이봐!

v. 2021.5.0

[버전 기록]

Unity용 종합 치트 방지 솔루션인 ACTk(Anti-Cheat Toolkit)를 구입해 주셔서 대단히 감사합니다 !

부인 성명:

이 플러그인에 사용된 치트 방지 기술은 100% 안전하고 깨지지 않는 척하지 않습니다(클라이언트 측에서는 불가능). 그러나 대부분의 치트가 앱을 해킹하려는 시도를 막아야 합니다.

명심하세요: 의욕이 있고 숙련된 해커는 무엇이든 깨뜨릴 수 있습니다!

스크립팅 측에서 플러그인 API 사용에 대해 질문이 있는 경우: codestage.net/uas_files/actk/api

내용물

설치 및 설정

새 버전을 프로젝트로 가져오기 전에 새 버전에서 가능한 파일 및 폴더 구조 변경으로 인한 문제를 방지하기 위해 이전 버전을 완전히 제거하는 것을 고려하십시오.

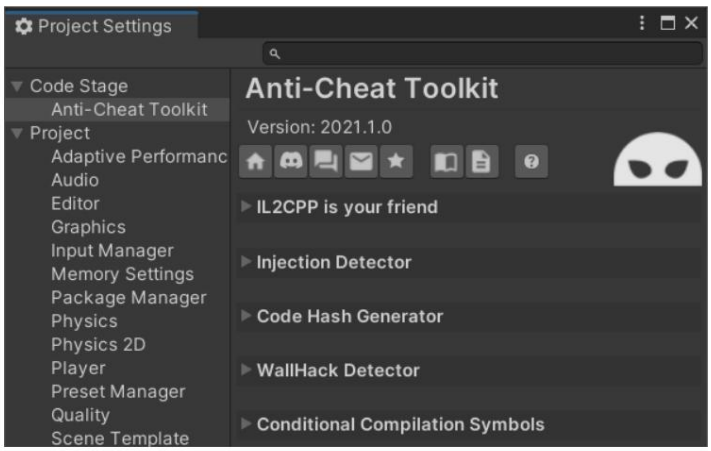
플러그인을 프로젝트로 가져온 후 추가 설정 및 제어를 위한 새 메뉴를 찾을 수 있습니다. • Tools > Code Stage > Anti-Cheat Toolkit • GameObject > Create Other > Code Stage > Anti-Cheat Toolkit

플러그인 기능, 모범 사례 및 힌트에 대해 자세히 알아보려면 이 문서를 읽으십시오.

설정 창

편집기 기본 설정 섹션을 사용하여 디버깅 및 호환성을 위해 감지기 및 조건부 컴파일 기호를 구성합니다. 모든 설정은 프로젝트의 "ProjectSettings/ACTkSettings.asset" 에 저장됩니다 .

ACTk 설정을 열려면 도구 > Code Stage > Anti-Cheat Toolkit > 설정 메뉴 명령을 사용하십시오.



여기에서 다음과 같은 항목을 볼 수 있습니다. • 프로젝트에서 사용 중인 ACTk 버전 • 유용한 링크 및 바로가기(홈페이지, Discord, 포럼, 지원, 리뷰, 이 매뉴얼, API 참조, 정보) • IL2CPP는 친구 섹션 – IL2CPP에 대해 설명 합니다 . 치트 방지 관점에서 Mono에 비해 이점이 있으며 쉽게 현재 플랫폼에서 지원하는 경우 IL2CPP로 전환하십시오. • 인젝션 감지기 설정 섹션 – 관리형 DLL 인젝션 감지 장에서 자세한 내용을 참조하십시오. • 코드 해시 생성기 – 코드 무결성 검사 장에서 자세한 내용을 참조하십시오. • WallHack 감지기 설정 섹션 - Wallhack 감지 장에서 자세한 내용을 참조하십시오. • 다양한 디버그 및 호환성 조정, 로그 및 전환을 허용하는 조건부 컴파일 기호 섹션

그런.

플러그인 기능 심층 분석

메모리 부정 행위 방지 [동영상 자습서]

중요: 일반 유형을 가려진 유형으로 교체하는 동안 주의하십시오. 인스펙터 값이 재설정됩니다!

이것은 다른 플랫폼에서 가장 인기 있는 부정 행위 방법입니다. 사람들은 메모리에서 변수를 검색하고 값을 변경하기 위해 특별한 도구를 사용합니다. 일반적으로 돈, 건강, 점수 등입니다. 몇 가지 예: Cheat Engine, ArtMoney(PC), Game CIH, Game Guardian(Android). 이러한 플랫폼과 다른 플랫폼을 위한 다른 도구도 많이 있습니다.

메모리 치트 방지 기술을 활용하려면 ObscuredInt, ObscuredFloat, ObscuredString 등 일반 유형 대신 가려진 유형을 사용하면 됩니다 (모든 기본 유형 + 일부 Unity 특정 유형 포함) ...

이러한 유형은 다음과 같이 일반 내장 유형 대신(및 함께) 사용될 수 있습니다.

```
// 이 줄을 .cs 파일의 시작 부분에 바로 배치하세요! CodeStage.AntiCheat.ObscuredTypes 사
용 ;

int totalCoins = 500;
ObscuredInt collectCoins = 100; int coinsLeft =
totalCoins - collectCoins;

// 출력: "수집된 동전: 100, 왼쪽: 400"
Debug.Log(" 수집된 코인: " + collectCoins + ", 왼쪽: " + coinsLeft);
```

모든 int <-> ObscuredInt 캐스트는 암시적으로 수행됩니다. 다른 가려진 유형과 동일합니다!

일반 int 와 ObscuredInt 사이에는 한 가지 큰 차이점이 있습니다 . 치터는 일반 int 에 대해 말할 수 없는 ObscuredInt 로 메모리에 저장된 값을 쉽게 찾고 변경할 수 없습니다 !

음, 실제로 사기꾼이 찾고 있는 것을 찾고 부정 행위 시도를 포착하도록 허용할 수 있습니다. 이러한 경우 실제 가려진 값은 여전히 안전합니다. 플러그인을 사용하면 사기꾼이 원하는 경우 가짜 암호화되지 않은 변수를 찾고 변경할 수 있습니다. ;)

이러한 부정 행위 감지를 위해 아래에 설명된 ObscuredCheatingDetector를 사용합니다 .

가려진 유형에 대한 추가 정보

플러그인과 함께 제공되는 "Examples/API Examples" 장면에서 모호한 유형 사용 예를 찾고 직접 속이려고 시도할 수도 있습니다 .

가려진 유형 전문가 팁:

- 모든 단순한 Obscured 유형에는 가려진 인스턴스의 암호화된 값으로 작업할 수 있도록 공용 정적 메서드 GetEnctypted() 및 SetEncrypted() 가 있습니다. 일부 사용자 지정 저장 엔진을 사용하려는 경우 도움이 될 수 있습니다.
- 경우에 따라 사기꾼은 소위 "알 수 없는 값" 검색을 사용하여 가려진 변수를 찾을 수 있습니다. 그들이 할 것이라는 사실에도 불구하고 암호화된 값을 찾으면 ObscuredCheatingDetector 에서 모든 부정 행위(치트 엔진에서 고정 또는 메모리 편집기 변경)를 감지할 수 있으며 여전히 찾을 수 있습니다. 이를 완전히 방지하려면 모든 기본 가려진 유형에서 특수 메서드인 RandomizeCryptoKey()를 사용할 수 있습니다 . 변수가 변경되지 않는 순간에 사용하여 원래 값을 그대로 유지하는 암호화된 표현을 변경합니다. Cheater는 변경되지 않은 값을 검색하지만 실제로는 값이 변경되어 변수 찾기를 방지합니다.
- GenerateKey() API를 사용하여 Encrypt(value, key) 메서드 에 대한 임의의 암호화 키를 생성할 수 있습니다 . 나중에 값을 Decrypt() 하기 위해 해당 키를 저장하는 것을 잊지 마십시오 .

중요한:

- 현재 이러한 유형 은 인스펙터 에 노출 될 수 있습니다 . _ _ _ _ 일반 유형을 가려진 유형으로 교체하는 동안 주의하십시오. 인스펙터 값이 재설정됩니다!

• 여전히 매우 빠르고 가볍지만 모호한 유형은 일반적으로
일반 것. 업데이트, 루프, 거대한 배열 등에서 가려진 변수를 멀리하고 특히 모바일 프로젝트에서 작업할 때 프로파일러를 주시하십시오.

• LINQ 및 XmlSerializer 는 현재 지원되지 않습니다.

• 이진 직렬화는 기본적으로 지원됩니다.

• 다음을 통해 JSON 직렬화도 가능합니다.

• ISerializable 구현을 사용하는 .NET용 Newtonsoft Json (예제) 또는 변환기 (예: by 맥모리)

• Unity 고유의 JsonUtility ISerializationCallbackReceiver 의 도움으로 (예).

• 일반적으로 고급 작업을 수행하고 이를 캐스팅하려면 가려진 변수를 일반 변수로 캐스팅하는 것이 좋습니다.
잘 작동하는지 확인하기 위해 다시 돌아갑니다.

파일 부정 행위 방지 [동영상 자습서]

가려진 파일

파일은 저장된 게임 및 기타 민감한 정보 저장을 위한 일반적인 장소입니다.
그러나 적절한 보호가 없으면 파일은 다음과 같은 위협에 취약합니다.

• 바이너리가 아닌 파일(게임 변수, 자격 증명, 게임 내 개발자 치트 등)에서 민감한 데이터 식별. • 데이터 수정(100에서 999999 등으로 돈을 속이기 위해). • 다른 사기꾼과 속이거나 수정된 파일 공유.

ObscuredFile은 다음과 같은 모든 위협을 다룹니다.

• 선택적으로 데이터를 암호화하여 호기심 많은 눈으로부터 숨길 수 있습니다.
• 변조 감지 기능이 내장되어 있어 데이터 수정을 발견할 수 있습니다(암호화된 파일과 일반 파일 모두). • 선택적으로 다른 장치에서 공유된 파일을 감지하기 위해 데이터를 장치 ID 또는 사용자 정의 ID로 잠급니다. 장치 잠금 참조
자세히 알아보기 위한 메모 .

ObscuredFile은 원시 바이트 배열만 읽고 쓸 수 있습니다. 사용자 친화적인 PlayerPrefs – alike API에 대해서는 아래의 ObscuredFilePrefs를 확인하십시오 .

설정 및 사용법은 매우 간단합니다.

```
// CodeStage.AntiCheat.Storage를 사용하여 .cs 파일의 시작 부분에 이 줄을 배치합니다 .

// 기본 설정으로 새 인스턴스 생성 var secureFile = new ObscuredFile();

// 데이터 쓰기 var
writeResult = secureFile.WriteAllBytes(abstractBytes); if (쓰기결과.성공)

    Debug.Log($"{secureFile.FilePath} 에 저장된 데이터 "); else

Debug.LogError($"데이터를 저장할 수 없습니다 : {writeResult.Error}");

// 데이터 읽기 var
readResult = secureFile.ReadAllBytes(); if (readResult.Success) //
readResult.Data 처리; 그렇지 않으
    면 (readResult.CheatingDetected)

    // 사기꾼을 처벌하고 DataFromAnotherDevice \ DataIsNotGenuine readResult 속성을 확인합니다. else Debug.LogError($" 데이터를 읽
는 동
    안 문제가 발생했습니다: {readResult.Error}");
```

위의 예에서 abstractBytes는 기본 설정으로 저장 및 로드되지만 ObscuredFileSettings API를 사용하여 파일 경로 및 이름, 암호화 설정, 장치 기능 설정 잠금 및 변조 방지를 자유롭게 구성할 수 있습니다.

ObscuredFile을 구성하고 사용하는 방법에 대한 자세한 내용은 API 설명서를 참조하십시오 . 및 사용 예: Examples\API Examples\Scripts\Runtime\UsageExamples\ObscuredFilePrefsExamples.cs

중요한:

• UnityApiResultsHolder.InitForAsyncUsage(true)를 호출합니다. 백그라운드 스레드에서 가려진 파일로 작업하기 전에 기본 스레드에서

• ObscuredFile은 추가 암호화로 인해 일반 파일 에 비해 느리게 작동합니다. 핫 경로에서 사용하지 않도록 하십시오.

가려진 파일 기본 설정

이것은 일반 API를 사용하기 매우 쉬운 정적 ObscuredFile 래퍼입니다. 다양한 유형의 환경 설정을 읽고 쓸 수 있습니다(모든 기본 C# 유형, BigInteger, byte[], DateTime 및 소수의 Unity 유형이 지원됨).

그것은 ObscuredFile이 가지고 있는 모든 동일한 기능을 가지고 있으며 치터로부터 보호되는 ObscuredFile에 환경 설정을 저장합니다.
다음은 간단한 사용 예입니다.

```
// CodeStage.AntiCheat.Storage를 사용하여 .cs 파일의 시작 부분에 이 줄을 배치합니다 .


// 기본 설정으로 초기화 ObscuredFilePrefs.Init();


// 치팅 수신 대기
ObscuredFilePrefs.NotGenuineDataDetected += OnDataCheat;
ObscuredFilePrefs.DataFromAnotherDeviceDetected += OnLockCheat;

// 기본 쓰기
ObscuredFilePrefs.Set("pref key", data);

// pref 데이터 읽기
= ObscuredFilePrefs.Get("pref key", defaultValue);

// 또는 데이터 유형을 지정하고 defaultValue 인수 생략 // data = ObscuredFilePrefs.Get<T>("pref key");


// 반복할 모든 pref 키 가져오기 var keys =
ObscuredFilePrefs.GetKeys();
```

ObscuredFile 예제와 유사하게 기본 설정이 사용되었으며 ObscuredFileSettings API 를 사용하여 필요에 따라 ObscuredFilePrefs를 구성할 수 있습니다 .

중요한:

• UnityApiResultsHolder.InitForAsyncUsage(true)를 호출합니다. 백그라운드 스레드에서 가려진 파일로 작업하기 전에 기본 스레드에서

• ObscuredFilePrefs는 추가 암호화로 인해 일반 File 또는 PlayerPrefs 에 비해 느리게 작동합니다. 핫 경로에서 사용하지 않도록 하십시오.

• ObscuredFilePrefs 에는 기본적으로 활성화되는 자동 저장 기능이 있습니다. 저장되지 않은 데이터의 손실 가능성을 방지하고
앱 종료(데스크톱) 및 포커스를 잃은 앱(모바일)에 저장합니다. 모든 권한을 갖고 싶다면 ObscuredFileSettings 에서 위험을 감수하고 비활성화할 수 있습니다 .

Player Prefs 부정 행위 방지 [비디오 자습서]

Unity 개발자는 종종 PlayerPrefs(PP) 클래스를 사용하여 소량의 민감한 데이터(돈, 게임 진행 등)를 저장하지만, 거의 노력하지 않고 찾아서 변조할 수 있습니다. 예를 들어 Windows에서 regedit를 열고 HKEY_CURRENT_USER\Software\Your Company Name\Your Game Name 으로 이동하는 것만 큼 간단합니다 !

이것이 내가 이 툴킷에 ObscuredPrefs(OP) 클래스를 포함하기로 결정한 이유입니다. 이를 통해 평소와 같이 데이터를 저장할 수 있지만 보기 및 변경으로부터 데이터를 안전하게 보호할 수 있습니다. 이는 부정 행위 방지 저장을 유지하는 데 정확히 필요한 것입니다!

다음은 간단한 예입니다.

```
// CodeStage.AntiCheat.Storage를 사용하여 .cs 파일의 시작 부분에 이 줄을 배치합니다 .


ObscuredPrefs.Set<float>("currentLifeBarObscured", 88.4f); var currentLifeBar =
ObscuredPrefs.Get<float>("currentLifeBarObscured");

// 인쇄: "Life bar: 88.4"
Debug.Log("생명 막대: " + currentLifeBar);
```

보시다시피 사용 방법에 변경된 사항은 없습니다. 모든 것이 이전의 좋은 일반 PlayerPrefs 클래스와 동일하게 작동하지만 이제 저장 내용이 부정 행위자로부터 안전합니다(음, 대부분의 부정 행위자로부터)!
추가 기능:

- NotGenuineDataDetected 이벤트를 구독할 수 있습니다 . 저장된 데이터 무결성 위반에 대해 알 수 있습니다.
  변조된 데이터를 읽자마자 한 번(전체 세션 동안) 실행됩니다.
- OP는 일반 PP 에 비해 많은 추가 데이터 유형을 지원합니다 . 모든 기본 C# 유형, BigInteger, byte[], DateTime 및 소수의 Unity 유형이 지원됩니다.
- 장치 잠금 기능이 지원됩니다. 자세한 내용은 장치 잠금 참고 사항을 참조하십시오 .

ObscuredPrefs 전문가 팁:

- PP 에서 OP 로 쉽게 마이그레이션할 수 있습니다 . 프로젝트에서 PP 항목을 OP 로 바꾸면 됩니다(단, ACTk 클래스에서 PP를 OP 로 바꾸지 않도록 주의하십시오)! OP는 처음 읽을 때 PP 로 저장된 모든 데이터를 자동으로 암호화합니다 . 원래 PP 키는 기본적으로 삭제됩니다. preservePlayerPrefs 플래그를 사용하여 보존할 수 있습니다 . 이 경우 PP 데이터는 평소와 같이 OP 로 암호화되지만 원본 키는 유지되므로 PP를 사용하여 다시 읽을 수 있습니다. API 예제 장면에서 preservePlayerPrefs가 작동하는 것을 볼 수 있습니다 .
- 일반 PP 와 OP를 혼합할 수 있습니다 (다른 키 이름을 사용해야 함!). 민감한 데이터만 저장하려면 모호한 버전을 사용하십시오. 마이그레이션하는 동안 프로젝트의 모든 PP 호출을 교체할 필요가 없으며 일반 PP는 OP 에 비해 더 빠르게 작동합니다 .
- OP를 사용하여 로컬 순위표, 플레이어 점수, 돈 등과 같은 적절한 양의 데이터를 저장하고 맵 배열, 자신의 데이터베이스 등과 같은 데이터의 막대한 부분을 저장하는 데 사용하지 마십시오. 이를 위해 ObscuredFile 또는 ObscuredFilePrefs를 사용 하십시오 .
- ArrayPrefs2 와 같이 일반 PlayerPrefs를 확장하는 몇 가지 훌륭한 기여가 있습니다 . 예를 들어. OP 는 이러한 클래스의 PP를 쉽게 교체하여 저장된 모든 데이터를 안전하게 만듭니다.

중요한:

- OP는 모든 데이터를 암호화하고 해독하기 때문에 일반 PP 에 비해 느리게 작동합니다.
  일부 추가 자원을 소비합니다. API 예제 장면의 PerformanceTests 게임 개체 에서 Performance Obscured Tests 구성 요소를 사용하여 직접 확인하십시오 .
- 내부를 적절하게 지우려면 PlayerPrefs.DeleteAll() 대신 ObscuredPrefs.DeleteAll()을 사용하여 모든 기본 설정을 제거하십시오.
  DeleteAll() 호출 후 가려진 새 기본 설정을 저장할 때 데이터 손실을 방지합니다.

ObscuredPrefs / PlayerPrefs 편집기 창

ACTk에는 유용한 편집기 창인 Prefs 편집기가 포함되어 있습니다.
Unity 에디터에서 PlayerPrefs 및 ObscuredPrefs를 편집하는 데 사용하십시오.

메뉴 명령을 통해 엽니다.
Tools > Code Stage > Anti-Cheat Toolkit > Prefs Editor as Tab (유틸리티 창 모드도 사용 가능)

빠른 둘러보기:

• "+" 버튼: 새 기본 설정 추가 - 유형 선택, 선택적으로 암호화 활성화, 키 및 값 설정, 확인 버튼 누르기 • "새로 고침" 버튼: 모든 기본 설정 다시 읽기
및 목록 업데이트 • "키 내림차순" - 정렬 유형, 기타 가능 값: Key
Ascending, Type, Obscurance • 검색 필드 - 입력을 시작하면 지정된 텍스트가 포함된 이름을 가진 레코드를 필터링합니다.
• 페이징이 있는 ObscuredPrefs 및 PlayerPrefs 목록(페이지당 50개의 레코드 표시) • 레코드 분류:

• "X" 버튼 - 환경 설정에서 레코드 삭제 • "S" 버튼 - 환경
설정 저장소에 변경 사항 저장 • "E" / "D" 버튼 - 환경 설정 암호화/해
독(가려진 환경 설정은 녹색으로 표시됨) • "…" 버튼 - 클립보드에 pref 복사와 같은 추가 작업 • 키 및 값 필드는
pref의 키 및/또는 값을 변경할 수 있습니다(pref가 편집 가능한 유형인 경우) •
유형 레이블은 pref 유형을 표시합니다. • 그룹 작업을 위한 하단의 일부 버튼(자명한 설명 포함) 이름) • 덮어쓰기 확인 기능이 있습니
다. • 엄청난 양의 기본 설정(1k 이상)으로
작업하는 경우 기본 설정 분석 진행률 표시줄이 있습니다. • Win, Mac, Linux에서 작동합니다.

Prefs Editor는 Unity가 자체 요구 사항(예: UnitySelectMonitor, UnityGraphicsQuality 등)에 사용하는 몇 가지 표준 기본 설정을 무시합니다.

Windows에서는 Editor와 Standalone Player에 저장된 기본 설정이 서로 다른 위치에 배치되므로 Unity 에디터 외부에 저장된 기본 설정에 Prefs Editor를 사용할 수 없습니다.

일반 장치 잠금 기능 참고 사항

ObscuredPrefs, ObscuredFile 또는 ObscuredFilePrefs 와 함께 장치 잠금 기능을 사용하는 경우 다음을 고려하십시오.

• DeviceLockSettings 속성을 사용하면 저장된 데이터를 현재 장치에 잠글 수 있습니다. 이것은 저장 게임이 한 장치에서 다른 장치로 이동하는 것을 방지하는 데 도움이
될 수 있습니다(예: 100% 게임 진행률 또는 구매한 게임 내 상품으로 저장).

경고: iOS에서는 위험을 감수하고 사용하세요! iOS에서 영구적인 장치 ID를 얻을 수 있는 신뢰할 수 있는 방법은 없습니다. 따라서 사용을 피하거나
DeviceIdHolder.DeviceId 와 함께 사용하여 자신의 장치 ID(예: 사용자 이메일)를 설정하십시오.
세 가지 다른 수준의 데이터 잠금 엄격성을 지정할 수 있습니다. 없음: 잠긴 데이터와
잠금 해제된 데이터를 모두 읽을 수 있으며 저장된 데이터는 잠금 해제된 상태로 유지됩니다.
소프트: 여전히 잠긴 데이터와 잠금 해제된 데이터를 모두 읽을 수 있지만 저장된 모든 데이터는 현재 장치에 잠깁니다.
Strict: 현재 장치 데이터에 잠긴 읽기만 가능합니다. 저장된 모든 데이터는 현재 장치에 고정됩니다.
API 문서 의 DeviceLockSettings 설명 참조 더 많은 정보를 위해서.
Android 사용자: 이 기능을 사용하지 않는 경우 아래 문제 해결 섹션을 참조하십시오.

• DataFromAnotherDeviceDetected 이벤트를 수신하여 다른 장치에서 데이터를 감지합니다. ObscuredPrefs에서 호출
ObscuredFile \ ObscuredFilePrefs에서 한 번만(세션당) – 파일을 읽을 때마다 호출합니다.

- DeviceLockLevel.Strict를 사용하면 기본적으로 다른 장치의 저장을 읽을 수 없습니다. 줄일 수 있습니다
  DeviceLockSettings.DeviceLockTamperingSensitivity 를 낮음 (감지 이벤트가 계속 실행됨) 또는 비활성화됨 (이벤트가 실행되지 않음)으로 설정
  합니다. 장치 ID가 예기치 않게 변경된 경우 데이터를 복원하는 데에도 사용할 수 있습니다.

- 장치 ID에 처음 액세스하면 일부 플랫폼에서 CPU 스파이크가 발생하여 처음 액세스할 때 게임 플레이 문제가 발생할 수 있습니다.
  데이터를 로드하거나 저장합니다. 이를 방지하려면 DeviceIdHolder.ForceLockToDeviceInit()를 호출하여 장치 잠금이 활성화된 상태에서 첫
  번째 데이터 로드/저장 시 가능한 스파이크를 방지합니다 . 로딩 페이드 또는 스플래시 화면과 같은 정적인 것을 표시하면서 원하는 시간에 장
  치 ID를 강제로 가져오려면 이 방법을 사용하십시오.

- DeviceIdHolder.DeviceId 속성을 사용하여 장치 ID를 명시적으로 설정합니다. 이는 서버 측 승인이 있는 경우 고유한 사용자 ID(또는 이메일)에 대한 저장을 잠그는 데 유용할
  수 있습니다. 일관된 장치 ID(iOS 7 이상)를 얻을 수 있는 방법이 없기 때문에 iOS에 가장 적합합니다. 우리가 가진 모든 것은 공급업체 및 광고 ID뿐이며 둘 다 변경할 수
  있고 제한이 있을 수 있습니다.

코드 난독화 및 전체 코드 보호 참고 사항

ACTk는 소스 코드를 난독화하거나 보호하지 않으므로 모든 ACTk 기능을 사용하더라도 여전히 해커와 리버스 엔지니어링에 취약합니다.

따라서 가능하면 좋은 코드 난독화기와 IL2CPP 스크립팅 백엔드를 모두 사용하여 컴파일된 애플리케이션 코드를 분석하고 변경하는 것을 훨씬 더 어렵게 만드는 것이 좋습니다.

IL2CPP는 Mono의 IL 바이트코드가 모든 IL 리버싱 도구를 쓸모없게 만드는 경우(예: dnSpy) 대신 메타데이터가 포함된 원시 바이너리 코드를 생성합니다. 등) 메서드 본문을 제대로 디컴파일하기가 훨씬 더 어려워집니다(비싸고 사용하기 쉽지 않은 기본 디스어셈블러 및 디컴파일을 통해서만 가능). 부수적으로 관리되는 어셈블리가 Mono ApplicationDomain에 삽입되는 것을 완전히 방지합니다.

메타데이터는 제한된 반사 목적으로 생성되며 여전히 메서드 코드 없이 IL 어셈블리를 구성할 수 있지만 모든 네임스페이스, 클래스, 필드 등을 사용하여 치터는 IL2CPP Dumper 와 같은 도구를 적극적으로 사용합니다. 전체 코드 구조를 보기 위해 IL 어셈블리를 재구성합니다.

그렇기 때문에 Unity의 빌드 프로세스에 통합되고 IL2CPP가 빌드를 만들기 전에 코드를 난독화하는 이름 난독화 기능이 있는 우수한 코드 난독화기로 IL2CPP를 보완하는 것이 매우 중요합니다. 이러한 경우 대부분의 IL2CPP 메타데이터는 쓸모없는 이름으로 엉망이 되어 재구성된 IL 어셈블리를 리버스 엔지니어링 및 분석하기가 매우 어렵습니다.

무료 코드를 포함하여 좋은 코드 obfuscator가 많이 있지만 Unity와 제대로 통합되는 것은 많지 않으므로 Obfuscator를 살펴보는 것이 좋습니다. 에셋 스토어의 플러그인 (이 플러그인에 대한 자세한 내용은 타사 섹션 참조).

특정 플랫폼 및 사용 사례에 대한 기본 보호 장치도 많이 있습니다. 가능한 경우 기본 보호기 사용을 고려하십시오. 애플리케이션을 편집하거나 리버스 엔지니어링하려
는 해커 기술(예: Denuvo, VMProtect 등)에 대한 추가 보호 계층 증가 요구 사항이 추가됩니다.

어쨌든 클라이언트 측(예: 해커가 도달할 수 없는 서버 측이 아님)에 코드의 일부 민감한 부분이 있고 ACTk가 여기에 있는 경우 코드가 진짜이고 변조되지 않았는지 확인하는 것
이 좋습니다. CodeHashGenerator 사용을 돕기 위해 .

코드 무결성 검증

중요한:

• 지금까지는 Android 및 Windows PC 빌드만 지원됩니다. • 주로 추가 코딩이 필요하
므로 고급 Unity 개발자용

코드 무결성 검증 이면의 일반적인 아이디어는 코드의 고유한 지문(해시)을 만드는 것입니다. 이 지문은 지문을 얻은 후 코드가 변경되는 즉시 변경되고 해당 지문을 런타임 애플리케이
션이 가지고 있는 현재 지문과 비교하는 것입니다. .

초보자 치터를 처리해야 할 때 대부분의 간단한 경우에 응용 프로그램에서 바로 비교를 수행하는 것으로 충분하며 초보자 치터가 알아차리지 못할 가능성이 더 높으므로 바로 간
단한 해시 비교부터 시작할 수 있습니다. 애플리케이션의 C# 코드입니다.

그러나 치료의 특성으로 인해 고급 사기꾼이 수표를 변경할 수도 있으므로 C# 코드 외부에서 유효성 검사를 수행하는 것이 더 안전합니다. 여기에서 최선의 선택 - 서버측 유효성 검사를 수행합니다. 현재 해시를 서버로 보내고 초기에 생성된(허용된) 해시와 비교하여 코드가 변경되지 않았는지 확인하기만 하면 됩니다.

따라서 유효성 검사 절차는 일반적으로 참조 해시 가져오기, 실제 런타임 해시 가져오기, 해시 비교의 세 단계로 구성됩니다. 아래의 각 단계에 대한 자세한 내용을 읽어보십시오.

중요한:

- 해시 생성 작업(에디터에서 또는 런타임 시 모두)은 빌드의 각 파일에 대한 파일별 해시를 생성 하고 가져옵니다.
  해당 파일의 요약 해시 .
- 따라서 파일당 해시는 편집기(CodeHashGeneratorPostprocessor에서 가져옴) 와 런타임(CodeHashGenerator에서 가져옴) 모두에서 변경되지 않은 상태로 유지 되어야 하지만 요약 해시는 경우에 따라 다를 수 있습니다(예: 런타임 앱이 AAB 번들의 일부이고 내부에 플랫폼별 파일이 거의 없습니다).

- 권장 해시 비교 전략: 요약 해시를 먼저 비교하고 일치하지 않는 경우 미리 생성된 파일별 해시 화이트리스트에 대해 모든 런타임 파일별 해시를 확인합니다(그리고 없는 파일을 무시하면서 알 수 없는 해시를 빌드 변경 트리거로 처리). .

CodeHashGenerator포스트 프로세서

이 편집기 스크립트를 사용하면 결과 빌드 파일에서 코드의 참조 해시를 생성할 수 있습니다 .
HashesGenerated 이벤트를 수신하여 각 빌드 후에 결과 해시를 얻을 수 있습니다.
이 이벤트를 활성화하려면 ACTk 설정 에서 "빌드 완료 시 코드 해시 생성" 옵션을 활성화하십시오 . 결과 해시는 Editor Console에도 출력됩니다(이벤트를 구독하지 않은 경우에도 마찬가지임).

내장 메뉴 항목을 통해 외부 빌드 코드 해시를 수동으로 가져올 수도 있습니다. 도구 > 코드 스테이지 > 치트 방지 툴킷 > 외부 빌드 해시 계산 또는 동일한 결과를 얻기 위해 편집기 코드에서 직접
CalculateExternalBuildHashes()를 호출할 수 있습니다 .

이 포스트 프로세서는 런타임에 빌드를 실행하지 않고 해시를 생성하도록 만들어졌습니다.

편집기 해시 생성에 문제가 있거나 Unity 빌드를 마친 후 코드를 사후 처리하는 경우 정품 빌드에서 CodeHashGenerator를 사용하여 런타임에 참조 해시를 생성하십시오.

"Examples/Code Genuine Validation/Scripts/Editor" 에 있는 Editor PC Build Hashing 예제 대응 항목 에서 사용 예제를 참조하십시오.

CodeHashGenerator

이 런타임 스크립트를 사용하면 실행 중인 앱에서 바로 참조 또는 런타임 해시를 생성할 수 있습니다 .
정적 이벤트 HashGenerated를 구독하고 Generate() 메서드를 호출하여 해시 생성을 시작합니다. 기본 해시 생성 코드는 플랫폼마다 다르지만 일반적으로 가능한 경우 별도의 스레드 또는 코루틴에서 작동하여 생성 프로세스를 원활하게 만들고 CPU 스파이크로 인해 전반적인 앱 성능 저하를 방지합니다.

런타임에 참조 해시를 자유롭게 생성하여 나중에 런타임 해시와 비교하여 코드 무결성을 검증할 수 있습니다.

"Examples/Code Genuine Validation/Scripts/Runtime" 및 "Examples/Code Genuine Validation/GenuineValidator" 장면에 배치된
Runtime PC Build Hashing 예제 대응 항목 에서 사용 예제를 참조하십시오 .

이 예는 개념을 보여주고 단순하게 유지하는 데 더 적합하기 때문에 Windows PC 플랫폼에만 해당됩니다.

일반 감지기 기능 및 설정

ACTk에는 다양한 치트를 감지하고 그에 따라 반응할 수 있도록 다양한 감지기가 포함되어 있습니다.
모든 감지기에는 몇 가지 공통 기능과 설정 프로세스가 있습니다.

일반적으로 감지기를 설정하고 사용하는 세 가지 방법이 있습니다.

- Unity Editor를 통한 설정

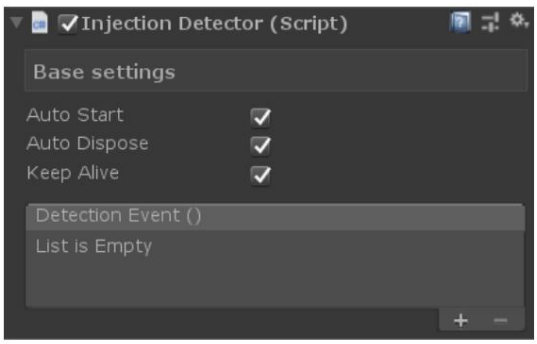
• 코드를 통한 설정 • 혼합 모드

Unity 에디터를 통한 설정

먼저 장면에 감지기를 추가해야 합니다. 두 가지 옵션이 있습니다.

1. GameObject > Create Other > Code Stage > Anti-Cheat Toolkit 메뉴를 사용하여 탐지기를 장면에 빠르게 추가합니다.
   "Anti-Cheat Toolkit Detectors" 감지기가 있는 게임 개체가 장면에 생성됩니다(존재하지 않는 경우).
   감지기를 설정하는 데 권장되는 방법입니다.
2. 또는 다음을 통해 선택한 기존 게임 개체에 탐지기를 추가할 수 있습니다.
   구성 요소 > 코드 단계 > Anti-Cheat Toolkit 메뉴

다음 단계 – 추가된 탐지기를 구성합니다. 모든 감지기에는 구성할 몇 가지 공통 옵션이 있습니다.
다음은 예를 들어 새로 추가된 주입 감지기 입니다 (일반적인 옵션을 제외하고는 다른 옵션이 없기 때문).



자동 시작: 장면 로드 후 탐지기가 자동으로 시작되도록 활성화합니다. 그렇지 않으면 코드에서 명시적으로 시작해야 합니다(아래 세부 정보 참조). 이 기능을 사용하려면 아래에 설명된 감지 이벤트를 구성해야 합니다 .

Auto Dispose: 탐지기가 치트 탐지 후 자동으로 자체를 처리하고 자체 구성 요소를 파괴하도록 활성화합니다.
그렇지 않으면 감지기가 중지됩니다.

Keep Alive: 감지기가 새 레벨(장면) 로드를 유지하도록 합니다. 그렇지 않으면 감지기가 새로운 수준의 로드에서 자체 폐기됩니다.

감지 이벤트: 표준 Unity 이벤트 입니다. 치트 감지 시 실행되는 감지기.

권장 설정 – 모든 옵션을 활성화하고 감지를 위해 적절한 이벤트를 설정합니다.
이 경우 장면을 통해 이동하는 탐지기가 항상 작동하며 탐지 후 자체적으로 파괴됩니다.
감지 콜백이 있는 스크립트를 감지기가 있는 동일한 게임 개체에 첨부할 수 있으며 둘 다 장면 재로드에서 살아남습니다.

코드를 통한 설정

코드에서 항목을 설정하는 것을 선호하는 경우 - 기본 매개변수로 해당 감지기를 실행하려면 코드의 아무 곳에서나 감지기의 정적 StartDetection(callback) 메서드를 한 번만 호출하면 됩니다. 감지기가 존재하지 않는 경우 장면에 자동으로 추가됩니다.

옵션을 조정하려면 정적 인스턴스 속성을 사용하여 감지기를 시작한 직후 구성 가능한 모든 필드와 속성에 도달하십시오(장면에 그러한 감지기가 없으면 인스턴스는 null이 됩니다).
일반적으로 오버로드된 StartDetection() 메서드 의 인수를 통해 초기 구성에 사용할 수 있는 탐지기 옵션에 특정한 일부 옵션은 초기 조정에 사용할 수 있는 옵션을 파악하려면 탐지기에 대한 API 문서를 참조하세요.
주입 감지기 의 예는 다음과 같습니다 .

```
// 이 줄을 .cs 파일의 시작 부분에 바로 배치하세요! CodeStage.AntiCheat.Detectors 사용 ;


// 지정된 콜백으로 탐지를 시작합니다 .
InjectionDetector.StartDetection(OnInjectionDetected);

// 이 메서드는 주입 감지 시 호출됩니다. private void OnInjectionDetected()
{ Debug.Log("Gotcha!"); }
```

혼합 설정

탐지기를 사용하면 인스펙터의 구성과 코드를 통한 수동 실행을 결합하여 혼합 방식으로 설정할 수 있습니다.

Unity 편집기 항목을 통해 설정 에 설명된 대로 감지기를 장면에 추가하고 감지 이벤트를 비워 두고 코드 항목을 통해 설정 에 설명된 대로 StartDetection(콜백) 메서드를 사용하여 코드에서 수동으로 시작할 수 있습니다 .

이 방법을 사용하면 감지기가 시작되어야 하는 순간을 제어할 수 있으며 검사기를 시작하기 전에 Instance 속성 을 통해 코드에서 검사기 모두에서 감지기를 구성할 수 있습니다 .

이러한 접근 방식을 사용하려면 탐지기 검사기에서 자동 시작 옵션을 비활성화해야 합니다.

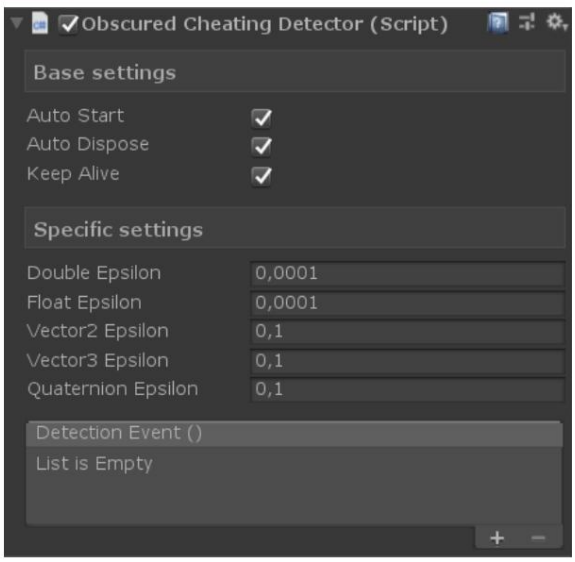
또한 인스펙터에서 감지 이벤트를 채우고 StartDetection() 메서드(인수 없이)를 사용하여 감지기를 시작할 수 있습니다.

가려진 유형의 부정 행위 감지

Anti-Cheat Toolkit의 Obscured Cheating Detector를 사용 하면 ObscuredPrefs를 제외한 모든 가려진 유형 의 부정 행위를 탐지할 수 있습니다 (다음 섹션에서 다루는 자체 탐지 메커니즘이 있음).

실행할 때 이 탐지기는 가려진 변수가 사기꾼을 위한 허니팟으로 암호화되지 않은 가짜 값을 사용할 수 있도록 허용합니다. 원하는 값을 찾을 수 있지만 값을 변경하거나 고정해도 부정 행위 감지 트리거 외에는 아무 작업도 수행되지 않습니다.

감지기의 설정 및 공통 기능에 대한 자세한 내용은 위의 공통 감지기 기능 및 설정 항목을 참조하세요 .



엡실론: 치트 시도가 감지되기 전에 가짜와 실제 암호화 변수 사이에 허용되는 최대 차이.
기본값은 대부분의 경우에 적합하지만 사용자의 경우에 맞게 조정할 수 있습니다(예를 들어 어떤 이유로 잘못된 긍정이 있는 경우).

중요한:

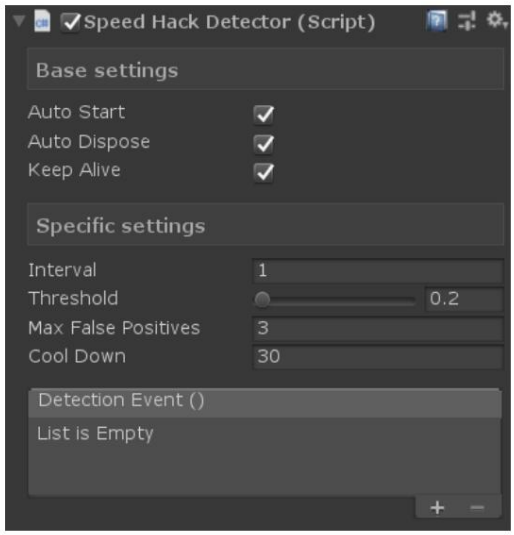
- 부정 행위 탐지가 작동하지 않으며 탐지기가 없는 동안 암호화되지 않은 가짜 변수가 메모리에 존재하지 않습니다.
  달리기. 따라서 ObscuredCheatingDetector를 사용하지 않거나 검사기에서 자동 실행을 비활성화하고 코드에서 실행하지 않으면 값이 메모리 검색기에 전혀 표시되지 않습니다.

스피드핵 감지 [동영상 튜토리얼]

이 유형의 부정 행위는 매우 사용하기 쉽고 주변의 모든 어린이가 게임에서 시도할 수 있기 때문에 매우 인기 있고 자주 사용됩니다. Speed Hack을 사용하면 몇 번의 클릭만으로 다양한 치팅 도구에서 게임 속도를 높이거나 낮출 수 있습니다.

Anti-Cheat Toolkit의 Speed Hack Detector를 사용하면 Cheat Engine, GameGuardian 등과 같은 도구에서 Speed Hack 사용을 감지할 수 있습니다. 사용법도 정말 간편해요 ;)

감지기의 설정 및 공통 기능에 대한 자세한 내용은 위의 공통 감지기 기능 및 설정 항목을 참조하세요 .



간격: 탐지 기간(초). 추가 오버헤드를 피하기 위해 이 값을 1초 이상으로 유지하는 것이 좋습니다.

임계값: 속도를 높이거나 낮추기 위해 허용되는 속도 배율입니다. 하드웨어 다양성으로 인해 타이머가 약간 더 빠르거나 느린 드문 경우에 대해 가능한 오탐을 줄이기 위해 도입되었습니다. 기본값 0.2는 1.2x 및 0.8x 게임 속도를 모두 허용하며 일반적으로 충분히 안정적이며 대부분의 알려진 타이머 문제를 해결해야 합니다.

최대 오탐지: 매우 드문 경우(시스템 클럭 오류, OS 결함 등) 감지기가 오탐지를 생성할 수 있습니다. 이 값을 사용하면 치트에 반응하기 전에 지정된 양의 스피드 해킹 감지를 건너뛸 수 있습니다. 실제 스피드 핵이 적용되면 탐지기가 모든 검사에서 이를 감지합니다. 빠른 예: 간격을 1로 설정하고 최대 오탐지를 5로 설정했습니다. 애플리케이션에 스피드 해킹이 적용된 경우 탐지기는 그 후 ~5초 내에 감지 이벤트를 실행합니다(간격 * 최대 오탐지 + 다음 확인까지 남은 시간).).

Cool Down: 행에서 지정된 양의 성공적인 촬영 후 내부 오탐 카운터를 재설정할 수 있습니다.
앱에 드물지만 주기적인 오탐지가 있는 경우(예: 일부 지속적인 OS 타이머 문제의 경우) 오탐지 카운터를 천천히 증가시키고 전혀 잘못된 스피드 해킹 탐지로 이어지는 경우 유용할 수 있습니다.
냉각 기능을 완전히 비활성화하려면 값을 0으로 설정하십시오.

참고용으로 "내부적으로" 발생하는 몇 가지 예를 보여드리겠습니다. 다음 매개변수를 사용하겠습니다. 간격 = 1, 최대 오탐지 = 5, 쿨 다운 = 30

전. 1: 응용 프로그램은 속도 해킹 없이 작동합니다. Detector는 매초 내부 검사를 실행합니다 (간격 == 1). 무언가 잘못되어 타이머가 동기화되지 않으면 오탐지 카운터가 1씩 증가합니다. 30초 (간격 * 냉각) 의 원활한 작업(OS 딸꾹질 없이) 후 오탐지 카운터는 다시 0으로 설정됩니다. 이는 원하지 않는 감지를 방지합니다.

전. 2: 애플리케이션이 시작되고 얼마 후 애플리케이션에 Speed Hack이 적용되었습니다. Detector는 1초마다 내부 검사를 실행하여 가양성 카운터를 1씩 올립니다. 5초 후에 (간격 * 최대 가양성 수) 치트가 감지됩니다.
치터가 감지를 피하려고 시도하고 3초 후에 스피드 해킹을 중지하면 치트를 다시 적용하기 전에 30초를 더 기다려야 합니다. 꽤 짜증나. 그리고 Cool Down을 최대 60(1분 대기)까지 늘리면 더 짜증날 수 있습니다. 그리고 사기꾼은 그것을 활용하기 위해 Cool Down에 대해 전혀 알아야 합니다.

SpeedHackProofTime API

SpeedHackDetector 와 함께 SpeedHackProofTime 클래스를 활용하여 일반적으로 속도 핵 문제가 있는 Unity의 Time.* 타이머 대신 신뢰할 수 있는 타이머를 사용할 수 있습니다. Mimics Time.* 고정* 물리 API를 제외한 API.
SpeedHackDetector가 실행 중일 때만 작동하며 실제 스피드 해킹이 감지될 때까지 Unity의 타이머로 돌아갑니다.
애니메이션 및 내부 시간 관련 계산이 감지된 속도 해킹에 영향을 받지 않도록 만드는 데 사용합니다.

"Examples/API Examples/Scripts/Runtime/InfiniteRotatorReliable" 스크립트 및 "API Examples" 장면 에서 사용 예를 참조하십시오 .

중요한:

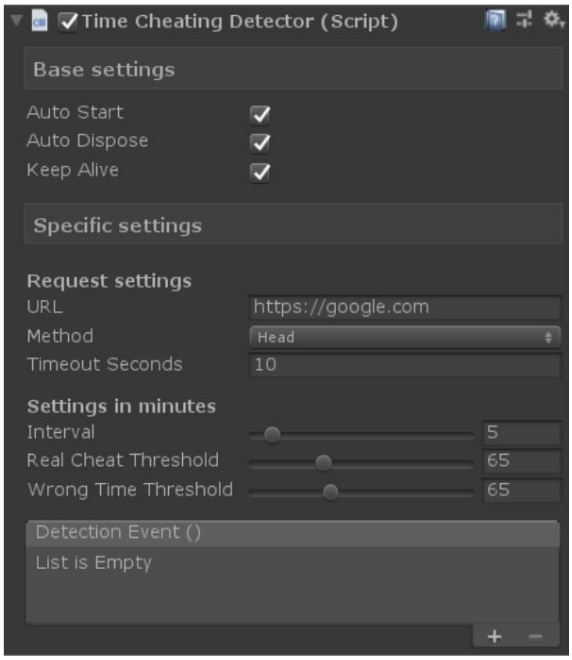
• SpeedHackProofTime은 경우에 따라 작동하지 않을 수 있습니다(예: 신뢰할 수 있는 타이머에 도달할 법적 방법이 없는 경우).
(Chrome 브라우저의 프로세스와 같은 샌드박스 프로세스에서 발생할 수 있음).

시간 부정 행위 감지

플레이어는 종종 이러한 유형의 부정 행위를 사용하여 몇 시간 또는 며칠 동안 건물 진행 또는 에너지 냉각/복구와 같은 게임의 일부 장기 프로세스 속도를 높입니다. 모든 부정 행위자는 이러한 부정 행위를 활용하기 위해 해야 할 일입니다. 시스템 시간을 변경하여 게임이 새 항목을 만들기 시작한 후 하루 또는 몇 시간이 경과하도록 "생각"하도록 하십시오.

Anti-Cheat Toolkit의 Time Cheating Detector를 사용 하면 온라인 시간 서버를 사용하여 이러한 부정 행위를 감지할 수 있습니다. 이 감지기를 사용하려면 인터넷 연결이 필요합니다.

감지기의 설정 및 공통 기능에 대한 자세한 내용은 위의 공통 감지기 기능 및 설정 항목을 참조하세요 .



URL: HEAD 또는 GET 요청에 올바른 날짜 응답 헤더 값을 반환할 리소스의 절대 URL입니다.
WebGL에서 실행할 때 CORS 제한을 피하기 위해 필요한 경우 URL이 자동으로 현재 도메인으로 변경되었습니다.

방법: 사용할 요청 방법. 헤드는 더 빠르고 트래픽을 적게 사용하지만 일부 웹 서버는 너무 자주 HEAD 요청을 봇 활동 및 임시 차단 클라이언트로 처리할 수 있습니다. 문제가 있을 경우에는 Get 방식이 더 호환성이 좋기 때문에 사용하십시오.

Timeout Seconds: 요청을 중단하기 전에 서버 응답을 기다리는 시간입니다.

간격: 탐지 기간(분). 너무 짧은 간격(1분 미만)을 사용하지 않도록 하십시오.
자원 사용량.

실제 치트 임계값: 온라인 및 오프라인 시간 차이의 두 후속 측정 사이에 허용되는 최대 차이(분). 차이가 이 값을 초과할 때 등록된 실제 치트입니다.

잘못된 시간 임계값: 온라인 시간과 오프라인 시간 사이에 허용되는 최대 차이입니다. 차이가 이 값을 초과하면 다음 확인 결과와 함께 콜백 또는 이벤트가 발생합니다(스크립팅에서만 구독할 수 있음): TimeCheatingDetector.CheckResult.WrongTimeDetected

중요한:

• 추가 이벤트인 CheatChecked를 구독하여 부정행위 확인 결과에 대한 전체 정보를 얻을 수 있습니다.
감지를 시작할 때 스크립팅을 통해 설정할 수 있는 모든 콜백에 동일한 대리자가 사용됩니다.

• 부정 행위를 확인하는 동안 인터넷이 없으면 CheckResult.Error 와 함께 CheatChecked 이벤트가 발생합니다.
결과 인수 및 ErrorKind.OnlineTimeError 오류 인수, 검사를 건너뛰고 지정된 간격으로 다시 시도하도록 예약됩니다. 감지기는 인터넷 연결이 좋지 않거나 때때로 연결되는 장치에서 제대로 작동해야 합니다.

• 이 감지기는 인터넷이 필요하므로 Android의 INTERNET 권한(일반적으로 자동 부여됨) 과 같이 모바일 플랫폼에서 추가 권한 요청을 생성합니다 . 이 감지기가 필요하지 않고 추가 권한을 제거하려면 ACTk 설정 창 에서 ACTK_PREVENT_INTERNET_PERMISSION을 확인하십시오 .

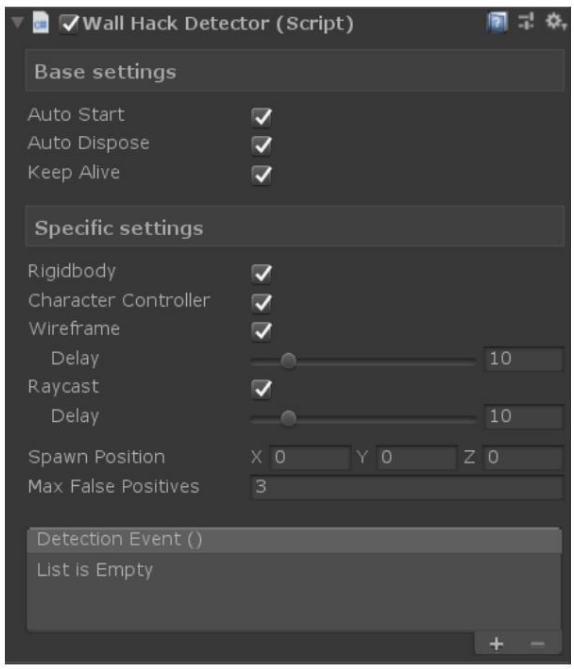
- 로컬 UTC 시간과 온라인 UTC 시간 간의 차이를 유발하는 수동 시스템 시간 변경은 부정 행위로 처리되지만 대신 잘못된 시간으로 처리됩니다.
- ForceCheck* 메서드를 사용하여 수동으로 부정 행위 검사를 실행하고 결과를 얻을 수 있습니다(사용 예는 이러한 메서드의 API 문서 참조). 원하는 순간에 수동으로 검사를 실행할 때 0 간격과 결합하여 탐지기를 완전 수동 모드로 전환할 수 있습니다.

Wallhack 감지 [동영상 자습서]

일반적으로 "벽 핵"으로 언급되는 치팅 방법은 거의 없습니다. 플레이어는 투명 또는 와이어프레임 벽을 통해 볼 수 있고, 유령처럼 벽을 통과할 수 있으며, 벽을 통해 쏠 수 있습니다.

ACTk에는 이 모든 것을 다루는 WallHack Detector가 있습니다.
그것은 몇 가지 다른 모듈로 구성되어 있으며 각각 다른 종류의 치트를 감지합니다.

실행하는 동안 WallHack Detector는 장면에 서비스 컨테이너 "[WH Detector Service]"를 생성하고 설정에 따라 다른 항목을 생성하는 3x3x3 큐브 내의 가상 샌드박스로 사용합니다.
감지기의 설정 및 공통 기능에 대한 자세한 내용은 위의 공통 감지기 기능 및 설정 항목을 참조하세요 .



Rigidbody: 이 모듈을 활성화하여 Rigidbody 해킹을 통해 만들어진 "벽을 통과하는" 유형의 치트를 확인합니다. 캐릭터에 Rigidbody를 사용하지 않는 경우 일부 리소스를 저장하려면 비활성화하십시오.

캐릭터 컨트롤러: 이 모듈을 활성화하여 캐릭터 컨트롤러 해킹을 통해 만들어진 "벽을 통과하는" 유형의 치트를 확인합니다. 캐릭터 컨트롤러를 사용하지 않는 경우 일부 리소스를 저장하려면 비활성화하십시오.

와이어프레임: 이 모듈을 활성화하여 셰이더 또는 드라이버 해킹을 통해 만들어진 "벽을 통해 보기" 종류의 치트(벽이 와이어프레임, 알파 투명 등이 됨)를 확인합니다. 그러한 치트에 신경 쓰지 않는 경우를 대비하여 일부 리소스를 저장하려면 비활성화하십시오.

참고: 런타임에 제대로 작동하려면 특정 셰이더가 필요합니다. 자세한 내용은 아래를 참조하십시오.

와이어프레임 지연: 와이어프레임 모듈 검사 사이의 시간(1초에서 최대 60초).

Raycast: 이 모듈을 활성화하여 Raycast 해킹을 통해 만들어진 "벽을 뚫고 쏘는" 종류의 치트를 확인합니다. 그러한 치트에 신경 쓰지 않는 경우를 대비하여 일부 리소스를 저장하려면 비활성화하십시오.

Raycast Delay: Raycast 모듈 확인 사이의 시간(1초에서 최대 60초).

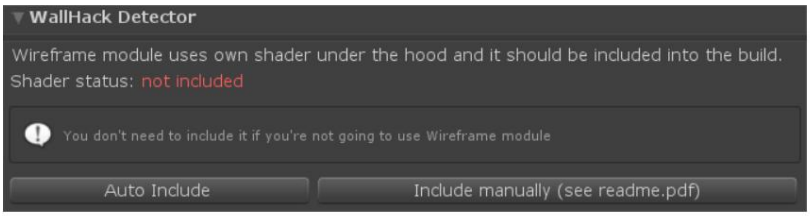
스폰 위치: 장면에서 3x3x3 빨간색 와이어프레임 큐브로 표현되는 동적 서비스 컨테이너의 세계 좌표(탐지기로 게임 개체를 선택하면 표시됨).

최대 오탐지: 실제 탐지 전에 연속으로 허용된 탐지. 각 모듈에는 자체 감지 카운터가 있습니다. 치트가 감지될 때마다 증가하고 해당 모듈의 첫 번째 성공 샷에서 재설정됩니다(치트가 감지되지 않은 경우). 이러한 카운터 중 하나라도 최대 오탐지 값보다 커지면 감지기는 최종 실제 감지를 등록합니다.

와이어프레임 모듈 셰이더 설정

Wireframe 모듈은 내부적으로 Hidden/ACTk/WallHackTexture 셰이더를 사용합니다. 따라서 이러한 셰이더는 런타임에 존재하도록 빌드에 포함되어야 합니다. Hidden/ACTk/WallHackTexture 셰이더가 포함되지 않은 경우 런타임 시 로그에 오류가 표시되며 셰이더가 포함되지 않은 상태에서 WallhackDetector를 실행하면 Editor에 셰이더를 포함하라는 메시지가 표시됩니다.

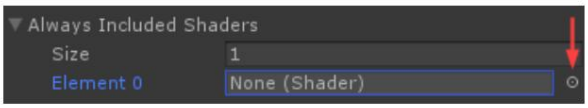
ACTk 설정 창 을 통해 셰이더를 쉽게 추가하거나 제거할 수 있습니다 .



항상 포함된 셰이더 목록 에 Hidden/ACTk/WallHackTexture 셰이더를 자동으로 추가하려면 "자동 포함" 버튼을 누르십시오 .

또한 두 번째 버튼 ("수동 포함") 을 눌러 프로젝트의 그래픽 설정을 열고 수동으로 항상 포함된 셰이더 목록에 셰이더를 추가할 수 있습니다.

셰이더를 목록에 수동으로 추가하려면: • 목록에 요
소를 하나 더 추가합니다. • 새 빈 요소 옆에 있는 전구
를 클릭합니다(아래 이미지의 빨간색 화살표 참조).

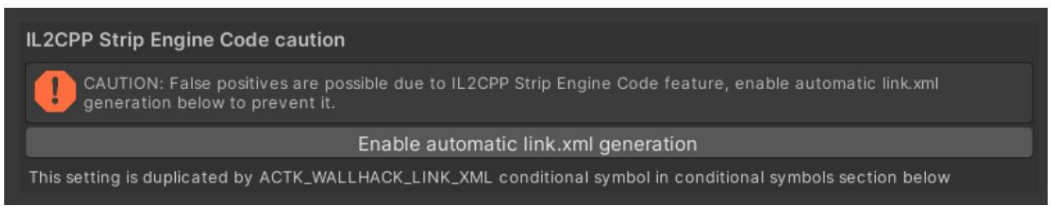


• 열린 창에서 "wallhack"을 검색하고 더블 클릭으로 WallHackTexture를 선택합니다.

이것이 와이어프레임 모듈 설정을 위한 것입니다.

WallHack Detector 및 IL2CPP 공지

IL2CPP로 빌드할 때 Unity 엔진 스트리핑이 시작되어 WallHack Detector에서 사용하는 항목을 때때로 제거하여 오탐으로 이어질 수 있는 사용되지 않는 구성 요소를 제거합니다. 이를 방지하려면 ACTk 설정 (WallHack Detector 섹션 아래) 에서 자동 link.xml 생성을 활성화해야 합니다 .



중요한:

- 월핵은 일반적으로 데스크톱 FPS 게임에 적용되는 매우 특정한 종류의 치트이므로 게임이 어려움을 겪을 수 있는지 확인하십시오.
  탐지에는 상당한 양의 추가 리소스가 필요하기 때문에 탐지기를 사용하기 전에
- ACTk 설정 에서 ACTK_WALLHACK_DEBUG 조건부 컴파일 기호를 활성화 하여
  서비스 개체의 렌더러를 게임에서 볼 수 있습니다. 또한 와이어프레임 모듈의 결과 텍스처의 OnGUI 출력을 활성화합니다. 이 기호는 개발 빌드 또는 편집기에서만 작동합니다.
- 서비스 컨테이너 내의 모든 객체는 "Ignore Raycast" 레이어에 배치되며 기본적으로 렌더러를 비활성화합니다.
- 3x3x3 서비스 샌드박스와 객체가 오탐 및 오작동으로 이어지는 충돌을 방지하려면 Spawn Position을 비어 있고 격리된 공간으로 설정하는 것이 중요합니다.
- 탐지기가 지속적으로 물체를 생성 및 이동하고 물리학을 사용하며 다른
  설정에 따라 실행 중 샌드박스에서 계산합니다.

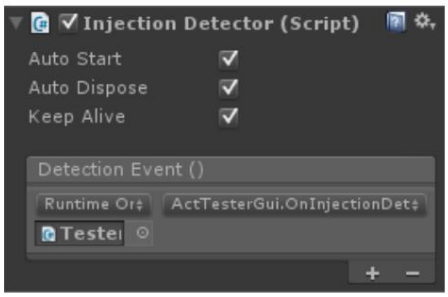
관리형 DLL 삽입 감지 [동영상 자습서]

중요 #1: Mono Android 및 Mono PC 빌드에서만 작동합니다!
중요 #2: IL2CPP 빌드에서는 가능한 어셈블리 주입이 없습니다. 전용 코드 보호 섹션 에서 자세히 알아보십시오 .
중요 #3: 가양성을 유발하는 특정 어셈블리로 인해 편집기에서 비활성화됩니다. ACTK_INJECTION_DEBUG 기호를 사용하여 Editor에서 강제 실행합니다.
경고: 대상 장치에서 앱을 테스트하고 주입 감지기에서 오탐지가 없는지 확인하십시오. 이러한 문제가 있는 경우 감지된 어셈블리를 허용 목록(아래에서 설명)에 추가해 보십시오.

이러한 종류의 부정 행위는 다른 부정 행위만큼 인기가 없지만 여전히 Unity 앱에 사용되므로 무시할 수 없습니다. 이러한 주입을 구현하려면 치터는 고급 기술이 필요하므로 많은 사람들이 특수 포털에서 숙련된 사람으로부터 치트 인젝터를 구입합니다. 무언가를 주입하는 가장 쉬운 방법 중 하나 – 단일 조립 주입기를 사용 또는 유사한 소프트웨어. 일부 미치광이들은 Cheat Engine의 Auto Assemble을 사용하기도 합니다. Cheat Engine이 얼마나 멋진지 아십니까?

Anti-Cheat Toolkit의 삽입 감지기를 사용하면 관리되는 어셈블리(DLL)를 앱에 삽입할 때 대응할 수 있습니다 .
런타임에 사용하기 전에 ACTk 설정 (모노 주입 검출기 지원 추가 체크박스) 에서 활성화해야 합니다 .

감지기의 설정 및 공통 기능에 대한 자세한 내용은 위의 공통 감지기 기능 및 설정 항목을 참조하세요 .



이 감지기에는 일반적인 옵션을 제외하고 조정할 추가 옵션이 없습니다(코드에서 시작할 때 문자열 인수 – 감지 원인을 허용하는 콜백을 전달함).

고급 치트를 감지하는 간단한 방법!

주입 감지기 내부(고급)

통찰력을 제공하기 위해 중요한 주입 감지기 "내부" 주제에 대해 몇 마디 말씀드리겠습니다.
일반적으로 모든 Unity 앱은 세 가지 어셈블리 그룹을 사용할 수 있습니다.

- 시스템 어셈블리: System.Core, mscoree, UnityEngine 등
- 사용자 어셈블리: DOTween.dll과 같은 타사 어셈블리를 포함하여 프로젝트에서 사용할 수 있는 모든 어셈블리(훌륭한 DOTween 의 어셈블리) 트위닝 라이브러리) 또는 외부 패키지의 어셈블리 + 코드에서 Unity가 생성한 어셈블리 - Assembly-CSharp.dll, 어셈블리 정의 어셈블리 등
- 런타임 생성 및 외부 어셈블리. 일부 어셈블리는 리플렉션을 사용하여 런타임에 생성되거나 로드될 수 있습니다.
  외부 소스(예: 웹 서버)에서. 이것은 일반적으로 드문 경우입니다.

주입 감지기는 앱에서 외부 어셈블리를 감지하기 위해 신뢰할 수 있는 모든 유효한 어셈블리에 대해 알아야 합니다.
처음 두 그룹을 자동으로 처리합니다. 마지막 그룹은 수동으로 다루어야 합니다(아래 참조).
Detector는 화이트리스트 접근 방식을 활용하여 허용된 어셈블리를 건너뜁니다. 애플리케이션을 빌드할 때 이러한 목록을 자동으로 생성하고 빌드에 포함합니다 (ACTk 설정 창에서 Enable Injection Detector 확인란을 선택한 경우).

이 화이트리스트는 두 부분으로 구성됩니다.

- 프로젝트에 사용된 어셈블리의 동적 화이트리스트(첫 번째 및 두 번째 그룹 포함)
- 사용자 정의 화이트리스트(세 번째 그룹, 아래 세부 정보 참조).

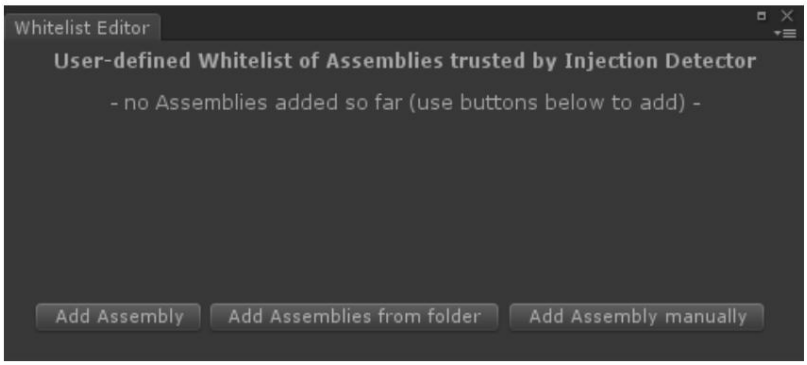
대부분의 경우 설정에서 주입 감지기 지원을 활성화하고 장면 \ 코드에서 설정하여 제대로 작동하도록 하는 것 외에는 아무것도 하지 말아야 합니다.

그러나 프로젝트가 외부 소스에서 일부 어셈블리를 로드하거나 이러한 어셈블리가 프로젝트 자산에 없는 경우 런타임에 어셈블리를 생성하는 경우 탐지기가 이러한 어셈블리에 대해 알려 오탐을 방지해야 합니다. 이러한 이유로 사용자 정의 화이트리스트 편집기가 구현되었습니다.

사용자 정의 화이트리스트를 채우는 방법

화이트리스트에 어셈블리를 추가하려면 다음의 간단한 단계를 따르십시오.

- "Tools > Code Stage > Anti-Cheat Toolkit > Injection Detector Whitelist Editor" 메뉴를 사용하여 Whitelist Editor를 열거 나
  설정 창의 "허용 목록 편집" 버튼:



- 세 가지 가능한 옵션을 사용하여 새 어셈블리를 추가합니다. 단일 파일("어셈블리 추가"), 재귀 폴더 스캔("어셈블리 추가")
  폴더에서"), 수동 추가 - 전체 조립품 이름 채우기("수동으로 조립품 추가").

그게 다야!

감지된 어셈블리를 수동으로 추가하고 전체 이름을 모르는 경우 주입 감지기에서 디버그를 활성화하십시오. ACTk 설정 창 에서 ACTK_INJECTION_DEBUG를 확인 하고 감지기가 어셈블리를 감지하도록 하십시오. "[ACTk] Injected Assembly found:" 문자열 다음에 콘솔 로그에 전체 이름이 표시됩니다. 이 기호는 개발 빌드 또는 편집기에서만 작동합니다.

화이트리스트 편집기를 사용하면 목록에서 어셈블리를 하나씩 제거하고(어셈블리 이름 옆에 작은 "-" 버튼이 있음) 전체 화이트리스트도 지울 수 있습니다.

사용자 정의 화이트리스트는 다른 모든 설정과 함께 ProjectSettings/ACTkSettings.asset 파일에 저장됩니다.

중요한:

- 원하는 경우 주입 예제에 대한 인보이스 ID를 이메일로 보내주십시오(이 문서 끝에 있는 지원 연락처 참조).
  야생에서 InjectionDetector를 사용해보십시오 .
- ACTk 설정 창 에서 ACTK_INJECTION_DEBUG_VERBOSE 및 ACTK_INJECTION_DEBUG_PARANOID 옵션을 확인하여 추가 디버그 정보에 대해 이러한 컴파일 기호
  를 활성화할 수 있습니다. 이러한 기호는 개발 빌드 또는 편집기에서만 작동합니다.

타사 플러그인 통합 및 참고 사항

중요한:

ACTk를 지원하는 일부 타사 플러그인은 프로젝트에 ACTK_IS_HERE 조건부 기호가 필요할 수 있습니다.
ACTk 설정 에서 활성화할 수 있습니다 .

난독화기

Anti-Cheat Toolkit은 부정 행위를 방지하고 탐지하는 다양한 방법을 제공합니다.
또한 설정을 완료하기 위해 우수한 코드 보호 기능으로 ACTk를 보완할 것을 강력히 제안합니다.
다음은 코드 리버스 엔지니어링의 난이도를 높이기 위해 제안하는 두 가지 첫 번째 단계입니다.

- 가능한 경우 Mono 대신 IL2CPP를 사용하는 것이 좋습니다. 치터가 코드를 검사하고 분석하기가 더 어렵기 때문입니다. IL2CPP 빌드에는 사기꾼이 일반적으로 ILSpy 등
  에서 디컴파일하는 IL 바이트코드가 없습니다. 사기꾼은 여전히 모든 코드(네임스페이스, 클래스 이름, 메서드 이름 등)의 메타데이터를 가지고 있지만 악용될 수 있
  습니다. 그래서 두 번째 단계가 존재합니다.

- 사기꾼에게 클래스 이름, 메서드, 필드 및 의미 없는 난장판을 만들기 위해 obfuscator를 사용하는 것을 고려하십시오. 그것은
  코드 리버스 엔지니어링의 난이도를 크게 높입니다.

고맙게도 상점에는 격차를 메우고 코드를 쉽고 원활하게 보호할 수 있는 멋진 자산이 있습니다. Obfuscator, 갑자기 =) Unity용으로 만들어졌기 때문에 Unity 이벤트, 메시지, 코루틴 등에 대해 "알고" 있으므로 ACTK_EXCLUDE_OBFUSCATION 기호 를 적용할 필요가 없습니다 (이 기호에 대한 자세한 내용은 아래 참조).

플레이메이커

현재 ACTk는 PlayMaker를 부분적으로 지원합니다. (오후). 통합/PlayMaker.unitypackage 패키지에서 사용할 수 있는 PM 작업이 거의 없습니다 .
PM Actions Browser에 새 작업을 추가하려면 프로젝트로 가져와야 합니다.
Scripts/PlayMaker/Examples 폴더 에서 몇 가지 예를 확인하세요 .

ObscuredPrefs 및 ObscuredFilePrefs 는 완전히 지원됩니다. Actions Browser의 PlayerPrefs 섹션에서 Obscured * 액션을 찾을 수 있습니다 .

- ObscuredCheatingDetector를 제외한 모든 탐지기가 완전히 지원됩니다. Anti-Cheat 에서 탐지기 작업을 찾을 수 있습니다.
  Actions Browser의 툴킷 섹션.
- PM에 대한 기본 가려진 유형은 현재 지원되지 않습니다. PM은 새로운 변수 유형을 추가하는 것을 허용하지 않습니다. 하지만 그것은
  손으로 통합할 수 있습니다. 예가 있습니다 그리고 설명 크라이슈바이데.

행동 설계자

현재 ACTk는 Opsive Behavior Designer를 완벽하게 지원 합니다. (BD). Integration/BehaviorDesigner.unitypackage 패키지에서 사용할 수 있는 BD 작업(동작 및 조건) 및 SharedVariables가 거의 없습니다 .
Anti-Cheat Toolkit을 BD와 통합하려면 프로젝트로 가져오십시오.
Scripts/BehaviorDesigner/Examples 폴더 에서 몇 가지 예를 참조하십시오 .

ACTk를 지원하는 기타 타사 플러그인

ACTk를 지원하는 다른 타사 플러그인이 있습니다.

- Mad Level Manager: ACTk는 스토리지 백엔드로 사용됩니다 .
- Stan의 자산: Android 기본 플러그인 및 궁극의 모바일 플러그인.

여기에서 플러그인을 보고 싶다면 알려주세요.

문제 해결

- 업데이트 후 오류가 발생했습니다.
  업데이트 문제를 방지하려면 업데이트된 ACTk 패키지를 프로젝트로 가져오기 전에 이전 버전(전체 CodeStage/AntiCheatToolkit 폴더)을 완전히 제거하십시오.
- 감지기 중 하나에 대해 거짓 긍정 또는 기타 오작동이 있습니다.
  코드를 통해 기존 장면 감지기를 시작하는 경우 Awake 또는 OnEnable 단계가 아닌 MonoBehaviour의 Start() 단계에서 StartDetection() 메서드를 사용하고 있는지 확인하세요. ACTk 설정 창 에서 ACTK_DETECTION_BACKLOGS 플래그를 활성화하면 InjectionDetector를 제외한 모든 감지기(원인이 있는 콜백이 있으므로)는 감지에 대한 세부 정보를 인쇄합니다 . 가능한 경우 이 플래그를 활성화한 상태에서 개발 빌드에서 생성된 로그에 대한 오탐지를 보고하십시오.
- InjectionDetector 가양성이 있습니다 .
  게임에서 사용하는 모든 외부 라이브러리를 허용 목록에 추가했는지 확인하세요(메뉴: 도구 > 코드 단계 > 치트 방지 도구 키트 > 삽입 감지기 허용 목록 편집기). 이 화이트리스트를 채우는 방법은 위의 사용자 정의 화이트리스트를 채우는 방법 섹션을 참조하십시오. 도움이 되지 않으면 저에게 연락하십시오.
- WallHackDetector 오탐지가 있습니다 .
  탐지기의 Spawn Position을 올바르게 구성 했는지, 탐지기의 서비스 개체가 게임의 개체와 교차하지 않는지 확인하십시오 . 또한 Edit > Project Settings > Physics의 Layer Collision Matrix에서 Ignore Raycast 레이어가 자신과 충돌하는지 확인하십시오 . WallHackDetector는 모든 서비스 개체를 Ignore에 배치합니다.

Raycast 레이어는 게임 개체와의 불필요한 충돌을 방지하고 이러한 개체를 서로 충돌시키는 데 도움이 됩니다. 그렇기 때문에 Ignore Raycast 레이어가 자신과 충돌하는지 확인해야 합니다.

- WallHackDetector 에서 '클래스 *가 존재하지 않기 때문에 구성 요소를 추가할 수 없습니다' 오류가 발생합니다 .
  WallHackDetector는 BoxCollider, MeshCollider, CapsuleCollider, Camera, Rigidbody, CharacterController, MeshRenderer와 같은 Unity 구성 요소를 사용합니다. 이러한 오류는 이러한 구성 요소 중 하나가 예를 들어 IL2CPP 스트립 엔진 코드 에 의해 제거될 때 나타날 수 있습니다. 옵션. 구성 요소가 벗겨지는 것을 방지하려면 빌드에 있는 모든 장면의 게임 개체에 구성 요소를 추가하거나 해당 link.xml 파일을 Assets 폴더의 아무 곳에나 넣을 수 있습니다. <linker> < assembly fullname="UnityEngine"> <type fullname="

  UnityEngine.BoxCollider"preserv="all"/
  > <type fullname="UnityEngine.MeshCollider"preserv="all"/
  > <type fullname="UnityEngine.CapsuleCollider"preserve="모
  두"/> <type fullname="UnityEngine. 카메라"preserve="모
  두"/> <type
  fullname="UnityEngine.Rigidbody"preserve="모두"/
  > <type fullname="UnityEngine.MeshRenderer"preserve="모
  두"/> <type fullname="UnityEngine.CharacterController"

  preserve="all"/> </assembly> </linker>

  ACTk는 이 경우를 자동으로 처리하고 재생 모드에서 감지하여 경고하고 자동 link.xml 파일 생성을 활성화할 수 있는 설정을 엽니다 (ACTK_WALLHACK_LINK_XML 조건부 컴파일 기호 사용).

- Android에서 READ_PHONE_STATE 권한 요구 사항을 제거하려면 어떻게 해야 합니까 ?
  ObscuredPrefs / ObscuredFile / ObscuredFilePrefs 에서 장치 잠금 기능을 사용하지 않고 Android 애플리케이션을 빌드하는 경우 ACTk 설정 창 에서 ACTK_PREVENT_READ_PHONE_STATE를 확인하여 Android에서 READ_PHONE_STATE 권한 요구 사항을 제거하는 것이 좋습니다 . 그렇지 않으면 ACTk가 장치에 대한 저장을 잠그도록 하려면 이 요구 사항이 필요합니다.

- Android에서 인터넷 권한 요구 사항을 제거하려면 어떻게 해야 합니까 ?
  TimeCheatingDetector를 사용하지 않는 경우 ACTk 설정 창 에서 ACTK_PREVENT_INTERNET_PERMISSION을 확인하여 Android에서 인터넷 권한 요구 사항을 제거하는 것이 좋습니다 . 그렇지 않으면 ACTk가 시간 부정 행위를 감지하도록 하려면 이 요구 사항이 필요합니다.

- ACTk를 사용할 때 obfuscator가 Unity 빌드를 중단합니다.
  이름으로 메서드를 호출할 수 있는 Unity의 메시지, Invoke() 및 코루틴을 인식하지 못하는 obfuscator를 사용하고 obfuscator가 System.Reflection.ObfuscationAttribute와 호환되는 경우 ACTk 설정 창 에서 ACTK_EXCLUDE_OBFUSCATION을 확인하여 이러한 메소드에 [Obfuscation(Exclude = true)] 속성을 추가하십시오 . 도움이 되지 않으면 저에게 연락하십시오.

- 매우 오래된 일부 버전에서 ACTk를 업데이트했는데 이제 내 ObscuredPrefs를 읽을 수 없습니다.
  ObscuredPrefs를 사용 중이고 ACTk를 업데이트하려면 다음을 고려하십시오. ACTk >= 1.4.0은 ACTk < 1.2.5에서 ObscuredPrefs로 저장된 데이터를 해독할 수 없으므로 버전 < 1.2에서 점프하지 않도록 하십시오. 5 버전 >= 1.4.0으로 직접. prefs를 새 형식으로 자동 변환하거나 암호 해독 코드를 가져오고 ACT >= 1.4.0에 대한 추가 폴백으로 가져오려면 중간에 있는 모든 버전을 사용해야 합니다. 마이그레이션에 문제가 있는 경우 언제든지 저에게 연락해 주시면 원활하게 처리할 수 있도록 도와드리겠습니다.

- Assets/Resources/fndid.bytes 파일 에 병합 충돌이 있습니다 .
  프로젝트에 버전 제어를 사용하고 있고 InjectionDetector를 사용하고 있다면 병합 충돌이 있을 수 있습니다. 괜찮습니다. 해당 파일을 VCS의 무시 목록에 추가하기만 하면 됩니다. fndid.bytes 는 자동으로 생성되며 시스템마다 다를 수 있습니다.

- 콘솔에 StackOverflowException 오류가 표시됩니다.
  다음 메시지가 표시되는 경우:
  StackOverflowException: 요청한 작업으로 인해 스택 오버플로가 발생했습니다.
  CodeStage.AntiCheat.ObscuredTypes.ObscuredPrefs.HasKey(System.String 키)(*CodeStage/AntiCheatToolkit/Scripts/ObscuredTypes/ObscuredPrefs.cs:*)
  클래스뿐만 아니라 ACTk 클래스에서도 실수로 모든 PlayerPrefs.* 호출을 교체했을 가능성이 큽니다.
  프로젝트에서 ACTk를 제거하고 다시 가져오면 문제가 해결됩니다.

- ObscuredString.GetEncrypted() \ EncryptDecrypt() 메서드 의 결과로 문자열이 잘리거나 너무 짧습니다 .
  가려진 문자열은 암호화 중에 줄 바꿈, 줄 끝 등과 같은 특수 문자를 포함할 수 있습니다. 따라서 깨진 것처럼 보일 수 있지만 길이와 실제 내용은 여전히 양호해야 합니다.

- ACTk를 1.5.8.0 또는 이전 버전에서 업데이트한 후 손상된 ObscuredFloats / ObscuredDoubles가 표시됩니다.
  이러한 유형은 1.6.0에서 새로운 형식을 얻었으며 저장되거나 직렬화된 모든 인스턴스를 새 형식으로 마이그레이션해야 합니다.

  이렇게 하려면 다음을 사용합니다.

• Unity 직렬화된 인스턴스 마이그레이션을 위한 메뉴 명령: Tools > Code Stage > Anti-Cheat Toolkit > Migrate • GetEncrypted()로 저장하기 위한 런타임 API: ObscuredFloat/ObscuredDouble.MigrateEncrypted() • ACTk Settings 창 에서 런타임 활성화를 위한 조건부 컴파일 플래그 ACTK_OBSCURED_AUTO_MIGRATION 새 값으로 다시 작성될 때까지 인스턴스에 대한 확인 및 자동 마이그레이션 ObscuredFloat/ObscuredDouble의 성능을 약간 줄입니다.

1.5.1.0 이하에서 업데이트하는 경우 첫 번째 형식 변경 후 1.5.2.0에 추가된 추가 마이그레이션 단계를 수행해야 합니다. 전체 *.*.*.* - 1.5.2.0 - 1.6.0 마이그레이션을 수행하려면 이전 ACTk 버전을 요청하십시오.

• 이전 Android 장치(Android 6 이전)의 Time Cheating Detector 로그에 오류 응답 코드: 0 또는 java.io.EOFException 오류가 표시됩니다 .

UnityWebRequest에는 이전 Android 기기에서만 재현되는 알려진 버그가 있습니다. 이 버그를 해결하려면 요청 방법을 HEAD 대신 GET으로 설정해 보십시오.

• Android 또는 WebGL의 StreamingAssets에서 ObscuredFile 또는 ObscuredFilePrefs를 읽거나 쓸 수 없습니다. 이것은 이러한 플랫폼의 StreamingAssets 특성으로 인해 예상되는 동작(Unity 매뉴얼 참조) 상세 사항은). 이 문제를 해결하려면 이 간단한 예제 와 같이 ObscuredFile로 파일을 읽기 전에 StreamingAssets 폴더에서 File.IO 액세스 가능 위치(예: Application.dataPath) 로 파일을 복사하십시오. 아이디어를 보여주기 위해 만들어졌습니다.

호환성

아래에 나열된 몇 가지 예외를 제외하고 모든 기능은 알려진 모든 플랫폼에서 완벽하게 작동해야 합니다. • InjectionDetector 는 Android Mono 및 Standalone Mono 빌드에서만 작동합니다. • CodeHashGenerator는 Android 및 Windows Standalone 빌드에서만 작동합니다. • ObscuredFIle은 지원 중단으로 인해 UWP .NET에서 작동하지 않습니다(UWP IL2CPP가 지원됨).

플러그인은 독립 실행형 (Win, Mac, Linux, WebGL), iOS, Android, UWP 변형 플랫폼에서 테스트되었습니다 .

또한 고객은 Windows Phone 8, Apple TV(thx atmuc)에서 작동하는 것으로 보고했습니다.

특정 플랫폼에서 플러그인이 작동하지 않는 경우 알려주세요. 도움을 주고 수정하도록 노력하겠습니다.

Apple 암호화 수출 규정 호환성

이러한 ACTk 기능은 기본적으로 Apple의 수출 준수와 호환되지 않습니다. • ObscuredFile 및 ObscuredFilePrefs

• ObscuredLong, ObscuredULong, ObscuredDouble 및 ObscuredDecimal 유형

앱에서 암호화를 사용하는 경우 Apple App Store에 게시할 때 암호화를 사용하고 있음을 선언해야 합니다.

조건부 컴파일 기호 설정에서 ACTK_US_EXPORT_COMPATIBLE 옵션을 사용하여 이를 방지할 수 있습니다 .
대칭 암호화를 위해 56비트를 초과하는 키 생성을 방지하므로 산업 보안국 상거래 제어 목록 카테고리 5 파트 2, 2.a.1.a의 암호화 정의에 더 이상 속하지 않습니다. 수출 규정과 완벽하게 호환되므로 ACTk는 Apple App Store에 게시할 때 응용 프로그램에서 암호화를 사용한다고 선언하도록 강요하지 않습니다.

그러나 다음과 같은 부작용이 있습니다.

• ObscuredFile 및 ObscuredFilePrefs 암호화 강도는 128비트 키가 있는 AES에서 56비트 키가 있는 RC2로 전환하여 약화됩니다(데이터는 여전히 암호화되고 읽을 수 없음).

• 부분적인 ObscuredLong, ObscuredULong, ObscuredDouble 및 ObscuredDecimal 가려짐 약화, 드문 경우지만 완전히 암호화됩니다. ObscuredCheatingDetector는 어쨌든 그들을 주시할 것입니다.

ProGuard 공지

Android 빌드에 대해 최소화를 활성화한 경우 제외해야 합니다. 즉, 다음을 proguard-user.txt에 추가합니다. -keep class net.codestage.actk.androidnative.ACTkAndroidRoutines { *; } -유지 클래스 net.codestage.actk.androidnative.CodeHashGenerator {public void GetCodeHash(...);} -유지 클래스 net.codestage.actk.androidnative.CodeHashCallback { *; }

타사 라이선스 포함

xx해시샤프 https://
github.com/noricube/xxHashSharp BSD
2-Clause 라이선스 (http://www.opensource.org/licenses/bsd-license.php)

SharpZipLib
https://github.com/icsharpcode/SharpZipLib
MIT 라이선스 (https://opensource.org/licenses/MIT)

저자의 마지막 말

다시 한 번 말씀드리지만 제 툴킷은 매우 숙련되고 의욕이 있는 사기꾼을 막을 수 있는 것이 아닙니다.
실제로 그러한 솔루션은 없으며 한 달에 수천 달러의 비용이 드는 거대한 안티 치트 솔루션조차도 여전히 결함과 부정 행위 청중이 있습니다.

ACTk는 대부분의 치팅 플레이어를 상대로 도움이 되며, 고급 솔로 치터의 삶을 더 어렵게 만들 것입니다 :P

귀하의 필요에 적합한 Anti-Cheat Toolkit을 찾고 귀중한 시간을 절약할 수 있기를 바랍니다 !
고객님의 후기를 남겨 주세요 플러그인의 에셋 스토어 페이지에서 버그 보고서, 기능 제안 및 기타 의견을 포럼이나 아래 지원 연락처를 통해 자유롭게 알려주세요.

시간을 내어 이 문서를 읽어주셔서 감사합니다!

Anti-Cheat Toolkit 링크 및 지원

에셋 스토어 | 홈페이지 | API | 법정 | 유튜브

지원 연락처:
discord.gg/KrU4psWffA
codestage.net/contacts
support@codestage.net

업데이트 및 뉴스 구독:
Discord 공지 채널 @codestage_net
@codestage

행운을 빕니다,
Dmitriy Yukhanov
Asset Store 게시자
codestage.net

추신 #0 Unity 에셋 스토어에서 저를 지원하고 매일 행복하게 해준 가족에게 감사드립니다!
PS #1 Daniele Giardini 에게 큰 감사를 전하고 싶습니다. (닷윈, HO도구, 고스커리 그리고 이 툴킷에 대한 멋진 로고, 집중적인 도움말 및 귀중한 피드백을 제공하는 다른 많은 행복 생성자)!