Journal of The Korea Society of Computer and Information
Vol. 20, No. 5, May 2015
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.5.041

리눅스 서버에서 인터렉티브 서비스 Stepping Stone 자가진단을 위한 brute-force 기법

강구홍*

A Brute-force Technique for the Stepping Stone Self-Diagnosis of Interactive Services on Linux Servers

Koo-Hong Kang*

요 약

인터넷을 통해 악의적으로 접근하는 공격자들은 자신의 노출을 최대한 감추기 위해 중간 호스트(소위, stepping stone으로 불림)를 경유한다. 본 논문에서는 텔넷, SSH, 그리고 rlogin 등 인터렉티브 서비스를 이용하여 리눅스 서버에 접근하여 다시 이러한 인터렉티브 서비스를 이용하여 다른 컴퓨터로 원격 접속을 시도하는 행위를 brute-force한 방법으로 검출하는 기법을 제안한다. 제안된 기법은 인터렉티브 서비스 데몬(Daemon) 프로세스의 시스템 콜(system call)을 감시하여 stepping stone 여부를 진단하기 때문에 ssh 접속과 같은 암호화 연결에서 대해서도 완벽한 검출 결과를 제공할 수 있다. 본 논문에서는 ptrace 시스템 콜과 간단한 쉘 스크립트를 작성하여 제안된 기법을 CentOS 6.5 64비트 환경에서 실질적으로 구현하였다. 마지막으로 몇몇 실험 시나리오를 대상으로 실시한 현장 운영을 통해 제안된 brute-force 기법을 검증하였다.

▸ Keywords : Stepping stone, 역추적, 연결체인

Abstract

In order to hide their identities, intruders on the Internet often attack targets indirectly by staging their attacks through intermediate hosts known as stepping stones. In this paper, we propose a brute-force technique to detect the stepping stone behavior on a Linux server where some shell processes remotely logged into using interactive services are trying to connect other hosts using the same interactive services such as Telnet, Secure Shell, and rlogin. The proposed scheme can provide an absolute solution even for the encrypted connections using SSH because it traces the system calls of all processes concerned with

•제1저자 : 강구홍 •교신저자 : 강구홍
•투고일 : 2015. 2. 11, 심사일 : 2015. 3. 3, 게재확정일 : 2015. 5. 6.
* 서원대학교 정보통신공학과(Dept. of Information and Communication Engineering, Seowon University)

the interactive service daemon and their child processes. We also implement the proposed technique on a CentOS 6.5 x86_64 environment by the ptrace system call and a simple shell script using strace utility. Finally the experimental results show that the proposed scheme works perfectly under test scenarios.

▸ Keywords : Stepping stone, Trace-back, Connection chain

I. 서 론

네트워크를 통한 해커들의 침입은 인터넷 사용자들에게 심각한 위협을 주고 있다. 그러나 이들 침입자들은 인터넷 프로토콜과 서비스를 이용한 다양한 방법으로 범죄 행위로부터 자신들의 정체와 위치를 쉽게 숨기고 있다[1-5]. 이들 해커들이 사용하는 가장 흔한 방법 중 하나는 최종 타겟 호스트(target host)를 침입하기 전에 여러 개의 중간 호스트(소위, stepping stone으로 불림)를 경유하는 것이다. 예를 들어, 호스트 A에 로그인(login)한 해커가 호스트 B로 원격 접속하기 위해 텔넷(Telnet)하고, 다시 원격 접속된 호스트 B에서 호스트 C로 원격 접속하는 행위를 한다고 가정한다. 호스트 C에서 트래픽 분석을 통해 호스트 B로부터 공격을 받았다고 확인할 수 있어도 공격의 근원지인 호스트 A의 정보를 쉽게 확인하는 것은 불가능하다. 물론, 이때 호스트 B와 C사이에 더 많은 stepping stones를 가정할 수 있다. 한편, 이와 같은 stepping stones로 연결된 일련의 연결들을 연결체인(connection chain) 혹은 확장연결(extended connection)이라고 부른다. 2013년 3월 20일 대한민국의 주요 언론과 기업의 전산망이 마비되고, 많은 컴퓨터가 부팅조차 되지 않는 엄청난 공격을 받았으며 이들 공격의 근원지에 대한 많은 이견이 아직까지 진행되고 있다[12]. 뿐만 아니라, 국제적 이슈가 되고 있는 2014년 11월 24일에 발생한 미국 소니픽쳐스 해킹사건 역시 공격의 근원지로 지목된 당사자들은 여전히 사실을 부정하고 있는 현실이다. 따라서 공격의 근원지를 역추적하는 기술은 현재 우리가 당면하고 있는 가장 절박한 기술 요구중 하나가 될 것이다.

1990년 초부터 텔넷과 rlogin과 같은 인터렉티브(interactive) 서비스를 이용한 연결 체인을 통한 침입이 검출될 때 공격자의 최초 근원지를 추적하는 역추적 기술(trace-back technology)이 연구되기 시작했다[1,2] 그러나 이들 초창기 역추적 기술은 stepping stone으로 이용되는 모든 호스트들이 역추적 과정에 직접 관여하는 방법을 채택함으로써 실제적으로 인터넷에 설치하여 사용하기에는 한계가 있었다. 즉 연결체인 상에 존재하는 stepping stone 호스트들이 모두 정상적인 역추적 기능을 제대로 수행할 때 비로소 근원지를 추적할 수 있게 된다. 따라서 인터넷 상에 존재하는 모든 호스트들이 이러한 역추적 기능을 수행한다는 것은 당연히 현실적으로 불가능하다는 문제점에 봉착하게 된다. 이와 같은 초창기 역추적 기술을 호스트-기반 (host-based) 역추적 기술이라고 부른다. 이러한 호스트-기반 역추적 기술의 한계를 극복하기 위하여 Staniford-Chen과 Heberlein[3]은 1995년 네트워크에서 패킷 스트림의 콘텐츠(contents)를 직접 모니터링하여 연결 체인을 검출하는 방법을 제안하였다. 즉 네트워크에 여러 개의 모니터링 장치를 설치하고 이들 장치들로부터 stepping stones를 검출하기 위한 다양한 정보를 기반으로 연결 체인을 검출하게 된다. 이러한 방법을 네트워크-기반 (network-based) 역추적 기술이라고 부른다. 2000년 초부터는 SSH(Secure Shell)과 같은 암호화된 연결에 대해서도 적용가능 하도록 다양한 형태의 네트워크-기반 역추적 기술이 제안되고 있다[4-6].

본 논문에서 하나의 리눅스 서버에서 stepping stone 역할을 진행하는 관련 프로세스들의 존재 여부를 실시간 자동으로 검출하는 방법을 제안한다. 그러나 제안된 방법은 공격의 근원지를 역추적하는 시스템 차원의 기법을 제안하는 것은 아니다. 물론 본 논문에서 제안한 기법을 앞에서 설명한 호스트-기반 역추적 기술과 같이 모든 호스트에서 이루어지고 이들 호스트들로부터 관련 정보를 수신하여 연결추적을 수행하는 중앙집중식 혹은 분산처리 등을 수행하는 분석 장치가 있다면 역추적 시스템으로 확장 구현하는 것이 불가능한 것은 아니다. 그러나 본 연구는 네트워크-기반 역추적 기술에 대한 검증 혹은 보완 기술이라고 보는 것이 보다 정확할 것이다. 이러한 연구 동기를 다음과 같이 보다 구체적으로 서술한다.

앞에서 설명한 바와 같이 호스트-기반 역추적 시스템은 하

나의 소규모 도메인 혹은 네트워크 관리 영역 내에서 제한적으로 구현될 수 있다. 따라서 실제적으로 인터넷에서 역추적을 구현하기 위해서는 네트워크-기반 기술 적용이 불가피 하다. 그러나 암호화된 연결체인 상에서 stepping stones를 검출하기 위해서는 콘텐츠 매칭을 이용한 방법은 불가능하며, 결국 통계적 혹은 타이밍 정보 등을 이용하는 방법이 제안되고 있다. 따라서 이러한 기법들은 필연적으로 오탐율(false rate)을 동반하게 된다. 즉 제안된 네트워크-기반 역추적 기법을 검증 (stepping stone 연결이 아님에도 불구하고 stepping stone이라고 잘못 판단하는 false positives와 stepping stone임에도 불구하고 탐지해 내지 못하는 false negatives 측정) 하기 위해서는 절대적인 레퍼런스 데이터를 확보해야만 한다. 본 논문에서는 레퍼런스 데이터를 확보할 수 있는 자동화된 도구를 제공하는 것에 초점을 맞춘다. 예를 들어, 하나의 리눅스 서버에서 인터렉티브 서비스에 의한 stepping stone 역할을 진행하는 관련 프로세스들에 대한 구체적인 레퍼런스 데이터가 있다면 네트워크-기반 역추적 기법을 검증하기 위한 가장 절대적인 자료가 될 것으로 기대한다.

서론에 이어, 제2장에서는 공격자 근원지 역추적 기술인 호스트-기반 접근과 네트워크-기반 접근 방법에 대해 간략히 언급하고, 제3장에서는 본 논문에서 제안하는 인터렉티브 서비스 원격 접속 쉘 프로세스의 시스템 콜을 전수 조사하여 stepping stone 여부를 실시간 자동 검출하는 brute-force 기법[1]을 제안한다. 제4장에서는 제 3장에서 제안된 알고리즘을 CentOS 6.5 (x86_64)에서 구현하기 위한 자세한 프로그래밍 방법을 설명하고 실제 네트워크에서 stepping stone 관련 실험을 수행한 결과를 기술하였다. 마지막으로 제5장에서 결론 및 향후 연구 방향에 대해 기술하였다.

II. 관련 연구

1. 호스트-기반 역추적 기술

호스트-기반 역추적 기술은 역추적이 이루어지는 영역 내 모든 호스트들이 직접 추적에 개입하는 방법을 채택하고 있다. 그러나 각 호스트에서 관찰된 정보들을 분석하는 방법에 있어서는 중앙집중식(centralized)과 분산구조(distributed)로 구분할 수 있다. UC Davis에서 개발된 DIDS (Distributed Intrusion Detection System)[1]는 중앙집중식 구조를 이용하는 반면, 서울대에서 개발된 CIS (Caller Identification System)[2]는 분산구조를 채택하고 있다.

DIDS는 감시 영역 내에 포함된 호스트들은 최초 로그인 이벤트가 발생되면 〈session_start, user-id, host-id, time〉으로 구성되는 하나의 4-tuple 사용자 인스텐스(instance)가 생성된다. 이들 이벤트는 그림 1에서 보는 바와 같이 중앙에 위치한 DIDS director로 전달되고 해당 사용자에 대한 하나의 유일한 NID(Network-user identification)가 생성된다. 만약 이 사용자가 유닉스의 su 명령어를 이용해 자신의 identity를 변경하여도 중앙에 위치한 DIDS director는 여전히 최초 생성된 동일한 NID를 사용하게 된다. 물론 이 사용자가 다른 호스트로 원격 로그인을 수행하여도 DIDS director는 최초 생성된 동일한 NID를 사용하게 된다. 예를 들어, 그림 1의 위쪽에서 보는 바와 같이 3개의 호스트로 이루어진 연결체인 상의 공격 근원지 kims@host1가 로컬 로그인되면 로그인 이벤트가 DIDS director로 전달되고 해당 사용자에 대한 NID가 생성된다. 한편, 원격로그인을 이용하여 parks@host2에 접근하고 다시 lees@host3로 원격 접속하는 이벤트가 발생되면 이들 이벤트 역시 모두 DIDS director로 전송된다. 이때 DIDS director는 이들 세 로그인 이벤트들을 모두 하나의 동일한 NID로 인식하게 되어 자연스럽게 DIDS director는 실시간으로 연결체인을 검출할 수 있게 된다. 한편, 그림 1의 아래 부분은 분산처리방식을 채택하고 있는 CIS의 동작 과정을 간략히 보여준다. CIS는 각 호스트 내에 Caller Identification 서버 (CIS) 프로그램이 동작한다. 예를 들어, 그림 1에서 kims@host1과 먼저 연결 체인을 형성한 parks@host2는 해당 연결이 kims@host1과의 연결임을 자신의 CIS에 유지한다. 이후 parks@host2가 lees@host3로 원격접속이 이루어질 때, 그림 1에서 보듯이 lees@host3는 parks@host2로 인증요구(authentication request)를 요청하고 parks@host2는 자신이 kims@host1으로부터 연결된 사실을 인증응답 메시지에 실어 응답한다. lees@host3는 kims@host1으로 다시 인증요구를 요청하여 kims@host1이 parks@host2로 연결한 사실이 있는지 최종확인하게 된다.

호스트-기반 역추적 기술은 중앙집중식 구조이든 분산 구조이든 상관없이 현재 형성된 연결체인상에 존재하는 호스트들이 추적을 위한 역할을 정상적으로 수행하여야만 가능한 방법이다. 따라서 이들 호스트 중 하나라도 잘못 동작한다면 전

1) brute-force 기법은 컴퓨터의 컴퓨팅 능력을 이용하여 모든 가능성을 전수 조사하여 문제를 해결하는 다소 투박한 기법을 일컫는 용어이다. 본 논문에서는 문제 해결을 위해 관련 시스템 콜을 전수 조사하는 방법을 채택하였다.

체 시스템 동작에 문제를 일으키게 된다. 따라서 이와 같은 호스트-기반 역추적 기술을 실제 대규모 인터넷 망에 구현해 사용하기에는 현실적으로 불가능한 것으로 판단된다. 이러한 이유들로 인해 1990년 초반에 이루어진 일부 연구결과 이외에 오늘날에는 거의 관련연구가 이루어지고 있지 않는 실정이다. 호스트-기반 역추적 기술은 중앙집중식 구조이든 분산 구조이든 상관없이 현재 형성된 연결체인상에 존재하는 호스트들이 추적을 위한 역할을 정상적으로 수행하여야만 가능한 방법이다. 따라서 이들 호스트 중 하나라도 잘못 동작한다면 전체 시스템 동작에 문제를 일으키게 된다. 따라서 이와 같은 호스트-기반 역추적 기술을 실제 대규모 인터넷 망에 구현해 사용하기에는 현실적으로 불가능한 것으로 판단된다. 이러한 이유들로 인해 1990년 초반에 이루어진 일부 연구결과 이외에 오늘날에는 거의 관련연구가 이루어지고 있지 않는 실정이다.

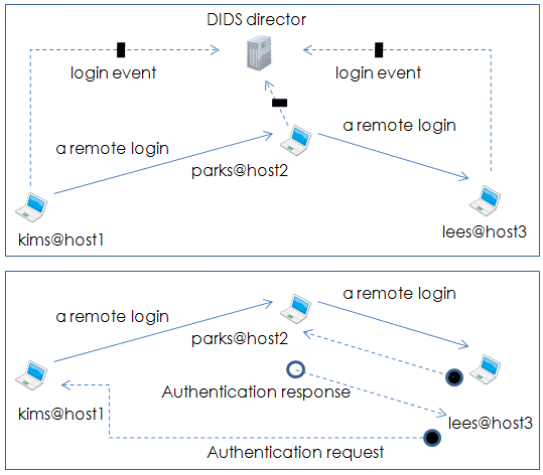

그림 1. 호스트-기반 역추적 기술 예: (i) 중앙집중식방식 - DIDS (위쪽), (ii) 분산처리방식 - CIS (아래)
Fig. 1. An example of the host-base traceback technology: (i) The centralized method - DIDS (upper of the figure), (ii) The distributed method - CIS (the bottom of the figure))

2. 네트워크-기반 역추적 기술

네트워크-기반 역추적 기술은 연결체인 상의 호스트들의 개입을 전혀 요구하지 않는다. 이들 새로운 기법들은 연결체인상에 유지되고 있는 트래픽의 다양한 성질들을 이용한다. 이 분야의 최초 연구자인 Staniford-Chen과 Heberlein[3]은 연결을 요약할 수 있는 적은양의 정보를 지문(thumbprint) 형태로 만들고 이를 이용한 상관관계(correlation relationship)가 있는 연결을 판정해 연결 체인을 찾는 방법을 제안하였다. 그러나 연결의 콘텐츠를 이용한 지문 방법은 암호화된 연결을 대상으로는 전혀 동작하지 못하는 단점이 있다. 따라서 Zhang과 Paxson[4]은 인터렉티브 트래픽의 OFF 기간(휴지 시간)의 끝 시간 (혹은 ON 기간(버스트 시간)의 시작 시간)의 타이밍 상관관계를 이용한 stepping stones 검출 기법을 제안하였다. 예를 들어, 두 개의 연결 C_1과 C_2의 OFF 기간이 비슷한 시점에 끝나는 횟수가 일정 값 이상이 되면 두 연결은 동일한 연결체인 상에 있는 것으로 판단한다. 한편 Wang et. al[6]은 트래픽 스트림의 inter-packet 지연(IPD: Inter-packet Delay)을 이용한 타이밍 정보를 활용하여 연결체인을 검출하는 방법을 제안하였다. 즉 두 연결 IPD의 상관관계가 일정한 값 이상이 되면 두 연결은 동일한 연결체인 상에 존재하는 것으로 판단한다. 이상의 두 가지 방법 이외에도 네트워크-기반 역추적 기술은 연결 트래픽 스트림의 다양한 특징을 네트워크에 분산 배치된 모니터링 장치를 통해 관찰하고 이들 정보를 기반으로 연결들 간의 상관관계 정도를 계산하여 stepping stones 여부를 결정하게 된다.

네트워크-기반 역추적 기술은 인터넷 상에 트래픽 모니터링 장치를 배치하여 연결 트래픽 스트림의 특징을 이용하기 때문에 호스트-기반 접근 방법과 비교해 실제 인터넷 망에 적용 가능한 기술이라고 판단된다. 하지만, 이들 방법들은 호스트-기반 역추적 기술과는 달리 각 호스트의 정확한 로그(log) 정보를 이용하지 않기 때문에 당연히 오탐율이 존재한다. 따라서 제안된 네트워크-기반 역추적 기법의 정확성을 검증하기 위해서는 해당 호스트의 stepping stone 여부를 판단할 수 있는 확실한 레퍼런스 데이터를 제공할 수 있어야 한다. 본 논문에서는 기존의 호스트-기반 역추적 기술과 같이 시스템 차원의 매우 복잡한 형태의 방법이 아니라 네트워크-기반 역추적 기법을 검증할 수 있는 수준의 매우 간단한 방법을 제안하는 것이다. 즉 네트워크-기반 역추적 기술에 의해 stepping stone으로 검출된 하나의 리눅스 서버가 있다고 가정하고, 이 서버가 stepping stone 연결을 제공하고 있는지 100% 확신할 수 있는 레퍼런스 데이터를 제공할 수 있는 brute-force 기법을 제안하는 것이다.

III. Stepping Stone 진단을 위한 Brute-force 기법

1. 원격접속을 위한 인터렉티브 서비스

비록 특정 응용 프로그램 혹은 서비스를 위한 클라이언트

/서버 프로그램들이 인터넷에서 동작하지만, 사용자들은 원격 컴퓨터에 직접 접근해 응용 프로그램들을 사용할 수 있는 일반적인 용도의 클라이언트/서버 프로그램을 선호한다. 즉 사용자들이 원격 컴퓨터에 로그인할 수 있도록 허락하고 일단 로그인에 성공하면 원격 컴퓨터에서 가용한 허락된 모든 서비스들을 이용할 수 있으며 그 결과들을 다시 로컬 컴퓨터에서 확인할 수 있는 인터렉티브 서비스가 사용되고 있다. 이러한 인터렉티브 서비스 중 가장 흔히 사용되는 것은 텔넷(Telnet)과 SSH(Secure Shell)이다[7]. 최근 보안에 대한 중요성이 한층 강조됨에 따라 암호화되지 않은 plain-text 형태의 연결을 제공하는 텔넷은 더 이상 서비스하지 않는 서버들이 대부분이다. 따라서 암호화 연결을 제공하는 SSH 서비스를 이용하는 것이 오늘날 일반적 추세이다. 그러나 보안에 대한 인식이 부족하거나 텔넷 서비스 제공이 불가피한 서버들이 존재하고 있음은 부정할 수 없는 현실이다.

리눅스 서버에서 텔넷, rlogin, 그리고 ftp와 같은 서비스들은 웹 클라이언트가 자주 요청해 오는 웹 서비스(httpd)와는 달리 사용자들이 아주 가끔 사용하는 서비스 유형이다. 따라서 리눅스는 시스템 부하를 줄이기 위해서 이들 서비스에 필요한 개별 서비스 데몬(deamon) 프로세스를 항상 구동하는 것은 아니다. 즉 해당 클라이언트의 요청이 들어오면 xinetd 이라고 불리는 최상위(super) 데몬이 이들 클라이언트의 서비스 요청에 먼저 응답하여 접속권한 체크 등을 수행한 후에 정당한 접속일 경우에만 해당 서비스 데몬을 구동시킨다[8]. 그림 2는 텔넷과 rlogin 서비스가 요청되면 시스템에서 xinetd 프로세스로부터 해당 서비스 데몬 및 연결된 쉘(shell) 프로세스를 할당하기 까지 만들어진 프로세스들을 계층화(부모-자식 프로세스 관계)시켜서 보여준다. 리눅스 서버의 SSH 서비스 제공을 위한 sshd 데몬 역시 xinetd 에 등록하여 텔넷 서비스 데몬의 동작원리와 동일하게 서비스 요청시 구동시킬 수 있지만, 그림 2에서 보듯이 sshd 데몬을 standalone 형태로 동작시키는 경우도 가능하다. 그림 2에서와 같이 sshd으로부터 해당 SSH 연결을 위한 쉘 프로세스가 할당되는 프로세스까지 확인할 수 있다. 만약 sshd 데몬을 xinted 에 등록해 사용한다면, 그림 2에서 xinetd 의 자식 프로세스로 sshd 가 생성될 것이며 그 밑으로 쉘 프로세스가 생성될 것으로 예상된다.

한편, 그림 2에서 보듯이 리눅스 서버에서 인터렉티브 서비스를 이용한 원격접속이 이루어지는 여부를 정확하게 진단하는 방법은 앞에서 설명한 xinetd 혹은 sshd로부터 쉘 프로세스가 생성되는 전 과정을 정확히 추적하면 가능한 것으로 결론 내릴 수 있다. 본 논문에서는 이와 같은 brute-force한 방법을 사용할 것이다.

2. 인터렉티브 서비스 관련 프로세스와 시스템 호출

유닉스 계열의 다른 운영체제들과 마찬가지로 리눅스는 모든 디바이스를 파일과 동일한 방법으로 취급하고, 디바이스와 파일 모두에 대해 투명하게 접근할 수 있는 인터페이스로 가상 파일시스템(VFS: Virtual File System)을 제공한다[8-9]. 인터렉티브 서비스를 이용해 원격 로그인된 가상 터미널 역시 /dev/pts 디렉토리 아래 파일의 형태로 표현되며 운영체제는 디바이스 인터페이스 공통된 기본 동작인 read() 와 write() 시스템 호출을 통해 가상 터미널과 통신하게 된다. 그림 3에서, SSH 연결 형성 시 관련 프로세스 생성 과정을 write() 시스템 호출을 관찰하면 보다 자세하게 확인할 수 있다. 리눅스는 시스템 호출을 관찰할 수 있는 strace라고 불리는 유틸리티를 제공한다[9].

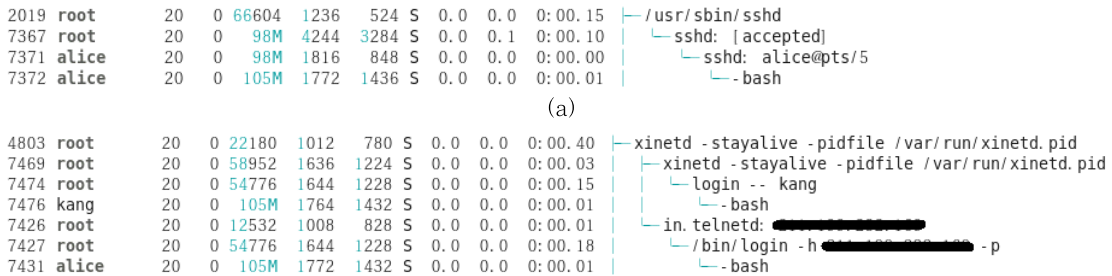

그림 2. 인터렉티브 서비스 클라이언트 (텔넷, rlogin, 그리고 ssh) 처리 과정의 프로세스 트리
Fig. 2. Process tree on the Linux server after the client requests for Interactive Services such as Telnet, rlogin, and ssh

그림 3은 윈도우즈 호스트에서 putty를 사용한 클라이언트로부터 리눅스 서버가 SSH 연결 요청을 수신할 때 발생되는 write() 시스템 호출을 strace를 이용해 관찰할 결과를 보여 준다. 논문 지면 수를 줄이기 위해 설명에 필요한 일부 write() 시스템 호출만 그림 3에 보여준다. 예를 들어, 키(key) 교환을 위한 프로세스 생성 및 관련 write() 시스템 호출은 생략되었다.

```
[root@localhost shellProg]# strace -f -etrace=write -s 64 -p 2019
Process 2019 attached - interrupt to quit
Process 26592 attached (waiting for parent)
Process 26592 resumed (parent 2019 ready)
[pid  2019] write(8, "\0\0\2;\0", 5)     = 5
[pid  2019] write(8, "\0\0\0022\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nProtocol
2\n\n\n\n\n\n\n\n\n\n\n\n\n\nSyslogFacility "..., 570) = 570
[pid 26592] write(10, "0\n", 2)          = 2
[pid 26592] write(3, "SSH-2.0-OpenSSH_5.3\r\n", 21) = 21
```

(a) 새로운 서비스 프로세스 생성

```
Process 26629 attached
[pid 26629] write(6, "unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023\0",
54) = 54
[pid 26629] write(3, "\31\227\313\276\231\275\246zR\211\17\216[5\6m?P\37\23\371`
\367\256z_\276\371\330\232\366\271\26\324\242\276=\316\2\301\235\341\277\2750yy'
-\324\220\3707\257\26\347\362[\370\35\230\7I7", 64) = 64

[pid 26592] write(6, "\0\0\0\1\0\0\0\n/dev/pts/5\0\0\0;Last login: Sun Dec 28 14
:35:18 2014 from "..., 81) = 81
```

(b) ssh 원격 접속 처리

```
Process 26630 attached
[pid 26629] write(3, "\246\377r\26\212\340(\34\33\6\324\314\n\37u\35\347\20S\33
2\217\332qP\222\t\363~Tsn\2156'E\2018\202_\332\270\236\356E\34|lD\\\232KX\211\17
\2637eN\354\361\325\317:/"..., 160) = 160
[pid 26630] write(1, "Last login: Sun Dec 28 14:35:18 2014 from [redacted]169\
r\n", 59) = 59
[pid 26629] write(3, "\tn\24hm\334\261Kw\271\314\370I54\33\23\253\0039\233\346Y\
200hW\6\217\324\206\34\363T\26\333\312\237\255\233\276{(\10\r\223\215\313Mkw\375
~\253\254&\21\237\317\2147S\r\326\346"..., 112) = 112
```

(c) 원격로그인 쉘 생성

그림 3. strace 유틸리티를 사용한 ssh 클라이언트 접속에 따른 프로세스 추적 예

Fig. 3. An example of the process tracing during an ssh client connection by the strace utility

그림 3(a)에서 보듯이, 클라이언트로부터 SSH 연결 요청이 수신되면 서버는 해당 연결을 처리하기 위해 새로운 프로세스 PID 26592가 생성되고 이 프로세스는 SSH 프로토콜 처리를 위한 작업을 시작하게 된다. 한편 ssh 원격 접속을 위한 pts 처리를 위한 새로운 프로세스 PID 26629가 생성되고 파이프(pipe) 와 소켓(socket)을 이용한 사용자 및 패스워드 인증을 위한 내부 및 외부 통신이 시작되는 것을 그림 3(b)를 통해 확인할 수 있다. 마지막으로 새로운 사용자 쉘 프로세스 PID 26630이 생성되고 생성된 쉘 프로세스를 시작으로 앞에서 설명한 가상 터미널 프로세스 (PID 26629)를 통해 클라이언트로 메시지가 전달되는 과정을 그림 3(c)에서 확인할 수 있다. 텔넷과 rlogin과 같은 다른 인터렉티브 서비스들도 앞에서 설명한 ssh와 동일한 방법으로 분석하면 그림 2에서 생성된 관련 프로세스들과 시스템 호출들을 정리할 수 있다.

3. 기본 알고리즘

본 논문에서 제시하게 될 stepping stone 자가 진단 brute-force 기법은 다음과 같은 요구조건을 만족시킨다.

- 클라이언트가 접속해 사용하는 쉘의 종류에 무관
- 동시에 여러 stepping stone 접속을 처리
- Stepping stone 진단 시, 네트워크를 통해 접속된 호스트 (근원지)의 접속 시간과 IP 주소, 그리고 계층 4 포트 번호와 네트워크를 통해 외부 호스트로 접속한 호스트(목적지)의 접속 시간과 IP 주소, 그리고 계층 4 번호를 실시간 자동으로 로그파일 형태로 저장하여 레퍼런스 데이터로 활용

기본적으로 하나의 리눅스 서버가 stepping stone으로 이용되기 위해서는 네트워크를 통해 원격 접속이 이루어져야 한다. 따라서 원격 접속이 가능한 사용자 계정이 존재하며 해당 사용자 계정의 쉘은 매우 다양할 수 있다[8]. 사용자 쉘의 종류에 따라서 시스템 콜의 패턴이 조금 상이한 것으로 조사되었다 (상기 그림 3은 CentOS 6.5의 기본 쉘로 사용되고 있는 Bash(Bourne Again Shell) 쉘을 사용했을 경우에 대한 실험 결과임). 따라서 본 논문을 통해 제안될 기법은 이들 사용자 쉘에 무관하게 동작할 수 있어야 할 것이다. 뿐만 아니라, 동시에 여러 개의 stepping stone 연결 시도를 검출할 수 있도록 구현되어야 한다.

이러한 요구사항을 만족할 수 있도록 본 논문에서는 그림 4와 같이 2단계 - 1단계: 원격 인터렉티브 서비스 클라이언트 접속 검출 단계, 2단계: 원격 접속 쉘 프로세스 관찰을 통한 stepping stone 검출 단계 - 구조를 제안하며, 특히 2단계는 스레드(Thread) 프로그램을 사용해 동시에 여러 원격 접속 연결 프로세스를 관찰함으로써 stepping stone 여부를 진단하는 brute-force한 기법을 제안한다. 그림 4에서 보듯이, 1단계에서는 관찰하고자 하는 인터렉티브 데몬을 등록하고 이들 데몬의 시스템 콜을 관찰하여 원격 로그온을 무한 루프를 돌면서 검출하게 된다. 만약 1단계에서 외부로부터 새로운 원격 접속이 검출되면 그림 4와 같이 새로운 스레드를 생성하여 원격로그인 쉘 프로세스를 찾아내고 이 프로세스로부터 다시 외부로 인터렉티브 서비스를 연결하는 행위 즉 stepping stone 행위를 검출하여 해당 정보를 로그메시지로 기록하게 된다. 한편, 외부로부터 원격 로그온이 여러 개 동시에 발생될 경우 그림 4의 2단계 과정은 여러 개 생성되어 병렬 처리된다.

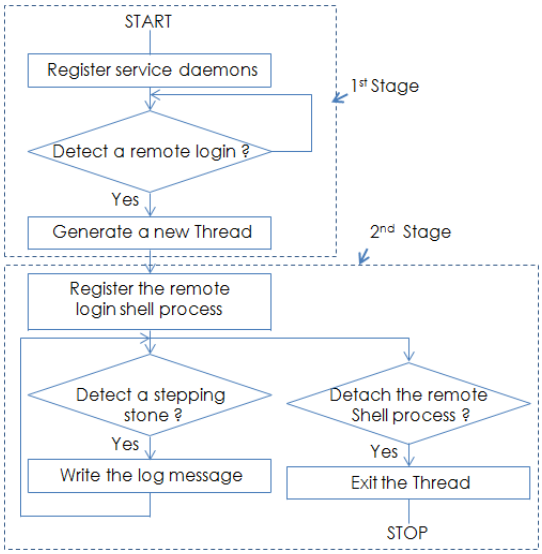

그림 4. 기본 알고리즘
Fig. 4. Basic algorithm

```
1  int main(int argc, char *argv[])
2  {
3     pid_t daemon_pid, c_pid;
4     struct user_regs_struct regs;
5     pthread_t *tid;
6     int toggle=0;
7     long scno;
8
9     ptrace(PTRACE_ATTACH, daemon_pid, NULL, NULL);
10    while(true) {
11       wait(&status);            // wait the attached deamon system calls
12       ptrace(PTRACE_GETREGS, daemon_pid, NULL, &regs);
13       scno = regs.orig_rax; // get the system call number
14       if (scno == SYS_fork || scno == SYS_clone) {
15                if (toggle ==  0)
16                   toggle=1;
17                else {
18                   toggle=0;
19                   ptrace(PTRACE_GETGEGS, daemon_pid, NULL, &regs);
20                   c_pid = regs.rax; // get the return value from system call
21                   pthread_create(tid, NULL, &REMOT_Shell_Module, &c_pid);
22                }
23       } // end of SYS_clone
24       ptrace(PTRACE_SYSCALL, daemon_pid, NULL, NULL);
25    } // end of while
26 } // end main
27 // Now a new pthread
28 REMOT_Shell_Module (process ID)
29 {
30    do SHELL_Script_Module (process ID);
31    pthread_exit;
32 } // end module
```

그림 5. ptrace 시스템 콜을 이용한 인터렉티브 서비스 데몬에 새로운 클라이언트 접속 검출 (기본 알고리즘의 1단계 프로그램 예)
Fig. 5. Detection of a new client connection to the interactive service daemon using the ptrace system call (An example program of the first step for the basic algorithm)

IV. 구현 및 실험 결과

1. 구현

제3장에서 제안된 기본 알고리즘을 구현하고 실험하기 위해 본 논문에서는 CentOS 6.5 (x86_64) Quad Core i5-2300 CPU(2.8Ghz) 메모리 5.6GiB 컴퓨터를 사용하였다. 구현은 그림 4에서 설명한 바와 같이, 1단계와 2단계로 나누어 프로그램 하였으며 최대한 가볍게 구현하기 위해 많은 노력을 기울였다. 특히 시스템 콜 추적 과정에서는 write 시스템 콜만 사용해 구현하였다.

시스템 콜은 개념상으로 함수 호출과 유사하지만 사용자 모드에서 커널 모드로 전환하여 실제 시스템 콜 명령을 실행한다는 차이점이 있다[9]. ptrace 시스템 콜은 하나의 프로세스가 다른 프로세스의 실행을 제어할 수 있도록 도와주며 필요시 다른 프로세스의 코어 이미지까지 변경할 수 있도록 한다. 따라서 리눅스 관련 시스템에서 ptrace는 디버깅 용도로 꾸준히 사용되어 왔다[9]. 본 논문에서는 그림 5와 같이 ptrace 시스템 콜을 이용해 등록된 데몬 프로세스에 새로운 외부 접속 클라이언트 생성 프로세스 번호를 확인하는 과정을 제3장에서 설명한 기본 알고리즘의 1단계에 구현하였다 (그림 5는 알고리즘 구현을 설명하기 위해 필요한 일부 코드만 나타내었으며 오류 처리 및 기타 자세한 코드 부분은 생략하였음을 밝힌다).

그림 5의 주요 라인을 설명하면 다음과 같다.

- 라인 9: ptrace 시스템 콜을 이용해 인터렉티브 서비스 데몬 프로세스 등록
- 라인 11: 등록된 프로세스에서 발생되는 시스템 콜 이벤트를 대기
- 라인 12, 13: 발생된 시스템 콜 획득
- 라인 14: 발생된 시스템 콜이 새로운 프로세스를 생성하는 것인지 판단
- 라인 19, 20: fork 와 clone 시스템 콜의 리턴값 즉 새로이 생성된 프로세스 ID를 획득
- 라인 21: 알고리즘 2단계 진입을 위한 새로운 스레드 생성
- 라인 24: 멈춤 상태의 데몬 프로세스를 계속 진행
- 라인 30: stepping stone 검출을 위한 쉘 스크립트 REMOTE_Shell_Module 실행

그림 5에서 (알고리즘 1단계) 하나의 스레드가 생성되면 알고리즘 2단계 수행을 위한 그림 6과 같은 쉘 스크립트를 수행하게 된다. strace 유틸리티[9]는 리눅스에서 사용 가능한 가장 강력한 시스템 콜 추적 도구로 이용되어 왔다. 본 연구에서도 그림 6에서 보듯이, 1단계에서 새롭게 생성된 원격접속 클라이언트 처리를 위한 프로세스 ID를 입력받아 strace의 -f 옵션 (자식 프로세스를 포함한 모든 프로세스 추적) 그

리고 -etrace=write 옵션 (write 시스템 콜만 추적)을 사용하여 시스템 콜 추적을 시작한다(그림 6의 라인 3 참조). 이러한 strace를 이용한 프로세스 추적은 해당 원격 접속 쉘 프로세스가 종료 (그림 6의 라인 44) 될 때 까지 while 루프 (그림 6의 라인 3)를 돌면서 계속 수행된다.

```
1  // shell script
2  // SHELL_Script_Module (process ID)
3  Strace –f –etrace=write –s 64 –p $1 2>&1 | while read WRITESYS
4  do
5     MATCH_ID=$(echo $WRITESYS 2>&1 | grep 0pts)    # check the
6                                                    # fingerprint
7     if [ \( -n "$MATCH_ID" \) –a \( $MODE = 0 \) ]; then
8        MATCH_ID_PID=$1{WRITESYS%%]"}
9        MATCH_ID_PID=${MATCH_ID_PID#[pid}          # find the process ID
10       if [ $1 = $MATCH_ID_PID ]; then
11             MODE=1
12             TIME_IN=$(echo $(date))
13       fi
14    fi          # end MODE=0
15    if [ $MODE = 1]; then
16       SH_ID =$(echo $WRITESYS 2>&1 | grep Process | grep attached)
17       if [ -n "SH_ID" ]; then
18             SH_PID=`echo "SH_ID" | awk '{print $2}'`
19             echo "SHELL PID >>> $SH_PID"
20       fi
21    fi          # end MODE=1
22    if [ $MODE = 2]; then
23       STEP_ID=$(echo $WRITESYS 2>&1 | grep SSH)
24       if [ -n "$STEP_ID" ]; then
25             STEP_ID_PID =$1{WRITESYS%%]"}
26             STEP_ID_PID =${MATCH_ID_PID#[pid}  # find the process ID
27             TIME_OUT =$(echo $(date))
28             nestat -antp | while read NETINFO
29             do
30                STEP_IN=$(echo $NETINFO | grep $MATCH_ID_PID)
31                if [ -n "STEP_IN" ]; then
32                   echo "Inbound: $TIME_IN: $STEP_IN"
33                fi
34                STEP_OUT=$(echo $NETINFO | grep $MATCH_ID_PID)
35                if [ -n "STEP_OUT" ]; then
36                   echo "Outbound: $TIME_OUT: $STEP_OUT"
37                fi
38             done
39       fi
40    fi
41    if [ $MODE = 2 ]; then
42       MATCH=$(echo $WRITESYS 2>&1 | grep detached | grep $SH_PID)
43       if [ -n "MATCH" ]; then
44             break
45       fi
46    fi
47 done
```

그림 6. strace 유틸리티를 이용한 stepping stone 검출 쉘 스크립트 (기본 알고리즘의 2단계 프로그램 예)
Fig. 6. A shell script to detect stepping stones using the strace utility (An example program of the second step for the basic algorithm)

그림 6의 쉘 스크립트는 크게 4부분으로 구성된다(쉘 스크립트의 각 명령어 해석은 참고문헌 [11]을 참고하면 된다). 이것은 그림 7에서 보듯이 Mode에 따른 상태 천이에 따라 추적의 대상을 구분한 것이다. 즉 Mode=0에서는 가상 터미널(pts) 프로세스 ID를 검출하고, Mode=1에서는 원격접속에 의해 생성되는 쉘 프로세스 ID를 검출하고, Mode=2에서는 검출된 쉘 프로세스의 자식 프로세스들로부터 외부 호스트로 인터렉티브 서비스 원격접속을 시도하는 이벤트를 검출하여 그 결과를 로그 메시지로 출력하며, 동시에 Mode=2에서는 외부로부터 접속된 쉘 프로세스가 종료되면 해당 쉘 스크립트를 종료하는 작업을 수행하게 된다.

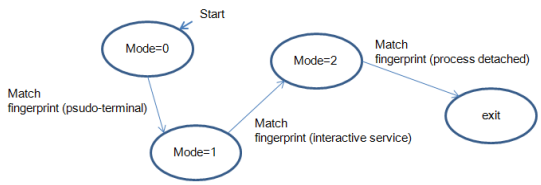

그림 7. Mode 변환에 따른 상태 천이
Fig. 7. State Transition Diagram of Mode

그림 8은 원격접속 쉘 프로세스 ID 27837이 최초 생성되는 과정을 시스템 콜 write 메시지를 통해 관찰한 그림이다. 그림 8의 첫 번째 라인에서 보듯이 외부 접속에 의한 가상터미널 0pts/5를 확인할 수 있다. 한편, 숫자 '5'는 해당 가상터미널 번호에 해당한다. 따라서 원격접속 처리를 위한 새로운 프로세스가 가상터미널을 잡고 이후 생성되는 최초 프로세스가 원격접속 쉘 프로세스가 되는 것을 최종 확인할 수 있다. 그림 6의 라인 5와 라인 16에서는 이들 핑거프린트 (라인 5의 0pts, 라인 16의 Process attached)를 사용하여 각각 구현하였다.

이제 원격접속 쉘 프로세스가 확인되면, 해당 프로세스로부터 다시 외부로 접속을 시도하는 모든 이벤트들을 검출하게 된다. 이러한 작업은 그림 6의 라인 23을 통해 수행된다(본

```
[pid 27816] write(4, "\7\0\0\0\250l\0\0pts/5\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0005\0\0\0alice\0\0\0\0\0\0\0\0\0\0\0\0\0\0". , 384) = 384
[pid 27816] write(1, "Last login: Sun Dec 28 21:32:57 from 211.[redacted]169\n", 5
3) = 53
[pid 27816] write(4, "\200\372\237Tpts/5\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\000211.199.232.169\0\0\0\0\0\0\0\0\0\0\0\0\0\0". . . , 292) = 292
[pid 27815] write(0, "Last login: Sun Dec 28 21:32:57 from 211.[redacted]169\r\n",
 54 <unfinished . . .>
[pid 27816] --- SIGHUP (Hangup) @ 0 (0) ---
[pid 27815] <... write resumed> )       = 54
[pid 27816] --- SIGCONT (Continued) @ 0 (0) ---
Process 27837 attached
```

그림 8. 원격접속 쉘 프로세스
Fig. 8. Remote Login Shell Process

예제 쉘 스크립트에서는 ssh 연결만 나타내었다). 한편 stepping stone이 검출되면 그림 6의 라인 28에서 보듯이 네트워크 연결정보를 보여주는 netstat의 -antp 옵션을 사용하여 stepping stone 입.출력에 관련된 연결 정보를 연결 시간과 함께 출력하도록 한다. 마지막으로 mode=1에서 검출된 원격접속 쉘 프로세스 (그림 6의 SH_PID) 가 종료되면 (그림 6의 42라인) 쉘 스크립트를 종료하게 된다.

2. 실험 결과

제안된 알고리즘과 이를 적용한 구현을 직접 검증하기 위해 그림 9와 같은 아령(dumbbell) 구조의 실험 시나리오를 구성하였다. 그림 9 실험 시나리오에서 윈도우즈 7 호스트(211.x.x.169)는 원격접속 프로그램 putty를 사용하여 리눅스 서버(211.x.x.184)로 ssh 원격 접속을 하고 이를 경유하여 리눅스 서버 호스트(211.x.x.182)로 ssh 원격 접속을 시도하였다(이하 시나리오 1이라고 약칭).

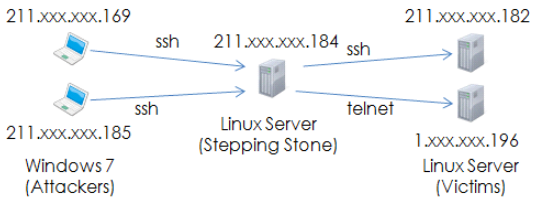

그림 9. 실험 시나리오
Fig. 9. Test Scenario

한편 이와 같은 연결 체인이 이루어지는 도중에 그림 9에서 보듯이, 또 다른 호스트(211.x.x.185)가 stepping stone 211.x.x.184를 경유해 리눅스 서버 1.x.x.196으로 텔넷 원격 접속을 시도하였다(이하 시나리오 2라고 약칭). 이러한 시나리오 구성은 제안된 방법 및 구현이 동시에 여러 개의 연결시도를 병렬로 처리 가능함을 보이면서 다양한 인터렉티브 서비스에 대해 적용 가능함을 검증하기 위함이다.

그림 10은 실험 시나리오를 통해 출력되는 로그메시지를 실시간으로 캡처한 화면이다. 한편 그림 11은 리눅스에서 사용하는 텍스트 모드 프로세스 뷰어 htop[10]을 사용하여 실험 시나리오에서 발생되는 프로세스를 실시간 관찰하여 캡쳐한 화면이다. 이와 같이 그림 10과 그림 11을 상호 비교 분석하면 본 논문을 통해 구현된 기법이 관리자의 개입 없이 자동으로 stepping stone을 실시간 검출 가능함을 확인할 수 있다.

그림 10에서 시나리오 1 (혹은 시나리오 2) 에 의해 새로이 생성된 ssh 원격접속 클라이언트 처리 프로세스 ID 5657 (혹은 프로세스 ID 7083)을 검출해 알고리즘 2단계 과정인 스레드를 생성하는 (그림 10의 I am threading processing: 5657, 7083) 것을 확인할 수 있다. 이러한 실험 결과는 그림 11의 htop을 관찰해 트리모양의 프로세스 구조를 통해 보다 정확하게 검증할 수 있다. 즉 PID=7083과 PID=5657 (그림 11의 Command: sshd: [accepted]), 그리고 알고리즘 2단계 쉘 스크립트 프로세스 PID=7085와 PID=5659를 각각 그림 11에서 확인할 수 있다. 한편 원격접속에 의해 새롭게 생성된 쉘 프로세스 PID=5926과 PID=7315를 그림 10 (SHELL PID 〉〉〉 5929, 7315)과 그림 11 (bash 프로세스)에서 각각 확인할 수 있다. 최종적으로 그림 10을 통해 시나리오 1과 시나리오 2에 의한 Inbond와 Outboud의 쌍으로 출력되는 stepping stone 연

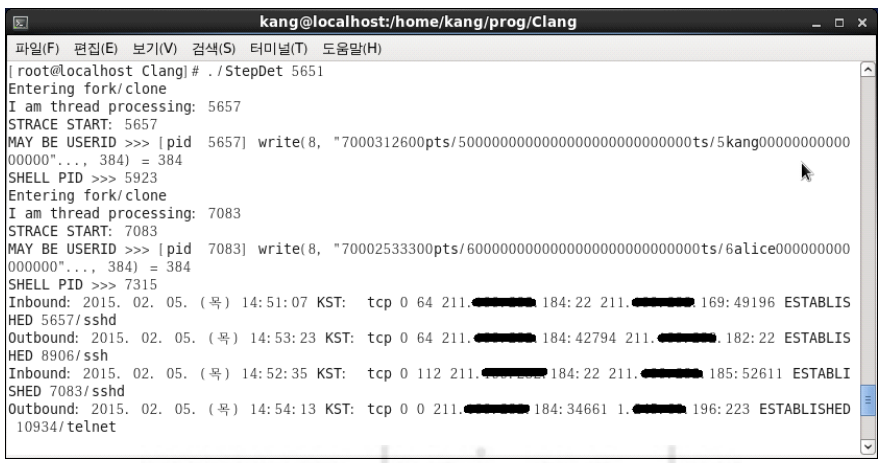

그림 10. 실험 시나리오 결과
Fig. 10. Result of Test Scenario

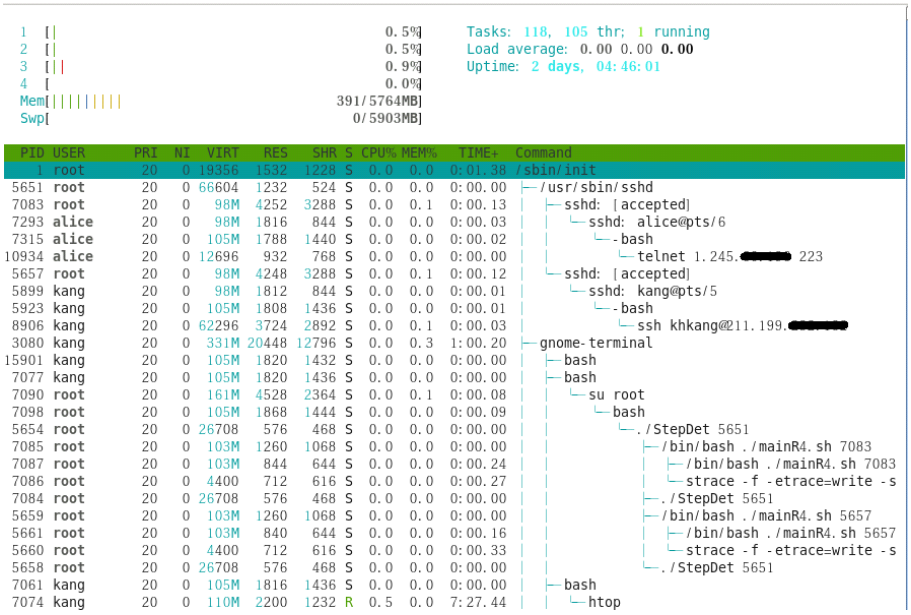

그림 11. htop 유틸리티를 이용한 프로세스 생성 관찰
Fig. 11. Monitoring the Process by htop

결체인 정보를 획득할 수 있다. 물론 이러한 외부 연결 접속 쉘 프로세스는 그림 11의 프로세스 트리를 통해서도 확인할 수 있다.

V. 결 론

최근 사이버테러에 대한 전 세계적인 관심과 함께 인터넷 상에서 공격의 근원지를 역추적하는 다양한 기술이 소개되고 있다. 특히 이들 역추적 기술 중 인터렉티브 서비스를 이용한 stepping stone을 경유하는 연결 체인을 추적하는 네트워크 기반 연구결과들에 대해 많은 관심을 보이고 있다. 이러한 네트워크 기반 역추적 기술의 오탐율 관련 성능을 검증하기 위해서는 절대적인 레퍼런스 데이터 확보가 필요하다. 본 논문에서는 리눅스 서버에서 관련 프로세스의 시스템 콜을 모두 전수 조사하는 brute-force한 방법으로 stepping stone 여부를 자가 진단하는 알고리즘 및 구현 방법을 제안하였다. 본 연구결과를 통해 제공되는 stepping stone 관련 정보는 원격접속 쉘 프로세스의 시스템 콜을 전수 조사하여 그 결과를 제공함으로써 100% 신뢰성을 가진 정보를 제공하게 된다.

한편 본 논문에서 제안된 방법으로 각 호스트에서 검출된 stepping stones 관련 정보를 중앙에 위치한 분석 장치로 전송하여 소규모 도메인 내에서 공격자 근원지 역추적 기술로 활용 가능하다. 그러나 이러한 방법을 대규모 인터넷 망에 적용하기에는 기존 호스트-기반 접근과 마찬가지로 실현 가능성이 부족한 것만은 사실이다.

REFERENCES

[1] S.R. Sanpp, J. Brentano, G.V. Dias, T.L Goan, L.T. Heberlein, C.L. Ho, K.N. Levitt, B. Mukherjee, S.E. Smaha, T. Grance, D.M. Teal, and D. Mansur, "DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and An Early Prototype," In Proceedings of the 14th National Computer Security Conference, pp. 167-176, Oct. 1991.

[2] H.T. Jung, H.L. Kim, Y.M. Seo, G. Choe, S.L. Min, C.S. Kim, "Caller Identification System in the Internet Environment" In Proceedings of UNIX Security Symposium IV, pp. 69-78, Oct. 1993.

[3] S. Staniford-Chen and L.T. Heberlein, "Holding Intruders Accountable on the Internet," In Proceedings of the IEEE Symposium on

Security and Privacy, pp. 39-49, May 1995.
[4] Y. Zhang and V. Paxson, "Detecting Stepping Stones," In Proceedings of the 9th Conference on USENIX Security Symposium, pp. 184-194, 2000.
[5] K. Yoda and H. Etho, "Finding a Connection Chain for Tracing Intruders," In Proceedings of the Computer Security - European Symposium on Research in Computer Security (ESORICS 2000), pp. 191-205, 2000.
[6] X. Wang, D.S. Reeves, and S.F. Wu, "Inter-Packet Delay Based Correlation for tracing Encrypted Connections Through Stepping Stones," In Proceedings of the Computer Security - European Symposium on Research in Computer Security (ESORICS 2002), pp. 244-263, 2002.
[7] B.A. Forouzan, "TCP/IP Protocol Suite" Fourth Edition, McGraw-Hill, pp. 610-629, 2010.
[8] M.G. Sobell, "A Practical Guide to Fedora and Red Hat Enterprise Linux" Prentice Hall, pp. 227-301, 2013.
[9] M. Wilding and D. Behman, "Self-Service Linux: Mastering the Art of Problem Determination" Prentice Hall, pp. 41-88, 2005.
[10] htop, "htop - an interactive process viewer for Linux," http://hishan.hm/htop
[11] A. Robbins and N. Beebe, "Classic Shell Scripting," O'Reilly Media, pp. 109-266, 2005.
[12] K. Kang, "An Implementation Strategy for the Physical Security Threat Meter Using Information Technology," Journal of the Korea Society of Computer and Information, Vol. 19, No. 7, pp. 47-57, July 2014.

저 자 소 개

강 구 홍
1985: 경북대학교
전자공학과 공학사.
1990: 충남대학교
전자공학과 공학석사.
1998: 포항공과대학교
전자계산학과 공학박사
현 재: 서원대학교
정보통신공학과 교수
관심분야: 네트워크 보안
Email : khkang@seowon.ac.kr