Blackbox design

# Purpose

The purpose of the Blackbox is to prevent a potential hacker from reverse engineering the entire CSS environment. The Blackbox will be made up of a single executable file, downloaded upon each login attempt. There will be subtle differences between each version downloaded; to allow a server side check that can validate that the correct code was executed. With a "trusted" module on the client's computer, we can properly validate other parts of the CSS. This will be accomplished by allowing trust to be passed between the components. If the server trusts the Blackbox and the Blackbox trusts the client, the server can also trust the client.

# Requirements

Two main requirements must be met in order to keep this system successful. They are:

* Validation of the Launch executable
* Providing the encryption key for Launch to communicate with the server

In order to trust hardware results, login credentials, and Allegiance validation, we must first check the Launch executable. Without this first check we cannot guarantee any values obtained from Launch. A token (or key) will be used as proof of validation for Launch. If the server receives the token from the client within an alotted timespan, we will presume that the Blackbox correctly authenticated the client.

# Creation

The authentication server will be responsible for Blackbox creation. The server will have access to the code, swapping in three blocks of code before it compiles. The resulting binary is transmitted to the client.

The three blocks of code are one method's logic, a randomly generated seed, and a public encryption key (one of a key pair). The method's logic should be picked from a group, premade by CSS' authors. When this method is run with the same seed and assembly hash, it will produce the same string.

To derive the key on the server, run the "random" method on the chosen seed, and the known hash of the correct launch.exe. This key, along with the public encryption key is stored in the database along with the reference to the BB on the server. To derive the key on the client, the random method will retrieve the Hash from its calling assembly, and along with the seed stored in the BB, derive the same key.

The method logic, seed, and public key must change upon each compile.

# Launch Validation

With .NET, we are able to accurately identify the executable that loaded the Blackbox module. With this information, we are able to get a hash of Launch. This hash is used in combination with a seed and the "random" method to obtain the token/key from the Blackbox. If the token matches what the server expects, then the hash of Launch is correct. More on this later.

# Key Transport

The key is derived from a random method, a hash, and a seed, so it is not stored on the BB and can only be retrieved by decompiling the blackbox, replacing the hash value and then recompiling and running it again. All this is done to ensure that the BB never has to do any *decryption*, which can cause errors on the client. If a hacker is attempting to bypass the security, and he calls the BB, the BB will always return a result regardless of what information is input, the information is then checked for validity on the server, if the key in the decrypted machine information matches what is stored in the database, then we can trust that launch.exe is valid.

In order to obtain the key, the Blackbox does the following:

* Retrieves machine information (authinfo) from the calling assembly as an input parameter.
* Derive a key by calling the random method (whose logic varies with each build) in the BB (X). Internally the random method generates a hash of the calling assembly (launch.exe) and uses that along with the seed to derive the key.
* Append the key to the authinfo.
* Encrypt the authinfo + key with the public key stored in the BB.
* Return encrypted data back to calling assembly.

