Windows CryptoAPI (<bcrypt.h>)

30 NOV – 4 DEC 20

Tasking

I was tasked with getting exposure to the Windows CryptoAPI.

bcrypter

After spending some time looking through MS documentation on bcrypt.h, I snagged their example implementation and moved some things around. After quickly getting lost in the different variables I had to supply to each function in turn, I created a struct which became a wrapper class for the functions. It's pretty clear how to interact with the API, and this gave me a lot of experience with C++ and its unique (relative to C) features (e.g., templates). bcrypter adds a significant amount of flexibility to interaction with the CryptoAPI. BCrypter objects offer the following:

- **Optionally** reduce the interface to only bare essentials
  - No impact on control granularity
- Store & track all values made available by earlier calls to the instance
- Dynamically infer/identify/interpret/cast many arguments and argument characteristics (e.g., type and length)
  - Pass cleanly, properly formatted to native function
- Negate required argument-supplied output variables
  - Store outputs in self for user to optionally retrieve
  - Default values

For example, calling BCryptGenerateSymmetricKey directly

```
BCryptGenerateSymmetricKey(hAlg, &hKey, pbKeyObject, cbKeyObject,
(PBYTE)AESkey, sizeof(AESkey), 0);
```

becomes (using BCrypter instance b)

```
b.GenerateSymmetricKey(AESkey);
```

TLDR

1. Got familiar with Windows CryptoAPI
2. Developed object to streamline its use
3. Could it have always been this easy? Wait, it ***is*** this easy outside of WINAPI?

Future Plans

- Finish supporting all functions in the MS example as a demo
- Extend to support NCrypt* (et al.) functions
- Pair with hooking project to steal keys