네트워크 취약점 검색공격에 대한 개선된 탐지시스템

유 일 선† · 조 경 산††

요 약

본 논문에서는 네트워크 취약점 검색공격에 대한 기존의 탐지알고리즘들이 갖는 문제점을 분석하고 대규모 네트워크에서의 종합적인 탐지 및 대응을 지원하는 개선된 탐지시스템을 제안한다. 가상 공격에 의한 모의 실험을 통하여 제안된 시스템은 소수의 취약점 포트 위주의 공격과 협동공격, 느린 스캔 및 느린 협동공격을 정확히 탐지할 뿐 아니라 에이전트와 서버사이의 유기적인 연동을 통해 보다 종합적이고 계층적으로 공격에 대응함을 검증하였다.

An Improved Detection System for the Network Vulnerability Scan Attacks

You Il-Sun† · Cho Kyungsan††

ABSTRACT

In this paper, an improved detection system for the network vulnerability scan attacks is proposed. The proposed system improves the methodology for detecting the network vulnerability scan attacks and provides a global detection and response capability that can counter attacks occurring across an entire network enterprize. Through the simulation, we show that the proposed system can detect vulnerable port attacks, coordinated attacks, slow scans and slow coordinated attacks. We also show our system can achieve more global and hierarchical response to attacks through the correlation between server and agents than a stand-alone system can make.

키워드 : 침입탐지(intrusion detection), 네트워크 취약점 분석(network vulnerability analysis), 네트워크보안(network security)

1. 서 론

컴퓨터와 정보통신 기술의 발달이 혁명이라고 불릴 만큼 우리의 환경을 발전시킨 반면 시스템 불법 침입, 중요 정보의 유출 및 변경, 컴퓨터 바이러스 및 서비스 거부 공격 등 네트워크 보안 문제점을 발생시키게 되었다. 해커가 성공적으로 특정 네트워크 시스템에 침입하기 위한 전제 조건은 얼마만큼 해당 시스템에 관한 취약점 정보를 파악하고 있는 가이며 이러한 목적으로 공격하기 이전에 시스템에 관한 정보를 수집하는 과정을 침입시도라 한다. 특히 최근에는 네트워크 보안 취약점을 자동으로 검색해주는 SATAN이 처음 공개된 이후, mscan, sscan, nmap 등과 같이 네트워크 침입시도를 강력하게 지원하는 자동화된 공격도구가 계속해서 공개되어 이를 이용한 네트워크 침입시도 공격이 급증하고 있는 실정이다[1-3]. 침입시도 기술은 크게 인터넷 등을 이용하여 합법적으로 자유롭게 접근할 수 있는 정보를 수집하는 Foot printing기법, 어떠한 시스템이 작동중인지 인터넷을 통하여 어떠한 시스템을 접근할 수 있는지 그들이 제공하는 서비스는 무엇인지를 검색하는 스캐닝기법, 시스템으로부터 노출되어지는 자원의 이름이나 유효한 계정을 추출하는 Enumerating 기법으로 나누어지며 네트워크시스템의 침입시도를 위해서 많이 사용되는 기법인 스캐닝기법에는 핑스윕스(Ping Sweeps)와 포트스캐닝(Port Scanning), 운영체제식별, 파이어워킹(Firewalking)등이 있다[7-10]. 이러한 네트워크 침입시도 공격에 대응하기 위하여 scanlogd, snort, RTSD등 다양한 도구들이 개발되어 왔다[1,5,6]. 개발된 도구들 중 자동화된 탐지를 지원하는 대부분의 도구는 한 호스트에서 일정시간 간격으로 일정한 수의 패킷을 전송할 경우, 이를 취약점 검색공격으로 탐지하는 알고리즘을 적용하고 있다. 이 탐지 알고리즘은 단순하여 작업부하가 적고 빠른 반면에 침입시도 판단의 근거를 시간간격과 전송된 패킷 수에 대한 고정된 임계값에 두고 있어서 느린 스캐닝(slow scanning)공격과 협동공격(coordinated attack)처럼 이들을 회피하는 경우와 이미 취약점으로 알려진 포트와 일반 포트에 대한 접근을 동일하게 취급하여 소수의 취약점 포트 위주로 침입을 시도하는 경우에 공격을 탐지

† 준 회 원 : 단국대학교 대학원 전산통계학과
†† 종신회원 : 단국대학교 전산통계학과 교수
논문접수 : 2001년 8월 3일, 심사완료 : 2001년 8월 28일

못하는 단점을 갖고 있다. 또한 대부분의 도구들은 stand-alone 구조이기 때문에 대규모 네트워크 공격이 발생한 경우에 에이전트 수준이상의 적절한 대응을 하고 있지 못하다. 본 논문에서는 이러한 탐지 알고리즘의 문제점을 개선하고 대규모 네트워크에서의 계층적인 탐지 및 대응을 지원하는 개선된 침입시도 탐지시스템을 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 침입시도 탐지 기술에 대한 관련연구를 분석하고, 3장에서는 분석된 문제점을 개선한 확장된 탐지 알고리즘을 제안한다. 4장에서는 제안된 알고리즘을 바탕으로 침입시도 탐지시스템을 제시하고 가상공격을 통한 모의실험을 통해 시스템을 분석하며 5장에서는 결론을 맺는다.

2. 관련 연구

침입시도 탐지방법에는 시스템 로그파일을 분석하여 탐지하는 방법, 패킷 모니터링 도구를 이용하여 탐지하는 방법, SATAN과 같은 특정 포트 검색공격을 탐지하는 방법, 일반적인 포트 검색공격 패턴을 탐지하는 방법, 이미 알려진 침입시도 공격에 대한 signature를 통해 탐지하는 오용탐지기법, 그리고 네트워크 트래픽 분석 및 패킷 내용을 검사하여 공격을 탐지하는 방법 등이 존재하며 <표 1>에서는 이러한 방법을 적용한 탐지도구들을 보여주고 있다[1, 5, 6].

〈표 1〉 침입시도 탐지도구

구 분	탐지 도구	특징 및 탐지 방법
시스템 로그 분석	syslog, messages 분석	시스템 로그파일 분석
패킷 모니터링	tcpdump, snoop, netfind	패킷 모니터링 로그 분석
특정검색 공격 탐지	courtney, gabriel, Natas	SATAN이 검색하는 특정 포트만 감시
일반 검색 공격 탐지	Netlog,Iplog, portwatch, inflog, scan-detector, detect-scans	모든 포트에 대한 접속 감시 및 기록
오용탐지	오용탐지 기법을 사용하는 일반적인 침입탐지 시스템	알려진 침입시도 공격에 대한 signature를 통해 탐지
Advanced 탐지	tcplogd, scanlogd, Abacus_Sentry, snort, RTSD, libnids	네트워크 트래픽과 패킷 분석, 호스트 접근 차단 등 공격대응 기능

RTSD, snort, Scanlogd, detect-scans, libnids등과 같이 자동화된 탐지를 지원하는 대부분의 도구들은 한 호스트에서 일정시간 간격으로 일정한 수 이상의 패킷이 전송될 경우, 이를 취약점 검색공격으로 탐지하는 알고리즘을 적용하고 있다. 이 알고리즘은 단순하여서 구현이 용이하고 작업부하가 적고 빠르기 때문에 실제 네트워크 시스템에 구현되어 비교적 우수한 성능을 발휘하고 있다[1-3]. 특히 한국정보보호진흥원에서 개발된 RTSD(Real Time Scan Detector)와 Scanlogd, snort는 호스트 기반인 기존의 도구들과 달리 네트워크 기반의 구조를 갖고 있어서 다양한 네트워크 침입시도 공격을 탐지할 수 있으며 RTSD의 경우 침입시도 탐지 에이전트에서 탐지된 침입시도 공격을 메일을 통해 침입시도 매니저에게 통지하도록 하여 종합적이고 체계적으로 대응할 수 있는 구조를 제시하고 있다[1, 6].

그러나 이와 같은 장점에도 불구하고 기존 탐지알고리즘은 다음과 같은 문제점을 갖고 있다. 첫째로 침입시도 판단의 근거를 시간간격과 전송된 패킷 수에 대한 고정된 임계값에 두고 있어서 이를 회피하는 공격시도에 취약하다는 단점이 있다. 특히 장기간에 걸쳐 임계값이내의 패킷을 보내는 느린 스캔공격이나 여러 공격호스트가 공격영역을 분담하여 호스트별로 임계값이내의 패킷을 보내는 협동공격과 같은 기법에 취약하다. 둘째로 취약점 포트와 일반 포트를 모두 동일하게 취급하기 때문에 최근에 가장 많이 발생하고 있는 취약점 포트 위주의 스캔공격이 발생할 경우 취약점 포트에 대한 공격이 이루어지고 있음에도 탐지하지 못하는 문제점을 갖고 있다. 셋째로 대부분의 도구들이 Stand-Alone의 호스트기반 구조이기 때문에 다양한 대규모 네트워크 침입시도 공격을 정확히 탐지하지 못할 뿐만 아니라 에이전트 수준이상의 종합적 탐지와 대응이 불가능하다. 따라서 대규모 네트워크 공격에 적절히 대응하기 위해서 1차적으로는 에이전트 수준의 침입시도 탐지를 지원하고 2차적으로는 에이전트와 상위계층 서버와의 긴밀한 연계를 통한 종합적인 탐지 및 대응을 지원할 수 있는 시스템이 요구된다. RTSD는 상위계층에서 종합적 탐지 분석이 지원되는 계층적인 대응체계를 갖도록 설계되었으나 중앙 서버상에서의 자동화된 실시간 종합탐지를 지원하지 못하며 탐지결과에 따른 위험수준과 대응방안을 에이전트에게 실시간으로 통보하지 못하기 때문에 에이전트 별로 현재의 위험수준에 적절한 대응을 할 수 없고 특히 아직 침입 받지 않은 에이전트의 경우 같은 공격에 취약할 수밖에 없는 문제점을 갖고 있다.

3. 확장된 탐지알고리즘

본 장에서는 2장에서 언급된 기존 연구의 문제점을 개선하는 확장된 알고리즘을 제시한다. 확장된 알고리즘은 탐지알고리즘과 계층적인 대응을 위한 에이전트 및 서버의 알고리즘으로 분리되어서 언급될 것이며 보다 구체적인 이해를 위해 알고리즘은 C언어 형식으로 표기되어 진다.

```
1 : DetectSatus CheckNormalAttack(CPacket* crntPacket)
2 : {
3 :   Host* srcHost ;
4 :   Time crntTime ;
5 :
```

```
6 :  // 현재 패킷의 소스호스트와 목적호스트, 목적포트와 동일한
       패킷이 이미 존재하는가?
7 :  if(IsThereInSourceHostList(crntPacket) ;
8 :  { // 패킷의 소스 호스트리스트에서 현재와 동일한 패킷을 보
       낸 소스 호스트의 정보를 가져온다.
9 :   srcHost = getHostFromSourceHostList(crntPacket) ;
10 :  crntTime = GetcurrentTime() ;
11 :  // 현재 패킷이 전송된 시간이 시간 범위내에 존재하는가?
12 :  if(crntTime - srcHost -> theLastUpdateTime)
        < Normal_Threshold)
13 :  { // 소스호스트의 패킷 전송횟수가 임계치를 넘어서는가?
14 :      if(srcHost-> Connection > SCAN_MAX)
15 :          return DetectNormalAttack ;  // 공격탐지
16 :      else
17 :          srcHost -> Connection++ ; // 패킷전송횟수 증가
18 :      UpdateHost(scrHost, crntTime, crntPacket) ; // 호스
                                                  트정보 수정
19 :  }
20 :  else
21 :  { // 호스트정보 초기화
22 :      InitHost(srcHost, crntPacket) ;
23 :  }
24 : }
25 : else
26 : {
27 : // 패킷의 소스호스트리스트 용량이 그 한계를 초과하였다면
       가장 오래된 호스트 정보를 제거함
28 :  if(IsSourceHostListFull())
            RemoveTheOldestHostFromSourceHostList() ;
29 :  srcHost = MakeNewHost(crntPacket) ; // 새로운 호스트
                                          정보 생성
30 :  AddHostToSourceHostList(srcHost) ; // 생성된 호스트
                                          정보 추가
31 :  }
32 :  return Normal ;
33 : }
```

(그림 1) 기존의 탐지알고리즘

3.1 탐지알고리즘의 개선

2장에서 언급된 기존 탐지알고리즘의 취약점을 이용할 수 있는 공격은 크게 취약점 포트 위주의 공격, 협동공격, 느린 스캔공격, 느린 협동공격으로 분류할 수 있다. 본 절에서는 이러한 공격에 대응할 수 있는 확장된 탐지알고리즘을 제시할 것이다.

3.1.1 취약점 포트 위주의 공격에 대한 개선

(그림 1)의 17행에 있는 srcHost -> Connection++ ; 처럼 기존의 탐지알고리즘은 취약점으로 알려진 포트와 일반 포트에 대한 접근을 동일하게 취급하기 때문에 소수의 취약점 포트 위주 공격을 탐지하지 못한다. 이러한 문제점을 개선하기 위해 포트를 위험도에 따라 여러 그룹으로 분류하고 그룹별로 가중치를 할당하여서 위험도가 큰 포트에 침입을 시도한 공격일수록 탐지될 확률이 높게 한다.

(그림 1)의 14행과 17행은 다음과 같이 수정된다.

```
14 : if(srcHost->TotalWeight > SCAN_MAX)
  // 여기서 SCAN_MAX의 값은 최대빈도수가 아니라 최
     대가중치로 의미가 전환된다.
17 : srcHost -> Connection++ ;
     weight = getPortWeight(crntPacket -> DestPort) ;
     srcHost -> TotalWeight += weight ;
```

3.1.2 협동공격에 대한 개선

(그림 1)의 5줄에 있는 if(IsThereInSourceHostList(crntPacket -> SourceAdd ress))처럼 기존의 탐지 알고리즘은 단지 패킷을 전송한 소스호스트(source host)만을 고려하기 때문에 여러 호스트에서 동시에 협동하여 임계값 이내의 패킷을 전송할 경우, 공격을 탐지하지 못한다. 본 논문에서는 이러한 문제에 대응하기 위해 협동공격의 중요한 단서인 "여러 IP가 하나의 공격대상 호스트를 목표로 공격을 할 때 공격이 발생하는 시간의 길이"를 이용하여 알고리즘을 확장한다[7]. 알고리즘은 (그림 2)와 같다. 제안된 알고리즘은 RecentPacketList를 통해 최근에 전송된 서로 중복되지 않은 패킷정보를 유지하고 있다. 만일 패킷정보리스트 RecentPacketList에서 패킷의 총갯수 nRecentPacketCount가 임계값 RecentPacketMAX를 넘고 MinConnectNum이상의 패킷을 전송한 소스호스트의 수 nRecentSourceHostCount가 임계값 RecentSourceHostMax를 넘을 경우 협동공격으로 간주한다.

3.1.3 느린 스캔에 대한 개선

느린 스캔은 주어진 시간의 범위 내에서 임계값이하의 패킷을 보내어 탐지도구들을 무력화시키는 공격이며 본 논문에서는 이에 대응하기 위해 느린 스캔 탐지기법을 적용한다. 느린 스캔 탐지기법은 시간을 초과한 소스호스트정보를 메모리에서 추출하여 로그에 남기면서 비교적 장기간으로 설정된 일정한 탐지주기마다 (탐지시스템의 오버헤드와 정책에 따라 결정됨) 로그를 검사하여 느린 스캔공격을 탐지하는 기법이다. 느린 스캔 탐지알고리즘은 기존의 방법과 비슷하지만 메모리가 아닌 디스크에서 소스호스트 정보를 검색하고 탐지주기가 일단위 혹은 시간단위 등 장기간으로 설정되는 것이 특징이다.

```
CPacket RecentPacketList[RecentPacketMax] ;
int nRecentPacketCount ;
int nRecentSourceHostCount ;

1 : DetectStatus CheckCoordinatedAttack(CPacket* Packet)
2 : {
3 :  CPacket *tmpPacket ;
4 :
5 :  // 현재 시간을 중심으로 일정한 시간 범위이전에 들어온 패킷
       정보 삭제
6 :  nRecentPacketCount = RemoveTheOldPackets() ;
7 :
8 :   // RecentPacketList에서 MinConnectNum 이상의 패킷을 전
       송한 소스호스트의 수를 가져온다.
```

```
9 :    nRecentSourceHostCount
       = GetSourceHostCountFromPacketList(MinConnectNum) ;
10 :
11 :   // 아직도 협동공격이 진행되고 있는 상태인가?
12 :   if(nRecentPacketCount >= RecentPacketMAX
13 :    && nRecentSourceHostCount >= RecentSourceHostMax)
14 :   return CoordinatedAttackYet ;   // 여전히 협동공격이 진행
                                          된 상태임
15 :
16 : // 현재 패킷의 소스호스트와 목적호스트 및 목적포트를 가진
       패킷정보가 이미 리스트에 존재하는가?
17 : if(IsThereTheSampPacketInRecentPacketList(Packet) ;
18 : {
19 :   // 리스트에서 패킷정보를 가져온다.
20 :    tmpPacket
       = getTheSamePacketFromRecentPacketList(Packet) ;
21 :    UpdatePacket(tmpPacket, Packet) ;
22 : }
23 : else
24 : {
25 :   // 만일 리스트 용량의 한계가 초과한다면 가장 오래된 패킷
       을 리스트에서 삭제
26 :   if(IsRecentPacketListFull())
       RemoveTheOldestPacketFromRecentPacketList() ;
27 :
28 :  // 새로운 패킷을 추가
29 :  nRecentPacketCount
      = AddPacketToRecentPacketList(Packet) ;
30 :
31 :   // RecentPacketList에서 MInConnectNum 이상의 패킷을 전
       송한 소스호스트의 수를 가져온다.
32 :   nRecentSourceHostCount
       = GetSourceHostCountFromPacketList(MinConnectNum) ;
33 :
34 :   //협동공격이 진행되고 있는 상태인가?
35 :   if(nRecentPacketCount >= RecentPacketMAX
36 :   && nRecentSourceHostCount >= RecentSourceHostMax)
37 :   return DetectCoordinateAttack ; //새로운 협동공격탐지
38 :  }
39 :  return Normal ;
40 : }
```

(그림 2) 협동공격 탐지 알고리즘

3.1.4 느린 협동공격에 대한 개선

느린 협동공격은 협동공격의 단서인 "여러 IP가 하나의 공격대상 호스트를 목표로 공격할 때 공격이 발생하는 시간의 길이"를 길게 하여 탐지도구를 무력화하는 기법이다. 느린 협동공격을 탐지하기 위해서 (3)과 같이 남겨진 로그를 통하여 길게 설정된 시간주기마다 로그를 검사한다. 제안된 알고리즘은 (2)의 경우와 비슷하지만 탐지주기가 일단위 혹은 시간단위 등 장기간으로 설정되어지며 이 주기동안 남겨진 로그를 검사한다.

```
CPacket* Packet ;
DetectStatus eStatus ;
ReportAttack attackReport ;
Time tmCrnt, tmSlowScanChkTime ;

    tmSlowScanChkTime = GetCrntTime() ;
1 : while
2 : {
3 :   // 패킷을 읽어 온다.
4 :   Packet = PacketCapture() ;
5 :
6 :   // 일반적인 공격체크
7 :   eStatus = CheckNormalAttack(Packet) ;
8 :           if(eStatus != Normal) AlertAttack(eStatus, Packet)
9 :   // 협동공격체크
10 :  eStatus = CheckCoordinatedAttack(Packet) ;
11 :  if(eStatus != Normal)AlertAttack(eStatus, Packet) ;
12 :
13 :  tmCrnt = GetCrntTime(0 ;
14 :
15 :  // 현재시간을 중심으로 오래된 패킷을 로그에 저장한다.
16 :  LogOldPacket(tmCrnt) ;
17 :
18 :  // 느린 공격 체크주기가 된다면
19 :  if(tmCrnt - tmSlowScanChkTime >= _SlowScanPeriod)
20 :  {
21 :     //느린 스캔공격 체크
22 :     eStatus = CheckSlowScan(&attackReport) ;
23 :     if(eStatus != Norml) AlertSlowAttack(eStatus,
                    attackReport) ;
24 :
25 :    //느린 협동공격 체크
26 :    eSatus = CheckSlowCoordinatedAttack(&attackReport) ;
27 :    if(eStatus != Normal) AlertSlowattack(eStatus,
                   attackReport) ;
28 :
29 :      tmSlowScanChkTime = GetCrntTime() ;
30 :   }
31 : }
```

(그림 3) 개선된 탐지알고리즘

만일 탐지주기 내에 전송된 패킷들 중 목적 호스트의 주소와 포트가 서로 중복되지 않은 패킷의 수가 임계값 SLOW_COORDINATED_ATTACK_MAX 이상이고 Slow MinConnectNum이상의 패킷을 전송한 소스호스트의 수가 임계값 SLOW_COORDINATED_ATTACK_MAXHOST를 넘을 경우 느린 협동공격이 탐지된다. (그림 3)은 앞서 언급되었던 알고리즘을 모두 적용한 개선된 전체 알고리즘을 보여 주고 있다.

3.2 계층적인 대응을 위해 확장된 알고리즘

본 절에서는 에이전트와 중앙서버사이의 유기적인 연동을 통해 다양한 네트워크공격에 대한 계층적인 대응을 할 수 있는 알고리즘을 제안한다. (그림 4)에 계층적인 알고리즘에 관하여 자세히 언급되어 있다. 공격이 탐지되면 에이전트는 공격에 대한 정보를 생성하여 서버에게 보고를 한다. 서버가 에이전트로부터 공격정보를 보고 받으면 로그DB에 공격정보를 저장하고 현재시간을 중심으로 주어진 시간범위내에 보고되었던 공격정보를 통해 현재의 위험수준을(alert level) 결정한다. 위험수준은 현재 연결된 에이전트 대비 공격을 당한 에이전트의 비율과 주어진 시간동안 발생한 공격의 빈도수에 근거하여 결정되어 진다. 만일 결정된 위험수준이 네트워크공격상태를 나타내고 있다면 서

버는 위험수준을 에이전트에게 통보한다. 서버는 위험수준의 강도에 따라 혹은 관리자의 권고요청이 발생할 경우, 위험수준과 더불어 블랙리스트를 에이전트에게 전송하게 되는데 블랙리스트는 일정기간동안 보고되어 왔던 공격정보에서 특이하게 많이 사용되었거나 현재 탐지된 공격에서 사용되고 있는 혹은 관리자가 경계하도록 요청한 IP와 포트들로 구성된다. 서버로부터 공격메시지를 전달받으면 에이전트는 전달된 위험수준과 블랙리스트를 현재의 침입시도 공격 탐지정책에 반영한다. 에이전트는 서버가 전송한 위험수준을 고려하여 현재 침입시도 탐지에 적용하고 있는 주요 임계값들을 재설정하고((그림 1)의 SCAN_MAX값이나 (그림 2)의 RecentPacketMAX, MinConnectNum, Recent SourceHostMax 같은 값을 감소시킴) 더욱 엄격하게 침입시도 공격을 검사하게 된다. 즉 에이전트는 위험수준이 높을

```
CAttackReport* pAttackReport;
pAttackReport = MakeAttackReport();
SendAttackReportToServer(pAttackReport);
```

```
CAttackReport* pAttackReport;
AttackLevel eAttackLevel;

while(1)
{
  ...
  // 공격에 대한 정보를 에이전트로 부터 받음
  if(ReceiveAttackReportFromAgent(pAttackReport)
  {
   // 공격에 대한 정보를 LogDB에 추가
    AddAttackReportToLogDB(pAttackReport);
   // 현재의 위험수준을 구함
    eAttackLevel = CalcAttackLevel();
   // 공격이 심각한 수준이라면...
    if(IsCriticalAttack(eAttackLevel))
    {
     // 현재 탐지된 공격의 소스호스트와 IP를 추출
      blackIPs = GetBlackIPsFromLogDB();
      blackPorts = GetBlackPortsFromLogDB();
     // 에이전트에게 전송
      SendAlertToAgent(eAttackLevel,blackIPs,blackPorts);
    }
  }
  else
  {
   // 관리자가 전송요청하거나 최근에 특별히 많이 사용된
   // IP나 포트가 존재하는가?
    if(IsThereRequestedBlackIPs())
        blackIPs = GetRequestedBlackIPList();
    if(IsThereRequestedBlackPorts())
        blackPorts = GetRequestedBlackPortList();
   // 에이전트에게 전송
    SendRequestToAgent(blackIPs,blackPorts);
  }
}
```

```
AttackLevel eAttackLevel;

while(1)
{
  ...
  // 위험수준을 서버로 부터 전송받음
  if(!ReceiveAlertFromServer(&eAttackLevel,
                    blackIPs, blackPorts)) continue;

  // blackIPs를 에이전트의 BlackIPList에 추가한다.
    // 향후, 이들 IP로 들어 오는 모든 패킷은 공격으로 간주함
  AddBlackIPsToBlackIPList(blackIPs);

  // blackPorts를 에이전트의 포트그룹에 추가한다.
    // 각 포트의 위험정보에 따라 적절한 그룹에 추가한다.
  AddBlackPortsToPortGroup(blackPorts);
  // 전송된 위험도에 따라 주요 임계값을 재설정함
  TuningAllThreshold(eAttackLevel);
  ...
}
```

(그림 4) 에이전트와 서버사이의 계층적 탐지알고리즘

수록 False Positive의 오류를 감수하면서 침입시도 공격에 대한 개연성있는 모든 네트워크연결들을 더욱 엄격하게 검사하게 되는 것이다. 서버가 전송한 블랙IP리스트는 에이전트의 블랙IP리스트에 추가되어 이후에 리스트에 포함되는 IP로부터 전송되어진 패킷이 발견되면 공격으로 간주된다.

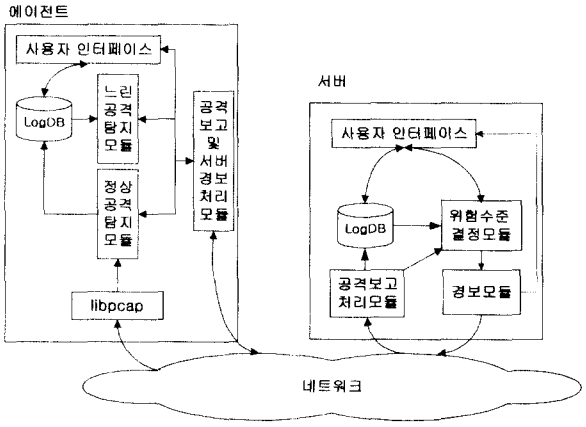
또한 블랙포트리스트는 리스트에 포함된 각 포트의 위험도에 따라 포트그룹에 추가되어 진다. 제안된 알고리즘을 통해 대규모 네트워크 공격이 발생하게 되면 각 에이전트는 서버로부터 적절한 경보메시지를 받게 되고 이에 대한 실질적인 대응을 할 수 있게 된다.

4. 제안된 침입시도 탐지시스템 구현

본 장에서는 침입시도 탐지시스템을 구현하고 모의 실험을 통해 3장에서 제안된 알고리즘의 타당성을 검증한다.

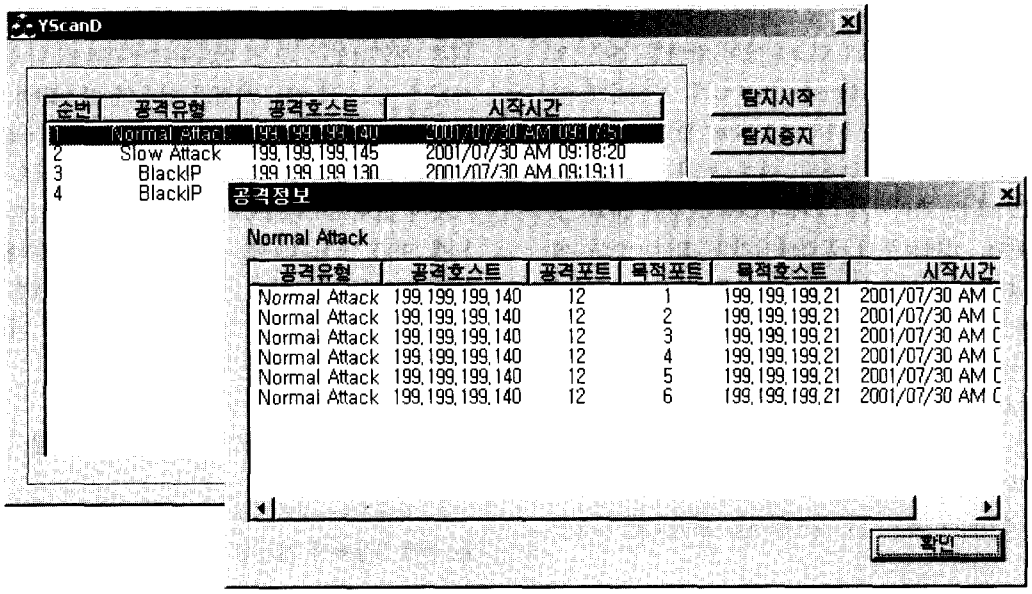
4.1 시스템구조

제안된 탐지시스템의 구조는 (그림 5)와 같다.



(그림 5) 침입시도 탐지시스템 구조도

제안된 시스템은 Windows2000 Server/Professional 환경에서 Visual C++ 6.0을 통해 구현되었으며 패킷캡쳐를 위해 시스템 독립적이고 이식성이 뛰어난 libpcap모듈을 적용하였다. 에이전트의 정상 공격 탐지모듈과 느린 공격 탐지모듈, 공격보고 및 서버경보 처리모듈 그리고 서버의 공격보고 처리모듈, 위험수준 결정모듈, 경보모듈은 각각 쓰레드로 구성되어 동시에 수행되도록 하였고 사용자 인터페이스를 통해 환경에 따라 적합한 정책설정이 가능하도록 하였다. 시스템에서는 위험수준이 4개의 범주로 나뉘며 포트그룹은 5개의 범주로 나뉜다. 위험수준은 앞서 언급된 것처럼 현재 서버에 연결된 에이전트의 수 대비 공격이 발생한 에이전트의 수의 비율과 주어진 시간범위내에 발생된 공격의 빈도수의 범위에 따라 결정되며 <표 2>는 위험수준별 비율과 빈도수의 범위를 나타내고 있다. <표 3>과 <표 4>



(그림 6) 침입시도 탐지결과

는 각각 시스템에 적용한 임계값과 포트그룹 및 그룹별 가중치를 보여주고 있다. (그림 6)은 공격이 탐지되었을 때, 에이전트의 화면이다.

〈표 2〉 위험수준 결정범위

비 율	빈 도 수	위험수준
0.0	0	Alert 1
0.0~0.33	1~4	Alert 2
0.33~0.66	4~8	Alert 3
0.66 이상	8 이상	Alert 4

〈표 3〉 시스템의 임계값

임 계 변 수	사 용 모 듈	값
SCAN_MAX	정상공격탐지모듈	12
RecentPacketMAX	정상공격탐지모듈	8
MinConnectNum	정상공격탐지모듈	2
RecentSourceHostMax	정상공격탐지모듈	2
SLOW_SCAN_MAX	느린공격탐지모듈	12
SlowMinConnectNum	느린공격탐지모듈	2
SLOW_COORDINATED_ATTACK_MAX	느린공격탐지모듈	8
Normal_time_Threshold	정상공격탐지모듈	3초
Slow_time_Threshold	느린공격탐지모듈	1시간

〈표 4〉 포트그룹과 가중치

포 트 그 룹	가 중 치	포 트
_SCAND_Goup 1	SCAN_MAX	10008,12345,1243,31337,27374
_SCAND_Goup 2	4	21,22,23,25,53,111
_SCAND_Goup 3	3	1,1114,2766
_SCAND_Goup 4	2	1~1024내의 포트
_SCAND_Goup 5	1	그외 포트

4.2 실험 분석

구현된 시스템에서는 크게 3.1에서 언급된 개선된 탐지알고리즘과 3.2에서 언급된 계층적 탐지알고리즘으로 나뉘어서 실험분석이 수행된다.

4.2.1 개선된 일반 탐지알고리즘

본 논문에서는 개선된 탐지알고리즘에 대한 실험분석을 위해 기존의 탐지알고리즘이 취약했던 공격을 위주로 <표 5>에서 제시된 공격유형과 시나리오에 따라 스캔공격을 시도하였다[8, 9]. <표 5>에서 제시된 공격은 [8]과 [9]에서 언급된 바처럼 해커들이 탐지시스템을 우회하기 위해 주로 사용하는 침입시도 공격기법들이며 특히 취약점 위주의 공격은 최근에 가장 많이 사용되고 있는 침입시도 공격기법이다[1-3]. 본 논문에서는 공격의 실행을 위해 침입시도 공격을 적용할 수 있는 공격도구를 개발하였으며 이를 통해 제안시스템 및 RTSD, SCANLOGD에 침입시도 공격을 수행함으로써 그 성능을 비교분석 하였다.

<표 6>은 <표 5>에서 제시한 공격에 대한 각 시스템별 탐지결과를 보여주고 있다. <표 6>의 탐지결과를 보면 기존의 알고리즘을 적용한 RTSD와 SCANLOG는 일반공격 이외의 공격에 취약한 반면 제안된 탐지알고리즘을 적용한 시스템은 일반공격이외에 취약점 포트 위주의 공격, 협동공격, 느린 스캔, 느린 협동공격을 정확히 탐지하는 것을 알 수 있다. 특히 취약점 포트 위주의 공격의 경우, 미리 취약포트로 분류해 놓은 포트의 가중치가 일반 포트보다 크기 때문에 RTSD나 SCANLOGD와 달리 소수의 공격시도에도 공격이 탐지되어 질 수 있으며 임계값을 우회하기 위한 느린 공격이나 느린 협동공격의 경우, 주기적인 로그DB 분석

을 통해 탐지되어 질 수 있음을 알 수 있다.

〈표 5〉 공격시나리오

공 격 유 형	시 나 리 오
일 반 공 격	포트를 1부터 순차적으로 증가하면서 공격
취약점 포트 위주의 공격	포트번호 21,23,25,111와 같은 취약점 포트 공격
협 동 공 격	여러 소스호스트에서 협동하여 공격
느 린 스 캔	포트번호 21,23,25,111를 위주로 1시간에 걸쳐 공격
느린 협동공격	여러 소스호스트에서 협동하여 1시간에 걸쳐 공격

〈표 6〉 탐지결과 (공격시도 각각 50회)

공 격 유 형	탐지율 (탐지횟수/공격시도횟수)		
	프로토타입	RTSD	SCANLOGD
일 반 공 격	100%	100%	100%
취약점 포트 위주의 공격	100%	0%	0%
협 동 공 격	100%	0%	0%
느 린 스 캔	100%	0%	0%
느린 협동공격	100%	0%	0%

4.2.2 계층적 탐지알고리즘

계층적 탐지알고리즘의 실험분석은 1대의 서버와 4대의 에이전트에서 실행되었다. 침입시도 공격은 단계별로 공격대상 에이전트를 하나씩 증가하면서 동시에 수행하였고 각 공격상황에 따른 위험수준 및 각각의 임계값들의 변화, 공격이 발생하지 않은 에이전트에 대한 재공격을 할 경우 공격이 탐지되는 속도를 측정하였다.

〈표 7〉 에이전트별 위험수준

공격유형	에이전트별 위험수준			
	에이전트 1	에이전트 2	에이전트 3	에이전트 4
1대공격	공격발생	Alert 2	Alert 2	Alert 2
2대공격	공격발생	공격발생	Alert 3	Alert 3
3대공격	공격발생	공격발생	공격발생	Alert 4

<표 7>은 공격에 따른 에이전트별 위험수준을 나타내고 있고 <표 8>은 공격에 따른 임계값의 변화와 공격이 발생하지 않은 에이전트에 대한 재공격결과를 보여주고 있다.

〈표 8〉 공격유형에 따른 임계값의 변화

공격유형	공격발생에 따른 임계값		다른 에이전트 재공격 탐지속도	
	SCAN_MAX	RecentPacketMAX	동일한 공격 호스트	다른 공격 호스트
1대공격	10	7	5번시도	5번시도
2대공격	9	6	1번시도	5번시도
3대공격	8	5	1번시도	4번시도

<표 8>을 보면 위험수준이 증가함에 따라 제안된 시스템의 임계값이 보다 엄격하게 변화되어짐을 알 수 있으며 이로 인해 공격당하지 않은 다른 에이전트에 재공격을 할 경우, 위험수준이 Alert1일 때보다(6번째 패킷전송시도에서 탐지됨) 더 빠르게 공격을 탐지하는 것을 볼 수 있다. 특히 위험수준이 Alert3이상일 때는 공격호스트의 주소가 블랙리스트에 추가되기 때문에 동일한 공격호스트를 적용하여 공격을 시도하면 처음시도에서 공격이 탐지되는 것을 볼 수 있다.

4.3 시스템한계

본 논문에서 제시한 알고리즘은 임계값 기반의 기법이기 때문에 부적절한 임계값의 설정은 침입 오탐지의 가능성을 크게 한다. 예를 들면 엄격한 임계값 설정은 거짓탐지의 가능성을 높게 하고 느슨한 임계값 설정은 탐지미스(miss)의 가능성을 높게 한다. 따라서 침입 오탐지를 최소화하기 위해서는 각 네트워크 환경에 따른 적절한 임계값의 설정이 강력히 요구되어진다. 그러나 적절한 임계값의 설정은 네트워크 트래픽의 특성을 세부적으로 정확히 파악해야 하기 때문에 수많은 시간과 시행착오가 선행되어야 한다. 또한 네트워크 트래픽의 특성은 시간에 따라 변하기 때문에 이에 따른 임계값들의 학습도 고려되어야 한다. 본 논문에서는 이처럼 적절한 임계값 설정과 학습에 관한 연구를 향후 연구과제로 남긴다.

5. 결 론

본 논문에서는 기존의 침입시도 탐지알고리즘이 갖고 있는 문제점을 분석하여 개선된 계층적인 침입탐지와 대응을 지원하는 침입탐지 시스템을 제안하였다. 개선된 탐지알고리즘은 포트를 취약성에 따라 그룹별로 나누고 가중치를 할당하여 소수의 취약점 포트 위주의 공격에 대응하였으며 주어진 시간이내에 서로 중복되지 않은 패킷이 일정한 수 이상이고 서로 중복되지 않은 일정한 수 이상의 패킷을 전송한 호스트가 일정한 수 이상일 때를 탐지하여 협동공격에 대응하였다. 느린 스캔공격이나 느린 협동공격에 대응하기 위해 서는 패킷을 로그DB에 남기면서 장기간으로 설정된 주기별로 로그DB를 검색하도록 하였다.

한편 계층적인 침입탐지와 대응을 위해서 탐지시스템을 에이전트와 서버로 구성하여 에이전트가 공격을 받으면 서버에게 공격상황을 보고하도록 하고 서버에서는 에이전트의 공격보고를 통하여 현재의 위험수준과 그에 따른 블랙리스트를 구하여 에이전트에게 통보하도록 하였다. 통보된 위험수준에 따라 에이전트는 탐지를 위한 임계값을 재설정하고 블랙리스트를 반영하여 적절한 대응을 하게 된다. 본 논문에서는 개선된 알고리즘에 근거하여 시스템을 구현하였고 정해진 공격시나리오를 통해 RTSD, SCANLOGD와 함께 시스템에 대한 모의실험을 하였다.

가상 공격을 통한 모의실험의 결과로부터 제안된 시스템이 기존 알고리즘이 취약했던 소수의 취약점 포트 위주의 공격과 협동공격, 느린 스캔, 느린 협동공격을 정확히 탐지

할 수 있음을 보였다. 또한, 여러 에이전트와 서버가 공존하는 계층적인 환경에서 공격이 발생하여 위험수준이 높아질수록 더욱 엄격하게 공격을 탐지하여 같은 공격이나 유사한 공격을 더욱 빠르게 탐지함을 보였다.

향후 연구과제로서 실제 네트워크 환경에서의 적용을 위해 다양한 네트워크 환경에서의 시스템 구축을 통한 실질적인 성능분석이 요구되며 침입 오탐지의 최소화를 위한 네트워크 특성에 따른 적절한 임계값 설정 및 학습에 관한 연구와 에이전트와 서버사이의 표준화를 지원하는 신뢰성 있는 통신프로토콜 설계가 요구된다.

참 고 문 헌

[1] 이현우외 4명, 대규모 네트워크 취약점 검색공격 패턴분석 및 탐지도구, http : //www.certcc.or.kr, 1999.

[2] 한국정보보호진흥원, "Hacking 통계자료", http : //www.certcc.or.kr, 2001.

[3] 한국정보보호진흥원, "RTSD 2001년 통계자료", http : //www.certcc.or.kr, 2001.

[4] 한국정보보호진흥원, "Firewalking 분석 보고서", http : //www.certcc.or.kr, 1998.

[5] Solar Designer, "Designing and Attacking Port Scan Detection Tools," Phrack Magazine Vol.8 Issue 53, 1998.

[6] Martin Roesch, "Snort - Lightweight Intrusion Detection for Networks," USENIX LISA '99 conference, 1999.

[7] Ofir Arkin, "Network Scanning Techniques," http : //www.sys-security.com, 1999.

[8] Fydor, "The Art of Port Scanning," Phrack Magazine Vol.7 Issue 51, 1997.

[9] Stephen Northcutt, "Intelligence Gathering Techniques," http : //www.microsoft.com/technet/security/intel.asp.

[10] Chrostppher Klaus, "Stealth Scanning - Bypassing Firewalls/SATAN Detectors," http : //www.netsys.com/firewalls/firewalls-9512/0085.html.

유 일 선

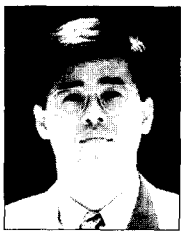
e-mail : qjemfahr@hanmail.net

1995년 단국대학교 전산통계학과 졸업(이학사)

1997년 단국대학교 일반대학원 전산통계학과(이학석사)

1997년~현재 단국대학교 일반대학원 전산통계학과(박사과정수료)

관심분야 : 침입탐지, 네트워크보안, 사용자 인증 및 접근통제



조 경 산

e-mail : kscho@dankook.ac.kr

1979년 서울대학교 전자공학과 졸업(학사)

1981년 한국과학원 전기 및 전자공학과 졸업(공학석사)

1988년 텍사스 대학교(오스틴소재) 전기전산공학과 졸업(Ph.D.)

1988년~1990년 삼성전자 컴퓨터부문 책임연구원

1990년~현재 단국대학교 전산통계학과 교수

관심분야 : 컴퓨터 구조, 성능분석, 시뮬레이션