























Daniel Hoynoski

Golf King Client Version: v0.34f

Turn In (extended): Friday, May 4th, 2018

Game Crack 3 - The Encryption Wall Falls!

For the last big progress I made during the semester on getting some game-play back, I turned my attention to figuring out what was in all the encrypted files. Take for example these files:

 Input	5/3/2018 11:26 PM
 Camera.enc	2/21/2005 8:52 PM
 Chat.enc	3/24/2005 10:11 AM
 ClickGauge.enc	10/12/2005 3:05 PM
 CombinationPlay.enc	2/23/2005 4:16 PM
 DebugDirectStart.enc	2/21/2005 8:52 PM
 G3dX.enc	2/21/2005 8:52 PM
 Game.enc	7/13/2005 11:12 AM
 GRes.enc	2/21/2005 8:52 PM
 Grid.enc	2/21/2005 8:52 PM
 Hq.enc	9/6/2005 12:00 PM
 Intro.enc	10/5/2005 12:19 PM
 Local.enc	11/15/2005 2:24 PM
 Log.enc	2/24/2006 3:36 PM
 MiniMission.enc	2/21/2005 8:52 PM
 Mode.enc	2/21/2005 8:52 PM
 Nation.enc	6/1/2005 5:20 PM
 PlayEnv.enc	1/10/2006 5:40 PM
 Server.enc	3/8/2006 6:27 PM
 Sound.enc	3/31/2005 7:24 AM
 TestAvatar.enc	2/21/2005 8:52 PM
 Video.enc	6/29/2005 12:16 PM

Files such as “server.enc” and “local.enc” looked very promising for offering some more options and information on this game and how it worked. However, these files are hopelessly encrypted with RSA based encryption (discussed below).

Obviously the game has to decrypt the files at some point of execution, so I figured I would analyze the binary to see if I could find any useful information:

0050B07B	CALL	EBX	ADVAPI32.CryptAcquireContextA
0050B09B	CALL	EBX	ADVAPI32.CryptAcquireContextA
00B2265D	CALL	DWORD PTR	[<&ADVAPI32.CryptAcquireContextA
00B228FB	CALL	DWORD PTR	[<&ADVAPI32.CryptAcquireContextA
00B22961	CALL	DWORD PTR	[<&ADVAPI32.CryptAcquireContextA
00B22A3E	CALL	DWORD PTR	[<&ADVAPI32.CryptAcquireContextA
00B22C6C	CALL	DWORD PTR	[<&ADVAPI32.CryptAcquireContextA
00B22F8E	CALL	DWORD PTR	[<&ADVAPI32.CryptAcquireContextA
0050A6E3	CALL	DWORD PTR	[<&ADVAPI32.CryptCreateHash
0050B0C2	CALL	EBX	ADVAPI32.CryptCreateHash
0050B11B	CALL	EBX	ADVAPI32.CryptCreateHash
00B229C4	CALL	DWORD PTR	[<&ADVAPI32.CryptCreateHash
00B22CA6	CALL	DWORD PTR	[<&ADVAPI32.CryptCreateHash
00B22FC8	CALL	DWORD PTR	[<&ADVAPI32.CryptCreateHash
0050A679	CALL	DWORD PTR	[<&ADVAPI32.CryptDecrypt
0050B635	CALL	DWORD PTR	[<&ADVAPI32.CryptDecrypt
00B22D46	CALL	DWORD PTR	[<&ADVAPI32.CryptDecrypt
00B2306E	CALL	DWORD PTR	[<&ADVAPI32.CryptDecrypt
0050B0F5	CALL	DWORD PTR	[<&ADVAPI32.CryptDeriveKey
00B22D14	CALL	DWORD PTR	[<&ADVAPI32.CryptDeriveKey
00B2303A	CALL	DWORD PTR	[<&ADVAPI32.CryptDeriveKey
0050A6D0	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyHash
0050B102	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyHash
0050B159	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyHash
0050B1C9	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyHash
0050B1F9	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyHash
00B226A2	CALL	EBX	ADVAPI32.CryptDestroyHash
00B229B0	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyHash
00B22C92	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyHash
00B22FB4	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyHash
0050B14B	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyKey
0050B1BB	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyKey
0050B1EB	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyKey
00B226B3	CALL	EDI	ADVAPI32.CryptDestroyKey
00B22921	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyKey
00B22CFE	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyKey
00B2301F	CALL	DWORD PTR	[<&ADVAPI32.CryptDestroyKey
0050A63C	CALL	DWORD PTR	[<&ADVAPI32.CryptEncrypt
0050B556	CALL	DWORD PTR	[<&ADVAPI32.CryptEncrypt
0050A6C2	CALL	DWORD PTR	[<&ADVAPI32.CryptGetHashParam
00B237D0	CALL	DWORD PTR	[<&ADVAPI32.CryptGetHashParam
0050B0D7	CALL	DWORD PTR	[<&ADVAPI32.CryptHashData
00B229F0	CALL	DWORD PTR	[<&ADVAPI32.CryptHashData
00B22CD8	CALL	DWORD PTR	[<&ADVAPI32.CryptHashData
00B22FF9	CALL	DWORD PTR	[<&ADVAPI32.CryptHashData
00B22942	CALL	ESI	ADVAPI32.CryptImportKey
0050B168	CALL	DWORD PTR	[<&ADVAPI32.CryptReleaseContext
0050B1D8	CALL	DWORD PTR	[<&ADVAPI32.CryptReleaseContext
0050B208	CALL	DWORD PTR	[<&ADVAPI32.CryptReleaseContext
00B226D4	CALL	DWORD PTR	[<&ADVAPI32.CryptReleaseContext
00B22A1B	CALL	DWORD PTR	[<&ADVAPI32.CryptVerifySignatureA
00B22A69	CALL	DWORD PTR	[<&ADVAPI32.CryptVerifySignatureA

While looking at the intermodular calls the game was making, I noticed some very useful calls! In particular, I noticed two: CryptDecrypt and CryptEncrypt. My thoughts at this moment figured the game could also re-encrypt the files too (which would be a plus for testing).

These cryptographic functions are part of Microsoft's CryptoAPI, a very well documented and easily assessable API! My next task was to create a C program that reproduced the steps that the game did in its binaries. It was NOT easy and after LITERAL days of reversing and researching the cryptoAPI, I produced 414 lines of code that mimicked this encryption/decryption process (see crypto.c). Because you cannot just use decrypt/encrypt functionalities right away, an environment has to be setup correctly.

I set breakpoints at all the CryptoAPI function calls and then ran the game. I noticed the order in which the functions were being called in (i.e. for setting up the environment) and the parameters it was using. I even got the secret encryption key it was holding out on me:

0xEA, 0xCA, 0x31, 0x01

I was quite surprised to find that this key could be used to re-encrypt the files and after looking at the encrypted files in the game, the local.enc file allowed me to setup a local game (i.e. against the computer). The file was obviously kept for internal debugging for the programmers.

And to my dismay, I finally got game play:



In the above screenshot, I am using the English client I got my hands on, but the same code works for the Korean client too.