

RTOS 환경 취약점 분석 및 경량화

부산대학교 정보컴퓨터공학부

지도 교수 : 손준영

학생 : 202155654 임준식

202155656 정혁준

202155644 김성문

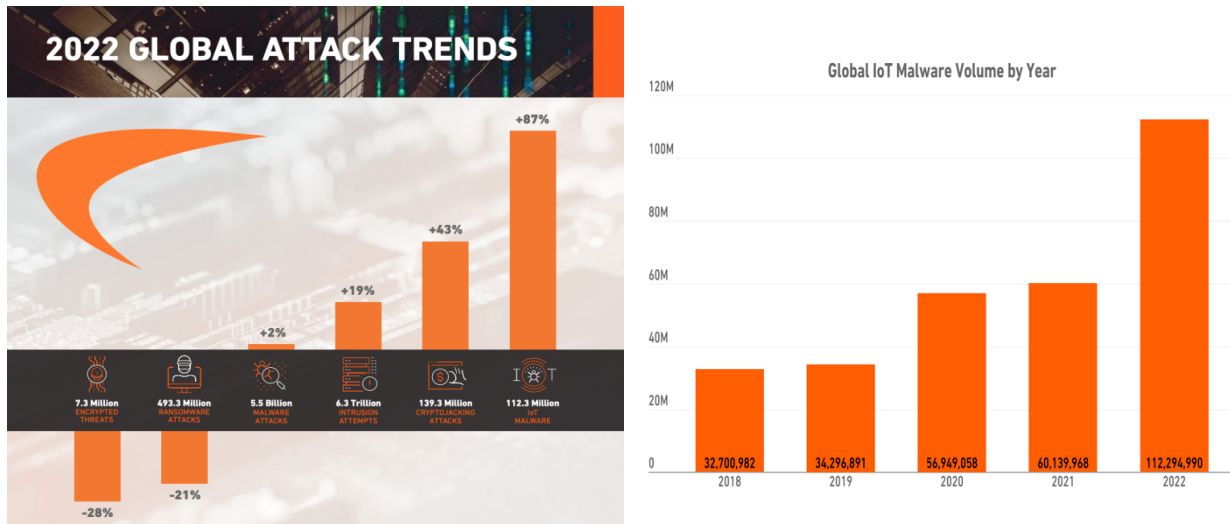
개요

1. 과제 배경 및 목표-----	3
1.1 과제 배경	
1.2 IAR	
1.3 QNX	
1.4 RISC-V	
1.5 목표	
2. 진행방안-----	7
2.1 취약점 분석 방안	
2.2 최적화 방안	
3. 현실적 제약 사항 분석 결과 및 대책-----	12
4. 추진 체계 및 일정-----	13
5. 구성원 역할 분담-----	13

1. 과제 배경 및 목표

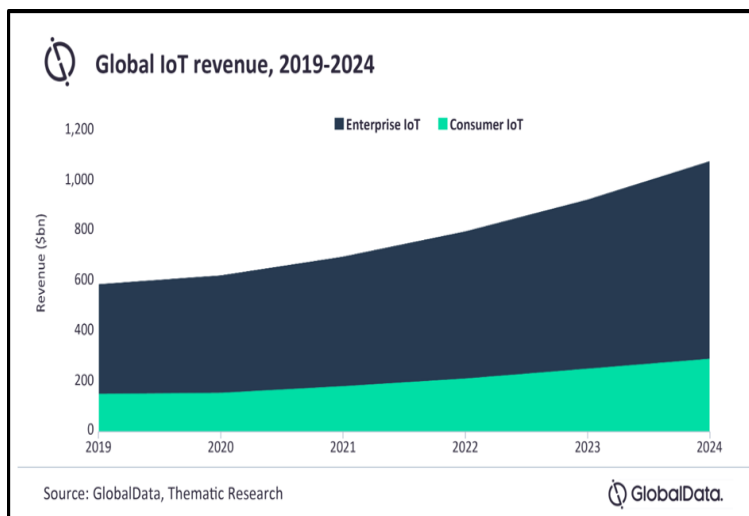
1.1 과제 배경

IT기술이 발전하면서 AI, IOT, 임베디드 등 다양한 IT관련 산업이 발전하고 있다. 그에 맞춰서 다양한 TOOL들이 개발되고 있고 기술이 발전하는 속도는 빠른 속도로 상승하고 있다. 정말 좋은 소식이지만 보안과 관련된 문제들도 지속적으로 발생하고 있다.

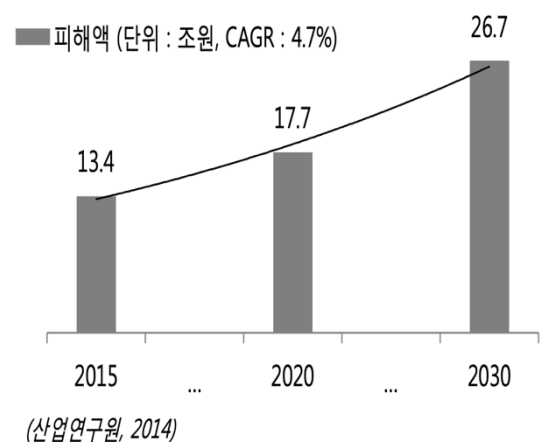


출처 : Techopedia

특히 2022년도에는 크립토재킹, IOT멀웨어와 관련된 공격이 각각 43%, 87%씩 증가한걸 볼 수 있다. 물론 랜섬웨어 및 다양한 보안과 관련된 공격으로 인해서 여러가지 문제가 발생하고 있지만 이러한 수치로 인해서 2018년 이후 처음으로 전체적인 멀웨어 트렌드에 전략적 변화를 가지고 왔다고 이야기 하고 있다.



출처 : GlobalData2021년6월2일



출처 : 산업연구원, 2014

위의 표에서 볼 수 있듯이 앞으로도 IOT시장은 계속 커질 것으로 전망하고 있으며 임베디드 개발도 함께 많은 관심을 받게 될 것으로 예상하고 있다.

이에 따라서 보안과 관련한 문제도 다양한 방식으로 더 많이 발생할 것이다.

임베디드 및 IOT보안에 대한 문제로 발생하는 피해는 여러가지 형태로 발생한다.

1. 자동차 또한 수많은 임베디드 시스템으로 연결되어 있는데 자동차 문을 잠그거나, 시동을 켜고 끌 수 있는 문제가 있다. 심각하면 속도를 조정해서 큰 인명피해를 불러올 수 있다.
실제로 독일의 데이비드 콜롬보라는 해커는 미국 테슬라의 전기차 25대를 원격으로 해킹한 사례가 있다.
2. 요즘엔 스마트 홈이 많이 생기고 있는데 여기서 흔히 사용하는 월패드를 해킹해서 실내용 카메라를 이용해 집안을 보거나, 난방, 전등 등을 임의로 조작해서 문제를 발생시킬 수 있게 된다. 실제로 2022년에 아파트의 월패드를 해킹해서 집안 내부를 촬영하고 사진과 영상을 판매하려 한 일도 있었다.

이처럼 기존에 존재하는 사물들이 편리해짐에 따라서 점점 데이터화 되는 게 많이 보이기 시작하고 관련된 문제는 계속 증가하고 있다. 본 졸업과제 팀은 여러 정보보안 문제들 중 임베디드 시스템 및 IOT와 관련된 문제를 집중적으로 다루고 보안 알고리즘의 최적화를 통해서 보안성을 높여주고 알고리즘의 효율성을 증가시킬 수 있는 방안을 탐색하고자 한다.

현재 임베디드 시스템은 하나의 공간에 수많은 정보를 저장하고 활용해야 하기 때문에 경량화를 통해서 크기 대비 최대의 효율성을 가지고자 많은 사람들이 경량화에 힘을 쓰고 있다. 특히 AI가 발전됨에 따라서 로봇, 냉장고, 스마트 홈, 스마트공장 등 다양한 곳에서 영향을 주고 있는데 냉장고에서 오늘의 메뉴를 추천해주거나, 로봇이 사람과 소통을 해서 문제를 해결하거나, 공장의 시스템에서 가장 효율적인 방법을 이용해서 시스템을 가동 시키는 등 여러 데이터를 기반으로 시스템이 작동하는 것을 알 수 있다. 이렇게 하기 위해서는 다양한 데이터를 가지고 있어야 하고, 여러가지 AI알고리즘, 정보보안 알고리즘이 한정된 공간에 들어가야 한다. 그래서 경량화에 초점을 두는 기업이 많아지고 있는 추세다. 반도체의 경우에도 지속적인 경량화를 통해서 기술력을 상승시키고 있다. 이처럼 경량화는 현대 시대의 중요한 부분으로 자리잡고 있다. 그래서 알고리즘 최적화를 통해 더 작은 용량에도 효율적으로 작동할 수 있게 만들 것이다. 최적화를 함으로써 얻을 수 있는 이점은

1. 에너지 효율성 개선
2. 비용 절감
3. 성능 향상
4. 보안 강화

가 존재한다. 본 팀은 알고리즘의 깊은 이해를 하여 어셈블리어를 이용한 최적화를 통해서 더 좋은 성능을 갖춘 알고리즘을 개발하는 것을 목표로 졸업과제를 진행해보고자 한다.

여기에 더해서 보안에 대한 배경을 바탕으로 RTOS환경의 취약점을 분석할 것이다.

경량화를 통해서 알고리즘의 효율성을 높여주는 것도 중요하지만 기존에 사용하는 운영체제에 대한 취약점을 방지하는 것 또한 중요한 요소이다. 과거 취약점의 사례로 존재하는 것은

1. FressRTOS에서 2018년도 원격코드 실행과 정보 유출, 디도스 공격을 가능하게 하는 13가지 취약점을 발견하였다. 그중 CVE-2018-16522는 SOCKETS_SetSockOpt함수에서 발생하는 초기화 되지 않은 포인터 해제 문제가 있고, 이 취약점을 이용해 원격 코드 공격이 가능한 치명적 취약점이 발견되었다.
2. QNX에서 2021년 디도스 및 원격 코드 실행 공격을 할 수 있는 취약점을 발견하였다. CVE-2021-22156에 대한 보안 권고문이 발표되었는데 C 런타임 라이브러리의 calloc() 함수의 정수 오버플로우 취약점이었고, 공격자가 잠재적으로 서비스 거부를 수행하거나 임의 코드를 실행할 수 있는 문제가 발생할 수 있었다.

기존의 범용적으로 사용되고 있는 운영체제들도 시간이 흐르면서 다양한 취약점이 발견되고 있다. 우리 팀은 기존의 취약점을 분석하는 것뿐만 아니라 나아가서 RTOS에 존재하는 취약점을 발견 및 분석하여 결과를 도출하는 걸 목표로 졸업과제를 진행할 것이다.

1.2 IAR



임베디드 시스템 개발을 위한 통합 개발 환경(IDE)다. 흔히 IAR Embedded Workbench로 알려져 있으며 다양한 마이크로 아키텍처를 제공하는 개발 툴이다.

<pre>main.c x main() 1 #include <stdio.h> 2 3 extern void fool(); 4 5 int d2 @ 0x80000000; 6 7 int main(void) { 8 int t; 9 t=f001(); 10 printf("d2=%d, ret=%d\n", d2,t); 11 t=f001(); 12 printf("d2=%d, ret=%d\n", d2,t); 13 return 0; 14 } 15</pre>	<pre>test.S x 1 2 public f001 3 4 SECTION `.text':CODE:NOROOT(2) 5 f001: 6 lui t0, 0x800000; 7 addi t0,t0, 0x000; 8 addi t1, zero, 0x001; 9 10 amoadd.w a0,t1,(t0) 11 ret 12 13 14 END 15</pre>
--	--

IAR툴을 이용해서 C언어로 작성된 코드를 RISC-V5 어셈블리어로 컴파일 하여서 직접 어셈블리어에 접근하여서 보안 알고리즘을 최적화할 것이다.

1.3 QNX



임베디드 시스템에서 사용되는 실시간 운영체제(RTOS, Real-Time-Operating-System)으로 실시간 처리 능력, 안전성 및 보안성, 다양한 하드웨어를 지원하고 있고, 우리나라의 대표적인 사용처는 월성 원자력 발전소이다.

1.4 RISC-V

2010년부터 미국의 UC버클리에서 개발중인 무료 오픈 소스 RISC 명령어셋 아키텍처이다.

```
float f2c (float fahr) {  
    return ((5.0/9.0)*(fahr - 32.0));  
}
```

- fahr in f10, result in f10, literals in global memory space

■ Compiled RISC-V code:

```
f2c:  
    flw    f0,const5(x3) // f0 = 5.0f  
    flw    f1,const9(x3) // f1 = 9.0f  
    fdiv.s f0, f0, f1 // f0 = 5.0f / 9.0f  
    flw    f1,const32(x3) // f1 = 32.0f  
    fsub.s f10,f10,f1 // f10 = fahr - 32.0  
    fmul.s f10,f0,f10 // f10 = (5.0f/9.0f) * (fahr-32.0f)  
    jalr   x0,0(x1) // return
```

c언어 RISC-V code로 컴파일 예시

이렇게 보안 알고리즘을 RISC-V코드로 컴파일 하여서 최적화를 시키는 방법을 찾을 것이다.

1.5 목표

1. IAR workbench를 통해서 여러가지 언어로 작성된 보안 알고리즘을 RISC-V 코드로 컴파일 하여서 최적화할 수 있는 방법을 찾을 것이다.
2. QNX를 이용해서 기존에 존재하는 보안 취약점을 분석하여서 어떤 방식으로 RTOS의 취약점을 분석하고 이로 인해서 발생하는 문제점이 무엇이 있는지 찾을 것이다. 원래 없던 취약점을 발견하고 발생할 수 있는 문제가 무엇이 있는지 찾을 것이다.

두개의 목표를 모두 성취하고 가능하다면 논문까지 작성해보는 것을 최종적인 목표로 졸업과제를 진행할 것입니다.

2.1 취약점 분석 방안

1. 기존 취약점 분석:

CVE(Common Vulnerabilities and Exposures) Records를 통해 알려진 취약점들을 분석하고, 해당 취약점이 어떤 공격에 사용될 수 있는지 알아볼 것이다. 또, Google Scholar와 GitHub를 보조적으로 활용해 QNX의 취약점에 관련된 코드나 논문 또는 PoC(Proof of Concept)등을 찾아 분석해 볼 것이다.



HOME > CVE > SEARCH RESULTS

Search Results

There are 54 CVE Records that match your search.

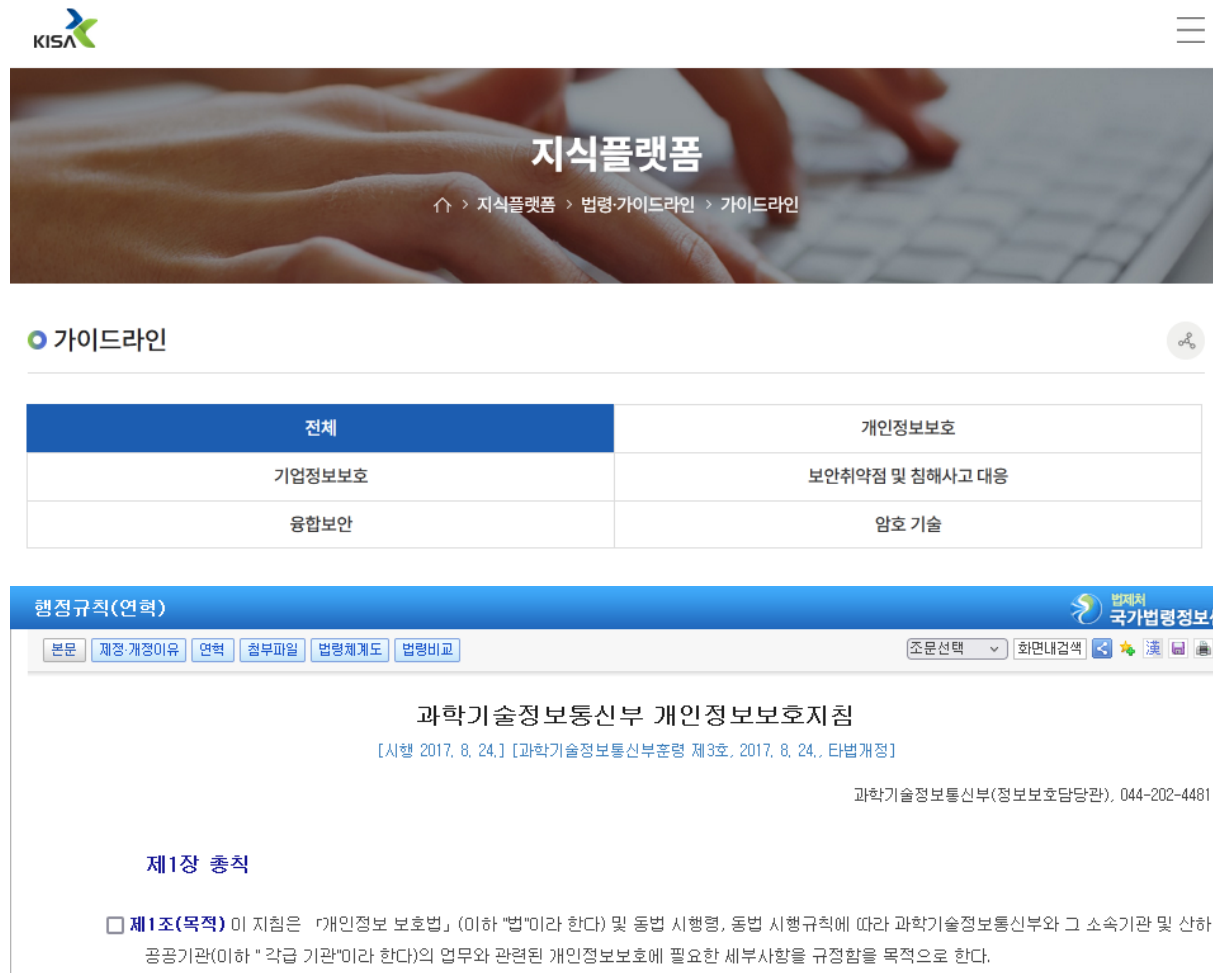
Name	Description
CVE-2024-1309	Uncontrolled Resource Consumption vulnerability in Honeywell Niagara Framework on Windows, Linux, QNX allows Content Spoofing. This issue affects Niagara Framework: before Niagara AX 3.8.1, before Niagara 4.1.
CVE-2023-32701	Improper Input Validation in the Networking Stack of QNX SDP version(s) 6.6, 7.0, and 7.1 could allow an attacker to potentially cause Information Disclosure or a Denial-of-Service condition.
CVE-2022-36043	Rizin is a UNIX-like reverse engineering framework and command-line toolset. Versions 0.4.0 and prior are vulnerable to a double free in bobj.c:rz_bin_reloc_storage_free() when freeing relocations generated from qnx binary plugin. A user opening a malicious qnx binary could be affected by this vulnerability, allowing an attacker to execute code on the user's machine. Commit number a3d50c1ea185f3f642f2d8180715f82d98840784 contains a patch for this issue.
CVE-2021-32025	An elevation of privilege vulnerability in the QNX Neutrino Kernel of affected versions of QNX Software Development Platform version(s) 6.4.0 to 7.0, QNX Momentics all 6.3.x versions, QNX OS for Safety versions 1.0.0 to 1.0.2, QNX OS for Safety versions 2.0.0 to 2.0.1, QNX for Medical versions 1.0.0 to 1.1.1, and QNX OS for Medical version 2.0.0 could allow an attacker to potentially access data, modify behavior, or permanently crash the system.
CVE-2021-32024	A remote code execution vulnerability in the BMP image codec of BlackBerry QNX SDP version(s) 6.4 to 7.1 could allow an attacker to potentially execute code in the context of the affected process.
CVE-2021-22156	An integer overflow vulnerability in the calloc() function of the C runtime library of affected versions of BlackBerry® QNX Software Development Platform (SDP) version(s) 6.5.0SP1 and earlier, QNX OS for Medical 1.1 and earlier, and QNX OS for Safety 1.0.1 and earlier that could allow an attacker to potentially perform a denial of service or execute arbitrary code.
CVE-2020-6932	An information disclosure and remote code execution vulnerability in the slinger web server of the BlackBerry QNX Software Development Platform versions 6.4.0 to 6.6.0 could allow an attacker to potentially read arbitrary files and run arbitrary executables in the context of the web server.
CVE-2019-8998	An information disclosure vulnerability leading to a potential local escalation of privilege in the procs service (the /proc filesystem) of BlackBerry QNX Software Development Platform version(s) 6.5.0 SP1 and earlier could allow an attacker to potentially gain unauthorized access to a chosen process address space.
CVE-2018-20785	Secure boot bypass and memory extraction can be achieved on Neato Botvac Connected 2.2.0 devices. During startup, the AM335x secure boot feature decrypts and executes firmware. Secure boot can be bypassed by starting with certain commands to the USB serial port. Although a power cycle occurs, this does not completely reset the chip; memory contents are still in place. Also, it restarts into a boot menu that enables XMODEM upload and execution of an unsigned QNX IFS system image, thereby completing the bypass of secure boot. Moreover, the attacker can craft custom IFS data and write it to unused memory to extract all memory contents that had previously been present. This includes the original firmware and sensitive information such as Wi-Fi credentials.
CVE-2017-9371	In BlackBerry QNX Software Development Platform (SDP) 6.6.0 and 6.5.0 SP1 and earlier, a loss of integrity vulnerability in the default configuration of the QNX SDP could allow an attacker being able to reduce the entropy of the PRNG, making other blended attacks more practical by gaining control over environmental factors that influence seed generation.
CVE-2017-9369	In BlackBerry QNX Software Development Platform (SDP) 6.6.0 and 6.5.0 SP1 and earlier, an information disclosure vulnerability in the default configuration of the QNX SDP could allow an attacker to gain information relating to memory layout of higher privileged processes by manipulating environment variables that influence the loader.
CVE-2017-3893	In BlackBerry QNX Software Development Platform (SDP) 6.6.0, the default configuration of the QNX SDP system did not in all circumstances prevent attackers from modifying the GOT or PLT tables with buffer overflow attacks.
CVE-2017-3892	In BlackBerry QNX Software Development Platform (SDP) 6.6.0, an information disclosure vulnerability in the default configuration of the QNX SDP could allow an attacker to gain information relating to memory layout that could be used in a blended attack by executing commands targeting procs resources.
CVE-2017-3891	In BlackBerry QNX Software Development Platform (SDP) 6.6.0, an elevation of privilege vulnerability in the default configuration of the QNX SDP with QNet enabled on networks comprising two or more QNet nodes could allow an attacker to access local and remote files or take ownership of files on other QNX nodes regardless of permissions by executing commands targeting arbitrary nodes from a secondary QNX 6.6.0 QNet node.

```
Code Blame Executable File · 65 lines (60 loc) · 1.85 KB

1  #!/bin/sh
2  #
3  #          QNX 6.4.x/6.5.x ifwatchd local root exploit by cenobyte 2013
4  #          <vincitamorpatriae@gmail.com>
5  #
6  # - vulnerability description:
7  # Setuid root ifwatchd watches for addresses added to or deleted from network
8  # interfaces and calls up/down scripts for them. Any user can launch ifwatchd
9  # and provide arbitrary up/down scripts. Unfortunately ifwatchd does not drop
10 # privileges when executing user supplied scripts.
11 #
12 # - vulnerable platforms:
13 # QNX 6.5.0SP1
14 # QNX 6.5.0
15 # QNX 6.4.1
```

2. KISA 및 과학기술정보통신부의 정보보호 가이드라인 활용:

KISA(Korea Internet & Security Agency)에서 제공하는 '보안취약점 및 침해사고 대응' 가이드라인과 과학기술정보통신부(Ministry of Science and ICT)에서 제공하는 '과학기술정보통신부 개인 정보보호지침'을 활용하여, QNX가 여러 보안 요구사항을 충족시키고 있는지 검토해볼 것이다.



The screenshot shows the KISA Knowledge Platform interface. At the top, there's a header with the KISA logo and a navigation menu. Below the header, a large banner reads '지식플랫폼' (Knowledge Platform) with a sub-navigation path: '홈 > 지식플랫폼 > 법령·가이드라인 > 가이드라인'. A sidebar on the left lists '가이드라인' (Guidelines) and '법령' (Laws). The main content area displays a table of guidelines:

전체	개인정보보호
기업정보보호	보안취약점 및 침해사고 대응
융합보안	암호 기술

Below the table, there's a section for '과학기술정보통신부 개인정보보호지침' (Ministry of Science and ICT Personal Information Protection Guidelines). It includes a search bar, a list of documents, and a detailed view of the '제1장 총칙' (Chapter 1 General Provisions). The document title is '과학기술정보통신부 개인정보보호지침' (Ministry of Science and ICT Personal Information Protection Guidelines), dated 2017. 8. 24., and amended on 2017. 8. 24. The document number is 044-202-4481.

제1장 총칙

☐ **제1조(목적)** 이 지침은 「개인정보 보호법」(이하 "법"이라 한다) 및 동법 시행령, 동법 시행규칙에 따라 과학기술정보통신부와 그 소속기관 및 산하 공공기관(이하 "각급 기관"이라 한다)의 업무와 관련된 개인정보보호에 필요한 세부사항을 규정함을 목적으로 한다.

3. MITRE ATT&CK 분석 및 적용:

MITRE ATT&CK Matrix의 공격 방식들을 분석해 정리하고, 기존 QNX 취약점들이 MITRE ATT&CK Matrix의 어떤 기법에 해당되는지(Initial Access, Execution, Persistence, Privilege Escalation 등등) 매칭시켜 볼 것이다. 또한 QNX의 새로운 취약점을 찾아 볼 것이다.

Initial Access	Execution	Persistence	Privilege Escalation
10 techniques	14 techniques	20 techniques	14 techniques
Content Injection	Cloud Administration Command	Account Manipulation (6)	Abuse Elevation Control Mechanism (6)
Drive-by Compromise	Command and Scripting Interpreter (10)	BITS Jobs	Access Token Manipulation (5)
Exploit Public-Facing Application	Container	Boot or Logon Autostart Execution (14)	Account Manipulation (6)

2.2 최적화 방안

1. 코드 크기 최적화

IAR 컴파일러는 사용되지 않는 함수나 변수를 제거하여 코드의 바이너리 크기를 줄이는 최적화 기술을 사용합니다.

예를 들어, 프로그램에서 절대 호출되지 않는 함수가 있는 경우 컴파일러는 해당 함수를 바이너리에서 제거하여 코드 크기를 줄일 수 있습니다.

2. 속도 최적화

IAR 컴파일러는 코드 생성 프로세스를 최적화하여 프로그램 실행 속도를 높이는 기술을 사용합니다.

예를 들어, 컴파일러는 루프, 인라인 함수를 풀거나 더 빠르게 실행할 수 있는 프로세서별 명령어를 사용할 수 있습니다.

3. 전력 최적화

IAR 컴파일러는 프로그램의 전력 소비를 줄일 수 있는 전력 최적화 기술을 제공합니다.

예를 들어, 컴파일러는 프로세서의 저전력 모드를 사용하거나, 프로세서가 실행되는 빈도를 줄이거나, 사용하지 않는 주변 장치를 비활성화하도록 코드를 최적화할 수 있습니다.

4. 컴파일러 옵션

IAR 컴파일러는 개발자가 특정 요구 사항에 맞게 최적화 프로세스를 미세 조정할 수 있는 다양한 컴파일러 옵션을 제공합니다.

예를 들어, 개발자는 다양한 최적화 수준 중에서 선택하거나, 특정 최적화를 활성화 또는 비활성화하거나, 코드 생성 방법에 영향을 줄 수 있는 컴파일러 플래그를 설정할 수 있습니다.

다양한 방법을 사용해서 최적화를 수행할 것이다.

최적화 방법 예시

1. 함수 인라인

함수 호출을 호출 사이트에서 함수의 실제 코드로 바꾸는 작업

특히 작은 기능이 포함될 때 성능이 크게 향상될 수 있지만, 코드 크기가 늘어날 수도 있습니다.

```
int mul(int num1,int num2)
{
    return num1*num2;
}

int main(void)
{
    int result = 0;
    result = mul(3,3);
}
```

```
mul:
    push    rbp
    mov     rbp, rsp
    mov     DWORD PTR [rbp-4], edi
    mov     DWORD PTR [rbp-8], esi
    mov     eax, DWORD PTR [rbp-4]
    imul    eax, DWORD PTR [rbp-8]
    pop     rbp
    ret

main:
    push    rbp
    mov     rbp, rsp
    sub     rsp, 16
    mov     DWORD PTR [rbp-4], 0
    mov     esi, 3
    mov     edi, 3
    call    mul
    mov     DWORD PTR [rbp-4], eax
    mov     eax, 0
    leave
    ret
```

이렇게 선언하는 것도 방법이지만.

```
int main(void)
{
    int result = 0;
    result = 3*3;
}
```

```
main:
    push    rbp
    mov     rbp, rsp
    mov     DWORD PTR [rbp-4], 0
    mov     DWORD PTR [rbp-4], 9
    mov     eax, 0
    pop     rbp
    ret
```

간단한 코드라면 main문 안에다가 선언하는 게 더 효율적일 수 있다.

하지만 위에서 언급한 것처럼 main문안에 포함되는 코드의 양이 늘어난다는 단점이 있다.

2. 데드 코드 제거

실행되지 않은 코드를 제거하는 작업

실행 코드의 크기를 줄이고 실행해야 하는 명령어 수를 줄여 코드 성능을 향상시킬 수 있습니다.

```
int square(int num) {
    int result = 0;
    for(int i =0;i<100;i++)
    {
    }
    for(int i =0;i<100;i++)
    {
        result++;
    }
}
```

```
square:
    push    rbp
    mov     rbp, rsp
    mov     DWORD PTR [rbp-20], edi
    mov     DWORD PTR [rbp-4], 0
    mov     DWORD PTR [rbp-8], 0
    jmp     .L2
.L3:
    add     DWORD PTR [rbp-8], 1
.L2:
    cmp     DWORD PTR [rbp-8], 99
    jle     .L3
    mov     DWORD PTR [rbp-12], 0
    jmp     .L4
.L5:
    add     DWORD PTR [rbp-4], 1
    add     DWORD PTR [rbp-12], 1
.L4:
    cmp     DWORD PTR [rbp-12], 99
    jle     .L5
    nop
    pop     rbp
    ret
```

여기서 처음에 나온 for문은 실행하지 않아도 상관없기 때문에 제거해서 효율성을 증가시킨다.

그럼 필요 없는 분기를 줄여줄 수 있다. -> 효율성 증가

3. 루프 풀기

루프 본문을 여러 번 복제하여 루프를 완료하는 데 필요한 반복 횟수를 줄이는 작업
루프 관리 명령의 오버헤드를 줄여 코드 성능을 향상시킬 수 있습니다.

```
int square(int num) {
    int arr[3];
    for(int i =0;i<3;i++)
    {
        arr[i] = i;
    }
    // arr[0] = 0;
    // arr[1] = 1;
    // arr[2] = 2;
}
```

```
square:
    push    rbp
    mov     rbp, rsp
    mov     DWORD PTR [rbp-20], edi
    mov     DWORD PTR [rbp-4], 0
    jmp     .L2
.L3:
    mov     eax, DWORD PTR [rbp-4]
    cdqe
    mov     edx, DWORD PTR [rbp-4]
    mov     DWORD PTR [rbp-16+rax*4], edx
    add     DWORD PTR [rbp-4], 1
.L2:
    cmp     DWORD PTR [rbp-4], 2
    jle     .L3
    nop
    pop     rbp
    ret
```

이렇게 선언된 구문은 어셈블리어로 컴파일하면 오른쪽의 그림과 같아진다 저 구문에서 L2구문이 3번은 더 반복될 것이다.

```
int square(int num) {
    int arr[3];
    // for(int i =0;i<3;i++)
    // {
    //     arr[i] = i;
    // }
    arr[0] = 0;
    arr[1] = 1;
    arr[2] = 2;
}
```

```

    push    rbp
    mov     rbp, rsp
    mov     DWORD PTR [rbp-20], edi
    mov     DWORD PTR [rbp-12], 0
    mov     DWORD PTR [rbp-8], 1
    mov     DWORD PTR [rbp-4], 2
    nop
    pop     rbp
    ret
```

이렇게 바꿔준다면 어셈블리어상으로 작업 수를 훨씬 줄일 수 있다. 이처럼 루프가 효율적일때도 있지만 비효율적인 부분을 판단하여 문법을 바꿔줘서 코드를 최적화가 가능하다.

4. 레지스터 할당

메모리 위치 대신 프로세서 레지스터에 변수를 할당하는 작업
변수를 읽고 쓰는 데 필요한 메모리 액세스 횟수를 줄여 코드 성능을 향상시킬 수 있습니다.

5. 코드 모션

가능한 경우 루프 또는 조건문 외부로 코드를 이동하는 작업
동일한 코드를 실행해야 하는 횟수를 줄여 코드 성능을 향상시킬 수 있습니다.

```

;for(i=0; i<n; i++){
0x800000d0: 0x4827    LDR.N    R0, ??DataTable0_5    ; i
0x800000d2: 0x2100    MOVNS   R1, #0
0x800000d4: 0x0001    STR     R1, [R0]
; for(i=0; i<n; i++){
??main_2:
0x800000d6: 0x4826    LDR.N    R0, ??DataTable0_5    ; i
0x800000d8: 0x6800    LDR     R0, [R0]
0x800000da: 0x4929    LDR.N    R1, ??DataTable0_9    ; n
0x800000dc: 0x6009    LDR     R1, [R1]
0x800000de: 0x4929    CMP     R0, R1
0x800000e0: 0x4d14    BGE.N    ??main_3              ; 0x800000e0
; ab[i] = 7 * i + x * x;
0x800000e2: 0x4823    LDR.N    R0, ??DataTable0_5    ; i
0x800000e4: 0x6800    LDR     R0, [R0]
0x800000e6: 0x2107    MOVNS   R1, #7
0x800000e8: 0x4a24    LDR.N    R2, ??DataTable0_8    ; x
0x800000ea: 0x4912    LDR     R2, [R2]
0x800000ec: 0x4b23    LDR.N    R3, ??DataTable0_8    ; x
0x800000ee: 0x681b    LDR     R3, [R3]
0x800000f0: 0x435a    MULS    R2, R3, R2
0x800000f2: 0x2b01    MLA     R0, R1, R0, R2
0x800000f4: 0x491e    LDR.N    R1, ??DataTable0_5    ; i
0x800000f6: 0x6009    LDR     R1, [R1]
0x800000f8: 0x4a1e    LDR.N    R2, ??DataTable0_6    ; ab
0x800000fa: 0x2942    STR.W    R0, [R2, R1, LSL #2]
...
0x800000fa: 0x7e4     B.N     ??main_2              ; 0x800000e6
```

```
for(i=0; i<n; i++){
    ab[i] = 7 * i + x * x;
}
```

```

; for(i=0; i<n; i++){
0x2: 0x4826    LDR.N    R0, ??DataTable0_9    ; i
0x4: 0x2100    MOVNS   R1, #0
0x6: 0x6001    STR     R1, [R0]
0x8: 0x4820    LDR.N    R0, ??DataTable0_5    ; ab
0xa: 0x4922    LDR.N    R1, ??DataTable0_7    ; x
0xc: 0x6009    LDR     R1, [R1]
0xe: 0x460a    MOV     R2, R1
0x10: 0x4351    MULS    R1, R2, R1
0x12: 0x4b22    LDR.N    R3, ??DataTable0_9    ; i
; ab[i] = 7 * i + x * x;
??main_1:
0x14: 0x2f40    STR.W    R1, [R0], #0x4
; for(i=0; i<n; i++){
0x18: 0x481a    LDR     R2, [R3]
0x1a: 0x4c52    ADDS    R2, R2, #1
0x1c: 0x601a    STR     R2, [R3]
0x1e: 0x4dc9    ADDS    R1, R1, #7
; for(i=0; i<n; i++){
0x20: 0x481a    LDR     R2, [R3]
0x22: 0x2a05    CMP     R2, #5
0x24: 0x4bf6    BLT.N    ??main_1              ; 0xb4
```

```
t = x * x;
for(i=0; i<n; i++){
    ab[i] = 7 * i + t;
}
```

보이는 것처럼 같은 성능을 가지고 있지만 코드를 바꿔서 코드의 행 자체는 줄어들지 않은 것처럼 보일지 몰라도 어셈블리어상으로는 많은 수가 줄어든 걸 볼 수 있다. 이처럼 High level 언어의 코드를 최적화도 하겠지만 IAR 툴을 이용해 어셈블리어로 컴파일을 하고 어셈블리어 자체를 조작하여 코드를 최적화 하는 작업을 할 것이다.

3. 현실적 제약 사항 분석 결과 및 대책

조원들이 다들 보안과 임베디드에 관심이 있어서 졸업과제로 선택하였지만 실제로 보안과 관련해서 깊이 있게 공부를 해본 적은 없기 때문에 진행하는 과정에서 많은 난관이 있을 것으로 예상하고 있다. 또한 QNX 및 IAR은 라이선스를 받아놓은 상태지만 이전에 이 툴을 이용해서 작업을 해본 경험이 없기 때문에 툴을 다루는데 걸리는 시간도 있을 것으로 생각하고 있다.

하지만, 공부의 경우 교수님과 상담을 통해서 모르는 부분은 질문할 수 있도록 하고, 개개인이 각자 다른 분야를 맡아서 공부를 진행 후 스터디를 통해서 알려주는 방식으로 학습을 하여서 학습의 효율성을 높일 것이다.

툴에 대한 이해도는 졸업과제를 진행하면서 학습할 것이다..

라이선스는 유료 구매를 했어야 했지만 교육용 라이선스를 이용해서 해결했다. 어셈블리어의 경우 팀원 전체가 현재 관련된 수업을 듣고 있기 때문에 기본적인 이해도가 있다.

현재 툴의 학습 및 보안에 대한 전반적인 이해를 하고 대상 알고리즘을 선정해 졸업과제를 진행하고 취약점 분석은 QNX를 이용해서 진행할 예정입니다.

4. 추진 체계 및 일정

월	계획
5월	- 앞으로 사용할 툴과 최적화, 취약점 분석 방법에 대해서 논의 및 라이선스 습득
6월	- 취약점 분석 방법 및 취약점 분석 학습(스터디 진행) - 보안 알고리즘 선정
7월	- 취약점 분석 방법 및 취약점 분석 학습(스터디 진행) - 코드 최적화 방안 적용 및 분석 진행 - RTOS취약점 분석
8월	- 코드 최적화 방안 적용 및 분석 진행 - RTOS취약점 분석 - 중간보고서 및 중간 평가표 제출
9월	- 현재까지 나온 내용을 바탕으로 논문 작성
10월	- 논문 보완 및 최종본 완성 - 최종보고서 및 최종평가표 제출
11월	- 졸업과제 발표심사 진행 - 결과물 업로드

4. 구성원 역할 분담

조원	역할
임준식	- 보안 알고리즘 분석 - RTOS 취약점 사례 분석 및 발견 - 어셈블리어 분석 방법에 대한 스터디 준비 - 최적화 방안 적용 - 어셈블리어 분석
정혁준	- 보안 알고리즘 분석 - RTOS 취약점 사례 분석 및 발견 - 최적화 방안 학습 및 스터디 준비 - 최적화 방안 적용 - 어셈블리어 분석
김성문	- 보안 알고리즘 분석 - RTOS 취약점 사례 분석 및 발견 - 취약점 분석 방안 학습 및 스터디 준비 - 최적화 방안 적용 - 어셈블리어 분석