Vector Rewrite Attack

Exploitable NULL Pointer Vulnerabilities on ARM and XScale Architectures

Barnaby Jack

Staff Security Research Engineer

Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, California 94089
USA
408.745.2000
1.888 JUNIPER
www.juniper.net

Rev. 5/3/07

Table of Contents

Introduction

NULL pointer dereference flaws typically account for the majority of published denial-of -service attacks, both locally and remotely. A NULL pointer dereference occurs when a pointer with the value of 0 is assumed to be a valid memory location, and that pointer is then accessed. A NULL pointer dereference is rarely more than an annoyance, with the worst case scenario typically resulting in a software crash. A write from, or read to, the memory address 0x0 will generally reference invalid or unused memory.

In the case of the ARM and XScale architectures, the 0x0 address is not only mapped in memory, but also serves an important purpose; the Exception Vector Table is located at memory offset 0. Because many Real-time Operating Systems on ARM based devices run in Supervisor (SVC) mode, memory access is unrestricted.

ARM Exception Vectors

When an exception is generated on the ARM processor, execution is forced to a fixed memory offset that corresponds to the thrown exception. These fixed offsets are called the exception vectors. There are eight exception vectors, each 32 bits in length, mapped starting at address 0.

The Exception Vector Table

- 0x00000000: Reset vector
- 0x00000004: Undefined instruction vector
- 0x00000008: Software interrupt (SWI) vector
- 0x0000000c: Prefetch abort vector
- 0x00000010: Data abort vector
- 0x00000014: Address exception trap
- 0x00000018: Interrupt request (IRQ) vector
- 0x0000001c: Fast interrupt request (FIQ) vector

When an exception occurs, the R14 register for the current exception mode holds the callers address.

Some interesting exceptions that may aid in exploitation are:

- The **Reset** exception occurs when a reset is performed on the processor. Execution then begins at address 0.
- The **Undefined Instruction** exception occurs when an attempt is made to execute an invalid instruction.
- The **Software Interrupt** exception. This exception occurs when an SWI instruction is executed. The SWI instruction is used to switch to Supervisor mode and call operating system functions.
- The **Interrupt Request (IRQ)** exception occurs when an external hardware interrupt is triggered. The IRQ is used for general purpose interrupt handling.
- The **Fast Interrupt Request (FIQ)** exception occurs when an interrupt with a high priority (fast) is raised.

The Vector Rewrite Attack

The following is a disassembly snippet of the vector table beginning at memory address 0:

```
ROM:00000000 18 00 00 EA            B       loc_68
ROM:00000004 6C F0 9F E5            LDR     PC, =loc_9A080
ROM:00000008 6C F0 9F E5            LDR     PC, =loc_9A084
ROM:0000000C 6C F0 9F E5            LDR     PC, =loc_9A088
ROM:00000010 6C F0 9F E5            LDR     PC, =loc_9A08C
ROM:00000014 6C F0 9F E5            LDR     PC, =loc_9A090
ROM:00000018 6C F0 9F E5            LDR     PC, =sub_9A094
ROM:0000001C 6C F0 9F E5            LDR     PC, =loc_9A0A8
```

The exception vectors are simply branch instructions, each corresponding to a specific exception type. They are instructions that branch to other code locations. As the ARM assembler language allows operating on PC (program counter) directly, a LDR (Load Register) or a MOV instruction may be used to set the program counter – essentially this is the same as a branch. The vector table is a set of branch instructions that are executed when the corresponding exception is raised. If one were to overwrite this table, the flow of execution could be altered by an attacker, allowing for straight-forward exploitation.

Consider the following wrapper for malloc():

```
void *
xmalloc(size_t size)
{
        if (size!=0)
        {
                void *block = malloc(size);
                if (block == NULL)
                {
                        fprintf(stderr, "Out of memory\n");
                        exit(EXIT_FAILURE);
                }
                return block;
        }
        return NULL;
}
```

This wrapper will return 0 or NULL after being passed a 0 length size. If no error checking is performed on the return value, the program attempts to write to the NULL memory offset.

The actual exploitation scenario arises when the source data being written is attacker-defined. It is then possible to replace the vector table and insert a new branch to code of our choosing.

Here is a real world example taken from an earlier version of the libpng library – A vulnerable version of this library has been seen on a popular ARM based cell phone. Libpng implements its own wrapper to **malloc()** that will return a NULL pointer on error, or on being passed a 0 byte length. The length being passed to **malloc()** is taken from the PNG image.

```
#ifdef PNG_MAX_MALLOC_64K
   if (length > (png_uint_32)65535L)
   {
      png_warning(png_ptr, "iCCP chunk too large to fit in memory");
      skip = length - (png_uint_32)65535L;
      length = (png_uint_32)65535L;
   }
#endif

   chunkdata = (png_charp)png_malloc(png_ptr, length + 1);
```

This is a common integer wrap. If a length field of -1 is supplied, the addition rounds to 0, and the **malloc()** wrapper returns a NULL pointer.

chunkdata is later used as a destination for a **memcpy()**. It is now possible to copy user-defined data from the image and overwrite memory starting at address 0.
To gain control of the processor and execute code, an exception vector must be overwritten with a new branch and the corresponding exception must be raised. What exception vector is the best choice?

The *reset* vector would work, but a reset would have to be somehow forced, or an attacker would have to wait for a reset to be issued.

The obvious choice for a vector to overwrite is one of the interrupt vectors:

The Software Interrupt Vector is called whenever an SWI instruction is executed. The SWI instruction is used to execute system calls on the underlying RTOS (Real Time Operating System). If the RTOS makes use of system calls, then the SWI vector is a good choice to overwrite. Overwriting the SWI vector would execute the replaced branch on the next system call.

If the embedded system does not make use of system calls, then the IRQ vector is the next best option. The IRQ vector will be triggered continually, as it handles each vectored interrupt, which include timers, UART, and so on. The IRQ vector is at address 0x18.

The vector table of the target is static and will only change with firmware revisions. Once a copy of the vector table is acquired, the vector table can be copied and re-sent verbatim.

Assuming the target is using the vector table shown earlier, a successful attack would be as follows:

The firmware vector table would be sent byte for byte from offset 0x0-0x14 (the reset vector may need to be modified to remove NULL bytes). The IRQ vector at 0x18 would be replaced with a branch instruction to branch to the location after the vector table. The code that follows the vector table must first disable both FIQ and IRQ interrupts by setting bit 6 and bit 7 of the CPSR register, rewrite the IRQ vector address with its original vector address, and then execute the attack code.

Prevention

If the processor is equipped with an MMU (Memory Management Unit), the vector area may be marked as non-writeable.

On ARM9 and newer cores, the vector table can be relocated to a high address by driving the HIVECS processor pin HIGH. Vectors are then mapped to address 0xFFFF0000.

On the XScale core, the vector table can be mapped high by setting bit 13 (Exception Vector Relocation Bit) of the ARM control register to 1. This will map vectors to 0xFFFF0000.

Mapping the vectors to a high address essentially protects from this vulnerability class.

Conclusion

These flaws are not as common as the stack or heap overflow, but when found, they are certainly more reliable to exploit. This is yet another weapon to add to an attacker's arsenal. In this case, prevention is fairly simple. Vendors, take note.

About Juniper Networks

Juniper Networks develops purpose-built, high-performance IP platforms that enable customers to support a wide variety of services and applications at scale. Service providers, enterprises, governments and research and education institutions rely on Juniper to deliver a portfolio of proven networking, security and application acceleration solutions that solve highly complex, fast-changing problems in the world's most demanding networks. Additional information can be found at www.juniper.net.