]));
6:  }
```

외부 입력에 따라 호스트 ID를 수정하지 말아야 한다.

안전한 코드의 예 - C

```
1:  #include <stdlib.h>
2:  #include <unistd.h>
3:  main(int argc, char *argv[])
4:  {
5:      sethostid(0xC0A80101);
6:  }
```

다음의 예제는 FTP 서비스의 포트 번호를 외부에서 받아서 설정하도록 하고 있다. 이 포트 번호가 만약 기존에 사용되고 있는 포트 번호라면 FTP 서비스나 기존 서비스 중 최소한 1개 이상은 오동작하게 된다.

■ 안전하지 않은 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:
3:  int main(void)
4:  {
5:    const char * COMMAND_PARAM  = "command";
6:    const char * CHANGE_FTP_PORT = "change_ftp_port";
7:
8:    char * queryStr;
9:    queryStr = getenv("QUERY_STRING");
10:
11:   if (queryStr == NULL)
12:   {
13:     // Error 처리
14:     ...
15:   }
16:   char * command = GetParameter(queryStr, COMMAND_PARAM);
17:   if (!strcmp(command, CHANGE_FTP_PORT))
18:   {
19:     ...
20:     const char * PORT_PARM = "port";
21:     const char * servicePort = GetParameter(queryStr, PORT_PARM);
22:     ...
23:     ChangeServicePort(FTP_SERIVCE, servicePort);
24:
25:     free(servicePort);
26:   }
27:   return EXIT_SUCCESS;
28: }
```

다음의 예제는 FTP 서비스의 포트 번호를 외부에서 입력 받은 값이 아니라, 프로그램에서 미리 설정한 값을 사용하게 하였다. 미리 설정한 값만을 사용함으로써, 기존에 사용되는 포트 번호로 FTP 서비스의 포트 번호가 변경되는 것을 막을 수 있다.

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:
3:  int main(void)
4:  {
5:    const char * COMMAND_PARAM  = "command";
6:    const char * CHANGE_FTP_PORT = "change_ftp_port";
7:
8:    const int DEFAULT_FTP_PORT  = 21;
9:    const int ALTERNATIVE_FTP_PORT  = 2121;
```

```
10:
11:     char * queryStr;
12:     queryStr = getenv("QUERY_STRING");
13:
14:     if (queryStr == NULL)
15:     {
16:       // Error 처리
17:       ...
18:     }
19:
20:     char * command = GetParameter(queryStr, COMMAND_PARAM);
21:     if (!strcmp(command, CHANGE_FTP_PORT))
22:     {
23:       ...
24:       const char * PORT_INDEX_PARM = "port_index";
25:       const char * servicePortIndex = GetParameter(queryStr, PORT_INDEX_PARM);
26:       ...
27:       ChangeServicePort(FTP_SERIVCE, servicePort);
28:
29:       if(servicePortIndex == DEFAULT)
30:       {
31:         ChangeServicePort(FTP_SERVICE, DEFAULT_FTP_PORT);
32:       }
33:       else if(servicePortIndex == ALTERNATIVE)
34:       {
35:         ChangeServicePort(FTP_SERVICE, ALTERNATIVE_FTP_PORT);
36:       }
37:
38:       free(servicePort);
39:     }
40:     return EXIT_SUCCESS;
41:   }
```

라. 참고문헌

[1] CWE-15 External Control of System or Configuration Setting, http://cwe.mitre.org/data/definitions/15.html

13. 프로세스 제어(Process Control)

가. 정의

신뢰되지 않은 소스나 신뢰되지 않은 환경으로부터 라이브러리를 적재하거나 명령을 실행하면 악의적인 코드의 실행이 가능하다.

나. 안전한 코딩기법

- 라이브러리 적재 시 상대 경로보다 신뢰할 수 있는 디렉터리의 절대 경로를 사용한다.
- 신뢰되지 않은 라이브러리의 경우 소스코드를 검사한 다음 사용한다.

다. 예제

dlopen() 함수에 전달되는 인수인 **filename**이 환경 변수 값에 의해 영향을 받는다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <dlfcn.h>
3:  int main()
4:  {
5:     char *filename;
6:     int *handle;
7:     filename = getenv("SHAREDFILE");
8:     /* RRLD_LAZY: 동적 라이브러리가 실행되면서 코드의 정의되지 않은 심볼 처리*/
9:     if ((handle = dlopen(filename, RTLD_LAZY)) != NULL)
10:    {
11:       exit(1);
12:    }
13:    ...
14:    return 0;
15: }
```

라이브러리의 경로는 외부에서 입력받지 않도록 하며 절대 경로로 명시하여야 한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <dlfcn.h>
3:  int main()
4:  {
5:     char *filename;
6:     int *handle;
7:     /* 절대경로 지정을 통해 해당 경로에서만 라이브러리를 찾음*/
8:     filename = "/usr/lib/hello.so";
9:     if ((handle = dlopen(filename, RTLD_LAZY)) != NULL);
10:    {
```

```
11:         exit(1);
12:     }
13:     ...
14:     return 0;
15: }
```

외부에서 입력받은 **documentName**와 **additionalOperation**를 그대로 이용하여 **system**함수에서 사용하고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const static char * COMMAND_PARAM  = "command";
10: const static char * UPLOAD_DOCUMENT_COMMAND      = "upload_document";
11: const static char * DOCUMENT_NAME_PARAM          = "document_name";
12: const static char * CONVERT_OPERATION_PARAM      = "convert_operation";
13:
14: int main(void)
15: {
16:     char * queryStr;
17:     queryStr = getenv("QUERY_STRING");
18:     if (queryStr == NULL)
19:     {
20:         // Error 처리
21:         ...
22:     }
23:     string command  = GetRequestParameter(queryStr, COMMAND_PARAM);
24:
25:     if(command.compare(UPLOAD_DOCUMENT_COMMAND) == 0)
26:     {
27:         string documentName  =                  GetRequestParameter(queryStr,
    DOCUMENT_NAME_PARAM);
28:         string additionalOperation      =      GetRequestParameter(queryStr,
    CONVERT_OPERATION_PARAM);
29:
30:         if(additionalOperation != null)
31:         {
32:             // 문서 format이 html이 아닐 경우, html로 변환을 수행
33:             ...
```

```
34:                     additionalOperation          = additionalOperation + " " + documentName;
35:                     system(additionalOperation);
36:                     ...
37:             }
38:             ...
39:     }
40:     ...
41: }
```

입력받은 **additionalOperation**을 미리 정의해둔 **additionalOperations**와 매치시켜 통과된 값을 이용하여 **OS Command**를 생성한 후 사용한다.

■ 안전한 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const static char * COMMAND_PARAM  = "command";
10: const static char * UPLOAD_DOCUMENT_COMMAND       = "upload_document";
11: const static char * DOCUMENT_NAME_PARAM           = "document_name";
12: const static char * CONVERT_OPERATION_PARAM       = "convert_operation";
13: static map<string, string>          additionalOperations;
14:
15: int main(void)
16: {
17:     additionalOperations.insert("DocToHtml", "convertDocToHtml");
18:     additionalOperations.insert("UTF16ToUTF8", "changeEncoding");
19:     ...
20:
21:     char * queryStr;
22:     queryStr = getenv("QUERY_STRING");
23:     if (queryStr == NULL)
24:     {
25:         // Error 처리
26:         ...
27:     }
28:
29:     string command  = GetRequestParameter(queryStr, COMMAND_PARAM);
30:
31:     if(command.compare(UPLOAD_DOCUMENT_COMMAND) == 0)
32:     {
```

```
33:         string documentName = GetRequestParameter(queryStr, DOCUMENT_NAME_PARAM);
34:         string additionalOperation = GetRequestParameter(queryStr, CONVERT_OPERATION_PARAM);
35:
36:         if(additionalOperation != null)
37:         {
38:             // 문서 format이 html이 아닐 경우, html로 변환을 수행
39:             ...
40:
41:             string osCommand= additionalOperations[additionalOperation] + " " + documentName;
42:             system(osCommand);
43:             ...
44:         }
45:         ...
46:     }
47:     ...
48: }
```

라. 참고문헌

[1] CWE-114 Process Control, http://cwe.mitre.org/data/definitions/114.html

[2] 2010 OWASP Top 10 - A4 Insecure Direct Object Reference, https://www.owasp.org/index.php/Top_10_2010-A4

14. 버퍼 시작 지점 이전에 쓰기(Buffer Underwrite, Buffer Underflow)

가. 정의

포인터나 인덱스를 통해서 버퍼 시작 지점 이전에 데이터를 기록하는 오류이다.
배열인 경우 인덱스의 최저 위치 보다 적은 위치에 데이터를 기록하는 경우 발생하며 루프 내에서 범위 점검 없이 포인터 감소 연산을 계속했을 경우 발생한다.

나. 안전한 코딩기법

- 버퍼에 인덱스나 포인터를 사용해서 데이터를 기록할 때 인덱스가 음수 값이 되거나 포인터 연산의 결과가 버퍼 이전의 값을 가지지 않도록 점검 후 사용한다.

다. 예제

음수 인덱스를 사용하여 버퍼에 접근함으로써 취약점을 발생시킨다.

안전하지 않은 코드의 예 - C

```
1:  int main()
2:  {
3:    int a[10];
4:    a[-1] = 0;
5:    return 0;
6:  }
```

버퍼의 시작 지점 위치를 명확하게 제시한다.

안전한 코드의 예 - C

```
1:  int main()
2:  {
3:    int a[10];
4:    a[0] = 0;
5:    return 0;
6:  }
```

words의 크기는 이미 정해져 있는 반면, 입력될 스트링의 길이는 알 수 없는 상태에서 작업이 스트링이 끝날 때까지 무한으로 반복되므로 오버플로우가 일어날 가능성이 크다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  void inverseWordOrder(char * string)
3:  {
4:    const int MAX_WORD_COUNT      = 256;
5:    int wordIndex     = MAX_WORD_COUNT - 1;
6:    char * words[MAX_WORD_COUNT];
```

```
7:      memset(words, NULL, MAX_WORD_COUNT * sizeof(int));
8:      char * token        = strtok(string, " \t");
9:      while(token != NULL)
10:     {
11:             words[wordIndex]        = token;
12:             token = strtok(NULL, " \t");
13:             wordIndex--;
14:     }
15: }
```

try catch구문으로 예기치 못한 에러 발생을 미리 대비해둔다.

■ 안전한 코드의 예 - C

```
1:  #include <vector>
2:  #include <stdio.h>
3:  using namespace std;
4:
5:  void inverseWordOrder(char * string)
6:  {
7:      const int MAX_WORD_COUNT = 256;
8:      vector<char *> words(MAX_WORD_COUNT, NULL);
9:      int wordIndex      = MAX_WORD_COUNT - 1;
10:
11:     try
12:     {
13:             char * token  = strtok(string, " \t");
14:             while(token != NULL)
15:             {
16:                     words.at(wordIndex)          = token;
17:                     token = strtok(NULL, " \t");
18:                     wordIndex--;
19:             }
20:     }
21:     catch(const std::exception & exception)
22:     {
23:             // out_of_range에 대한 exception 처리
24:     }
25: }
```

라. 참고문헌

[1] CWE-124 Buffer Underwrite(Buffer Underflow),
http://cwe.mitre.org/data/definitions/124.html

15. 범위 초과해서 읽기(Out-of-Bounds Read)

가. 정의

버퍼의 범위를 벗어나서 읽는 경우에 발생하는 취약점이다. 배열인 경우 그 배열이 가지는 인덱스의 범위가 있다. 그러나 프로그램이 최대의 인덱스 값보다 많은 값으로 배열을 참조하는 경우에 발생한다. 대부분 인덱스의 최대값에 대한 유효성 점검 없이 배열을 참조하는 경우 발생한다.

나. 안전한 코딩기법

- 버퍼의 범위를 벗어난 곳에서 읽는 경우를 방지한다. 버퍼의 공간 내의 읽기가 아닌 버퍼의 시작 전의 위치나 버퍼의 공간 뒤의 메모리를 읽게 되면 의도하지 않은 오류가 발생한다.

다. 예제

배열이 가지는 범위를 벗어난 값을 읽어오고 있다.

안전하지 않은 코드의 예 - C

```
1:  int main()
2:  {
3:     int a[3] = {1,2,3};
4:     int b = a[3];
5:     return 0;
6:  }
```

배열의 범위 내에 존재하는 값을 읽어와야 한다.

안전한 코드의 예 - C

```
1:  int main()
2:  {
3:     int a[3] = {1,2,3};
4:     int b = a[2];
5:     return 0;
6:  }
```

다음의 예제는 **END_MARK**가 발견될 때까지 **data_pool**의 인덱스인 **curIdx**를 증가시키고 있다. 만약 **data_pool**의 어디에도 **END_MARK**가 없다면 **data_pool**의 끝부분보다 큰 숫자의 인덱스 값으로 접근을 시도하여 out-of-bound read가 발생한다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  int readNextData(int * data_pool)
3:  {
4:     static int curIdx  = 0;
```

```
5:      int END_MARK  = -1;
6:      // End mark를 넣는 것을 잊어 먹었을 경우, 에러 발생
7:      if(data_pool[curIdx] != END_MARK)
8:      {
9:              curIdx++;
10:             return data_pool[curIdx];
11:     }
12:     return END_MARK;
13: }
```

data_pool이 허용 가능한 인덱스의 최대값(**size**)을 같이 입력값으로 받아 인덱스의 값이 허용된 최대 수치의 초과 여부를 체크한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:
3:  int readNextData(int * data_pool, int size)
4:  {
5:      static int curIdx  = 0;
6:      int END_MARK  = -1;
7:      if(curIdx >= size)
8:      {
9:              return END_MARK;
10:     }
11:     // End mark를 넣는 것을 잊어 먹었을 경우, 에러 발생
12:     if(data_pool[curIdx] != END_MARK)
13:     {
14:             curIdx++;
15:             return data_pool[curIdx];
16:     }
17:     return END_MARK;
18: }
```

라. 참고문헌

[1] CWE-125 Out-of-bounds Read, http://cwe.mitre.org/data/definitions/125.html

[2] CWE-129 Improper Validation of Array Index, http://cwe.mitre.org/data/definitions/129.html

[3] CWE-131 Incorrect Calculation of Buffer Size, http://cwe.mitre.org/data/definitions/131.html

[4] CWE-805 Buffer Access with Incorrect Length Value, http://cwe.mitre.org/data/definitions/805.html

[5] 2011 SANS Top 25 - RANK 20 (CWE-131), http://cwe.mitre.org/top25/

16. 검사되지 않은 배열 인덱싱(Improper Validation of Array Index)

가. 정의

외부 인자에 대한 적절한 검사 없이 그 인자가 배열 참조의 인덱스로 사용하는 경우이다. 외부의 인자 값이 버퍼의 인덱스로 사용이 되는 경우 해당 인덱스가 0 보다 같거나 크고, 버퍼의 크기보다 작은지 검사를 하고 사용해야 한다.

배열 인덱스를 사용 또는 계산시 신뢰할 수 없는 입력값을 사용하거나, 참조하는 배열 인덱스의 유효성을 검사하지 않을 경우, 시스템 마비 등의 문제를 발생시킨다.

나. 안전한 코딩기법

- 배열 정의시 인덱스의 최대치를 정하고, 배열 참조시 인덱스가 배열의 유효값 내에 있는지 검사한다.
- 루프 안에서 비교 구문을 쓸 때, '>' 보다는 '>='를 사용한다.

다. 예제

num의 허용된 최대치를 초과하는지 검사하지 않고 **sizes** 배열의 특정 인덱스에 **size**를 할당하고 있다. 배열의 범위를 초과한 인덱스 참조는 허가되지 않은 메모리 참조 또는 시스템 마비를 야기한다.

■ 안전하지 않은 코드의 예 - C

```
1:   int getsizes(int sock, int count, int *sizes)
2:   {
3:       char buf[BUFFER_SIZE];
4:       int ok;
5:       int num, size;
6:
7:       while ((ok = gen_recv(sock, buf, sizeof(buf))) == 0)
8:       {
9:       if (hasDotInBuffer(buf))
10:              break;
11:      else if (sscanf(buf,"%d %d", &num, &size) == 2)
12:              sizes[num-1] = size;
13:      }
14:        ...
15:  }
```

buf 변수로부터 입력값에 대한 **num** 값이 허용된 최대치를 초과하는지 검사한다.

■ 안전한 코드의 예 - C

```
1:   int getsizes(int sock, unsigned int MAXCOUNT, int *sizes)
2:   {
```

```
3:      char buf[BUFFER_SIZE];
4:      int ok;
5:      int num, size;
6:
7:      while ((ok = gen_recv(sock, buf, sizeof(buf))) == 0)
8:      {
9:              if (hasDotInBuffer(buf)) break;
10:   // buf를 num, size로 입력받는다.
11:             if (sscanf(buf,"%d %d", &num, &size) == 2)
12:             {
13:   // num의 최대치를 검사한다.
14:                     if (num > 0 && num <= MAXCOUNT)
15:                             sizes[num-1] = size;
16:                     else { printf("에러"); return(FAIL); }
17:             }
18:     }
19:     ...
20:   }
```

data배열의 인덱스 값인 **rowIndex**와 **colIndex**의 값이 배열의 최대 크기를 초과하는지 검사하지 않고 data로부터 데이터를 가져와서 사용하고 있다. 배열의 범위를 초과한 인덱스 참조는 허가되지 않은 메모리 참조 또는 시스템 마비를 야기한다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const static char * COMMAND_PARAM   = "command";
10: const static char * GET_CELL_DATA   = "get_cell_data";
11: const static char * ROW_INDEX_PARAM   = "row_index";
12: const static char * COLUMN_INDEX_PARAM   = "col_index";
13:
14: int main(void)
15: {
16:   char * queryStr;
17:   queryStr = getenv("QUERY_STRING");
18:   if (queryStr == NULL)
19:   {
20:     // Error 처리
21:     ...
```

```
22:     }
23:
24:     string command  = GetRequestParameter(queryStr, COMMAND_PARAM);
25:
26:     if(command.compare(GET_CELL_DATA) == 0)
27:     {
28:       int rowIndex = atoi(GetRequestParameter(queryStr, ROW_INDEX_PARAM));
29:       int colIndex = atoi(GetRequestParameter(queryStr, COLUMN_INDEX_PARAM));
30:
31:       ..
32:       print data[rowIndex][colIndex];
33:       ...
34:     }
35:     ...
36: }
```

rowIndex와 **colIndex**의 값이 배열의 최대 크기를 초과하는지 먼저 검사한 후 **data**배열의 인덱스로 사용한다.

■ 안전한 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const static char * COMMAND_PARAM  = "command";
10: const static char * GET_CELL_DATA  = "get_cell_data";
11: const static char * ROW_INDEX_PARAM  = "row_index";
12: const static char * COLUMN_INDEX_PARAM  = "col_index";
13:
14: int main(void)
15: {
16:   char * queryStr;
17:   queryStr = getenv("QUERY_STRING");
18:   if (queryStr == NULL)
19:   {
20:     // Error 처리
21:     ...
22:   }
23:   string command  = GetRequestParameter(queryStr, COMMAND_PARAM);
24:
25:   if(command.compare(GET_CELL_DATA) == 0)
```

```
26:     {
27:       int rowIndex = atoi(GetRequestParameter(queryStr, ROW_INDEX_PARAM));
28:       int colIndex = atoi(GetRequestParameter(queryStr, COLUMN_INDEX_PARAM));
29:
30:       ..
31:       if((rowIndex < MAX_DATA_ROW) && (colIndex < MAX_DATA_COL))
32:       {
33:         print data[rowIndex][colIndex];
34:       }
35:       ...
36:     }
37:     ...
38:   }
```

라. 참고문헌

[1] CWE-129 Improper Validation of Array Index, http://cwe.mitre.org/data/definitions/129.html

[2] CWE-120 Buffer Copy without Checking Size of Input(Classic Buffer Overflow), http://cwe.mitre.org/data/definitions/120.html

[3] 2011 SANS Top 25 - RANK 3 (CWE-120), http://cwe.mitre.org/top25

[4] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 5, "Public Enemy #1: The Buffer Overrun" Page 127. 2nd Edition. Microsoft. 2002

[5] M. Howard, D. LeBlanc and J. Viega. "24 Deadly Sins of Software Security". "Sin 5: Buffer Overruns." Page 89. McGraw-Hill. 2010

17. 널 종료 문제(Improper Null Termination)

가. 정의

C에서 문자열은 문자의 배열인데, 문자열의 끝을 나타내는 것이 널 문자이다. 널 문자가 붙지 않은 문자열은 프로그램 작동 시 여러 오류를 발생시킬 수 있다. 개발자의 의도와 상관없이 널 문자가 붙지 않은 문자열이 생성될 수 있는데, (1) 문자열이 필요한 크기보다 작은 배열에 문자열을 복사하는 경우 (2) 문자열을 strncpy() 함수를 사용하여 복사할 때 실제 문자열보다 지정 크기가 작은 경우 등에 발생한다. 널 문자로 종료되지 않은 문자열을 다른 문자열에 복사할 때, 많은 양의 메모리가 복사되어 버퍼 오버플로우가 발생 가능하다.

나. 안전한 코딩기법

- read(), readlink()로 읽은 문자열에 strcpy(), strcat(), strlen() 함수를 적용하면 안된다.
- strlcpy()나 strlcat() 과 같은 버퍼 오버플로우 위험이 없는 함수를 사용한다.

다. 예제

read() 함수를 이용해 읽어온 **inputbuf** 문자열은 널 문자로 끝나는 것이 보장되지 않는다. 이러한 경우 **strcpy()** 함수를 사용하면 문자열이 널 문자로 끝나는 것으로 간주하므로 주어진 범위를 넘어서 메모리를 접근하게 된다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <string.h>
3:  #include <unistd.h>
4:  #define MAXLEN 1024
5:  extern char *inputbuf;
6:  extern int cfgfile;
7:  void readstr()
8:  {
9:    char buf[MAXLEN];
10:   read(cfgfile, inputbuf, MAXLEN);
11:   strcpy(buf, inputbuf);
12: }
```

read() 함수를 이용해 읽어온 문자열의 수를 **MAXLEN**으로 정의한 길이만큼 **strcpy()** 함수를 사용하여 복사한다.

안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <string.h>
3:  #include <unistd.h>
4:  #include <string.h>
```

```
5:  #define MAXLEN 1024
6:  extern char *inputbuf;
7:  extern int cfgfile;
8:  void readstr()
9:  {
10:   char buf[MAXLEN];
11:   read(cfgfile, inputbuf, MAXLEN);
12:   strlcpy(buf, inputbuf, MAXLEN);
13: }
```

다음의 예제에서는 **userID**로부터 **firstName**과 **lastName**을 추출하여 **fullName**배열에 길이에 대한 측정 없이 대입하고 있다. 만약 **firstName**과 **lastName**의 길이 합이 **MAX_NAME_LENGTH** 보다 큰 경우 배열의 한계 바깥쪽의 메모리를 접근할 가능성이 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:  using namespace std;
7:  const static char * COMMAND_PARAM = "command";
8:  const static char * GET_USER_NAME = "get_user_name";
9:  const static char * ID_PARAM = "user_id";
10: int main(void)
11: {
12:   const int MAX_NAME_LENGTH = 32;
13:   char * queryStr;
14:   queryStr = getenv("QUERY_STRING");
15:   if (queryStr == NULL)
16:   {
17:     // Error 처리
18:     ...
19:   }
20:   string command = GetRequestParameter(queryStr, COMMAND_PARAM);
21:   if(command.compare(GET_CELL_DATA) == 0)
22:   {
23:     char * userId = GetRequestParameter(queryStr, ID_PARAM);
24:     const char * firstName = GetFirstName(userId);
25:     const char * lastName  = GetLastNam(userId);
26:     char fullName[MAX_NAME_LENGTH];
27:     strncpy(fullName, firstName, MAX_NAME_LENGTH);
28:     strncat(fullName," ", MAX_NAME_LENGTH - strlen(fullName));
29:     strncat(fullName, lastName, MAX_NAME_LENGTH - strlen(fullName));
```

```
30:         ...
31:     }
32:     ...
33: }
```

안전한 **String**객체를 이용하여 문자열을 저장한다.

■ 안전한 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const static char * COMMAND_PARAM  = "command";
10: const static char * GET_USER_NAME     = "get_user_name";
11: const static char * ID_PARAM  = "user_id";
12:
13: int MAX_NAME_LENGTH       = 32;
14:
15: int main(void)
16: {
17:     char * queryStr;
18:     queryStr = getenv("QUERY_STRING");
19:     if (queryStr == NULL)
20:     {
21:         // Error 처리
22:         ...
23:     }
24:
25:     string command   = GetRequestParameter(queryStr, COMMAND_PARAM);
26:
27:     if(command.compare(GET_CELL_DATA) == 0)
28:     {
29:         char * userId = GetRequestParameter(queryStr, ID_PARAM);
30:
31:         const char * firstName = GetFirstName(userId);
32:         const char * lastName          = GetLastNam(userId);
33:
34:         // string 객체를 사용한다.
35:         string fullName          = firstName;
36:         fullName    = fullName + " " + lastName;
37:         ...
```

```
38:     }
39:     ...
40: }
```

라. 참고문헌

[1] CWE-170 Improper Null Termination, http://cwe.mitre.org/data/definitions/170.html

[2] CWE-665 Improper Initialization, http://cwe.mitre.org/data/definitions/665.html

18. 의도하지 않은 부호 확장(Unexpected Sign Extension)

가. 정의

큰 수를 표현하는 데이터 형으로 값이 변환될 때 음수 값이 큰 양수 값으로 변환될 수 있다. 숫자형 데이터를 처리할 때 해당 데이터 형보다 큰 데이터 형으로 복사되는 경우, 부호 확장이 수행된다. 특히, 음수값을 unsigned로 복사하는 경우 잘못된 부호의 확장으로 엄청난 큰 숫자가 발생 할 수 있다.

나. 안전한 코딩기법

- 큰 사이즈의 변수와 작은 사이즈의 변수 사이의 값 교환 시 타입 체크를 통하여 잘못된 부호 확장이 일어나지 않도록 해야 한다.

다. 예제

4바이트인 int 값(**id**)이 2바이트인 **short** 변수(s)에 저장될 때 정수 오버플로우가 발생하고 다시 해당 값이 4바이트 **unsigned** 변수에 저장될 때 예기치 않은 부호 확장이 발생한다.

안전하지 않은 코드의 예 - C

```
1:  extern int info[256];
2:  int signed_overflow(int id)
3:  {
4:    short s;
5:    unsigned sz;
6:    /* int형 변수값을 short형 변수에 저장함으로써 오버플로우 발생 */
7:    s = id;
8:    if (s > 256) return 0;
9:    /* 위의 값을 unsigned형 변수에 저장함으로써 부호 확장 발생 */
10:   sz = s;
11:   return info[sz];
12: }
```

크기가 다른 변수의 값을 서로 교환하지 못하도록 필요한 값의 조건을 정확하게 검사하고 부득이하게 교환할 경우 형 변환을 통해 교환되는 값의 타입을 일치시켜야 한다.

안전한 코드의 예 - C

```
1:  extern int info[256];
2:  int signed_overflow(int id)
3:  {
4:    int s;
5:    unsigned sz;
6:    s = id;
7:    if (s > 256 || s < 0)
```

```
8:    return 0;
9:    sz = (unsigned) s;
10:   return info[sz];
11:   }
```

다음의 예제에서는 사용자의 행동에 점수를 매겨 잘못된 행위를 하면 유저등급을 깎는 코드이다. **userLevel**을 연산함에 있어 값의 범위를 고려하지 않고 연산을 가해 오버플로우 발생 가능성이 잠재하고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const static char * COMMAND_PARAM   = "command";
10: const static char * GET_DATA_COMMAND      = "get_data";
11: const static char * USER_ID_PARAM   = "user_id";
12: const static char * DATA_INDEX_PARAM   = "data_index";
13: const static char * FIELD_INDEX_PARAM   = "field_index";
14: const static char * VALUE_PARAM   = "value";
15:
16: static int NORMAL_USER_LEVEL   = 0x80000000;
17:
18: static int DATA_SIZE   = 256;
19:
20: int main(void)
21: {
22:   char * queryStr;
23:   queryStr = getenv("QUERY_STRING");
24:   if (queryStr == NULL)
25:   {
26:     // Error 처리
27:     ...
28:   }
29:
30:   string command   = GetRequestParameter(queryStr, COMMAND_PARAM);
31:
32:   if(command.compare(GET_DATA_COMMAND) == 0)
33:   {
34:     string userId   = GetRequestParameter(queryStr, USER_ID_PARAM);
35:
```

```
36:         unsigned int userLevel   = GetBaseUserLevel(userId);
37:         userLevel   -= GetStrangeMistake(userId) * 0x40000000;
38:         userLevel   -= GetNormalMistake(userId) * 0x1000000;
39:
40:         if(userLevel < NOMAL_USER_LEVEL)
41:         {
42:           // 권한이 없으므로 error 처리
43:           return;
44:         }
45:         ...
46:     }
47:     ...
48: }
```

값을 연산할 때에는 반드시 결과값이 해당 변수의 데이터 타입 한계를 넘어 오버플로우가 되지 않는지 먼저 확인한 후 결과값을 다른 곳에 사용하는 것이 안전하다.

■ 안전한 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const static char * COMMAND_PARAM   = "command";
10: const static char * GET_DATA_COMMAND       = "get_data";
11: const static char * USER_ID_PARAM   = "user_id";
12: const static char * DATA_INDEX_PARAM   = "data_index";
13: const static char * FIELD_INDEX_PARAM   = "field_index";
14: const static char * VALUE_PARAM   = "value";
15:
16: static int NORMAL_USER_LEVEL   = 0x80000000;
17:
18: static int DATA_SIZE   = 256;
19:
20: int main(void)
21: {
22:   char * queryStr;
23:   queryStr = getenv("QUERY_STRING");
24:   if (queryStr == NULL)
25:   {
26:     // Error 처리
27:     ...
```

```
28:     }
29:
30:     string command   = GetRequestParameter(queryStr, COMMAND_PARAM);
31:
32:     if(command.compare(GET_DATA_COMMAND) == 0)
33:     {
34:       string userId   = GetRequestParameter(queryStr, USER_ID_PARAM);
35:
36:       // userLevel이 음수가 되지 않도록 조건 check를 해준다.
37:       unsigned int userLevel   = GetBaseUserLevel(userId);
38:       if(userLevel > GetStrangeMistake(userId) * 0x40000000)
39:       {
40:         userLevel   -= GetStrangeMistake(userId) * 0x40000000;
41:       }
42:       else
43:       {
44:         userLevel   = 0;
45:       }
46:
47:       if(userLevel > GetNormalMistake(userId) * 0x1000000)
48:       {
49:         userLevel   -= GetNormalMistake(userId) * 0x1000000;
50:       }
51:       else
52:       {
53:         userLevel   = 0;
54:       }
55:
56:       if(userLevel < NOMAL_USER_LEVEL)
57:       {
58:         // 권한이 없으므로 error 처리
59:         return;
60:       }
61:       ...
62:     }
63:     ...
64:   }
```

라. 참고문헌

[1] CWE-194 Unexpected Sign Extension, http://cwe.mitre.org/data/definitions/194.html

19. 무부호 정수를 부호 정수로 타입 변환 오류(Unsigned to Signed Conversion Error)

가. 정의

무부호 정수 (unsigned integer)를 부호 정수 (signed integer)로 변환하면서 큰 양수가 음수로 변환될 수 있다. 이 값이 배열의 인덱스로 사용되는 경우 배열의 한계 범위를 넘어서 접근이 될 수도 있어 프로그램 오동작이 발생된다.

나. 안전한 코딩기법

- 타입체크를 통해서 signed int를 묵시적으로 unsigned int로 변환되지 않도록 한다.

다. 예제

chunkSz() 함수는 **chunkSize**가 초기화 되어 있지 않은 경우 에러를 의미하는 리턴값인 -1을 반환한다. 그러나 **chunkCpy** 함수에서는 이 리턴값을 그대로 **memcpy**의 크기를 나타내는 파라미터로 사용한다. 이때 크기를 나타내는 변수인 **size**의 타입이 **unsigned**이므로 -1을 리턴 받으면 큰 양수 값으로 잘못 해석된다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdlib.h>
2:  #include <string.h>
3:  extern int initialized, chunkSize;
4:  int chunkSz()
5:  {
6:    if (!initialized) return -1;
7:    return chunkSize;
8:  }
9:  void* chunkCpy(void *dBuf, void *sBuf)
10: {
11:   unsigned size;
12:   size = chunkSz();
13:   return memcpy(dBuf, sBuf, size);
14: }
```

chunkSz() 함수의 리턴값을 **signed int** 변수에 받아 에러를 의미하는 음수값이 넘어왔는지 확인하고 그렇지 않은 경우에만 **signed int**를 **unsigned int**로 변환하여 사용한다. 이때는 **signed int**가 영 또는 양수이므로 잘못된 변환이 일어나지 않게 된다.

■ 안전한 코드의 예 - C

```
1:  #include <stdlib.h>
2:  #include <string.h>
3:  extern int initialized, chunkSize;
4:  int chunkSz()
```

```
5:  {
6:    if (! initialized) return -1;
7:    return chunkSize;
8:  }
9:  void* chunkCpy(void *dBuf, void *sBuf)
10: {
11:   int size;
12:   size = chunkSz();
13:   if (size < 0) return NULL;
14:   return memcpy(dBuf, sBuf, (unsigned)size);
15: }
```

다음의 예제는 유저의 슈퍼유저 등급을 함수를 통해 검증하는데, unsigned int로 선언한 **userLevel**변수를 signed int형태로서 **isSuperUser**의 입력값으로 넣었다. 이는 **userLevel**의 값에 따라 예기치 않은 오동작을 일으킬 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <iostream>
2:  #include <fstream>
3:  #include <string>
4:  #include <map>
5:  #include <set>
6:
7:  using namespace std;
8:
9:  const static char * COMMAND_PARAM  = "command";
10: const static char * GET_DATA_COMMAND      = "get_data";
11: const static char * USER_ID_PARAM  = "user_id";
12: const static char * DATA_INDEX_PARAM  = "data_index";
13: const static char * FIELD_INDEX_PARAM  = "field_index";
14: const static char * VALUE_PARAM  = "value";
15:
16: static int SUPER_USER_LEVEL  = 0x70000000;
17:
18: static int DATA_SIZE  = 256;
19:
20: int main(void)
21: {
22:   char * queryStr;
23:   queryStr = getenv("QUERY_STRING");
24:   if (queryStr == NULL)
25:   {
26:     // Error 처리
27:     ...
28:   }
```

```
29:
30:     string command   = GetRequestParameter(queryStr, COMMAND_PARAM);
31:
32:     if(command.compare(GET_DATA_COMMAND) == 0)
33:     {
34:       string userId   = GetRequestParameter(queryStr, USER_ID_PARAM);
35:
36:       // userLevel이 음수가 되지 않도록 조건 check를 해준다.
37:       unsigned int userLevel   = GetBaseUserLevel(userId);
38:       userLevel   += GetWritingCount(userId) * 0x40000000;
39:       userLevel   += GetUploadCounting(userId) * 0x1000000;
40:
41:       if(isSuperUser(userLevel))
42:       {
43:         // 슈퍼 유저를 위한 특수 data 제공
44:         return;
45:       }
46:       ...
47:     }
48:     ...
49:   }
50:
51:   bool isSuperUser(int userLevel)
52:   {
53:     return userLevel > SUPER_USER_LEVEL;
54:   }
```

함수의 인자값을 정확하게 원래의 데이터 타입에 맞게 설정해 주어야 오동작을 방지할 수 있다.

■ 안전한 코드의 예 - C

```
1:   #include <iostream>
2:   #include <fstream>
3:   #include <string>
4:   #include <map>
5:   #include <set>
6:
7:   using namespace std;
8:
9:   const static char * COMMAND_PARAM   = "command";
10:  const static char * GET_DATA_COMMAND       = "get_data";
11:  const static char * USER_ID_PARAM   = "user_id";
12:  const static char * DATA_INDEX_PARAM   = "data_index";
13:  const static char * FIELD_INDEX_PARAM   = "field_index";
14:  const static char * VALUE_PARAM   = "value";
```

```
15:
16:   static unsigned int SUPER_USER_LEVEL  = 0x70000000;
17:
18:   static int DATA_SIZE  = 256;
19:
20:   int main(void)
21:   {
22:     char * queryStr;
23:     queryStr = getenv("QUERY_STRING");
24:     if (queryStr == NULL)
25:     {
26:       // Error 처리
27:       ...
28:     }
29:     string command  = GetRequestParameter(queryStr, COMMAND_PARAM);
30:
31:     if(command.compare(GET_DATA_COMMAND) == 0)
32:     {
33:       string userId  = GetRequestParameter(queryStr, USER_ID_PARAM);
34:
35:       unsigned int userLevel  = GetBaseUserLevel(userId);
36:       userLevel  += GetWritingCount(userId) * 0x40000000;
37:       userLevel  += GetUploadCounting(userId) * 0x1000000;
38:       if(isSuperUser(userLevel))
39:       {
40:         // 슈퍼 유저를 위한 특수 data 제공
41:         return;
42:       }
43:       ...
44:     }
45:     ...
46:   }
47:   // 인자와 SUPER_USER_LEVE의 type을 unsignd int로 변경하였음
48:   bool isSuperUser(unsigned int userLevel)
49:   {
50:     return userLevel > SUPER_USER_LEVEL;
51:   }
```

라. 참고문헌

[1] CWE-196 Unsigned to Signed Conversion Error, http://cwe.mitre.org/data/definitions/196.html

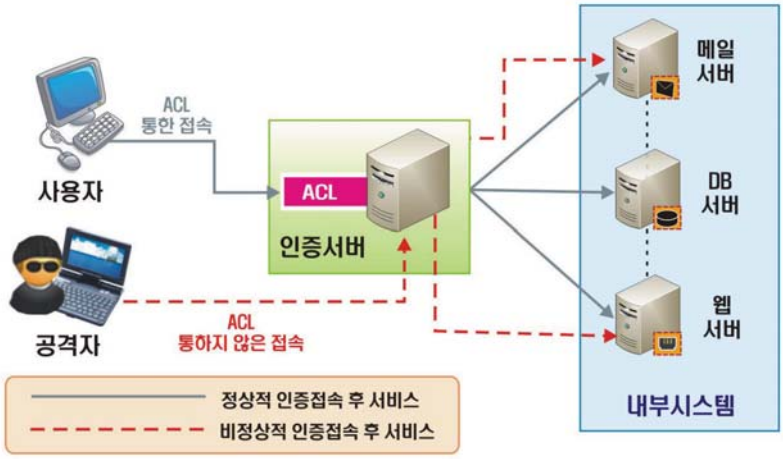
제2절 보안기능

기본적인 보안기능을 다룰 때는 세심한 주의가 필요하다. 부적절한 보안특성의 사용은 오히려 성능이나 부가적인 문제를 야기시킬 수 있다. 보안기능에는 인증, 접근제어, 기밀성, 암호화, 권한 관리 등이 포함된다.

1. 부적절한 인가(Improper Authorization)

가. 정의

SW가 모든 가능한 실행 경로에 대해서 접근제어를 검사하지 않거나 불완전하게 검사하는 경우, 공격자는 접근 가능한 실행 경로를 통해 접근하여 정보를 유출할 수 있다.



<그림 2-3> 부적절한 인가

나. 안전한 코딩기법

- 응용프로그램이 제공하는 정보와 기능을 역할에 따라 배분함으로써 공격자에게 노출되는 공격표면(attack surface)을 감소시킨다.
- 사용자의 권한에 따른 ACL(Access Control List)을 관리한다.

다. 예제

사용자 인증이 성공적으로 끝나면, 어떤 LDAP 검색도 가능하다. 또한 사용자의 로그인 정보와 인자로 넘겨오는 **username**의 비교가 없기 때문에 타인의 정보를 볼 수가 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #define FIND_DN "uid=han,ou=staff,dc=example,dc=com"
2:
3:  int searchData2LDAP(LDAP *ld, char *username)
4:  {
```

```
5:     unsigned long rc;
6:     char filter[20];
7:     LDAPMessage *result;
8:     snprintf(filter, sizeof(filter), "(name=%s)", username);
9:     rc = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter, NULL, 0, NULL,
       NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
10:    ……
11:    return rc;
12:  }
```

LDAP 검색을 수행하기 전에 **username**을 인증하고, **username**이 로그인 정보와 일치하는지 점검한다.

■ 안전한 코드의 예 - C

```
1:  if ( ldap_simple_bind_s(ld, username, password) != LDAP_SUCCESS )
2:  {
3:    printf("인증 에러");
4:    return(FAIL);
5:  }
6:  if ( strcmp(username,getLoginName()) != 0 )
7:  {
8:    printf("인증 에러");
9:    return(FAIL);
10: }
11: snprintf(filter, sizeof(filter), "(name=%s)", username);
12: rc = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter, NULL, 0, NULL, NULL,
    LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
```

다음의 예제는 게시판 읽기 / 쓰기 권한을 체크한 후 사용 권한을 부여하고 있다. **userID**와 **password**를 체크하는 부분은 있지만 각 유저의 읽기 / 쓰기 권한을 구분하지 않기 때문에 어떤 사용자라도 읽기 / 쓰기가 가능하다.

■ 안전하지 않은 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:    const char * COMMAND_PARAM = "command";
5:    const char * READ_DOCUMENT_CMD = "read_document";
6:    const char * WRITE_DOCUMENT_CMD = "write_document";
7:
8:    char * queryStr;
9:    queryStr = getenv("QUERY_STRING");
10:   if (queryStr == NULL)
11:   {
```

```
12:     // Error 처리
13:     ...
14:   }
15:   char * command = GetParameter(queryStr, COMMAND_PARAM);
16:   if (!strcmp(command, READ_DOCUMENT_CMD))
17:   {
18:     const char * USER_ID_PARAM = "user_id";
19:     const char * PASSWORD_PARAM = "password";
20:     const int MAX_PASSWORD_LENGTH =16;
21:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
22:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
23:
24:     if(isCorrectUser(userId, password))
25:     {
26:       ...
27:     }
28:
29:     ...
30:     free(userId);
31:     free(password);
32:   }
33:
34:   if (!strcmp(command, WRITE_DOCUMENT_CMD))
35:   {
36:     const char * USER_ID_PARAM = "user_id";
37:     const char * PASSWORD_PARAM = "password";
38:     const int MAX_PASSWORD_LENGTH = 16;
39:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
40:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
41:
42:     if(isCorrectUser(userId, password))
43:     {
44:       ...
45:     }
46:     ...
47:     free(userId);
48:     free(password);
49:   }
50:   return EXIT_SUCCESS;
51: }
```

userID를 몇 개의 그룹으로 나누고 그에 따라 권한 등급을 미리 설정해놓았다. 읽기 / 쓰기가 이루어지기 전에 권한에 대한 검사가 미리 이루어지고 충분한 권한이 있는 사용자 쓰기가 가능하게 되어 있다.

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:    const char * COMMAND_PARAM = "command";
5:    const char * READ_DOCUMENT_CMD = "read_document";
6:    const char * WRITE_DOCUMENT_CMD = "write_document";
7:
8:    const int READ_PERMISSION = 0x01;
9:    const int WRITE_PERMISSION = 0x02;
10:
11:   map<string, int> userGroupPermissions;
12:   userGroupPermissions["Normal"] = READ_PERMISSION | WRITE_PERMISSION;
13:   userGroupPermissions["Admin"] = READ_PERMISSION | WRITE_PERMISSION;
14:   userGroupPermissions["Free"] = READ_PERMISSION;
15:
16:   char * queryStr;
17:   queryStr = getenv("QUERY_STRING");
18:   if (queryStr == NULL)
19:   {
20:     // Error 처리
21:     ...
22:   }
23:   char * command = GetParameter(queryStr, COMMAND_PARAM);
24:   if (!strcmp(command, READ_DOCUMENT_CMD))
25:   {
26:     const char * USER_ID_PARAM = "user_id";
27:     const char * PASSWORD_PARAM = "password";
28:     const int MAX_PASSWORD_LENGTH = 16;
29:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
30:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
31:
32:     char * userGroup = getUserGroup(userId);
33:     if((userGroupPermissions.count(userGroup) == 0) || ((userGroupPermission[userGroup]
    & READ_PERMISSION)) == 0)
34:     {
35:       // 부적절한 사용에 대한 처리
36:       return;
37:     }
38:
39:     if(isCorrectUser(userId, password))
40:     {
41:       ...
42:     }
43:
```

```
44:      ...
45:      free(userId);
46:      free(password);
47:    }
48:
49:    if (!strcmp(command, WRITE_DOCUMENT_CMD))
50:    {
51:      const char * USER_ID_PARAM = "user_id";
52:      const char * PASSWORD_PARAM = "password";
53:      const int MAX_PASSWORD_LENGTH = 16;
54:      const char * userId = GetParameter(queryStr, USER_ID_PARAM);
55:      const char * password = GetParameter(queryStr, PASSWORD_PARAM);
56:
57:      char * userGroup = getUserGroup(userId);
58:      if((userGroupPermissions.count(userGroup) == 0) || ((userGroupPermission[userGroup]
       & WRITE_PERMISSION)) == 0)
59:      {
60:        // 부적절한 사용에 대한 처리
61:        return;
62:      }
63:
64:      if(isCorrectUser(userId, password))
65:      {
66:        ...
67:      }
68:      ...
69:      free(userId);
70:      free(password);
71:    }
72:    return EXIT_SUCCESS;
73:  }
```

라. 참고문헌

[1] CWE-285 Improper Authorization, http://cwe.mitre.org/data/definitions/285.html

[2] CWE-219 Sensitive Data Under Web Root, http://cwe.mitre.org/data/definitions/219.html

[3] 2010 OWASP Top 10 - A8 Failure to Restrict URL Access https://www.owasp.org/index.php/Top_10_2010-A8

[4] NIST. "Role Based Access Control and Role Based Security"

[5] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 4, "Authorization" Page 114; Chapter 6, "Determining Appropriate Access Control" Page 171. 2nd Edition. Microsoft. 2002

2. 중요한 자원에 대한 잘못된 권한허용(Incorrect Permission Assignment for Critical Resource)

가. 정의

SW가 중요한 보안관련 자원에 대하여 읽기 또는 수정하기 권한을 의도하지 않게 허가할 경우, 권한을 갖지 않은 사용자가 해당자원을 사용하게 된다.

나. 안전한 코딩기법

- 설정파일, 실행파일, 라이브러리 등은 SW 관리자에 의해서만 읽고 쓰기가 가능하도록 설정한다.
- 설정파일과 같이 중요한 자원을 사용하는 경우, 허가받지 않은 사용자가 중요한 자원에 접근 가능 여부를 검사한다.

다. 예제

파일 생성 시 가장 많은 권한을 허가하는 형태로 **umask**를 사용하고 있어 모든 사용자가 읽기/쓰기 권한을 갖게 된다.

■ 안전하지 않은 코드의 예 - C

```
1:  // 파일 권한 : rw-rw-rw-, 디렉터리 권한 : rwxrwxrwx
2:  umask(0);
3:  FILE *out = fopen("important_file", "w");
4:  if (out)
5:  {
6:      fprintf(out, "민감한 정보\n");
7:      fclose(out);
8:  }
```

파일에 대한 설정을 가장 제한이 많도록 즉, 사용자를 제외하고는 읽고 쓰기가 가능하지 않도록 **umask**를 설정하는 것이 필요하다.

■ 안전한 코드의 예 - C

```
1:  umask(077); // 파일 권한 : rw-------, 디렉터리 권한 : rwx------
2:  FILE *out = fopen("important_file", "w");
3:  if (out)
4:  {
5:      fprintf(out, "민감한 정보\n");
6:      fclose(out);
7:  }
```

다음의 예제에서는 패스워드 파일을 권한에 대한 확인 없이 열어볼 수 있게 허용하고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  int main()
2:  {
3:    ...
4:    FILE * fp = fopen("/etc/password/", "r");
5:    ...
6:  }
```

파일이름이 보호대상 파일 목록에 포함되어 있으면 admin권한을 가진 경우에만 파일 access를 허용하고 있다.

■ 안전한 코드의 예 - C

```
1:  // 보호 받아야 하는 file이면 관리자만 열수 있도록 함
2:
3:  set<string> protectedFiles;
4:  int main()
5:  {
6:    protectedFiles.insert("/etc/password");
7:    protectedFiles.insert("/root/keys.cfg");
8:    ...
9:    FILE * fp = secureFileOpen("/etc/password/", "r", "normal_user");
10:   ...
11: }
12: FILE * secureFileOpen(const char * filename, const char * mode, const char * userId)
13: {
14:   if(protectedFiles.count(filename) > 0){ // file이 보호 받아야 하는 file이면
15:     if(isAdminUserId(userId) == FALSE){
16:       return NULL;
17:     }
18:   }
19:   return fopen(filename, mode);
20: }
```

라. 참고 문헌

[1] CWE-732 Incorrect Permission Assignment for Critical Resource, http://cwe.mitre.org/data/definitions/732.html

[2] CWE-276 Incorrect Default Permissions, http://cwe.mitre.org/data/definitions/276.html

[3] CWE-277 Insecure Inherited Permissions, http://cwe.mitre.org/data/definitions/277.html

[4] CWE-278 Insecure Preserved Inherited Permissions, http://cwe.mitre.org/data/definitions/278.html

[5] CWE-279 Incorrect Execution-Assigned Permissions, http://cwe.mitre.org/data/definitions/279.html

[6] CWE-281 Improper Preservation of Permissions, http://cwe.mitre.org/data/definitions/281.html

[7] CWE-285 Improper Authorization, http://cwe.mitre.org/data/definitions/285.html

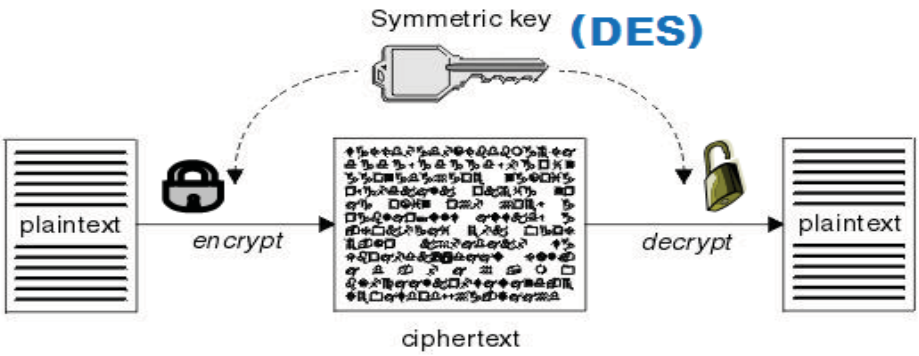
[8] 2011 SANS Top 25 –RANK 17 (CWE-732), http://cwe.mitre.org/top25/

3. 취약한 암호화 알고리즘 사용(Use of a Broken or Risky Cryptographic Algorithm)

가. 정의

SW 개발자들은 환경설정 파일에 저장된 패스워드를 보호하기 위하여 간단한 인코딩 함수를 이용하여 패스워드를 감추는 방법을 사용하기도 한다. 그렇지만 base64와 같은 지나치게 간단한 인코딩 함수를 사용하는 것은 패스워드를 제대로 보호할 수 없다.

정보보호 측면에서 취약하거나 위험한 암호알고리즘을 사용해서는 안 된다. 표준화되지 않은 암호알고리즘을 사용하는 것은 공격자가 알고리즘을 분석하여 무력화시킬 수 있는 가능성을 높일 수도 있다. 몇몇 오래된 암호알고리즘의 경우는 컴퓨터의 성능이 향상됨에 따라 취약해지기도 해서, 예전에는 해독하는데 몇 십 억년이 걸리던 알고리즘이 최근에는 며칠이나 몇 시간 내에 해독되기도 한다. RC2, RC4, RC5, RC6, MD4, MD5, SHA1, DES 알고리즘이 여기에 해당된다.



<그림 2-4> 취약한 암호화 알고리즘 사용

나. 안전한 코딩기법

- 3-DES, AES와 같은 암호화 알고리즘을 사용하여야 한다.
- 자신만의 암호화 알고리즘을 개발하는 것은 위험하며, 학계 및 업계에서 이미 검증된 표준화된 알고리즘을 사용한다. 기존에 취약하다고 알려진 DES, RC5등의 암호알고리즘을 대신하여, 3DES, AES, SEED 등의 안전한 암호알고리즘으로 대체하여 사용한다.
- 또한, 업무관련 내용, 개인정보등에 대한 암호알고리즘 적용시, IT보안인증 사무국이 안전성을 확인한 검증필 암호모듈을 사용해야한다.

참고 : 안전한 암호알고리즘 및 키길이

분류		보호함수 목록
최소 안전성 수준		112비트
블록암호		ARIA(키 길이 : 128/192/256), SEED(키 길이 : 128)
블록암호 운영모드	기밀성	ECD, CBC, CFB, OFB, CTR
	기밀성/인증	CCM, GCM
해쉬함수		SHA-224/256/384/512
메시지 인증코드	해쉬함수기반	HMAC
	블록기반	CMAC, GMAC
난수발생기	해쉬함수 /HMAC 기반	HASH_DRBG, HMAC_DRBG
	블록기반	CTR_DRBG
공개키 암호		RSAES - (공개키 길이) 2048, 3072 - RSA-OAEP에서 사용되는 해쉬함수 : SHA-224/256
전자서명		RSA-PSS, KCDSA, ECDSA, EC-KCDSA
키 설정 방식		DH, ECDH

보호함수		보호함수 파라미터
시스템 파라미터	RSA-PSS	(공개키 길이) 2048, 3072
	KCDSA, DH	(공개키 길이, 개인키 길이) (2048, 224), (2048, 256)
	ECDSA, EC-KCDSA, ECDH	(FIPS) B-233, B-283 (FIPS) K-233, K-283 (FIPS) P-224, P-256

※ 출처 : 암호알고리즘 검증기준 Ver 2.0(2012.3), 암호모듈시험기관

다. 예제

취약한 DES 알고리즘으로 암호화하고 있다. 3-DES, AES 알고리즘과 같이 보다 안전한 암호화 알고리즘을 사용해야 한다.

■ 안전하지 않은 코드의 예 - C

```
1:    ... ...
2:     EVP_CIPHER_CTX ctx;
3:     EVP_CIPHER_CTX_init(&ctx);
4:     EVP_EncryptInit(&ctx, EVP_des_ecb(), NULL, NULL);
5:    ... ...
```

DES 알고리즘보다 안전한 3-DES, AES 알고리즘을 사용해서 프로그램을 설계한다.

■ 안전한 코드의 예 - C

```
1. ... ...
2.    EVP_CIPHER_CTX ctx;
3.    EVP_CIPHER_CTX_init(&ctx);
4.    EVP_EncryptInit(&ctx, EVP_aes_128_cbc(), key, iv);
5. ... ...
```

보다 강력한 암호화를 위해서는 RSA 암호화 알고리즘을 OAEP 패딩 옵션과 함께 사용한다.

■ 안전한 코드의 예 - C

```
1:    #include <stdio.h>
2:    #include <stdlib.h>
3:    #include <openssl/rsa.h>
4:    #define MAX_TEXT 512
5:    void RSAEncrypt(char *text, int size)
6:    {
7:       char out[MAX_TEXT];
8:       RSA *rsa_p = RSA_new();
9:       RSA_public_encrypt(size, text, out, rsa_p, RSA_PKCS1_OAEP_PADDING);
10:       ...
11:    }
```

라. 참고문헌

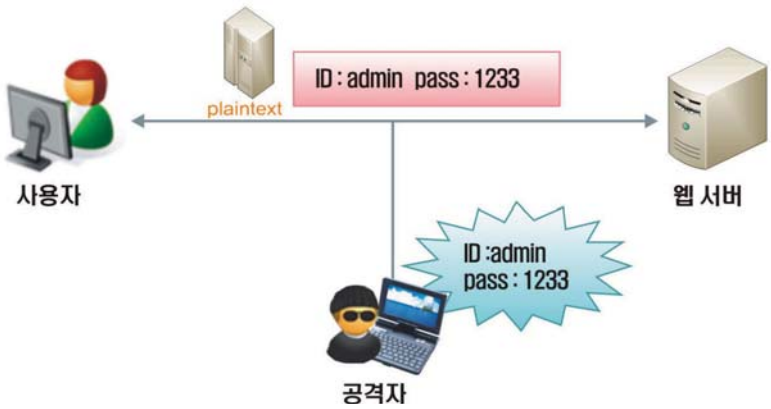
[1] CWE-327 Use of a Broken or Risky Cryptographic Algorithm, http://cwe.mitre.org/data/definitions/327.html

[2] 2010 OWASP Top 10 - A7 Insecure Cryptographic Storage, https://www.owasp.org/index.php/Top_10_2010-A7

4. 사용자 중요정보 평문 저장(또는 전송)(Missing Encryption of Sensive Data)

가. 정의

프로그램이 보안과 관련된 민감한 데이터를 평문으로 통신채널을 통해서 송수신할 경우, 통신채널 스니핑을 통해 인가되지 않은 사용자에게 민감한 데이터가 노출될 수 있는 보안 취약점이다.



<그림 2-5> 사용자 중요정보 평문 저장(또는 전송)

나. 안전한 코딩기법

- 개인정보(주민등록번호, 여권번호 등), 금융정보(카드·계좌번호), 패스워드 등을 저장할 때에는 반드시 암호화 하여 저장하고 중요한 정보를 통신 채널을 전송할 때에도 암호화한다.

다. 예제

속성 파일에서 읽어 들인 패스워드를 암호화하지 않고 네트워크를 통하여 서버에 전송하고 있다. 이 경우 패킷 스니핑을 통하여 패스워드가 노출될 수 있다.

■ 안전하지 않은 코드의 예 C

```
1:  #include "std_testcase.h"
2:
3:
4:  #include <wchar.h>
5:
6:
7:
8:  #ifdef _WIN32
9:  # include <winsock2.h>
10: # include <windows.h>
11: # include <direct.h>
12: # pragma comment(lib, "ws2_32") /* include ws2_32.lib when linking */
13: # define CLOSE_SOCKET closesocket
14: # define PATH_SZ 100
```

```
15:  #else /* NOT _WIN32 */
16:  # define INVALID_SOCKET -1
17:  # define SOCKET_ERROR -1
18:  # define CLOSE_SOCKET close
19:  # define SOCKET int
20:  # define PATH_SZ PATH_MAX
21:  #endif
22:
23:  #define TCP_PORT 27015
24:
25:
26:  #ifdef _WIN32
27:  #include <windows.h>
28:  #pragma comment(lib, "advapi32.lib")
29:  #endif
30:
31:
32:
33:  #ifndef OMITBAD
34:
35:  void KRD_205_Missing_Encryption_of_Sensitive_Data__char_connect_sock-
     et_recv_0101_bad()
36:  {
37:      char * data;
38:      char data_buf[100] = "";
39:      data = data_buf;
40:      {
41:  #ifdef _WIN32
42:          WSADATA wsa_data;
43:          int wsa_data_init = 0;
44:  #endif
45:          int recv_rv;
46:          struct sockaddr_in s_in;
47:          char *replace;
48:          SOCKET connect_socket = INVALID_SOCKET;
49:          size_t data_len = strlen(data);
50:          do
51:          {
52:  #ifdef _WIN32
53:              if (WSAStartup(MAKEWORD(2,2), &wsa_data) != NO_ERROR) break;
54:              wsa_data_init = 1;
55:  #endif
56:              connect_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
57:              if (connect_socket == INVALID_SOCKET) break;
58:              memset(&s_in, 0, sizeof(s_in));
59:              s_in.sin_family = AF_INET;
60:              s_in.sin_addr.s_addr = inet_addr("127.0.0.1");
61:              s_in.sin_port = htons(TCP_PORT);
62:              if (connect(connect_socket, (struct sockaddr*)&s_in, sizeof(s_in)) ==
     SOCKET_ERROR) break;
63:              /* Abort on error or the connection was closed, make sure to recv
     one
64:               * less char than is in the recv_buf in order to append a terminator
     */
65:              recv_rv = recv(connect_socket, (char *)data+data_len,
```

```
    (int)(100-data_len-1), 0);
66:                 if (recv_rv == SOCKET_ERROR || recv_rv == 0) break;
67:                 /* Append null terminator */
68:                 data[recv_rv] = '\0';
69:                 /* Eliminate CRLF */
70:                 replace = strchr(data, '\r');
71:                 if (replace) *replace = '\0';
72:                 replace = strchr(data, '\n');
73:                 if (replace) *replace = '\0';
74:             }
75:             while (0);
76:             if (connect_socket != INVALID_SOCKET) CLOSE_SOCKET(connect_socket);
77: #ifdef _WIN32
78:             if (wsa_data_init) WSACleanup();
79: #endif
80:         }
81:
82:         {
83: #ifdef _WIN32 /* this is WIN32 specific */
84:             HANDLE pHandle;
85:             char * username = "User";
86:             char * domain = "Domain";
87:             /* Use the password in LogonUser() to establish that it is "sensitive" */
88:             /* POTENTIAL FLAW: Using sensitive information that was possibly sent
    in plaintext over the network */
89:             if (LogonUserA(
90:                         username,
91:                         domain,
92:                         data,
93:                         LOGON32_LOGON_NETWORK,
94:                         LOGON32_PROVIDER_DEFAULT,
95:                         &pHandle) != 0)
96:             {
97:                 printLine("User logged in successfully.");
98:                 CloseHandle(pHandle);
99:             }
100:            else
101:            {
102:                printLine("Unable to login.");
103:            }
104: #endif
105:        }
106:
107: }
108:
109: #endif /* OMITBAD */
110:
111:
112: /* Below is the main(). It is only used when building this testcase on
113:    its own for testing or for building a binary to use in testing binary
114:    analysis tools. It is not used when compiling all the testcases as one
115:    application, which is how source code analysis tools are tested. */
116:
117: #ifdef INCLUDEMAIN
118:
```

```
119: int main(int argc, char * argv[])
120: {
121:     /* seed randomness */
122:     srand( (unsigned)time(NULL) );
123: #ifndef OMITBAD
124:     printLine("Calling bad()...");
125:     KRD_205_Missing_Encryption_of_Sensitive_Data__char_connect_sock-
     et_recv_0101_bad();
126:     printLine("Finished bad()");
127: #endif /* OMITBAD */
128:     return 0;
129: }
130:
131: #endif
```

다음의 코드는 패스워드를 네트워크를 통하여 서버에 전송하기 전에 암호화를 수행하고 있다.

■ 안전한 코드의 예 C

```
1:  #include "std_testcase.h"
2:
3:
4:  #include <wchar.h>
5:
6:
7:
8:  #ifdef _WIN32
9:  # include <winsock2.h>
10: # include <windows.h>
11: # include <direct.h>
12: # pragma comment(lib, "ws2_32") /* include ws2_32.lib when linking */
13: # define CLOSE_SOCKET closesocket
14: # define PATH_SZ 100
15: #else /* NOT _WIN32 */
16: # define INVALID_SOCKET -1
17: # define SOCKET_ERROR -1
18: # define CLOSE_SOCKET close
19: # define SOCKET int
20: # define PATH_SZ PATH_MAX
21: #endif
22:
23: #define TCP_PORT 27015
24:
25:
26: #ifdef _WIN32
27: #include <windows.h>
28: #pragma comment(lib, "advapi32.lib")
29: #endif
30:
31:
32:
33:
34: #ifndef OMITGOOD
```

```
35:
36:  /* goodG2B uses the GoodSource with the BadSink */
37:  static void good1()
38:  {
39:      char * data;
40:      char data_buf[100] = "";
41:      data = data_buf;
42:      /* FIX: Benign input preventing command injection */
   strcat(data, "a8f9s2@#d"); /* encrypted string */
43:
44:      {
45:  #ifdef _WIN32 /* this is WIN32 specific */
46:          HANDLE pHandle;
47:          char * username = "User";
48:          char * domain = "Domain";
49:          /* Use the password in LogonUser() to establish that it is "sensitive" */
50:          /* POTENTIAL FLAW: Using sensitive information that was possibly sent
    in plaintext over the network */
51:          if (LogonUserA(
52:                      username,
53:                      domain,
54:                      data,
55:                      LOGON32_LOGON_NETWORK,
56:                      LOGON32_PROVIDER_DEFAULT,
57:                      &pHandle) != 0)
58:          {
59:              printLine("User logged in successfully.");
60:              CloseHandle(pHandle);
61:          }
62:          else
63:          {
64:              printLine("Unable to login.");
65:          }
66:  #endif
67:      }
68:
69:  }
70:
71:  void                KRD_205_Missing_Encryption_of_Sensitive_Data__char_connect_sock-
   et_recv_0101_good()
72:  {
73:      good1();
74:  }
75:
76:  #endif /* OMITGOOD */
77:
78:  /* Below is the main(). It is only used when building this testcase on
79:     its own for testing or for building a binary to use in testing binary
80:     analysis tools. It is not used when compiling all the testcases as one
81:     application, which is how source code analysis tools are tested. */
82:
83:  #ifdef INCLUDEMAIN
84:
85:  int main(int argc, char * argv[])
86:  {
```

```
87:         /* seed randomness */
88:         srand( (unsigned)time(NULL) );
89:     #ifndef OMITGOOD
90:         printLine("Calling good()...");
91:         KRD_205_Missing_Encryption_of_Sensitive_Data__char_connect_sock-
        et_recv_0101_good();
92:         printLine("Finished good()");
93:     #endif /* OMITGOOD */
94:         return 0;
95:     }
96:
97:     #endif
```

라. 참고문헌

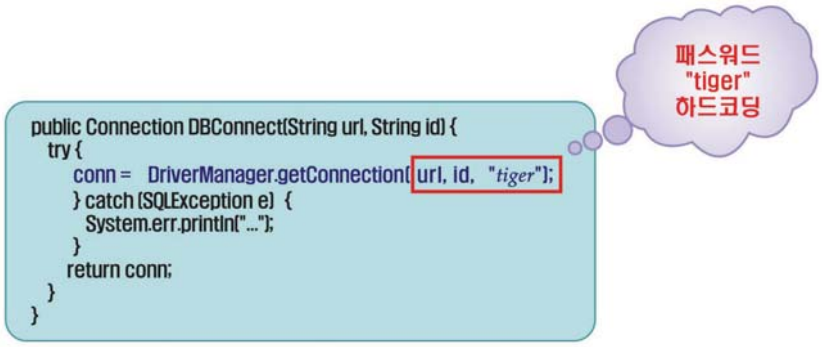
[1] CWE-311 Missing Encryption of Sensitive Data,
http://cwe.mitre.org/data/definitions/311.html

[2] 2011 SANS Top 25 - RANK 8 (CWE-311), http://cwe.mitre.org/top25/

5. 하드코드된 패스워드(Use of Hard-coded Password)

가. 정의

프로그램 코드 내부에 하드코드된 패스워드를 포함하고, 이를 이용하여 내부 인증에 사용하거나 외부 컴포넌트와 통신을 하는 경우 관리자 정보가 노출 될 수 있어 위험하다. 또한, 코드 내부에 하드코드된 패스워드가 인증실패를 야기하는 경우 시스템 관리자가 그 실패의 원인을 파악하기 쉽지 않은 단점이 있다.



<그림 2-6> 하드코드된 패스워드

나. 안전한 코딩기법

- 외부 컴포넌트는 설정파일에 암호화되어 저장된 패스워드를 사용한다.
- 서버에서 관리자를 인증할 때 설정파일이나 데이터베이스에 일방향 함수로 암호화되어 저장된 패스워드를 사용한다.

다. 예제

소스 내에 패스워드가 평문형태의 상수로 정의된 경우 공격자에게 쉽게 노출될 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #include <sqlext.h>
5:  int dbaccess(char *server, char *user)
6:  {
7:    SQLHENV henv;
8:    SQLHDBC hdbc;
9:    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
10:   SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
11:   /* 패스워드가 "asdf"로 하드코딩 되어있다.*/
12:   SQLConnect(hdbc, (SQLCHAR*) server, strlen(server), user, strlen(user),
      "asdf", 4);
13:   return 0;
14: }
```

패스워드를 외부에서 얻어오는 경우 암호화를 사용하여 관리하여야 하며 적절한 검증 과정을 거쳐야 한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #include <sqlext.h>
5:  int dbaccess(char *server, char *user, char *passwd)
6:  {
7:    SQLHENV henv;
8:    SQLHDBC hdbc;
9:    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
10:   SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
11:   SQLConnect(hdbc, (SQLCHAR*) server, strlen(server), user, strlen(user),
      passwd, strlen(passwd) );
12:   SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
13:   SQLFreeHandle(SQL_HANDLE_ENV, henv);
14:   return 0;
15: }
```

다음의 예제는 편한 관리자 페이지 로그인을 위해 코드 내부에 상수 형태로 정의된 패스워드를 사용하여 취약점을 발생시키고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  int VerifyAdmin(char * password)
2:  {
3:   if(strcmp(password, "68af404b513073584c4b6f22b6c63e6b") == 0)
4:   {
5:     printf("Entering Diagnostic Mode...");
6:     return TRUE;
7:   }
8:   else
9:   {
10:    return FALSE;
11:  }
12: }
```

보통의 경우 관리자 페이지는 해커에게 잘 보이지 않게 설계하기 때문에 개발시 로그인 패스워드 체크를 소홀히 하는 경우가 많다. 하지만 기본적인 패스워드 체크 부분은 엄격히 이루어져야 최소한의 보안성을 지킬 수 있다.

■ 안전한 코드의 예 - C

```
1:  int VerifyAdmin(char * password)
2:  {
```

```
3:    char * adminPassword = loadPassword(ADMIN);
4:    if(strcmp(password, adminPassword) == 0)
5:    {
6:      printf("Entering Diagnostic Mode...");
7:      return TRUE;
8:    }
9:    else
10:   {
11:     return FALSE;
12:   }
13: }
```

라. 참고문헌

[1] CWE-259 Use of Hard-coded Password, http://cwe.mitre.org/data/definitions/259.html

[2] CWE-798 Use of Hard-coded Credentials, http://cwe.mitre.org/data/definitions/798.html

[3] 2010 OWASP Top 10 - A7 Insecure Cryptographic Storage
https://www.owasp.org/index.php/Top_10_2010-A7

6. 충분하지 않은 키 길이 사용(Cryptographic Issue)

가. 정의

길이가 짧은 키를 사용하는 것은 암호화 알고리즘을 취약하게 만들 수 있다. 키는 암호화 및 복호화에 사용되는데 검증된 암호화 알고리즘을 사용하더라도 키 길이가 충분히 길지 않으면 짧은 시간 안에 키를 찾아낼 수 있고 이를 이용해 공격자가 암호화된 데이터나 패스워드를 복호화할 수 있게 된다.

나. 안전한 코딩기법

- RSA 알고리즘은 적어도 2048 비트 이상의 길이를 가진 키와 함께 사용해야 하고, 대칭암호화 알고리즘(Symmetric Encryption Algorithm)의 경우에는 적어도 128비트 이상의 키를 사용한다.

다. 예제

RSA_generate_key()함수에서 사용하는 키 값의 크기가 512 bit로 설정되어있다. 키 값은 최소한 2048 bit 이상으로 설정되어야 한다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <openssl/rsa.h>
4:  #include <openssl/evp.h>
5:  EVP_PKEY *RSAKey()
6:  {
7:    EVP_PKEY *pkey;
8:    RSA *rsa;
9:     /* 키값이 512bit로 너무 짧다 */
10:   rsa = RSA_generate_key(512, 35, NULL, NULL);
11:   if (rsa == NULL)
12:   {
13:     printf("Error\n");
14:     return NULL;
15:   }
16:   pkey = EVP_PKEY_new();
17:   EVP_PKEY_assign_RSA(pkey, rsa);
18:   return pkey;
19: }
```

RSA_generate_key()함수에서 사용하는 키 값은 최소한 2048 bit 이상으로 설정되어야 한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <openssl/rsa.h>
4:  #include <openssl/evp.h>
5:  EVP_PKEY *RSAKey()
6:  {
7:    EVP_PKEY *pkey;
8:    RSA *rsa;
9:    /* 키의 길이는 최소 2,048Bit 이상이여야 한다.*/
10:   rsa = RSA_generate_key(2048, 35, NULL, NULL);
11:   if (rsa == NULL)
12:   {
13:     printf("Error\n");
14:     return NULL;
15:   }
16:   EVP_PKEY *pkey = EVP_PKEY_new();
17:   EVP_PKEY_assign_RSA(pkey, rsa);
18:   return pkey;
19: }
```

가장 강력한 RSA알고리즘 사용 예는 다음의 예제처럼 OAEP패딩을 같이 사용하는 경우이다. RSA_private_decrypt함수에서 옵션 RSA_PKCS1_OAEP_PADDING 사용을 권장한다. 다음의 예제에서 키 값인 cpassword는 2048bit 이상의 길이를 가진 키 값이어야 한다.

■ 안전한 코드의 예 - C

```
1:  int dbaccess(char *user, char *cpasswd, RSA *rsa)
2:  {
3:    char *server = "DBserver";
4:    unsigned char *passwd;
5:    SQLHENV henv;
6:    SQLHDBC hdbc;
7:    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &;henv);
8:    SQLAllocHandle(SQL_HANDLE_DBC, henv, &;hdbc);
9:    /* 강력한 암호화 알고리즘 사용 */
10:   RSA_private_decrypt(strlen(cpasswd), cpasswd, passwd, rsa,
      RSA_PKCS1_OAEP_PADDING);
11:   SQLConnect(hdbc,
12:     (SQLCHAR*) server,
13:     (SQLSMALLINT) strlen(server),
14:     (SQLCHAR*) user,
15:     (SQLSMALLINT) strlen(user),
```

```
16:         (SQLCHAR*) passwd,
17:         (SQLSMALLINT) strlen(passwd));
18:     }
```

라. 참고문헌

[1] CWE-310 Cryptographic Issues, http://cwe.mitre.org/data/definitions/310.html

[2] 2010 OWASP Top 10 - A7 Insecure Cryptographic Storage, https://www.owasp.org/index.php/Top_10_2010-A7

7. 적절하지 않은 난수 값의 사용(Use of Insufficiently Random Values)

가. 정의

예측 가능한 난수를 사용하는 것은 시스템에 취약점을 유발시킨다. 예측 불가능한 숫자가 필요한 상황에서 예측 가능한 난수를 사용한다면, 공격자는 SW에서 생성되는 다음 숫자를 예상하여 시스템을 공격하는 것이 가능하다.

나. 안전한 코딩기법

- 랜덤 seed를 변경하여 사용하도록 설계한다.

다. 예제

랜덤함수 생성에서 1)seed를 사용하지 않거나 2)상수 seed를 사용하거나 3)예측 가능한 seed를 사용하는 경우는 충분한 난수 값을 얻을 수 없다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdafx.h>
2:  #include <stdio.h>
3:  #include <stdlib.h>
4:  #include <time.h>
5:  int main(void)
6:  {
7:    int count = 0;
8:    int temp;
9:    printf("\n%s\n%s\n",
10:         "Some randomly distributed integers will be printed.",
11:         "How many do you want to see? ");
12:   /* srand()에서 상수 형태의 seed를 사용하고 있다.*/
13:   srand( 100 );
14:   while ( 1 )
15:   {
16:     if ( count % 6 == 0) printf("%s", "\n");
17:     temp = rand()%101;
18:     if( temp != 100 )
19:       count++;
20:     else
21:       break;
22:     printf("%5d",  temp );
23:   }
24:  printf("\n100이 나오기 전까지 나온 숫자의 갯수 : %d \n" , count );
```

```
25:    return 0;
26:  }
```

랜덤 seed를 변경하여 사용할 수 있도록 프로그램을 설계한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdafx.h>
2:  #include <stdio.h>
3:  #include <stdlib.h>
4:  #include <time.h>
5:  int main(void)
6:  {
7:    int count = 0;
8:    int temp;
9:    int randNum = 0;
10:   printf("\n%s\n%s\n",
11:           "Some randomly distributed integers will be printed.",
12:           "How many do you want to see? ");
13:   printf("%s\n", "Init random number : ");
14:   /*임의에 랜덤값을 받아 옴*/
15:   randNum = getch();
16:   while ( 1 )
17:   {
18:     if ( count % 6 == 0)
19:       printf("%s", "\n");
20:     /*랜덤 seed를 임의의 값으로 변경*/
21:     srand(randNum);
22:     temp = rand()%101;
23:     if( temp != 100 )
24:       count++;
25:     else
26:       break;
27:     printf("%5d",  temp );
28:   }
29:   printf("\n100이나오기전까지나온숫자의갯수: %d \n" , count );
30:   return 0;
31: }
```

랜덤함수 생성에서 1)seed를 사용하지 않거나 2)상수 seed를 사용하거나 3)예측 가능한 seed를 사용하는 경우는 충분한 난수 값을 얻을 수 없다.

■ 안전하지 않은 코드의 예 - C

```
1:  int main()
2:  {
3:    const int RANDOM_RANGE = 200;
4:    srand(time(NULL));
5:    ...
6:    int randomNum = rand() % RANDOM_RANGE;
7:    ...
8:    return 0;
9:  }
```

다음의 예제에서 사용한 **SufficientRand**함수는 랜덤한 사이즈의 배열을 잡아 랜덤한 위치에서 값을 뽑아 시드값으로 사용하는 함수이다. 이는 메모리 할당 시 무작위한 값이 초기화되어 있지 않은 상태로 들어가 있는 것을 이용한 것이다.

■ 안전한 코드의 예 - C

```
1:  int main()
2:  {
3:    const int RANDOM_RANGE = 200;
4:    srand(time(NULL));
5:    ...
6:    int randomNum = SufficientRand() % RANDOM_RANGE;
7:    ...
8:    return 0;
9:  }
10: unsigned int SufficientRand()
11: {
12:   int size = rand() % 100;
13:   // malloc은 할당된 메모리를 초기화 하지 않기 때문에 어떤 값이 들어 있을지 예측할 수
      없다. 그 상태에서 배열의 무작위 위치로부터 시드값을 뽑아온다.
14:   unsigned int * unknownMem = (unsigned int *)malloc(size * sizeof(unsigned int));
15:   int index = rand() % size;
16:   srand(unknownMem[index]);
17:
18:   return rand();
19: }
```

라. 참고문헌

[1] CWE-330 Use of Insufficiently Random Values, http://cwe.mitre.org/data/definitions/330.html

[2] 2010 OWASP Top 10 - A7 Insecure Cryptographic Storage, https://www.owasp.org/index.php/Top_10_2010-A7

8. 패스워드 평문 저장(Plaintext Storage of Password)

가. 정의

패스워드를 암호화되지 않은 텍스트의 형태로 저장하는 것은 시스템 손상의 원인이 될 수 있다. 환경설정 파일에 평문으로 패스워드를 저장하면, 환경설정 파일에 접근할 수 있는 사람은 누구나 패스워드를 알아낼 수 있다. 패스워드는 높은 수준의 암호화 알고리즘을 사용하여 관리되어야 한다.

패스워드를 설정파일에 저장하는 경우 패스워드를 암호화되지 않은 상태로 저장하게 되면, 암호가 외부에 직접적으로 드러날 위험성이 있다. 따라서 패스워드는 쉽게 접근할 수 없는 저장소에 저장하거나 암호화된 상태로 저장하여야 한다.

나. 안전한 코딩기법

- 패스워드는 암호화해서 파일로 저장하도록 설계한다.

다. 예제

패스워드를 파일에서 읽어서 직접 DB 연결에 사용하고 있다. 이것은 패스워드를 암호화하고 있지 않음을 의미한다.

안전하지 않은 코드의 예 - C

```
1:  int dbaccess()
2:  {
3:    FILE *fp; char *server = "DBserver";
4:    char passwd[20];
5:    char user[20];
6:    SQLHENV henv;
7:    SQLHDBC hdbc;
8:    fp = fopen("config", "r");
9:    fgets(user, sizeof(user), fp);
10:   fgets(passwd, sizeof(passwd), fp);
11:   fclose(fp);
12:   SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
13:   SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
14:   SQLConnect(hdbc,
      (SQLCHAR*) server,
      (SQLSMALLINT) strlen(server),
      (SQLCHAR*) user,
      (SQLSMALLINT) strlen(user),
      /* 파일로부터 패스워드를 평문으로 읽어들이고 있다 */
      (SQLCHAR*) passwd,
```

```
        (SQLSMALLINT) strlen(passwd) );
15:     return 0;
16: }
```

외부에서 입력된 패스워드는 검증의 과정을 거쳐서 사용하여야 한다.

■ 안전한 코드의 예 - C

```
1:  int dbaccess()
2:  {
3:      FILE *fp;
4:      char *server = "DBserver";
5:      char passwd[20];
6:      char user[20];
7:      char *verifiedPwd;
8:      SQLHENV henv;
9:      SQLHDBC hdbc;
10:     fp = fopen("config", "r");
11:     fgets(user, sizeof(user), fp);
12:     fgets(passwd, sizeof(passwd), fp);
13:     fclose(fp);
14:     verifiedPwd = verify(passwd);
15:     SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
        SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
        SQLConnect(hdbc,
        (SQLCHAR*) server,
        (SQLSMALLINT) strlen(server),
        (SQLCHAR*) user,
        (SQLSMALLINT) strlen(user),
        /* 파일로부터 암호화된 패스워드를 읽어들이고 있다 */
        (SQLCHAR*) verifiedPwd,
         (SQLSMALLINT) strlen(verifiedPwd ) );
16:     return 0;
17: }
```

사용자명과 패스워드를 키보드로부터 읽어서 문자열을 그대로 DB 연결에 사용하고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
```

```
3:  {
4:    const char * COMMAND_PARAM = "command";
5:    const char * GET_USER_INFO_CMD = "get_user_info";
6:
7:    char * queryStr;
8:    queryStr = getenv("QUERY_STRING");
9:    if (queryStr == NULL)
10:   {
11:     // Error 처리
12:     ...
13:   }
14:   char * command = GetParameter(queryStr, COMMAND_PARAM);
15:   if (!strcmp(command, GET_USER_INFO_CMD))
16:   {
17:     const char * USER_ID_PARAM = "user_id";
18:     const char * PASSWORD_PARAM = "password";
19:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
20:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
21:
22:     ...
23:     char * dbUserId, * dbUserPassword;
24:     loadDbUserInfo(&dbUserId, &dbUserPassword);
25:     SQLRETURN retcode = SQLConnect(hdbc, (SQLCHAR*) "173.234.56.78", SQL_NTS,
    (SQLCHAR*) dbUserId, strlen(dbUserId), dbUserPassword, strlen(dbUserPassword));
26:     ...
27:     free(userId);
28:     free(password);
29:   }
30:   return EXIT_SUCCESS;
31: }
32: void loadDbUserInfo(char ** userId, char ** password)
33: {
34:   const int MAX_USER_LENGTH = 8;
35:   const int MAX_PASSWORD_LENGTH  = 16;
36:   *userId = (char *)malloc(MAX_USER_LENGTH);
37:   *password = (char *)malloc(MAX_PASSWORD_LENGTH);
38:   FILE * fp = fopen("dbUserInfo.cfg","r");
39:   fscanf(fp, "%s", *userId);
40:   fscanf(fp, "%s", *password);
41: }
```

외부에서 입력된 사용자명과 패스워드를 **loadDbUserInfo**함수에서 OAEP패딩을 사용한 RSA알고리즘으로 암호화한 후 DB연결에 사용하고 있다.

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  const Key decryptPassword="we37kor$^jkl315!#^!";  //private Key
3:
4:  int main(void)
5:  {
6:   const char * COMMAND_PARAM = "command";
7:   const char * GET_USER_INFO_CMD = "get_user_info";
8:
9:   char * queryStr;
10:  queryStr = getenv("QUERY_STRING");
11:  if (queryStr == NULL)
12:  {
13:   // Error 처리
14:   ...
15:  }
16:  char * command = GetParameter(queryStr, COMMAND_PARAM);
17:  if (!strcmp(command, GET_USER_INFO_CMD))
18:  {
19:   const char * USER_ID_PARAM = "user_id";
20:   const char * PASSWORD_PARAM = "password";
21:   const char * userId = GetParameter(queryStr, USER_ID_PARAM);
22:   const char * password = GetParameter(queryStr, PASSWORD_PARAM);
23:   ...
24:   char * dbUserId, * dbUserPassword;
25:   loadDbUserInfo(&dbUserId, &dbUserPassword);
26:   SQLRETURN retcode = SQLConnect(hdbc, (SQLCHAR*) "173.234.56.78", SQL_NTS,
    (SQLCHAR*) dbUserId, strlen(dbUserId), dbUserPassword, strlen(dbUserPassword));
27:   ...
28:   free(userId);
29:   free(password);
30:  }
31:  return EXIT_SUCCESS;
32: }
33: void loadDbUserInfo(char ** userId, char ** password)
34: {
35:  const int MAX_USER_LENGTH = 8;
36:  const int MAX_PASSWORD_LENGTH  = 16;
37:  *userId = (char *)malloc(MAX_USER_LENGTH);
38:  *password = (char *)malloc(MAX_PASSWORD_LENGTH);
39:
40:  FILE * fp = fopen("dbUserInfo.cfg","r");
41:  fscanf(fp, "%s", *userId);
42:  fscanf(fp, "%s", *password);
43:
```

```
44:    RSA_private_decrypt(strlen(decryptPassword), decryptPassword, *password, rsa,
       RSA_PKCS1_OAEP_PADDDING);
45:  }
```

라. 참고문헌

[1] CWE-256 Plaintext Storage of a Password, http://cwe.mitre.org/data/definitions/256.html

[2] J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

9. 하드코드된 암호화 키(Use of Hard-coded Cryptographic Key)

가. 정의

코드 내부에 하드코드된 암호화키를 사용하여 암호화를 수행하면 암호화된 정보가 유출될 가능성이 높아진다. 많은 SW 개발자들이 코드 내부의 고정된 패스워드의 해쉬를 계산하여 저장하는 것이 패스워드를 악의적인 공격자로부터 보호할 수 있다고 믿고 있다. 그러나 많은 해쉬 함수들이 역계산이 가능하며, 적어도 brute-force 공격에는 취약하다는 것을 고려해야만 한다.

나. 안전한 코딩기법

- 암호화되었더라도 패스워드를 상수의 형태로 프로그램 내부에 저장하여 사용하면 안된다. 대칭형 알고리즘으로 AES, ARIA, SEED, 3DES 등의 사용이 권고되며, 비대칭형 알고리즘으로 RSA 사용시 키 길이는 2048bit 이상을 권고한다. 해쉬 함수로 MD4, MD5, SHA1은 사용하지 말아야 한다.

다. 예제

암호화된 문자열을 상수로 직접 비교하도록 코드가 작성되는 경우이다. 이러한 경우 암호가 보다 쉽게 노출될 가능성이 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  extern char *salt;
2:  typedef int SQLSMALLINT;
3:  int dbaccess(char *user, char *passwd)
4:  {
5:    char *server = "DBserver";
6:    char *cpasswd;
7:    SQLHENV henv;
8:    SQLHDBC hdbc;
9:    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
10:   SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
11:   cpasswd = crypt(passwd, salt);
12:
13:   /* 암호화 키가 소스코드내에 상수로 하드코드 되어 있다 */
14:   if (strcmp(cpasswd, "68af404b513073582b6c63e6b") != 0)
15:   {
16:     printf("Incorrect password\n");
17:     return -1;
18:   }
19:   ……
20: }
```

암호화되어 저장된 암호를 적절한 검증 과정을 거쳐서 얻어오는 별도의 모듈을 작성하고, 이를 암호화된 스트링과 비교하도록 코드가 작성되어야 한다.

■ 안전한 코드의 예 - C

```
1:  extern char *salt;
2:  typedef int SQLSMALLINT;
3:  char* getPassworld()
4:  {
5:     static char* pass="password";
6:     return pass;
7:  }
8:  int dbaccess(char *user, char *passwd)
9:  {
10:    char *server = "DBserver";
11:    char *cpasswd;
12:    char* storedpasswd;
13:    SQLHENV henv;
14:    SQLHDBC hdbc;
15:    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
16:    SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
17:    cpasswd = crypt(passwd, salt);
18:    storedpasswd = getPassword();
19:    if (strcmp(cpasswd, storedpasswd) != 0)
20:    {
21:       printf("Incorrect password\n");
22:       SQLFreeHandle(SQL_HANDLE_DBC, &hdbc);
23:       SQLFreeHandle(SQL_HANDLE_ENV, &henv);
24:       return -1;
25:    }
26:    ……
27: }
```

암호화된 문자열을 상수로 직접 비교하도록 코드가 작성되는 경우이다. 이러한 경우 암호가 보다 쉽게 노출될 가능성이 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:   const char * COMMAND_PARAM = "command";
5:   const char * VISIT_ADMIN_PAGE_CMD = "visit_admin_page";
6:
7:   char * queryStr;
8:   queryStr = getenv("QUERY_STRING");
9:   if (queryStr == NULL)
10:  {
11:   // Error 처리
```

```
12:     ...
13:   }
14:   char * command = GetParameter(queryStr, COMMAND_PARAM);
15:   if (!strcmp(command, VISIT_ADMIN_PAGE_CMD))
16:   {
17:     const char * PASSWORD_PARAM = "password";
18:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
19:
20:     char * encryptedPassword  = (char *)malloc(rsa);
21:     RSA_private_encrypt(strlen(password), password, encryptedPassword, rsa,
      RSA_PKCS1_OAEP_PADDDING);
22:
23:     if(strcmp(encryptedPassword, "dafsklj2w35ajgol") == 0)
24:     {
25:       ...
26:     }
27:     ...
28:     free(userId);
29:     free(password);
30:   }
31:   return EXIT_SUCCESS;
32: }
```

다음의 예제는 패스워드 파일로부터 관리자 패스워드를 읽은 후 암호화하여 가져와 그것을 외부에서 입력된 암호화된 스트링과 비교, 검증하도록 작성되었다.

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:    const char * COMMAND_PARAM = "command";
5:    const char * VISIT_ADMIN_PAGE_CMD = "visit_admin_page";
6:
7:    char * queryStr;
8:    queryStr = getenv("QUERY_STRING");
9:    if (queryStr == NULL)
10:   {
11:     // Error 처리
12:     ...
13:   }
14:   char * command = GetParameter(queryStr, COMMAND_PARAM);
15:   if (!strcmp(command, VISIT_ADMIN_PAGE_CMD))
16:   {
17:     const char * PASSWORD_PARAM = "password";
18:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
```

```
19:
20:     char * encryptedPassword   = (char *)malloc(rsa);
21:     RSA_private_encrypt(strlen(password), password, encryptedPassword, rsa,
        RSA_PKCS1_OAEP_PADDDING);
22:     char * ecnryptedAdminPagePassword = loadEncryptedAdminPagePassword();
23:
24:     if(strcmp(encryptedPassword, ecnryptedAdminPagePassword) == 0)
25:     {
26:       ...
27:     }
28:     ...
29:     free(userId);
30:     free(password);
31:   }
32:   return EXIT_SUCCESS;
33: }
34: char * loadEncryptedAdminPagePassword(void)
35: {
36:   const int MAX_PASSWORD_LENGTH   = 16;
37:   char * password = (char *)malloc(MAX_PASSWORD_LENGTH);
38:   FILE * fp = fopen("adminPasssword.cfg","r");
39:   fscanf(fp, "%s", password);
40:
41:   return password;
42: }
```

라. 참고문헌

[1] CWE-321 Use of Hard-coded Cryptographic Key, http://cwe.mitre.org/data/definitions/321.html

10. 주석문안에 포함된 패스워드 등 시스템 주요정보(Information Exposure Through Comments)

가. 정의

패스워드를 주석문에 넣어두면 시스템 보안이 훼손될 수 있다. SW 개발자가 편의를 위해서 주석문에 패스워드를 적어둔 경우, SW가 완성된 후에는 그것을 제거하는 것이 매우 어렵게 된다. 또한 공격자가 소스코드에 접근하거나 역공학(Reverse Engineering)을 통해 시스템의 패스워드를 쉽게 확인할 수 있다.

나. 안전한 코딩기법

- 개발시에 주석 부분에 남겨놓은 패스워드 및 보안에 관련된 내용은 개발이 끝난 후에는 반드시 제거해야 한다.

다. 예제

소스코드 주석에 패스워드와 관련된 정보가 포함되어 있는 경우이다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  /* default password is "abracadabra".  주석문안에 패스워드가 포함되어있다 */
4:  int verifyAuth(char *ipasswd, char *orgpasswd)
5:  {
6:     char* admin="admin";
7:     if (strncmp(ipasswd, orgpasswd, sizeof(ipasswd)) != 0)
8:     {
9:        printf("Authetication Fail!\n");
10:    }
11:    return admin;
12: }
```

관리자 아이디, 패스워드 등의 중요 정보는 개발이 끝난 후 배포 전에 모두 제거하여야 한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  /* 주석에 패스워드를 기입하지 않는다 */
4:   int verifyAuth(char *ipasswd, char *orgpasswd)
5:  {
6:     char* admin="admin";
7:     if (strncmp(ipasswd, orgpasswd, sizeof(ipasswd)) != 0)
8:     {
9:        printf("Authetication Fail!\n");
```

```
10:     }
11:     return admin;
12:   }
```

소스코드 주석에 개발시에 작성된 에러 디버깅 정보가 포함되어 있는 경우이다.

■ 안전하지 않은 코드의 예 - C

```
1:   // 예측값을 계산 하는 요청을 했다. 이를 위하여 MakeEstimates 함수를 호출했으나
     Warning message에 프로그램 결함정보가 있었다.
2:   const char * GetParameter(const char * queryString, const char * key);
3:   int main(void)
4:   {
5:    const char * COMMAND_PARAM = "command";
6:    const char * CACULATE_ESTIMATES = "calculate_estimates";
7:
8:    char * queryStr;
9:    queryStr = getenv("QUERY_STRING");
10:   if (queryStr == NULL)
11:   {
12:    // Error 처리
13:    ...
14:   }
15:   char * command = GetParameter(queryStr, COMMAND_PARAM);
16:   if (!strcmp(command, CACULATE_ESTIMATES))
17:   {
18:    const char * ITEMS_PARAM = "items";
19:    const char * itemStrs = GetParameter(queryStr, ITEMS_PARAM);
20:    ....
21:    estimates = MakeEstimates(items, itemCount);
22:    ...
23:   }
24:   return EXIT_SUCCESS;
25:  }
26:  Estimates MakeEstimates(char * items[], int itemCount)
27:  {
28:   if(itemCount > 30)
29:   {
30:    printf("Warning : The count of item is over 30, then the database will be unstable");
31:   }
32:   ...
33:  }
```

개발이나 디버깅시에 작성된 중요 정보가 포함된 주석문은 배포 전에 모두 제거하여야 한다.

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:   const char * COMMAND_PARAM = "command";
5:   const char * CACULATE_ESTIMATES = "calculate_estimates";
6:
7:   char * queryStr;
8:   queryStr = getenv("QUERY_STRING");
9:   if (queryStr == NULL)
10:  {
11:   // Error 처리
12:   ...
13:  }
14:  char * command = GetParameter(queryStr, COMMAND_PARAM);
15:  if (!strcmp(command, CACULATE_ESTIMATES))
16:  {
17:   const char * ITEMS_PARAM = "items";
18:   const char * itemStrs = GetParameter(queryStr, ITEMS_PARAM);
19:   ....
20:   estimates = MakeEstimates(items, itemCount);
21:   ...
22:  }
23:  return EXIT_SUCCESS;
24: }
25: Estimates MakeEstimates(char * items[], int itemCount)
26: {
27:  if(itemCount > 30)
28:  {
29:   printf("Warning : The count of item is over 30, then the database will be unstable");
30:  }
31:  ...
32: }
```

라. 참고문헌

[1] CWE-615 Information Exposure Through Comments,
http://cwe.mitre.org/data/definitions/615.html

[2] 2010 OWASP Top 10 - A7 Insecure Cryptographic Storage
https://www.owasp.org/index.php/Top_10_2010-A7

[3] Web Application Security Consortium 24 + 2 - (WASC 24 + 2) Information Leakage

11. 솔트 없이 일방향 해쉬 함수 사용(Use of a One-Way Hash without a Salt)

가. 정의

패스워드를 저장 시 일방향 해쉬 함수의 성질을 이용하여 패스워드의 해쉬값을 저장한다. 만약 패스워드를 솔트 없이 해쉬하여 저장한다면, 공격자는 레인보우 테이블과 같이 가능한 모든 패스워드에 대해 해쉬값을 미리 계산하고, 이를 전수조사에 이용하여 패스워드를 찾을 수 있게 된다.

나. 안전한 코딩기법

- 패스워드를 저장 시 패스워드와 솔트를 해쉬 함수의 입력으로 하여 얻은 해쉬값을 저장한다.

다. 예제

다음 예제는 패스워드 저장 시 솔트 없이 패스워드에 대한 해쉬값을 얻는 과정을 보여준다.

안전하지 않은 코드의 예 - C

```
1:  #include "std_testcase.h"
2:
3:
4:  #ifdef _WIN32
5:  # include <winsock2.h>
6:  # include <windows.h>
7:  # include <direct.h>
8:  # pragma comment(lib, "ws2_32") /* include ws2_32.lib when linking */
9:  # define CLOSE_SOCKET closesocket
10: # define PATH_SZ 100
11: #else /* NOT _WIN32 */
12: # define INVALID_SOCKET -1
13: # define SOCKET_ERROR -1
14: # define CLOSE_SOCKET close
15: # define SOCKET int
16: # define PATH_SZ PATH_MAX
17: #endif
18:
19: #define TCP_PORT 27015
20:
21: #include <krdcryptutil.h>
22:
23: #pragma comment(lib, "advapi32.lib")
24: #pragma comment(lib, "user32.lib")
25:
26:
27: #ifndef OMITBAD
28:
```

```
29: void KRD_215_Unsalted_One_Way_Hash__char_connect_socket_2_KrdMD5Encrypt_0101_bad()
30: {
31:     char * data;
32:     char data_buf[100] = "";
33:     data = data_buf;
34:     {
35: #ifdef _WIN32
36:         WSADATA wsa_data;
37:         int wsa_data_init = 0;
38: #endif
39:         int recv_rv;
40:         struct sockaddr_in s_in;
41:         char *replace;
42:         SOCKET connect_socket = INVALID_SOCKET;
43:         size_t data_len = strlen(data);
44:         do
45:         {
46: #ifdef _WIN32
47:             if (WSAStartup(MAKEWORD(2,2), &wsa_data) != NO_ERROR) break;
48:             wsa_data_init = 1;
49: #endif
50:             connect_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
51:             if (connect_socket == INVALID_SOCKET) break;
52:             memset(&s_in, 0, sizeof(s_in));
53:             s_in.sin_family = AF_INET;
54:             s_in.sin_addr.s_addr = inet_addr("127.0.0.1");
55:             s_in.sin_port = htons(TCP_PORT);
56:             if (connect(connect_socket, (struct sockaddr*)&s_in, sizeof(s_in)) == SOCKET_ERROR) break;
57:             /* Abort on error or the connection was closed, make sure to recv one
58:              * less char than is in the recv_buf in order to append a terminator */
59:             recv_rv = recv(connect_socket, (char *)data+data_len, (int)(100-data_len-1), 0);
60:             if (recv_rv == SOCKET_ERROR || recv_rv == 0) break;
61:             /* Append null terminator */
62:             data[recv_rv] = '\0';
63:             /* Eliminate CRLF */
64:             replace = strchr(data, '\r');
65:             if (replace) *replace = '\0';
66:             replace = strchr(data, '\n');
67:             if (replace) *replace = '\0';
68:         }
69:         while (0);
70:         if (connect_socket != INVALID_SOCKET) CLOSE_SOCKET(connect_socket);
71: #ifdef _WIN32
72:         if (wsa_data_init) WSACleanup();
```

```
73:  #endif
74:      }
75:
76:      {
77:          char * p_enc = NULL;
78:          /* FLAW */
79:          p_enc = KrdMD5Encrypt(data, NULL);
80:          printLine(p_enc);
81:      }
82:
83:  }
84:
85:  #endif /* OMITBAD */
86:
87:
88:  /* Below is the main(). It is only used when building this testcase on
89:     its own for testing or for building a binary to use in testing binary
90:     analysis tools. It is not used when compiling all the testcases as one
91:     application, which is how source code analysis tools are tested. */
92:
93:  #ifdef INCLUDEMAIN
94:
95:  int main(int argc, char * argv[])
96:  {
97:      /* seed randomness */
98:      srand( (unsigned)time(NULL) );
99:  #ifndef OMITBAD
100:     printLine("Calling bad()...");
101:     KRD_215_Unsalted_One_Way_Hash__char_connect_sock-
    et_2_KrdMD5Encrypt_0101_bad();
102:     printLine("Finished bad()");
103: #endif /* OMITBAD */
104:     return 0;
105: }
106:
107: #endif
```

패스워드만을 해쉬함수의 입력으로 사용하기에 레인보우 테이블을 이용하여 사전에 공격이 가능하며, 이를 방지하기 위해 솔트를 해쉬함수의 입력으로 해쉬값을 계산하여 사용한다.

■ 안전한 코드의 예 - C

```
1:  #include "std_testcase.h"
2:  #ifdef _WIN32
3:  # include <winsock2.h>
4:  # include <windows.h>
5:  # include <direct.h>
```

```
6:  # pragma comment(lib, "ws2_32") /* include ws2_32.lib when linking */
7:  # define CLOSE_SOCKET closesocket
8:  # define PATH_SZ 100
9:  #else /* NOT _WIN32 */
10: # define INVALID_SOCKET -1
11: # define SOCKET_ERROR -1
12: # define CLOSE_SOCKET close
13: # define SOCKET int
14: # define PATH_SZ PATH_MAX
15: #endif
16:
17: #define TCP_PORT 27015
18:
19:
20:
21:
22: #include <krdcryptutil.h>
23:
24: #pragma comment(lib, "advapi32.lib")
25: #pragma comment(lib, "user32.lib")
26:
27:
28:
29: #ifndef OMITGOOD
30:
31: /* goodG2B uses the GoodSource with the BadSink */
32:
33: /* goodB2G uses the BadSource with the GoodSink */
34: static void goodB2G()
35: {
36:     char * data;
37:     char data_buf[100] = "";
38:     data = data_buf;
39:     {
40: #ifdef _WIN32
41:         WSADATA wsa_data;
42:         int wsa_data_init = 0;
43: #endif
44:         int recv_rv;
45:         struct sockaddr_in s_in;
46:         char *replace;
47:         SOCKET connect_socket = INVALID_SOCKET;
48:         size_t data_len = strlen(data);
49:         do
50:         {
```

```
51:  #ifdef _WIN32
52:                  if (WSAStartup(MAKEWORD(2,2), &wsa_data) != NO_ERROR) break;
53:                  wsa_data_init = 1;
54:  #endif
55:                  connect_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
56:                  if (connect_socket == INVALID_SOCKET) break;
57:                  memset(&s_in, 0, sizeof(s_in));
58:                  s_in.sin_family = AF_INET;
59:                  s_in.sin_addr.s_addr = inet_addr("127.0.0.1");
60:                  s_in.sin_port = htons(TCP_PORT);
61:                  if (connect(connect_socket, (struct sockaddr*)&s_in, sizeof(s_in)) ==
     SOCKET_ERROR) break;
62:                  /* Abort on error or the connection was closed, make sure to recv one
63:                   * less char than is in the recv_buf in order to append a terminator */
64:                  recv_rv = recv(connect_socket, (char *)data+data_len, (int)(100-data_len-1), 0);
65:                  if (recv_rv == SOCKET_ERROR || recv_rv == 0) break;
66:                  /* Append null terminator */
67:                  data[recv_rv] = '\0';
68:                  /* Eliminate CRLF */
69:                  replace = strchr(data, '\r');
70:                  if (replace) *replace = '\0';
71:                  replace = strchr(data, '\n');
72:                  if (replace) *replace = '\0';
73:              }
74:              while (0);
75:              if (connect_socket != INVALID_SOCKET) CLOSE_SOCKET(connect_socket);
76:  #ifdef _WIN32
77:              if (wsa_data_init) WSACleanup();
78:  #endif
79:          }
80:
81:          {
82:              char * p_enc = NULL;
83:              p_enc = KrdMD5Encrypt(data, "$1$/iSaq7rB$EoUw5jJPPvAPECNaaWzMK/");
84:              printLine(p_enc);
85:          }
86:
87:  }
88:
89:  void                                    KRD_215_Unsalted_One_Way_Hash__char_connect_sock-
     et_2_KrdMD5Encrypt_0101_good()
90:  {
91:
92:      goodB2G();
93:  }
```

```
94:
95:  #endif /* OMITGOOD */
96:
97:  /* Below is the main(). It is only used when building this testcase on
98:     its own for testing or for building a binary to use in testing binary
99:     analysis tools. It is not used when compiling all the testcases as one
100:    application, which is how source code analysis tools are tested. */
101:
102: #ifdef INCLUDEMAIN
103:
104: int main(int argc, char * argv[])
105: {
106:     /* seed randomness */
107:     srand( (unsigned)time(NULL) );
108: #ifndef OMITGOOD
109:     printLine("Calling good()...");
110:     KRD_215_Unsalted_One_Way_Hash__char_connect_socket_2_KrdMD5Encrypt_0101_good();
111:     printLine("Finished good()");
112: #endif /* OMITGOOD */
113:     return 0;
114: }
115:
116: #endif
```

라. 참고문헌

[1] CWE-759 Use of a One-Way Hash without a Salt, MITRE, http://cwe.mitre.org/data/definitions/759.html

[2] 2011 SANS Top 25 - RANK 25 (CWE-759), http://cwe.mitre.org/top25/

12. 하드코드된 사용자 계정(Hard-coded Username)

가. 정의

SW가 코드 내부에 고정된 사용자 계정 이름을 포함하고, 이를 이용하여 내부 인증에 사용하거나 외부 컴포넌트와 통신을 하는 것은 위험하다. 코드 내부에 하드코드된 사용자 계정이 인증 실패를 야기하게 되면, 시스템 관리자가 그 실패의 원인을 찾아내기가 매우 어렵다. 원인이 파악이 되더라도 하드코드된 패스워드를 수정해야 하기 때문에 시스템 관리자는 SW 시스템 전체를 중지시켜 해결해야 하는 경우가 발생할 수 있다.

나. 안전한 코딩기법

- 사용자 계정이나 패스워드를 코드 내부에 하드코드하여 사용하지 않고 검증과정을 거쳐서 얻은 값을 사용해야 한다.

다. 예제

사용자 이름을 코드상에 상수로 하드코드하여 사용하는 경우이다. 이를 통해 인증 없이 DB 접근이 가능하다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #include <sqlext.h>
5:  int dbaccess(char *server, char *passwd)
6:  {
7:    SQLHENV henv;
8:    SQLHDBC hdbc;
9:    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
10:   SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
11:   SQLConnect(hdbc,
12:                (SQLCHAR*) server,
13:                (SQLSMALLINT) strlen(server),
14:                 /*사용자 이름을 코드에 상수로 사용하는 경우*/
15:                 "root",
16:                  4,
17:                passwd,
18:                strlen(passwd));
19:                return 0;
20: }
```

사용자 이름과 패스워드는 검증과정을 거쳐서 얻어지도록 프로그램을 설계하여야 한다.

안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
```

```
3:  #include <string.h>
4:  #include <sqlext.h>
5:  int dbaccess(char *server, char *user, char *passwd)
6:  {
7:    SQLHENV henv;
8:    SQLHDBC hdbc;
9:    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
10:   SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
11:   SQLConnect(hdbc,
12:              (SQLCHAR*) server,
13:              strlen(server),
14:              user,
15:              strlen(user),
16:              passwd,
17:              strlen(passwd));
18:     return 0;
19:  }
```

DB관리자 계정과 패스워드를 코드 상에 상수로 하드코드하여 사용하는 경우이다. 이를 통해 어떤 유저라 해도 프로그램만 실행할 수 있다면 어떠한 인증도 없이 DB 접근이 가능하다.

■ 안전하지 않은 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:    const char * COMMAND_PARAM = "command";
5:    const char * GET_USER_INFO_CMD = "get_user_info";
6:
7:    char * queryStr;
8:    queryStr = getenv("QUERY_STRING");
9:    if (queryStr == NULL)
10:   {
11:     // Error 처리
12:     ...
13:   }
14:   char * command = GetParameter(queryStr, COMMAND_PARAM);
15:   if (!strcmp(command, GET_USER_INFO_CMD))
16:   {
17:     const char * USER_ID_PARAM = "user_id";
18:     const char * PASSWORD_PARAM = "password";
19:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
20:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
21:     ...
```

```
22:    SQLRETURN retcode = SQLConnect(hdbc, (SQLCHAR*) "173.234.56.78", SQL_NTS,
       (SQLCHAR*) "DBAdmin", sizeof("DBAdmin"), "qwkejgjek", 9);
23:    ...
24:    free(userId);
25:    free(password);
26:   }
27:   return EXIT_SUCCESS;
28:  }
```

사용자 이름과 패스워드를 **loadDbUserInfo(&dbUserId, &dbUserPassword);**를 통해 검증을 거친 후 프로그램에서 사용하게 프로그래밍 되어 있다.

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:   const char * COMMAND_PARAM = "command";
5:   const char * GET_USER_INFO_CMD = "get_user_info";
6:
7:   char * queryStr;
8:   queryStr = getenv("QUERY_STRING");
9:   if (queryStr == NULL)
10:  {
11:   // Error 처리
12:   ...
13:  }
14:  char * command = GetParameter(queryStr, COMMAND_PARAM);
15:  if (!strcmp(command, GET_USER_INFO_CMD))
16:  {
17:   const char * USER_ID_PARAM = "user_id";
18:   const char * PASSWORD_PARAM = "password";
19:   const char * userId = GetParameter(queryStr, USER_ID_PARAM);
20:   const char * password = GetParameter(queryStr, PASSWORD_PARAM);
21:   ...
22:   char * dbUserId, * dbUserPassword;
23:   loadDbUserInfo(&dbUserId, &dbUserPassword);
24:   SQLRETURN retcode = SQLConnect(hdbc, (SQLCHAR*) "173.234.56.78", SQL_NTS,
      (SQLCHAR*) dbUserId, strlen(dbUserId), dbUserPassword, strlen(dbUserPassword));
25:   ...
26:   free(userId);
27:   free(password);
28:  }
29:  return EXIT_SUCCESS;
30: }
```

라. 참고문헌

[1] CWE-255 Credentials Management, http://cwe.mitre.org/data/definitions/255.html

[2] 2010 OWASP Top 10 - A7 Insecure Cryptographic Storage, https://www.owasp.org/index.php/Top_10_2010-A7

[3] Security Technical Implementation Guide Version 3 - (STIG 3) APP3210.1 CAT II

13. 잘못된 권한 부여(Incorrect Privilege Assignment)

가. 정의

SW의 일부분에서 의도하지 않게 잘못된 권한을 부여해 SW의 특정부분을 수행할 때 상위 권한을 가지게 되는 취약점이다.

나. 안전한 코딩기법

- 응용프로그램을 실행할 때는 사용자에 권한만을 통해 수행되도록 설계한다.

다. 예제

권한을 높여서 사용 가능하지 않은 파일에 접근하는 것은 정보를 유출시킬 가능성이 있다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <sys/types.h>
3:  #include <unistd.h>
4:  char buf[100];
5:  char *privilegeUp()
6:  {
7:      FILE *fp;
8:      /* 사용자 UID 생성 */
9:      seteuid(0);
10:     fp = fopen("/etc/passwd", "r");
11:     fgets(buf, sizeof(buf), fp);
12:     /* 현재 프로세스의 유효 사용자 UID를 획득 */
13:     seteuid(getuid());
14:     fclose(fp);
15:     return buf;
16: }
17:
18: int main()
19: {
20:    printf("비밀번호의 내용을 찍어보아요.\n");
21:    char *buffer=privilegeUp();
22:    printf("비밀번호의 내용은 %s입니다.\n",buffer);
23:    return 0;
24: }
```

사용자 권한을 높이지 않고 접근가능한 자원만을 사용하도록 프로그래밍 하여야 한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <sys/types.h>
3:  #include <unistd.h>
4:  char buf[100];
5:  char *privilegeUp()
6:  {
7:    FILE *fp;
8:    /* 잘못된 권한 부여는 하지 않는다. */
9:    fp = fopen("/etc/passwd", "r");
10:   fgets(buf, sizeof(buf), fp);
11:   fclose(fp);
12:   return buf;
13: }
14: int main()
15: {
16:   printf("비밀번호의 내용을 찍어보아요.\n");
17:   char *buffer;
18:   buffer=privilegeUp();
19:   printf("비밀번호의 내용은 %s입니다.\n",buffer);
20:   return 0;
21: }
```

사용자에 대한 정확한 권한을 체크하지 않고 작업을 진행시키고 있다. 권한에 대한 체크 없이 정상적인 유저인지에 대한 사항만 체크하고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:   const char * COMMAND_PARAM = "command";
5:   const char * READ_CONFIGURATION_CMD = "read_document";
6:
7:   char * queryStr;
8:   queryStr = getenv("QUERY_STRING");
9:   if (queryStr == NULL)
10:  {
11:   // Error 처리
12:   ...
13:  }
14:  char * command = GetParameter(queryStr, COMMAND_PARAM);
15:  if (!strcmp(command, READ_CONFIGURATION_CMD))
16:  {
```

```
17:     const char * USER_ID_PARAM = "user_id";
18:     const char * PASSWORD_PARAM = "password";
19:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
20:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
21:     ...
22:     if(isCorrectUser(userId, password))
23:     {
24:       seteuid(0);
25:       ...
26:     }
27:     ...
28:     free(userId);
29:     free(password);
30:   }
31:   ...
32:   return EXIT_SUCCESS;
33: }
```

각 유저의 권한을 미리 정의해두고 그에 맞춘 권한에 따른 작업만을 허용하고 있다.

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:    const char * COMMAND_PARAM = "command";
5:    const char * READ_CONFIGURATION_CMD = "read_document";
6:
7:    map<string, uid_t> userGroupMinimumEffectUid;
8:    userGroupMinimumEffectUid["Normal"] = 1;
9:    userGroupMinimumEffectUid["Admin"] = 0;
10:   userGroupMinimumEffectUid["Free"] = 2;
11:
12:   char * queryStr;
13:   queryStr = getenv("QUERY_STRING");
14:   if (queryStr == NULL)
15:   {
16:     // Error 처리
17:     ...
18:   }
19:   char * command = GetParameter(queryStr, COMMAND_PARAM);
20:   if (!strcmp(command, READ_CONFIGURATION_CMD))
21:   {
22:     const char * USER_ID_PARAM = "user_id";
23:     const char * PASSWORD_PARAM = "password";
24:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
```

```
25:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
26:     ...
27:     if(isCorrectUser(userId, password))
28:     {
29:       String userGroup = GetUserGroup(userId);
30:       seteuid(userGroupMinimumEffectUid[usrGroup]);
31:       ...
32:     }
33:     ...
34:     free(userId);
35:     free(password);
36:   }
37:   ...
38:   return EXIT_SUCCESS;
39: }
```

라. 참고문헌

[1] CWE-266 Incorrect Privilege Assignment, http://cwe.mitre.org/data/definitions/266.html

14. 최소 권한 적용 위배(Least Privilege Violation)

가. 정의

SW의 일부분에서 필요에 따라서는 상위 권한을 가지고 수행되어야 할 부분이 있을 수 있으며, 작업이 끝난 후에는 본래의 권한으로 되돌아 가야한다. 그렇지 않을 경우 의도하지 않게 상위 권한을 가지고 수행하게 되는 부분이 나타날 수 있는 취약점이다.

나. 안전한 코딩기법

- 권한을 높여 수행이 완료된 직후에는 원래 권한으로 바로 복귀시켜야 한다.

다. 예제

chroot()와 같은 함수는 root 권한이 필요한 함수인데 사용이 끝난 이후에 **setuid()**와 같은 함수로 원래 사용자 권한으로 복귀하는 것이 필요하다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <fcntl.h>
4:  #include <sys/types.h>
5:  #include <unistd.h>
6:  #define AAP_HOME "/var/tmp"
7:    char buf[100];
8:    char *privilegeDown()
9:  {
10:       FILE *fp;
11:       /* root 권한을 주고 작업이 끝나도 원래대로 복귀시키지 않았다 */
12:       chroot(APP_HOME);
13:       chdir("/");
14:       fopen("important_file", "r");
15:       fgets(buf, sizeof(buf), fp);
16:       fclose(fp);
17:    return buf;
18:  }
```

chroot() 함수 사용 직후에 **seteuid()**를 호출해서 원래 사용자 권한으로 복귀하는 것이 필요하다.

안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <fcntl.h>
4:  #include <sys/types.h>
5:  #include <unistd.h>
6:  #define AAP_HOME "/var/tmp"
```

```
7:     char buf[100];
8:     char *privilegeDown()
9:   {
10:     FILE *fp;
11:     chroot(APP_HOME);
12:     chdir("/");
13:     fp=fopen("important_file", "r");
14:     fgets(buf, sizeof(buf), fp);
15:     /* 작업 후 원래 사용자 권한으로 복귀 */
16:     seteuid(1);
17:     fclose(fp);
18:   return buf;
19:   }
```

seteuid(userGroupMinimumEffectUid[usrGroup]);구문을 통해 권한을 획득하고 작업을 마친 이후에 원래 사용자 권한으로 복귀시키지 않았다. 잠재적인 권한 오용 가능성이 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:   const char * COMMAND_PARAM = "command";
5:   const char * READ_CONFIGURATION_CMD = "read_document";
6:
7:   map<string, uid_t> userGroupMinimumEffectUid;
8:   userGroupMinimumEffectUid["Normal"] = 1;
9:   userGroupMinimumEffectUid["Admin"] = 0;
10:  userGroupMinimumEffectUid["Free"] = 2;
11:
12:  char * queryStr;
13:  queryStr = getenv("QUERY_STRING");
14:  if (queryStr == NULL)
15:  {
16:   // Error 처리
17:   ...
18:  }
19:  char * command = GetParameter(queryStr, COMMAND_PARAM);
20:  if (!strcmp(command, READ_CONFIGURATION_CMD))
21:  {
22:   const char * USER_ID_PARAM = "user_id";
23:   const char * PASSWORD_PARAM = "password";
24:   const char * userId = GetParameter(queryStr, USER_ID_PARAM);
25:   const char * password = GetParameter(queryStr, PASSWORD_PARAM);
26:   ...
```

```
27:     if(isCorrectUser(userId, password))
28:     {
29:       seteuid(userGroupMinimumEffectUid[usrGroup]);
30:       ...
31:     }
32:     ...
33:     free(userId);
34:     free(password);
35:   }
36:   ...
37:   return EXIT_SUCCESS;
38: }
```

seteuid(uidBackup);를 호출해서 미리 백업해 두었던 원래의 사용자 권한으로 복귀시켰다.

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:    const char * COMMAND_PARAM = "command";
5:    const char * READ_CONFIGURATION_CMD = "read_document";
6:
7:    map<string, uid_t> userGroupMinimumEffectUid;
8:    userGroupMinimumEffectUid["Normal"] = 1;
9:    userGroupMinimumEffectUid["Admin"] = 0;
10:   userGroupMinimumEffectUid["Free"] = 2;
11:
12:   char * queryStr;
13:   queryStr = getenv("QUERY_STRING");
14:   if (queryStr == NULL)
15:   {
16:     // Error 처리
17:     ...
18:   }
19:   char * command = GetParameter(queryStr, COMMAND_PARAM);
20:   if (!strcmp(command, READ_CONFIGURATION_CMD))
21:   {
22:     const char * USER_ID_PARAM = "user_id";
23:     const char * PASSWORD_PARAM = "password";
24:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
25:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
26:     ...
27:     if(isCorrectUser(userId, password))
28:     {
```

```
29:       uid_t uidBackup = geteuid();
30:       seteuid(userGroupMinimumEffectUid[usrGroup]);
31:       ...
32:       seteuid(uidBackup);
33:     }
34:     ...
35:     free(userId);
36:     free(password);
37:   }
38:   ...
39:   return EXIT_SUCCESS;
40: }
```

라. 참고문헌

[1] CWE-272 Least Privilege Violation, http://cwe.mitre.org/data/definitions/272.html

15. 취약한 암호화: 적절하지 못한 RSA 패딩(Weak Encryption: Inadequate RSA Padding)

가. 정의

OAEP 패딩을 사용하지 않고 RSA 알고리즘을 이용하는 것은 위험하다. RSA 알고리즘은 실제 사용시 패딩 기법과 함께 사용하는 것이 일반적이다. 패딩 기법을 사용함으로써 패딩이 없는 RSA 알고리즘의 취약점을 이용하는 공격을 막을 수 있게 된다.

나. 안전한 코딩기법

- RSA_pubic_encrypt()함수에서 RSA_NO_PADDING 파라미터를 사용하지 않는다.

다. 예제

RSA 암호화 함수에서 **NO_PADDING**과 같이 적절하지 않은 파라미터를 사용하는 경우이다. 보다 안전한 암호화를 위해서는 **RSA_pubic_encrypt()**함수에서 **RSA_NO_PADDING** 파라미터를 사용하지 않거나 다른 파라미터를 사용하여야 한다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <openssl/rsa.h>
4:  #define MAX_TEXT 512
5:  void RSAEncrypt(char *text, int size)
6:  {
7:    char out[MAX_TEXT];
8:    RSA *rsa_p = RSA_new();
9:    /* RSA함수에서 NO_PADDING과 같이 적절하지 않은 파라미터를 사용하는 경우*/
10:   RSA_public_encrypt(size, text, out, rsa_p, RSA_NO_PADDING);
11: }
```

보다 강력한 암호화를 위해서는 **RSA_NO_PADDING** 이외의 다른 파라미터를 사용하여야 한다.

안전한 코드의 예 - C

```
12:  #include <stdio.h>
13:  #include <stdlib.h>
14:  #include <openssl/rsa.h>
15:  #define MAX_TEXT 512
16:  void RSAEncrypt(char *text, int size)
17:  {
18:    char out[MAX_TEXT];
19:    RSA *rsa_p = RSA_new();
20:    RSA_public_encrypt(size, text, out, rsa_p, RSA_PKCS1_OAEP_PADDING);
21:  }
```

라. 참고문헌

[1] CWE-325 Missing Required Cryptographic Step,
http://cwe.mitre.org/data/definitions/325.html

[2] 2010 OWASP Top 10 - A7 Insecure Cryptographic Storage
https://www.owasp.org/index.php/Top_10_2010-A7

16. 취약한 암호화 해쉬함수: 하드코드된 솔트(Weak Cryptographic Hash: Hardcoded Salt)

가. 정의

코드에 고정된 솔트값을 사용하는 것은 프로젝트의 모든 개발자가 그 값을 볼 수 있으며, 추후 수정이 매우 어렵다는 점에서 시스템의 취약점으로 작용할 수 있다. 만약 공격자가 솔트값을 알게 된다면, 해당 응용프로그램의 레인보우 테이블을 작성하여 해쉬 결과값을 역으로 계산할 수 있다.

나. 안전한 코딩기법

- crypt() 등의 암호화 함수에서 사용하는 솔트값을 상수로 사용하지 않도록 설계한다.

다. 예제

crypt()함수에서 사용하는 솔트값이 상수 스트링을 사용하고 있다. 상수 값이 노출되었을 때 암호화된 값이 노출될 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <unistd.h>
4:  #define MAX_TEXT 100
5:  void hardSalt(const char *text)
6:  {
7:    char *out;
8:    /* salt값으로 상수를 사용하고 있다 */
9:    out = (char*) crypt(text, "xp");
10: }
```

암호화에 사용하는 솔트값은 예측이 어려운 난수 값으로 사용할 때마다 새로 만들어서 사용하여야 한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <unistd.h>
4:  #define MAX_TEXT 100
5:  void hardSalt(const char *text, const char *os)
6:  {
7:    char *out;
8:    out = (char *) crypt(text, os);
9:  }
```

encryptPassword함수에서 사용하는 솔트값으로 상수 스트링인 **salt_string**을 사용하고 있다. 솔트의 상수 값이 노출되는 경우, 보다 쉽게 암호를 풀 수 있는 단서를 제공하게 된다.

■ 안전하지 않은 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:    const char * COMMAND_PARAM = "command";
5:    const char * CHANGE_USER_PASSWORD_CMD = "change_user_password";
6:
7:    char * queryStr;
8:    queryStr = getenv("QUERY_STRING");
9:    if (queryStr == NULL)
10:   {
11:     // Error 처리
12:     ...
13:   }
14:   char * command = GetParameter(queryStr, COMMAND_PARAM);
15:   if (!strcmp(command, CHANGE_USER_PASSWORD_CMD))
16:   {
17:     const char * USER_ID_PARAM = "user_id";
18:     const char * PASSWORD_PARAM = "password";
19:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
20:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
21:
22:     char * encryptedPassword = encryptPassword(password, "salt_string");
23:     ...
24:     changePassword(user_id, encryptedPassword);
25:     free(userId);
26:     free(password);
27:   }
28:   ...
29:   return EXIT_SUCCESS;
30: }
```

암호화에 사용하는 솔트값은 예측이 어려운 난수 값으로 사용할 때마다 새로 만들어서 사용하여야 한다.

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:    const char * COMMAND_PARAM = "command";
5:    const char * CHANGE_USER_PASSWORD_CMD = "change_user_password";
6:
```

```
7:    srand ( time(NULL) );
8:
9:    char * queryStr;
10:   queryStr = getenv("QUERY_STRING");
11:   if (queryStr == NULL)
12:   {
13:     // Error 처리
14:     ...
15:   }
16:   char * command = GetParameter(queryStr, COMMAND_PARAM);
17:   if (!strcmp(command, CHANGE_USER_PASSWORD_CMD))
18:   {
19:     const char * USER_ID_PARAM = "user_id";
20:     const char * PASSWORD_PARAM = "password";
21:     const char * userId = GetParameter(queryStr, USER_ID_PARAM);
22:     const char * password = GetParameter(queryStr, PASSWORD_PARAM);
23:     ...
24:     char * salt = generateSalt();
25:     char * encryptedPassword = encryptPassword(password, salt);
26:     ...
27:     changePassword(user_id, encryptedPassword);
28:
29:     free(userId);
30:     free(password);
31:   }
32:   ...
33:   return EXIT_SUCCESS;
34: }
35: char * generateSalt()
36: {
37:   const int SALT_LENGTH = 8;
38:   char * codes =
39:     "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
40:
41:   char * salt = (char *)malloc(SALT_LENGTH + 1);
42:   int saltIndex;
43:   for(saltIndex = 0; saltIndex < SALT_LENGTH; saltIndex++)
44:   {
45:     salt[saltIndex] = codes[rand() % strlen(codes)];
46:   }
47:
48:   return salt;
49: }
```

라. 참고문헌

[1] CWE-326 Inadequate Encryption Strength,
http://cwe.mitre.org/data/definitions/326.html

[2] 2010 OWASP Top 10 - A7 Insecure Cryptographic Storage,
https://www.owasp.org/index.php/Top_10_2010-A7

17. 같은 포트번호로의 다중 연결(Multiple Binds to the Same Port)

가. 정의

하나의 포트에 다수의 소켓이 연결되는 것을 허용하는 경우, 주어진 포트에서 수행되는 서비스로 전달되는 패킷이 도난당하거나 혹은 공격자가 서비스를 도용할 수 있다.

나. 안전한 코딩기법

- 소켓 옵션 SO_REUSEADDR와 서버 주소값 INADDR_ANY를 동시에 사용하지 않는다.

다. 예제

소켓의 옵션을 **SO_REUSEADDR**으로 세팅하고, 서버 주소 값으로 **INADDR_ANY**을 사용하는 경우 하나의 포트에 여러 개의 소켓이 바인딩 될 수 있다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <sys/types.h>
3:  #include <sys/socket.h>
4:  #include <netinet/in.h>
5:  void bind_socket(void)
6:  {
7:    int server_sockfd;
8:    int server_len;
9:    struct sockaddr_in server_address;
10:   int optval;
11:   unlink("server_socket");
12:   server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
13:   optval = 1;
14:   setsockopt(server_sockfd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof optval);
15:   server_address.sin_family = AF_INET;
16:   server_address.sin_port = 21;
17:   server_address.sin_addr.s_addr = htonl(INADDR_ANY);
18:   /* 소켓 옵션 SO_REUSEADDR과 서버 주소값 INADDR_ANY을 동시에 사용 */
19:   server_len = sizeof(struct sockaddr_in);
20:   bind(server_sockfd, (struct sockaddr *) &server_address, server_len);
21: }
```

소켓의 옵션 **SO_REUSEADDR**을 서버 주소 값으로 **INADDR_ANY**과 동시에 사용하지 않는다.

안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <sys/types.h>
3:  #include <sys/socket.h>
```

```
4:  #include <netinet/in.h>
5:  void bind_socket(void)
6:  {
7:     int server_sockfd;
8:     int server_len;
9:     struct sockaddr_in server_address;
10:    unlink("server_socket");
11:    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
12:    server_address.sin_family = AF_INET;
13:    server_address.sin_port = 21;
14:    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
15:    server_len = sizeof(struct sockaddr_in);
16:    bind(server_sockfd, (struct sockaddr *) &server_address, server_len);
17:    closesocket(server_sockfd);
18: }
```

라. 참고문헌

[1] CWE-605 Multiple Binds to the Same Port, http://cwe.mitre.org/data/definitions/605.html

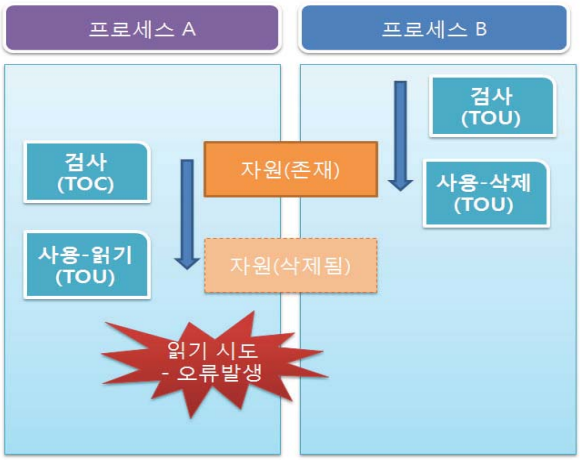
제3절 시간 및 상태

시간과 상태에 대한 취약점이란 프로그램의 동작 과정에서 시간적 개념을 포함한 개념이나 시스템 상태에 대한 정보에 관련된 취약점을 말한다. 이러한 취약점에 속하는 것들로는 자원에 대한 경쟁조건 등을 들 수 있다.

1. 경쟁 조건: 검사시점과 사용시점(Time-of-check Time-of-use(TOCTOU) Race Condition)

가. 정의

병렬시스템(멀티프로세스로 구현한 응용프로그램)에서는 자원(파일, 소켓 등)을 사용하기에 앞서 자원의 상태를 검사한다. 하지만 자원을 사용하는 시점과 검사하는 시점이 다르기 때문에, 검사하는 시점(time of check)에 존재하던 자원이 사용하던 시점(time of use)에 사라지는 등 자원의 상태가 변하는 경우가 발생한다. 하나의 자원에 대하여 동시에 검사시점과 사용시점이 달라 생기는 취약점으로 인해 동기화 오류 뿐 아니라 교착상태 등과 같은 문제점이 발생한다.



<그림 2-7> 경쟁조건 : 검사시점과 사용시점(TOCTOU)

나. 안전한 코딩기법

- 파일 이름으로 권한을 검사하고 오픈하면 보안에 취약하다. 파일 핸들을 사용하길 권장한다.
- 상호배제(mutex)를 사용한다.

다. 예제

access() 함수로 파일의 존재를 확인할 때와 **fopen()** 함수로 실제로 파일을 오픈하여 사용하는 부분에서 시간차가 발생하는 경우 다른 파일을 오픈하게 되는 등의 프로그램이 예상하지 못하는 형태로 수행될 수 있기 때문에 취약하다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <unistd.h>
3:  void file_operation(char* file)
4:  {
5:  /* 파일검사 후 파일열기를 수행 시 자원의 상태가 다를 수 있어 취약하다 */
6:    if(!access(file,W_OK))
7:    {
8:     f = fopen(file,"w+");
9:     operate(f);
10:   }
11:   else
12:   {
13:   fprintf(stderr,"Unable to open file %s.\n",file);
14:   }
15:  }
```

파일 이름으로 권한 검사하고 오픈하는 것이 아니라, 파일 핸들을 사용하여 파일을 오픈한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <unistd.h>
3:  void file_operation(char* file)
4:  {
5:    char *file_name;
6:    int fd;
7:    /* 파일을 열 때, open()을 사용하여, mode를 지정한다. */
8:    fd = open( file_name, O_WRONLY | O_CREAT | O_EXCL, S_IRWXU);
9:    if (fd == -1)
10:   {
11:     ...
12:   }
13:   /* chmod 대신 fchmod를 사용하여, open의 fd를 그대로 사용한다. */
14:   if (fchmod(fd, S_IRUSR) == -1)
15:   {
16:     ...
17:   }
18:   close(fd);
19: }
```

pthread_mutex_lock()의 반환값에 대하여 에러검사를 하지 않는다. **pthread_mutex_lock()**이 mutex를 얻어 오는데 실패하면, 경쟁상태를 발생하게 되고 예측할 수 없는 동작을 하게 된다.

■ 안전하지 않은 코드의 예 - C

```
1:  void handle_mutex(pthread_mutex_t *mutex)
2:  {
3:      ... ...
4:      pthread_mutex_lock(mutex);
5:      /* 임계코드 안에서 공유메모리 참조 */
6:      pthread_mutex_unlock(mutex);
7:      ... ...
8:  }
```

임계코드에 대한 lock을 실행 후 결과를 체크하여 예측할 수 없는 결과를 방지한다.

■ 안전한 코드의 예 - C

```
1:  int handle_mutex(pthread_mutex_t *mutex)
2:  {
3:      int result;
4:      // mutex 사용 시에 함수 결과를 항상 체크
5:      result = pthread_mutex_lock(mutex);
6:      if (0 != result)
7:          return result;
8:      /* 공유메모리 참조 */
9:      return pthread_mutex_unlock(mutex);
10: }
```

다음의 예제는 로그인 기록을 로그로 남기고 있다. mutex개념을 넣지 않았기 때문에 로그를 쓰는 동안 같은 작업 요청이 또 들어오는 경우, 같은 로그파일에 대해 동시에 쓰기를 수행하려고 해서 오작동이 발생할 가능성이 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  /**
2:   * Servlet implementation class SqlInjection
3:   */
4:   int main(void)
5:  {
6:   const char * COMMAND_PARAM = "command";
7:   const char * GET_USER_INFO_CMD = "get_user_info";
8:
9:   char * queryStr;
10:  queryStr = getenv("QUERY_STRING");
11:  if (queryStr == NULL)
12:  {
13:    // Error 처리
14:    ...
```

```
15:     }
16:     char * command = GetParameter(queryStr, COMMAND_PARAM);
17:     if (!strcmp(command, GET_USER_INFO_CMD))
18:     {
19:       const char * USER_ID_PARAM = "user_id";
20:       const char * PASSWORD_PARAM = "password";
21:       const int MAX_PASSWORD_LENGTH =16;
22:       const char * userId = GetParameter(queryStr, USER_ID_PARAM);
23:       const char * password = GetParameter(queryStr, PASSWORD_PARAM);
24:       free(userId);
25:       free(password);
26:     }
27:     return EXIT_SUCCESS;
28:   }
29:
30:   int main()
31:   {
32:     const char * COMMAND_PARAM = "command";
33:
34:     // Command 관련 정의
35:     const char * LOGIN_CMD = "login";
36:     const char * USER_ID_PARM = "user_id";
37:     const char * PASSWORD_PARM = "password";
38:
39:     char * queryStr;
40:     queryStr = getenv("QUERY_STRING");
41:     if (queryStr == NULL)
42:     {
43:       // Error 처리
44:       ...
45:     }
46:     char * command = GetParameter(queryStr, COMMAND_PARAM);
47:     if (!strcmp(command, GET_USER_INFO_CMD))
48:     {
49:       const char * USER_ID_PARAM = "user_id";
50:       const char * PASSWORD_PARAM = "password";
51:       const int MAX_PASSWORD_LENGTH = 16;
52:       const char * userId = GetParameter(queryStr, USER_ID_PARAM);
53:       const char * password = GetParameter(queryStr, PASSWORD_PARAM);
54:
55:       if(checkLoginInfo(userId, password) == true)
56:       {
57:         recordLoginLog(userId, password);
58:       }
59:       ...
```

```
60:     free(userId);
61:     free(password);
62:   }
63:   return EXIT_SUCCESS;
64: }
65:
66: void recordLoginLog(String userId, String password)
67: {
68:   if(canWrite("log") == true)
69:   {
70:     // login log를 기록한다.
71:   }
72: }
```

log파일을 쓰기 전에 먼저 lock시켜놓고 로그를 쓰도록 하며, 로그를 다 쓴 후에는 lock을 풀도록 설계한다.

■ 안전한 코드의 예 - C

```
1:  /**
2:   * Servlet implementation class SqlInjection
3:   */
4:   int main(void)
5:  {
6:   const char * COMMAND_PARAM = "command";
7:   const char * GET_USER_INFO_CMD = "get_user_info";
8:
9:   char * queryStr;
10:  queryStr = getenv("QUERY_STRING");
11:  if (queryStr == NULL)
12:  {
13:    // Error 처리
14:    ...
15:  }
16:  char * command = GetParameter(queryStr, COMMAND_PARAM);
17:  if (!strcmp(command, GET_USER_INFO_CMD))
18:  {
19:    const char * USER_ID_PARAM = "user_id";
20:    const char * PASSWORD_PARAM = "password";
21:    const int MAX_PASSWORD_LENGTH = 16;
22:    const char * userId = GetParameter(queryStr, USER_ID_PARAM);
23:    const char * password = GetParameter(queryStr, PASSWORD_PARAM);
24:    free(userId);
25:    free(password);
26:  }
27:  return EXIT_SUCCESS;
```

```
28:  }
29:
30:  int main()
31:  {
32:   const char * COMMAND_PARAM = "command";
33:
34:   // Command 관련 정의
35:   const char * LOGIN_CMD = "login";
36:   const char * USER_ID_PARM = "user_id";
37:   const char * PASSWORD_PARM = "password";
38:
39:   char * queryStr;
40:   queryStr = getenv("QUERY_STRING");
41:   if (queryStr == NULL)
42:   {
43:    // Error 처리
44:    ...
45:   }
46:   char * command = GetParameter(queryStr, COMMAND_PARAM);
47:   if (!strcmp(command, GET_USER_INFO_CMD))
48:   {
49:    const char * USER_ID_PARAM = "user_id";
50:    const char * PASSWORD_PARAM = "password";
51:    const int MAX_PASSWORD_LENGTH = 16;
52:    const char * userId = GetParameter(queryStr, USER_ID_PARAM);
53:    const char * password = GetParameter(queryStr, PASSWORD_PARAM);
54:
55:    if(checkLoginInfo(userId, password) == true)
56:    {
57:     recordLoginLog(userId, password);
58:    }
59:    ...
60:    free(userId);
61:    free(password);
62:   }
63:   return EXIT_SUCCESS;
64:  }
65:
66:  pthread_mutex_t logMutex = PTHREAD_MUTEX_INITIALIZER;
67:  void recordLoginLog(String userId, String password)
68:  {
69:   pthread_mutex_lock(&logMutex);
70:   if(canWrite("log") == true)
71:   {
72:    // login log를 기록한다.
```

```
73:    }
74:    pthread_mutex_unlock(&logMutex);
75:  }
```

라. 참고문헌

[1] CWE-367 Time-of-check Time-of-use(TOCTOU) Race Condition, http://cwe.mitre.org/data/definitions/367.html

[2] CWE-362 Concurrent Execution using Shared Resource with Improper Synchronization (Race Condition), http://cwe.mitre.org/data/definitions/362.html

[3] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 13: Race Conditions." Page 205. McGraw-Hill. 2010

[4] Andrei Alexandrescu. "volatile - Multithreaded Programmer's Best Friend". Dr. Dobb's. 2008-02-01

[5] Steven Devijver. "Thread-safe webapps using Spring"
David Wheeler. "Prevent race conditions". 2007-10-04

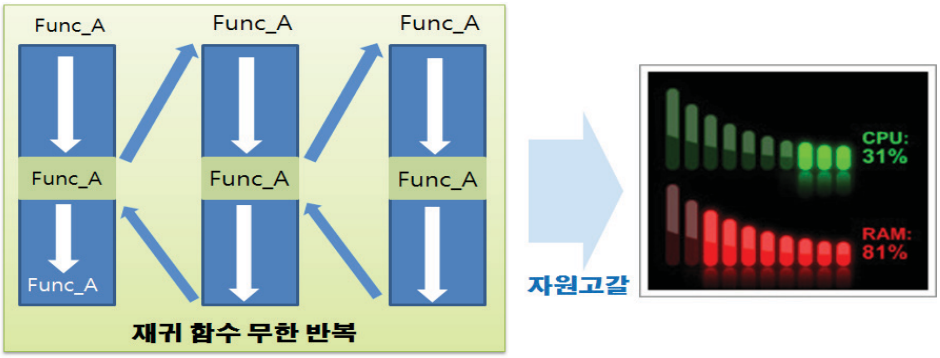
[6] Matt Bishop. "Race Conditions, Files, and Security Flaws; or the Tortoise and the Hare Redux". September 1995

[7] Johannes Ullrich. "Top 25 Series - Rank 25 - Race Conditions". SANS Software Security Institute. 2010-03-26

2. 제대로 제어되지 않은 재귀(Uncontrolled Recursion)

가. 정의

재귀의 순환횟수를 제어하지 못하여 할당된 메모리나 프로그램 스택 등의 자원을 과다하게 사용하면 위험하다. 귀납 조건(base case)이 없는 재귀함수는 대부분의 경우 무한 루프에 빠져 들게 되고 자원고갈을 유발함으로써 시스템의 정상적인 서비스를 제공할 수 없게 한다.



<그림 2-8> 제어문을 사용하지 않는 재귀함수

위 그림은 함수 Func_A가 재귀 함수 형태로 무한 반복함으로써 자원 고갈을 유발하는 경우를 보여주고 있다.

나. 안전한 코딩기법

- 무한 재귀를 방지하기 위하여 모든 재귀 호출을 조건문 블럭이나 반복문 블럭 안에서만 수행해야 한다.
- 모든 재귀 호출을 조건문 블럭이나 반복문 블럭 안에서만 수행한다.

다. 예제

무한 루프에 빠지는 재귀호출 함수이다.

안전하지 않은 코드의 예 - C

```
1:  int fac(n)
2:  {
3:     return n*fac(n-1);
4:  }
```

이와 같은 재귀 호출은 조건문이나 반복문 같은 제어문을 통하여 다음과 같이 해당 루프를 빠져 나올 수 있게 기술되어야 한다.

■ 안전한 코드의 예 - C

```
1:
1:   int fac(n)
2:   {
3:     if (n <= 0) return 1;
4:     else return n*fac(n-1);
5:   }
```

재귀함수 호출이 10000번 이상 되면 프로그램이 강제 종료되도록 했다. 경우에 따라서는 일정 시간 안에 결과가 도출되지 않으면 강제로 프로그램을 종료시켜주는 것이 동작 안정성을 보장해줄 수 있다.

■ 안전한 코드의 예 - C

```
1:   #include <stdio.h>
2:   #include <unistd.h>
3:   #include <string.h>
4:   int i=0;
5:   int fac(n)
6:   {
7:    if (n <= 0)
8:      return 1;
9:    else
10:   {
11:     if(i>10000) exit(1);
12:     else i+=1;
13:     return n*fac(n-1);
14:   }
15:  }
```

라. 참고문헌

[1] CWE-674 Uncontrolled Recursion, http://cwe.mitre.org/data/definitions/674.html

3. 심볼릭명이 정확한 대상에 매핑되어 있지 않음(Symbolic Name not Mapping to Correct Object)

가. 정의

어떤 대상에 대한 심볼릭 참조가 시간에 따라 바뀌는 경우를 말한다. 프로그램이 심볼릭 명을 사용하여 특정 대상을 지정하는 경우 공격자는 심볼릭명이 가리키는 대상을 조작하여 프로그램이 원래 의도했던 동작을 못하게 할 수 있다.

나. 안전한 코딩기법

- 심볼릭 링크를 사용한 파일 바꿔치기를 염두해 두고 프로그래밍 해야 한다.

다. 예제

TOCTOU 문제(KCWE-367)와 같은 취약점으로 파일이 심볼릭 링크인 경우 파일이름을 조작하여 공격자가 파일을 바꿔치기 할 수 있다. 즉, **access()** 함수로 검사할 때와 **fopen()** 함수으로 파일을 열 때 시간차가 있어 다른 파일을 열게 할 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  int main(int argc, char* argv[])
4:  {
5:    char* file;
6:    FILE *f;
7:    /* 파일검사 후 파일을 열 때 생기는 시간에 심볼릭 링크를 변경할 수 있는 취약점이 존재  한다 */
8:    if(!access(file,W_OK))
9:    {
10:     f = fopen(file,"w+");
11:     operate(f);
12:     ……
13:   }
14:   else
15:   {
16:     fprintf(stderr,"Unable to open file %s.\n",file);
17:   }
18: }
```

심볼릭 링크를 사용한 파일 바꿔치기를 하지 못하도록 파일 이름을 유기적으로 생성한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <unistd.h>
```

```
3:  #include <string.h>
4:  int main (int argc, char* argv[])
5:  {
6:    char* filename;
7:    /* 랜덤한 임시 파일 이름 생성*/
8:    if(mkstemp(filename))
9:    {
10:     FILE* tmp = fopen(filename,"wb+");
11:     while((recv(sock,recvbuf,DATA_SIZE, 0) > 0) && (amt!=0))
12:     amt = fwrite(recvbuf,1,DATA_SIZE,tmp);
13:   }
14: }
```

다음의 예제는 디렉토리에서 특정 파일을 찾는 프로그램이다. 디렉터리나 파일 중 심볼릭 링크나 바로가기 같은 것이 존재하면 엉뚱한 결과 값을 얻거나 무한루프에 빠질 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  //Symbolic link로 연결된 file을 읽음
2:  lrwxrwxrwx passwd 1 user Domain Users .... passwd -> /etc/passwd
3:  ----------------------------------------
4:  FILE * fp = fopen("./passwd");
5:  char * line = fgets(fp);
```

재귀 호출에서는 심볼릭 링크를 따라가지 않도록 막는 루틴이 필요하다.

■ 안전한 코드의 예 - C

```
1:  // Directory 중에 Symbolic link(or 바로가기)가 있을 경우, 끝나지 않을 수 있다.
2:  public char * findFile(const char * rootDir, const char * fileName)
3:  {
4:   char * fileList [] = getSubFiles(rootDir);
5:   char * path = NULL;
6:   for(int fileIdx = 0; fileList[fileIdx] != NULL; fileIdx++)
7:   {
8:    if(path == NULL)
9:    {
10:     if(isDirectory(fileList[fileIdx]) == TRUE)
11:     {
12:      char * foundPath = fileFile(fileList[fileIdx], fileName);
13:      if(foundPath != NULL)
14:      {
15:       path = (char *)malloc(strlen(fileList[fileIdx]) + strlen(foundPath));
16:       strcpy(path, fileList[fileIdx]);
17:       strcat(path, foundPath);
18:       free(foundPath);
```

```
19:         }
20:       }
21:       else
22:       {
23:         if(fileList[fileIdx].equals(fileName) == true)
24:         {
25:           path =   strdup(fileName);
26:         }
27:       }
28:     }
29:     free(fileList[fileIdx]);
30:   }
31:   free(fileList);
32:   return path;
33: }
```

라. 참고문헌

[1] CWE-386 Symbolic Name not Mapping to Correct Object, http://cwe.mitre.org/data/definitions/386.html

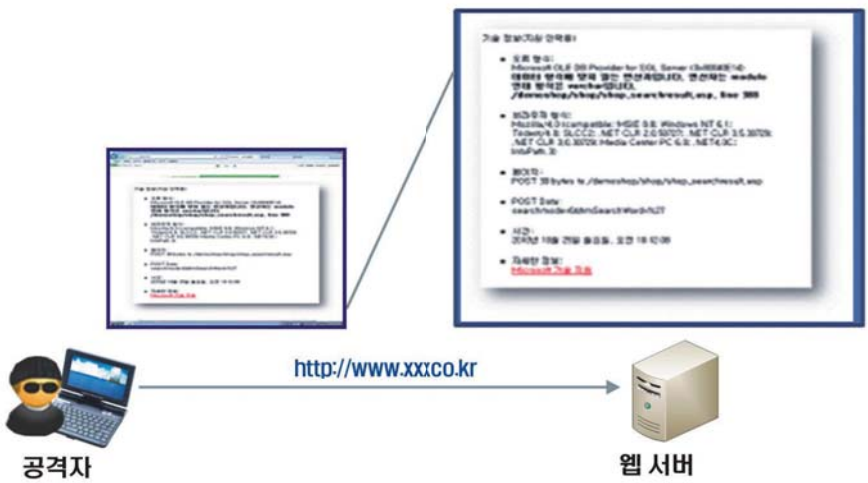
제4절 에러 처리

정상적인 에러는 사전에 정의된 예외사항이 특정 조건에서 발생하는 에러이며, 비정상적인 에러는 사전에 정의되지 않은 상황에서 발생하는 에러이다. 개발자는 정상적인 에러 및 비정상적인 에러 발생에 대비한 안전한 에러처리 루틴을 사전에 정의하고 프로그래밍 함으로써 에러처리 과정 중에 발생할 수 있는 보안 위협을 미연에 방지할 수 있다. 에러를 부적절하게 처리(혹은 전혀 처리) 하지 않을 때 혹은 에러 정보가 과도하게 많은 정보를 포함하여 이를 공격자가 악용할 수 있을 때 보약취약점이 발생할 수 있다.

1. 오류 메시지 통한 정보 노출(Information exposure through an error message)

가. 정의

응용프로그램이 실행환경, 사용자, 관련 데이터에 대한 민감한 정보를 포함하는 오류 메시지를 생성하여 외부에 제공하는 경우 공격자의 악성 행위를 도와줄 수 있다. 예외발생 시 예외이름이나 스택트레이스를 출력하는 경우 프로그램 내부구조를 쉽게 파악할 수 있다.



<그림 2-9> 오류 메시지 통한 정보노출

<그림 2-9>는 “오류 메시지 통한 정보 노출”을 나타내주고 있으며, 오류화면을 통해서 해당 시스템의 운영체제가 윈도우 계열이며 데이터베이스는 MS-SQL을 사용함을 알 수 있다.

나. 안전한 코딩기법

- 오류 메시지는 정해진 사용자에게 유용한 최소한의 정보만 포함하도록 한다. 소스코드에서 예외상황은 내부적으로 처리하고 사용자에게 민감한 정보를 포함하는 오류를 출력하지 않도록 설정하고 적절한 환경설정을 통해 에러 정보를 노출하지 않고, 미리 정의된 페이지를 제공하도록 설정한다.

다. 예제

오류 메시지에 환경변수(**MYPATH**) 정보를 출력하면 공격자가 환경변수에 정의된 문제의 파일을 알 수 있게 된다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  int main (int argc, char* argv[])
5:  {
6:     char* path=getenv("MYPATH");
7:     /* 공격자가 환경변수에 정의된 파일을 유추할 수 있다 */
8:     fprintf(stderr,path);
9:     return 0;
10: }
```

오류정보에 대한 공개는 사용자가 유추할 수 없게 최대한 간단하게 사용한다.

■ 안전한 코드의 예 - C

```
1:  ##include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  int main (int argc, char* argv[])
5:  {
6:     char* path=getenv("MYPATH");
7:     return 0;
8:  }
```

오류 메시지에 구체적인 에러 발생 위치에 대한 정보와 세부내용을 출력하면 공격자가 시스템 환경에 대한 정보를 수집할 수 있게 된다.

■ 안전하지 않은 코드의 예 - C

```
1:  public void ReadConfiguration()
2:  {
3:     char buffer[BUFFER_SIZE];
4:     File * fp     = fopen("config.cfg", "r");
5:     if(fp == NULL)
6:     {
7:           printf("config.cfg를 찾을 수 열수 없습니다.");
8:           return;
9:     }
10:    if(fgets(buffer, BUFFER_SIZE, fp) == NULL)
11:    {
```

```
12:             printf("config.cfg에서 option1을 읽을 수 없습니다.");
13:     }
14:     strlcpy(cofiguration.option1, buffer);
15:
16:     if(fgets(buffer, BUFFER_SIZE, fp) == NULL)
17:     {
18:             printf("config.cfg에서 option2을 읽을 수 없습니다.");
19:     }
20:     strlcpy(cofiguration.option2, buffer);
21:
22:     if(fgets(buffer, BUFFER_SIZE, fp) == NULL)
23:     {
24:             printf("config.cfg에서 option3을 읽을 수 없습니다.");
25:     }
26:     strlcpy(cofiguration.option3, buffer);
27: }
```

오류정보에 대한 공개는 사용자가 유추할 수 없게 최대한 간단하게 사용한다. 다음의 예제에서는 어떠한 종류의 에러가 발생하든 공격자는 한 가지 메시지밖에 볼 수 없게 되어 있다.

■ 안전한 코드의 예 - C

```
1:  public int _ReadConfiguration(Configuration * configuration)
2:  {
3:      char buffer[BUFFER_SIZE];
4:      File * fp     = fopen("config.cfg", "r");
5:      if(fp == NULL){   return FALSE;         }
6:
7:      if(fgets(buffer, BUFFER_SIZE, fp) == NULL){      return FALSE;      }
8:      strlcpy(cofiguration->option1, buffer);
9:
10:     if(fgets(buffer, BUFFER_SIZE, fp) == NULL){      return FALSE;      }
11:     strlcpy(cofiguration->option2, buffer);
12:
13:     if(fgets(buffer, BUFFER_SIZE, fp) == NULL){      return FALSE;      }
14:     strlcpy(cofiguration->option3, buffer);
15:     ...
16:     return TRUE;
17: }
18: public void ReadConfiguration(Configuration * configuration)
19: {
20:     if(_ReadConfiguration(configuration) == FALSE)
21:     {
22:             printf("환경 설정에 실패하였습니다.");
23:     }
24: }
```

라. 참고문헌

[1] CWE-209 Information Exposure Through an Error Message, http://cwe.mitre.org/data/definitions/209.html

2. 오류상황 대응 부재(Detection of Error Condition Without Action)

가. 정의

오류가 발생할 수 있는 부분을 확인하였으나, 이러한 오류에 대하여 예외처리를 하지 않을 경우에는 프로그램이 충돌하거나 종료되는 등의 개발자가 의도하지 않은 결과가 발생한다.

나. 안전한 코딩기법

- 오류가 발생할 수 있는 부분에 대하여 제어문을 사용하여 적절하게 예외 처리(C/C++에서 if와 switch, Java에서 try-catch 등)를 한다.

다. 예제

다음 예제는 if 블록에서 발생하는 오류를 포착하고 있지만, 그 오류에 대해서 아무 조치를 하고 있지 않음을 보여준다. 따라서, 프로그램이 계속 실행되기 때문에 프로그램에서는 어떤 일이 있어났는지 전혀 알 수 없게 된다.

안전하지 않은 코드의 예 - C

```
1:  #include "std_testcase.h"
2:
3:  #ifndef OMITBAD
4:
5:  void KRD_402_Error_Without_Action__char_fputs_char_fputs_0101_bad()
6:  {
7:      {
8:          /* FLAW: check the return value, but do nothing if there is an error */
9:          if (fputs("string", stdout) == EOF)
10:         {
11:             /* do nothing */
12:         }
13:     }
14: }
15:
16: #endif /* OMITBAD */
17:
18:
19: /* Below is the main(). It is only used when building this testcase on
20:    its own for testing or for building a binary to use in testing binary
21:    analysis tools. It is not used when compiling all the testcases as one
22:    application, which is how source code analysis tools are tested. */
23:
24: #ifdef INCLUDEMAIN
25:
26: int main(int argc, char * argv[])
```

```
{
    /* seed randomness */
    srand( (unsigned)time(NULL) );
#ifndef OMITBAD
    printLine("Calling bad()...");
    KRD_402_Error_Without_Action__char_fputs_char_fputs_0101_bad();
    printLine("Finished bad()");
#endif /* OMITBAD */
    return 0;
}

#endif
```

예외를 포착(if)한 후, 각각의 예외상황에 대하여 적절하게 처리해야 한다.

■ 안전한 코드의 예 - C

```
#include "std_testcase.h"


#ifndef OMITGOOD

static void good1()
{
    {
        /* FIX: check the return value and handle errors properly */
        if (fputs("string", stdout) == EOF)
        {
            printLine("fputs failed!");
            exit(1);
        }
    }
}

void KRD_402_Error_Without_Action__char_fputs_char_fputs_0101_good()
{
    good1();
}

#endif /* OMITGOOD */

/* Below is the main(). It is only used when building this testcase on
   its own for testing or for building a binary to use in testing binary
   analysis tools. It is not used when compiling all the testcases as one
   application, which is how source code analysis tools are tested. */
```

```
29:
30:   #ifdef INCLUDEMAIN
31:
32:   int main(int argc, char * argv[])
33:   {
34:       /* seed randomness */
35:       srand( (unsigned)time(NULL) );
36:   #ifndef OMITGOOD
37:       printLine("Calling good()...");
38:       KRD_402_Error_Without_Action__char_fputs_char_fputs_0101_good();
39:       printLine("Finished good()");
40:   #endif /* OMITGOOD */
41:       return 0;
42:   }
43:
44:   #endif
```

라. 참고문헌

[1] CWE-390 Detection of Error Condition Without Action,
http://cwe.mitre.org/data/definitions/390.html

3. 적절하지 않은 예외처리(Improper Check for Unusual or Exceptional Conditions)

가. 정의

프로그램 수행 중에 함수의 결과 값에 대한 적절한 처리 또는 예외상황에 대한 조건을 적절하게 검사하지 않을 경우, 예기치 않은 문제를 야기할 수 있다.

나. 안전한 코딩기법

- 값을 반환하는 모든 함수의 결과 값을 검사하여 그 값이 기대한 값인지 검사하고, 예외 처리를 사용하는 경우에 광범위한 예외처리 대신 구체적인 예외처리를 한다.

다. 예제

개발자는 **buf** 길이가 10보다 작은 '\0'문자로 종료될 것이라고 기대하지만, 중간에 에러 발생에 의해 '\0'로 끝나지 않을 경우 **strcpy()**에서 버퍼 오버플로우가 발생할 수 있다.

안전하지 않은 코드의 예 - C

```
1:  char fromBuf[10], toBuf[10];
2:  fgets(fromBuf, 10, stdin);
3:  strcpy(toBuf, fromBuf);
4:  .....
```

fgets() 함수호출 이후 두개의 버퍼를 비교함으로써, **strcpy()**에 의해서 발생하는 버퍼 오버플로우를 미연에 방지할 수 있다.

안전한 코드의 예 - C

```
1:  char fromBuf[10], toBuf[10];
2:  char *retBuf = fgets(fromBuf, 10, stdin);
3:  // 함수호출 이후 결과 값을 비교한다.
4:  if ( retBuf != fromBuf )
5:  {
6:      println("에러");
7:      return;
8:  }
9:  strcpy(toBuf, fromBuf);
10:  ...
```

다음의 예제에서는 파일로부터 데이터를 읽는 작업이 실패하는 경우에 대한 에러 처리 부분이 없다.

안전하지 않은 코드의 예 - C

```
1:  public void ReadConfiguration()
2:  {
```

```
3:      char buffer[BUFFER_SIZE];
4:      File * fp     = fopen("config.cfg", "r");
5:
6:      fgets(buffer, BUFFER_SIZE, fp);
7:      strlcpy(cofiguration.option1, buffer);
8:
9:      fgets(buffer, BUFFER_SIZE, fp);
10:     strlcpy(cofiguration.option2, buffer);
11:
12:     fgets(buffer, BUFFER_SIZE, fp);
13:     strlcpy(cofiguration.option3, buffer);
14:             ...
15:     }
16: }
```

다음의 예제에서 **_ReadConfiguration**함수의 리턴값이 false라면 3개의 작업 중 한 곳에서 에러가 발생한 경우를 뜻하며, 이런 경우에는 모든 옵션을 디폴트값으로 초기화시키고 있다.

■ 안전한 코드의 예 - C

```
1:  public int _ReadConfiguration(Configuration * configuration)
2:  {
3:      char buffer[BUFFER_SIZE];
4:      File * fp     = fopen("config.cfg", "r");
5:      if(fp == NULL){   return FALSE; }
6:
7:      if(fgets(buffer, BUFFER_SIZE, fp) == NULL){      return FALSE;      }
8:      strlcpy(cofiguration->option1, buffer);
9:
10:     if(fgets(buffer, BUFFER_SIZE, fp) == NULL){      return FALSE;      }
11:     strlcpy(cofiguration->option2, buffer);
12:
13:     if(fgets(buffer, BUFFER_SIZE, fp) == NULL){      return FALSE;      }
14:     strlcpy(cofiguration->option3, buffer);
15:     ...
16:     return TRUE;
17: }
18:
19: public void ReadConfiguration(Configuration * configuration)
20: {
21:     if(_ReadConfiguration(configuration) == FALSE)
22:     {
23:             // Configuration을 읽어들이는 것을 실패할 경우, default 값으로 채운다.
24:             if(configuation.option1 == NULL)
25:             {
26:                     configuation.option1        = DEFAULT_OPTION1;
```

```
27:             }
28:             if(configuation.option2 == NULL)
29:             {
30:                     configuation.option2        = DEFAULT_OPTION2;
31:             }
32:             if(configuation.option3 == NULL)
33:             {
34:                     configuation.option3        = DEFAULT_OPTION3;
35:             }
36:     }
37: }
```

라. 참고 문헌

[1] CWE-754 Improper Check for Unusual or Exceptional Conditions, http://cwe.mitre.org/data/definitions/754.html

[2] CWE-252 Unchecked Return Value, http://cwe.mitre.org/data/definitions/252.html

[3] CWE-253 Incorrect Check of Function Return Value, http://cwe.mitre.org/data/definitions/253.html

[4] CWE-273 Improper Check for Dropped Privileges, http://cwe.mitre.org/data/definitions/273.html

[5] CWE-296 Improper Following of Chain of Trust for Certificate Validation, http://cwe.mitre.org/data/definitions/296.html

[6] CWE-297 Improper Validation of Host-specific Certificate Data, http://cwe.mitre.org/data/definitions/297.html

[7] CWE-298 Improper Validation of Certificate Expiration, http://cwe.mitre.org/data/definitions/298.html

[8] CWE-299 Improper Check for Certificate Revocation, http://cwe.mitre.org/data/definitions/299.html

[9] M. Howard, D. LeBlanc, Writing Secure Code, Second Edition, Microsoft Press

제5절 코드 오류

작성 완료된 프로그램은 기능성, 신뢰성, 사용성, 유지보수성, 효율성, 이식성 등을 충족하기 위하여 일정 수준에 코드품질을 유지하여야 한다. 프로그램 코드가 너무 복잡하면 관리성, 유지보수성, 가독성이 떨어질 뿐 아니라 다른 시스템에 이식하기도 힘들며, 프로그램에는 안전성을 위협할 취약점들이 코드 안에 숨겨져 있을 가능성이 있다.

1. 널포인터 역참조(Null Pointer Dereference)

가. 정의

Null 포인터 역참조는 '일반적으로 그 객체가 Null이 될 수 없다'라고 하는 가정을 위반했을 때 발생한다. 공격자가 의도적으로 Null 포인터 역참조를 실행하는 경우, 그 결과 발생하는 예외 사항을 추후의 공격을 계획하는 데 사용될 수 있다.

나. 안전한 코딩기법

- 포인터 참조 응용프로그램에서 포인터가 Null이 아님을 예상하고 접근하는 경우에 프로그램의 충돌이 발생하여 종료가 일어난다. 따라서 변경될 수 있는 모든 포인터는 사용하기 전에 점검한다.
- Null이 될 수 있는 레퍼런스(reference)는 참조하기 전에 Null 값인지를 검사하여 안전한 경우에만 사용한다.

다. 예제

환경변수에서 **CGI_HOME** 문자열로 설정된 값이 없을 경우 **getenv()** 함수에서 Null이 반환된다. Null을 검사하지 않고 사용할 경우 문제가 발생한다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #define BUFSIZE 100
5:  int main()
6:  {
7:     char *p = NULL;
8:     char cgi_home[BUFSIZE];
9:     p = getenv("CGI_HOME");
10:    strncpy(cgi_home, p, BUFSIZE-1);
11:    cig_home[BUFSIZE-1] = '\0';
12:    return 0;
13: }
```

strncpy() 함수를 호출하기 전에 **getenv()** 함수가 반환한 값이 Null인지 여부를 검사한 후 사용한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #define BUFSIZE 100
5:  int main()
6:  {
7:    char *p = NULL;
8:    char cgi_home[BUFSIZE];
9:    p = getenv("CGI_HOME");
10:   if (p == NULL)
11:   {
12:     exit(1);
13:   }
14:   strncpy(cgi_home, p, BUFSIZE-1);
15:   cig_home[BUFSIZE-1] = '\0';
16:   return 0;
17: }
```

다음의 예제는 **user_supplied_addr, hp**에 대한 Null체크를 하지 않고 사용하고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  void host_lookup(char *user_supplied_addr)
2:  {
3:    struct hostent *hp;
4:    in_addr_t *addr;
5:    char hostname[64];
6:    in_addr_t inet_addr(const char *cp);
7:
8:    /*routine that ensures user_supplied_addr is in the right format for conversion */
9:    validate_addr_form(user_supplied_addr);
10:   addr = inet_addr(user_supplied_addr);
11:   hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
12:   strcpy(hostname, hp->h_name);
13: }
```

함수의 입력값으로 사용하는 변수들은 Null여부를 먼저 체크하고 쓰는 것이 안전하다.

■ 안전한 코드의 예 - C

```
1:  void host_lookup(char *user_supplied_addr)
2:  {
3:    struct hostent *hp;
4:    in_addr_t *addr;
5:    char hostname[64];
```

```
6:    in_addr_t inet_addr(const char *cp);
7:
8:    if(user_supplied_addr == NULL)  return;
9:
10:   /*routine that ensures user_supplied_addr is in the right format for conversion */
11:   validate_addr_form(user_supplied_addr);
12:   addr = inet_addr(user_supplied_addr);
13:   hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
14:   if(hp == NULL)  return;
15:   if(hp->h_name == NULL)  return;
16:   strcpy(hostname, hp->h_name);
17: }
```

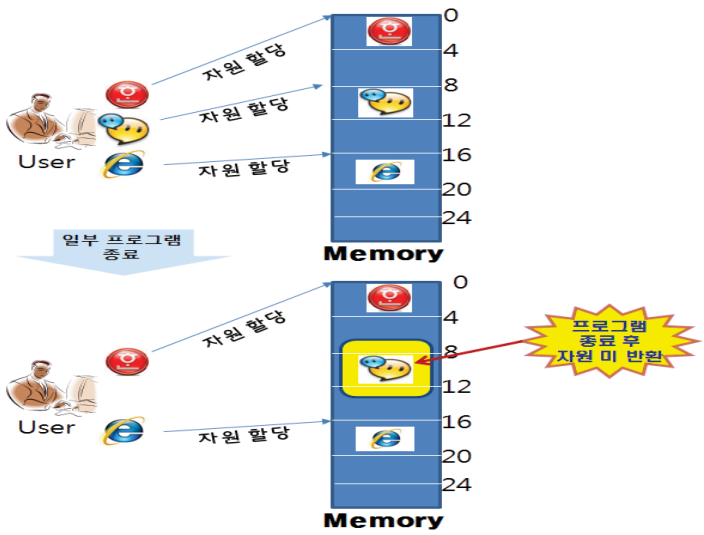
라. 참고문헌

[1] CWE-476 NULL Pointer Dereference, http://cwe.mitre.org/data/definitions/476.html

2. 부적절한 자원 해제(Improper Resource Shutdown or Release)

가. 정의

프로그램의 자원, 예를 들면 열린 파일 기술자(open file descriptor), 힙 메모리 (heap memory), 소켓(socket) 등은 유한한 자원이다. 이러한 자원을 할당받아 사용한 후, 더 이상 사용하지 않는 경우에는 적절히 반환하여야 하는데 프로그램 오류 또는 에러로 사용이 끝난 자원을 반환하지 못 하는 경우가 발생할 수 있다.



<그림 2-10> 부적절한 자원 해제

나. 안전한 코딩기법

- 자원을 할당하고 반환하는 함수들의 경우 모드 패스에서 한 쌍으로 존재하도록 강제한다.

다. 예제

ODBC API를 사용해서 environment, dbconnection 핸들을 할당 받고 사용한 후에 반환하지 않고 있다.

안전하지 않은 코드의 예 - C

```
1:  ……
2:  void sqlDB()
3:  {
4:    SQLHANDLE env_hd, con_hd;
5:    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env_hd);
6:    SQLAllocHandle(SQL_HANDLE_DBC, env_hd, &con_hd);
7:  }
8:  void serverSock()
9:  {
```

```
10:     struct sockaddr_in serverAddr;
11:     struct sockaddr *server = (struct sockaddr *)(&serverAddr);
12:     int listenFd = socket(AF_INET, SOCK_STREAM, 0);
13:     bind(listenFd, server, sizeof(serverAddr));
14:     listen(listenFd, 5);
15:     while (1)
16:     {
17:       int connectFd = accept(listenFd, (struct sockaddr *) NULL, NULL);
18:       shutdown(connectFd, 2);
19:     }
20:   }
```

ODBC API를 사용해서 environment, dbconnection 핸들을 할당 받고 사용한 후에 반환하도록 한다.

■ 안전한 코드의 예 - C

```
1:    ……
2:    void sqlDB()
3:    {
4:      SQLHANDLE env_hd, con_hd;
5:      SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &;env_hd);
6:      SQLAllocHandle(SQL_HANDLE_DBC, env_hd, &;con_hd);
7:      SQLFreeHandle(SQL_HANDLE_DBC, con_hd);
8:      SQLFreeHandle(SQL_HANDLE_ENV, env_hd);
9:    }
10:    void serverSock()
11:    {
12:      struct sockaddr_in serverAddr;
13:      struct sockaddr *server = (struct sockaddr *)(&serverAddr);
14:      int listenFd = socket(AF_INET, SOCK_STREAM, 0);
15:      bind(listenFd, server, sizeof(serverAddr));
16:      listen(listenFd, 5);
17:      while (1)
18:      {
19:        int connectFd = accept(listenFd, (struct sockaddr *) NULL, NULL);
20:        shutdown(connectFd, 2);
21:        close(connectFd);
22:      }
23:      shutdown(listenFd, 2);
24:      close(listenFd);
25:    }
```

다음의 예제에서는 **fork()**가 실패하면 소켓자원을 반환하지 않고 그대로 프로그램을 종료하고 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  int main(int argc, char * argv[])
2:  {
3:    int listenSocket, connectionSocket;
4:    struct sockaddr_in serv_addr, cli_addr;
5:    socklen_t clilen;
6:
7:    listenSocket = socket(AF_INET_SOCK_STREAM, 0);
8:    ...
9:    while(TRUE)
10:   {
11:     connectionSocket  = accpet(listenSocket, (struct sockaddr *) &cli_addr, &clilen);
12:     if(fork() == 0)
13:     {
14:       comminucation();
15:     }
16:     ...
17:   }
18:   close(listenSocket);
19: }
20: void comminucation()
21: {
22:   exit(1);  // socket을 반환하지 않고 process 종료
23: }
```

다음의 예제처럼 어떤 경우가 발생하더라도 쓰여진 자원들을 반환하고 프로그램을 종료할 수 있는 견고한 프로그램 작성 요령이 필요하다.

■ 안전한 코드의 예 - C

```
1:  int main(int argc, char * argv[])
2:  {
3:    int listenSocket =  -1, connectionSocket = -1;
4:    struct sockaddr_in serv_addr, cli_addr;
5:    socklen_t clilen;
6:
7:    listenSocket = socket(AF_INET_SOCK_STREAM, 0);
8:    if(listenSocket == -1)  return 0;
9:    ...
10:   while(TRUE)
11:   {
12:     connectionSocket  = accpet(listenSocket, (struct sockaddr *) &cli_addr, &clilen);
13:     if(connectionSocket == -1)
14:     {
```

```
15:           break;
16:       }
17:       switch(fork())
18:       {
19:       case 0:
20:           listenSocket  = -1;
21:           comminucation();
22:           break;
23:       case -1:  // error case
24:           break;
25:       default:
26:           connectionSocket  = -1;
27:       }
28:   }
29:
30:   // 어떠한 경우에도 열린 socket을 close하고 종료
31:   if(listenSocket  != -1)
32:   {
33:       close(listenSocket);
34:   }
35:   if(connectionSocket  != -1)
36:   {
37:       close(connectionSocket);
38:   }
39: }
40: void comminucation()
41: {
42:     ...
43: }
```

라. 참고문헌

[1] CWE-404 Improper Resource Shutdown or Release,
http://cwe.mitre.org/data/definitions/404.html

3. 부호 정수를 무부호 정수로 타입 변환 오류(Signed to Unsigned Conversion Error)

가. 정의

부호 정수 (signed integer)를 무부호 정수 (unsigned integer)로 변환하면서 음수가 큰 양수로 변환될 수 있다. 이 값이 배열의 인덱스로 사용되는 경우 배열의 범위를 넘어서 접근이 될 수도 있고, 정수 오버플로우로 인한 SW의 오작동을 만들어 낼 수 있다.

나. 안전한 코딩기법

- 강한 타입체크를 수행한다.

다. 예제

문자열 길이를 구하는 **len()**함수에서 입력 문자열이 Null인 경우 -1을 반환하는데 이 값이 **unsigned int**로 변환되면 **int**가 4byte일 경우 4,294,967,295(0xffffffff) 값을 갖는다. 이 값을 사용했을 경우 버퍼 오버플로우 취약점을 가질 수 있으며 이외에도 잘못된 값이 코드 다른 부분의 정확성에 영향을 미쳐 취약한 코드가 될 가능성이 있다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #define BUFSIZE 100
5:  unsigned int len(char *s)
6:  {
7:    unsigned int l = 0;
8:    if (s == NULL)
9:    {
10:     return -1;
11:   }
12:    l = strnlen(s, BUFSIZE-1);
13:    return l;
14:  }
15:  int main(int argc, char **argv)
16:  {
17:    char buf[BUFSIZE];
18:    unsigned int l = 0;
19:    l = len(argv[1]);
20:    strncpy(buf, argv[1], l);
21:    buf[l] = '\0';
22:    printf("last character : %c\n", buf[l-1]);
23:
24:    return 0;
25:  }
```

len() 함수의 입력 값이 Null인 경우 음수 -1 대신 0을 반환하도록 하고 문자열의 길이가 0보다 큰 경우만 **strncpy()** 등을 사용한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #define BUFSIZE 100
5:  unsigned int len(char *s)
6:  {
7:    unsigned int l = 0;
8:    if (s == NULL)
9:    {
10:     return 0;
11:   }
12:   l = strnlen(s, BUFSIZE-1);
13:   return l;
14: }
15: int main(int argc, char **argv)
16: {
17:   char buf[BUFSIZE];
18:   unsigned int l = 0;
19:   l = len(argv[1]);
20:   if (l > 0)
21:   {
22:     strncpy(buf, argv[1], l);
23:     buf[l] = '\0';
24:     printf("last character : %c\n", buf[l-1]);
25:   }
26:   return 0;
27: }
```

strm값으로 문자열 -1 을 주면 **len**은 signed short타입의 숫자 -1이 되며 **sizeof()**함수가 오작동을 일으키게 된다. 외부 입력에 대해서는 허용 범위에 대한 확인이 선행되어야 한다.

■ 안전하지 않은 코드의 예 - C

```
1:  char* processNext(char* strm)
2:  {
3:    char buf[512];
4:    short len = *(short*) strm;
5:    strm += sizeof(len);
6:    if (len <= 512)
7:    {
```

```
8:      memcpy(buf, strm, len);
9:      process(buf);
10:     return strm + len;
11:   }
12:   else
13:   {
14:     return -1;
15:   }
16: }
```

len 값이 음수가 아닐 경우에만 이후의 프로세스를 진행하도록 하였다.

■ 안전한 코드의 예 - C

```
1:  char* processNext(char* strm)
2:  {
3:    char buf[512];
4:    short len = *(short*) strm;
5:    if(len < 0)  return -1;
6:
7:    strm += sizeof(len);
8:    if (len <= 512)
9:    {
10:     memcpy(buf, strm, len);
11:     process(buf);
12:     return strm + len;
13:   }
14:   else
15:   {
16:     return -1;
17:   }
18: }
```

라. 참고문헌

[1] CWE-195 Signed to Unsigned Conversion Error,
http://cwe.mitre.org/data/definitions/195.html

4. 정수를 문자로 변환(Type Mismatch: Integer to Character)

가. 정의

정수를 문자 (character)로 변환하면서 표현할 수 없는 범위의 값이 잘려 나가는 경우이다. 4byte인 정수를 1byte인 문자에 값을 저장하는 경우 표현할 수 없는 범위의 값에 대해서는 값이 삭제되어서 문자에 저장된 값이 제대로 된 값이 아닌 경우가 있다. 이러한 에러는 이후에 SW에 영향을 미치면서 코드의 오작동이 발생할 수 있다.

나. 안전한 코딩기법

- 강한 타입체크를 수행한다.

다. 예제

1 byte 크기인 char타입의 변수에 int타입의 변수/값을 지정할 경우 char타입 변수는 오버플로우 가능성이 있다. 따라서 잘못된 값으로 프로그램의 다른 부분에 영향을 미칠 가능성이 있다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  char mismatch(char bc)
4:  {
5:     return bc;
6:  }
7:  char char_type()
8:  {
9:     char bA;
10:    int  iB;
11:    iB = 24;
12:    bA = iB;
13:    f(iB);
14:    printf("int = %d char = %d\n", iB, bA);
15:    return iB;
16: }
```

충분한 크기의 정확한 타입으로 수정했다.

안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  int mismatch(int bc)
4:  {
5:     return bc;
6:  }
```

```
7:  int char_type()
8:  {
9:    int bA;
10:   int  iB;
11:   iB = 24;
12:   bA = iB;
13:   f(iB);
14:   printf("int = %d char = %d\n", iB, bA);
15:   return iB;
16:  }
```

다음의 예제는 데이터를 읽어서 \n이 나올 때까지의 한줄 당 문자 갯수를 **lineStartOffset**에 저장하고 있다. 그러나 integer 타입인 **lineOffset**과 달리 저장하는 장소인 **lineStartOffset**은 char배열 형태이므로 오버플로우 되어 엉뚱한 값이 삽입된다.

■ 안전하지 않은 코드의 예 - C

```
1:  struct DocumentInfo
2:  {
3:    char * contents;
4:    int lineCount;
5:    char lineStartOffset[1024];
6:    ...
7:  };
8:
9:  DocumentInfo * AnalysisDocument
10: {
11:   DocumentInfo * info  = (DocumentInfo *)malloc(sizeof(struct DocumentInfo));
12:
13:   char * nextCharPos  = contents;
14:   int lineCount  = 0;
15:   int lineOffset  = 1;
16:   while(*nextCharPos)
17:   {
18:     if(*nextCharPos == '\n')
19:     {
20:       lineStartOffset[lineCount]  = lineOffset;
21:       lineCount++;
22:       llineOffset  = 0;
23:     }
24:     lineOffset++;
25:   }
26: }
```

타입을 정확히 용도에 맞게 선언하고 데이터 타입의 최대 크기를 넘어서는 값이 나오면 에러로 처리하도록 안전하게 처리하였다.

■ 안전한 코드의 예 - C

```
1:  struct DocumentInfo
2:  {
3:    char * contents;
4:    int lineCount;
5:    int lineStartOffset[1024];
6:    ...
7:  };
8:
9:  DocumentInfo * AnalysisDocument
10: {
11:   DocumentInfo * info  = (DocumentInfo *)malloc(sizeof(struct DocumentInfo));
12:
13:   char * nextCharPos  = contents;
14:   int lineCount  = 0;
15:   int lineOffset  = 1;
16:   while(*nextCharPos)
17:   {
18:     if(*nextCharPos == '\n')
19:     {
20:       lineStartOffset[lineCount]  = lineOffset;
21:       lineCount++;
22:       llineOffset  = 0;
23:     }
24:     if(lineOffset >= INT_MAX)
25:     {
26:       return NULL;
27:     }
28:     lineOffset++;
29:   }
30: }
```

라. 참고문헌

[1] CWE-398 Indicator of Poor Code Quality, http://cwe.mitre.org/data/definitions/398.html

5. 스택 변수 주소 리턴(Return of Stack Variable Address)

가. 정의

함수가 스택의 주소를 반환하는 경우 그 주소가 더 이상 유효하지 않는 경우가 발생한다. 프로그램의 특정 함수가 지역 변수의 주소를 반환할 때 스택 주소가 반환된다. 이후의 동일 함수 호출은 같은 스택 주소를 다시 사용할 가능성이 높아 지역 변수의 포인터 값을 덮어쓰게 된다.

이 경우에는 예상치 않게 포인터의 값이 변경되는 문제가 발생한다. 이로 인해 유효하지 않은 포인터 주소가 참조되며, 결과적으로 프로그램이 중단되거나 증상이 나타나지 않는 경우도 있어 디버깅이 어려울 수 있다.

나. 안전한 코딩기법

- 스택 변수 주소의 리턴 스택상의 지역변수의 주소는 반환하지 않는다. 스택의 지역변수들은 함수의 소멸과 동시에 같이 제거되므로 스택상의 지역변수의 주소를 리턴값을 주게 되면 할당되지 않은 공간을 참조하게 되는 위험이 발생하게 된다.

다. 예제

스택상의 변수 **name**의 주소를 반환하지만 스택의 지역변수는 함수가 끝난 후에 소멸되므로 스택상의 지역변수의 주소를 반환하는 것은 문제가 발생한다.

■ 안전하지 않은 코드의 예 - C

```
1:  char *rpl()
2:  {
3:    char p[10];
4:    /* 로컬 버퍼의 주소를 반환함 */
5:    return p;
6:  }
7:  int main()
8:  {
9:    char *p;
10:   p = rpl();
11:   *p = '1';
12:   return 0;
13: }
```

힙 영역의 주소를 반환하거나 메모리 영역을 호출하는 함수로부터 반환값을 입력받는다.

■ 안전한 코드의 예 - C

```
1:  #include <stdlib.h>
2:  #include <string.h>
3:  char *rpl()
```

```
4:  {
5:    char p[10];
6:    int size = 10;
7:    if( size < 20 )
8:    {
9:      /* 메모리를 새롭게 할당하여, 로컬 버퍼의 내용을 복사해 넣은 후 새롭게 할당된 메모
      리 주소를 반환함 */
10:     char *buf = (char *)malloc(size);
11:     if (!buf)
12:       exit(1);
13:     memcpy(buf,p,10);
14:   }
15:   return buf;
16: }
17: int main()
18: {
19:   char *p;
20:   p = rpl();
21:   *p = '1';
22:   free(p);
23:   return 0;
24: }
```

result를 **matrix**타입으로 선언했기 때문에 return을 위해서는 **&result**를 쓸 수 밖에 없는데, 함수가 종료되면 소멸되므로 프로그램이 오작동하게 된다.

■ 안전하지 않은 코드의 예 - C

```
1:  typedef struct matrix
2:  {
3:    int num11, num12;
4:    int num21, num22;
5:  };
6:
7:  typedef struct matrix matrix;
8:
9:  matrix * addMatrix(matrix * operand1, matrix * operand2)
10: {
11:   matrix result;
12:   result.num11  = operand1->num11 + operand2->num11;
13:   result.num12  = operand1->num12 + operand2->num12;
14:   result.num21  = operand1->num21 + operand2->num21;
15:   result.num22  = operand1->num22 + operand2->num22;
16:   return &result;
17: }
```

함수가 종료되어도 보존이 가능한 힙 메모리에 데이터를 저장하고 전체 구조체의 포인터를 return하였다.

■ 안전한 코드의 예 - C

```
1:  typedef struct matrix
2:  {
3:     int num11, num12;
4:     int num21, num22;
5:  };
6:
7:  typedef struct matrix matrix;
8:
9:  matrix * addMatrix(matrix * operand1, matrix * operand2)
10: {
11:    matrix * result   = (matrix *)malloc(sizeof(matrix));
12:    result->num11   = operand1->num11 + operand2->num11;
13:    result->num12   = operand1->num12 + operand2->num12;
14:    result->num21   = operand1->num21 + operand2->num21;
15:    result->num22   = operand1->num22 + operand2->num22;
16:    return result;
17: }
```

라. 참고문헌

[1] CWE-562 Return of Stack Variable Address, http://cwe.mitre.org/data/definitions/562.html

6. 매크로의 잘못된 사용(Code Correctness: Macro Misuse)

가. 정의

매크로에 따라서는 특정한 사용 규칙을 준수해야 하는 경우가 존재한다. "매크로의 잘못된 사용" 취약점은 위와 같은 사용규칙을 위반할 때 발생한다. 매크로 중에서는 두개의 매크로가 같은 레벨의 스코프에서 같이 쌍으로 사용되어야 하는 것이 있다. 이러한 매크로를 다른 스코프에서 사용한다면 잘못된 코드가 생성된다.

나. 안전한 코딩기법

- 매크로 사용 시 공유 자원에서 동작하는 특정한 함수 패밀리는 함께 사용되어야 한다.

다. 예제

pthread_cleanup_push()와 **pthread_clenaup_pop()**은 같은 레벨의 스코프에서 짝으로 사용되어야 한다. 하나만 사용할 수 없다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <pthread.h>
4:  void routine(void *i)
5:  {
6:     int j = (*(int*)(i))++;
7:  }
8:  void helper()
9:  {
10:    int a = 0;
11:    pthread_cleanup_push (routine, ((void*)&a));
12: }
```

pthread_cleanup_push()와 **pthread_clenaup_pop()** 두개의 매크로 함수를 함께 사용하면 된다.

안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <pthread.h>
4:  void routine(void *i)
5:  {
6:     int j = (*(int*)(i))++;
7:  }
8:  void helper()
9:  {
10:    int a = 0;
```

```
11:     pthread_cleanup_push (routine, ((void*)&a));
12:     pthread_cleanup_pop (1);
13:   }
```

START_CHECK_EXECUTION_TIME과 **END_CHECK_EXECUTION_TIME**은 같은 레벨의 스코프에서 짝으로 사용되어야한다. 하나만 사용할 수 없다.

■ 안전하지 않은 코드의 예 - C

```
1:  #define START_CHECK_EXECUTION_TIME() struct timespec startExeCheckTeim,
    endExeCheckTeim; clock_gettime(10, &startExeCheckTime);
2:  #define END_CHECK_EXECUTION_TIME() clock_gettime(10, &endExeCheckTime);
    _record_execution_time(&startExeCheckTime, &endExeCheckTime);
3:
4:  int main()
5:  {
6:    START_CHECK_EXECUTION_TIME();
7:    heavyJobs();
8:
9:    return 0;
10: }
```

START_CHECK_EXECUTION_TIME과 **END_CHECK_EXECUTION_TIME**을 **heavyJobs**가 실행되기 전과 후에 각각 실행하였다.

■ 안전한 코드의 예 - C

```
1:  #define START_CHECK_EXECUTION_TIME() struct timespec startExeCheckTeim,
    endExeCheckTeim; clock_gettime(10, &startExeCheckTime);
2:  #define END_CHECK_EXECUTION_TIME() clock_gettime(10, &endExeCheckTime);
    _record_execution_time(&startExeCheckTime, &endExeCheckTime);
3:
4:  int main()
5:  {
6:    START_CHECK_EXECUTION_TIME();
7:    heavyJobs();
8:    END_CHECK_EXECUTION_TIME();
9:
10:   return 0;
11: }
```

라. 참고문헌

[1] CWE-730 OWASP Top Ten 2004 Category A9 - Denial of Service, http://cwe.mitre.org/data/definitions/730.html

7. 코드정확성: 스택 주소 해제(Code Correctness : Memory Free on Stack Variable)

가. 정의

스택 버퍼를 해제하면 예상치 않은 프로그램 동작이 발생한다.

나. 안전한 코딩기법

- 지역 변수로 배열을 선언하면 스택에 버퍼가 할당되고, 함수가 반환할 때 버퍼를 자동으로 제거하므로 명시적으로 해제하면 안 된다.

다. 예제

로컬 영역에 선언된 스택 영역에 존재하는 변수에 대해서 메모리를 해제해서는 안 된다.

안전하지 않은 코드의 예 - C

```
1:  #include stdlib.h
2:  int main()
3:  {
4:     char p[10];
5:     /* 지역변수내에 배열에 대한 메모리 해제 */
6:     free(p);
7:     return 0;
8:  }
```

로컬영역에 선언된 변수는 Call Stack이 종료되어 자동으로 메모리가 반환되면 메모리를 해제할 필요가 없다.

안전한 코드의 예 - C

```
1:  #include stdlib.h
2:  int main()
3:  {
4:     char p[10];
5:     return 0;
6:  }
```

strtok()는 별도로 memory를 allocation하지 않고 기존 메모리(여기서는 **str**)를 사용한다. **strtok()**에서 return 되는 값을 free하면 local 변수에 대한 메모리 해제를 시도하게 된다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <string.h>
3:
4:  int main ()
```

```
5:  {
6:     char str[] ="- This, a sample string.";
7:     char * pch;
8:     printf ("Splitting string \"%s\" into tokens:\n",str);
9:     pch = strtok (str," ,.-");
10:    free(pch);
11:    while (pch != NULL)
12:    {
13:       printf ("%s\n",pch);
14:       pch = strtok (NULL, " ,.-");
15:       free(pch);
16:    }
17:    return 0;
18: }
```

로컬영역에 선언된 변수는 자동으로 메모리가 반환되므로 메모리를 해제할 필요가 없다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <string.h>
3:
4:  int main ()
5:  {
6:     char str[] ="- This, a sample string.";
7:     char * pch;
8:     printf ("Splitting string \"%s\" into tokens:\n",str);
9:     pch = strtok (str," ,.-");
10:    while (pch != NULL)
11:    {
12:       printf ("%s\n",pch);
13:       pch = strtok (NULL, " ,.-");
14:    }
15:    return 0;
16: }
```

라. 참고문헌

[1] CWE-730 OWASP Top Ten 2004 Category A9 - Denial of Service, http://cwe.mitre.org/data/definitions/730.html

8. 코드 정확성: 스레드 조기 종료(Code Correctness: Premature thread Termination)

가. 정의

부모 스레드가 자식 스레드 보다 먼저 종료되는 경우, 자식 스레드에 분배된 자원 회수를 못 하는 경우가 발생한다. 자식 스레드를 생성할 경우 내부적으로 스레드 제어를 위한 자료구조를 생성하게 되는데 이러한 자료 구조는 부모 스레드가 pthread_join()과 같은 종류의 함수를 통해 자식 스레드가 종료되기를 기다렸다가 스레드 제어용 자료구조를 반환하게 된다.

만일, 자식 스레드의 종료를 기다리지 않고 부모 스레드가 종료하는 경우, 자식 스레드에 할당된 자료구조가 회수되지 않아 메모리 누수 현상을 발생시키게 된다. PTHREAD에서는 부모 스레드가 pthread_detach()를 통해서 자식 스레드를 분리(detach)하던가 생성 시에 스레드의 attribute를 "분리" 상태로 생성하게 하여 종료 시 스스로 자료구조를 반환하게 하여야 한다.

나. 안전한 코딩기법

- 부모 스레드가 join을 통해 자식을 기다리거나 부모 스레드가 먼저 종료되어야 하는 경우 detach를 통해서 자식 스레드를 분리한다.

다. 예제

스레드 생성 후 **join**으로 기다리거나 **detach**를 이용하여 스스로 자원반납을 하도록 해야 한다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <pthread.h>
4:  int th_worker(void *data)
5:  {
6:     int a = *(int *)data;
7:     return a + 100;
8:  }
9:   int run_thread1(void)
10:  {
11:     pthread_t th;
12:     int status, a = 1;
13:     if (pthread_create(&th, NULL, (void*(*)(void*))th_worker, (void *)&a) < 0)
14:     {
15:        perror("thread create error: ");
16:        exit(0);
17:     }
18:     printf("Return without waiting for child thread\n");
19:     return 0;
20:  }
```

pthread_join()으로 자식 스레드를 기다려서 자원을 반납하고 반환하도록 수정하였다. **pthread_create()** 수행 직후 **pthread_join()**를 호출해서 자식 스레드가 종료될 때 까지 대기 후 부모 스레드가 종료될 수 있도록 하였다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <pthread.h>
4:  int th_worker(void *data)
5:  {
6:     int a = *(int *)data;
7:     return a + 100;
8:  }
9:  int run_thread1(void)
10: {
11:    pthread_t th;
12:    int status, a = 1;
13:    if (pthread_create(&th, NULL, (void*(*)(void*))th_worker, (void *)&a) < 0)
14:    {
15:       perror("thread create error: ");
16:       exit(0);
17:    }
18:    printf("Return after waiting for child thread\n");
19:    pthread_join(th, (void**)&status);
20:    return 0;
21: }
```

스레드 생성 후 **join**으로 기다리거나 **detach를 이용하여** 스스로 자원반납을 하도록 해야 한다.

■ 안전하지 않은 코드의 예 - C

```
1:  void * _MakeEstimate(void *parm)
2:  {
3:     char * stockName = param;
4:     // 부모 thread와 무관한 동작을 수행
5:     Estimate * estimate = MakeEstimate(stockName);
6:     RecordEstimate(estimate);
7:     return NULL;
8:  }
9:
10: int main(int argc, char **argv)
11: {
12:    pthread_attr_t  attr;
13:    pthread_t  thread;
14:    int rc=0;
15:    int STOCK_COUNT = 3452;
16:    char * stockNames[STOCK_COUNT]    = { ... };
```

```
17:     int stockIdx;
18:
19:     for(stockIdx = 0; stockIdx < STOCK_COUNT; stockIdx++)
20:     {
21:             pthread_create(&thread, NULL, _MakeEstimate, stockNames[stockIdx]);
22:     }
23:     ...
24:     return 0;
25: }
```

스레드를 **detach**상태로 생성하고, 사용한 후 **pthread_attr_destroy(&attr);**를 호출하여 자원을 반환하였다.

■ 안전한 코드의 예 - C

```
1:  void * _MakeEstimate(void *parm)
2:  {
3:      char * stockName = param;
4:      // 부모 thread와 무관한 동작을 수행
5:      Estimate * estimate = MakeEstimate(stockName);
6:      RecordEstimate(estimate);
7:      return NULL;
8:  }
9:
10: int main(int argc, char **argv)
11: {
12:     pthread_attr_t          attr;
13:     pthread_t               thread;
14:     int                     rc=0;
15:     int STOCK_COUNT = 3452;
16:     char * stockNames[STOCK_COUNT]    = { ... };
17:     int stockIdx;
18:
19:     pthread_attr_init(&attr);
20:     pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED); // detach 상태로
    thread를 생성한다.
21:     for(stockIdx = 0; stockIdx < STOCK_COUNT; stockIdx++){
22:             pthread_create(&thread, &attr, _MakeEstimate, stockNames[stockIdx]);
23:     }
24:     pthread_attr_destroy(&attr);
25:     ...
26:     return 0;
27: }
```

라. 참고문헌

[1] CWE-730 OWASP Top Ten 2004 Category A9 - Denial of Service, http://cwe.mitre.org/data/definitions/730.html

9. 무한 자원 할당(Allocation of Resources Without Limits or Throttling)

가. 정의

프로그램이 자원을 사용 후 해제하지 않거나, 한 사용자당 서비스할 수 있는 자원의 양을 제한하지 않고, 서비스 요청마다 요구하는 자원을 할당한다.

나. 안전한 코딩기법

- 프로그램에서 자원을 오픈하여 사용하고 난 후, 반드시 자원을 해제한다.
- 사용자가 사용할 수 있는 자원의 사이즈를 제한한다.

※ 사용자가 접근할 수 있는 자원의 양을 제한한다. 특히 제한된 자원을 효율적으로 사용하기 위해서 Pool(Thread Pool, Connection Pool 등)을 사용한다.

다. 예제

인자값이 메시지 길이와 메시지 **Body**의 이차원 배열로 된 경우, 메시지 길이에 대한 제한이 없기 때문에 메시지 **Body**에 아주 큰 데이터가 올 경우, 시스템 메모리를 고갈시킬 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  int processMessage(char **message)
2:  {
3:    char *body;
4:    int length = getMessageLength(message[0]);
5:    if (length > 0)
6:    {
7:      body = &message[1][0];
8:      processMessageBody(body);
9:      ......
10:   }
11:   else {......}
12: }
```

길이체크에 음수가 발생하지 않도록 unsigned int형을 사용하고, 메시지 처리를 위한 최대치를 설정하여 메모리 사용량을 제한한다.

■ 안전한 코드의 예 - C

```
1:  int processMessage(char **message)
2:  {
3:    char *body;
4:    unsigned int length = getMessageLength(message[0]);
5:    // 자원 고갈을 방지위해 사용자의 자원을 제한한다.
6:    if (length > 0 && length < MAX_LENGTH)
7:    {
8:      body = &message[1][0];
```

```
9:          processMessageBody(body);
10:         ......
11:     }
12:     else {......}
13: }
```

스레드를 생성할 때마다 새로운 소켓자원을 할당하므로, 무한정 스레드수가 무한정 증가할 경우 시스템 자원을 고갈시킬 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  int main(int argc, char * argv[])
2:  {
3:      int listenSocket, connectionSocket;
4:      struct sockaddr_in serv_addr, cli_addr;
5:      socklen_t clilen;
6:      pthread_t thread;
7:
8:      listenSocket = socket(AF_INET_SOCK_STREAM, 0);
9:      ...
10:     while(TRUE)
11:     {
12:             connectionSocket= accpet(listenSocket, (struct sockaddr *) &cli_addr, &clilen);
13:             pthread_create(&thread, NULL, communication, &connectionSocket);
14:             connectionSocket = -1;
15:             ...
16:     }
17:     close(listenSocket);
18: }
19: void * communicate(void * argument)
20: {
21:     int socket   = *((int *)argument)
22:     ...
23:     close(socket);
24: }
```

사용이 끝난 메모리를 pool에 넣고, 새로 스레드가 생성되는 경우 pool에서 먼저 사용하면 메모리 낭비를 줄일 수 있다.

■ 안전한 코드의 예 - C

```
1:  ThreadPool threadPool;
2:
3:  int main(int argc, char * argv[])
4:  {
5:      int listenSocket, connectionSocket;
6:      struct sockaddr_in serv_addr, cli_addr;
```

```
7:      socklen_t clilen;
8:
9:      listenSocket = socket(AF_INET_SOCK_STREAM, 0);
10:     ...
11:     while(TRUE)
12:     {
13:        connectionSocket = accpet(listenSocket, (struct sockaddr *) &cli_addr, &clilen);
14:        // Thread pool에서 thread를 할당한다.
15:        if(threadPool.alloc(communication, &connectionSocket) == TRUE)
16:        {
17:            connectionSocket          = -1;
18:        }
19:        else
20:        {
21:            close(connectionSocket);
22:        }
23:        ...
24:     }
25:     close(listenSocket);
26: }
27: void * communicate(pthread_t thread, void * argument)
28: {
29:     pthread_t thread;
30:     int socket    = *((int *)argument)
31:     ...
32:     close(socket);
33:     // Thread pool에서 thread를 해지한다.
34:     threadPool.free(thread);
35: }
```

라. 참고 문헌

[1] CWE-400 Uncontrolled Resource Consumption(Resource Exhaustion), http://cwe.mitre.org/data/definitions/400.html

[2] CWE-744 CERT C Secure Coding Section 10 - Environment(ENV), http://cwe.mitre.org/data/definitions/744.html

[3] CWE-789 Uncontrolled Memory Allocation, http://cwe.mitre.org/data/definitions/789.html

[4] CWE-770 Allocation of Resources Without Limits or Throttling, http://cwe.mitre.org/data/definitions/770.html

[5] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 17, "Protecting Against Denial of Service Attacks" Page 517. 2nd Edition. Microsoft. 2002

[6] J. Antunes, N. Ferreira Neves and P. Verissimo. "Detection and Prediction of Resource-Exhaustion Vulnerabilities". Proceedings of the IEEE International

제6절 캡슐화

소프트웨어가 중요한 데이터나 기능성을 불충분하게 캡슐화 하는 경우, 인가된 데이터와 인가되지 않은 데이터를 구분하지 못하게 되어 허용되지 않은 사용자들 간의 데이터 누출이 가능하다. 캡슐화는 단순히 일반 소프트웨어 개발 방법상의 상세한 구현 내용을 숨길뿐만 아니라 소프트웨어 보안 측면의 좀 더 넓은 의미로 사용된다.

1. 제거되지 않고 남은 디버그 코드(Leftover Debug Code)

가. 정의

디버깅 목적으로 삽입된 코드는 개발이 완료되면 제거해야 한다. 디버그 코드는 설정 등의 민감한 정보를 내포하거나 시스템 제어와 관련된 부분을 내포하고 있는 경우도 있다. 만일 남겨진 채로 배포될 경우 공격자가 식별 과정을 우회하거나 의도하지 않은 정보와 제어 정보가 노출될 수 있다.

<그림 2-11> 제거되지 않고 남은 디버그 코드

나. 안전한 코딩기법

- 개발단계에서 디버깅 목적으로 사용한 프로그램에서 main으로 시작되는 프로그램과 콘솔에 출력되는 문장들을 모두 제거한다.

다. 예제

디버깅 목적으로 사용한 **main()**으로 시작되는 코드로 **println()** 함수를 통해서 정보가 콘솔에 출력된다.

■ 안전하지 않은 코드의 예 - C

```
1:  int main(int argc,char** argv)
2:  {
3:      println("Debug Info...");
4:  }
```

디버깅 목적으로 사용한 **main()**으로 시작하는 코드는 삭제한다.

■ 안전한 코드의 예 - C

```
1:  // main()으로 시작하는 디버그 코드는 삭제한다.
2:  .....
```

디버깅 목적으로 사용한 **main()**으로 시작되는 코드로 **puts()** 함수를 통해서 정보가 콘솔에 출력된다.

■ 안전하지 않은 코드의 예 - C

```
1:  int main()
2:  {
3:      puts("Debug message");
4:  }
```

#ifdef 분기문으로 **DEBUG_ENABLE**을 미리 설정해두고 디버그를 할 때에만 활성화시켜 **puts**로 실제적인 오류내용을 콘솔에 표시해준다. 평상시에는 아무런 메시지도 콘솔에 출력되지 않도록 프로그래밍 되어 있다.

■ 안전한 코드의 예 - C

```
1:  #define DEBUG_ENABLE
2:  #ifdef DEBUG_ENABLE
3:      #define DEBUG(message)    { puts( message ); }
4:  #else
5:      #define DEBUG(message)
6:  #endif
7:
8:  int main()
9:  {
10:     ...
11:     DEBUG("Debug message");
12:     ...
13: }
```

라. 참고 문헌

[1] CWE-489 Leftover Debug Code, http://cwe.mitre.org/data/definitions/489.html

[2] M. Howard and D. LeBlanc. "Writing Secure Code". Page 505. 2nd Edition. Microsoft. 2002

2. 시스템 데이터 정보노출(Exposure of System to an Unauthorized Control Sphere)

가. 정의

시스템·관리자DB 정보 등 시스템의 내부 데이터가 공개되면, 이를 통해 공격자에게 아이디어를 제공하는 등 공격의 빌미가 된다.

<그림 2-12> 시스템 데이터 정보 노출

나. 안전한 코딩기법

- 일부 개발자들은 예외상황이 발생할 경우 시스템 메시지 등의 정보를 화면에 출력하도록 하는 경우가 많다. 예외상황이 발생했을 때 시스템의 내부 정보가 화면에 출력하지 않도록 개발한다.

다. 예제

예외 발생시 getenv()를 통해 오류와 관련된 시스템 에러정보 등 민감한 정보가 유출될 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  #include "std_testcase.h"
2:
3:  #include <stdio.h>
4:
5:  #ifndef OMITBAD
6:
7:  void
    KRD_603_Exposure_of_System_Data_to_an_Unauthorized_Control_Sphere__w32_sprintf_0101_
    bad()
8:  {
9:      {
10:
11:     {
12:         char* path = getenv("PATH");
13:         /* FLAW */
```

```
14:             sprintf(stderr, "cannot find exe on path %s\n", path);
15:         }
16:
17:         }
18:     }
19:
20:     #endif /* OMITBAD */
21:
22:
23:     /* Below is the main(). It is only used when building this testcase on
24:        its own for testing or for building a binary to use in testing binary
25:        analysis tools. It is not used when compiling all the testcases as one
26:        application, which is how source code analysis tools are tested. */
27:
28:     #ifdef INCLUDEMAIN
29:
30:     int main(int argc, char * argv[])
31:     {
32:         /* seed randomness */
33:         srand( (unsigned)time(NULL) );
34:     #ifndef OMITBAD
35:         printLine("Calling bad()...");
36:
        KRD_603_Exposure_of_System_Data_to_an_Unauthorized_Control_Sphere__w32_sprintf_0101_
        bad();
37:         printLine("Finished bad()");
38:     #endif /* OMITBAD */
39:         return 0;
40:     }
41:
42:     #endif
```

가급적이면 공격의 빌미가 될 수 있는 오류와 관련된 상세한 정보는 최종 사용자에게 출력되지 않도록 개발한다.

■ 안전한 코드의 예 - C

```
43:     #include "std_testcase.h"
44:     #include <stdio.h>
45:     #ifndef OMITGOOD
46:
47:     static void good1()
48:     {
49:         {
50:
51:         {
```

```
52:             char* path = getenv("PATH");
53:             /* FIX */
54:             sprintf(stderr, "cannot find exe on path \n");
55:         }
56:
57:         }
58:     }
59:
60:     void
    KRD_603_Exposure_of_System_Data_to_an_Unauthorized_Control_Sphere__w32_sprintf_0101_
    good()
61:     {
62:         good1();
63:     }
64:
65:     #endif /* OMITGOOD */
66:
67:     /* Below is the main(). It is only used when building this testcase on
68:        its own for testing or for building a binary to use in testing binary
69:        analysis tools. It is not used when compiling all the testcases as one
70:        application, which is how source code analysis tools are tested. */
71:
72:     #ifdef INCLUDEMAIN
73:
74:     int main(int argc, char * argv[])
75:     {
76:         /* seed randomness */
77:         srand( (unsigned)time(NULL) );
78:     #ifndef OMITGOOD
79:         printLine("Calling good()...");
80:
    KRD_603_Exposure_of_System_Data_to_an_Unauthorized_Control_Sphere__w32_sprintf_0101_
    good();
81:         printLine("Finished good()");
82:     #endif /* OMITGOOD */
83:         return 0;
84:     }
85:
86:     #endif
 1:
```

라. 참고문헌

[1] CWE-497 Exposure of System Data to an Unauthorized Control Sphere, http://cwe.mitre.org/data/definitions/497.html

제7절 API 오용

API(Application Programming Interface)는 운영체제와 응용프로그램간의 통신에 사용되는 언어나 메시지 형식 또는 규약으로, 프로그램 개발시 개발 편리성 및 효율성을 제공하는 이점이 있다. 그러나 API 오용 및 취약점이 알려진 API의 사용은 개발 효율성 및 유지보수성의 저하 및 보안상의 심각한 위협요인이 될 수 있다.

1. DNS lookup에 의존한 보안결정(Reliance on DNS Lookups in a Security Decision)

가. 정의

공격자가 DNS 엔트리를 속일 수 있으므로 도메인명에 의존하여 보안결정(인증 및 접근통제 등)을 하지 않아야 한다. 만약 로컬 DNS 서버의 캐시가 공격자에 의해 조작된 상황이라면 사용자와 특정 서버간의 네트워크 트래픽이 공격자를 경유하도록 설정할 가능성도 있다. 또한 공격자가 마치 동일 도메인에 속한 서버인 것처럼 위장할 수도 있다.

나. 안전한 코딩기법

- 보안결정에서 도메인명을 이용한 DNS lookup에 의존하지 않도록 한다.
- DNS 이름 검색 함수를 사용한 후 조건문에서 인증여부를 수행하는 것보다 IP 주소를 이용하는 것이 DNS 이름을 직접 사용하는 것 보다는 안전하다.

다. 예제

다음의 예제는 계약서를 지사로 보내는 프로그램으로써 DNS명(www.trust.com/www.faith.com)을 신뢰할 수 있는 사이트에 등록하고, 보내려는 목표 사이트가 등록된 사이트의 이름 안에 있으면 신뢰할 수 있는 사이트로 간주하여 계약 내용을 전송한다. 그러나 이러한 DNS 명이 신뢰할 수 있는 사이트에 등록되어 있을지라도 공격자는 DNS 캐쉬 내 IP 정보를 공격자의 IP 정보로 조작함으로써 해당 요청에 대한 신뢰성 결정을 우회할 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:
3:  int main(void)
4:  {
5:      const char * COMMAND_PARAM  = "command";
6:      const char * SEND_CONTRACT = "send_contract";
7:
8:      set<char *>  trustedSites;
9:      trustedSites.insert("www.trust.com");
10:     trustedSites.insert("www.faith.com");
11:
```

```
12:     char * queryStr;
13:     queryStr = getenv("QUERY_STRING");
14:
15:     if (queryStr == NULL)
16:     {
17:       // Error 처리
18:       ...
19:     }
20:
21:     char * command = GetParameter(queryStr, COMMAND_PARAM);
22:     if (!strcmp(command, SEND_CONTRACT))
23:     {
24:       const char * TARGET_SITE_PARM = "target_site";
25:       const char * targetSite = GetParameter(queryStr, TARGET_SITE_PARM);
26:
27:       hostent * record  = gethostbyname();
28:       in_addr * address = (in_addr * )record->h_addr;
29:       char * ip_address = inet_ntoa(* address);
30:
31:       if(trustedSites.count(targetSite) > 0)
32:       {
33:         SendContract(ip_address);
34:       }
35:       free(targetSite);
36:     }
37:     return EXIT_SUCCESS;
38: }
```

다음의 예제와 같이 DNS 이름 대신 IP 주소를 사용하는 것이 안전하다. IP 주소 역시 DNS 서버를 가장하여 위조 가능하지만, 호스트 명을 사용하는 경우보다 안전성을 확보할 수 있다.

■ 안전한 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:
3:  int main(void)
4:  {
5:    const char * COMMAND_PARAM  = "command";
6:    const char * SEND_CONTRACT = "send_contract";
7:    set<char *>  trustedSiteIPs;
8:    trustedSiteIPs.insert("232.234.89.52");
9:    trustedSiteIPs.insert("87.123.56.92");
10:   char * queryStr;
11:   queryStr = getenv("QUERY_STRING");
12:   if (queryStr == NULL)
```

```
13:     {
14:       // Error 처리
15:       ...
16:     }
17:     char * command = GetParameter(queryStr, COMMAND_PARAM);
18:     if (!strcmp(command, SEND_CONTRACT))
19:     {
20:       const char * TARGET_SITE_PARM = "target_site";
21:       const char * targetSiteIp = GetParameter(queryStr, TARGET_SITE_PARM);
22:       if(trustedSiteIPs.count(targetSiteIp) > 0)
23:       {
24:         SendContract(targetSiteIp);
25:       }
26:       free(targetSite);
27:     }
28:     return EXIT_SUCCESS;
29:   }
```

라. 참고문헌

[1] CWE-247 Reliance on DNS Lookups in a Security Decision, http://cwe.mitre.org/data/definitions/247.html

[2] CWE-807 Reliance on Untrusted Inputs in a Security Decision, http://cwe.mitre.org/data/definitions/807.html

[3] 2011 SANS Top 25 - RANK 10 (CWE-807), http://cwe.mitre.org/top25/

2. 위험하다고 알려진 함수 사용(Use of Inherently Dangerous Function)

가. 정의

이미 위험하다고 알려진 라이브러리 함수를 사용하는 것은 바람직하지 않다. 특정 라이브러리 함수들은 보안취약점을 전혀 고려하지 않고 개발되어 사용할 경우 취약점이 될 수 있다. 예를 들면, gets() 함수는 입력 크기 제한사항을 점검하지 않기 때문에 입력 버퍼가 초과될 수 있다. 즉, 이러한 함수들은 사용 자체만으로 보안취약점이 될 수 있다.

나. 안전한 코딩기법

- 잘 알려진 취약점으로 인해 이미 위험하다고 알려진 함수들의 사용을 금지한다. (예: gets 함수)

다. 예제

buf는 **BUFSIZE** 크기의 배열인데 **gets()** 함수는 그 크기와 상관없이 입력값을 **buf**에 저장하기 때문에 버퍼 오버플로우를 유발할 수 있다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #define BUFSIZE 100
4:  void requestString()
5:  {
6:    char buf[BUFSIZE];
7:    gets(buf);
8:  }
```

버퍼 오버플로우 공격에 취약한 **gets()** 함수 대신 **fgets()** 함수를 사용한다.

안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #define BUFSIZE 100
4:  void requestString()
5:  {
6:    char buf[BUFSIZE];
7:    /* buf 할당된 메모리 이내에서만 문자들을 입력 받음*/
8:    fgets(buf, BUFSIZE, stdin);
9:  }
```

vfork()는 자식 프로세스가 부모 프로세스의 공간을 잠시 빌려쓰기 때문에 부모 프로세스의 의도를 왜곡시키거나 의도적인 오작동을 야기시킬 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  char *filename = /* something */;
2:
3:  pid_t pid = vfork();
4:  if (pid == 0) /* child */
5:  {
6:      if (execve(filename, NULL, NULL) == -1)
7:              {
8:              /* Handle error */
9:      }
10:     _exit(1); /* in case execve() fails */
11: }
```

부모 프로세스와 별도의 공간을 사용하는 **fork()**함수를 사용하는 것이 안전하다.

■ 안전한 코드의 예 - C

```
1:  char *filename = /* something */;
2:
3:  pid_t pid = fork();
4:  if (pid == 0) /* child */
5:  {
6:      if (execve(filename, NULL, NULL) == -1)
7:              {
8:              /* Handle error */
9:      }
10:     _exit(1); /* in case execve() fails */
11: }
```

라. 참고문헌

[1] CWE-242 Use of Inherently Dangerous Function, http://cwe.mitre.org/data/definitions/242.html

[2] CWE-676 Use of Potentially Dangerous Function, http://cwe.mitre.org/data/definitions/676.html

3. 작업 디렉터리 변경 없는 chroot Jail 생성(Creation of chroot Jail Without Change Working Directory)

가. 정의

프로그램에서 chroot 함수를 사용하여 접근 가능한 디렉터리를 제한할 때 작업 디렉터리를 변경하지 않는 경우 공격자가 제한된 디렉터리 외부에 접근할 수 있다. chroot는 프로그램이 특정 디렉터리를 파일 시스템의 루트 디렉터리로 인지하도록 하여 제한된 디렉터리 내에서 파일 접근이 가능하도록 하는 함수이다. 이때 프로그램의 작업 디렉터리를 변경해야 하며 만약 변경하지 않을 경우 상대 경로를 이용하여 다른 디렉터리에 접근할 수 있다.

나. 안전한 코딩기법

- chroot() 이후에 chdir("/")를 사용하여 현재 디렉터리를 새로운 루트 디렉터리의 하위 경로로 변경되도록 한다.

다. 예제

다음의 예제는 ftp 서버 프로그램인데, ftp 서버 입장에서 특정 디렉터리인 **/var/ftproot**를 루트 디렉터리로 설정하여 클라이언트가 해당 하위 경로만 접근 가능하도록 하였다. 하지만 작업 디렉터리를 변경하지 않았기 때문에 클라이언트가 **../../etc/passwd** 경로를 이용하여 패스워드 파일에 접근할 수 있다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <unistd.h>
4:  void changeRoot(FILE *network)
5:  {
6:     FILE *localfile;
7:     char filename[80], buf[80];
8:     int len;
9:
10:    chroot("/var/ftproot");
11:
12:    fgets(filename, sizeof(filename), network);
13:    localfile = fopen(filename, "r");
14:    while ((len = fread(buf, 1, sizeof(buf), localfile)) != EOF)
15:    {
16:       fwrite(buf, 1, sizeof(buf), network);
17:    }
18:    fclose(localfile);
19: }
```

chroot() 함수 사용 이후에 **chdir("/")**를 사용하여 현재 디렉터리를 새로운 루트 디렉터리 하위 경로로 변경되도록 한다.

■ 안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <unistd.h>
4:  void changeRoot(FILE *network)
5:  {
6:    FILE *localfile;
7:    char filename[80], buf[80];
8:    int len;
9:
10:   chroot("/var/ftproot");
11:
12:   /* 현재 디렉터리를 새로운 루트 디렉터리 밑으로 변경 */
13:   chdir("/");
14:
15:   fgets(filename, sizeof(filename), network);
16:   localfile = fopen(filename, "r");
17:   while ((len = fread(buf, 1, sizeof(buf), localfile)) != EOF)
18:   {
19:     fwrite(buf, 1, sizeof(buf), network);
20:   }
21:   fclose(localfile);
22: }
```

다음의 예제는 파일을 다루기 직전 특정 디렉터리인 **userHomePath**를 루트 디렉터리로 설정하여 클라이언트가 해당 하위 경로만 접근 가능하도록 하였다. 하지만 작업 디렉터리를 변경하지 않았기 때문에 클라이언트가 **../../etc/passwd** 경로를 사용하여 패스워드 파일에 접근할 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  int main(void)
2:  {
3:    const char * COMMAND_PARAM = "command";
4:    const char * GET_USER_INFO_CMD = "get_user_info";
5:
6:    char * queryStr;
7:    queryStr = getenv("QUERY_STRING");
8:    if (queryStr == NULL)
9:    {
10:     // Error 처리
```

```
11:         ...
12:     }
13:     char * command = GetParameter(queryStr, COMMAND_PARAM);
14:     if (!strcmp(command, GET_USER_INFO_CMD))
15:     {
16:         const char * READ_DOCUMENT   = "read_document";
17:         const char * USER_ID_PARAM   = "user_id";
18:         const char * FILE_NAME_PARAM = "file_name";
19:         const char * userId = GetParameter(queryStr, USER_ID_PARAM);
20:         const char * fileName = GetParameter(queryStr, FILE_NAME_PARAM);
21:
22:         char * userHomePath = getUserHomeDir(userId);
23:         chroot(userHomePath);
24:
25:         // File Process
26:
27:         free(fileName);
28:         free(userId);
29:     }
30:     return EXIT_SUCCESS;
31: }
```

chroot() 함수 사용 이후에 **chdir("/")**를 사용하여 현재 디렉터리를 새로운 루트 디렉터리의 하위 경로로 변경되도록 한다.

■ 안전한 코드의 예 - C

```
1:  int main(void)
2:  {
3:      const char * COMMAND_PARAM = "command";
4:      const char * GET_USER_INFO_CMD = "get_user_info";
5:
6:      char * queryStr;
7:      queryStr = getenv("QUERY_STRING");
8:      if (queryStr == NULL)
9:      {
10:         // Error 처리
11:         ...
12:     }
13:     char * command = GetParameter(queryStr, COMMAND_PARAM);
14:     if (!strcmp(command, GET_USER_INFO_CMD))
15:     {
16:         const char * READ_DOCUMENT   = "read_document";
17:         const char * USER_ID_PARAM   = "user_id";
18:         const char * FILE_NAME_PARAM = "file_name";
19:         const char * userId = GetParameter(queryStr, USER_ID_PARAM);
```

```
20:         const char * fileName = GetParameter(queryStr, FILE_NAME_PARAM);
21:
22:         char * userHomePath = getUserHomeDir(userId);
23:         chroot(userHomePath);
24:         chdir("/");
25:
26:         // File Process
27:
28:         free(fileName);
29:         free(userId);
30:     }
31:     return EXIT_SUCCESS;
32: }
```

라. 참고문헌

[1] CWE-243 Creation of chroot Jail Without Changing Working Directory, http://cwe.mitre.org/data/definitions/243.html

4. 오용: 문자열 관리(Often Misused: String Management)

가. 정의

버퍼 오버플로우가 발생할 수 있는 문자열 연산 함수를 호출하면 위험하다. 윈도우에서 제공하는 멀티 바이트 문자열 함수들은 모두 버퍼 오버플로우를 유발할 수 있다.

나. 안전한 코딩기법

- 마이크로소프트 윈도우에서 제공하는 _mbsXXX()류의 멀티 바이트 문자열 함수를 사용하지 않는다.

다. 예제

마이크로소프트 윈도우에서는 멀티 바이트 문자열을 위해 _mbsXXX()류의 함수들을 제공한다. 이 때 멀티 바이트 형식에 맞지 않는 문자열을 전달하게 되면 버퍼 오버플로우가 발생할 수 있다.

안전하지 않은 코드의 예 - C

```
1:  #include <mbstring.h>
2:  void changeString(char *str1, char *str2)
3:  {
4:    _mbscpy(str1, str2);
5:  }
```

마이크로소프트 윈도우에서는 멀티 바이트 문자열을 위해 제공하는 안전한 함수인 **_mbscpy_s()**를 사용한다.

안전한 코드의 예 - C

```
1:  #include <mbstring.h>
2:  void changeString(char *str1, char *str2, unsigned size)
3:  {
4:    _mbscpy_s(str1, size, str2);
5:  }
```

위험한 함수로 분류된 **_mbscat**을 사용하고 있다.

안전하지 않은 코드의 예 - C

```
1:  const char * GetParameter(const char * queryString, const char * key);
2:  int main(void)
3:  {
4:    const int MAX_NAME_LENGTH = 32;
5:    const char * COMMAND_PARAM = "command";
6:    const char * GET_USER_INFO_CMD = "get_user_info";
7:
```

```
8:    char * queryStr;
9:    queryStr = getenv("QUERY_STRING");
10:   if (queryStr == NULL)
11:   {
12:    // Error 처리
13:    ...
14:   }
15:   char * command = GetParameter(queryStr, COMMAND_PARAM);
16:   if (!strcmp(command, GET_USER_INFO_CMD))
17:   {
18:    const char * USER_ID_PARAM = "user_id";
19:    const char * PASSWORD_PARAM = "password";
20:    const int MAX_PASSWORD_LENGTH = 16;
21:    const char * userId = GetParameter(queryStr, USER_ID_PARAM);
22:    const char * password = GetParameter(queryStr, PASSWORD_PARAM);
23:
24:    const char firstName[MAX_NAME_LENGTH], lastName[MAX_NAME_LENGTH];
25:    GetUserName(firstName, lastName);
26:    _mbscat(firstName, lastName);
27:    free(userId);
28:    free(password);
29:    ...
30:   }
31:   ...
32:   return EXIT_SUCCESS;
33:  }
```

안전한 함수인 **_mbscat_s** 를 사용한다.

■ 안전한 코드의 예 - C

```
1:   const char * GetParameter(const char * queryString, const char * key);
2:   int main(void)
3:   {
4:    const int MAX_NAME_LENGTH = 32;
5:    const char * COMMAND_PARAM = "command";
6:    const char * GET_USER_INFO_CMD = "get_user_info";
7:
8:    char * queryStr;
9:    queryStr = getenv("QUERY_STRING");
10:   if (queryStr == NULL)
11:   {
12:    // Error 처리
13:    ...
14:   }
15:   char * command = GetParameter(queryStr, COMMAND_PARAM);
```

```
16:    if (!strcmp(command, GET_USER_INFO_CMD))
17:    {
18:      const char * USER_ID_PARAM = "user_id";
19:      const char * PASSWORD_PARAM = "password";
20:      const int MAX_PASSWORD_LENGTH = 16;
21:      const char * userId = GetParameter(queryStr, USER_ID_PARAM);
22:      const char * password = GetParameter(queryStr, PASSWORD_PARAM);
23:
24:      const char firstName[MAX_NAME_LENGTH], lastName[MAX_NAME_LENGTH];
25:      GetUserName(firstName, lastName);
26:      _mbscat_s(firstName, lastName);
27:      free(userId);
28:      free(password);
29:    }
30:    return EXIT_SUCCESS;
31:  }
```

라. 참고문헌

[1] CWE-251 Often Misused: String Management,
http://cwe.mitre.org/data/definitions/251.html

[2] CWE-176 Improper Handling of Unicode Encoding,
http://cwe.mitre.org/data/definitions/176.html

5. 다중 스레드 프로그램에서 getlogin() 사용(Use of getlogin() in Multithreaded Application)

가. 정의

사용자 계정은 다중 스레드 환경에서 잠금장치를 하지 않고 획득할 수 있다. 이러한 경우 다른 스레드가 사용 중인 사용자 계정을 읽어올 수 있다. getlogin() 함수는 호출된 스레드와 연관된 사용자 이름을 리턴하는 함수이다. 이를 다중 스레드 환경에서 사용하면 리턴받은 사용자 이름값이 다른 스레드에 의해 변경될 수 있어 부정확한 값을 얻을 수 있다.

나. 안전한 코딩기법

- getlogin() 함수 대신 안전한 getlogin_r() 함수를 사용한다.

다. 예제

getlogin() 함수로 계정 정보를 획득하여 권한을 부여했다. 다중 스레드 환경에서 **getlogin()**은 다른 스레드에 의해서 반환 값이 변경될 수 있다. 따라서 이 함수의 반환 값을 근거로 권한 부여를 하는 것은 보안상 위험할 수 있다.

안전하지 않은 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <unistd.h>
4:  #include <sys/types.h>
5:  #include <pwd.h>
6:  int isTrustedGroup(int);
7:  int loginproc()
8:  {
9:    struct passwd *pwd = getpwnam(getlogin());
10:  if (isTrustedGroup(pwd->pw_gid))
11:    return 1;  // allow
12:  else
13:    return 0;  // deny
14: }
```

다중 스레드 환경에서 안전한 **getlogin_r()** 함수로 계정 정보를 획득하여 권한을 부여 한다.

안전한 코드의 예 - C

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:  #include <unistd.h>
4:  #include <sys/types.h>
5:  #include <pwd.h>
6:  #define MAX 100
```

```
7:  int isTrustedGroup(int);
8:  int loginproc()
9:  {
10:   char id[MAX];
11:   struct passwd *pwd;
12:   if (getlogin_r(id, MAX) != 0)
13:     return 0;
14:   pwd = getpwnam(id);
15:   if (isTrustedGroup(pwd->pw_gid))
16:     return 1;  // allow
17:   else
18:     return 0;  // deny
19: }
```

getlogin() 함수로 계정 정보를 획득하여 권한을 부여했다. 다중 스레드 환경에서 **getlogin()**은 다른 스레드에 의해서 반환 값이 변경될 가능성이 존재하기 때문에 이 함수의 반환 값을 근거로 권한 부여를 하는 것은 보안상 위험할 수 있다.

■ 안전하지 않은 코드의 예 - C

```
1:  void downloadVipAccountInfos()
2:  {
3:   char * userId = getlogin();
4:   UserList * list = getAuthentifiedUser();
5:   if(isContainedInList(list, userId) == true)
6:   {
7:    AccountInfos * accountInfos = GatherVipAccountInfo();
8:    // User ID가 변경가 변경됨
9:    downloadAccountInfo(accountInfos, userId); //변경된 User ID로 정보를 download
10:  }
11: }
```

다중 스레드 환경에서 안전한 **getlogin_r()** 함수로 계정 정보를 획득하여 권한을 부여한다.

■ 안전한 코드의 예 - C

```
1:  void downloadVipAccountInfos()
2:  {
3:   const int MAX_USER_ID_LENGTH = 32;
4:   char userId[MAX_USER_ID_LENGTH];
5:   if(getlogin_r(userId, MAX_USER_ID_LENGTH - 1) != 0)
6:   {
7:    // Error 처리
8:   }
9:
```

```
10:    UserList * list = getAuthentifiedUser();
11:    if(isContainedInList(list, userId) == true)
12:    {
13:      AccountInfos * accountInfos = GatherVipAccountInfo();
14:      downloadAccountInfo(accountInfos, userId);
15:    }
16:  }
```

라. 참고문헌

[1] CWE-558 Use of getlogin() in Multithreaded Application, http://cwe.mitre.org/data/definitions/558.html

[2] CWE-807 Reliance on Untrusted Inputs in a Security Decision http://cwe.mitre.org/data/definitions/807.html

[3] 2011 SANS Top 25 - RANK 10 (CWE-807), http://cwe.mitre.org/top25/

제3장 용어정리 및 참고문헌

제1절 용어정리

■ **동적 SQL(Dynamic SQL)**
프로그램의 조건에 따라 SQL문이 다를 경우, 프로그램 실행시 전체 질의문이 만들어져서 DB에 요청하는 SQL을 말한다.

■ **상호배제(Mutual Exclusion)**
동시 프로그래밍에서 공유 불가능한 자원의 동시 사용을 피하기 위해 사용되는 알고리즘으로, 임계구역(critical section)으로 불리는 코드 영역에 의해 구현된다.

■ **소프트웨어 개발보안**
SW 개발과정에서 개발자 실수, 논리적 오류 등으로 인해 SW에 내재된 보안취약점을 최소화하는 한편, 해킹 등 보안위협에 대응할 수 있는 안전한 SW를 개발하기 위한 일련의 과정을 의미한다. 넓은 의미에서 SW 개발보안은 SW 생명주기(SDLC, SW Development Lifecycle)의 각 단계별로 요구되는 보안활동을 모두 포함하며, 좁은 의미로는 SW 개발과정에서 소스코드를 작성하는 구현단계에서 보안약점(잠재적인 보안취약점)을 배제하기 위한 '시큐어코딩(Secure Coding)'을 의미한다.

■ **소프트웨어 보안약점**
소프트웨어 결함, 오류 등으로 해킹 등 사이버 공격을 유발할 가능성이 있는 잠재적인 보안취약점을 말한다.

■ **소프트웨어 보안약점 진단도구**
개발과정에서 소스코드상의 소프트웨어 보안약점을 찾기 위하여 사용하는 도구를 말한다.

■ **소프트웨어 보안약점 진단원**
소프트웨어 보안약점이 남아있는지 진단하여 조치방안을 수립하고 조치결과 확인 등의 활동을 수행하는 자를 말한다.

■ **임계구역(Critical Section)**
병렬 컴퓨팅에서 두개 이상의 스레드가 동시에 접근해서는 안되는 공유 자원(자료 구조 또는 장치)을 접근하는 코드의 일부를 말한다.

■ **전자정부**
정보기술을 활용하여 행정기관 및 공공기관(이하 "행정기관등"이라 한다)의 업무를 전자화하여 행정기관 등의 상호 간의 행정업무 및 국민에 대한 행정업무를 효율적으로 수행하는 정부를 말한다.

■ **정적 SQL(Static SQL)**
동적 SQL과 달리 프로그램 소스코드에 이미 질의문이 완성되고 고정되어 있다.

■ **침입탐지시스템**
외부 네트워크로부터 내부 네트워크 또는 내부 시스템으로 유입되는 공격 트래픽 또는 비정상적인 트래픽을 실시간으로 탐지하는 보안제품이다.

■ **해쉬 함수**
주어진 원문에서 고정된 길이의 의사난수를 생성하는 연산기법이며 생성된 값은 '해쉬 값'이라고 한다. MD5, SHA, SHA-1, SHA-256 등의 알고리즘이 있다.

■ **AES(Advanced Encryption Standard)**
미국 정부 표준으로 지정된 블록 암호 형식으로 이전의 DES를 대체하며, 미국 표준 기술 연구소(NIST)가 5년의 표준화 과정을 거쳐 2001년 11월 26일에 연방 정보처리표준(FIPS 197)으로 발표하였다.

■ **DES 알고리즘**
DES(Data Encryption Standard)암호는 암호화키와 복호화키가 같은 대칭키 암호로 64비트의 암호화 키를 사용한다. 전수공격(Brute Force)공격에 약하다.

■ **HTTPS(Hypertext Transfer Protocol over Secure Socket Layer)**
인터넷 통신 프로토콜인 HTTP의 보안이 강화된 버전이다

■ **ISC2(International Information Systems Security Certification Consortium)**
조직 전체의 보안을 책임질 수 있는 보안 전문가와 정보보호 전문가 자격증 개발에 관심 있는 국제 조직들이 1989년에 컨소시엄을 형성하여 설립한 조직이다.

■ **LDAP(Lightweight Directory Access Protocol)**
TCP/IP 위에서 디렉터리 서비스를 조회하고 수정하는 응용 프로토콜이다.

■ **OAEP(Optimal Asymmetric Encryption Padding)**
Bellare와 Rogaway에 의해서 소개된 RSA를 보완하는 암호 수단으로 제작된 padding scheme이다.

■ **Private Key**
공개키 기반구조에서 개인키란 암·복호화를 위해 비밀 메시지를 교환하는 당사자만이 알고 있는 키이다

■ **Public Key**
공개키는 지정된 인증기관에 의해 제공되는 키 값으로서, 이 공개키로부터 생성된 개인키와 함께 결합되어, 메시지 및 전자서명의 암·복호화에 효과적으로 사용될 수 있다. 공개키를 사용하는 시스템을 공개키 기반구조라 한다.

■ **SHA(Secure Hash Algorithm)**

해쉬 알고리즘의 일종으로 MD5의 취약성을 대신하여 사용한다. SHA, SHA-1, SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512) 등의 다양한 버전이 있으며, 암호 프로토콜인 TLS, SSL, PGP, SSH, IPSec 등에 사용된다.

■ **RC5**

1994년 RSA Security사의 Ronald Rivest에 의해 고안된 블록 암호화 알고리즘이다.

■ **Umask**

파일 또는 디렉터리의 권한을 설정하기 위한 명령어이다.

■ **Whitelist**

블랙리스트(Black List)의 반대 개념으로서 신뢰할 수 있는 사이트나 IP 주소 목록을 의미한다. 즉, 신뢰할 수 있는 글이나 문자열의 목록을 말한다.

■ **Wraparound**

int 또는 long으로 정의된 변수의 값이 한계치를 상회했을 경우, MSB(Most Significant Bit)가 바뀌어 양수는 음수, 음수는 양수로 전환된다.

제2절 참고문헌

[1] 소프트웨어 개발보안 가이드, 행자부, May 2012.

[2] 소프트웨어 보안약점 진단가이드, 행자부, May 2012.

[3] CWE(Common Weakness Enumeration), MITRE, http://cwe.mitre.org

[4] CWE/SANS Top 25 Most Dangerous Software Errors(2011), MITRE, http://cwe.mitre.org/top25/

[5] CVE(Common Vulnerabilities and Exposures), MITRE, http://cve.mitre.org

[6] CERT Secure Coding Standards, CERT, https://www.securecoding.cert.org

[7] NIST(National Institute of Standards and Technology), http://www.nist.gov

[8] OWASP Top Ten Project 2010, OWASP, https://www.owasp.org/index.php/Top_10_2010

[9] OWASP Top Ten Project 2007, OWASP, https://www.owasp.org/index.php/Top_10_2007

[10] Robert C. Seacord, “The CERT C Secure Coding Standard”, Addison-Wesley, October 2008.

[11] Robert C. Seacord, “Secure Coding in C and C++”, Addison-Wesley, May 2010.

전자정부 SW 개발 · 운영자를 위한

C 시큐어코딩 가이드

2012년 9월 인쇄
2012년 9월 발행

발행처: 행정자치부 · 한국인터넷진흥원

서울특별시 종로구 세종대로 209
정부중앙청사

서울특별시 송파구 중대로 135
IT 벤처타워

인쇄처: 한올
Tel: (02) 2279-8494

(비매품)