CS 351 : The “Tricky Jump” Technique

Motivation

Virus writers often try to insert calls to their virus code on top of existing code, overwriting as little of the existing code as possible. They prefer to have the existing code of the infected application run correctly, so that the user of the application does not suspect that it has been infected. In order to accomplish this, many virus writers prefer to accomplish a function call to their virus code with as little disturbance of the machine state as possible. They accomplish this by overwriting a single instruction in the application with a technique known as a tricky jump.

Example

A typical application might have several functions that end with the following code:

\[
\text{xor eax, eax ; zero out return value} \\
\text{ret }
\]

The function returns an integer (perhaps representing a boolean) and in the example code, the return value (held in register eax) is set to zero. This is a common code pattern. If the virus can find such a function that is near the location that will hold the virus code, the distance to the virus code will be small enough to permit the following instruction to be the same size as the xor instruction above:

\[
\text{push offset _virus_code_func_name}
\]

This permits the virus to simply overwrite the xor instruction, producing the following epilogue code for the infected function:

\[
\text{push offset _virus_code_func_name} \\
\text{ret}
\]

Let us analyze what will happen here. The stack frame has the address to which the infected function is supposed to return. The address of the virus code is pushed onto the top of the stack. The ret instruction immediately pops this virus address off the stack and transfers control to it. Thus, the ret instruction has been used not to return to the caller, but to jump to the virus
code. This is why it is called a *tricky jump*. Notice that after this pair of instructions (push followed by ret, which pops the stack), the stack has been returned to its original state, with the proper return address residing at the top of the stack.

What happens when execution reaches the end of the virus code, after the virus has replicated itself and delivered its payload code somewhere into the system? The virus is written with the same epilogue as the infected function had before it got infected:

```
xor eax,eax ; zero out return value
ret
```

The return value is set to zero. The ret instruction pops the original return address for the infected function, transferring control to where it would have been in the uninfected program. Thus, the flow of control has simply taken a detour through the virus code, but the machine state has been left unaffected (except for whatever the malicious effects of the virus were, of course).

For the purposes of Assignment 1, you may accomplish a tricky jump by simply adding the push offset instruction before the ret instruction in one of your functions; you do not have to overwrite any code.