|  |
| --- |
| **IoT 디바이스 펌웨어 취약점 탐지 방법의 비교 분석** |
|  |
| **Comparative Analysis of IoT Device Firmware Vulnerability Detection Method** |
|  |
| **요 약**  다양한 IoT 기기들의 소프트웨어적 기반이 되고 있는 대표적인 오픈소스 펌웨어들을 선정하고 이를 대상으로 여러 가지 취약점 탐지 방법과 무료 도구들을 활용한 취약점 분석을 실시한다. 결과의 비교 분석을 통해 각 탐지 방법론에 대한 특성을 조사하고, 분석 도구들의 장단점 비교 및 취약점별 특성을 파악하여 취약점 탐지의 발전 방향을 제시해본다. |

1. **서 론**

최근 IoT 기술의 발전으로, 사물 인터넷은 개인의 편리성을 개선하는 분야뿐만 아니라 산업 모니터링, 공공 시스템 구축 등 사회의 전 분야에 걸쳐 확장되고 있다. 정보 기술 연구 및 자문회사 가트너(Gartner)에 따르면 2009년까지 IoT 기술을 사용하는 사물의 개수는 9억 여개였으나, 2020년에는 260억개에 이를 것으로 전망하고 있다. 이렇게 IoT 기술이 보편화되면서 미국의 보안 소프트웨어회사 Symantec은 IoT 기기들의 해킹 가능성에 대해 지적하였다. 다수의 IoT 기기의 기반인 리눅스 운영체제가 해당 기기들에서 올바른 보안요건을 갖추지 못하거나, 업데이트가 이루어지지 않을 경우 리눅스 웜에 의해 해킹 당할 위험이 존재하며, 실제로 미국에서는 TRENDnet 사의 아기 모니터링 카메라인 SecurView는 보안 취약점을 가진 채로 제품을 유통하여 700여 가구의 가정 내부 영상이 해커들에게 유출된 사례가 있었다.

이처럼 IoT 기기들의 모든 기기가 해킹의 대상이 될 수 있는 특성 때문에 보안의 중요성이 강조된다. 특히, IoT 디바이스들은 임베디드 시스템으로서 오픈소스 펌웨어들을 기반으로 발전하고 있다. 여러 기기에 널리 사용되고, 해당 오픈소스를 기반으로 하여 또 다른 애플리케이션이 제작되는 등 취약 코드가 확산되기에 매우 적합한 오픈소스의 특징 때문에, 기반 오픈소스들에 대한 취약점 분석을 통해 취약점들을 조기에 탐지하고 개선하는 연구가 필요하다.

우리는 다양한 취약점 분석 방법에 따라, 공개되어 있는 여러 가지 취약점 탐지 도구들을 이용하여 대표적인 기반 오픈소스 펌웨어들에 대해 분석하여 각 도구와 방법들의 장단점 비교분석 및 취약점별 탐지 특성 등을 파악하여 취약점 탐지의 발전 방향을 제시해보고자 한다.

1. **취약점 분석 방법**
   1. **소스코드 취약점 탐지**

소스코드 기반 취약점이란 소스코드 구조상 발생할 수 있는 소프트웨어 결함 및 버그에 관련된 취약점을 말한다. 이러한 취약점 중에는 보안상의 위험이 없는 것도 있다. 초보적인 실수나 잘못된 코드 등 개발자가 고려하지 못한 상황으로 인해 발생되며, 개발자의 document등의 구조분석으로는 탐지가 어렵다는 특징이 있다.

현재 다수의 소스코드 기반 취약점 분석 도구는 CVE 등의 사례 데이터베이스를 기반으로 한 rule-based 방식으로 이루어진다. 새로운 취약 사례가 발생하면 수동으로 분석하여 rule을 추가하는 방식이다.

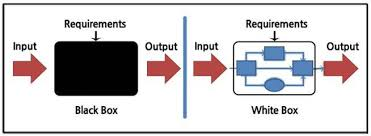
소스코드 자체를 기반으로 하는 정적 분석은 사전에 해당 코드 구조에 대한 이해에 대한 요구도가 낮으며, 퍼징, 기호실행 또는 다양한 동적 취약점 분석 등에 비해 분석도구의 사용이 용이하다는 장점을 가진다. 또한, visual studio나 eclipse 등 다양한 IDE 도구에서도 추가 플러그인 등으로 제공하기 때문에 개별 설치를 필요로 하는 몇몇 도구들에 비해 접근성 면에서 강점을 가진다. 그리고 분석 도구의 대상이 오픈소스나 외부 라이브러리 등일 경우에는 CVE등의 사례 DB를 통해 분석도구를 업데이트하여 효율적인 취약점 관리가 가능하다는 장점이 있다.

반면에 일반적 패턴이 아닌 고유 코드 상의 문제일 경우에는 일반적으로 기존의 분석이 부족할 가능성이 높아 코드의 구조를 일일이 파악하는 직접적인 분석을 요구로 한다는 점, rule의 개수와 취약 패턴의 종류에 의존적이어서 단일 방법론으로서는 퍼징과 기호실행의 탐지범위를 커버하기 어렵다는 점 등이 단점으로 꼽힌다.

* 1. **퍼징**

퍼징은 무작위로 생성한 데이터를 입력하여 버퍼오버플로우(BOF), 포맷스트링버그(FSB), 해제후 재사용 등과 같은 프로그램의 결함을 발견하기 위한 기법이다. 퍼징의 주목적은 미처 발견하지 못한 프로그램의 결함이나 보안취약점을 찾아 보완하려는 것이다.

퍼징 방법은 테스팅 방식에 따라 블랙박스 퍼징과 화이트박스 퍼징으로 분류할 수 있다.

**  
[Fig. 1] Black box testing and White box testing**

블랙박스 테스트는 소프트웨어를 검사할 때 내부 구조나 작동 원리를 모르는 상태에서 소프트웨어의 동작을 검사하는 방법을 이르는 말이다. 올바른 입력과 올바르지 않은 입력을 다 동원하여 올바른 출력을 판별하는 방식으로 검사가 이루어지게 되므로 대상 소프트웨어의 코드나 내부 구조 및 개발 노하우에 대한 정보는 필요하지 않다. 큰 시스템 테스팅에 유용하다는 장점이 있지만, 짧은 기간에 모든 입력 값을 테스트하기에 무리가 있고 논리적 오류를 검출하지 못한다는 단점이 있다.

화이트박스 테스트는 소프트웨어 내부 소스코드를 확인하는 방식으로 테스트가 진행된다. 동작의 유효성뿐만아니라 실행되는 과정을 살펴보면서, 어떤 경로로 실행되고 불필요한 코드가 있는지에 대한 여부, 테스트되지 못한 부분이 있는지에 대한 여부 등을 확인할 수 있다. 화이트박스 테스팅은 모든 코드를 명확히 확인할 수 있고 구체적인 테스트 시나리오를 짤 수 있는 장점이 있지만 화이트박스 테스트를 수행하기 위해 소스코드 상 입력 값에 대한 출력 값의 결과를 정확히 이해해야 하는 스킬이 필요하고 그러기 위해서는 자원이 많이 소모되며 모든 입력 값에 대한 테스팅을 진행할 수는 없다는 단점이 있다.

퍼징을 이용하면 프로그램의 예상하지 못한 충돌이나 잠재적 메모리 누수 등의 문제점을 무작위 데이터의 입력의 자동화를 통해 찾아 낼 수 있다는 장점이 있다. 그러나 퍼징을 위해서 대상 소프트웨어의 프로토콜 이나 스펙 또는 파일의 포맷을 파악하는 것은 쉽지 않은 작업이며, 입력 값이 랜덤하게 주어지므로 경계 값을 탐색하기 어렵다는 단점이 있으며 기호실행과 병행하여 테스팅을 하는 것으로 단점을 보완한다.

* 1. **기호 실행**

기호 실행은 변수들을 이용해 악성 코드들의 분기를 찾아내는 기법이다. 기호 실행은 바이너리의 분기를 제약조건으로 보고 제약조건들을 풀어내는 과정을 거친다. 즉, 특정 소프트웨어 프로그램의 분기를 수식화 하여 생성되는 분기점을 수행되도록 입력을 진행하는 것이다. 프로그램의 실행 단위를 분기로 구분하여 특정 조건에서 각 분기에 도달하는 과정을 수행시켜 정상 동작여부를 판별한다.

프로그램의 오작동을 재현하여 탐지하는 기법에는 기호 실행과 함께 퍼징도 포함된다. 기호 실행은 기호화를 통해 특정 코드 커버리지 조건을 만족하게 설정이 가능하므로 퍼징과 대비하여 불확실성 측면에서 우세하다. 기호실행의 경우 코드 단위에서만 탐지를 수행하기 때문에 바이너리 레벨에서 입력 값에 대한 조작을 수행하는 퍼징에 비하여 상대적으로 부족하다. 또한, 특정 프로토콜을 사용하는 프로그램의 경우 프로토콜의 잘못된 설계로 인해 발생하는 취약점은 탐지가 불가능하다. 해당 프로그램은 개발자가 의도한 방향으로 정상 동작하기 때문이다. 기호화를 위해서는 프로그램의 구조 파악이 선행되어야 하며, 분기가 너무 많아지는 프로그램은 기호 실행의 대상으로 적절하지 않다.

1. **취약점 탐지 도구 및 분석 대상 선정**
   1. **소스코드 취약점 탐지**

주요 탐지대상 application들이 c 또는 c++로 작성되었기 때문에 분석 도구 역시 c와 c++언어에 대하여 탐지할 수 있는 도구들로 선정하였다. 그리고 쉽게 접근하여 사용할 수 있는 오픈소스 도구들로 한정하였다. 그래서 YASCA, Boon, Flawfinder, Smatch, Uno, Clang, Splint, BLAST, Coccinelle, Cppcheck, Eclipse, Frama-C로 총 11개의 오픈소스 도구를 선정하였고, 각 도구별 사용방법에 맞추어 확보한 CVE 발생지점 코드와 패치버전들에 대해 정적 분석을 실시하고자 한다.

* 1. **퍼징**

프로그램에 무작위 데이터를 입력하여 버그 및 취약점을 찾아주는 자동화된 툴 중 하나인 AFL은 현재 Linux netlink, libvorbis, PHP 등과 같이 오늘날 프로그램에서 수많은 취약점을 발견하는데 도움이 되는 오픈 소스 fuzzer이므로 AFL를 퍼징 도구로 선정하였다.

Peach는 데이터 변이를 기반으로 퍼징을 수행하는 도구로써 일부 퍼징 도구들은 입력 데이터 생성 및 주입까지만 다루는 것과 달리 Peach는 다양한 모니터링 기능을 제공하고 있으며, 결함 발생 시 중복결함에 대한 분류와 결함에 대한 자동분석과 같은 기능을 지원하며, 다른 도구들에 비해 더 많은 자동화를 적용하여 퍼징 수행에 따른 결과 분석을 용이하게 하고 있으므로 Peach를 퍼징 도구로 선정하였다.

Peach 퍼저의 경우 퍼징 도구들 중 가장 발전된 형태로 알려져 있는 스마트 fuzzer이며, Mutation 방식과 Generation 방식을 둘다 사용할 수 있다.

LibFuzzer는 화이트박스 테스트를 이용한 퍼저로써 피치 퍼저와 마찬가지로 입력 값을 변경하여 코드 커버리지를 넓히는 스마트 퍼저의 일종이다. LLVM 컴파일러 인프라 프로젝트를 통해 퍼징을 지원하며 함수의 반복 호출로 크래시를 발생시키는 방식으로 작동하게 된다. AFL과는 달리 단일 프로세스 내에서 입력 데이터를 모두 퍼징하는 방식으로 더욱 빠르게 동작하며 퍼징 자체의 시간적인 이득을 볼 수 있다는 이점이 있어 선정하였다.

* 1. **기호 실행**

기존 기호 실행의 한계점인 경로 폭발을 개선하기 위한 시도로서 기호실행과 실제 수행을 결합하여 실행 경로를 분석하는 콘콜릭 실행이 있는데, 이에 대한 대표적인 연구로 KLEE가 있다. KLEE는 심볼릭 프로세스의 운영체제와 인터프리터 역할을 동시에 수행한다. 심볼릭 프로세스는 State라고 표현하는데, 실제 콘콜릭 실행 경로를 따라가며 분기 조건을 저장하는 프로세스이다. 각 State는 프로세스와 동일하게 스택, 힙, 프로그램 카운터(PC) 등의 정보를 갖고 있고, 자신이 수행한 경로 조건 정보를 축적해 두었다가 경로 종료 시 테스트 케이스 도출에 활용한다.[2]

* 1. **취약점 분석 대상 오픈소스 SW**

취약점 분석 도구들의 비교분석이 용이하도록, 취약점 보고 사례(CVE)가 존재하는 오픈소스 펌웨어들로 대상을 한정하였다. 그리고 IoT 기기의 다양한 오픈소스들을 대표하기 위해 리눅스 기반의 application과 baremetal 기반 application을 고루 선정하고, 도구간의 실효성 있는 비교분석 결과를 얻기 위해 IoT 디바이스에서 널리 사용되는 application 위주로 선정하였다. 그리하여 리눅스 기반 application으로는 무선통신에 관여하는 codeaurora 관련 manufacture와 블루투스 통신에 관여하는 Bluez의 최신 및 CVE 발생지점의 과거 버전코드를 선정하였고, baremetal 기반 application으로는 mbed와 contiki, freeRTOS의 최신 release와 CVE발생 지점 과거 코드를 선정하였다. 또한, 이외에도 최종사용자 입장에서 자주 사용하는 application(LibreOffice, VLC, pidgin, clementine 등)에 대해서도 분석할 수 있도록 소스코드를 확보하였다.

1. **취약점 탐지 방법 비교 분석**

bluez의 cve가 발생한 버전의 코드를 대상으로 yasca와 cppcheck으로 정적 분석을 실행한 결과, yasca에서는 327건, cppheck에서는 612건의 취약점을 검출하였다. 검출 내용 면에서 yasca는 use-after-free나 double free, unsafe function 등 대분류적인 검출 결과를 보인 반면, cppcheck에서는 passing null, behavior undefined 등 보다 구체적인 분류를 가지고 검출한 것을 확인할 수 있었다. 특히, 항상 참인 조건이나 연산자간 계산우선순위 명시 필요 등의 검출지점들은 퍼징이나 기호실행을 위한 전체 코드 부분을 한정하는 데에 긍정적인 단서를 제공할 수 있을 것으로 보인다.

peach의 경우 퍼징을 하기 위해 PeachPit이라는 XML파일을 작성해야 하는데, 해당 XML파일은 4가지 영역으로 나뉘어져 있다. DataModel, StateModel, Agent,Test로 나뉘어 있다. DataModel은 퍼징을 위한 데이터 블록 구조를 정의하는 부분으로 대상 인풋값의 파일 형식이라든지, 어떻게 퍼징을 진행할 것인지 데이터의 구조를 정의하는 부분이고 StateModel은 데이터를 보내고 받는 방법을 정의하고 Agent는 경로, Test는 결과값이 나타날 로그파일의 위치 등의 정보를 가지고 있다. 프로그램 퍼징을 진행할 시 인풋 값을 XML에 따라 진행하므로 XML파일을 잘 작성하여야 하는 단점이 있으나 일단 분석을 마치고 퍼징을 진행하면 페이지 폴트가 일어났는지, 일어났다면 어디서 일어났는지에 대한 정보를 알 수 있었다.

CVE에 보고된 취약점을 가진 프로그램을 퍼징하기 전, 먼저 취약점이 있는 간단한 Buffer Overflow가 발생하는 소스코드를 구현하고 적절한 입력 데이터 파일을 작성하여 AFL fuzzer를 실행해보고 충돌에 대한 기본 정보 탐색 과정을 수행하였다.

Out of bound access를 일부러 유발시키는 예제코드를 퍼징 해 보고 그 결과 로그를 분석함으로써 립 퍼저를 이해하고 학습하였다.

KLEE를 사용해서 CVE를 재현해내기 위해 먼저 간단히 n개의 분기로 나눠지는 함수에 대한 경로 탐지를 재현했다. 추후 규모가 큰 프로그램의 분석을 위해서는 분기가 많이 나눠지는 프로그램을 분석해야 하는데, 이를 위해 2차원 공간에 대해 4방향으로 1칸씩 이동하며 목적지까지의 길을 찾는 미로 프로그램에 대해 경로 탐색을 마쳤다.

본 논문에서 제시한 3가지 방법을 상대적으로 비교 분석한 결과 Table 1과 같은 결과를 얻을 수 있었다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tool | Method | Automated | Code Coverage | Speed |
| Peach | Smart Fuzzing | Easy | Low | Fast |
| AFL | Smart Fuzzing | Medium | Low | Medium |
| KLEE | Concolic Execution | Hard | High | Slow |
| YASCA | Static Analyzer | Easy | Low | Medium |
| Cppcheck | Static Analyzer | Easy | Low | Fast |

**[Table 1.] 분석 방법에 따른 분석 도구 비교 결과**

다양한 CVE 발생지점들에 대해서 분석하면 취약점에 따른 각 방법의 성능 차이, 특성 등과 더불어 각 분석 도구 별 탐지 경향 등을 파악할 수 있을 것으로 기대된다.

또한, CVE가 보완된 패치버전들에 대해서도 동일하게 적용하여 패치 전과 패치 후의 탐지 내역 비교를 통해 취약점 보완 방법의 경향이나 효용성 등도 비교해 볼 수 있을 것이다. 이를 토대로 취약점 탐지방법의 발전 방향을 제시할 수 있을 것으로 기대된다.

1. **결론 및 향후 연구**

IoT 디바이스가 빠르게 늘어나면서 보안의 중요성이 대두되고 있다. 특히, 인터넷에 연결되고 다양한 기기와 융합되는 IoT의 특성과, 코드가 쉽게 확산되는 오픈소스의 특성이 맞물려 다양한 형태의 보안 문제를 야기할 수 있기 때문에 취약점들에 대한 연구가 중요하다. IoT 디바이스에 자주 사용되는 오픈소스 펌웨어들을 선정하였고, CVE에 보고된 취약점을 가진 과거의 코드들과 패치된 버전의 코드들을 확보하였다. 다양한 분석 방법과 분석 도구들을 이용하여 해당 코드들을 분석하여 분석 방법들 간의 탐지 성능 및 특성 비교, 분석 도구 간의 비교, 그리고 취약점별 특성 비교 및 보완 양상 등을 파악하고자 한다. 얻어진 결과로 취약점 탐지 방법들의 발전 방향, 탐지 도구들의 발전 가능성 등에 대해 살펴보고, 나아가 취약점별 효과적인 보완 패턴 등 다양한 가능성에 대해 제시해보고자 한다.

1. **참고 문헌**

[1] Source Code Security Analyzers[웹사이트].(2019. 4.21). URL:

<https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html>

[2] 오상환, “SW 보안 취약점 자동 탐색 및 대응 기술 분석,” 한국산학기술학회논문지 제18권 제11호, 2017