iOS GAME HACKING

FROM ZERO TO GOD MODE

- PUSH START -

PLAYER 1: LUIS DE LA ROSA

PLAYER 2: STEEVEN RODRIGUEZ

STAGE 00

SELECT YOUR HACKER

- PLAYER ID -

LUIS DE LA ROSA



CLASS: OFFENSIVE

> Senior Security Consultant @ BF
> CTF player, UFC fan
> Boxing Practicioner

STATUS: READY

- PLAYER ID -

STEEVEN RODRIGUEZ



CLASS: OFFENSIVE

> Ecuadorian
> Taquiqueso Fan
> reverses for fun

STATUS: READY

STAGE 01

IOS SECURITY MYTH

MYTH

[devs believe]

X iOS apps can't be modified

X Code is platform-protected

X Jailbreak is dead

X Compilation = obfuscation

REALITY

[what happens]

O Code runs on the client

O Memory can be inspected

O Runtime can be hooked

>> CONTROL THE RUNTIME, CONTROL THE GAME <<

STAGE 02

WHAT IS GAME HACKING

INTRO

$ Game hacking is modifying the behavior of a game by manipulating its runtime.

POWER-UPS:

+ GOD MODE
+ INFINITE COINS
+ SPEED HACKS
+ INSTANT WIN



– TARGET –

>> GAME STATE LIVES IN CLIENT MEMORY

STAGE 03

WHY MOBILE GAMES

$200B+

GLOBAL REVENUE

> bigger than console+PC

$ IN-APP $$$

real money at stake

COMPETITIVE

esports, ranked

X CHEATERS WIN

ruin matches

U UNITY 70%+

shared attack surface

STAGE 04

REAL-WORLD CASES

- TARGET -

POKEMON GO

> GPS spoofing
> joystick mods
> auto-catch bots

- TARGET -

CLASH ROYALE

> gem generators
> resource hacks
> bot armies

- TARGET -

PUBG MOBILE

> wallhacks
> aimbots
> speed hacks

>> SAME TOOLS. DIFFERENT GAMES. UNIVERSAL ATTACKS.

STAGE 05

MOBILE 101 - WHAT IS AN APP

THE APP

[what ships to the phone]

> a game = one .ipa file
> an .ipa is just a .zip
> unzip it and you get
> a folder: MyGame.app
> that folder IS the game

INSIDE

[what is in that folder]

> one big compiled binary
> images, sounds, levels
> and the game logic:
> health, coins, speed...
> all of it, on the phone

>> THE WHOLE GAME IS IN YOUR HAND. <<

STAGE 06

SO WHY IS IT HARD?

– LOCK 01 –

SANDBOX

> every app is caged
> no file access
> no shell on a phone

– LOCK 02 –

CODE SIGNING

> Apple signs the app
> change 1 byte = dead
> iOS refuses to run it

– LOCK 03 –

ENCRYPTION

> App Store apps are encrypted on disk
> Ghidra just sees noise

>> ONE THING OPENS ALL THREE: A JAILBREAK.

STAGE 07

WHAT IS A JAILBREAK

WHAT IT IS

```
$ jailbreak --explain

[+] not an app you install
[+] it is an exploit chain
[+] it turns off the 3 locks
[+] you end up with root

// you jailbreak the PHONE,
// not the game.
```

WHAT YOU GET

```
[ UNLOCKED ]
» open the app folder
» get the real binary
» attach a debugger
» run Frida on the device

[ GOOD NEWS ]
» you jailbreak once
» older device = easier
» or skip it: emulate
```

STAGE 08

THE LAB — WHERE TO RUN IT

— OPTION 01 —

REAL DEVICE

> old iPhone or iPad
> jailbreak it once
> cheapest, most real

— OPTION 02 —

MAC + VPHONE

> Apple Silicon Mac
> vphone-cli, free
> virtual iPhone, no HW

— OPTION 03 —

CORELLIUM

> virtual iOS, cloud
> what the pros use
> costs real money

>> NO PHONE? YOUR M-SERIES MAC IS THE LAB.

STAGE 09

WHAT IS FRIDA

WHAT IT IS

```
$ frida -U -f com.the.game
```

[+] it injects into a
[+] running app
[+] you write JavaScript
[+] it runs inside the game

// think cheat engine,
// but scriptable.

WHAT IT DOES

[AT RUNTIME]
» hook any function
» read and write memory
» change the arguments
» change what it returns

[WHY IT WINS]
» you never patch a file
» you never re-sign it
» you change it live

STAGE 10

THE LOADOUT

$0

TOOLING COST

> all of it is free

1. **A DEVICE OR A MAC**
   jailbroken iPhone, or vphone-cli
2. **THE APP ITSELF**
   pulled clean off the device
3. **FRIDA**
   the runtime hooking engine
4. **CPP2IL + GHIDRA**
   to read what the binary does

STAGE 11

UNITY DEEP DIVE

70%+

OF MOBILE GAMES

RUN UNITY

> ONE TARGET. MANY GAMES.

ANATOMY

» Game logic = C# scripts

» Everything = GameObject + Components

» Lifecycle:

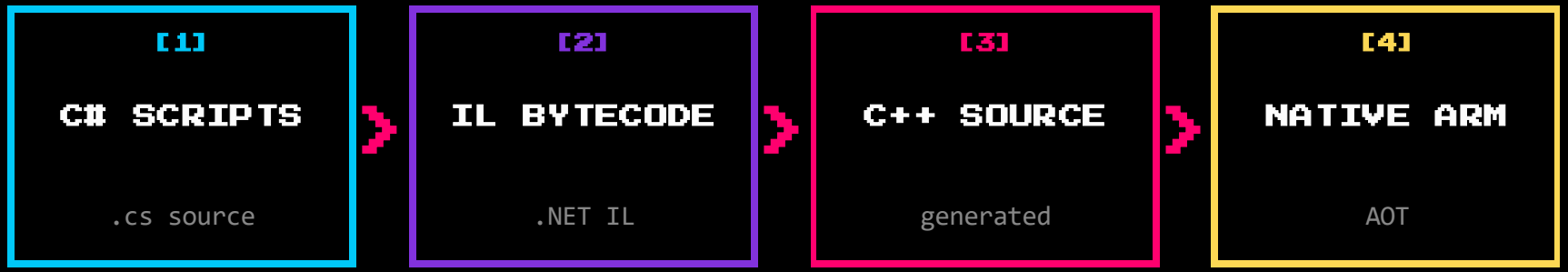
Awake() → Start() → Update()

» Health, speed, score = fields

>> KNOW STRUCTURE = KNOW ATTACKS

STAGE 12

IL2CPP PIPELINE



BUT...

>> IL2CPP converts .NET IL → C++ for AOT compilation

>> No .NET runtime — runs as native code

>> *Sounds secure...* but there's a catch.

STAGE 13

METADATA EXPOSURE

METADATA.DAT

// ships with every build

size: 8.1 MB
format: binary
stripped: never

contains:

+ class names
+ method names
+ field offsets
+ signatures

ATTACKER VIEW

```
$ Cpp2IL --metadata

[+] reconstructing hierarchy...
[+] 5687 classes recovered
[+] 71411 methods found
[+] field offsets resolved

// no guessing required.
// find TakeDamage()
// by name, not by luck.
```

STAGE 14

GAME STATE & LOGIC

PlayerStats.cs

```
class PlayerStats : MonoBehaviour
{
  int   health = 100;
  float speed  = 5.0f;
  int   score  = 0;

  void  TakeDamage(int amount);
  void  SetSpeed(float speed);
  void  HasWon();
}
```



EVERY FIELD
EVERY METHOD

= TARGET

STAGE 15

CLIENT-SIDE RISKS

! DANGER ZONE

[!] Health/damage local

[!] Movement speed client-side

[!] Win conditions local

[!] IAP validation skippable

// if it runs on the device,

// it can be analyzed,

// intercepted, modified.

- RULE 01 -

NEVER

TRUST THE CLIENT

// the golden rule

STAGE 16

STEP 1 - APP STRUCTURE

$ tree app

```
gamehacking101.app/
+-- gamehacking101
|   thin launcher (3 funcs)
+-- UnityFramework    119MB
|   ALL game logic here <-
+-- global-metadata.dat 8.1MB
    C# name database <-
```

NOTES

» Launcher just calls UnityFrameworkLoad()

» All C# → ARM64

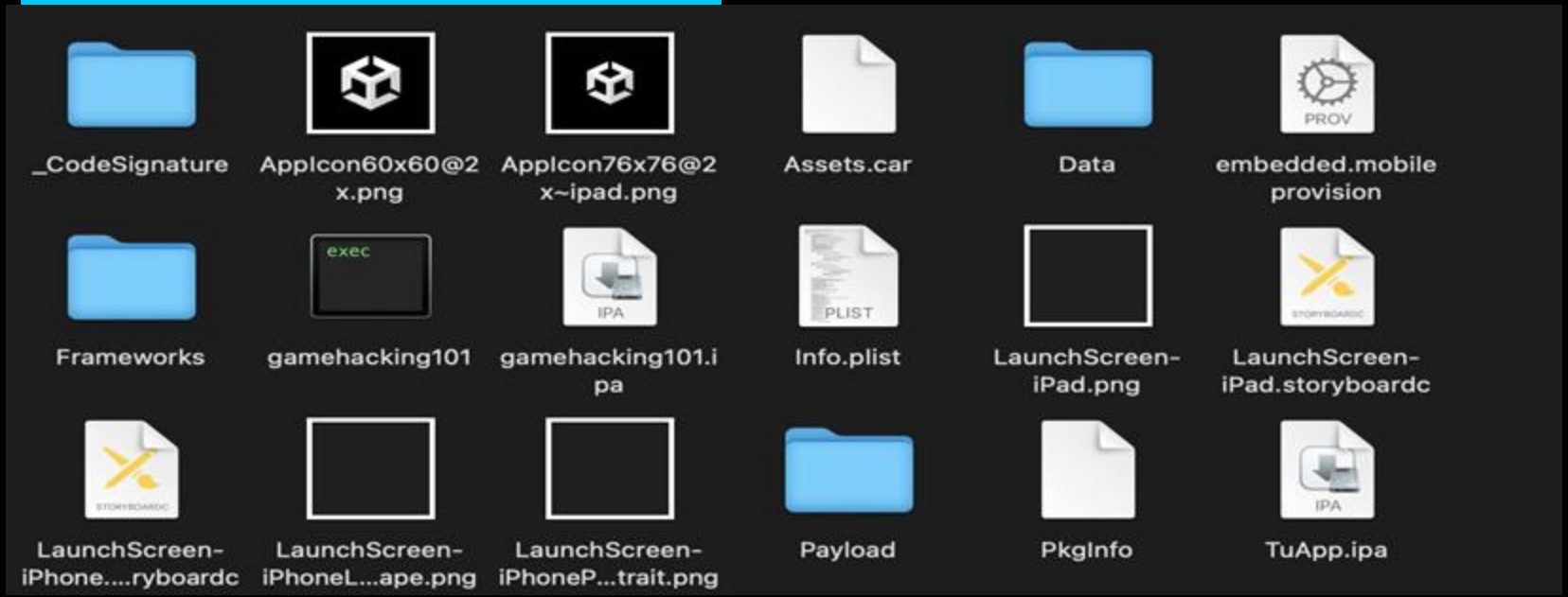
» metadata never stripped (Unity needs it @ runtime)

binary + metadata
= FULL PICTURE

STAGE 17

INSIDE THE .IPA

PAYLOAD / GAMEHACKING101.APP



RAW GHIDRA - PRE-CPP2IL

GHIDRA / TAKEDAMAGE

```
lVar1 = Box(DAT_04432b08,(long *)&currentHealthCopy);
uVar2 = _String_Format_mFB7DA489BD99F4670881FF50EC017BFB0A5C0987
                  (__stringLiteral992F9355FC85CA7618EC09864EB169C6ED06D194,boxedDamageAmount,
                   lVar1,0);
if (*(int *)(_Debug_t8394C7EEAECA3689C2C9B9DE9C7166D73596276F_il2cpp_TypeInfo_var + 0xe4) == 0)
{
  il2cpp_codegen_runtime_class_init
            (_Debug_t8394C7EEAECA3689C2C9B9DE9C7166D73596276F_il2cpp_TypeInfo_var);
  _Debug_Log_m87A9A3C761FF5C43ED8A53B16190A53D08F818BB(uVar2);
  currentHealth = *(int *)(playerStats + 0x24);
}
else {
  _Debug_Log_m87A9A3C761FF5C43ED8A53B16190A53D08F818BB(uVar2);
  currentHealth = *(int *)(playerStats + 0x24);
}
if (currentHealth < 1) {
  *(undefined4 *)(playerStats + 0x24) = 0;
  *(undefined1 *)(playerStats + 0x38) = 1;
```

! WTF

LOOKS SCARY RIGHT?

no class names.

no field names.

just offsets.

STAGE 19

STEP 2 - CPP2IL

PASS 1 - CS

```
$ Cpp2IL --diffable-cs

[+] Metadata v39 OK
[+] 71,411 methods
[+] 5,687 .cs files
[+] 69 assemblies
[+] total: 5027ms

$ done.
```

PASS 2 - ISIL

```
$ Cpp2IL --output-as isil

» ARM64 disasm per method
» simplified IR output
» exact virtual addrs
```

we now have:

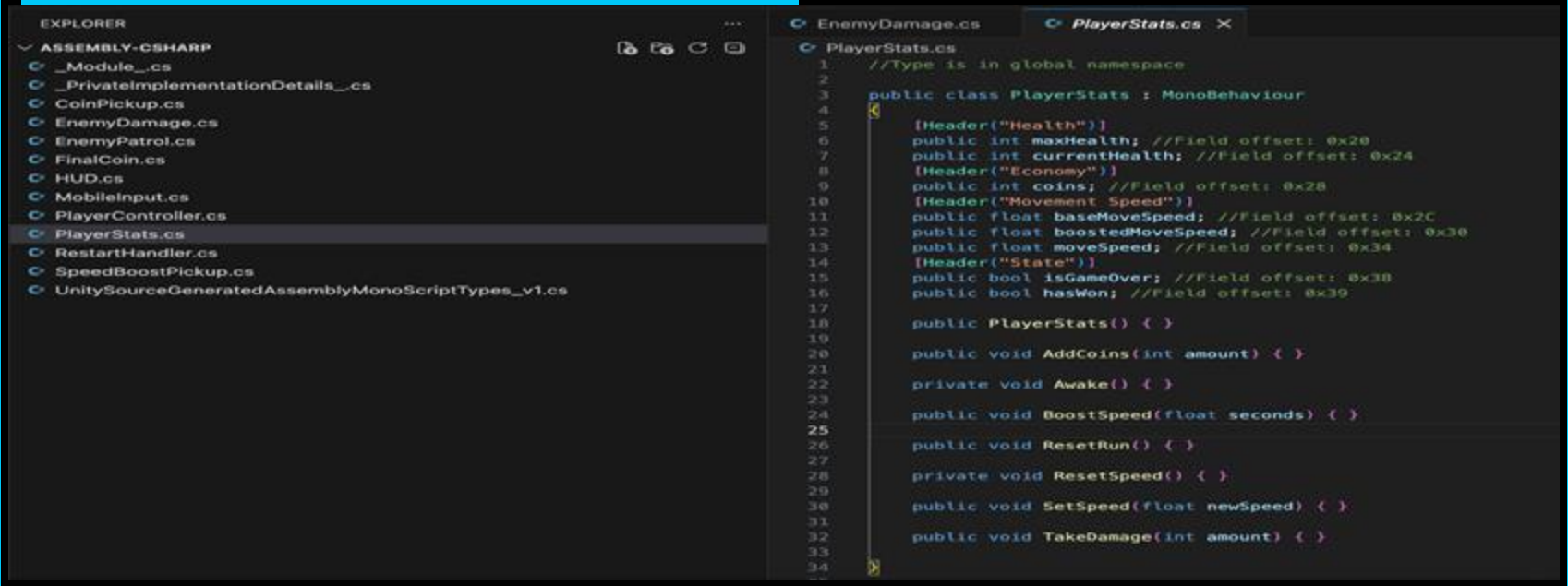
class names + **field offsets**
+ **signatures** + **addrs**

↳ github.com/SamboyCoding/Cpp2IL

STAGE 20

RECOVERED: PLAYERSTATS.CS

ASSEMBLY-CSHARP / PLAYERSTATS.CS



STAGE 21

PLAYERSTATS.TXT (ISIL)

ASSEMBLY-CSHARP / PLAYERSTATS.TXT

EXPLORER

ASSEMBLY-CSHARP

- CoinPickup.txt
- EnemyDamage.txt
- EnemyPatrol.txt
- FinalCoin.txt
- HUD.txt
- MobileInput.txt
- PlayerController.txt
- PlayerStats.txt
- RestartHandler.txt
- SpeedBoostPickup.txt
- UnitySourceGeneratedAssemblyMonoScriptTypes_v1.txt

PlayerStats.txt

PlayerStats.txt

```
143   Method: System.Void TakeDamage(System.Int32 amount)
144
145   Disassembly:
146       0x0135D9E4 SUB X31, X31, 0x40
147       0x0135D9E8 STP X22, X21, [X31 + 0x10]
148       0x0135D9EC STP X20, X19, [X31 + 0x20]
149       0x0135D9F0 STP X29, X30, [X31 + 0x30]
150       0x0135D9F4 ADD X29, X31, 0x30
151       0x0135D9F8 MOV X20, X1
152       0x0135D9FC MOV X19, X0
153       0x0135DA00 ADRP X21, 0x30CA000
154       0x0135DA04 LDRB W8, [X21 + 0x19A]
155       0x0135DA08 TBNZ X8, 0x0, 0x135DA38
156       0x0135DA0C ADRP X0, 0x2DDF000
157       0x0135DA10 ADD X0, X0, 0xF10
158       0x0135DA14 BL 0x1E524
159       0x0135DA18 ADRP X0, 0x2E15000
160       0x0135DA1C ADD X0, X0, 0xC00
161       0x0135DA20 BL 0x1E524
162       0x0135DA24 ADRP X0, 0x2E15000
163       0x0135DA28 ADD X0, X0, 0xC18
```

STAGE 22

STEP 3 - CLASS SKELETONS

PlayerStats

```
// fields recovered

maxHealth       // 0x20
currentHealth   // 0x24 <-
coins           // 0x28
moveSpeed       // 0x34
isGameOver      // 0x38
hasWon          // 0x39

methods:
TakeDamage, AddCoins,
BoostSpeed, SetSpeed,
ResetRun
```

PlayerController

```
// fields recovered

stats           // 0x20
rb              // 0x28
jumpForce       // 0x30
groundCheck     // 0x38

methods:
Update, IsGrounded

// bodies are { }
// Ghidra needed
```

STAGE 23

STEP 4 - GHIDRA ANALYSIS

ROSETTA STONE

Ghidra sees:

```
*(int*)(p + 0x24)
```

Cpp2IL says:

```
currentHealth  // 0x24
```

```
================
```

= read Ghidra
  output like
  original C#

$ ghidra.find()

» TakeDamage 0x0135D9E4
» AddCoins 0x0135D808
» BoostSpeed 0x0135F4A8
» Update 0x0135ED44
» IsGrounded 0x0135F0AC

>> target acquired:
TakeDamage

github.com/LaurieWired/
GhidraMCP

STAGE 24

STEP 5 - DECOMPILED

TakeDamage()

```
if (!isGameOver && !hasWon) {
  health -= damage;
  if (health < 1) {
    health = 0;
    isGameOver = true;
  }
}
// observations:
» guard checks isGameOver / hasWon
» local subtract — no server!
» game over @ health < 1
```

! ATTACK VECTOR

MAKE

TakeDamage()

DO NOTHING

+ player never loses health
+ isGameOver = false

= INVINCIBLE

STAGE 25

GHIDRA - FULL DECOMPILE

GHIDRA / TAKEDAMAGE FULL

Decompile: _PlayerStats_TakeDamage_m9748E4A4AEA86825EDC2DB858EEF6D9BF31AA2D5 - (UnityFramework)

```
void _PlayerStats_TakeDamage_m9748E4A4AEA86825EDC2DB858EEF6D9BF31AA2D5
               (long playerStats,int damageAmount)

{
  long boxedDamageAmount;
  long lVar1;
  undefined8 uVar2;
  undefined4 currentHealthCopy;
  int damageAmountCopy;
  int currentHealth;

  if ((PlayerStats_TakeDamage_m9748E4A4AEA86825EDC2DB858EEF6D9BF31AA2D5::s_Il2CppMethodInitialized &
      1) == 0) {
    il2cpp_codegen_initialize_runtime_metadata
              ((GlobalMetadata *)
               &_Debug_t8394C7EEAECA3689C2C9B9DE9C7166D73596276F_il2cpp_TypeInfo_var);
    il2cpp_codegen_initialize_runtime_metadata
              ((GlobalMetadata *)&__stringLiteral38A080254DF94B38A818FB27B65D41CD20CA9458);
    il2cpp_codegen_initialize_runtime_metadata
              ((GlobalMetadata *)&__stringLiteral992F9355FC85CA7618EC09864EB169C6ED06D194);
    PlayerStats_TakeDamage_m9748E4A4AEA86825EDC2DB858EEF6D9BF31AA2D5::s_Il2CppMethodInitialized = 1;
  }
  if (((*(byte *)(playerStats + 0x38) & 1) == 0) && ((*(byte *)(playerStats + 0x39) & 1) == 0)) {
    *(int *)(playerStats + 0x24) = *(int *)(playerStats + 0x24) - damageAmount;
    damageAmountCopy = damageAmount;
    boxedDamageAmount = Box(DAT_04432b08,(long *)&damageAmountCopy);
    currentHealthCopy = *(undefined4 *)(playerStats + 0x24);
    lVar1 = Box(DAT_04432b08,(long *)&currentHealthCopy);
    uVar2 = _String_Format_mFB7DA489BD99F4670881FF50EC017BFB0A5C0987
                      (__stringLiteral992F9355FC85CA7618EC09864EB169C6ED06D194,boxedDamageAmount,
                       lVar1,0);
    if (*(int *)(_Debug_t8394C7EEAECA3689C2C9B9DE9C7166D73596276F_il2cpp_TypeInfo_var + 0xe4) == 0)
    {
      il2cpp_codegen_runtime_class_init
                (_Debug_t8394C7EEAECA3689C2C9B9DE9C7166D73596276F_il2cpp_TypeInfo_var);
      _Debug_Log_m87A9A3C761FF5C43ED8A53B16190A53D08F818BB(uVar2);
      currentHealth = *(int *)(playerStats + 0x24);
    }
    else {
      _Debug_Log_m87A9A3C761FF5C43ED8A53B16190A53D08F818BB(uVar2);
      currentHealth = *(int *)(playerStats + 0x24);
```

STAGE 26

STEP 6 — 4 BYTES TO GOD

BEFORE

FF 03 01 D1 **SUB**

sp, sp, #0x40

AFTER

C0 03 5F D6 **RET**

<- return immediately

EFFECT

» TakeDamage() is called normally by the engine

» RET returns immediately — zero logic executes

» health never decreases » isGameOver never triggers

» @ 0x0135E908 = player is invincible.

STAGE 27

THE DIFF - BEFORE VS AFTER

BEFORE.BIN

```
0135d9e4 ff 03 01 d1    sub     sp,sp,#0x40
0135d9e8 f6 57 01 a9    stp     x22,x21,[sp, #0x10]
0135d9ec f4 4f 02 a9    stp     x20,x19,[sp, #0x20]
0135d9f0 fd 7b 03 a9    stp     x29,x30,[sp, #0x30]
0135d9f4 fd c3 00 91    add     x29,sp,#0x30
0135d9f8 f4 03 01 aa    mov     x20,damageAmount
0135d9fc f3 03 00 aa    mov     x19,playerStats
0135da00 55 86 01 d0    adrp    x21,0x4427000
0135da04 a8 6a 46 39    ldrb    w8,[x21, #0x19a]=>PlayerSta
0135da08 88 01 00 37    tbnz    w8,#0x0,LAB_0135da38
0135da0c e0 6e 01 f0    adrp    playerStats,0x413c000
```

AFTER.BIN

```
0135d9e4 c0 03 5f d6    ret
0135d9e8 f6 57 01 a9    stp     x22,x21,[sp, #0x10]
0135d9ec f4 4f 02 a9    stp     x20,x19,[sp, #0x20]
0135d9f0 fd 7b 03 a9    stp     x29,x30,[sp, #0x30]
0135d9f4 fd c3 00 91    add     x29,sp,#0x30
0135d9f8 f4 03 01 aa    mov     x20,damageAmount
0135d9fc f3 03 00 aa    mov     x19,playerStats
0135da00 55 86 01 d0    adrp    x21,0x4427000
0135da04 a8 6a 46 39    ldrb    w8,[x21, #0x19a]=>PlayerStats_
0135da08 88 01 00 37    tbnz    w8,#0x0,LAB_0135da38
0135da0c e0 6e 01 f0    adrp    playerStats,0x413c000
```

STAGE 28

VERIFY & DEPLOY

VERIFY

```
$ xxd -s 0x135E908 -l 4
0135e908: c0 03 5f d6
```

[+] RET opcode: OK
[+] size: 119MB (same)
[+] bytes changed: 4
[+] MD5: preserved*

DEPLOY

[JAILBROKEN]
» replace binary in app
» repack IPA
» no re-sign needed

[TROLLSTORE]
» scp .ipa to device
» install via TrollStore
» permanent sign, no expiry


9:41a.m. Tue 9 Jan
100%
Clock
Weather
Stocks
Find My
Translate
Shortcuts
iTunes Store
Contacts
Voice Memos
Measure
Magnifier
Passwords
Photo Booth
Freeform
Health
Calculator
Tips
Proton VPN
9
Sileo
palera1n
TrollStore Lite
Filza
Popular
Bee
gamehacking101
TUE
9

STAGE 29

THE TOOLBOX - FRIDA

- TOOL -

FRIDA

Dynamic
instrumentation.
Inject JS, hook funcs

- TOOL -

IL2CPP-BRIDGE

Unity-specific.
Reads metadata,
resolves classes

- TOOL -

FRIDA-COMPILE

TypeScript build.
Complex scripts
with type safety

STAGE 29B

WHY FRIDA-COMPILE

THE PROBLEM

```
$ frida -l godmode.js

[!] import il2cpp-bridge
[!] needs ESM bundling
[!] frida -l = ONE file
[!] no node_modules

// raw frida can't
// resolve the import.
```

THE FIX

```
[ FRIDA-COMPILE ]
» bundles bridge+script
» single .c.js output
» TS type-safety
» no runtime deps

[ THE FLOW ]
» npm i frida-il2cpp-bridge
» frida-compile out.c.js
» frida -l out.c.js
```

L1V3 D3M0S

- HOLD ON TO YOUR IPADS -

PRESS A TO CONTINUE

STAGE 30

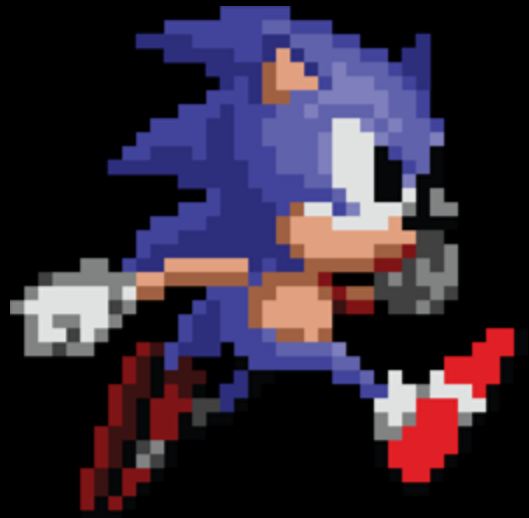
DEMO - INFINITE SPEED

speed_hack.ts

```
// modify speed @ runtime

[1] find speed field offset
[2] hook SetSpeed() + reset
[3] persist via onLeave hook
    → re-apply every frame

>> result: permanent speedboost.
```



- GOTTA GO FAST -

DEMO - INFINITE FIRE

infinite_fire.ts

```
// hook PlayerAttack.Update()

[1] find PlayerAttack class
[2] set fireCooldown = 0
[3] call Fire() each frame
[4] on main thread (safe)

>> result: endless laser stream.
```



- PEW PEW PEW -

STAGE 32

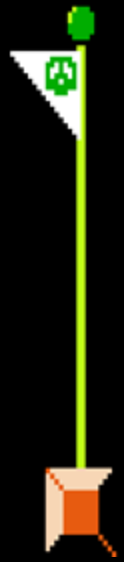
DEMO - INSTANT WIN

instant_win.ts

```
// trigger win directly

[1] find hasWon() method
[2] get ref to player instance
[3] call hasWon() from Frida
[4] game triggers victory screen

>> result: skip entire level instantly.
```



- FLAG CAPTURED -

STAGE 33

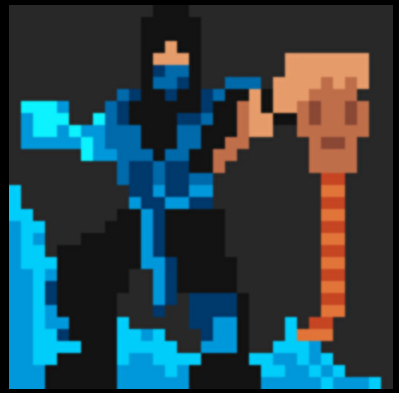
DEMO — ARMAGEDDON

armageddon.ts

```
// god + timer + boss @ once

[1] god: TakeDamage() = 0
[2] freeze timeLeft (no timeout)
[3] boss: dmg = 999999
[4] 3 hooks, 1 script

>> result: total domination.
```



— ARMAGEDDON —

STAGE 34

DEFENSE - BONUS STAGE

01

SERVER-SIDE VALIDATION

validate critical game state on server

02

INTEGRITY CHECKS

detect tampering, JB, debuggers

03

ANTI-TAMPERING

obfuscate metadata, encrypt strings

04

DEFENSE IN DEPTH

no single layer is enough

GAME OVER

FOR iOS SECURITY ASSUMPTIONS

iOS IS NOT UNHACKABLE

if game logic runs on the client,
it can be **analyzed**, **instrumented**, and **modified**.

– THANKS FOR PLAYING –