[ 유니티 취약한 라이브러리 및 함수 ]

1) PlayerPrefs

1. URL : <https://docs.unity3d.com/kr/530/ScriptReference/PlayerPrefs.html>

2. functions



3. 취약한 이유

- 물리적인 저장 장치에 xml 파일 형태로 저장된다.

On Android data is stored (persisted) on the device. The data is saved in SharerPreferences. C#/JavaScript, Android Java and Native code can all access the PlayerPrefs data. The PlayerPrefs data is physically stored in /data/data/pkg-name/shared\_prefs/pkg-name.xml.

- Unity PlayerPrefs 특성상 key-value 형태로 로컬에 데이터를 저장하는데, 암호화가 되어 있지 않다.

해결 URL : <https://www.ikpil.com/1342>

4. 저장 위치

- PC : 레지스트리에 등록됨.

- 안드로이드 : HKEY\_CURRENT\_USER/Software/ 유니티의 플레이어 셋팅에 보면 적혀져 있는 회사 이름 아래로 들어간다.

- WEB : %APPDATA%\Unity\WebPlayerPrefs 이하로 들어간다.

[company name]\[product name] 폴더

- IOS

1] 어플리케이션 : ~/Library/Preferences

2] 웹플레이어 : ~/Library/Preferences/Unity/WebPlayerPrefs

unity.[company name].[product name].plist 파일

5. 어떻게 사용자에게 알려줄 것인가?

- 암호화되지 않은 키를 저장한다는 걸 알려주고, 해싱된 키를 저장하도록 추천해주도록 한다. 암호화된 PlayerPrefs 사용 권유.

- PlayerPrefs 사용 경고 메시지 -> 다른 안전한 라이브러리 사용 권유

2) BinaryFormatter

- List를 이용해서, 클래스 정보를 담아, 테이블처럼 사용하는 방법.

- 정보저장 방법 : 클래스를 바이트 배열로 변환 -> 변환된 배열을 다시 문자열 값으로 변환 -> 변환된 문자열 값을 PlayerPrefs에 저장

1. 라이브러리 : using System.Runtime.Serialization.Formatters.Binary;

2. 사용 방법

void SaveScore() {

var b = new BinaryFormatter (); // Binary formatter 객체 선언

var m = new MemoryStream (); // MemoryStream에 있는 데이터를 저장하기 위해 객체 선언

b.Serialize (m, Score); // 스코어를 저장

// Serialize 함수를 통해서 MemoryStream에 저장된 값을 String으로 변환

PlayerPrefs.SetString("Score", Convert.ToBase64String (m.GetBuffer ()));

//PlayerPrefs에 저장 }

void Start() {

var data = PlayerPrefs.GetString ("Score "); // 데이터를 가져옴.

if (!string.IsNullOrEmpty (data)) { // 데이터 존재 여부 확인

var b = new BinaryFormatter(); // 바이너리 데이터를 b 에다가

var m = new MemoryStream(Convert.FromBase64String(data)); // 데이터에 있는 String을 MemoryStream에 올리기.

Score = (List<ScoreEntry>)b.Deserialize(m); // Deserialize로 메모리에 올라간 데이터 로드. } }

3. 문제점

- 메모리에 데이터가 로드됨 -> 실시간 메모리 덤핑에 약함

- PlayerPrefs 사용

3) RPC

1. 사용 목적 : 유니티 오브젝트 저장하기.

2. 사용 방법

- 어떠한 클래스라도 문자열로 변환 할 수 있음.

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

public class RPCTest : MonoBehaviour {

public class ScoreEntry {

public string name; //플레이어 이름

public int score; // 스코어 }

void ReceiveHighScores(byte[] highScores) {

var b = new BinaryFormatter();

var m = new MemoryStream (highScores);

var ortherPlayerScores = (List<ScoreEntry>)b.Deserialize (m); }

void UpdateScores(byte[] highScores) {

var b = new BinaryFormatter();

var m = new MemoryStream();

b.Serialize (m, highScores);

networkView.RPC ("ReceiveHighScores", RPCMode.Others, m.GetBuffer ()); }

//RPC 역시 byte[]를 보낼 수 있다. 이 값을 BinaryFormatter를 이용해서 원하는 값으로 변환 할 수 있도록 도와준다. }

3. 문제점

- 메모리에 데이터가 로드됨

- Spoofing에 취약

- 암호화되지 않은 데이터

4) Web으로 전달

1. 사용 방법

- WWWForm을 이용해서 웹 서버에 데이터를 전달하는 방법

- Convert.ToBase64String()을 이용해서 바이너리 데이터를 문자열로 변환하는 방법 또는, 사용하는 서버에 따라 여러가지 방법이 나눠진다.

2. 문제점

- 암호화되지 않은 데이터

5) 파일로 저장

1. 설명 : 유니티가 제공하는 Application.persistentDataPath를 이용해서 파일을 저장할 위치를 정하고, MemoryStream 대신 FileStream을 사용.

2. 사용 방법

using UnityEngine;

using System.Collections;

using System.Runtime.Serialization.Formatters.Binary; //Formatters 쓸려고..

using System.IO; //File 쓰기 위해

using System.Collections.Generic; //List쓰려고

public class FileStreamTest : MonoBehaviour {

public class ScoreEntry {

public string name; //플레이어 이름

public int score; // 스코어 }

public List<ScoreEntry> highScores = new List<ScoreEntry>(); //하이스코어 테이블

void SaveScores() {

var b = new BinaryFormatter (); //BinartFormatter를 받아옴

var f = File.Create (Application.persistentDataPath + "/highscores.dat"); //파일을 생성.

b.Serialize (f, highScores); // 스코어를 저장.

f.Close (); }

void Start() {

if (File.Exists (Application.persistentDataPath + "/highscores.dat")) { //비어있지 않으면 로드!

var b = new BinaryFormatter(); //바이너리 포맷터

var f = File.Open(Application.persistentDataPath + "/highscores.dat",FileMode.Open); // 파일 열기.

highScores = (List<ScoreEntry>)b.Deserialize(f); //스코어를 로드. 디 시리얼라이즈.

f.Close(); //파일 닫기. } } }

- 앞 부분의 다른 코드들과 합쳐 본 코드

using UnityEngine;

using System.Collections;

using System.Runtime.Serialization.Formatters.Binary; //Formatters 쓸려고..

using System.Runtime.Serialization;

using System.IO; //File 쓰기 위해

using System.Collections.Generic; //List쓰려고

using System; // [Serializable] 사용을 위해.

public class FileStreamTest : MonoBehaviour {

[Serializable]

public class ScoreEntry {

public string name; //플레이어 이름

public int score; // 스코어 }

string currentPlayerName;

int PlayScore;

public List<ScoreEntry> highScores = new List<ScoreEntry>(); //하이스코어 테이블

void Start() {

currentPlayerName = "TestPlayer";

PlayScore = 52346;

AddData ();

currentPlayerName = "TestPlayer2";

PlayScore = 11152;

AddData ();

SaveScores ();

OnGUI (); }

void SaveScores() {

var b = new BinaryFormatter (); //BinartFormatter를 받아옴

//var f = File.Create (Application.persistentDataPath + "/highscores.dat"); //파일을 생성.

Stream f = new FileStream (Application.persistentDataPath +"/highscores.dat", FileMode.Create, FileAccess.Write, FileShare.None);

b.Serialize (f, highScores); // 스코어를 저장.

f.Close (); }

void LoadScore() {

if (File.Exists (Application.persistentDataPath + "/highscores.dat")) { //비어있지 않으면 로드!

var b = new BinaryFormatter(); //바이너리 포맷터

var f = File.Open(Application.persistentDataPath + "/highscores.dat",FileMode.Open); // 파일 열기.

highScores = (List<ScoreEntry>)b.Deserialize(f); //스코어를 로드. 디 시리얼라이즈.

f.Close(); //파일 닫기. } }

void AddData() {

highScores.Add (new ScoreEntry{name = currentPlayerName, score = PlayScore}); }

void OnGUI() {

foreach (var score in highScores) {

GUILayout.Label(string.Format ("{0} : {1:#,0}",score.name,score.score)); } } }

3. 파일 위치

- C:\Users\유저이름\AppData\LocalLow\유니티 프로젝트 상의 회사 이름\프로젝트 이름\

4. 문제점

- 암호화 문제