Adnar Lozano
CSE 320
11/1/17

Lab # 6

A safe programming language is one that sticks to certain rules which help the language achieve safety. Rust is really good in being a safe language because of its rules. On the other hand, languages that do not have specific rules, are not safe and can often lead to them being vulnerable to malicious and non-malicious attacks.

A good example of vulnerability can be found in Microsoft's win32k.sys. Win32k.sys is a Windows Kernel-Mode driver that allows local users to gain privileges via a crafted application, as exploited in the wild in October 2014, aka "Win32k.sys Elevation of Privilege Vulnerability." This driver basically allows remote code execution, which can lead to disastrous consequences. This vulnerability could be exploited if an attacker somehow had a user open a crafted document via email, or visit an untrusted website that could contain some malicious software embedded in the crafted document. It could be a TrueType font or any other filesystem. This vulnerability can be triggered by abusing a NULL pointer through the "*xxxSendMessageTimeout*" which would result in code execution. In order for this to happen, the attacker must rely on the user accessing the email or clicking on that website link. If the attacker actually exploits this vulnerability, then the entire operating system could be compromised. Once the attacker has elevated privileges, the attacker would have access to all the data, resources, applications management, files, and user's information of the system. Essentially, the attacker is able to modify any file in the system for a specific user or for all the users. This vulnerability affected various Microsoft's systems such as: Windows Server 2003 SP2, Windows Vista SP2, Windows Server 2008 SP2 and R2 SP1, Windows 7 SP1, Windows 8, Windows 8.1, Windows Server 2012 Gold and R2, and Windows RT Gold and 8.1

This vulnerability is not difficult to exploit and could compromise the entire system at any moment's notice. However, here's where Rust comes in handy. Rust does not have a NULL pointer, therefore this vulnerability in non-existent in an environment written in Rust. Rust's system of ownership does not allow for NULL pointers to exist; therefore there's nothing to dereference going on that could lead to any NULL pointer vulnerabilities.

References:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4113