jkpatch

2/18/2018

golden

PS4RPC

The *PS4RPC* object is your key to success. Simply instantiate a new *PS4RPC* object, connect to the console and start doing what ever you want! You have two options to construct the class.

```
PS4RPC ps4 = new PS4RPC("X.X.X.X");
/* or with an IPAddress object */
PS4RPC ps4 = new PS4RPC(ipaddr);
```

Connecting

Once you have your *PS4RPC* object setup, you can now connect to the target PlayStation 4 console! Simply call the connect method…

```
ps4.Connect();
```

Disconnecting

When you are done, as a curtiosy, you can disconnect. This is not really needed, but will instantly kill the *rpchandler* thread.

```
ps4.Disconnect();
```

Process List

One of the first things you will want to do after connecting, is get the *ProcessList*. Through the process list pids (process id), you will be able to distinguish different proccesses and do operations on all of them. This process id should be passed into most of the functions in the *PS4RPC* object. Here are prototypes of all the *ProcessList* and *Process*...

```
/// <summary>
/// Get current process list
/// </summary>
/// <returns></returns>
public ProcessList GetProcessList();

public class Process
{
	public string name;
	public int pid;

	/// <summary>
	/// Initializes Process class
	/// </summary>
	/// <param name="name">Process name</param>
	/// <param name="pid">Process ID</param>
	/// <returns></returns>
	public Process(string name, int pid);
}

public class ProcessList
{
	public Process[] processes;

	/// <summary>
	/// Initializes ProcessList class
	/// </summary>
	/// <param name="number">Number of processes</param>
	/// <param name="names">Process names</param>
	/// <param name="pids">Process IDs</param>
	/// <returns></returns>
	public ProcessList(int number, string[] names, int[] pids);

	/// <summary>
	/// Finds a process based off name
	/// </summary>
	/// <param name="name">Process name</param>
	/// <param name="contains">Condition to check if process name contains
name</param>
	/// <returns></returns>
	public Process FindProcess(string name, bool contains = false);
}

// example
ProcessList list = ps4.GetProcessList();
Process eboot = list.FindProcess("eboot.bin");
Console.WriteLine("eboot pid: " + eboot.pid);
```

Process Info

After knowing your processes pid, you will want to learn more about the memory map. This includes all mapped virtual addresses that belong to your process. Includes the names of each mapping, start, end, and protection. This will help you resolve addresses for the reading, writing, and calling functions.

```
/// <summary>
/// Get process information (memory map)
/// </summary>
/// <param name="pid">Process ID</param>
/// <returns></returns>
public ProcessInfo GetProcessInfo(int pid);

public class MemoryEntry
{
	public string name;
	public ulong start;
	public ulong end;
	public ulong offset;
	public uint prot;
}

public class ProcessInfo
{
	public int pid;
	public MemoryEntry[] entries;

	/// <summary>
	/// Initializes ProcessInfo class with memory entries and process ID
	/// </summary>
	/// <param name="pid">Process ID</param>
	/// <param name="entries">Process memory entries</param>
	/// <returns></returns>
	public ProcessInfo(int pid, MemoryEntry[] entries);

	/// <summary>
	/// Finds a virtual memory entry based off name
	/// </summary>
	/// <param name="name">Virtual memory entry name</param>
	/// <param name="contains">Condition to check if process name contains name</param>
	/// <returns></returns>
	public MemoryEntry FindEntry(string name, bool contains = false);

	/// <summary>
	/// Finds a virtual memory entry based off size
	/// </summary>
	/// <param name="size">Virtual memory entry size</param>
	/// <returns></returns>
	public MemoryEntry FindEntry(ulong size);
}

// example
ProcessList list = ps4.GetProcessList();
Process eboot = list.FindProcess("eboot.bin");
ProcessInfo pi = ps4.GetProcessInfo(eboot.pid);
for(int i = 0; i < pi.entries.Length; i++)
{
	MemoryEntry me = pi.entries[i];
	if(me.name.Contains("executable") && me.prot == 5)
	{
		Console.WriteLine("executable base " + me.start.ToString("X"));
		break;
	}
}
```

Reading Memory

Simple functions, should be able to understand the prototypes.

```
/// <summary>
/// Read memory
/// </summary>
/// <param name="pid">Process ID</param>
/// <param name="address">Memory address</param>
/// <param name="length">Data length</param>
/// <returns></returns>
```

```
public byte[] ReadMemory(int pid, ulong address, int length);

public Byte ReadByte(int pid, ulong address);
public Char ReadChar(int pid, ulong address);
public Int16 ReadInt16(int pid, ulong address);
public UInt16 ReadUInt16(int pid, ulong address);
public Int32 ReadInt32(int pid, ulong address);
public UInt32 ReadUInt32(int pid, ulong address);
public Int64 ReadInt64(int pid, ulong address);
public UInt64 ReadUInt64(int pid, ulong address);
```

Writing Memory

Similar to reading memory but now you can write it!

```
/// <summary>
/// Write memory
/// </summary>
/// <param name="pid">Process ID</param>
/// <param name="address">Memory address</param>
/// <param name="data">Data</param>
public void WriteMemory(int pid, ulong address, byte[] data);

public void WriteByte(int pid, ulong address, Byte value);
public void WriteChar(int pid, ulong address, Char value);
public void WriteInt16(int pid, ulong address, Int16 value);
public void WriteUInt16(int pid, ulong address, UInt16 value);
public void WriteInt32(int pid, ulong address, Int32 value);
public void WriteUInt32(int pid, ulong address, UInt32 value);
public void WriteInt64(int pid, ulong address, Int64 value);
public void WriteUInt64(int pid, ulong address, UInt64 value);
```

Calling functions

The first step to call functions, is to set up the rpc calling stub. This piece of code is injected into the running process, and lies there sleeping until a call is requested. You will have to save this stub address and pass it the call routine when you want to make a call. (I may change this when I decide not to be lazy) Setup said stub like so…

```
ProcessList list = ps4.GetProcessList();
Process eboot = list.FindProcess("eboot.bin");
ulong stub = ps4.InstallRPC(eboot.pid);
```

Once the stub is installed, you can start calling functions. Note that if you specify types that would be pointers to data in C, such as a byte array or string, this will not work. You must pass in the address to this data as an argument. Function calling is the key to making cool tools to mess around with your console. Here is an example of calling the getpid and time functions…

```
int sys_getpid = (int)ps4.Call(p.pid, stub, vme.start + 0xE0);
Console.WriteLine("sys_getpid: " + sys_getpid);
int time = (int)ps4.Call(p.pid, stub, vme.start + 0x4430, 0);
Console.WriteLine("time: " + time);
```

Example

```
using System;
using System.Text;
using System.IO;

using librpc;

namespace Example
{
    class Program
    {
        public static string HexDump(byte[] bytes, int bytesPerLine = 16)
        {
            /* … */
        }

        static void Main(string[] args)
        {
            PS4RPC ps4 = new PS4RPC("192.168.1.107");
            ps4.Connect();

            ProcessList pl = ps4.GetProcessList();

            foreach(Process p in pl.processes)
            {
                Console.WriteLine(p.name);
            }

            Process p = pl.FindProcess("SceShellCore");

            ProcessInfo pi = ps4.GetProcessInfo(p.pid);
            ulong executable = 0;
            for(int i = 0; i < pi.entries.Length; i++)
            {
                MemoryEntry me = pi.entries[i];
                if(me.prot == 5)
                {
                    Console.WriteLine("executable base " + me.start.ToString("X"));
                    executable = me.start;
                    break;
                }
            }

            byte[] b = ps4.ReadMemory(p.pid, executable, 256);
            Console.Write(HexDump(b));

            ulong stub = ps4.InstallRPC(p.pid);

            ProcessInfo pi = ps4.GetProcessInfo(p.pid);
            MemoryEntry vme = pi.FindEntry("libSceLibcInternal.sprx", true);
            // dissasemble libSceLibcInternal.sprx to get these offsets (4.05)
            int sys_getpid = (int)ps4.Call(p.pid, stub, vme.start + 0xE0);
            Console.WriteLine("sys_getpid: " + sys_getpid);

            int time = (int)ps4.Call(p.pid, stub, vme.start + 0x4430, 0);
            Console.WriteLine("time: " + time);

            ps4.Disconnect();

            Console.ReadKey();
        }
    }
}
```