# 패스워드 알고리즘의 개선에 관한 연구

김영수* · 박연식** · 임재홍*

*한국해양대학교 · **경상대학교 정보통신공학과 해양연구소

# A Study on Improvement of Password Algorithm

Yeong-su Kim* · Yeon-sik Park** · Jae-hong Yim*

*Korea Maritime University · **Gyeongsang National University

E-mail : shalom@nongae.gsnu.ac.kr

## 요 약

컴퓨터 보안의 첫 단계는 패스워드이다. 완벽한 방화벽이 구축되어 있을지라도 패스워드 보안이 무력하다면 방화벽은 무용지물이나 다름없다. 패스워드의 관리는 시스템관리자보다는 일반 사용자(end-user)에게 달려 있으므로 관리의 허술성을 이미 안고 있다. 그것은 일반 사용자들은 고난이도의 패스워드 관리를 할 수 없기때문이다.

본 논문에서는 사용자들에게는 기존의 패스워드 입력 패턴을 유지시켜 주면서, 해킹이 어려워지도록 패스워드 알고리즘을 개선하였다.

## ABSTRACT

The first stage for computer security is password. If security of password is impotent even constructing of perfect fire-wall, fire-wall is not anything but a good-for-nothing. Because management of password is depend upon an end-user rather than a system-manager, carelessness of password management is an inevitable result. It is a reason that an end-user is actually not able to manage a high-difficulty-password.

In this paper, algorithm of password is improved to be difficult of hacking, having a existing password input pattern for an end-user.

## Ⅰ. 서론

인터넷의 확산으로 PC 사용자들의 증가와 더불어 원격접근(Remote Access)도 보편화되었다. 이것은 많은 PC 사용자들에게 편리성을 제공하는 반면에 악의가 있는 해커들의 공격통로가 되기도 한다. 그리고 기업이나 국가간의 복잡한 이해관계 속에 있는 현사회는 바로 옆의 동료도 해커가 될 수 있다.

호스트를 해킹 할 때 보통 3단계를 거친다.

① 목표 호스트에 접속하여 쉘(shell)을 사용할 수 있는 권한을 얻어낸다. 이를 위해서 사용자의 패스워드를 알아야 하는데, 전문 해커들을 주로 스니프(sniff)를 이용하는 것으로 알려져 있다.

② 호스트에 접속한 후에는 호스트 관리자(root)의 권한을 획득하는 것이다.

③ 호스트 관리자(root)의 권한을 획득한 뒤에는 자신이 다음에 다시 침입할 경우에 편의성을 기하기 위해 backdoor(뒷문)를 만들어 놓고 나가는 것이다. 물론 이 때에는 자신만의 재잠입을 위해서 자신만이 사용할 수 있는 내부 버그(트로이 목마)를 남겨놓는 것이 일반화되어 있다.

해커들은 첫 번째 단계를 통과하면 다음 단계는 OS의 버그를 이용한다. 현재는 UNIX 시스템을 거의 사용하고 있으며, UNIX에 대해서는 많이 알려져 있으므로 해커들이 유닉스를 사용하는

* 한국해양대학교 전자통신공학과

** 경상대학교 해양산업연구소 · 정보통신공학과

데는 별로 어렵지 않다. 그러므로 패스워드에 의한 방어는 그 무엇보다도 중요하다.

그러나 현재의 패스워드 입력 알고리즘은 손가락을 자세히 살펴보면 알아낼 수 있다. 그것은 현재의 패스워드 알고리즘은 8자 이내의 문자를 입력하도록 설계되어 있는 실정이고, 패스워드 보안을 중요시하는 웹사이트에서는 그 이상의 문자열 입력을 요구하고 있다. 그러나 사용자들은 까다로운 패스워드 사용을 회피하고 있기 때문에 패스워드의 기능을 충분히 발휘하지 못하고 있다. 또한 스니퍼에 의해서도 완전히 노출되고 있는 것이나 다름없다. 그것은 단말기에서 호스트로 들어가는 동안에는 사용자 ID와 암호가 보호되지 않은 채로 들어가고 있기 때문이다.

본 논문에서는 사용자들에게는 기존의 패턴을 유지시켜 주면서, 해킹이 어려워지도록 암호입력 알고리즘을 개선하였다.

## II. 패스워드관련 해킹

### 1. 스니핑(Sniffing)

#### (1) 스니핑을 이용한 해킹

스니핑이란 네크워크의 한 호스트에서 실행되어 그 주위를 지나 다니는 패킷들을 훔치는 것을 말하며 이러한 기능을 가진 프로그램을 스니퍼(Sniffer)라 한다. 스니퍼는 계정과 패스워드를 알아내기 위해서 해커들이 자주 사용하고있다.

다음의 내용은 스니퍼 프로그램을 가동한 후 훔쳐온 내용들이다.

```
mordor# ./sniffit

Log started at => Mon Apr 8 20:29:04 [pid 10937]

-- TCP/IP LOG -- TM: Mon Apr 8 20:29:44 --
PATH: rohan.kaist.ac.kr(1270) => gondor.kaist.ac.kr(telnet)
STAT: Mon Apr 8 20:29:48, 30 pkts, 77 bytes [TH_FIN]
D A T A :
(255)(253)^C(255)(251)^X(255)(251)^_(255)(251)
(255)(251)!(255)(251)"(255)(253)^E(255)(251)#(255)(251)$
(255)(250)^X
   :
   IRIS-ANSI-NET(255)(240)(255)(253)^A(255)(252)^Aaragorn
   : evenstar
   : cls
   : du -s -k *
   : elm awen
-- TCP/IP LOG -- TM: Mon Apr 8 20:31:57 --
PATH: mordor.kaist.ac.kr(2389) => gondor.kaist.ac.kr(telnet)
STAT: Mon Apr 8 20:32:24, 106 pkts, 128 bytes [DATA LIMIT]
D A T A :
(255)(253)&(255)(251)&(255)(253)^C(255)(251)^X(255)
(251)^_(255)(251)
(255)(251)!(255)(251)"(255)(251)$(255)(253)^E(255)(251)#(255)(250)^_
   : P
   : ^X(255)(240)(255)(250)
   : 9600,9600(255)(240)(255)(250)^X
   :
   XTERM(255)(240)(255)(253)^A(255)(252)^Agaladriel
   : shwjdtjr=JD
   :
   : setenv DISPLAY rohan.kaist.ac.kr:0.0
   : ne
```

gondor, mordor, rohan은 스니핑당한 컴퓨터인데 침입자는 이미 mordor에서 루트 권한을 따냈었고 스니퍼 프로그램을 실행시키고 있다(스니퍼가 동작하기 위해서는 네크워크 디바이스의 조작이 필요한데 이는 루트(시스템 관리자)만이 할 수 있다. 그렇다면 물리적인 네크워크 구도에서 인접해있는 호스트들은 모두 스니핑을 당하게 된다는 결론에 도달하게 된다.

처음 블럭에서 rohan에서 gondor로 telnet으로 gondor로 login했으며 계정은 aragorn이며, 패스워드는 evenstar임을 알 수 있다. 그리고 login 직후에 실행한 몇몇 명령까지 나타났다.

일반적으로 스니퍼 프로그램은 비교적 작은 크

기의 제한된 버퍼를 사용하는데 필요에 따라 그것을 늘이기도 한다.

(2) 이더넷의 약점과 스니퍼의 원리

이더넷이서 호스트 A가 호스트 B에게 패킷을 보낼 때는 A와 B가 배타적인 연결을 하는 것이 아니라 패킷을 이더넷에 날려보낸다. 패킷은 보통 수신주소의 호스트만이 받게 되지만 Promiscuous mode에서는 상황이 달라진다.

Promiscuous mode란 네크워크 디바이스가 자신과는 관련없는 다른 호스트를 향해 지나가는 패킷까지 끝을 수 있는 상태를 말한다.

스니퍼나 대부분의 네트워크 분석 프로그램들이 Promiscuous mode에서 동작한다.

(3) 스니퍼의 동작방법

스니퍼는 일반적으로 다음과 같은 일련의 동작을 하는 코드로 구성된다.

① 네트워크 디바이스를 Promiscuous mode로 설정한다.

② 지나가는 모든 패킷을 읽는다.

③ 패킷을 필터링해서 발신 및 수신 주소, 서비스(telnet, rlogin, ftp, smtp 등), 그리고 계정과 패스워드가 포함된 데이터를 구분해서 출력한다.

2. Terminal Hijacking

터미널 하이재킹이란 사용자가 로그인하기를 기다렸다가 로그인 한 후 호스트와 로컬 시스템에 establish된 stream에 자신이 원하는 명령어를 사용하는 것이다.

터미널 하이재킹을 위한 툴은 현재 SunOS4.1.3용만 발견되고 있지만 다른 운영체제용으로 포팅되었을 가능성도 상당히 높다.

이것도 해커들이 방화벽을 뚫을 때 많이 사용하는 방법중 하나이다. 예를 들어, 출장중인 사용자 A가 지방의 호스트 B에서 기업 C의 방화벽을 통해 합법적인 인증절차를 통해 로그인 했을 경우, 해커는 기업 C의 네트웍을 방화벽 때문에 뚫고 들어가지 못하지만 호스트 B의 시스템 관리자 권한을 얻어 하이재킹 툴을 설치하였을 경우 사용자 A가 기업 C에 establish한 session에 자신이 원하는 스트링을 읽고 쓸 수 있게 된다.

기업 C의 입장에서 볼 때 이러한 형태의 공격을 막을 방법이란 외부로부터의 로그인을 불가능하게 하는 방법밖에 없다

## III. 패스워드 보안책

1. 스니퍼 예방

스니퍼에 대비하는 방법의 원칙은 패스워드가 네트워크를 통해 전달되지 않게 하는 것으로, 암호화나 패스워드를 대신하는 토큰의 이용 등을 들 수 있다. 공개 소프트웨어 ssh[2]는 여기에 사용하기에 가장 무난하다. 이것은 패킷을 암호화해서 sshd(서버)에게 보내는 ssh, scp(클라이언트)가 telnet, rlogin, ftp를 대신하는 패키지이다.

2. 스니퍼 탐지

스니퍼는 Promiscuous mode에서 동작하므로 네크워크 디바이스의 상태 플래그에 PROMISC가 있다면 일단 스니퍼를 의심해 볼 필요가 있다.

SunOS 4.1.x 같은 BSD 계열의 유닉스나 IRIX에서는 ifconfig를 사용해서 스니퍼의 존재를 확인할 수 있으나 대부분의 다른 유닉스(특히 Solaris)에서는 그것이 불가능하다.

SunOS 4.1.3, IRIX 5.3 에서 스니퍼 프로그램을 실행시키면서 ifconfig로 확인하면 다음과 같다.

두 기계에서 모두 flags를 나타내는 행에 PROMISC가 있는 것으로 스니퍼가 활동중임이 확인된다. SunOS 4.1.x에서 사용가능한 cpm이라는 프로그램이 있는데 ifconfig로 플래그를 조사하는 것과 같은 기능을 가진다.

일반적으로 스니퍼는 로그 파일을 만들어 데이터를 모으므로, 로그 파일을 찾으면 스니퍼를 쉽게 찾을 수 있다.

① 다른 호스트로 연결하면 스니퍼 로그 파일에 로그가 새로 만들어지게 되므로 미리 만들어 두었던 /tmp/sniff_trap이란 파일을 기준으로 찾을 수가 있다. 그리고 빠른 검출을 위해 루트에서 프로세스의 우선권을 최대로 하는 것이 좋다. 또한 현재 열려져 있는 모든 파일의 리스트속에 스니퍼 로그파일도 포함되어 있을 것이므로 스니퍼를 찾는데 활용하록 한다. 필요하면 lsof[3] 툴을 사용하도록 한다.

② 다음으로 스니퍼 프로세스를 찾기 위해 시

2) Secure Shell:http://www.cs.hut.fi/ssh
3) (ftp://ftp.cert-kr.or.kr/pub/tools/lsof)

스템을 리부팅하면 되겠지만 침입자를 추적하고 싶다면 그 프로세스를 직접 찾아내어야 한다.

루트로 실행되고 있는 프로세스중에서 수상한 것들을 찾는다. ./a.out, ./in.telnetd, in.uucpd 등으로 숨겨지는게 보통이다.

BSD ps(/usr/ucb/ps)가 사용 가능하다면 e옵션을 추가해서 프로세스가 시작됐을 때의 환경변수를 알아내어 어떤 계정이 해킹에 사용되었는지를 알아낼 수 있다.

### 2. One Time Password System

#### 2.1 원타임 패스워드 시스템

기존의 패스워드 인증 시스템에서는 호스트에 로그인할 때 동일한 패스워드를 매번 사용하므로 네트워을 도청하게 되면 쉽게 알아낼 수 있다.

원타임 패스워드는 시스템에 로그인 할 때 마다 항상 다른 패스워드를 사용하도록 함으로써 도청의 문제점을 해결하고 있다. 일단 한번 사용된 패스워드는 재사용이 불가능하므로 침입자가 네트워크 도청을 통해서 원타임 패스워드를 알아내어도 더 이상 이용할 수 없게 된다[1].

원타임 패스워드 시스템을 구현하는 방법에는 다음과 같은 것이 있다.

① 동기화된 시간을 유지하여 Time-Stamp를 사용

② 양쪽의 임의의 패스워드 리스트 내의 위치를 동기화하여 패스워드 사용

③ Sequence generator의 상태를 동기화하여 임시적인 sequence number 사용

④ Challenge-Response Schemes 이용

현재 원 타임 패스워드 기법을 이용해 구현한 제품에는 Bellcore의 S/KEY (TM) 시스템, the US Naval Research Laboratory (NRL)의 "One-time Passwords In Everything" (OPIE), logdaemon 등이 있다[2].

OPIE 시스템에서 이용한 Challenge-Response Schemes을 요약하면 다음과 같다.

① 사용자가 로그인을 시도하면 서버쪽에서 임의의 Challenge 메세지를 생성해서 사용자에게 전송한다.

② 사용자는 PIN(Personal Identification Number)과 Challenge를 이용하여 서버에 전송할 원타임 패스워드를 생성하고, 서버에게 Response 메세지를 전송한다.

③ 서버는 동일한 Challenge와 등록된 사용자의 정보을 이용해 원타임 패스워드를 생성한 후 사용자가 전송한 Response와 비교하여 로그인을 허락한다[3].

## IV. 기존 패스워드

### 1. 구성

기존 패스워드의 길이는 보통 8자 이내로 제한하고 있으며, 보안에 신경을 쓰는 사이트는 그 이상의 문자열을 입력하도록 하고 있다. 그리고 양질의 패스워드 작성을 위해서 다음과 같은 방법을 권고하고 있다.

① 대소문자, 숫자, 특수문자를 섞어서 만든다.

② 기억하기 쉽도록 만들되 기록하지 않는다.

③ 빨리 입력할 수 있도록 구성한다.

④ 문장의 단어들의 첫 글자를 이용한다.

⑤ 영문상태로 한글단어를 만든다.

⑥ 짧은 두 단어와 그 단어 사이에 '+', ';', '?', ':'등의 특수문자를 혼용한다.

### 2. 패스워드 입력 알고리즘

```
Accept_Password:
accept Terminal_Password                    /*8자리 이내
                        /* 전송할 password의 암호화
call Encryption_Algorithm_Function(Terminal_Password)
                    /* 암호화된 password를 호스트로 전송
transfer Terminal_Password
if (Terminal_Password equal to Host_Password)
        then permission login
        else goto Accept_Password
```

### 3. 알고리즘의 문제점

기존의 패스워드 알고리즘은 대체적으로 8자 이내의 문자를 받아들이고 있으며, 패스워드 보안을 중요시하는 웹사이트는 그 이상의 문자열 입력을 요구하고 있다.

그러나 일반사용자들은 기억의 편리함을 주로 선호하므로 앞에서 언급한 양질의 패스워드 입력 방법에 대한 실효성이 없다. 이것은 과거 DOS 상

태에서 파일명을 그럴듯하게 작성하는 것만큼 어렵다.

그리고 이해관계가 복잡한 현대사회에서는 바로 옆의 동료가 산업스파이가 될 수도 있으며, 중요기관의 경우에는 어느 곳에나 이중간첩이 존재할 수 있다. 이들은 패스워드 입력과정에서 손가락을 보고도 패스워드를 훔칠 수 있다.

기존의 패스워드 알고리즘에서는 이러한 것에 대한 대책이 전혀 없다. 네트워크상의 스니핑을 방지하기 위한 완벽한 암호화 알고리즘이 준비되어 있을지라도 초등과정에서 헛점을 노출시키면 아무리 좋은 방화벽도 무용지물이 되는 것이다.

## V. 개선된 패스워드의 구성

### 1. 입력버퍼의 확장

일반적으로 입력할 수 있는 패스워드의 길이는 영문 8자 정도이다. 경우에 따라서는 그 이상의 입력을 지공하고 있으나 실제 패스워드만을 입력하도록 되어있다. 그러나 본 절에서 제안하는 방법은 다음과 같다.

① 패스워드 입력버퍼의 길이를 영문 256자로 한다. 필요하다면 그 이상을 사용할 수도 있다.

② 패스워드 입력과정에서 가짜의 문자열과 함께 실제 패스워드도 입력을 하며, 서버에서는 올바른 패스워드가 어느 위치에 있든지간에 검출만 되면 인증을 허락한다.

이렇게 하면 옆의 사람이 아무리 자세히 보아도 패스워드를 기억하지는 못하게 된다. 응답속도면에서는 해당 문자열을 검출하는데 시간이 더 걸릴 것이라고 생각되겠지만 컴퓨터의 처리속도가 엄청나게 향상되었으므로 사용자의 체감속도는 1~2초 이내가 된다. 그러나 이 방법은 스니퍼에게는 무용지물이 된다. 언뜻 보기에는 어떤 문자열이 패스워드인지 스니퍼가 모를 것으로 판단하기 쉽다. 이 때 스니퍼는 진짜 패스워드에 관심을 가지지 않고 패스워드 버퍼의 전체를 스니핑한 후에 그것을 그대로 서버로 보내면 정상적으로 인증을 받게된다. 이러한 취약점을 보완하기 위해 다음의 방법을 제안한다.

### 2. 접속된 최종버퍼를 활용

이 방법은 마지막으로 접속한 패스워드 버퍼의 내용을 저장하여 두었다가 다음에 같은 내용으로 접속하면 login을 거부하도록 한다. 스니퍼는 정상적인 사용자보다 항상 한걸음 늦을 수밖에 없다. 왜냐하면 정상 사용자의 패스워드를 스니핑한 후에 시도하기 때문이다.

본 절에서는 다음과 같은 과정을 거치면서 스니퍼나 하이재킹을 방어할 수 있는 방법을 제안한다

① 클라이언트에서 난수에 의하여 생성된 문자열을 버퍼에 저장한 후 이 버퍼의 내용을 서버로 보낸다. 이 때 패스워드는 난수에 의하여 버퍼의 임의의 위치 i번째부터 저장되도록 한다.

② 서버에서는 패스워드 리스트에 있는 패스워드가 적합한가를 확인한다.

③ 접속을 시도하는 버퍼의 내용이 서버에 저장되어 있는 버퍼의 내용과 일치하는 것으로 판명되면 인증을 거부한다.

해커는 ③번의 과정을 통과하기 위하여 다음의 문제들을 해결해야만 한다.

① 스니핑한 버퍼의 내용중에서 패스워드의 위치와 문자열의갯수를 알아내야 한다.

② 버퍼의 내용과 같으면 접속이 거부되므로 이 내용을 어떤 방법으로든지 변경은 시키야 하지만, 그 속에는 패스워드가 들어있음을 알아야 한다.

이러한 문제를 해결하기 위한 해커의 노력을 추측해 보면 자신의 상식선에서 버퍼를 일률적으로 분할하여 볼 것이다. 그런데 사용자들을 자신의 취향대로 보통 6자 내지 10자 정도를 정할 것이다. 그리고 정확하게 분할하더라도 정상적인 패스워드 문자열이 잘려질 가능성이 많으므로 해커가 시도해야 할 횟수는 엄청나다.

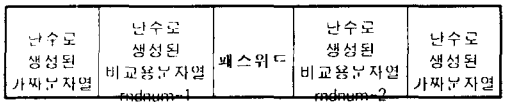
그런데 해킹 방지를 위해서 사용한 방법이 정상 이용자의 접속을 불가능하게 할 가능성을 생각해 볼 수도 있을 것이다. 그러나 새로운 난수를 발생시켰을 때 기존의 버퍼내용과 동일하게 될 확률을 전제로 해야한다. 이에 대해서는 다음 절에서 설명하기로 한다.

### 3. 정상 이용자와 해커의 판별

정상 이용자와 해커를 판별하기 위하여 패스워드를 제외한 문자열을 모두 비교해야 하며, 판별분석과정의 구현이 난이한 부분이 있고, 더 중요

한 것은 정상 이용자는 항상 접속이 가능하고 해커는 항상 불가능할 확률을 제시해 주어야 하는데 이러한 수학적 증명은 쉽지않다.

그래서 본 논문에서는 이러한 문제를 <그림1>과 같이 단순화시켜 구현하기로 한다.




<그림1> 패스워드 버퍼의 구성

서버에서는 패스워드를 확인하는 과정에서 패스워드 문자열의 시작위치를 기억해 둔다. 패스워드의 길이를 p, 패스워드의 시작위치를 i라고 하면 rndnum-1은 i-(p-1)부터 (p-1)개, rndnum-2는 (i+p)부터 (p-1)개를 비교하도록 한다. 비교는 상황에 따라 rndnum-1이나 rndnum-2를 선택한다.

비교문자열을 (p-1)로 하는 이유는 해커가 일정한 크기로 배열을 변경할 경우 패스워드가 잘리도록 하기 위해서이며, 비교문자열을 패스워드의 앞뒤에서만 선택하면 해커의 추측은 무기력하게 되고 알고리즘의 구현은 용이하게 된다.

rndnum-1이나 rndnum-2의 난수열(A)과 접속자의 난수열(B)이 일치하지 않아야 항상 접속할 수 있으며, 일치하지 않을 확률

$$P(A \neq B) = 1 - (\frac{1}{2^8})^{p-1}$$ 이 된다.

만일 패스워드의 길이를 6 byte로 했을 경우

$$P(A \neq B) = 1 - (\frac{1}{2^8})^5$$ 이 된다.

이 계산에서 보면 일치하지 않을 확률이 거의 1에 가까우므로 정상 이용자가 접속하는데는 어려움이 없다는 것을 암시하고 있다.

그러나 본 논문에서는 36종류(영문 소문자 26, 숫자 10)만 사용하였다. 왜냐하면 사용자들이 주로 사용하는 패스워드는 영문소문자와 숫자로 거의 이루어지므로 난수속에 섞어놓을 경우 오히려 눈에 잘 뜨일 수 있기 때문이다. 이렇게 될 경우에 위의 예를 든다면

$$P(A \neq B) = 1 - (\frac{1}{36})^5$$ 이 된다.

## VI. 알고리즘 구현

```
#include <stdio.h>
#include <stdlib.h>
main()
{
char string[36] =
 "0123456789abcdefghijklmnopqrstuvwxyz";
char server__password[20] = "pass123";
char c__random[20], s__random[20];
char client__password[100], client__buff[100],
      server__buff[100];
int  position1, position2, password__length,
      i, j, num, logon;
int  c__position, s__position;
/* 클라이언트에서 패스워드를 읽어들임 */
for (i=0; i<=99; i++)
      client__password[i] = " ";
printf("\n enter, password : ");
scanf("%s", client__password);
/* 클라이언트에서 버퍼에 난수를 채움 */
randomize();  /* initializing */
for (i=0; i<=99; i++)
        client__buff[i] = string[rand() % 36];
/* 클라이언트의 패스워드를 임의의 위치에 저장 */
position1 = rand() % 100;
password__length = strlen(client__password);
/* 임의의 위치가 버퍼크기를 초과할 경우 조정 */
num = position1 + password__length;
if (num > 100)
{
   position2 = position1 - password__length;
   for (i = 0; i <= password__length; i++)
   client__buff[position2+i]=client__password[i];
 }
   else
 {
   for (i = 0; i <= password__length; I++)
    client__buff[position1+i]=client__password[i];
  }
/* 클라이언트의 패스워드를 확인 */
  i = 0;
  j = 0;
  password__length = strlen(server__password);
  do {
       if (client__buff[i] == server__password[j])
        {
          i++; j++;
        }
   else
        {
          i++; j = 0;
        }
       } while(i<99 & j<password__length);

c__position = i - j - password__length + 1;
logon = 0;
/* 로그온 가능 여부 체크 */
if (j == password__length)  logon++;
/* 서버 버퍼의 패스워드 위치를 파악 */
i = 0;
j = 0;
do {
      if (server__buff[i] == server__password[j])
        {
          i++; j++;
        }
       else
         {
```

```
            i++; j = 0;
          }
      } while (i<99 & j<password_length);
/* 서버/클라이언트의 패스워드앞의 난수 비교 */
s_position = i - j - password_length + 1;
for (i = 0; i < password_length - 1; i++)
    {
      c_random[i] = client_buff[c_position];
      s_random[i] = server_buff[s_position];
      c_position++; s_position++;
    }

num = 0;
for (i = 0; i < password_length - 1; i++)
     {
      if (c_random[i] == s_random[i]) num++;
     }
/* 로그온 가능 여부 체크 */
if (num < password_length - 1)  logon++;
/* 최종 로그온 결정 */
if (logon < 2)
    printf("** ERROR, login failed **");
    else
      {
        printf("** OK, login successed **");
/* 다음 인증을 위해 저장 */
        for (i=0; i<= 99; i++)
           server_buff[i] = client_buff[i];
      }
}
```

본 절의 알고리즘으로 시뮬레이션 해 본 결과 다음과 같은 문제점이 도출되었다. 패스워드 주위의 문자열은 난수에 의해서 생성되므로 문자열들이 대체적으로 고르게 분포되었다. 이로 인해 사용자들이 특정단어를 사용하거나 숫자를 연속해서 사용할 경우 눈에 뜨일 수 있는 가능성을 안고 있다.

보안이란 언젠가는 무너질 수 있다는 것을 전제하는 것이 현명하다. 해커는 자신이 고안한 알고리즘을 사용하여 지속적으로 접속을 시도할 것이다. 이러한 것에 대비해서 서버쪽에서는 접속시도 로그파일을 작성할 필요가 있다. 만일 짧은 시간대에 접속을 실패한 빈도수가 높으면 해킹의 시도가 있었음을 의미하므로 정상 이용자가 접속을 했을 경우 이를 알려주도록 한다. 그리고 짧은 시간대에 접속실패의 빈도수가 지나치게 많으면 서버쪽에서 일방적으로 접속을 제한시키는 것도 보안유지상 필요할 것이다.

## VII. 결론

본 논문에서는 두 가지의 개선된 패스워드 입력 알고리즘을 제시하였다. 4장 1절의 "입력버퍼의 확장" 알고리즘은 구현이 간단하고 용량이 적으므로 CMOS용과 유사한 용도로 사용하면 적합할 것이다. 2절의 "접속되 최종버퍼의 활용" 알고리즘은 네트워크를 통해서 접속하는 곳에서 스니핑이나 하이재킹 등의 염려가 있을 때 사용하면 적합하다. 그리고 사용자의 기억이나 처리속도에 어느정도 영향을 받을 수 있겠으나 가급적이면 패스워드의 길이가 길고 버퍼의 크기도 해커는 불리하다.

그리고 패스워드의 정오(true-false)를 판별하는데 있어서 확률을 도입하는 것에 대해 이견(異見)이 있을 수도 있을 것이다. 그러나 이미 확률이 학문의 여러 분야에서 적용되고 있으며 실용화되고 있으므로 이 분야에서의 확률의 도입도 긍정적인 측면으로 검토되어야 할 것이다.

## 참고문헌


1. "A One-Time Password System RFC", ftp://ds.internic.net/rfc/rfc1938.txt
2. http://www.ietf.cnri.reston.va.us/html.charters/otp-charter.html
3. Brent Chapman and Elizabath D. Zwicjy, "Building Internet Firewalls", pp. 359-365