Study of Analyzing and Mitigating Vulnerabilities in uC/OS Real-Time Operating System

Myeonggeon Lee, Gwangjun Choi, Junsang Park, and Seong-je Cho
Department of Computer Science & Engineering
Dankook University
Gyeonggi-do, Rep. of KOREA 16890
{mglee, gjchoi, jsp0916, sjcho}@dankook.ac.kr

***Abstract*—** Programmable Logic Controllers (PLCs) have been widely used in real-time and embedded control applications including safety-critical control systems. Due to their ubiquity and network connectivity, PLCs are prone to various security attacks. Buffer overflow attacks, which target software vulnerabilities in operating system (OS) and application software, are the most common security attacks because of their relatively easy exploitation. Therefore, it is important to have knowledge about software vulnerabilities in OSs for PLCs in order to prevent or mitigate them in PLC design and implementation. Many PLCs use Micrium uC/OS as their OS. In this paper, we present an approach to analyzing and mitigating some software two vulnerabilities, buffer overflows and integer overflows in uC/OS. We first check if there are vulnerable functions in uC/OS system. We then propose a technique to prevent or mitigate the vulnerabilities associated with the functions.

Keywords—uC/OS; Vulnerability Mitigation; Buffer Overflow; Integer Overflow; PLC;

I. INTRODUCTION

Micrium's uC/OS is a small micro-kernel real-time operating system (RTOS) [1]. It is a priority-based multitasking operating system (OS) for microprocessors, written mainly in C programming language. Currently, uC/OS are used as an embedded OS for Programmable Logic Controllers (PLCs) or industry systems [2,3,4]. PLCs are computer-based, single processor devices which have been adapted for the control manufacturing processes [5]. PLCs are usually a main part of automatic systems in industry. Most of the control elements used to execute the logic of the industrial system has been replaced by the PLCs. Input/output devices being controlled are connected to the PLC and then the controller monitors the inputs and outputs according to the process or machine. Due to the development of IoT technology, PLCs have been widely used for many applications such as motor control, pumping systems, energy research, and system monitoring. Nowadays, some PLCs including Siemens S7 series have been to be connected to the Internet and accessed remotely. Since PLCs play an important role in the automation architectures of critical infrastructures including industrial control systems (ICS) and are more vulnerable than general PCs, they will gradually become targets of cyberattacks [6, 7].

Code injection and code reuse attacks are representative cyberattacks [8]. For example, a buffer overflow vulnerability can be exploited to write data beyond the boundaries of the buffer. As a result, an attacker can overwrite critical information and thereby trigger malicious task actions such as injection a malware, or leaking sensitive data. Code injection attacks require the injection of some malicious executable code into the address space of the task. In contrast, code-reuse attacks only leverage benign code already exist in the address space of the task. Buffer overflow vulnerability can be exploited to perform both code injection and code-reuse attacks. An integer overflow occurs when an operation creates a numeric value that is too large or small to store in the associated representation.

In this paper, we first analyze buffer overflow and integer overflow vulnerabilities in the library of uC/OS II RTOS running on an AVR microcontroller that can be used to develop embedded PLC [12]. We then mitigate them by substituting vulnerable operations/functions with safe ones. The safe functions against buffer overflows check if their input data exceeds its memory buffer size. If the input data exceeds the size of the buffer, our current implementation truncates the string not to incur a buffer overflow attack. Because the truncation results in a loss of data, we also consider another implementation that filters out the exceeded data and issues a warning message. Our method can give programmers an easier way to handle fixed length buffers and reduce the potential for overflows. As a result, we protect uC/OS II against code injection attacks as well as code reuse attacks related to buffer overflows.

II. RELATED WORK

Buffer overflow vulnerabilities are a major cause of cyberattacks on embedded systems as well as on other computing systems [8,9,10,11]. The stack-based buffer overflow attacks execute instructions injected intentionally into the stack region. Sikiligiri showed a stack-based buffer overflow attack using uC/OS II RTOS kernel on a system equipped with the 32-bit Altera Nios II softcore processor, and suggested a defense method to prevent the buffer overflow attack [10]. Sikiligiri prevented the stack-based buffer overflow attacks by disallowing the program counter to fetch the instructions injected into the stack region by an attacker. It is the same approach as the Write-XOR-Execute method published previously. Moreover, it only addressed the code injection attacks and did not handle the code reuse attacks.

Pike et al. proposed a security-aware RTOS, TrackOS that provided an approach to control-flow integrity (CFI) protection

for real-time systems [11]. Their RTOS checks the observed task's control stack against a statically-generated call graph stored in memory. At runtime, a dedicated task traverses the control stack from the top of the stack, containing the most recent return addresses, to the bottom of the stack, and compare the control stack against the static call graph. The CFI approach is known to defend code injection and code reuse attacks. Their approach needs callstack monitoring and can incur runtime overhead. Their method needs call graph generation, its memory space, and control stack traversal which can incur additional overhead.

Bhise et al. [12] developed an embedded PLC for teaching students after investigating the conception and features of PLC and embedded system. They have developed embedded PLC using the AVR ATmega328P microcontroller which is widely used across many embedded designs. Referring to their research, we assume that ATmega128 can be used to develop an embedded PLC. The ATMEL ATmega128 is a Harvard architecture microcontroller where program and data memories are physically separated. Because the program counter can only access program memory, data memory cannot be executed. According to the study conducted by Francillon et al. [13], most of Harvard-based microcontrollers are actually using a modified Harvard architecture. In such architecture, they showed how to exploit program vulnerabilities to permanently inject any piece of code into the program memory of an Atmel AVR-based sensor. Their attack even combined different techniques such as return oriented programming and fake stack injection. Their target system is Wireless Sensor Networks (WSN) that use Mica motes. In the study, they made the following assumption: (1) each node of WSN is running the same version of TinyOS and no changes were performed in the OS libraries, (2) each node is configured with a bootloader, (3) running code has at least one exploitable buffer overflow vulnerability, etc. They mentioned that all existing solutions to mitigate their attack had limitations and future work would explore efficient countermeasures.

III. Analyzing and Mitigating Vulnerabilities in uC/OS

In our study, we conducted experiments with uC/OS II running on an ATMEL ATmega128 device. The ATmega128 is the high-performance, low-power Microchip 8-bit AVR RISC-based microcontroller, which combines 128KB of programmable flash memory, 4KB SRAM, a 4KB EEPROM, and an 8-channel 10-bit A/D converter. We run three tasks, and check memory value using `printf` over UART with disabling/enabling interrupts. Since uC/OS II uses C standard library, there are several library functions known to be vulnerable to buffer overflow. The vulnerable functions are `strcpy()`, `strcat()`, `sprintf()`, `memset()`, `gets()`, etc. Thus, adversaries can attack the tasks which use the vulnerable function. Assume that Task2 contains a certain vulnerable code shown in Fig. 1. In the code, if an adversary know the addresses of `buf[]`, `printb()`, and global variables `a` and `b`, he/she can exploit buffer overflow. Fig. 2 shows each structure of data memory and program memory in this study. The memory address of `buf[0]`, the variable `a`, and `printf()` is 0x05ac, 0x05b5, and 0x017e respectively. In this case, we overflowed `buf[]` and changed the value of the variable `a` to 0x017e from 0x0179 by calling `strcpy (buf,"\x01\x02\x03\x04\x05\x06\x07\x08\x09 \x7e\x01\x7e\x01\xc9")`. Fig. 3 shows the memory structure after exploiting buffer overflow vulnerability.

```
void (*a)();
void (*b)();
void printa(void) {
    printf("normal\n");
}
void printb(void){
    unsigned int j = 0xFFFF
    int k = j;
    printf("hacked!!\n");
}
static void Task2(void *p_arg) {
    char buf[5];

    strcpy(TaskUserData.TaskName, "StartTask");
    a = printa;
    b = printb;
    while (1) {
        a();
        strpcpy(buf, (char *)p_arg[1]);  /* vulnerable function */
        printf("%s\n",buf);
        a();
        OSTimeDly(OS_TICKS_PER_SEC);
    }
}
```

Fig. 1. Sample code of Task2 with buffer overflow and integer overflow vulnerabilities

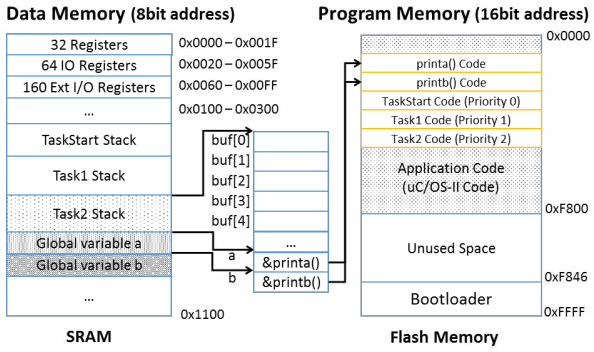

Fig. 2. Memory organization of uC/OS II on ATtmega128 (Before buffer overflow)

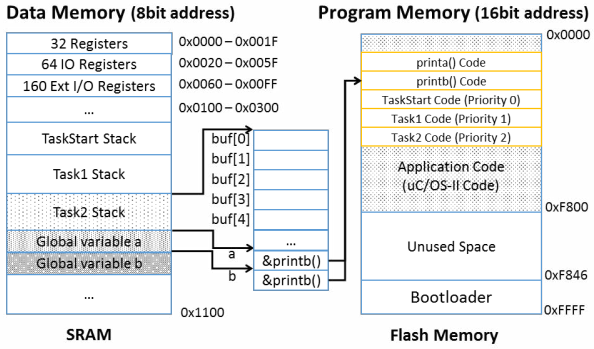

Fig. 3. Memory organization of uC/OS II on ATtmega128 (After buffer overflow)

In order to prevent buffer overflow attacks, we substitutes a vulnerable function with a safe function whose functionalities are same. The safer function is invoked using the macro "`#define strcpy(dest, src) strlcpy(dest, src, sizeof(dest))`" which takes the full size of the destination buffer and guarantee NULL-terminating the result if there is room. `strlcpy()` copies up to `sizeof(dest) - 1` characters from the string `src` to `dst`, NULL-terminating the result if `sizeof(dest)` is not 0 (See Fig. 4). This approach can be used to make writing computer programs easier, and do incur litter or no additional computation. The reasons why we substitute the vulnerable C library functions with the safe functions are as follows: (1) Easy implementation, and (2) Easy compatibility to legacy code.

```
size_t strlcpy(char *dst, const char *src, size_t siz) {
    register char *d = dst;
    register const char *s = src;
    register size_t n = siz;
    /* Copy as many bytes as will fit */
    if (n != 0 && --n != 0) {
        do {
            if ((*d++ = *s++) == 0)
                break;
        } while (--n != 0);
    }
    /* Not enough room in dst, add NUL and traverse rest of src */
    if (n == 0) {
        if (siz != 0)  *d = '\0';        /* NUL-terminate dst */
        while (*s++)  ;
    }
    return(s - src - 1); /* count does not include NUL */
}
```

Fig. 4. A Safer function, `strlcpy`, corresponding to a vulnerable function, `strcpy`

In `printb()` of Fig. 1, an integer overflow occurred, that is an signed integer k = -1 while a unsigned integer j = 0xffff. These can be a source of serious vulnerabilities, such as integer overflows in `OpenSSH` and `Firefox`, both of which allow attackers to execute arbitrary code.

```
#include <limits.h>
int assign_uint (unsigned int val, unsigned int max, unsigned int min) {
    if (val > max) { printf ("too big!\n");  val = max; }
    else if (val < min) { printf ("too small!\n");  val = min; }
    return val;
}
int assign_signed_int (int val, int max, int min) {
    if (val > max) { printf ("too big!\n");  val = max; }
    else if (val < min) { printf ("too small!\n");  val = min; }
    return val;
}
void printb(void) {
    unsigned int j = assign_uint (0xFFFF, UINT_MAX, UINT_MIN);
    int k = assign_signed_int (j, INT_MAX, INT_MIN);
}
```

Fig. 5. An Example code for preventing an integer overflow

The only safe way is to check for integer overflow before it occurs. Fig. 5 shows a safe code against integer overflow. This would be tedious to use systematically. Given that uC/OS is designed for use in mission critical systems, all standard library functions or arithmetic operations have been redeveloped to meet safe and high-quality criteria.

IV. Conclusion and Future Work

In this paper, we briefly address a method to mitigating buffer overflows and integer overflows in `uC/OS II` for PLC systems. Preventing buffer overflows is important due to their popularity and relative ease of implementation. To prevent these vulnerabilities, we have to get the knowledge about the operating system and the processor architecture. That is, it is necessary for the PLC system designer to be aware of the security aspects of the components used in the embedded system and apply suitable measures to prevent the security attacks. For future work, we plan to consider other software vulnerabilities and systematic mitigation techniques against them in embedded systems.

Acknowledgment

This research was supported by (1) the Korea Institute of Energy Technology Evaluation and Planning (KETEP) and the Ministry of Trade, Industry & Energy (MOTIE) of the Republic of Korea (No. 20171510102080), and (2) Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(no. RF-2015R1D1A1A02061946).

References

[1] Sukhyun Seo, Junsu Kim, and Su Min Kim, "An Analysis of Embedded Operating Systems: Windows CE, Linux, VxWorks, uC/OS-II, and OSEK/VDX," International Journal of Applied Engineering Research, India, vol. 12, no.18, pp. 7976-7981, 2017.

[2] Qingchao Wei, Qizhong Cai, and Congse Xie, "Research and implementation of plc editor system," Indonesian Journal of Electrical Engineering and Computer Science, vol. 12, no. 3, pp. 2133-2137, 2014.

[3] Dong Yulin, and Zheng Chunjiao, "Design and research of embedded PLC development system," Computer Research and Development(ICCRD), IEEE 3rd International Conference on, China, vol. 3, pp. 226-228, 2011.

[4] Sampat S. Pawar, and P. C. Bhaskar, "Design and Development of ARM based Real-Time Industry Automation System using GSM," International Research Journal of Engineering and Technology(IRJET), pp. 800-805, 2015.

[5] Alphonsus, Ephrem Ryan, and Mohammad Omar Abdullah, "A review on the applications of programmable logic controllers(PLCs)," Renewable and Sustainable Energy Reviews 60, pp. 1185-1205, 2016.

[6] Sayegh Naoum, Ali Chehab, Imad H. Elhajj, and Ayman Kayssi, "Internal security attacks on SCADA systems," Communications and Information Technology(ICCIT), IEEE Third International Conference on, pp. 22-27, 2013.

[7] Abbasi Ali, "Ghost in the PLC: stealth on-the-fly manipulation of programmable logic controllers' I/O," Proceedings of Black Hat EU, pp. 1-11, 2016.

[8] Davi Lucas Vincenzo, "Code-reuse attacks and defenses," Doctoral dissertation, University of Technische, 2015.

[9] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole, "Buffer overflows: Attacks and defenses for the vulnerability of the decade," DARPA Information Survivability Conference and Exposition(DISCEX'00), IEEE Proceedings, vol. 2, pp. 119–129, 2000

[10] Amjad Basha M Sikiligiri, "Buffer overflow attack and prevention for embedded systems," MS thesis, University of Cincinnati, 2011.

[11] L. Pike, P. Hickey, T. Elliott, E. Mertens, and A. Tomb, "Trackos: A security-aware real-time operating system." International Conference on Runtime Verification. Springer, pp. 302-317, 2016.

[12] Bhise, Kalpana, and Sharwari Amte. "Embedded PLC Trainer Kit with Industry Application." International Journal of Engineering and Technical Research (IJETR), Vol. 3, Issue 4, 2015.

[13] Francillon, Aurélien, and Claude Castelluccia, "Code injection attacks on harvard-architecture devices." Proceedings of the 15th ACM conference on Computer and communications security, 2008.