Ryan M. Gerdes
Department of Electrical and Computer Engineering
Utah State University

March 04, 2016

PROJECT II: RETURN-ORIENTED PROGRAMMING ON THE ARM CORTEX-M4

Problem 1. *System design.* You are to implement a basic *echo* program for the Tiva C (TM4C123GH6PM) UART interface that is vulnerable to a buffer overflow attack. The purpose of an echo program is to simply send back the same data it receives.

Your implementation will need to make use of the following vulnerable function

```
int echo()
{
  char buffer[32];

  gets(buffer);
  printf(buffer);

  return 1;
}
```

To ease development, you may make use of the UART equivalents to `gets()` and `printf()` found in `uartstdio` (part of the TivaWare driver library). If need be, you can modify these functions in order to effect the exploit.

Problem 2. *Exploitation of system.* You are to write a return-oriented program (ROP) that exploits the buffer overflow vulnerability in the above function to inject arbitrary code onto the Tiva C (i.e. perform a flash rewrite). Upon reset, the uC should blink one of the on-board LEDs at a human-observable rate (e.g. 1 Hz). Rather than waiting for the uC to be restarted, upon successful injection your exploit should reset the program counter to execute the above code. If not taking the course for 6000-level credit, you are allowed to disable any compiler protections for buffer overflows (the *ARM Compiler toolchain v5.02 for μVision Compiler Reference* will be useful here).

You are to confine your search for gadgets to the on-board ROM of the TivaC. Section 2.4 of the TM4C123GH6PM Microcontroller manual indicates that addresses `0x0100.0000`–`0x1FFF.FFFF` are reserved for ROM; furthermore, Section 8.2.2 states that at least part of this ROM contains the *TivaWare Peripheral Driver Library*. That is, each TivaC ships with embedded code that a developer can use to configure certain peripherals. You are to utilse this code for gadget discovery/selection as the authors of *The Geometry of Innocent Flesh on the Bone* paper used libc. You will be provided with the disassembly of the ROM.

Problem 3. *Reporting requirements.* Provide the assembly necessary to flash the uC that you base your ROP code on; discuss how your ROP program achieves the objective of the original code. You need to document all the individual gadgets used in the project, as well as how they fit together. Gadgets you may find useful would allow for polling a bit and branching when it has changed, loading data from an arbitrary memory location into a register, storing data from a register to an arbitrary memory location, pushing to the stack, and moving data between registers. Diagram your gadgets and the completed ROP based on Figure 16 of *The Geometry* *of Innocent Flesh on the Bone* paper. Additionally, provide the source of the ROP; i.e. the injected stack (give this as byte code like the shellcode in *Smashing The Stack For Fun And Profit*). Show the contents of the stack before and after overwriting the buffer and the contents of Flash before and after rewriting it, as well as a screenshot that shows the uC executing the injected code. Finally, follow the reporting guidelines/structure issued for the first project.

My grading criteria will include:

1. Demonstrated exploit of buffer overflow by showing contents of the stack
2. Discussion of individual gadgets and their connections
3. Diagram of gadgets and their connections
4. Explanation of how code was exploited
5. How gadgets were found
6. Breakdown of traditional flash procedure
7. Ingenuity (originality, obstacles overcome, exploit code in program form, etc)