Exploring the Security behind Embedded Systems

Devon Bautista, Ryan Jones

CSE 325 - Embedded Microprocessor Systems
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University

Spring 2018

Abstract

With the continual growth of smart technology and the Internet of Things (IoT), the popularity of embedded systems in real world applications is growing. Embedded systems are products found in electronics, semiconductors, telecommunications, and networking industries [5]. Examples of these products include radios, watches, vehicles, traffic signals, medical devices, and smart devices. Embedded systems play an important role in these fields, but what exactly are they? Most embedded systems have only one dedicated function inside of a larger system that hides complexity from the user. Unlike an operating system, which carries out multiple tasks at once, embedded systems remain closed systems intended for a single task. Having only one dedicated function reduces possible failures that could arise from the overhead involved in having multiple simultaneous functions. Operating systems impose too much risk of failure in real time applications, so it is absolutely essential to have embedded systems that do not fail. The standards of an OS are unacceptable for embedded systems as failure can be catastrophic. Airplanes and cars are examples of transportation systems which rely on the use of embedded technology. Through the aid of embedded devices, these media of transportation have become safer, reliable, and more efficient. Due to our reliance on embedded systems, it is important that these systems have good security and defense mechanisms so that system vulnerabilities do not jeopardize our safety. Embedded systems must not fail, so it is important to teach professionals and hobbyists how to prevent future bugs and vulnerabilities. Regardless of an individual's experience level, every programmer and designer needs to take into account security issues when designing embedded systems. This proposal offers a review in addressing the security vulnerabilities in designing embedded systems. In this paper, common vulnerabilities such as stack buffer overflows will be explored to demonstrate the risks of vulnerable embedded systems.

Introduction

IoT has allowed a new wave of computer hobbyists to develop projects in home automation. Microprocessors have become very cheap as the cost of the components that make up the system have become cheaper. Smart objects can now interact with other nodes of communication and collect data autonomously. The Arduino has easily become the de facto prototyping tool for embedded systems.[1] The Arduino is an excellent embedded platform for its beginner friendliness, easy-to-install IDE, and easy to follow documentation. Despite its popularity, the Arduino business model has never taken into account the security measures needed for an embedded device to work in a real world system. The Arduino still suffers from basic security vulnerabilities such as basic buffer overflows and reusable code attacks [1]. These boards impose the risk of teaching the developer unsafe practices which could result in unsafe and vulnerable systems. Proper security measures addressed could result in easily preventing these vulnerabilities and result in a safer system for the IoT hobbyist. It can be quite overwhelming for a developer to have to learn a new skill set such as programming and embedded design, as well as security. The challenges when working with Arduino include an unfamiliar microprocessor architecture and developer tools used with the board. This paper will help identify both software and hardware vulnerabilities, and show the best way to avoid these issues through recommended programming and design practices.

Syllabus Course Objectives

This research project is to be done alongside of CSE 325 - Embedded Microprocessor Systems at Arizona State University. It aligns with the following course objectives.

- **To develop an ability to analyze microprocessor-based embedded systems, memory components, and bus connections.**
  This project will analyze different microcontrollers with an emphasis on the Arduino platform. Through this project both the hardware and software of the board will be analyzed to determine potential security vulnerabilities such as stack buffer overflows. The research of this device intends to give the read a better understanding of the internal workings of the Arduino embedded platform.

- **To develop design skills for modular application and system software in microprocessor-based embedded systems**
  This project will help explore better security practices when designing microprocessor-based embedded systems. Having a general understanding of buffer overflows can help apply architecture specific attacks.

- **To apply software development tools to efficiently implement and debug programs running in microprocessor systems**
  The software development tools such as the IDE and compiler will be addressed to explain software vulnerabilities. The IDE and compiler are important for the user because they are tools in which can possible identify bugs in the system.

- **To gain an ability to analyze I/O interface units and to design software for managing I/O operations**
  Analyzing Arduino I/O interfaces and design patterns to help identify weakness in a system. I/O operations allow an embedded system to interact and collect data from the outside world. The I/O operations are also useful tools for finding bugs.

Arduino Boards

The Arduino models observed in this project are:

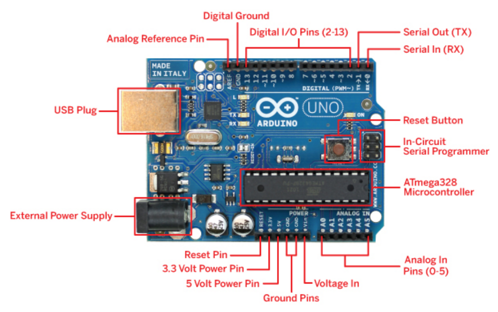

Figure 1: Arduino Uno (ATmega328)

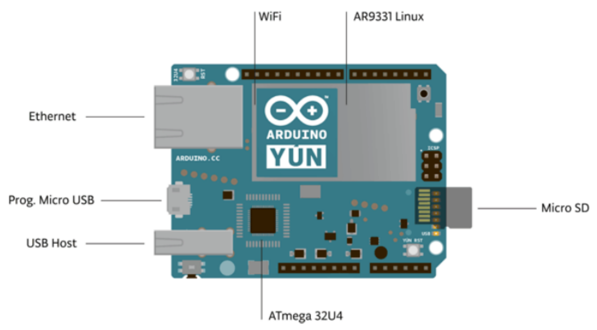

Figure 2: Arduino Yun (ATmega32U4)

Why do Embedded Systems use C?

When embedded systems were first being developed, microprocessor-specific assembly was the only method of writing code on an embedded device. Eventually C was introduced to the world by Dennis Ritchie and Ken Thompson to create UNIX, and developers moved away from assembly. C was easier to read, write, and maintain over assembly code which also took longer to produce. Assembly would continue to be used alongside embedded C, but only in areas where ultra high timing and efficiency were needed in the device [6]. C is what is known as an "unsafe" programming language, meaning it doesn't have a lot of the safety checks such as garbage collection that can help prevent memory vulnerabilities such as buffer overflow. As a result, C is a much more lightweight language compared to languages like C++, Java, and Python. It comes at the cost of giving the programmer the responsibility of writing safe code that would normally be checked under the compiler and garbage collection. While there are other safer programming languages that can prevent buffer overflow vulnerabilities, it would be impractical to make embedded systems all use the same safe language. In addition to having a safe language in embedded systems, all microprocessors use a wide range of different compilers. Embedded Systems are very limited in the amount of memory, storage, and processing power that they can have. Due to resource limitations, embedded systems need to have their own standard C library if they are to compile and execute C code. The standard C library is nothing more than a collection of standard functions that can be accessed by all programs written in C.[7] The most common standard C library is known as glibc which stands for GNU C Library and is used on mostly all Linux Distributions. As the need the have a smaller memory footprint is needed in embedded systems, the GNU C library is too large for most embedded systems. To take better measures in reducing their memory footprint, constructing smaller binaries is needed when working with embedded systems. For this reason other C libraries such as uClibc, dietlibc, and musl libc exist for the sole purpose of embedded system development. The Arduino also has its own C library and uses the avr-gcc compiler. C is a bare metal programming language that gives the user complete control over the physical memory of their device. The combination of using C and lack of debugging tools on the Arduino platform impose serious security risks. It is not a good beginner programming language and can cause severe memory issues if code is not designed and implemented properly. In addition to the difficulties with C, the lack of a debugger in the Arduino in particular makes it even more problematic.

Importance of Computer Architecture

In embedded systems, knowing the computer architecture can help one understand a lot about the internal workings of a system. Every microprocessor has its own instruction set and writing assembly is unique to its architecture. Assembly on the x86 architecture, the architecture used by most general-purpose personal computers, is very different compared to ARM assembly, which is mostly used on embedded systems. It is important to read the data sheets when trying to understand the architecture and assembly of a microprocessor. The Arduino Uno uses the AVR architecture while the Arduino Yun uses the MIPS architecture, but both AVR and MIPS follow the Reduced Instruction Set Computer (RISC) instruction set. RISC only allows for simple instructions on a single clock cycle. As a result, these microprocessors will have heavier RAM usage, limited address modes, and only have a small number of instructions. RISC also has some very important advantages. The reduced instructions require less transistors than complex instructions and leave more room for general purpose registers. This has little to do with buffer overflows and how they work, but it is worth understanding the basic inner-workings on the embedded system one is working with.

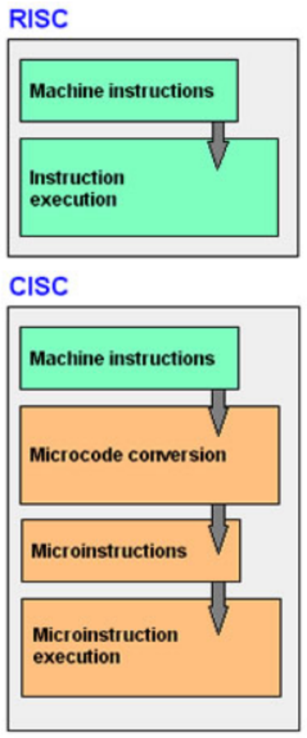

Figure 3: RISC vs. CISC Instruction Sets

Buffer Overflows

What Are They?

A buffer overflow is the root of most binary vulnerabilities. The basic concept of one is injecting code into an executable portion of memory that is supposed to be inaccessible by exceeding the limits of an input buffer. This is easily accomplished if the exploited program does not check the input size before writing it to the buffer. There are two types of buffer overflows that can occur in embedded systems, a stack-based overflow and a heap-based overflow. To understand these two security vulnerabilities, it is im-

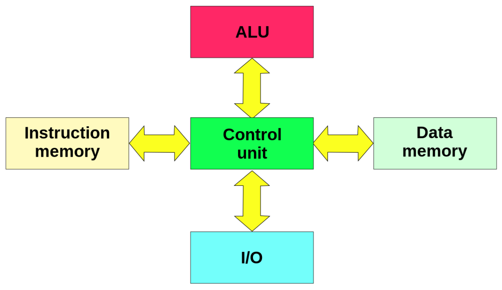

Figure 4: Harvard Architecture

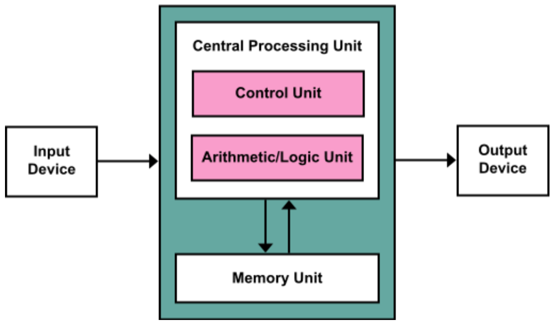

Figure 5: Princeton (Von Neumann) Architecture

portant to understand the memory segmentation of a compiled program. In every program, memory is divided up into five sections: text, data, bss, stack, and heap. Memory allocations are features given to the developer from the standard C library and done within the heap segment of memory. To prevent heap buffer overflows, it is the responsibility of the developer to release and free any memory they allocate, it is at their own risk. Some functions of the standard C library are vulnerable to buffer overflow exploits. In our study, we found stack buffer overflows to be a more common vulnerability in embedded systems.

Buffer overflows currently account for over more than 50% of system exploitation vulnerabilities and they are continually increasing. [3] Architecture plays a key role in determining the ease of performing a buffer overflow. In the Princeton (also known as Von Neumann, **Figure 5**) computer architecture, the program data memory and program instruction memory share the same bus, and are therefore contiguous in memory. This contiguity makes buffer overflow vulnerabilities more easily exploitable. This can also be a performance bottleneck in the Princeton computer architecture because instructions and data cannot be fetched at the same time; however, it is simpler than Harvard. This is because the Harvard computer architecture has separate buses for data and instruc-

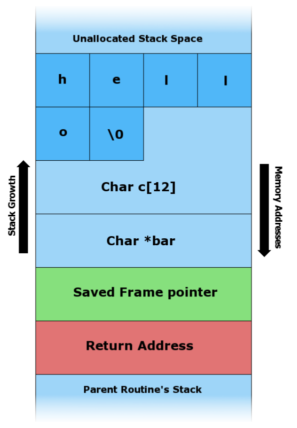

Figure 6: Buffer that has a size of 12. Since the value "hello" is less than 12 bytes the buffer has enough padding in memory

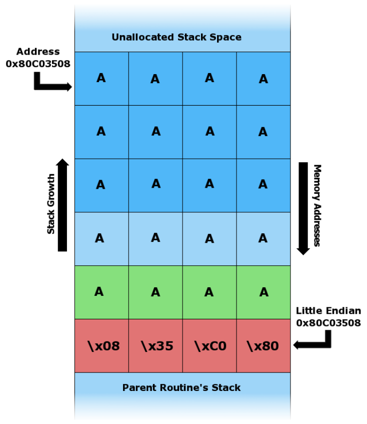

Figure 7: Buffer Overflow where the value is larger than the size of buffer and shell code is written over the return address

tions and can allow this simultaneity. Although it may seem impossible to perform a buffer overflow attack on a Harvard architecture computer, that is not the case albeit it is more difficult. Therefore, buffer overflows are more common in general-purpose computers which often have a Princeton architecture than embedded devices, which often implement the Harvard architecture, yet they are still possible on embedded systems so measures must be taken to ensure this vulnerability cannot be exploited. The concept behind a buffer overflow is writing shell-code located on the stack. Since the stack is arranged in a FILO (First In, Last Out) structure, a buffer that is overwritten will overwrite the next instruction in memory. In the case of this overflow, writing past the stack will overwrite the memory address .

Buffer Overflows in Arduino Uno

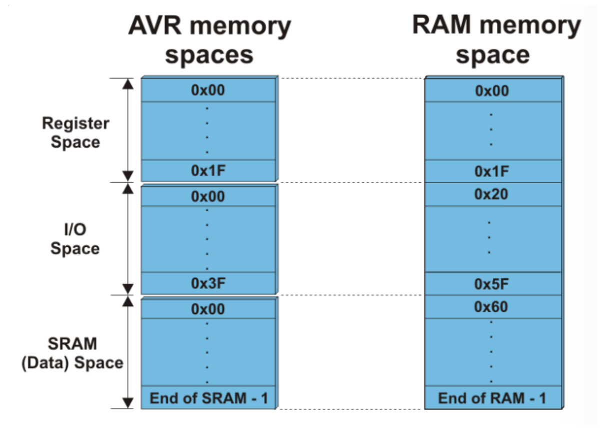

Figure 8: Memory Segmentation

The Arduino uses it's own IDE which is equipped with very limited tools and a compiler that makes it harder to find bugs in the system. It should be noted that the Atmel processor does not have sufficient debugging tools like x86 and ARM CPUs. The Arduino IDE does not warn the user of a segment violation if the interpreted program runs out of memory. [1] The ATmega328 microprocessor architecture of the Arduino Uno still suffers basic buffer overflow vulnerabilities despite using a Harvard Architecture. Until 2009, it was generally accepted that stack buffer overflow code injections were impossible on Harvard Architecture [2]. Previous stack buffer overflow exploits have involved writing shellcode to a known location in the stack memory and overwriting the return address to exploit the system. This risk of the ATmega328 and most other Harvard-inspired microcontrollers is that they follow the design pattern of the Harvard architecture. Due to the design of the Harvard architecture, the memory segment that contains the instructions is in a completely separate location of the memory that contains the data. This prevents the traditional buffer overflow technique explained above. The architecture of the Arduino is the AVR8 architecture. It's a modified version of the Harvard architecture which follows an 8-bit RISC design pattern. This modification has allowed possible stack buffer overflow exploitation by using assembly instructions that connect register memory and SRAM (data) memory. This makes a code injection for a simple stack buffer overflow impossible on this architecture. Although it is possible to override the return address, the area in memory that will be overwritten will be flash memory and not SRAM. The only way to execute code on the ATMega328 is through flash memory, so the shellcode will need to be written here first through the functions `WriteFlashByte()` and `WriteFlashPage()` and knowing the address of where to execute the function in memory. One can use the avr-objdump command to determine where the addresses reside in memory. In order to execute a buffer overflow on the ATMega328 you need to have access to source and the address in flash memory to perform the dump. Current buffer overflow techniques for the Harvard architecture include

- Return into libc attacks
- Return oriented programming

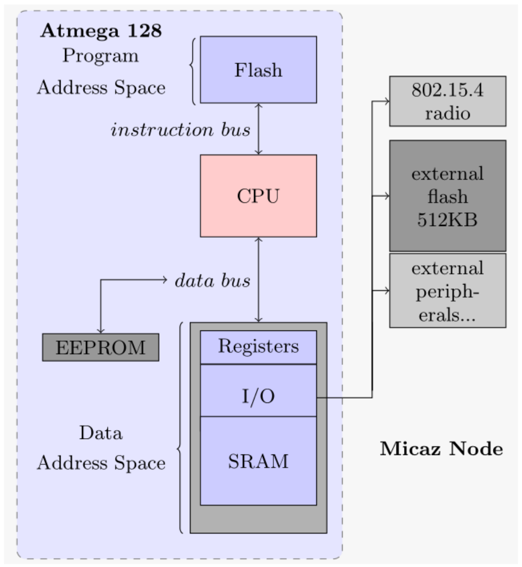

Figure 9: ATmega Memory Segmentation

Arduino Yun

This section intends to reflect on the design choices of the Arduino Yun and how it compromised its security. The Arduino Yun was an Arduino board specifically designed for IoT. IoT companies continue to produce products focused on cost instead of security [1] The intent of the Arduino Yun was to combine the features of a real time embedded system with a higher level Linux Operating System that could provide advanced communication capabilities. It combines the low level end of an embedded system with a higher level architecture of a Linux Operating System. The Arduino Yun has a lower level ATmega32u4 processor for embedded tasks and an Atheros AR9331 processor for the Linux Operating System. These two processors are connected through a bridge that is written in Python. Through the research done at Spanish Cyber Command, Universidad Carlos III de Madrid, Spain, and Bournemouth University, it was found that the security of the Linux environment could be compromised by exploiting the lower levels the security mechanisms. This was due to the ATmega32u4 chip executing all commands with root privileges. In addition, the Arduino Yun ships an outdated version of the Linux Kernel.[1] Users have to manually update the board from source, which is quite difficult for non-experienced users. This imposes major security risks in using an Arduino for embedded systems.

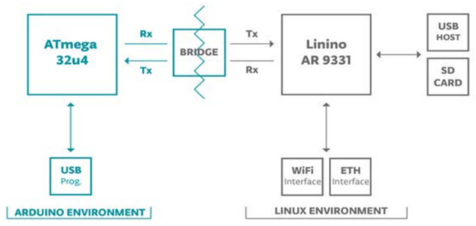

Figure 10: Arduino Yun Bridge Diagram

Conclusion

As shown, buffer overflows are simple, yet they are a foundation for many security vulnerabilities. C is a low-level programming language that allows precise memory management, which makes it an ideal language for embedded systems development. However, it doesn't natively limit writing to buffers, which is a big problem when handling raw user input. Although this is not easily exploitable on embedded systems due to its use of flash memory, it is important for both programmers and designers to be aware of this possibility when designing and deploying embedded systems. Embedded systems appear everywhere in our society, from transportation to life-saving devices and it is important that these devices remain secure to prevent catastrophic failure that could cost lives. This can be perpetuated by the understanding of general security theory and practices when creating an embedded systems, and the knowledge transfer of secure C programming practices to that of embedded systems.

This is by no means a comprehensive report on the security vulnerabilities of embedded systems, but an exploration of a particular vulnerability and how it can be exploited on particular devices. As of this writing, embedded systems security (and especially that pertaining to IoT) is a cutting-edge research topic. More research is needed in these fields to better understand the risks and fixes of security for embedded systems.

As aforementioned, this paper only covered a few particular vulnerabilities and how they can be exploited on embedded devices. It did not cover things that also may be high-risk, such as security vulnerabilities in communications between both embedded devices and the Internet. Risk here is actually much more likely to be exploited as IoT companies are more focused on production and less focused on security. Network security is another whole issue that needs to be addressed and researched, but this paper was more focused on binary security risks.

Nevertheless, the goal of this paper is to make embedded programmers aware of the risks and exploits present in C-based embedded systems and to start the transfer of knowledge from secure general-purpose C programming practices to that of embedded systems.

References

[1] Carlos Alberca, Guillermo Suarez-Tangil, Sergio Pastrana, Paolo Palmieri. *Security Analysis and Exploitation of Arduino devices in the Internet of Things* Workshop on Malicious Software and Hardware in Internet of Things. (Mal-IoT). Como, Italy, May 16, 2016.

[2] Aurélien Francillon, Claude Castelluccia *Code Injection Attacks on Harvard-Architecture Devices* Saint Ismier Cedex, France January 22, 2009.

[3] Zili Shao, Qingfeng Zhuge, Yi He, Edwin H.-M. Sha *Defending Embedded Systems Against Buffer Overflow via Hardware/Software* Las Vegas, NV, USA 8-12 Dec. 2003 IEEE

[4] *The Computer Language Benchmarks Game* https://benchmarksgame.alioth.debian.org/

[5] SRIVATHS RAVI and ANAND RAGHUNATHAN *Security in Embedded Systems: Design Challenges* Cryptography Research Texas Instruments Inc.

[6] Akshay Daga *Embedded C* https://www.engineersgarage.com/tutorials/emebedded-c-language

[7] *The Linux Man Pages*

[8] Karn *Where Are Static Variables Stored (in C/C++)?* Stack Overflow, Stack Exchange Inc, 28 July 2012, stackoverflow.com/questions/93039/where-are-static-variables-stored-in-c-c.

[9] Elias Bachaalany. *StarCraft: Remastered Emulating a Buffer Overflow for Fun and Profit.* 0xeb, Blizzard Entertainment, February 3, 2018. `0xeb.net/wp-content/uploads/2018/02/StarCraft_EUD_Emulator.pdf`.

[10] Jim Blandy, Jason Orendorff. *Programming Rust* Fast, Safe Systems Development O'Reilly Media Inc. Sebastopol, CA, January 5, 2018.