실행코드의 단편화를 통한 SVAM의 성능 향상에 관한 연구

김희승*, 한영주*, 양진석*, 정태명*
*성균관대학교 정보통신공학부
e-mail : hskim, yjhan, jsyang@imtl.skku.ac.kr, and
tmchung@ece.skku.ac.kr

A Study on Performance Improvement of SVAM using fragmentation of *v*.CC

Hee-Seung Kim*, Young-Ju Han*, Jin-Seok Yang*, and Tai-Myoung Chung
*School of Information and Communication Engineering,
Sungkyunkwan university

요 약

액티브 네트워크는 액티브 패킷을 이용하여 실행코드르 운반함으로서 새로운 서비스와 프로토콜을 빠르고 유연하게 배포할 수 있는 장점이 있다. 하지만 액티브 패킷은 실행코드를 운반해야 하기 때문에 네트워크 성능 상의 문제가 발생하고 있다. 따라서 본 논문에서는 실행코드를 클래스 별로 분리하고 필요한 클래스만을 전송하는 새로운 메커니즘을 이용하여 액티브 패킷이 운반해야 하는 실행코드의 크기를 줄여 액티브 네트워크의 성능을 향상시킬 수 있는 방법을 제시하고, 이를 이론적으로 증명한다.

1. 서론

인터넷은 그간 급격히 발달하면서 많은 사용자가 인터넷을 중심으로 정보를 공유하게 되었고, 이에 따라 많은 요구사항이 제기되었다. 이러한 많은 요구사항들은 다양한 응용프로그램들과 새로운 프로토콜로 만족될 수 있지만, 현재 네트워크의 기반 구조인 TCP/IP 네트워크에서는 고정되어 있는 스택 구조와 표준 제안의 시간문제 등으로 인해 응용프로그램들과 프로토콜들이 빠르게 전파될 수 없는 단점이 있다.이에 비해 액티브 네트워크는 패킷이 직접 응용프로그램을 운반하고 각 노드들은 이 응용프로그램을 실행할 수 있는 실행환경을 제공함으로서, 새로운 응용프로그램을 빠르게 전파할 수 있는 장점을 가진다[2]. 하지만 전달해야 하는 프로그램 크기가 큰 경우 네트워크 상에서 트래픽 문제, 패킷 분실 시 재전송 문제 등에 인해 효율성이 저하될 수 있다[1].

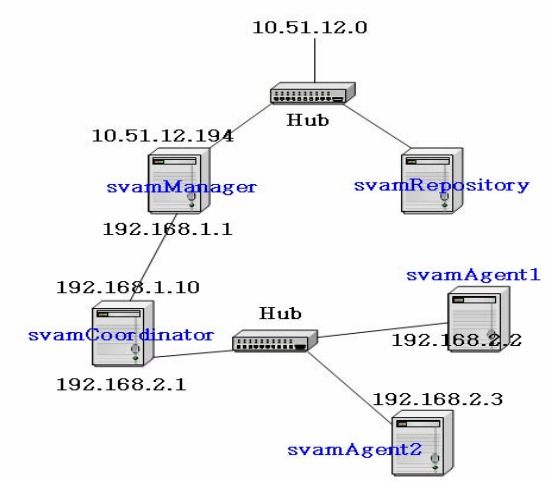
본 연구실에서는 액티브 네트워크 기반의 취약성 검사 시스템인 SVAM(the Scalable Vulnerability Analysis Model)을 개발했었고[3], 본 논문에서는 SVAM 시스템 기반에서 취약성 검사를 위해 패킷이 운반하는 실행코드를 여러 개의 클래스 형태로 분할하여 운반하고 에이전트 시스템에서는 필요한 클래스만을 다운로드하여, 실제 네트워크 상에서 흐르는 실행코드의 크기를 줄이는 방법에 대해 제안하고, 이에 대해 기존의 메커니즘과의 성능 분석을 통하여 제안된 메커니즘의 성능을 평가한다.

2. 테스트 환경

2.1. SVAM의 구조

본 논문은 보건복지부 보건의료기술진흥사업회 지원에 의하여 이루어진 것임(02-PJ3-PG6-EV08-0001)

SVAM 는 [그림 1]과 같이 svamManager, svamCoordinator, svamAgent, svamRepository 로 구성되어 있다. 각각의 구성 요소에 대한 설명은 다음과 같다.



[그림 1] SVAM 의 구성 요소

svamManager 는 자신이 속해 있는 관리 도메인 내에 존재하는 모든 관리 대상 노드를 대상으로 취약성 검사에 대한 책임을 가진다. 즉, 관리대상 노드, svamCoordinator 의 등록, 삭제, 변경 및 취약성 리스트의 등록, 삭제, 변경을 수행한다. 그리고 관리 대상 노드별 취약성 점검 정책 설정 및 관리대상 노드가 속한 svamCoordinator 로의 취약성 점검 정책 분배를 수행한다. 또한 svamCoordinator 로부터 취약성 검사 결과를 수신하여 관리자에게 보고한다. 더불어 관리자가 손쉽게 svamManager 를 제어할 수 있도록 사용자 인터페이스를 제공한다.

svamCoordinator 는 관리 도메인 내에 존재하는 액티브 노드로 구성된 에지 라우터 상에서 동작한다. svamCoordinator 는 svamManager 로부터 수신한 취약성 점검 정책에 따라 자신에게 연결되어 있는 서브넷 상의 관리대상 노드들을 대상으로 실질적인 취약성 검사를 수행한다. svamCoordinator 의 정보는 svamManager 에 등록되어 관리된다.

svamAgent 는 관리 도메인 내의 관리 대상 노드 상에서 동작한다. svamCoordinator 의 스케줄링에 의해 취약성 검사 주기가 되어 취약성 검사 요청을 수신하면, 해당 취약성 검사 코드인 *v*.CC (VulCheckCode)를 찾아 실행한다. 만약, 로컬 노드에 해당 *v*.CC 가 존재하지 않을 경우 svamRepository 로부터 해당 *v*.CC 를 수신하여 취약성 검사를 수행한 후 svamCoordinator 에게 전달한다. *v*.CC 는 정책에 따라서 일정시간동안 svamAgent 에 저장되고, 시간이 만료된 후에 삭제된다.

관리 도메인 내에는 취약성 검사 코드 저장소인 svamRepository 가 여러 개 존재할 수도 있다. 그 중 하나가 "primary" 권한을 가지게 되며, 나머지는 "secondary" 권한을 가진다. 취약성 검사 코드 저장소는 취약성 검사를 수행하는 *v*.CC 를 보관한다. 새로운 취약성에 대하여 취약성 검사 코드인 *v*.CC 가 생성되면 svamManager 를 통해 맨 처음 Primary svamRepository 에 등록되며, Primary svamRepository 가 관리 도메인 내에 있는 Secondary svamRepository 들로 새로 추가된 *v*.CC 를 분배한다. 본 테스트베드에서는 primary svamRepository 만 존재한다[3].

2.2. SVAM 의 실행코드 구성

SVAM 에서 사용되는 실행코드는 여러 개의 클래스로 이루어져 있고, 클래스는 다른 실행코드에서도 사용되는지의 여부에 따라서, 오직 한 실행코드에서만 사용되는 클래스를 “단독 클래스”, 2 개의 실행코드에서 사용되는 클래스를 “2 공용 클래스”, 3 개의 실행코드에서 사용되는 클래스를 “3 공용 클래스” 등과 같이 정의하였다.

SVAM 에서의 취약성 검사는 크게 두 가지로 구분되어 진다. 첫 번째는 svamCoordinator 에서 자신의 서브넷에 존재하는 노드를 대상으로 취약성 검사하는 원격 취약성 검사이다. 원격 취약성 검사에는 TCP 포트 검사, UDP 포트 검사, OS 검사 등이 있다[3]. 이러한 취약성 검사 프로그램은 다음 [표 1]과 같이 구성되어 있다.

[표 1] 원격 취약성 검사 실행코드의 구성

검사종류	TCP Scan		UDP Scan	OS Scan
	Open Scan	Half Scan		
단독 클래스	main openscan	main tcp_gen	main udp_gen recv_icmp	main find_open finger_print
2 공용 클래스		recv_tcp		recv_tcp
3 공용 클래스		name2ip, ip_gen, in_cksum, trans_check sweep, exchange		

네 가지의 원격 취약성 검사 실행코드는 단독 클래스로 각각의 main 클래스와 다른 실행코드에서는 사용되지 않는 몇 개의 클래스를 사용하고, TCP 기반의 raw 소켓을 사용하는 Half Scan 과 OS Scan 는 2 공용 클래스로 TCP 패킷을 받기 위한 recv_tcp 클래스를 사용한다. 또한 IP 헤더를 생성하는 ip_gen 클래스, 체크섬을 계산하는 in_cksum, trans_check 클래스등이 3 공용 클래스로 사용된다.

두 번째는 svamCoordinator 가 svamAgent 에게 취약성 검사 요청을 하여 svamAgent 가 직접 자신의 취약성을 검사하고 검사 결과를 svamCoordinator 에 통보하는 로컬 취약성 검사이다. 로컬 취약성 검사에는 Apache 버전 검사, httpd.conf 파일 문법 검사, Sendmail 버전 검사, Sendmail 원격 실행 권한 검사 등이 있다[3]. 이러한 취약성 검사 실행코드는 다음 [표 2]와 같이 구성되어 있다.

[표 2] 로컬 취약성 검사 실행코드의 구성

검사 종류	Apache Ver	httpd.conf File				Sendmail Ver
		User	Group	Dir	Syntax	
단독 클래스	main	main	main	main	main	main
2 공용 클래스			execAPI		execAPI	
3 공용 클래스		ChkConfSet, ChkPattern, FileManager				
4 공용 클래스	execPSAPI					

여섯 가지의 로컬 취약성 검사 실행코드는 단독클래스로 각각의 main 클래스를 사용하고, 특정 명령을 실행하는 execAPI 클래스가 2 공용 클래스로, 설정 파일의 형식을 검사하는 ChkConfSet, ChkPattern, FileManager 클래스를 3 공용 클래스로, ps 명령을 실행하는 execPSAPI 클래스를 4 공용 클래스로 사용한다.

[표 1]과 [표 2]와 같이 하나의 검사 실행코드는 여러 개의 클래스로 구분되어 질 수 있고, 비슷한 성격의 검사 실행코드에는 서로 공용으로 사용될 수 있는 클래스가 존재한다.

3. 성능 평가

새로 제안되는 메커니즘은 2 장에서 설명된 것과 같이 실행코드가 여러 개의 클래스로 구분되어 질 수 있고, 비슷한 성격의 실행코드에는 공통되는 클래스를 포함할 수 있는 특징을 이용한 것이다. 곧, 새로운 메커니즘은 SVAM 에서 svamRepository 와 svamCoordinator 간 또는 svamRepository 와 svamAgent 간에 전송되는 실행코드들의 크기를 줄여 전체 네트워크의 성능을 향상시킬 수 있다.

두 메커니즘의 성능평가를 위해 필요한 인자는 다음 표와 같이 정의될 수 있다.

[표 3] 성능 인자

인자	설명
L_{vcc}	*v*.CC 코드를 구성하는 클래스 목록
$\|vCC\|$	*v*.CC 코드의 크기
$\|L_{vcc}\|$	*v*.CC 코드를 구성하는 클래스 목록의 크기
$Lib_{vcc,n}$	*v*.CC 코드 중 n 클래스
$\mathrm{Re}q_m$	svamManager의 취약성 검사 요청
$\mathrm{Re}q_{ca,n}$	svamCoordinator 또는 svamAgent가 svamRepository로 보내는 n 요청 메시지
T_{ex}	기존의 메커니즘에서 전송되는 패킷의 총 크기
T_{new}	새로운 메커니즘에서 전송되는 패킷의 총 크기

새로운 메커니즘에서는 svamManager 가 검사 요청을 할 때, 실행코드에 속하는 클래스들의 목록을 svamCoordinator 나 svamAgent 에게 알려줘야 한다. L_{vcc} 는 이 목록을 의미하고 실제로 클래스의 이름을 나열한 텍스트 형태이기 때문에 $|L_{vcc}|$ 가 나타내는 L_{vcc} 의 크기는 단지 수십 바이트 정도이다. $Lib_{vcc,n}$ 는 *v*.CC 코드 중 n 이라는 클래스를 나타낸다. 따라서 *v*.CC 의 모든 클래스는 *v*.CC 와 같고 다음 공식을 만족한다.

$$vCC = \forall_n Lib_{vcc,n} \quad ①$$

$$|vCC| = \left|\forall_n Lib_{vcc,n}\right| \quad ②$$

기존의 메커니즘에서는 svamManager가 svamCoordinator나 svamAgent에게 취약성 검사 요청을 위해 $\mathrm{Re}q_m$ 를 전송하고 svamRepository에게서 *v*.CC를 받기 위하여 $\mathrm{Re}q_{ca,vcc}$ 를 svamRepository로 전송하여 *v*.CC를 받는다. 새로운 메커니즘에서는 svamManager가 svamCoordinator나 svamAgent에게 취약성 검사 요청을 위해 $\mathrm{Re}q_m$ 와 *v*.CC를 구성하는 클래스 목록인 L_{vcc} 를 전송하고, L_{vcc} 에서 필요한 클래스를 선택하여 svamRepository에게 $\mathrm{Re}\,q_{ca,\exists_n Lib_{vcc,n}}$ 을 전송하고 $\exists_n Lib_{vcc,n}$ 을 전해 받는다. 따라서 이 두 메커니즘을 통해 전송되는 패킷의 총 크기인 T_{ex} 와 T_{new} 는 다음과 같다.

$$T_{ex} = |\mathrm{Re}\, q_m| + |\mathrm{Re}\, q_{ca,vcc}| + |vCC| \quad ③$$

$$T_{new} = |\mathrm{Re} q_m| + |L_{ca,vcc}| + |\mathrm{Re} q_{ca,\exists_n Lib_{vcc}}| + |\exists_n Lib_{vcc,n}| \quad ④$$

첫 실행코드를 받게 되는 경우에는 실행코드의 모든 클래스를 받아야 하기 때문에 식④는 다음과 같아진다.

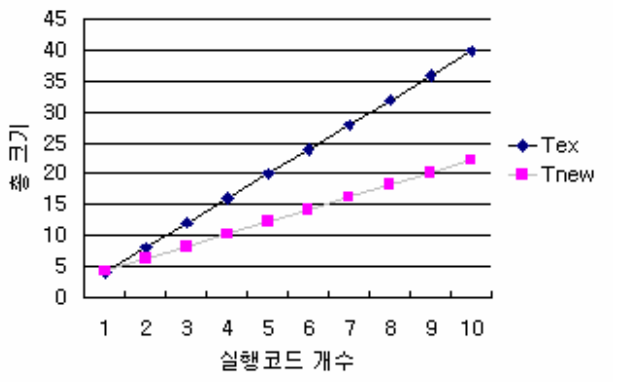
$$T_{new} = |\mathrm{Re}\, q_m| + |L_{vcc}| + |\mathrm{Re}\, q_{ca,\forall_n Lib_{vcc}}| + |\forall_n Lib_{vcc,n}| \quad ⑤$$

식⑤는 식①, ②에 의해 식⑥과 같이 변경된다.

$$T_{new} = T_{ex} + |L_{vcc}| \quad ⑥$$

식⑥에 의하면 새로운 메커니즘이 기존의 메커니즘에 비해 $|L_{vcc}|$ 만큼 더 큰 패킷을 전송하게 된다. 하지만 $|L_{vcc}|$ 은 수십 바이트도 되지 않은 작은 양이기 때문에 기존의 메커니즘과 새로운 메커니즘의 성능 차는 거의 없다. 하지만 하나 이상의 실행코드를 전송하거나 이전에 받아놓은 실행코드의 클래스들을 가지고 있다면, 식④의 마지막 항인 $|\exists_n Lib_{vcc,n}|$ 의 크기를 크게 줄일 수 있다. 또한 실행코드를 많이 전송하면 할수록 중복되는 클래스가 많아지게 되어 T_{new} 값은 현저히 낮아지게 된다.

예를 들어 같은 크기의 클래스를 4개 가지는 서로 다른 실행코드들이 서로 두 개의 클래스를 공통으로 가지고 있다고 가정하면, 아래 그래프와 같이 실행코드를 많이 전송할수록 T_{ex} 에 비해 T_{new} 값이 현저히 낮아지는 것을 볼 수 있다.



[그림 2] T_{ex} 와 T_{new} 비교 그래프

위 그래프를 보면 기존의 메커니즘과 새로운 메커니즘을 다음과 같은 공식으로 나타낼 수 있다. 실행코드를 구성하는 하나의 클래스를 라고 하면

$$T_{ex} = 4x \quad ⑦$$

$$T_{new} = 2(x-1) + 4 + \alpha = 2x + 2 + \alpha \quad ⑧$$

$$T_{ex} - T_{new} = 4x - (2x + 2 + \alpha) = 2x - 2 - \alpha \quad ⑨$$

식⑨를 보면 x 가 1일 때는 $-\alpha$ 만큼의 손해를 보게 되지만, 이 값은 매우 작은 값이고, x 값이 커짐에 따라 기울기 2값으로 인해 선형적으로 값이 증가하게 된다.

4. 결론 및 향후 계획

본 논문에서 액티브 패킷에 의해 전달되는 실행코드를 클래스 별로 구분하여 필요한 부분만 전송하게 하여 성능이 향상되는 것을 이론적으로 알아보았다. 현재 기존의 메커니즘으로 테스트베드가 구현되어 있고, 앞으로 새로운 메커니즘을 구현하여 실제 테스트 값으로 이 연구를 현실성을 증명할 것이다. 또한 본 논문에서 아직 고려하지 않은 액티브 패킷에 대한 인증 처리, 자주 사용되는 클래스에 대해 긴 저장 시간을 가지게 하여 클래스 중복 효과를 극대화할 수 있는 정책에 대한 방법을 연구할 것이다.

참고문헌

[1] D. L. Tennenhouse, et al., “ Towards an active network architecture”, In Multimedia Computing and Networking ’ 96, Jan. 1996

[2] K. L. Calvert, et al., “ Directions in Active Networks”, IEEE Communications, Oct. 1998, pp. 72-78

[3] Young J. Han, et al, "SVAM: the Scalable Vulnerability Analysis Model based on Active Networks", The International Conference on Information Networking(ICOIN) 2004, Proceedings Vol.I pp.313~322, Feb. 2004.