Journal of the Korea Academia-Industrial cooperation Society
Vol. 18, No. 11 pp. 94-103, 2017

https://doi.org/10.5762/KAIS.2017.18.11.94
ISSN 1975-4701 / eISSN 2288-4688

SW 보안 취약점 자동 탐색 및 대응 기술 분석

오상환, 김태은, 김환국*
한국인터넷진흥원

Technology Analysis on Automatic Detection and Defense of SW Vulnerabilities

Sang-Hwan Oh, Tae-Eun Kim, HwanKuk Kim*
Korea Internet & Security Agency

요 약 자동으로 해킹을 수행하는 도구 및 기법의 발전으로 인해 최근 신규 보안 취약점들이 증가하고 있다. 대표적인 취약점 DB인 CVE를 기준으로 2010년부터 2015년까지 신규 취약점이 약 8만건이 등록되었고, 최근에도 점차 증가하는 추세이다. 그러나 이에 대응하는 방법은 많은 시간이 소요되는 전문가의 수동 분석에 의존하고 있다. 수동 분석의 경우 취약점을 발견하고, 패치를 생성하기까지 약 9개월의 시간이 소요된다. 제로데이와 같은 빠른 대응이 필요한 취약점에 대한 위험성이 더 부각되는 이유이다. 이와 같은 문제로 인해 최근 자동화된 SW보안 취약점 탐색 및 대응 기술에 대한 관심이 증가하고 있다. 2016년에는 바이너리를 대상으로 사람의 개입을 최소화하여 자동화된 취약점 분석 및 패치를 수행하는 최초의 대회인 CGC가 개최 되었다. 이 외에도 세계적으로 Darktrace, Cylance 등의 프로젝트를 통해 인공지능과 머신러닝을 활용하여 자동화된 대응 기술들을 발표하고 있다. 그러나 이러한 흐름과는 달리 국내에서는 자동화에 대한 기술 연구가 미비한 상황이다. 이에 본 논문에서는 자동화된 SW 보안 취약점 탐색 및 대응 기술을 개발하기 위한 선행 연구로서 취약점 탐색과 대응 기술에 대한 선행 연구 및 관련 도구들을 분석하고, 각 기술들을 비교하여 자동화에 용이한 기술 선정과 자동화를 위해 보완해야 할 요소를 제안한다.

Abstract As automatic hacking tools and techniques have been improved, the number of new vulnerabilities has increased. The CVE registered from 2010 to 2015 numbered about 80,000, and it is expected that more vulnerabilities will be reported. In most cases, patching a vulnerability depends on the developers' capability, and most patching techniques are based on manual analysis, which requires nine months, on average. The techniques are composed of finding the vulnerability, conducting the analysis based on the source code, and writing new code for the patch. Zero-day is critical because the time gap between the first discovery and taking action is too long, as mentioned. To solve the problem, techniques for automatically detecting and analyzing software (SW) vulnerabilities have been proposed recently. Cyber Grand Challenge (CGC) held in 2016 was the first competition to create automatic defensive systems capable of reasoning over flaws in binary and formulating patches without experts' direct analysis. Darktrace and Cylance are similar projects for managing SW automatically with artificial intelligence and machine learning. Though many foreign commercial institutions and academies run their projects for automatic binary analysis, the domestic level of technology is much lower. This paper is to study developing automatic detection of SW vulnerabilities and defenses against them. We analyzed and compared relative works and tools as additional elements, and optimal techniques for automatic analysis are suggested.

Keywords : Automatic-Analysis, Concolic-Execution, Fuzzing, Patch-Generation, Security-Vulnerability, Symbolic-Execution

이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임.(no. 2017-0-00184, 자기학습형 사이버 면역 기술 개발)

*Corresponding Author : HwanKuk Kim(Korea Internet & Security Agency)
Tel: +82-61-820-1272 email: rinyfeel@kisa.or.kr

Received October 12, 2017 Revised November 2, 2017
Accepted November 3, 2017 Published November 30, 2017

1. 서론

초연결 시대를 맞아 스마트 카, 스마트 빌딩[1] 등 모든 사물이 상호 연동 되는 시대가 도래 하였다. 이는 모든 사물들에 SW가 설치되어 사용됨을 의미한다. 그에 따라 다양하게 사용되는 SW들의 보안 취약점에 대한 이슈가 발생하고 있다. 더욱이, 자동으로 해킹을 수행하는 도구 및 기법의 발전으로 인해 신규 SW 보안 취약점들이 증가하고 있다. 공개된 SW 보안 취약점 정보인 CVE(Common Vulnerabilities and Exposure)를 기준으로 2010년에서 2015년 사이 약 8만건의 신규 CVE가 등록되었고,[2] 이는 SW 보안 취약점 발생 속도가 빠르게 증가하고 있음을 나타내고 있다. 이와 같이 급증하는 취약점에 비해 대응하는 기술 대부분은 전문가의 수동 분석에 의존하고 있다.[3] 수동 분석을 통해 분석가가 취약점을 발견하고, 대응하는 패치를 생성하는데 약 9개월의 시간이 소요된다. 전문가가 취약점을 발생시키는 입력값을 생성하고, 발생한 취약점 유형 및 위치 등을 분석하고, 해당 취약점 유형과 위치에 적합한 패치 코드를 생성 및 검증하는 모든 과정을 수동으로 수행해야하기 때문이다. 이는 제로데이 취약점과 같이 빠른 대응이 필요한 경우 약 9개월이 소요되는 수동 분석으로 대응하기에는 큰 어려움이 발생 된다. 또한, 대부분의 SW 보안 취약점을 패치 하는 기술은 대상 SW의 소스 코드를 기반으로 분석하고, 해당 취약점에 대한 패치 코드 또한 소스 코드를 기반으로 작성하고 있다. 하지만, 패치 지원이 중단된 구버전 SW나 변형된 오픈소스 SW 등과 같은 레거시 SW의 경우 소스 코드 관리가 어려워 해당 SW에 취약점이 발생하면 대응하는데 많은 어려움이 발생한다. 이와 같은 한계점들로 인해 최근 SW 바이너리를 대상으로 보안 취약점을 자동으로 분석하여 패치를 수행하는 연구의 필요성이 대두되고 있다. 이러한 추세에 따라 2016년 미국 국방성(DARPA)에서 '데프콘 CTF 24'의 부대 행사로 자동화된 SW 바이너리 보안 취약점 자동 분석 및 패치를 수행하는 대회인 CGC(Cyber Grand Challenge)를 개최하였다.[4] 이 대회는 사람의 개입을 최소화 하여 기계가 자동으로 취약점을 찾아 공격하고, 방어 패치를 생성하는 최초의 대회였다. 이는 곧 사람이 아닌 기계가 해킹을 자동화하여 취약점을 찾고 공격하는 시대가 도래했음을 나타내고 있다. 이러한 추세에 맞춰 세계적으로 인공지능과 머신러닝을 활용하여 자동화된 기술들을 선보이고 있다.

먼저, 다크트레이스(Darktrace)의 머신러닝 기반 UBS(User Behavior Analysis) 솔루션은 영국 캠브리지에서 2013년 7월 설립하여 수학자들과 미국/영국의 정보 요원들의 협업으로 시작된 프로젝트로, 머신러닝과 수학알고리즘을 기본으로 하는 사용자 행위 분석 솔루션으로 룰(Rule)이나 패턴으로 탐지하기 힘든 이상 행위를 정상 행위에 기반한 비정상행위 탐지를 위해 인간의 면역체계를 모티브로 구성하였다.[5]

두 번째, Cylance의 차세대 안티 바이러스는 인공지능을 활용하여 엔드 포인트에서 실시간으로 악성코드를 탐지하는 솔루션으로 악성코드를 식별하기 위해 샌드박스나 시그니처 대신 머신러닝 기술을 사용하여 새로운 악성코드, 바이러스 및 변종을 탐지한다.[6] 이 외에도 CMU의 Mayhem[7], IBM의 Cognitive Security[8] 등 해외에서는 연구가 활발히 진행되고 있다. 그러나 국내에서의 자동화 연구는 기초 연구 단계 수준에 머물고 있다.

이에 본 논문에서는 Fig. 1과 같이 자동으로 SW 보안 취약점을 찾고, 대응하는 기술을 개발하기 위한 선행 연구로서 SW 보안 취약점을 탐색하는 기술과 대응하는 기술에 관한 선행 연구 및 관련 도구들을 분석하고, 각 기술들을 비교하여 자동화에 용이한 기술 선정과 보완할 요소를 제안한다.

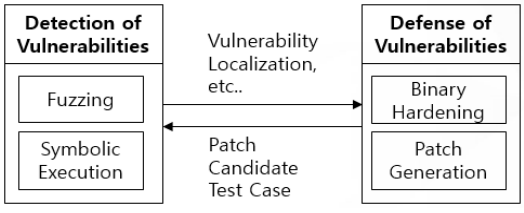


Fig. 1. Automatic Detection and Defense of Vulnerabilities Architecture

2. 관련 기술 개념

2.1 SW 보안 취약점 탐색 기술

2.1.1 퍼징 (Fuzzing)

일반적으로 SW의 취약점을 발견하기 위해 소스코드가 공개되어 있는 SW의 경우 해당 코드를 라인 별로 분석하고, 소스코드가 공개되지 않은 SW의 경우에는 리버

싱 등을 통해 분석하는 경우가 대부분이다. 하지만, 이 외에도 많이 사용되고 있는 방법이 바로 퍼징(Fuzzing)이다. 퍼징은 일종의 SW 보안 테스트 기법으로서 해당 SW에 입력되는 값을 무작위로 대입하여 주입함으로서 예상치 못한 결함(exception, crash 등)을 탐지하고, 그 결과를 분석하여 취약점을 발견하는 기술이다.[9] 이러한 퍼징 기술은 그 공격 기법이나 방식, 타겟 등에 따라 다양한 종류로 구분할 수 있다. 타겟의 경우 만해도 파일 포맷, 네트워크 프로토콜, 커맨드라인 파라미터, 환경변수, 웹 어플리케이션 등 다양하다. 이러한 분류 중에서 가장 기본이 되는 방법은 테스트 케이스(Test Case)를 어떻게 생성하는지에 따라 분류하는 것이다. 즉, 퍼징을 위해 입력 값을 어떻게 생성하는지에 따라 크게 뮤테이션(Mutation)과 스마트(Smart) 기법으로 분류 된다. 먼저 뮤테이션 기법은 주어진 입력 값을 무작위로 생성하여, 퍼징을 수행하는 기법으로 대상 SW에 대한 정확한 분석 과정이 필요 없다는 장점이 있으나, 취약점으로 이어지는 유효한 결함을 탐색하는 정확도가 떨어진다는 단점이 존재한다. 이와 반대로 스마트 퍼징은 대상 SW의 분석을 통해 해당 형식에 맞는 입력 값을 생성하여 보다 효율적인 결함 탐색 정확도를 보이는 기술로서 대표적인 연구로는 Peach 퍼저가 있다.[10] 이러한 분류 외에 가장 최근에 연구된 기술로 퍼징과 콘콜릭 실행(Concolic Execution)을 결합한 하이브리드 퍼징(Hybrid Fuzzing)이 발표되었다. 이 기술은 퍼징의 빠른 탐색 속도와 콘콜릭 실행의 높은 코드 커버리지 탐색 능력을 결합한 기술로서 상호 연동을 통해 무작위 대입이라는 퍼징의 불완전성과 경로 폭발이라는 콘콜릭 실행의 단점을 모두 해결한 기술로서 대표적인 연구로는 Driller가 존재한다.[11]

2.1.2 기호 실행 (Symbolic Execution)

기호 실행(Symbolic Execution)은 SW 안에 실행 가능한 모든 경로를 분석하는 기술로 입력 값을 실제 값(Value)이 아닌 특정 기호(Symbol)로 두고 분기문이나 반복문 등을 따라 최종 지점까지 가는 경로식을 기록하는 기술이다. 해당 기술은 버그를 발견하거나 프로그램 검증, 디버깅 등의 목적으로 사용되는 기술로 취약점 탐색에서는 다양한 테스트 케이스 중에서 취약점으로 도달할 수 있는 최적화된 경로를 파악하는 용도로 사용되고 있다. 하지만 실제 SW에 적용하는데 있어서 다수의 반복문이나 분기문에 의해서 너무 많은 실행 경로가 발생하거나 무한루프 등에 빠지는 경로 폭발(Path Explosion) 등의 한계점이 존재한다. 이러한 한계점을 개선하기 위한 시도로서 기호 실행과 실제 수행(Concrete Execution)을 결합하여 실행 경로를 분석하는 콘콜릭 실행이 있다. 이 기술은 실제 값을 입력하여 해당 입력 값에 해당하는 하나의 경로만을 실행하기 때문에 경로 폭발 문제를 해결하였고, 대표적인 연구로는 KLEE가 있다.[12] 다른 시도로는 기호 실행 엔진을 병렬화(Parallel)하여 끝없이 확장 가능한 리소스 확보를 통해 해결점을 제시한 Cloud9이 존재한다.[13]

2.2 SW 보안 취약점 대응 기술

본 논문에서는 대응 기술을 잠재적인 위험들로부터 보호하기 위한 예방 기법과 실질적인 취약점을 제거하는 패치 기술로 분류한다.

먼저 예방 기법은 공격자들이 대상 SW 바이너리의 취약점을 찾기 위한 행위를 방어 또는 예방하기 위한 바이너리 강화(하드닝) 기법이 존재한다.

대표적으로 메모리상의 공격을 어렵게 하기 위해 매 실행 시 스택, 힙, 라이브러리 등의 메모리 주소를 랜덤하게 배치하는 ASLR(Address Space Layout Randomization), 스택과 힙 영역의 코드가 실행되는 것을 막는 DEP/NX, 스택상의 변수들의 공간과 스택 프레임 포인터(SFP) 사이에 특정한 값(Canary)을 추가하여 스택의 무결성을 검증하는 SSP 등이 있다.

패치 기술은 취약한 부분의 코드를 삭제하거나 수정, 삽입 등을 통해 해당 취약점을 직접적으로 제거하는 기술이다.

대표적으로 유전알고리즘을 활용하여 패치 코드를 생성하여 적용한 Genprog,[14] 오류 보고서 정보를 기반으로 패치 코드를 생성하는 R2Fix,[15] 전문가가 작성한 패치 정보를 활용한 PAR[16] 등의 선행 연구들이 존재한다.

3. SW 보안 취약점 자동 탐색 기술

3.1 퍼징을 활용한 자동 탐색 기술

3.1.1 스마트 퍼징 (Peach 퍼저)

퍼징은 대상 SW에 무작위로 입력 값을 생성하여 주

입함으로서 예상치 못한 결함을 발견하는 기술로 해당 결함이 정말 치명적인 취약점으로 연결되는 유효한 결함을 찾는 정확도가 떨어지는 단점이 존재한다. 이에 대상 SW에 대한 분석을 통해 보다 효율적인 퍼징 기법인 스마트 퍼징이 존재한다.

스마트 퍼징 기법은 파일의 포맷이나 프로토콜 등을 분석하여, 해당 형식에 맞는 입력 값을 생성하는 기법으로 대상 SW를 분석하는 과정이 오래 걸린다는 단점이 있으나, 유효한 결함을 탐색하는 정확도가 높다는 장점이 존재한다. 스마트 퍼징을 수행하는 오픈소스 도구로는 Peach 이다. Peach 퍼저의 경우 퍼징 도구들 중 가장 발전된 형태로 알려져 있는 스마트 퍼저이며, 정확히는 뮤테이션과 스마트 기법 둘 다 사용가능한 도구이다.

```
<StateModel name="State" initialState="Initial">
    <State name="Initial">
  <Action type="call" method="StartServer" publisher="Peach.Agent" />
        <Action type="output">
            <DataModel ref="HttpRequest"/>
            <Data ref="HttpGet"/>
        </Action>
  <Action type="close" />
    </State>
</StateModel>
<Agent name="LinuxAgent">
  <Monitor class="LinuxCrashMonitor" />
  <Monitor class="Process">
    <Param name="Executable" value="web_server" />
    <Param name="StartOnCall" value="StartServer" />
  </Monitor>
  <Monitor class="LinuxDebugger">
    <Param name="Executable" value="web_server" />
  </Monitor>
</Agent>
<Test name="Default">
  <Agent ref="LinuxAgent"/>
        <StateModel ref="State"/>
        <Publisher class="Tcp">
                <Param name="Host" value="127.0.0.1" />
                <Param name="Port" value="8000" />
        </Publisher>
        <Logger class="File">
            <Param name="Path" value="logs"/>
        </Logger>
</Test>
```

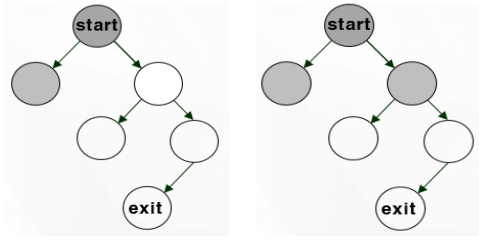
Fig. 2. PeachPit Format

Peach 퍼저를 사용하기 위해서는 대상 SW 입력 형식에 맞는 데이터 구조, 타입 등을 정의하는 Fig. 2와 같은 XML 형태의 PeachPit 파일 작성을 통해 대상 모델링을 수행하고 동작하게 된다. Peach 퍼저는 2.x 버전은 파이썬(Python), 3.x 버전부터는 C#으로 구현되어 있고, 리눅스(Linux)와 윈도우(Windows) 환경에서 동작이 가능하며 파일 퍼징(File Fuzzing)과 네트워크 퍼징(Network Fuzzing)이 가능한 도구이다. 특징으로는 크래시 모니터, 디버거 등의 기능을 제공하는 에이전트가 존재하고, Cert Triage Tool을 이용한 익스플로잇(Exploitable) 분류 정보를 제공하며 자동화가 가장 잘 구현되어 있는 도구이다. Peach 퍼저는 Agent, Analyzers, Fixups, Logger, Mutate Strategies, Mutator, Publisher, Transformers 등 8개의 주요 모듈로 구성되어 있고, 그 중 모니터링과 디버거 연동, 프로세스 제어기능을 제공하는 Agent 모듈, 입력 값 변이 수행 방법을 정의하는 Mutate Strategies 모듈, 실질적인 데이터 변이를 수행하는 Mutator 모듈, 데이터 송수신을 위한 I/O 인터페이스를 정의하는 Publisher 등 4개의 모듈이 핵심적인 역할을 수행한다.

3.1.2 하이브리드 퍼징 (Driller)

하이브리드 퍼징 기법은 무작위 입력 값을 생성하여 대입하는 퍼징과 프로그램의 실행 경로를 추적하는 콘콜릭 실행을 장점을 결합한 가장 진화된 취약점 자동 탐색 기술이다. 하이브리드 퍼징 은 퍼저의 불완전성과 콘콜릭 실행의 경로 폭발 문제를 해결하였다. 하이브리드 퍼징을 가장 잘 구현한 도구로는 Driller가 존재한다.

Driller는 퍼저를 사용하여 프로그램의 초기 구획을 탐색하고, 조건문 등에 의해 진행이 중지 되면 콘콜릭 엔진을 활용하여 다음 구획으로 유도하게 되고, 퍼저가 다시 인계받아서 탐색을 진행함으로서 더욱 빠르게 깊은 경로에 있는 취약점까지 자동으로 탐색하게 되었다. Driller는 AFL(American Fuzzy Lop)과 Angr[17]을 활용하여 하이브리드 퍼징을 구현하였다. AFL은 유전 알고리즘을 통해 입력 값을 생성 및 변형 하는 퍼저이고, Angr은 바이너리 코드를 Valgrind의 VEX IR로 변환하여 기호 실행을 수행하는 엔진으로 Mayhem과 S2E에 의해 가장 최적화된 형태의 기호 실행 엔진으로 알려져 있다.



(a) (b)

Fig. 3. Driller Flow 1
(a) The nodes initially found by the fuzzer
(b) The nodes found by the first invocation of concolic execution.

Driller는 Fig. 3. (a)와 같이 AFL을 통해 퍼징을 수행하고, 퍼저가 더 이상 추가적인 상태 전이를 찾을 수 없다 판단될 경우 즉, 새로운 상태 전이 경로를 찾기 위해 Fig. 3. (b)와 같이 콘콜릭 실행 엔진인 Angr을 호출한다. 이 때, 퍼저가 추가적인 상태 전이 경로를 찾지 못하

는 주된 원인은 SW 내에 복잡한 조건문을 충족시키기 위한 특정 입력 값을 생성할 수 없기 때문에 발생하게 된다. 해당 시점에서 제어권을 넘겨받은 콘콜릭 실행 엔진은 제약조건(Constraint) Solver를 활용하여 복잡한 조건을 충족시키는 입력 값을 생성하게 된다.

생성된 값은 퍼저의 큐에 전달되고, Fig. 4와 같이 제어권 역시 퍼저로 넘겨 퍼징을 수행하게 된다. Driller은 이러한 과정을 반복하여 수행함으로서 빠르고 깊은 경로를 탐색할 수 있게 된다. 이러한 분석 흐름에서 효율성을 결정짓는 중요한 요소는 콘콜릭 실행에 내제된 한계점인 경로 폭발을 피할 수 있다는 것이다. 이는 퍼징을 통해 생성된 입력 값에 의해 제한된 수행 경로만 분석하기 때문이다.

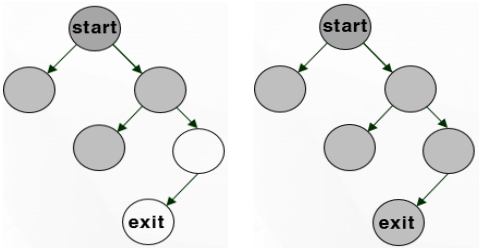


Fig. 4. Driller Flow 2
(a) The nodes found by the fuzzer, supplemented with the result of the first Driller
(b) The nodes found by the second invocation of concolic execution

3.2 기호 실행을 활용한 자동 탐색 기술

3.2.1 콘콜릭 실행 (KLEE)

오픈 소스로 제공되는 KLEE는 LLVM Bitcode로 컴파일 된 프로그램을 인터프리트해 콘콜릭 실행을 수행하는 대표적인 도구이다. KLEE는 심볼릭 프로세스의 운영체제와 인터프리터 역할을 동시에 수행한다. 심볼릭 프로세스는 실제 콘콜릭 실행 경로를 따라가며 분기 조건을 저장하는 프로세스로, State라 표현한다. 각 State는 프로세스와 동일하게 스택, 힙, 프로그램 카운터(PC) 등의 정보를 갖고 있고, 자신이 수행한 경로 조건 정보를 축적해 나중에 경로 종료 시 테스트 케이스 도출에 활용한다.

Fig. 5와 같이 구성된 KLEE는 현재 수행하는 State가 분기 지점에 도달하면 해당 분기의 조건이 true로 갈 수 있는지, false로 갈 수 있는지의 여부를 판단하기 위해 STP solver에 현재까지의 심볼릭 경로 수식에 현재 분기 조건을 더해서 쿼리를 보낸다. true와 false 모두 가능한 경우 State를 fork하여 자식 State를 생성해 그 State가 false인 경로 조건을 수행하도록 false 경로 조건을 추가하고, 자신은 true인 경로 조건을 추가한다. 그리고 현재 모든 실행 가능한 State들 중 하나를 택하고, 그 State의 프로그램 카운터가 가리키는 instruction 수행을 계속해 모든 가능한 State를 전부 분석할 때까지 또는 주어진 제한 시간까지 수행한다. KLEE는 각 instruction 마다 수행 가능한 State들 중 하나를 택해 수행하는 스케줄링 방법을 사용한다. depth, icnt, cpicnt, query-cost, md2u, covnew 등 총 6개의 NURS(Non Uniform Random Search) 방법을 제공하며 상응하는 조건을 만족하는 State를 확률적으로 우선 선택하는 스케줄링과 Round Robin 방식으로 일정 시간, 혹은 일정 instruction 개수 수행 후 다음 수행 State를 선택하는 배칭(Batching) 기법을 함께 사용한다.

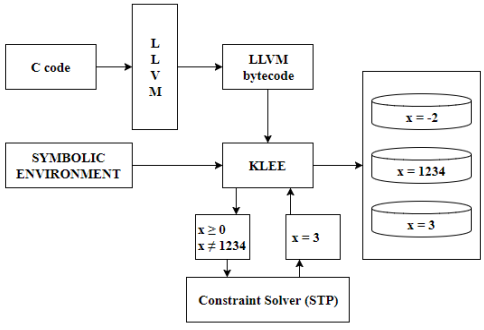


Fig. 5. KLEE Architecture

3.2.2 Parallel Symbolic Execution (Cloud9)

Cloud9은 효과적이지만 확장성이 낮은 기호 실행(Symbolic Execution)을 대형 비공유 클러스터에 병렬화한 최초의 기호 실행 엔진이다. 아마존(Amazon) EC2와 같은 클라우드 환경에서 실행되며, 기호 실행을 통해 대상 SW의 버그나 취약점을 일으키는 입력을 찾는 테스트 용도로 활용된다. Cloud9은 기호 실행의 가장 큰 제약 사항인 경로 폭발, 총 실행 시간의 절반 이상을 소모하는 제약조건 Solving Overhead, 경로 폭발로 인해 실행 경로의 상당 부분을 탐색하기도 전에 메모리 부족으로 종료되는 High Memory Usage 등을 클러스터에서 기호 실행 엔진을 병렬 처리함으로서 끝없이 확장 가능한 메모리 및 CPU를 확보함으로서 해결하였다. Cloud9

의 기본 구성은 작업 부하를 분산하여 배치하는 1개의 Load Balancer와 실질적인 기호 실행을 수행하는 다수의 Worker로 이루어져 있다. 각각의 Worker는 독립적으로 기호 실행과 제약조건 Solver를 활용하여 실행 경로를 탐색한다. 먼저 Worker가 프로그램의 분기를 만났을 경우, 실행을 중지하고 Searcher를 호출한다. Searcher는 어떤 경로를 다음에 실행할지 선택한 뒤 결과를 리턴하면, Worker는 제약조건 Solver를 호출하여 선택된 경로가 실행 가능한 경로인지를 검사하고, 가능한 경로일 경우 해당 경로를 따라 실행하며, 분기 조건을 경로 제약에 추가한다. 만약, 실행이 불가능한 경로일 경우 새로운 경로 선택을 요청하게 된다. Load Balancer는 각 Worker들의 상태 대기열에서의 평균 대기시간, 메모리 소비량, 제약조건 Solver에 전달되는 쿼리 수 등의 정보를 기반으로 Worker들 간의 부하를 조정하는 기능을 수행한다.

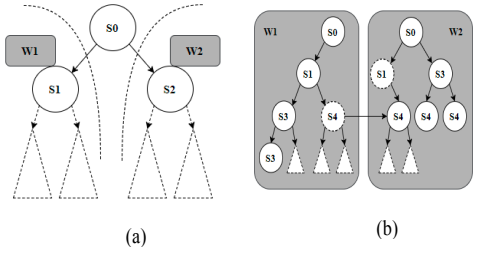


Fig. 6. Cloud9 Flow
(a) First Branch (b) Load Balancing

먼저 Fig. 6. (a)와 같이 첫 번째 분기 명령어의 한쪽 경로를 Worker W1이 탐색하고, 다른 경로는 Worker W2가 탐색한다. 탐색 중 Fig. 6. (b)처럼 W2는 W1이 탐색을 종료하기 이전에 탐색을 종료하기 때문에 작업 불균형이 발생하게 된다. 이 때, Load Balancer는 state S4를 root로 하는 하위 경로를 W2에 위임함으로서 효율적인 탐색을 수행하게 된다. 위와 같은 병렬 기호 실행 탐색 수행은 단일 노드 기호 실행 엔진에 비해 높은 탐색 속도를 확보할 수 있다.

4. SW 보안 취약점 대응 기술

4.1 바이너리 강화를 활용한 대응 기술

4.1.1 OS 레벨의 메모리 강화 기술

운영체제 레벨에서 적용 가능한 보안 기술로는 ASLR, DEP/NX, ASCII-Armor 등이 있다.

첫 번째, ASLR(Address Space Layout Randomization)은 프로그램이 가상메모리에 매핑 될 때, 이미지 베이스(Image Base) 값을 랜덤하게 부여하는 기술이다. 이는 공격자가 대상 프로그램의 메모리 구조 파악을 어렵게 함으로서 공격을 예방하는 보안 기술이다.

두 번째, DEP/NX(Data Execution Prevention/Not Executable)는 스택과 힙 등 데이터 영역에서 코드가 실행되는 것을 막는 기술이다. 이는 공격자가 버퍼 오버플로우 공격 시 DEP/NX가 적용된 상태에서는 스택이나 힙 영역에 실행 권한이 없어 프로그램이 종료됨으로서 공격을 예방하는 기술이다.

세 번째, ASCII-Armor는 공유 라이브러리 영역의 상위주소에 NULL(\x00)바이트를 삽입하여, 해당 영역을 보호하는 기술이다. 이는 공유 라이브러리를 호출하는 버퍼 오버플로우 공격 시, 상위주소에 삽입된 NULL 바이트로 인해 해당 주소로 접근이 불가능하게 함으로서 공격을 예방한다.

4.1.2 컴파일러 레벨의 바이너리 강화 기술

컴파일러에서 적용 가능한 보안 기술은 PIE, SSP, RELRO 등이 있다.

첫 번째, PIE(Position Independent Executable)은 운영체제에서 제공하는 ASLR과 유사한 기술로 컴파일 과정에서 상대주소를 적용하여 매 실행 시 매핑되는 주소가 랜덤으로 배치하게 된다. ASLR과의 차이점은 ASLR은 프로그램이 매핑되는 메모리 영역중 스택, 힙, 공유 라이브러리 영역에 랜덤 주소가 적용되고, PIE는 바이너리 자체를 상대주소를 적용하여 랜덤화 한다는 점이 가장 큰 차이점이다. 이와 유사한 방법으로 컴파일 시 공유 라이브러리 영역을 랜덤화 하는 PIC(Position Independent Code) 기술이 있다.

두 번째, SSP(Stack Smashing Protector)는 스택 상의 변수들의 공간(Buffer)과 SFP(Saved Frame Pointer)사이에 모니터링 역할을 하는 특정한 값(Canary)을 추가함으로서, SFP가 덮어지는 공격을 막는 기술이다. 공격자에 의해 버퍼 오버플로우 공격이 발생한 경우, SFP를 덮기 위해 Canary 값을 거쳐가는 과정에서 Canary 값의 변조가 일어나는 과정을 통해 공격을 탐지하게 된다. Canary 값은 종료문자(CR, LF 등)로 구성하는 Terminator

Canaries, 임의의 Canary 값을 구성하는 Random Canary, NULL 문자로 구성하는 Null Canary 등 3가지 형태로 구성할 수 있다. Terminator Canaries의 경우 종료문자로 구성되어 공격자는 Canary 값에 접근하는 순간 프로그램이 종료가 되고, Random Canary는 공격자가 Canary 값 예측을 불가능하게 함으로서 공격 코드 구성을 어렵게 만들고, Null Canary는 공격자가 공격 코드를 구성할 때 Null 값을 삽입할 수 없으므로, Canary 값에 접근할 수 없게 된다.

세 번째, RELRO(Relocation Read-Only)은 Elf 바이너리 또는 프로세스의 데이터 영역을 읽기전용상태(Read-Only)로 만들어, 메모리가 변경 되는 것을 보호하는 기술로 GOT 영역의 상태에 따라 부분 RELRO와 전체 RELRO 두 가지 적용 방법이 존재한다. GOT 영역이 쓰기가능(Writable)한 부분 RELRO는 전체 RELRO에 비해 적은 리소스를 소모하여, 속도가 빠르다는 장점이 있지만 GOT Over-write 공격과 같이 GOT 영역을 이용한 공격에는 취약하다는 단점이 존재한다. 반면에 GOT 영역도 읽기전용상태로 만들어주는 전체 RELRO는 부분 RELRO에 비해 많은 리소스를 소모하여, 속도가 느리다는 단점이 존재하나, GOT 영역을 이용한 공격까지 막을 수 있다는 장점이 있다.

4.2 자동 패치 생성을 활용한 대응 기술

4.2.1 유전 알고리즘을 활용한 패치 (Genprog)

2009년 발표된 Genprog는 유전 알고리즘을 활용하여 자동 패치를 수행하는 기술로 이후에 연구된 자동 패치 관련 연구들에 가장 많은 영향을 끼친 기술이다.

Genprog는 C언어 기반의 프로그램을 자동으로 패치하는 기술로 대상 SW의 소스 코드 구조를 트리 형태로 표현한 AST(Abstract Syntax Tree)를 생성한 후, 이상 노드에 대해 삭제, 추가, 교체 3가지 수정 작업을 통해 패치를 수행 한다. 노드 수정을 위해서는 각 오류에 대한 템플릿을 이용한다. 연구에 따르면 일반적인 버그 105개 중 55개를 수정하였으며, 취약점으로는 Heap Buffer Overflow, Non-Overflow Dos, Integer Overflow, Format String 취약점을 패치를 수행 하였다.

4.2.2 오류 보고서 정보를 활용한 패치 (R2Fix)

R2Fix는 이미 완성된 SW에도 오류 보고서에 이미 알려져 있는 오류를 수정할만한 리소스 부족으로 수정하지 못한 많은 오류가 존재한다는 점을 모티브로 출발하였다. R2Fix는 과거의 오류 수정 패턴, 기계 학습, 패치 생성 기술을 결합하여 자동으로 패치를 수행하는 기술이다.

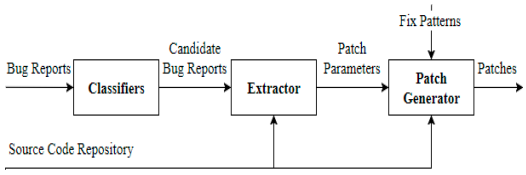


Fig. 7. R2Fix Architecture

R2Fix는 Fig. 7.와 같이 3종의 주요 모듈로 구성되어 있다. 기계 학습을 적용하여 오류 보고서로부터 오류 정보를 수집하고 자동으로 분류하는 Classifiers, 일반적인 파라미터 정보(파일명, 버전 등)와 오류 유형별 상세 정보(For Overflow: Buffer name, size, bound check condition 등) 2가지 형태의 파라미터를 추출하는 Extractor, 마지막으로 대상 SW에 적용되었던 과거 수정 패턴 등을 활용하여 패치 코드를 생성하고 적용하는 Patch Generator로 구성되어 있다.

연구에 따르면 R2Fix는 3종의 프로젝트인 Linux 커널, Mozilla, Apache를 대상으로 버퍼 오버플로우, Null Pointer 참조, 메모리 Leak 등 3가지 유형의 취약점 패치를 수행하였다. 또한, 일반적인 오류에 대한 57개의 패치를 생성하였고, 그 중 5개는 개발자가 예상하지 못한 새로운 오류에 대한 패치를 생성하였다.

4.2.3 전문가 작성 패치 정보를 활용한 패치 (PAR)

PAR는 Java를 기반으로 전문가들이 작성한 약 60,000개 이상의 패치 정보를 수동으로 분석하여 몇 가지 공통적인 패턴을 찾고, 해당 패턴 정보를 기반으로 자동으로 패치를 생성하는 연구이다.

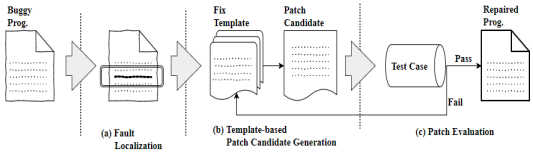


Fig. 8. Overview of PAR

PAR는 Fig. 8과 같이 (a) Fault Localization, (b) Template-based Patch Candidate Generation, (C) Patch Evaluation 등 총 3단계의 절차를 통해 패치를 생성한다.

(a) Fault Localization 단계에서는 Positive/Negative 테스트 케이스를 기반으로 각 실행 구문에 가중치를 할당하여 적용하는 통계적 결함 위치 파악 알고리즘을 사용한다. 이 단계에서 핵심은 Negative 테스트 케이스가 실행 된 구문이 결함일 가능성이 높다고 가정한다. 먼저, 두 가지 테스트 케이스를 모두 실행한 뒤 해당 경로를 기록한다. 해당 경로는 1) 두 그룹이 모두 방문한 실행 구문, 2) Positive만 방문한 실행 구문, 3) Negative만 방문한 실행 구문, 4) 두 그룹 모두 방문하지 않은 실행 구문 등 총 4가지 그룹으로 분류하여 가중치를 할당한다. 3)번에 해당되는 경우 1.0을 할당하고, 1)번의 경우 0.1, 그 외에는 0.0이 할당된다. 할당된 가중치에 따라 해당 위치에 후보 패치 생성을 수행하게 된다.

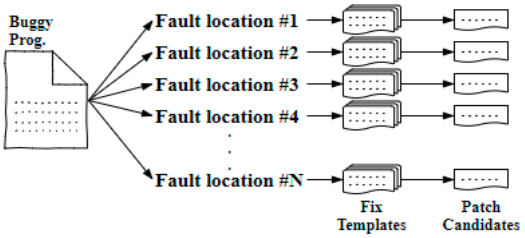


Fig. 9. Creating Patch Candidates

(b) Template-based Patch Candidate Generation 단계에서는 Parameter Replacer, Method Replacer, Expression Replacer 등 총 10개의 수정 템플릿을 각 결함 위치에 맞게 적용하여 Fig. 9과 같이 각각의 후보 패치를 생성한다. 후보 패치 생성에는 1) AST(Abstract Syntax Tree) Analysis, 2) Context Check, 3) Program Edition 등 3단계로 수행하게 된다. 1)단계에서는 대상 프로그램의 AST를 스캔하고 주어진 오류 위치와 그 인접 위치를 분석한다. 2)단계에서는 분석 된 AST를 검사하여 수정 템플릿으로 해당 위치를 수정할 수 있는지 여부를 검사하고, 수정이 가능한 경우 3)단계에서 수정 템플릿에 미리 정의 된 수정 스크립트를 기반으로 대상 프로그램의 AST를 다시 작성함으로서 후보 패치 생성을 완료하게 된다.

마지막 (C) Patch Evaluation 단계에서는 대상 프로그램의 이슈 트래커(Bugziila, JIRA 등)로부터 수집한 다양한 테스트 케이스를 활용하여 생성된 후보 패치의 적합성을 평가한다. 각 후보 패치를 대상으로 테스트 케이스를 수행하여 모든 테스트 케이스를 수행한 후보 패치를 최종 패치로 선정 및 적용하게 된다.

5. 결론 및 향후 연구 방향

본 논문에서 제시한 4종의 SW 취약점 탐색 관련 기술을 활용하여 구현된 도구들을 상대적으로 비교 분석한 결과 Table. 1과 같은 결과를 얻을 수 있었다.

Table 1. Fuzzing Tool Comparison Table

Tool	Method	Automated	code Coverage	Speed
Peach Fuzzer	Smart Fuzzing	Easy	Low	Fast
Driller	Hybrid Fuzzing	Medium	High	Medium
KLEE	Concolic Execution	Hard	High	Slow
Cloud9	Parallel Symbolic Execution	Hard	High	Slow

위 분석 결과 자동화의 용이성, 코드 커버리지, 탐색 속도를 비교했을 때, 가장 효과적인 취약점 탐색 기술은 하이브리드 퍼징이 적용된 Driller이다. 하지만, Driller도 대상 프로그램의 종류에 따라 천차만별의 실행 시간이 걸리는 퍼징 기술 자체의 문제점을 보유하고 있다. 이에 본 저자는 Driller에 대상 프로그램의 복잡도를 측정하여 가장 효율적인 퍼징 전략을 수립하는 기술을[18] 접목시켜 보다 빠른 속도로 취약점을 탐색하는 기술을 연구할 예정이다.

현재 국내·외적으로 바이너리를 대상으로 하는 대응 기술은 미비한 수준이다. 본 논문에서 설명한 대응 기술들 또한 소스 코드가 있는 SW를 대상으로만 적용 가능한 기술들이다. 때문에, 유지 보수가 중단된 레거시 SW나 서드파티(3-party) SW의 경우 소스 코드 확보가 어려워 해당 기술을 통한 취약점 대응에는 어려움이 있다. 이에 본 저자는 라이브러리 후킹을 통해 취약한 함수 대신 안전한 함수를 호출하는 LD_PRELOAD 후킹, PLT/GOT 테이블 수정을 통한 취약 함수 변환 등을 통해 소스 코드 없이 바이너리를 대상으로 취약점에 대응하는 연구를 진행할 것이다. 향후, 보다 개선된 취약점 탐색 기술과 바이너리 대상의 대응 기술을 연동하여, 자동으로 취약점을 탐색하고 대응하는 기술을 연구할 예정이다.

References

[1] Yeon-Suk Choi, “A Study on security characteristics and vulnerabilities of BAS(Building Automation System)”, Journal of the Korea Academia-Industrial, vol .18, no. 4, pp. 669-676, cooperation Society, 2017.
DOI: https://doi.org/10.5762/KAIS.2017.18.4.669

[2] U.S. National Vulnerability Database(NVD), CVE LIST, The MITRE Corporation, c2015(cited 1999), From: https://cve.mitre.org/cve/, (accessed Oct., 11, 2017).

[3] So-Yeon Min, Chan-Suk Jung, Kwang-Hyong Lee, Eun-Sook Cho, Tae-Bok Yoon, Seung-Ho You,“ Design of Comprehensive Security Vulnerability Analysis System through Efficient Inspection Method according to Necessity of Uptrading System Vulnerability”, Journal of the Korea Academia-Industrial cooperation Society, vol. 18, no. 7, pp. 1-8, 2017.
DOI: https://doi.org/10.5762/KAIS.2017.18.7.1

[4] Defense Advanced Research Projects Agency(DARPA), Program, DARPA, c2016, From: https://www.darpa.mil/program/cyber-grand-challenge, (accessed Oct., 11, 2017).

[5] Darktrace. Support_1, FRENTREE, c2013, From: http://www.frentree.com/Darktrace.pdf, (accessed Oct., 11, 2017).

[6] Cylance, White Papers, Cylance Inc, c2017, From: https://www.cylance.com/content/dam/cylance/pdfs/white_papers/MathvsMalware.pdf, (accessed Oct., 11, 2017).

[7] For All Secure, Unleashing-mayhem, For All Secure, 2016 Feb 9, From: https://forallsecure.com/blog/2016/02/09/unleashing-mayhem/, (accessed Oct., 11, 2017).

[8] IBM, Congnitive security white paper, IBM, c2000, From: http://www-03.ibm.com/security/kr/ko/cognitive/whitepaper/#cognitive-security-ibm-data-security, (accessed Oct., 11, 2017).

[9] P.Miller, L.Fredriksen, Bryan So, “An empirical study of the reliability of UNIX utilities”, Communications of the ACM, vol. 33, Issue 12, pp. 32-44, 1990.
DOI: https://doi.org/10.1145/96267.96279

[10] PeachTech, Peach Fuzzer Community Edition, Deja vu Security, 2014 Feb 23, From: http://community.peachfuzzer.com/WhatIsPeach.html, (accessed Oct., 11, 2017).

[11] Nick Stephens, John Grosen, Christopher Salls, Andrew Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, Giovanni Vigna, “Driller: Augmenting Fuzzing Through Selective Symbolic Execution”, the Network and Distributed System Security Symposium, 2016.
DOI: https://doi.org/10.14722/ndss.2016.23368

[12] Cristian Cadar, Daniel Dunbar, Dawson Engler, “KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs”, Operating Systems Design and Implementation, vol. 8, 2008.

[13] Liviu Ciortea, Cristian Zamfir, Stefan Bucur, Vitaly Chipounov, George Candea, “Cloud9: A Software Testing Service”, 3rd SOSP Workshop on Large Scale Distributed Systems and Middleware, vol. 43, no. 4, 2009.

[14] Stephanie Forrest, ThanhVu Nguyen, Westley Weimer, Claire Le Goues, “A Genetic Programming Approach to, Automated Software Repair”, Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 947-954, 2009.

[15] Chen Liu, Jinqiu Yang, Lin Tan, “R2Fix: Automatically Generating Bug Fixes from Bug Reports”, Proceedings of the International Conference on Software Testing, Verification and Validation, pp. 282-291, 2013.
DOI: https://doi.org/10.1109/ICST.2013.24

[16] Dong-Sun Kim, Jae-Chang Nam, Jae-Woo Song, Sunghun Kim, “Automatic patch generation learned from human-written patches”, Proceedings of the International Conference on Software Engineering, pp. 802-811, 2013.
DOI: https://doi.org/10.1109/ICSE.2013.6606626

[17] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, Giovanni Vigna, UC Santa Barbara, “SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis”, Security and Privacy (SP), pp. 138-157, 2016.

[18] Maksim O, Shudrak, Vyacheslav V.Zolotarev, “Improving Fuzzing Using Software Complexity Metrics”, International Conference on Information Security and Cryptology, pp. 246-261, 2015.

오 상 환(Sang-Hwan Oh) **[정회원]**

- 2014년 2월 : 전남대학교 컴퓨터공학과 (학사)
- 2014년 3월 ~ 현재 : 한국인터넷진흥원 보안기술R&D2팀 주임 연구원

<관심분야>
정보보호, 네트워크 보안, SW 취약점 분석

김 태 은(Tae-Eun Kim) **[정회원]**

- 2005년 2월 : 백석대학교 정보통신학부 (학사)
- 2007년 2월 : 숭실대학교 대학원 컴퓨터학과 (석사)
- 2017년 3월 ~ 현재 : 숭실대학교 대학원 컴퓨터학과 (박사과정)
- 2013년 7월 ~ 현재 : 한국인터넷진흥원 보안기술R&D2팀 선임 연구원

<관심분야>
네트워크 보안, 모바일 보안, 정보보호

김 환 국(Hwan-Kuk Kim) **[정회원]**

- 2000년 2월 : 한국항공대학교 컴퓨터공학과 (석사)
- 2001년 5월 ~ 2006년 12월 : 한국전자통신연구원 연구원
- 2017년 2월 : 고려대학교 정보보호대학원 정보보호학과 (공학박사)
- 2007년 1월 ~ 현재 : 한국인터넷진흥원 보안기술R&D2팀 팀장

<관심분야>
SW 취약점 분석, 네트워크 보안, IoT 보안