

HY558 Report

Stride Polymorphic SLED Detection Through Instruction Sequence Analysis

One of the most popularized attack is the buffer Overflow attack. This type of attack occurs when one passes data that exceed, in length, the buffer, thus overriding useful data. The aim usually is to override the return address of the current function in order to point to malicious code.

Defending against such attacks consists mainly of detection before the target system(e.g. in the firewall). In order to do so we exploit certain weaknesses of these attacks. One of which is the fact that when hijacking the return address, we usually do not know the exact address of the malicious code. To resolve this attackers typically place sequences of NOP before the malicious code. This sequence is called a SLED. SLED is a sequence of instructions that executes correctly from multiple if not all byte offsets. This way the attacker can jump at any point inside the SLED and eventually execute the malicious code.

There are seven type of SLEDs:

- 1. Simple NOP-SLED**

This simply a sequence of NOP instructions. It can be executed starting at any byte offset

- 2. One-byte NOP-equivalent SLED**

This is a sequence of one-byte instructions that have no effect.(e.g. increasing an unused register)

- 3. Multi-byte NOP-equivalent SLED**

This is the same as number 2 but we use multi byte instructions. Since the SLED has to be executable at every byte offset, the arguments of these instructions has to be One-byte NOP-equivalent instructions.

- 4. Four-byte Aligned SLED**

Based on the observation that modern compilers usually align the stack at 4 bytes, we can use pairs of 2-byte non-destructive instructions.

- 5. Trampoline SLED**

Once inside the SLED we know the distance to the malicious code and so we could instead of NOP-equivalent instructions place jumps to the malicious code.

- 6. Obfuscated trampoline SLED**

Since sequences of jump instruction can be easily detected, we can interleave jump instructions with NOP-equivalent instructions.

- 7. Static Analysis Resistant SLED**

This type of SLED consists of code that cannot be determined statically. Such an example would be a branch instruction whose target cannot be determined statically

The paper presents a new algorithm for detecting SLEDs called stride. Alongside that, the current available detection mechanisms consist of the following:

1. NDIS signatures

This is a pattern matching mechanism. It can easily detect sequences of NOP instructions and thus SLEDs of type 1, but nothing beyond that.

2. Fnord

Fnord extends Snort an NDIS signature detection system. Fnord can detect NOP-equivalent instructions and so it can detect SLEDs of type 1, 2 (and potentially 3).

3. Abstract Payload Execution (APE)

This detection mechanism looks for long enough sequences of valid instructions. An instruction is considered valid if it decodes properly and consists of valid memory operands. The length that a sequence has to be is defined as the MEL parameter. When APE encounters a branch it follows both paths and considers the longest valid of the 2. This detection mechanism can detect SLEDs of type 1 through 4.

4. STRIDE

Stride is the detection mechanism proposed in this paper. Stride is based on the fact that SLED are executable starting at every byte offset. Based on this observation STRIDE can detect even smaller SLEDs than APE.

```
stride(input, input_size, sled_length) {
    for (i=0; i < input_size-sled_length; i++) {
        if (find_sled(input+i, sled_length))
            return TRUE;
    }
    return FALSE;
}

find_sled(data, len) {
    for (j = 0; j < 4; j++)
        for (i = j; i < len; i+=4)
            if (!valid_sequence(data+i, len-i))
                return FALSE ;
    return TRUE;
}

is_valid_sequence(data, len) {
    /* decode "len" instructions in buffer "data" */
    res = decode(data, len);
    if (res == VALID_DECODE) return TRUE;
    if (res == ENDS_IN_JMP) return TRUE;
    return FALSE;
}
```

Figure 4. Pseudo-code for STRIDE algorithm

Stride can detect SLEDs types 1 through 6. Stride also outperforms all the previous mechanisms in detection rate, and false positives. Stride also seems significantly faster than APE. This is due to the fact that when APE encounters a branch it follows both code paths, where stride merely considers it a valid instruction.