Dark Souls III P2P Networking Exploits and Vulnerabilities

Contents

Preface

This document focuses on PC cheating and exploits in Dark Souls III (ver. 1.15 : reg. 1.35).

Cheating is nothing new to Dark Souls, and Dark Souls III is no exception. Over the years since Dark Souls: Prepare to Die Edition's PC release on Steam in 2011 the game's foundation code for networking hasn't evolved much; All online games on PC suffer from cheating problems to varying extents, but I believe that this point makes Dark Souls much more attractive to cheaters as the reverse engineering knowledge is the accomplishment of over a decade of hard work by talented individuals.

This wealth of information gives rise to amazing modding projects, which completely reimagine the game with new bosses, weapons, and event scripts by truly dedicated and inspiring mod authors. The modding community for the Dark Souls series is vast and active, with thousands of members worldwide and millions of people who make use of these mods over time.

The flip side to this, is that the malicious modding - Cheating - communities have also made use of this information. You only have to look back to the days of Dark Souls: Prepare to Die Edition in 2011 to see that the most prolific cheaters would use instant curse weapons on other players to make their experience unusually difficult, or scripts which would kill important NPCs like Andre of Astora making a player unable to upgrade their weapons and armour for the duration of their play through. These cheaters - while annoying - were only the start of the future generations of cheaters who would make use of and develop more complex and damaging tools, all in an effort to ruin other player's online experience.

At the time of writing this document in early 2022, the state of Dark Souls III's online experience is nightmarish. Cheats that soft-lock save files (e.g. instantly sending another player to new game+) are publicly available and widely abused by thousands, there has recently been a public demonstration of remote code execution - one of the most severe networking vulnerabilities around - performed solely using Dark Souls III's "Nexus Revolution Session Result" interface which is normally used to coordinate invasions between host and guest player.

This document will examine each p2p packet "ID" (the number value assigned to the p2p packets in the in-game code) and document vulnerabilities and exploits that myself and a few others have found throughout the past few years. Where possible, I have tried to label the function of the packet, however without access to the source code I can not reliably tell what each packet's true function is.

I have also only listed exploits that I know of, and feel like should be addressed ASAP – Minor or harmless ones have been omitted.

I've put two asterisks (**) next to the most pressing issues that may be an entry-point into RCE.

P2P Packets

1 + 24 - “Player frame update - Location” **

Please note, the data in this packet is packed using the ‘NetPackingVector’ for this packet and animation packets (24). This makes it difficult to map in a structure, so I have mapped out how the unpacked packet might look:

```
struct SPackedDataCoordinateSeg {
      float fX = 0;           //Used in coordinate packets only
      float fZ = 0;           //Used in coordinate packets only
      float fY = 0;           //Used in coordinate packets only
      float fR = 0;           //Used in coordinate packets only
};

struct SPacketDataCoordinateLargeSeg {
      float fX = 0;           //Used in coordinate packets only
      float fZ = 0;           //Used in coordinate packets only
      float fY = 0;           //Used in coordinate packets only
      float fR = 0;           //Used in coordinate packets only
      uint32_t iUnk1 = 0;
      uint8_t bObject = 0;
      uint32_t iUnk2 = 0;
};

struct SPacketDataValueSeg {
      uint32_t iUnk1 = 0;
      uint32_t iUnk2 = 0;
      uint16_t wHealth = 0;
      uint16_t wMaxHealth = 0;
      uint32_t iUnk3 = 0;
      uint32_t iUnk4 = 0;
};

struct SHavokBehAnim {
      uint32_t pAnimations[128];
};

#pragma pack(push,1)
struct SPackedSmallAnimation {
      uint8_t bAppend = 0;
      short wBeheaviourVariationID = 0;
};
struct SPackedMediumAnimation {
      uint16_t wAppend = 0;
      short wBeheaviourVariationID = 0;
};
struct SPackedLargeAnimation {
	uint32_t iAppend = 0;
	short wBeheaviourVariationID = 0;
};
struct SPackedHugeAnimation {
	uint64_t qAppend1 = 0;
	uint64_t qAppend2 = 0;
	short wBeheaviourVariationID = 0;
};
#pragma pack(pop)

struct SPacketDataSize {
	// Can only be used in location packets
	bool bIsObjStandTypeLocationPacket;
	SPackedDataCoordinateSeg sLocation;
	SPacketDataCoordinateLargeSeg sLocationLarge;
	SPacketDataValueSeg sPadSyncValues;

	// Should only be used in animations, but can occur in coordinates
(size = t)
	uint8_t bHavokBehAnimCount;
	uint8_t bSmallAnimCount;
	uint8_t bMediumAnimCount;
	uint8_t bLargeAnim1Count;
	uint8_t bLargeAnim2Count;
	uint8_t bHugeAnimCount;
	SHavokBehAnim sHavokBehAnim;
	SPackedSmallAnimation sSmallAnim[70];
	SPackedMediumAnimation sMediumAnim[40];
	SPackedLargeAnimation sLargeAnim1[70];
	SPackedLargeAnimation sLargeAnim2[140];
	SPackedHugeAnimation sHugeAnim[70];
};
```

This packet is sent regularly and contains information regarding whether the player is standing on stable ground, health, maximum health, location, idle animation, and current animation. I have put information for both packet ID 1 and packet ID 24 in this section as they are both practically the same. The packet parsing function for packet 1 is at sub_0x777540, and the packet parsing function for packet 24 is at sub_0x776730.

Insecure unpacking function

The unpacking function for these two packets is loc_0x77CE00. I can’t say for sure I understand exactly how it works, however I have managed to re-create it in the ‘Blue Sentinel’ mod for data integrity purposes.

This is a wholly insecure function, and the buffer can overrun if a malicious user sends a packet which unpacks to a size greater than what the function is able to handle (1200 bytes maximum).

Signalling NaN/Inf

If a guest player in the session moves their location to out of bounds coordinates (e.g X/Y/Z), specifically to NaN, Inf+, Inf-, or FLT_MAX/ FLT_MIN, this will cause all connected players to crash.

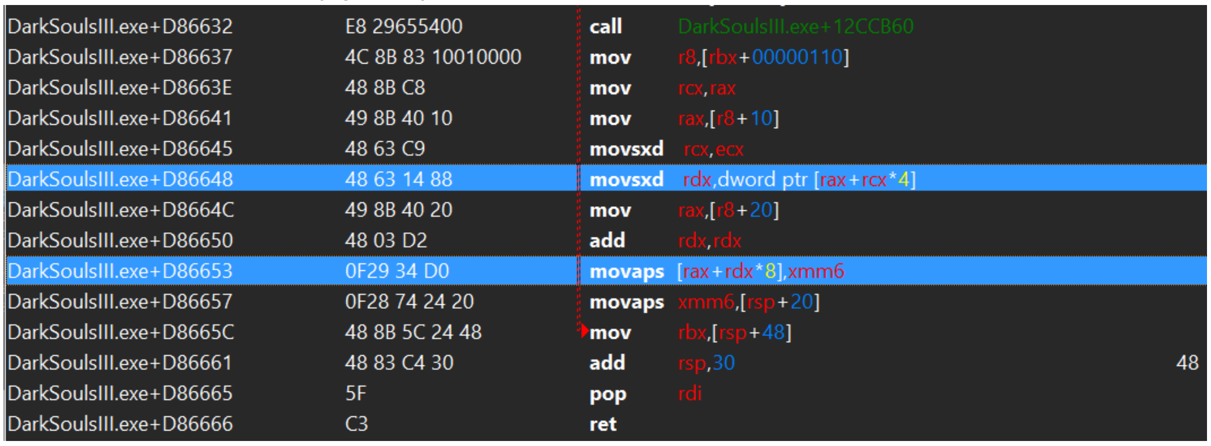
This is a recurring theme in this game. You can also signal NaN in:

- Spawn point coordinates (sent from the host to guests on join)
- NPC summon packets
- Grab packets

This can be addressed by either fixing the underlying issue with these coordinates, or by simply preventing players from going to these locations.

Large animation offsets

This is difficult to explain, so I apologise. Instead, I will give a specific data example. This is code at sub_0xD86580 (figure 1):



```
DarkSoulsIII.exe+D86632    E8 29655400          call     DarkSoulsIII.exe+12CCB60
DarkSoulsIII.exe+D86637    4C 8B 83 10010000    mov      r8,[rbx+00000110]
DarkSoulsIII.exe+D8663E    48 8B C8             mov      rcx,rax
DarkSoulsIII.exe+D86641    49 8B 40 10          mov      rax,[r8+10]
DarkSoulsIII.exe+D86645    48 63 C9             movsxd   rcx,ecx
DarkSoulsIII.exe+D86648    48 63 14 88          movsxd   rdx,dword ptr [rax+rcx*4]
DarkSoulsIII.exe+D8664C    49 8B 40 20          mov      rax,[r8+20]
DarkSoulsIII.exe+D86650    48 03 D2             add      rdx,rdx
DarkSoulsIII.exe+D86653    0F29 34 D0           movaps   [rax+rdx*8],xmm6
DarkSoulsIII.exe+D86657    0F28 74 24 20        movaps   xmm6,[rsp+20]
DarkSoulsIII.exe+D8665C    48 8B 5C 24 48       mov      rbx,[rsp+48]
DarkSoulsIII.exe+D86661    48 83 C4 30          add      rsp,30                              48
DarkSoulsIII.exe+D86665    5F                   pop      rdi
DarkSoulsIII.exe+D86666    C3                   ret
```

And here is some data which the attacker will send (figure 2):

```
struct SMaliciousAnimationPacket {
      uint8_t bSignalFlags = 2;
      uint8_t bFlags_01 = 20;
      uint8_t bFlags_02 = 1;
      int iMaliciousOffset = 1000000000;
      uint16_t wFlags_04 = 0;
      uint8_t bFlags_03 = 1;
      uint64_t qToWrite= 0xDEADBEEF;
};
```

In this demonstration, the animation packet parsing function will process the packet data with no data checks at all, and eventually we reach the code in figure 1. The offset in ‘iMaliciousOffset’ will be placed into ‘rdx’, then multiplied by 16 before being written to an array relating to the player’s hkbVariableValueSet.

The value in ‘xmm6’ is the ‘qToWrite’ value. This means that a malicious entity could manipulate this enormous custom out of bounds write (+/- 34,359,738,352 bytes outside of the original array) to potentially access executable memory. As Dark Souls III has no write protection on executable code, it would be easy for an attacker to write custom instructions and **re-direct execution to malicious code**.

This has some drawbacks, the attacker cannot know where executable memory may be as the memory is unreliable, however they can try as many times as they like and could potentially keep trying until they manage to harm another user. This has serious security implications.

Fixed vector game engine panic

In addition to the issues listing above, sending a quantity of animations greater than what the packet is able to handle will cause the game engine to panic, with the error message:

```
out of memory.
N:\FDP\Source\Library\Dantelion2\dist_win64_vc2012\include_utf8\dantelio
n2/Core/Util/DLFixedVector.inl
```

There are different but very similar functions for these, one such is sub_0x77DB00. This is rather late to be dealing with invalid data, and ideally a size verification would filter out these packets before it reached this point of forcing the game to panic.

4 - “WorldChrSync: Location”

```
struct SEnemyLocationUnit {
	uint32_t iEntityWho;
	uint32_t iMovementSync; // X, Z, and Y in 1 4-byte value
	uint32_t iRotationSync; // ?
	uint32_t iUnk3;
	uint32_t iUnk4;
	uint32_t iUnk5;
	uint32_t iUnk6;
};

struct SPacket4 {
	SEnemyLocationUnit sUnit[8];
};
```

The purpose of this packet is to synchronise enemy movement, position, and trajectories between game clients for a seamless online experience. The packet parsing function is located at: sub_0x8A8AC0.

Unfair boss / NPC deaths

As there are no position checks, and the location of the enemies is controlled by the client who is standing nearest to them, malicious users can teleport enemies on their game to out of bounds areas / kill zones and instantly kill the host area boss or important game NPCs this way.

8 - “Player param data”

Below is my estimation on the structure of this packet:

```
struct SPacket8 {
	int iPlayerNumber;
	int iPlayerID;
	int iCharID;
	int iMultiPlayCount;
	int iCoopPlaySuccessCount;
	int iPlayerRankS;
	int iPlayerRankA;
	int iRosariaRankingPoints;
	int iUnk2;
	char bPlayerRankC;
	char bRank_BladeOfTheDarkmoon;
	char bRank_WarriorOfSunlight;
	char bRank_MoundMakers;
	char bRank_SpearsOfTheChurch;
	char bRank_Rosaria;
	char bRank_WatchdogsOfFarron;
	char bRank_AldrichFaithful;
	unsigned int iVig;
	unsigned int iAtn;
	unsigned int iEnd;
	unsigned int iStr;
	unsigned int iDex;
	unsigned int iInt;
	unsigned int iFth;
	unsigned int iLck;
	unsigned int iNSt; //Empty stat slot
	unsigned int iVit;
	unsigned int iSlv;
	char bInvadeType;
	char bIsEmber;
	char bUnk4;
	char bRank_BlueSentinels;
	wchar_t wcCharName[16]; //0x5C
	short wZeroExtend1;
	wchar_t wcSteamHexID[16]; //0x7E
	short wZeroExtend2;
	char cFacePt[4]; //0xA0
	char pFaceData[0xF0];
	char bWeaponUpgradeLv; //
	char bRegion;
	char bQuickMatchTeamType;
	short wPack;
```

```
      float fUnk1;
      short wUnk1;
      char bUnk7;
      char bPack;
      short wUnk2;
      char bUnk8;
};
```

This packet is used to send basic player param structures of players to other clients in the session. This includes information such as stats, face data, and covenant progress. The parsing function for this packet is sub_0x778270.

Duplicate player data packets

In the game, it seems like a game client will send a player data packet whenever passing through a loading screen. This means that in situations like the duelling arena, players will inevitably send updated player data packets to one another under this normal game behaviour.

However, cheaters can send duplicate player data packets to rapidly change their face data, skin colour, and name to be as obnoxious as possible. This isn't especially harmful, but the game shouldn't accept additional player data packets from guests who are already connected.

PlayerID spoof

PlayerID is the value used by the master server for matchmaking purposes. It is assigned per-Steam account and is unique to that player. The PlayerID sent in this packet is used directly in the `RequestNotifyDisconnectSession` server packet which is sent if a player has abnormally disconnected. This means that, in theory, a player can change their PlayerID in this packet to be that of someone else's, and then repeatedly disconnect abnormally and cause the framed PlayerID to be "shadow-banned" by the matchmaking server for a period of time. The result of this is described in the exploit write-up for this packet.

I recommend removing the RequestNotifyDisconnectSession function, as it is too abusable.

Abnormally long names

The character name is contained within this packet. In-game, it is limited to 16 wide-chars, however you can edit the packet to have a name that goes far beyond that (up to 48 characters). This can cause various glitches with text and invasion messages.

I recommend capping this at 16 wide characters in the parsing function, which currently does not happen.

Face data crashes

By rapidly changing face data, players can crash other players by referencing an invalid face file.

This would be resolved by rejecting additional player data packets received from players.

11 - “Set character type”

Below is my estimation on the structure of this packet:

```
struct SPacket11 {
      unsigned int iChrType;
};
```

This packet has no underlying crashes as far as I am aware, however it should only be accepted **once** per game session. The only exception to this, is when a hollow host restores their humanity which should synchronise this accordingly. The parsing function for this packet is at sub_0x778180

As it can be repeatedly sent, players abuse this to change their character and team type to friendly phantoms when they are dark spirits which causes them to be invisible from the host but still be able to deal damage themselves.

There are also no sanity checks inside this packet. If a player sets their character type to an abnormal or non-existant value, the parsing function will accept this value which may cause unintended game behaviour (e.g. invisible players)

12 - “Player equipment data”

Below is my estimation on the structure of this packet:

```
struct SPacket0x0C {
	uint32_t iPlayerNumber;
	uint8_t bGender;
	uint8_t bDeceased;
	uint8_t bVoiceType;
	uint8_t bVowType;
	uint8_t bNATType;
	uint32_t iWeaponL1;
	uint32_t iWeaponR1;
	uint32_t iWeaponL2;
	uint32_t iWeaponR2;
	uint32_t iWeaponL3;
	uint32_t iWeaponR3;
	uint32_t iArrowSlot1;
	uint32_t iBoltSlot1;
	uint32_t iArrowSlot2;
	uint32_t iBoltSlot2;
	uint32_t iArrowSlot3;
	uint32_t iBoltSlot3;
	uint32_t iEquipmentHead;
	uint32_t iEquipmentTorso;
	uint32_t iEquipmentHands;
	uint32_t iEquipmentLegs;
	uint32_t iEquipmentHair;
	uint32_t iRingSlot1;
	uint32_t iRingSlot2;
	uint32_t iRingSlot3;
	uint32_t iRingSlot4;
	uint32_t iCovenantSlot;
	uint32_t iForceItemInvetoryID;
	uint32_t iUnused1;
	uint32_t iUnused2;
	float fHeadSize;
	float fBodySize;
	float fWaistSize;
	float fRArmSize;
	float fRLegSize;
	float fLArmSize;
	float fLLegSize;
	uint32_t iUnused3;
};
```

This packet is used to send basic player equipment structures of players to other clients in the session. This includes information such as equipment, gender, and body proportions. The parsing function for this packet is at sub_0x776E80

Gender changes / crash

There is no case where gender changes should be synchronised online and changing to a gender other than male or female in quick succession will cause connected clients to crash.

I recommend that this is fixed by not synchronising gender in this packet, and moving it to another packet which can be sent as a one-off if possible.

Abnormal body proportions

Body proportions tell the game how large to make different aspects of the player's model (e.g. head size, waist size, arm size, etc.). This is an uncapped float value, meaning that extremely low values will cause players to appear invisible, and extremely high or nonsensical values (e.g. FLT_NAN / FLT_INF) cause graphical lag and glitches which interfere with gameplay.

I recommend that these values be capped between what the game accepts as normal slider values in the character creation menu.

14 - “MsgMapList”

Below is my estimation on the structure of this packet:

```
struct SPacket14 {
	uint32_t iMsgID;
	int iParam1;
	int iParam2;
	int iParam3;
	int iParam4;
};
```

The purpose of this packet is to synchronise “MsgMapList” events in the session. The main issue with this packet is that there is no sanity check to ensure that the MsgMapList is one that should be sent to others in the multiplayer session. The parsing function for this packet is at sub_0x4863A0

Sending blocked network MsgMap events

Seeing as the check for whether an event should be sent over the network or not is only done on the client who sends the packet, this allows for cheaters to abuse functionality and cause MsgMapEvents that should not normally be allowed, for example ’LuaWarpEvent_01’ which will cause players who receive this packet to be warped to an invalid map. This results in the player free-falling in a black void over and over again.

I recommend that this is addressed by adding a check on the receiving game to check whether or not the MsgMapList event *should* be sent over the network in the first place.

Running illegal / cut-content MsgMapEvents

This is specific to only a few specific events. For example ‘OnThxKickOut’ with a ‘iParam2’ entry of “2” will bring up a pop-up box informing a player that they have returned home. These message boxes can stack, and if more than 1 is present on the screen then the game will crash when selecting “OK” as the message system is from Dark Souls 1.

PlayerIns identifier spoofing

For some events (e.g. ‘OnEvent_4010_1’) the ‘iParam4’ entry references a unique number to identify which player in the session will be affected by the event, and ‘iParam3’ will be an animation ID to run on that player. A malicious cheater can abuse this to play a variety of animations on other players, such as warping, gestures, NPC gestures, and frozen idle animations.

15 - “Complete event”

Below is my estimation on the structure of this packet:

```
struct SPacket15 {
      int iGameEvent;
};
```

This is a bizarre packet, which the game will never normally send under normal circumstances. It causes any event flag provided in `iGameEvent` to be automatically completed. This is the cause of the “NG+ cheat” which caused so much havoc in 2020/2021. The parsing function for this packet is at sub_0x777CB0

The pseudocode might look something like this:

```
int event_id = sub_0x777CB0();
if (event_id != -1) {
   SetGameFlag(event_flag_manager, event_id, true);
};
```

NG+ cheat

This was done by simply sending event ID “30” as the ‘iGameEvent’ in this packet to other players. Doing this will immediately progress that player to the next new game cycle. If it is a guest player who is affected, they will be unable to re-obtain the coiled sword and will be soft-locked in the firelink shrine.

Alternatively, a malicious cheater can set DLC game flags for a player who does now own the DLC, leading to a potential matchmaking restriction or game progression softlock.

I recommend this packet be removed completely, as it has no purpose.

I made a video demonstrating how this cheat can be used on invaders in a video linked here:

https://www.youtube.com/watch?v=H-Iz3hbESIc

16 - “Host world event data” **

Below is my estimation on the structure of this packet:

```
struct SEventInitHeaderSegment {
	uint32_t iOffset; // 0x1C0 on first, 0x300 on second and third, and 
0x80 on fourth
	uint16_t wAreaID; // Plus 1
	uint16_t w;         // Either 1, 2, 3, or 6
	uint32_t i;         // Segment size + ((SegmentCount*3*)*4) + 
([iOffset+CurrentSegment] * CurrentSegment)
};

struct SPacket0x10 {
	char pNexusString[4];
	uint16_t wUnk1;
	uint16_t wSegmentCount; //MUST be 4
	uint32_t iSegmentSize;  //MUST be 12
	SEventInitHeaderSegment sSegment[4];
};
```

The parsing function for this packet is sub_0x4C5470

REMOTE CODE EXECUTION

A few years ago, I found a severe security vulnerability in DARK SOULS III’s peer to peer netcode. I reported this in early 2020. Further to this, I have identified that the same exploit exists in the following FROMSOFTWARE titles:

- DARK SOULS: Prepare to Die Edition
- DARK SOULS: Remastered
- DARK SOULS III

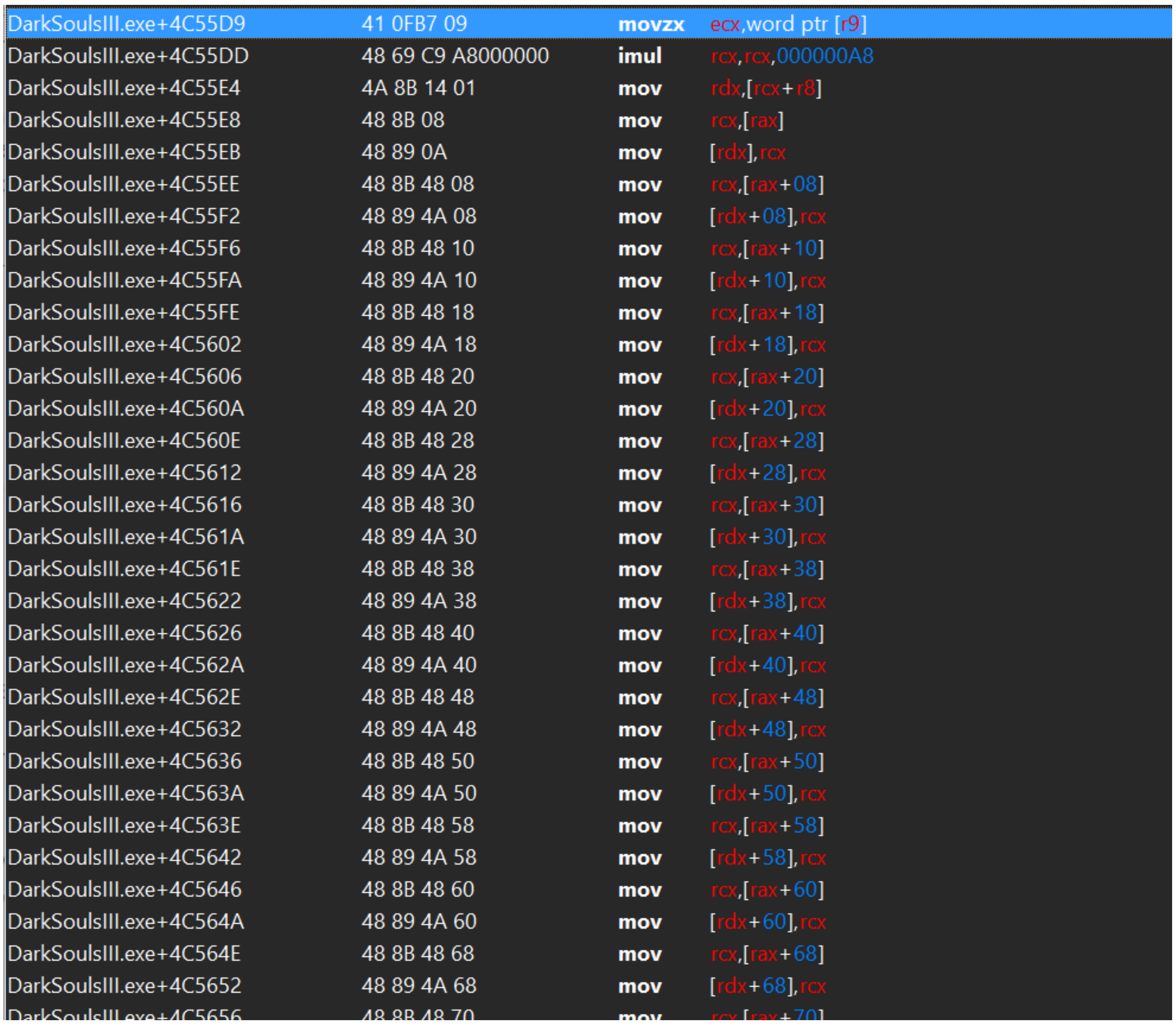
The exploit relies on the lack of bounds checking when accessing pointers to the “SprjEventFlagMan” class upon joining a session.

Using the exploit described below, an attacker is able to host a session and send modified packets which cause an out-of-bounds write over executable memory with attacker-defined instructions. This is a severe security vulnerability, as if the attacker is able to overwrite code that is being executed, they are able to have their custom code run almost immediately.

I recommend this issue is fixed as soon as possible. Although this exploit is not well known, or seen in the wild (as far as I know) the ever evolving cheating situation in the game means it is only a matter of time before players are at risk of being compromised.

The core issue lies inside the `area_id` portion of the `event_init_segment` struct. This 2-byte value is used to dereference a pointer in the “SprjEventFlagMan” class which would normally be used to copy an array of game event flags into this area. However, **this can be manipulated**, and there are **no bounds checking.** This means that a bad actor can write to dereferenced memory way outside the scope of the intended function and copy any array of bytes that they want to into this normally inaccessible memory.

Should this dereferenced happen to point to executable memory, the game will copy the array sent by the back actor over the top of the executable memory, allowing for them to directly inject code into the game of the victim.



```
DarkSoulsIII.exe+4C55D9   41 0FB7 09          movzx   ecx,word ptr [r9]
DarkSoulsIII.exe+4C55DD   48 69 C9 A8000000   imul    rcx,rcx,000000A8
DarkSoulsIII.exe+4C55E4   4A 8B 14 01         mov     rdx,[rcx+r8]
DarkSoulsIII.exe+4C55E8   48 8B 08            mov     rcx,[rax]
DarkSoulsIII.exe+4C55EB   48 89 0A            mov     [rdx],rcx
DarkSoulsIII.exe+4C55EE   48 8B 48 08         mov     rcx,[rax+08]
DarkSoulsIII.exe+4C55F2   48 89 4A 08         mov     [rdx+08],rcx
DarkSoulsIII.exe+4C55F6   48 8B 48 10         mov     rcx,[rax+10]
DarkSoulsIII.exe+4C55FA   48 89 4A 10         mov     [rdx+10],rcx
DarkSoulsIII.exe+4C55FE   48 8B 48 18         mov     rcx,[rax+18]
DarkSoulsIII.exe+4C5602   48 89 4A 18         mov     [rdx+18],rcx
DarkSoulsIII.exe+4C5606   48 8B 48 20         mov     rcx,[rax+20]
DarkSoulsIII.exe+4C560A   48 89 4A 20         mov     [rdx+20],rcx
DarkSoulsIII.exe+4C560E   48 8B 48 28         mov     rcx,[rax+28]
DarkSoulsIII.exe+4C5612   48 89 4A 28         mov     [rdx+28],rcx
DarkSoulsIII.exe+4C5616   48 8B 48 30         mov     rcx,[rax+30]
DarkSoulsIII.exe+4C561A   48 89 4A 30         mov     [rdx+30],rcx
DarkSoulsIII.exe+4C561E   48 8B 48 38         mov     rcx,[rax+38]
DarkSoulsIII.exe+4C5622   48 89 4A 38         mov     [rdx+38],rcx
DarkSoulsIII.exe+4C5626   48 8B 48 40         mov     rcx,[rax+40]
DarkSoulsIII.exe+4C562A   48 89 4A 40         mov     [rdx+40],rcx
DarkSoulsIII.exe+4C562E   48 8B 48 48         mov     rcx,[rax+48]
DarkSoulsIII.exe+4C5632   48 89 4A 48         mov     [rdx+48],rcx
DarkSoulsIII.exe+4C5636   48 8B 48 50         mov     rcx,[rax+50]
DarkSoulsIII.exe+4C563A   48 89 4A 50         mov     [rdx+50],rcx
DarkSoulsIII.exe+4C563E   48 8B 48 58         mov     rcx,[rax+58]
DarkSoulsIII.exe+4C5642   48 89 4A 58         mov     [rdx+58],rcx
DarkSoulsIII.exe+4C5646   48 8B 48 60         mov     rcx,[rax+60]
DarkSoulsIII.exe+4C564A   48 89 4A 60         mov     [rdx+60],rcx
DarkSoulsIII.exe+4C564E   48 8B 48 68         mov     rcx,[rax+68]
DarkSoulsIII.exe+4C5652   48 89 4A 68         mov     [rdx+68],rcx
DarkSoulsIII.exe+4C5656   48 8B 48 70         mov     rcx,[rax+70]
```

Above is a screenshot of some faulting code in the parsing function. [r9] is read as the ‘area_id’ portion of this packet. It is then multiplied by 168 (0xA8) and used to dereference a pointer inside SprjEventFlagMan. **Occasionally** this will point to executable memory, then the game will write over that executable memory with bytes received from the attacker. There are only 120 bytes available to use, however this can be chained with another exploit known as ‘executable static packet buffers' to execute 1000x’s more code than this.

NG+ cheat

Similar to packet 15 and 26, you can set the flag to send the player to the next new game cycle in this packet. This will only affect remote players, but may cause save data loss or corruption in the exact same way to packet 26.

17 - “Host world chr/obj data”

Below is my estimation on the structure of this packet:

```
struct SNetBonfire {
	int iSegmentSize;
	[unmapped]
};

struct SGXDecal {
	int iSegmentSize;
	int iDecalCount;
	[unmapped]
};

struct SWorldObject {
	int iSegmentSize;
	[unmapped]
};

struct SChrEntity {
	int iSegmentSize;
	[unmapped]
};

struct SPacket17 {
	SNetBonfire sNetBonfireSegment;
	SGXDecal sGXDecalSegment;
	SWorldObject sWorldObjectSegment;
	SChrEntity sChrEntitySegment;
};
```

The function of this packet is to synchronise host world elements with the connecting client for a seamless online experience. This includes information such as present blood / footstep decal stains, enemy death states, and the condition of objects (e.g. broken barrels). The parsing function for this packet is at sub_0x776DB0.

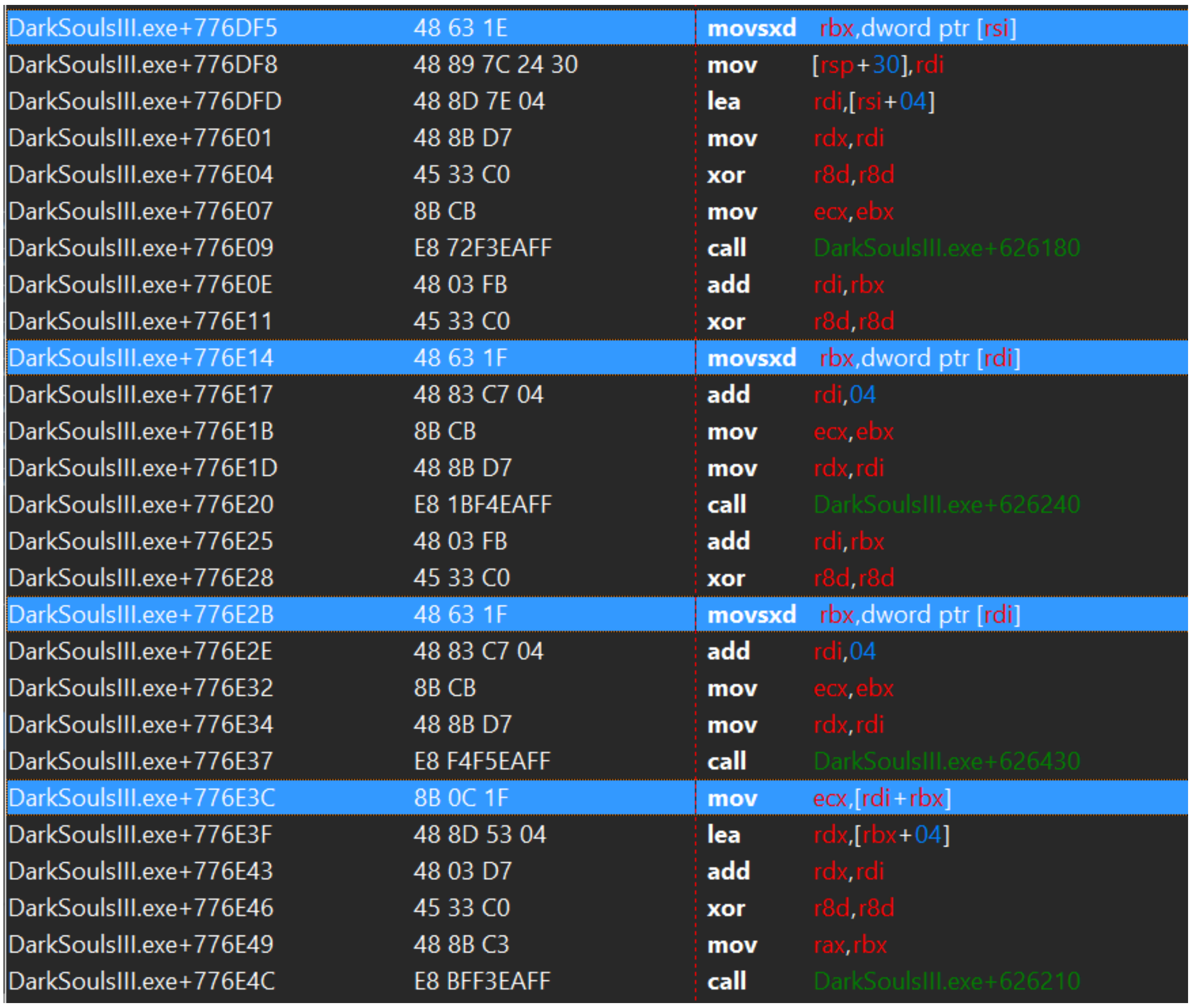
Decal count

The decal count read as an unsigned int. This means that it is possible to independently declare an illegal amount of decals contained inside the packet. A bad actor could place (MAX_UINT32) decals inside this packet causing an out-of-bounds read and crash of the target player.

This can be addressed by enforcing a maximum size of the decals depending on the size of the packet, rather than trusting a separate field from it.

Unchecked segment header sizes

As shown in this packet structure, each segment contains a value that declares the size of that segment in relation to the others. This has no checking whatsoever and will be the size that the parsing function requests from the heap. A malicious player can simply set this to an extremely high value to either deplete heap space on the target game or cause a null pointer dereference in memcpy when the heap fails to allocate a memory block with the size given. As the location of the next segment relies on the size of the previous segments, this only gets worse with each segment that is processed.

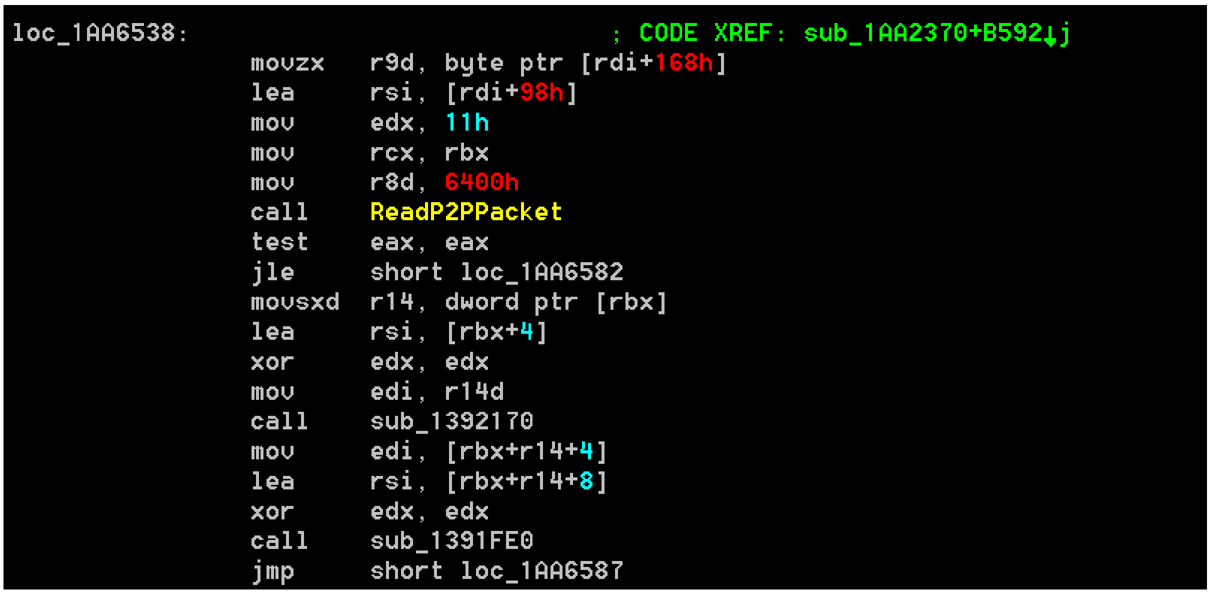


```
DarkSoulsIII.exe+776DF5    48 63 1E          movsxd  rbx,dword ptr [rsi]
DarkSoulsIII.exe+776DF8    48 89 7C 24 30    mov     [rsp+30],rdi
DarkSoulsIII.exe+776DFD    48 8D 7E 04       lea     rdi,[rsi+04]
DarkSoulsIII.exe+776E01    48 8B D7          mov     rdx,rdi
DarkSoulsIII.exe+776E04    45 33 C0          xor     r8d,r8d
DarkSoulsIII.exe+776E07    8B CB             mov     ecx,ebx
DarkSoulsIII.exe+776E09    E8 72F3EAFF       call    DarkSoulsIII.exe+626180
DarkSoulsIII.exe+776E0E    48 03 FB          add     rdi,rbx
DarkSoulsIII.exe+776E11    45 33 C0          xor     r8d,r8d
DarkSoulsIII.exe+776E14    48 63 1F          movsxd  rbx,dword ptr [rdi]
DarkSoulsIII.exe+776E17    48 83 C7 04       add     rdi,04
DarkSoulsIII.exe+776E1B    8B CB             mov     ecx,ebx
DarkSoulsIII.exe+776E1D    48 8B D7          mov     rdx,rdi
DarkSoulsIII.exe+776E20    E8 1BF4EAFF       call    DarkSoulsIII.exe+626240
DarkSoulsIII.exe+776E25    48 03 FB          add     rdi,rbx
DarkSoulsIII.exe+776E28    45 33 C0          xor     r8d,r8d
DarkSoulsIII.exe+776E2B    48 63 1F          movsxd  rbx,dword ptr [rdi]
DarkSoulsIII.exe+776E2E    48 83 C7 04       add     rdi,04
DarkSoulsIII.exe+776E32    8B CB             mov     ecx,ebx
DarkSoulsIII.exe+776E34    48 8B D7          mov     rdx,rdi
DarkSoulsIII.exe+776E37    E8 F4F5EAFF       call    DarkSoulsIII.exe+626430
DarkSoulsIII.exe+776E3C    8B 0C 1F          mov     ecx,[rdi+rbx]
DarkSoulsIII.exe+776E3F    48 8D 53 04       lea     rdx,[rbx+04]
DarkSoulsIII.exe+776E43    48 03 D7          add     rdx,rdi
DarkSoulsIII.exe+776E46    45 33 C0          xor     r8d,r8d
DarkSoulsIII.exe+776E49    48 8B C3          mov     rax,rbx
DarkSoulsIII.exe+776E4C    E8 BFF3EAFF       call    DarkSoulsIII.exe+626210
```

The image above highlights the points at which each of the four segment sizes are read from the packet in the parsing function. These values have no sanity checks to see whether they are matching with the current size of the packet, or even the maximum possible size of the packet (25,600 bytes).

I suggest that, in order to address this, you keep track of the size of these segment sizes and ensure that they do not exceed the size of the packet received to maintain data integrity.

THIS EXPLOIT ALSO EXISTS INSIDE ELDEN RING'S CNT! (see below)



```
loc_1AA6538:                            ; CODE XREF: sub_1AA2370+B592↓j
                movzx   r9d, byte ptr [rdi+168h]
                lea     rsi, [rdi+98h]
                mov     edx, 11h
                mov     rcx, rbx
                mov     r8d, 6400h
                call    ReadP2PPacket
                test    eax, eax
                jle     short loc_1AA6582
                movsxd  r14, dword ptr [rbx]
                lea     rsi, [rbx+4]
                xor     edx, edx
                mov     edi, r14d
                call    sub_1392170
                mov     edi, [rbx+r14+4]
                lea     rsi, [rbx+r14+8]
                xor     edx, edx
                call    sub_1391FE0
                jmp     short loc_1AA6587
```

18 - “Throw”

Below is my estimation on the structure of this packet:

```
struct SThrowCoordinate {
	float fX;
	float fZ;
	float fY;
	float fR;
};

struct SPacket0x12 {
	SThrowCoordinate sThroweeLoc;
	SThrowCoordinate sThrowerLoc;
	unsigned short wThrowee;
	unsigned short wThrower;
	int iThrowID;
	int iUnused1;
};
```

This packet is used to synchronise ‘grab’ or ‘throw’ animations and attacks between players. The parsing function for this packet is at sub_0x5F3D6.

‘WhoIs’ spoofing

The game relies on ‘wThrowee’ and ‘wThrower’ entries to get the PlayerIns pointers to the 2 entities involved in the throw attack. A malicious user could put anyone else in a throw with anyone else (including making a player throw themselves) which can cause animation issues and glitches, as well as making it appear that other users are cheating.

For example player A (malicious cheater) could send a packet that makes player B backstab player C, or vice versa.

Signalling NaN/Inf

An issue with this packet is that the throw coordinates can be set freely with no restrictions. This means that a malicious user could play a victim in a throw, and teleport them to an invalid location causing both the victim and everyone else in the session to crash.

19 - “Active goods/spell”

Below is my estimation on the structure of this packet:

```
struct SPacket19 {
	uint16_t wWhoIs;
	uint32_t iEquippedSpell;
	uint32_t iGoodsID;
};
```

This packet is used to normally update the currently, actively equipped consumable or spell on guest players in order for certain behaviours and animations to work correctly. The parsing function for this packet is at sub_0x776AA0.

Local ‘WhoIs’ use

By modifying the ‘wWhoIs’ entry to be the same as the local player, guests can change the equipped spell and goods ID briefly causing animation and graphical glitches.

I recommend this be addressed by rejecting the packet if the WhoIs ID matches the local player.

21 - “PvE damage”

Below is my estimation on the structure of this packet:

```
struct SPacket0x15 {
      unsigned short wDstID;
      unsigned short wPack;
      unsigned short wHealthDmg;
      unsigned short wStaminaDmg;
      unsigned char bUnk1;
      unsigned char bUnk2
      bool bIsNormalDamagePacket;
      bool bIsThrowType;
      unsigned short wSrcID;
      float fPhysicalDamage;
      float fMagicDamage;
      float fFireDamage;
      float fLightningDamage;
      float fDarkDamage;
      float fPhysicalDamage;
      float fMagicDamage;
      float fFireDamage;
      float fLightningDamage;
      float fDarkDamage;
      float fUnk0[7];
      unsigned short wPhysMultiplier;
      unsigned short wMagicMultiplier;
      unsigned short wFireMultiplier;
      unsigned short wLightningMultiplier;
      unsigned short wDarkMultiplier;
      char bUnk9;
      EHitType eHitType; //Bin 1 = Bullet
      float fUnk1;
      int iBehaviourID;
      int iAttackParamID;
      int iUnkID2;
      int iSpEffectIdOnHitBullet[5];
      int iSpEffectIdOnHit[3];
      int iUnkValueUsedInDamageBuffer1;
      int iWeaponID; //+0x70
      int iUnk7;
      float fUnkValueUsedInDamageBuffer3;
      float fUnkValueUsedInDamageBuffer4;
      float fUnk10;
      float fUnk11; //+84
      float fIsThrow;
      int iUnkValueUsedInDamageBuffer2;
```

```
       float fSpawnX;
       float fSpawnZ;
       float fSpawnY;
       float fSpawnU;
       uint16_t wThrowPlayer;
};
```

This packet is used to communicate damage done in PvE (player vs. enemy) scenarios between clients. The packet parsing function is located at sub_8D61D0.

'WhoIs' spoofing

The entry 'wDstID' is a 2-byte value that is used by the game to retrieve the EntityIns pointer to the entity who will receive the damage. This can be used by malicious cheaters to instantly kill or set other player's stamina bar to 0 by using the value which will retrieve a PlayerIns pointer instead of an EntityIns pointer as the packet was meant to be used for.

There is also no check to see whether the entity being damaged is protected (e.g. friendly NPC). So this has the potential for save file damage if a critical NPC is killed with this method.

This can be addressed by ensuring the target NPC is valid, and able to receive damage from the target player.

22 - “WorldChrSync Remote => Host”

Below is my estimation on the structure of this packet:

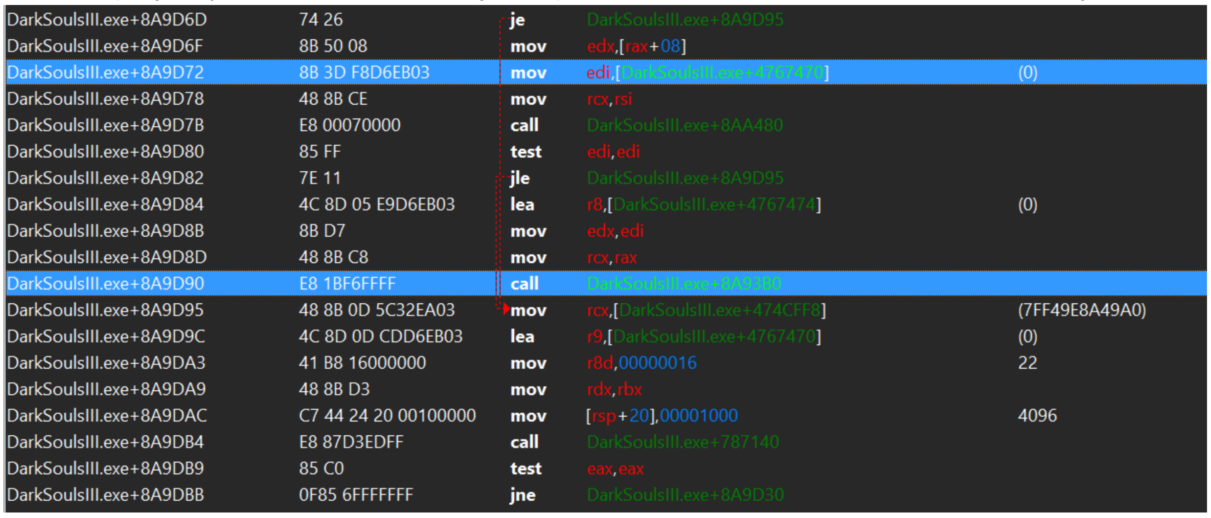
```
struct SPacket22 {
        int iEntryCount;
        int iEntries[1023];
};
```

This packet is sent by remote players to the host and is involved in coordinating which game client is in “control” of particular enemy actions. The parsing function for this packet is at sub_0x8A9BB0.

Abnormal entry counts

The main issue with this packet is that the ‘iEntryCount’ value has no sanity check. That means that it can be any signed value greater than 0. A malicious player could set that value to MAX_INT and the packet parsing function will increment and continue reading as many entries as the count has told it to.

This can be fixed by ensuring that the packet size (n) - 4 (sizeof(int)) / 4 (sizeof(int)) is equal to the amount of entries provided in the ‘iEntryCount’ segment. **This is actually already done in the case of packet 23** (which functions the same as this one, but from Host => Remote players). So I’m not sure why this packet hasn't been done in the same way.



```
DarkSoulsIII.exe+8A9D6D    74 26                  je    DarkSoulsIII.exe+8A9D95
DarkSoulsIII.exe+8A9D6F    8B 50 08               mov   edx,[rax+08]
DarkSoulsIII.exe+8A9D72    8B 3D F8D6EB03         mov   edi,[DarkSoulsIII.exe+4767470]      (0)
DarkSoulsIII.exe+8A9D78    48 8B CE               mov   rcx,rsi
DarkSoulsIII.exe+8A9D7B    E8 00070000            call  DarkSoulsIII.exe+8AA480
DarkSoulsIII.exe+8A9D80    85 FF                  test  edi,edi
DarkSoulsIII.exe+8A9D82    7E 11                  jle   DarkSoulsIII.exe+8A9D95
DarkSoulsIII.exe+8A9D84    4C 8D 05 E9D6EB03      lea   r8,[DarkSoulsIII.exe+4767474]       (0)
DarkSoulsIII.exe+8A9D8B    8B D7                  mov   edx,edi
DarkSoulsIII.exe+8A9D8D    48 8B C8               mov   rcx,rax
DarkSoulsIII.exe+8A9D90    E8 1BF6FFFF            call  DarkSoulsIII.exe+8A93B0
DarkSoulsIII.exe+8A9D95    48 8B 0D 5C32EA03      mov   rcx,[DarkSoulsIII.exe+474CFF8]      (7FF49E8A49A0)
DarkSoulsIII.exe+8A9D9C    4C 8D 0D CDD6EB03      lea   r9,[DarkSoulsIII.exe+4767470]       (0)
DarkSoulsIII.exe+8A9DA3    41 B8 16000000         mov   r8d,00000016                        22
DarkSoulsIII.exe+8A9DA9    48 8B D3               mov   rdx,rbx
DarkSoulsIII.exe+8A9DAC    C7 44 24 20 00100000   mov   [rsp+20],00001000                   4096
DarkSoulsIII.exe+8A9DB4    E8 87D3EDFF            call  DarkSoulsIII.exe+787140
DarkSoulsIII.exe+8A9DB9    85 C0                  test  eax,eax
DarkSoulsIII.exe+8A9DBB    0F85 6FFFFFFF          jne   DarkSoulsIII.exe+8A9D30
```

As shown above, a very basic check is done. The ‘iEntryCount’ value is moved into ‘edi’, where there is a basic check to see if it is above 0. If it is, then it will be put into sub_0x8A93B0 which will use it in a for/while loop to iterate over the rest of the received data provided in ‘r8’.

25 - “Player stat update”

Below is my estimation on the structure of this packet:

```
struct SPacket25 {
	unsigned int iVig = 0;
	unsigned int iAtn = 0;
	unsigned int iEnd = 0;
	unsigned int iStr = 0;
	unsigned int iDex = 0;
	unsigned int iInt = 0;
	unsigned int iFth = 0;
	unsigned int iLck = 0;
	unsigned int iNSt = 0;
	unsigned int iVit = 0;
	unsigned int iSlv = 0;
};
```

I am not sure on the function of this packet, as I am unable to find it’s use in normal game behaviour. I have never seen it sent before. The parsing function for this packet is at sub_0x7787D0.

There are no inherent flaws with the packet’s implementation, other than there being no sanity checks on the stat entries (e.g. players can set their soul level to be MAX_UINT32 and, when they die, players in opposing teams will receive an enormous soul reward)

26 - “Set remote event state”

Below is my estimation on the structure of this packet:

```
struct SPacket26 {
      int iEventID;
      bool bEventState;
};
```

This packet is used to set remote event flags for guest players. The host should send this packet to guest players in order to synchronise event flags that have occurred (such as activating elevators or opening doors). The parsing function for this packet is at sub_0x7773A0.

NG+ cheat

Similar to packet 15, this packet can also set event flags like event ID “30” which will immediately send guest players to new game+. As it is a remote event, the actual flag will not carry over onto the player’s world when they “return home”, however the process of sending the player to the next new game cycle will chain a series of other event flags which will persist and potentially soft-lock the affected player.

28 - “Register hit”

Below is my estimation on the structure of this packet:

```
struct SPacket0x1C {
	uint32_t iHitSfxFlags;
	uint16_t wWhoIsHit;
	uint16_t wPadding1;
	uint16_t wWhoIsHitter;
	uint16_t wPadding2;
	uint32_t iUnk2;
	uint32_t iHitEffectSfxParamType;
	uint32_t iUnk4;
	uint32_t iUnk5;
	uint32_t iUnk6;
	uint32_t iUnk7;
	uint32_t iDecalParamID1;
	uint32_t iDecalParamID2;
	uint32_t iUnk8;
	uint32_t iUnk9;
	uint32_t iHitEffectSfxConceptParamID1;
	uint32_t iHitEffectSfxConceptParamID2;
	uint32_t iHitEffectSeParamSoundType;
	uint32_t iHitEffectSoundType;
	uint32_t iUnused;
	uint32_t iUnk13;
	uint32_t iSeMaterialConvertParamID1;
	uint32_t iSeMaterialConvertParamID2;
	uint32_t iUnk14;
	float fHitLocX; // +0x58
	float fHitLocZ; // +0x5C
	float fHitLocY; // +0x60
	float fHitLoc88;// +0x64
	uint32_t iSfxValue24;
	uint32_t eHitType;
	uint32_t iUnk18;
	uint32_t iUnk19;
};
```

This is a packet used to send hit visual effects (e.g. armour sparks, blood spatters) when a player is hit, to other players in the session. The parsing function for this packet is at sub_0x8D64E1.

Local ‘WhoIs’ spoofing

By changing the ‘wWhoIsHit’ entry to the local player, it will cause screen shake to appear on the local player’s screen. If this is done excessively, it will crash the game and is also very difficult to look at.

Performance issues

Armour damaging visual effects are popular amongst cheaters to crash victim’s games. This can be done by hitting a player incredibly fast which causes the “sparks” and visual effects created by the hit to crash the game of anyone in the session.

36 - “SpEffect update”

Below is my estimation on the structure of this packet:

```
enum EEffectUpdateType : uint16_t {
      eEffectUpdateType_ApplyEffect = 0,
      eEffectUpdateType_RemoveEffect = 1,
      eEffectUpdateType_OnOffHandSwap = 2,
      eEffectUpdateType_OnMainHandSwap = 3,
};

struct SPacket36 {
      uint16_t wEntityWho;
      EEffectUpdateType ePacketType;
      int iEffectTimeStamp;
      int iEffectID;
};
```

This packet is used to synchronise “SpEffects” between different clients. This works well, and the implementation of a timestamp for latency mitigation is a nice touch. The parsing function for this packet is at sub_0x8D6630.

Local ‘WhoIs’ spoofing

Similar to other packets which use numerical IDs sent inside the packet to identify players, this is very much open to abuse. There is a popular cheat that allows other player’s to change the ID inside this packet and this will cause the packet parsing function to apply the received effect to the local players instead of the player who sent it.

I recommend this be addressed by applying the effect sent with the packet to the player who sent it by using their SteamSessionLite pointer (or platform equivalent), instead of using the ID sent with the packet. This would ensure that players cannot apply effects to others illegally using this method.

37 - “Apply buff SpEffect”

Below is my estimation on the structure of this packet:

```
struct SPacket37 {
      int iEffectID;
};
```

This packet is used to apply SpEffects from buffed weapons. For example if player A uses a “Rotten Pine Resin” to apply a poison build-up on their weapon, then hits player B, player A would send this packet with the poison build up SpEffect ID inside this packet. The parsing function for this packet is at sub_8BDB60.

No sanity check

Similar to most p2p packets, there is no sanity check for this packet. This means that the packet parsing function will apply whatever effect ID has been sent in this packet to the local player, and does not check if the player who sent it has that buff active.

For example, if player A sent this packet with ID “33” to everyone in the play session, everyone would be cursed automatically.

This can be addressed by having a check in the parsing function, which compares the effect ID inside this packet to the current buff effect on the sending player’s weapon. If it does not match, it may be discarded as invalid data.

38 - “Adjust soul counter”

Below is my estimation on the structure of this packet:

```
struct SPacket38 {
      int iSoulCount;
};
```

This packet will add the value inside ‘iSoulCount’ to the local player’s current soul counter. I cannot find this ever being used in normal game behaviour. I recommend the packet be removed, as it is popularly used by cheaters to set other player’s soul counters to 0 or the maximum count which concerns the player for the possibility of them being banned for invalid data. The parsing function for this packet is at sub_8C0011.

I recommend that this packet be removed, as it is not sent under normal game behaviour and doesn’t appear to be used.

44 - “Map item information”

Below is my estimation on the structure of this packet:

```
struct SItemPickupRequestStruct {
	uint32_t iRawItemType;
};

struct SItemDeleteStruct {
	uint32_t iRawItemType;
};

struct SItemContainer {
	unsigned int iItemID;
	unsigned int iItemQuantity;
	unsigned int iItemDurability;
};

enum EItemBagFlags : uint32_t {
	eItemBagFlags_PlayerDropInteractable = 1 << 6,
	eItemBagFlags_NetSync = 1 << 7,
	eItemBagFlags_PlayerDropEventItem = 1 << 19,
};

struct SItemDropStruct {
	int iUnusedInItemDrop;
	uint32_t iRawItemType;
	int iIsMsgMapOnCollection;
	int iEventIDOnCollection;
	int iMsgMapIDOnCollection;
	float fX;
	float fZ;
	float fY;
	float fR;
	int iUnkUnusedInItemBags1;
	float fActionButtonDistanceModifier;
	EItemBagFlags eItemBagFlags;
	int iUnkUnusedInItemBags2;
	int iUnkUnusedInItemBags3;
	unsigned int iAmount;
	SItemContainer sItem[8];
};

struct SItemGiveStruct {
	unsigned int iAmount;
	SItemContainer sItem[8];
};
```

```
enum EItemPacketType : uint16_t {
	eItem_RequestWorldItemInfo = 0,
	eItem_DeleteItemCategory = 1,
	eItem_OnDelete = 2,
	eItem_Remote2Host = 3,
	eItem_Host2Remote = 4,
	eItem_PickupRequest = 5,
	eItem_Give = 6,
};

struct SPacket44 {
	short wTrue;
	EItemPacketType eCase;
	uint8_t pData[156];
};
```

Item networking is host based, and involves a relay of different types of item object packets:

- Type 0: 'RequestWorldItemInfo'
- Type 1: 'DeleteItemCategory '
- Type 2: 'OnDelete'
- Type 3: 'RemoteToHost'
- Type 4: 'HostToRemote'
- Type 5: 'PickupRequest'
- Type 6: 'Give'

So in an example, if Player A was a host, and player B was a phantom. Player B joins the world and sends a type 0 packet to the host. The host then sends them a type 1 packet back which removes a certain category of items from displaying (for example ItemLot items, enemy drops, etc.) and also one type 4 packet for every existing item object there is in the world.

If player B drops an item, they will send a type 3 packet to the host, who will then send a type 4 packet to everyone who is connected, which registers the item on the ground.
If player B then picks that item up, they send a type 5 packet to the host, who sends a type 6 packet back with the item information inside it. If two players pick up the same item at the same time, only one of them will get it (whoever sends the packet type 5 first). The packet parsing function is located at sub_0x7BB850

Setting game progression flags

Using the 'eItemBagFlags' variable to include 'eItemBagFlags_PlayerDropEventItem', the game will set the game flag event specified in 'iEventIDOnCollection' to true. For example, if a malicious user set this value, they could cause the host of the world to set invalid game progression flags (e.g. new game cycle, killing bosses/NPCs) which can damage or softlock save game files.

Another requirement for this is to set the type of item to be a ‘MapItem’ rather than a ‘PlayerItem’ I suspect that this is because Dark Souls III’s networking networks entire game structures rather than actually sends only specific information over the network that is needed.

Excessive ‘iItemAmount’

This will cause a crash of the application if the ‘iAmount’ value is too high.

Invalid items

Item IDs in bags aren’t checked for validity, allowing malicious players to alter the bag contents to include items that the victims cannot discard (test data such as ?GoodsName?, cut content, gesture prompts), are items that they otherwise cannot obtain or will put them at risk (DLC spells, weapons upgraded beyond their max Weapon Level).

‘Give’ spoofing

Now the issue is that you can just skip right to type 6 packets. It has no prerequisites and doesn’t need the item to exist on the ground. This is how the ‘Item Give’ cheat works, where cheats can put items right into your inventory.

If a malicious user sends a ‘type 6’ packet to anyone else in the session, the item will automatically be added to their inventory. This will cause a crash of the application if the ‘iAmount’ value is too high.

45 - “Object action”

Below is my estimation on the structure of this packet:

```
struct SPacket45 {
	short wPlayerID;
	char bAreaID;
	char bUnkByte;
	short wEventID;
};
```

This packet is used to synchronise object interactions (e.g. operating a lever) to other players. The parsing function for this packet is located at sub_0x4BE9C0

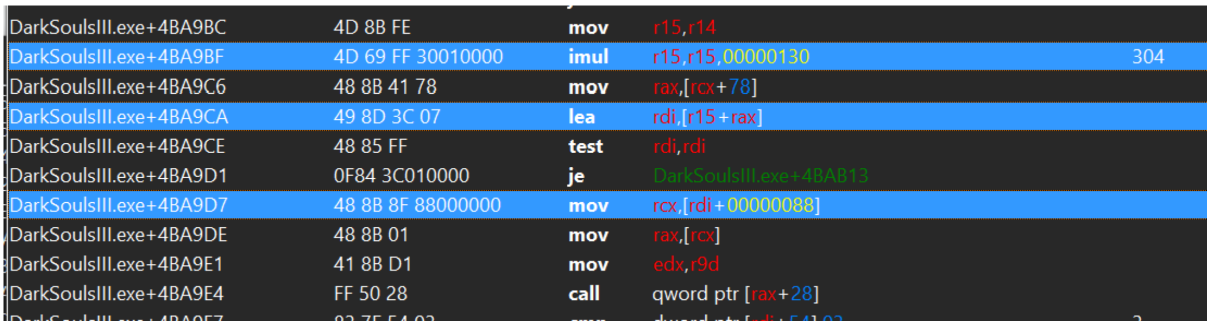
‘WhoIs’ spoofing

By changing the ‘wPlayerID’ entry to a different player, it will cause the local player to be teleported to the event object (e.g elevator lever, door) from wherever they were.

I recommend that this is fixed by ignoring this value, and instead applying the animation / interaction sequence to whichever player has sent the packet.

Invalid events

Setting the ‘wEventID’ to an invalid value for that particular area will cause the game to crash. This is because this value is used to access an array by doing (‘wEventID’ * 304) then using it to access an array. If this points to invalid memory it will crash the application, but may result in code misdirection if it happens to point to another virtual function table, however this is extremely unlikely. What will most likely occur is just a game crash.



```
DarkSoulsIII.exe+4BA9BC   4D 8B FE            mov    r15,r14
DarkSoulsIII.exe+4BA9BF   4D 69 FF 30010000   imul   r15,r15,00000130          304
DarkSoulsIII.exe+4BA9C6   48 8B 41 78         mov    rax,[rcx+78]
DarkSoulsIII.exe+4BA9CA   49 8D 3C 07         lea    rdi,[r15+rax]
DarkSoulsIII.exe+4BA9CE   48 85 FF            test   rdi,rdi
DarkSoulsIII.exe+4BA9D1   0F84 3C010000       je     DarkSoulsIII.exe+4BAB13
DarkSoulsIII.exe+4BA9D7   48 8B 8F 88000000   mov    rcx,[rdi+00000088]
DarkSoulsIII.exe+4BA9DE   48 8B 01            mov    rax,[rcx]
DarkSoulsIII.exe+4BA9E1   41 8B D1            mov    edx,r9d
DarkSoulsIII.exe+4BA9E4   FF 50 28            call   qword ptr [rax+28]
```

38

50 - “NPC summon”

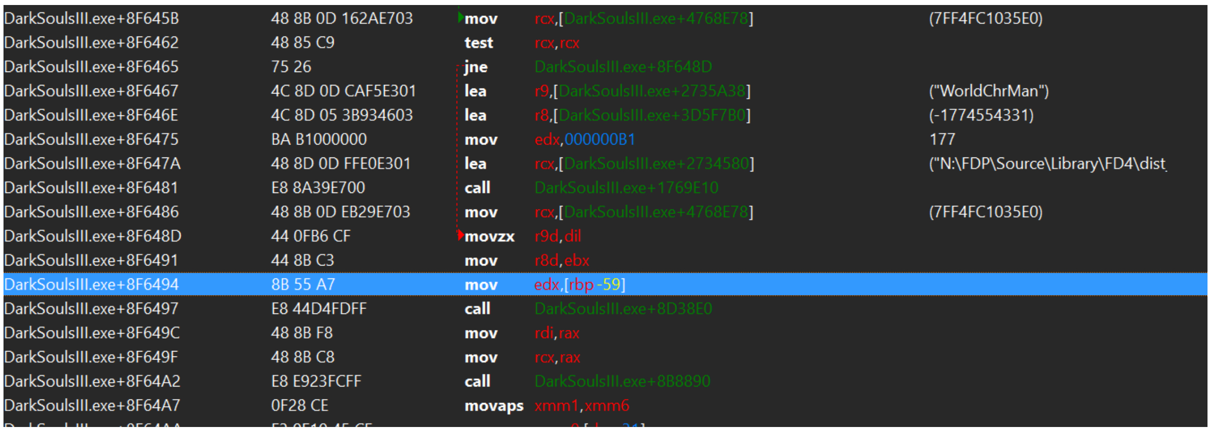
Below is my estimation on the structure of this packet:

```
struct SPacket50 {
	uint32_t iHandleID;
	uint32_t iSortID;
	uint32_t iUnk1;
	uint32_t iChrType;
	uint32_t iUnk2;
	int iPadding1;
	int iPadding2;
	int iPadding3;
	float fSpawnX;
	float fSpawnZ;
	float fSpawnY;
	float fSpawnQ;
	float fSpawnUnk1;
	float fSpawnR;
	float fSpawnUnk2;
	float fSpawnUnk3;
};
```

This packet is intended to reload and “summon in” NPC phantoms or invaders that join while a multiplayer session is active. The parsing function for this packet is located at sub_0x8F6420

Invalid entity handles

By inputting an invalid handle, or a handle that cannot be reloaded, the game will dereference null and crash.



```
DarkSoulsIII.exe+8F645B    48 8B 0D 162AE703    mov     rcx,[DarkSoulsIII.exe+4768E78]    (7FF4FC1035E0)
DarkSoulsIII.exe+8F6462    48 85 C9             test    rcx,rcx
DarkSoulsIII.exe+8F6465    75 26                jne     DarkSoulsIII.exe+8F648D
DarkSoulsIII.exe+8F6467    4C 8D 0D CAF5E301    lea     r9,[DarkSoulsIII.exe+2735A38]     ("WorldChrMan")
DarkSoulsIII.exe+8F646E    4C 8D 05 3B934603    lea     r8,[DarkSoulsIII.exe+3D5F7B0]     (-1774554331)
DarkSoulsIII.exe+8F6475    BA B1000000          mov     edx,000000B1                      177
DarkSoulsIII.exe+8F647A    48 8D 0D FFE0E301    lea     rcx,[DarkSoulsIII.exe+2734580]    ("N:\FDP\Source\Library\FD4\dist
DarkSoulsIII.exe+8F6481    E8 8A39E700          call    DarkSoulsIII.exe+1769E10
DarkSoulsIII.exe+8F6486    48 8B 0D EB29E703    mov     rcx,[DarkSoulsIII.exe+4768E78]    (7FF4FC1035E0)
DarkSoulsIII.exe+8F648D    44 0FB6 CF           movzx   r9d,dil
DarkSoulsIII.exe+8F6491    44 8B C3             mov     r8d,ebx
DarkSoulsIII.exe+8F6494    8B 55 A7             mov     edx,[rbp-59]
DarkSoulsIII.exe+8F6497    E8 44D4FDFF          call    DarkSoulsIII.exe+8D38E0
DarkSoulsIII.exe+8F649C    48 8B F8             mov     rdi,rax
DarkSoulsIII.exe+8F649F    48 8B C8             mov     rcx,rax
DarkSoulsIII.exe+8F64A2    E8 E923FCFF          call    DarkSoulsIII.exe+8B8890
DarkSoulsIII.exe+8F64A7    0F28 CE              movaps  xmm1,xmm6
```

The screenshot above shows the entity handle being moved into ‘edx’, which is used as an argument in sub_0x8D38E0. This function **can** return nullptr, but there is no check to see whether this is the case, and it is blindly trusted that the entity will be valid. The application will crash in sub_8B8890.

Use of local player handle

By inputting the local player's handle (normally 0x10068000) in 'iHandleID', this packet will apply the spawn coordinate and character type information to teleport and reload the local player. This means a malicious user could send this packet to teleport others, or send them to invalid coordinates to either crash or instantly kill them.

I recommend this be addressed by **explicitly** checking whether the target handle ID provided in the packet belongs to a valid, summonable, NPC character on the loaded map.

Signalling NaN/Inf

The spawn coordinates (X/Z/Y) provided in this packet are unchecked floats, this means that a bad actor could send anyone (including themselves) to invalid coordinates on other player's screens, resulting in a crash of the player's game.

A simple boundary check would resolve this issue, in addition to the fix listed above.

51 - “NPC leave”

Below is my estimation on the structure of this packet:

```
struct SPacket51 {
      uint32_t iTargetHandle;
      uint32_t iSortID;
};
```

Opposing packet 50, this packet synchronises NPC invaders/phantoms from leaving the host world.

Use of local player handle

By inputting the local player’s handle (normally 0x10068000) in ‘iHandleID’, the game will remove the local handle, which causes issues with the player’s interaction with the game world.

Similar to the issue listed in packet 50, I recommend this be addressed by **explicitly** checking whether the target handle ID provided in the packet belongs to a valid, summonable, NPC character on the loaded map.

58 - “Adjust hero point counter”

Below is my estimation on the structure of this packet:

```
struct SPacket58 {
      bool bIsAdd;
      uint8_t bChangeAmount;
};
```

I do not know why this packet exists, ‘Hero point’ (humanity) does not do anything as far as I can tell in Dark Souls 3. The parsing function for this packet is at sub_0x7B71C6

I recommend that this packet is removed as it is redundant.

60 - “Guest effect initiation allocation”

Below is my estimation on the structure of this packet:

```
struct SPacket60 {
      unsigned int iEffectCount;
      int iEffectInitialTimestamp;
};
```

This packet tells the game how much data to expect in packet 61 (‘guest effect initiation application’), it also will contain the initial join timestamp of the player.

I don’t know why this packet exists – ideally this would be sent *with* the actual effect data, but it isn’t.

Heap exhaustion

Malicious players can just use any number they like in the ‘iEffectCount’ entry. The game will try to allocate (‘iEffectCount’ * 8) bytes from the heap. There are no checks, so if the value is too high the heap will become depleted, return null, and the game will crash.

64 - “Bullet - initiation”

Below is my estimation on the structure of this packet:

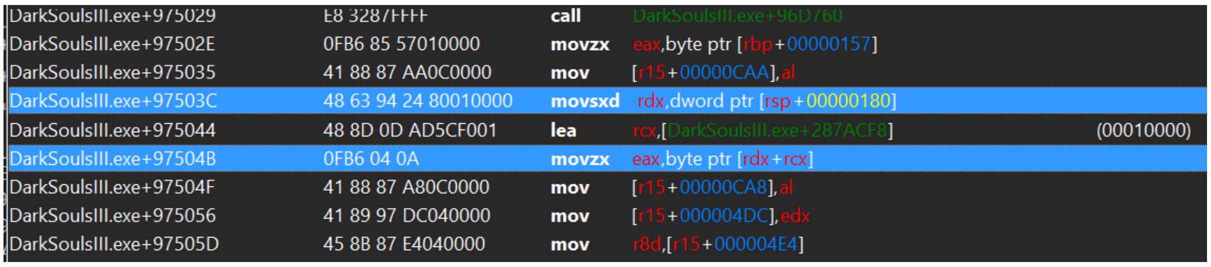
```
struct SBulletStruct64 {
	int iSrcHandle;
	float fCasterX;
	float fCasterZ;
	float fCasterY;
	int iUnk4;
	int iUnk5;
	int iUnk6;
	int iBulletHandle;
	int iUnk8;
	int iUnk9;
	int iUnk101;
	int iBulletID;
	int iUnk10;
	int iUnl11;
	int iUnk13;
	int iUnk14;
	int iUnl15;
	int iUnk16;
	int iUnk17;
	int iUnl18;
	int iUnk19;
	int iUnk20;
	int iUnl21;
	int iUnk22;
	float fSpawnX;
	float fSpawnXA1;
	float fSpawnXA2;
	float fSpawnXA3;
	float fSpawnZ;
	float fSpawnZA1;
	float fSpawnZA2;
	float fSpawnZA3;
	float fSpawnY;
	float fSpawnYA1;
	float fSpawnYA2;
	float fSpawnYA3;
	char pBuffer1[0x6C];
	int iBulletType;			//Crash at 0x14097504B, assert is < 4
	int iUnk25;
	uint32_t iUnkIterations;	//0x14096EC00 freeze
	char pBuffer2[0x08];
};
```

```
struct SPacket64 {
      char bBulletCount;
      SBulletStruct64 sBullet[128];
};
```

This is a packet that sends all active bullets in the host world to players who are just connecting. The parsing function for this packet is located at sub_0x97B321

Out of bounds read

The value ‘iBulletType’ is used to access an array to retrieve a byte.



```
DarkSoulsIII.exe+975029    E8 3287FFFF           call     DarkSoulsIII.exe+96D760
DarkSoulsIII.exe+97502E    0FB6 85 57010000      movzx    eax,byte ptr [rbp+00000157]
DarkSoulsIII.exe+975035    41 88 87 AA0C0000     mov      [r15+00000CAA],al
DarkSoulsIII.exe+97503C    48 63 94 24 80010000  movsxd   rdx,dword ptr [rsp+00000180]
DarkSoulsIII.exe+975044    48 8D 0D AD5CF001     lea      rcx,[DarkSoulsIII.exe+287ACF8]          (00010000)
DarkSoulsIII.exe+97504B    0FB6 04 0A            movzx    eax,byte ptr [rdx+rcx]
DarkSoulsIII.exe+97504F    41 88 87 A80C0000     mov      [r15+00000CA8],al
DarkSoulsIII.exe+975056    41 89 97 DC040000     mov      [r15+000004DC],edx
DarkSoulsIII.exe+97505D    45 8B 87 E4040000     mov      r8d,[r15+000004E4]
```

Unsigned int for/while loop

The ‘iUnkInterations’ portion of this packet causes the receiving game client to run some code for the count specified in the packet. There are little to no sanity checks in Dark Souls III’s netcode, therefore a malicious user could set this to MAX_UINT32 and cause the receiving game client to freeze indefinitely as long as the malformed packet was sent intermittently. (sub_0x96EC00)

Cut content / Test bullets

Test BulletParam Ids (e.g. 113) are frequently used by players to cause FPS issues and crash other players due to their expanding hitbox.

70 - “WorldChrSync: Enemy action”

Below is my estimation on the structure of this packet:

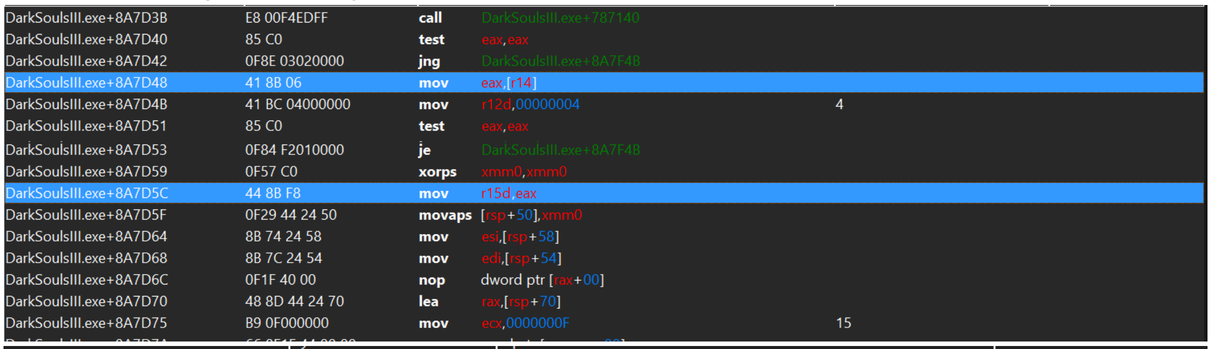
```
//You can't really create structs, because it's packed data
struct SEnemyActionData {
      short wEnemyHandle;
      short wPack;
};

struct SPacket70 {
      unsigned int iIterations;
      SEnemyActionData sData[50];
};
```

The packet functions to synchronise enemy actions (e.g. attacks, animations, grabs, deaths, etc.) with every other player in the session. The parsing function for this packet is at sub_8A7C10.

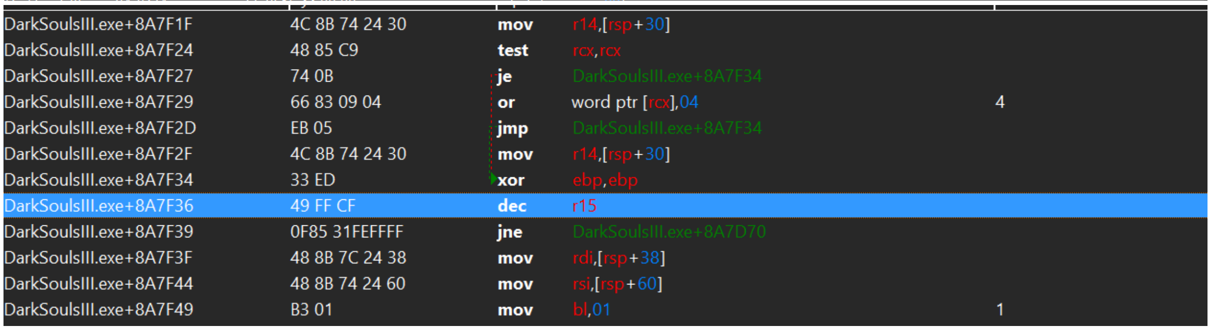
Unsigned int for/while loop

The first issue with this packet is that you notice the `iIterations` portion of the packet is an unsigned int. This is used to tell the packet parser how many enemy actions are contained within this packet. This means that a cheater can easily set this value to MAX_UINT32 and cause the receiving client’s game to freeze for the duration.

This can be addressed by having a sanity check to see whether the amount of data received matched the expected entry count.



```
DarkSoulsIII.exe+8A7D3B   E8 00F4EDFF      call    DarkSoulsIII.exe+787140
DarkSoulsIII.exe+8A7D40   85 C0            test    eax,eax
DarkSoulsIII.exe+8A7D42   0F8E 03020000    jng     DarkSoulsIII.exe+8A7F4B
DarkSoulsIII.exe+8A7D48   41 8B 06         mov     eax,[r14]
DarkSoulsIII.exe+8A7D4B   41 BC 04000000   mov     r12d,00000004              4
DarkSoulsIII.exe+8A7D51   85 C0            test    eax,eax
DarkSoulsIII.exe+8A7D53   0F84 F2010000    je      DarkSoulsIII.exe+8A7F4B
DarkSoulsIII.exe+8A7D59   0F57 C0          xorps   xmm0,xmm0
DarkSoulsIII.exe+8A7D5C   44 8B F8         mov     r15d,eax
DarkSoulsIII.exe+8A7D5F   0F29 44 24 50    movaps  [rsp+50],xmm0
DarkSoulsIII.exe+8A7D64   8B 74 24 58      mov     esi,[rsp+58]
DarkSoulsIII.exe+8A7D68   8B 7C 24 54      mov     edi,[rsp+54]
DarkSoulsIII.exe+8A7D6C   0F1F 40 00       nop     dword ptr [rax+00]
DarkSoulsIII.exe+8A7D70   48 8D 44 24 70   lea     rax,[rsp+70]
DarkSoulsIII.exe+8A7D75   B9 0F000000      mov     ecx,0000000F               15
```



```
DarkSoulsIII.exe+8A7F1F   4C 8B 74 24 30   mov     r14,[rsp+30]
DarkSoulsIII.exe+8A7F24   48 85 C9         test    rcx,rcx
DarkSoulsIII.exe+8A7F27   74 0B            je      DarkSoulsIII.exe+8A7F34
DarkSoulsIII.exe+8A7F29   66 83 09 04      or      word ptr [rcx],04          4
DarkSoulsIII.exe+8A7F2D   EB 05            jmp     DarkSoulsIII.exe+8A7F34
DarkSoulsIII.exe+8A7F2F   4C 8B 74 24 30   mov     r14,[rsp+30]
DarkSoulsIII.exe+8A7F34   33 ED            xor     ebp,ebp
DarkSoulsIII.exe+8A7F36   49 FF CF         dec     r15
DarkSoulsIII.exe+8A7F39   0F85 31FEFFFF    jne     DarkSoulsIII.exe+8A7D70
DarkSoulsIII.exe+8A7F3F   48 8B 7C 24 38   mov     rdi,[rsp+38]
DarkSoulsIII.exe+8A7F44   48 8B 74 24 60   mov     rsi,[rsp+60]
DarkSoulsIII.exe+8A7F49   B3 01            mov     bl,01                      1
```

In the images above, you can see ‘eax’ being assigned the value for ‘iIterations’. It is then placed into ‘r15d’ which holds the count for the remainder of the parsing function. No validation check is made, so the function will run that many times (or until it fails)

Stack cookie corruption:

Another issue with this packet is the function that it makes use of (sub_0x780960). This function is insecure, and can write outside of the stack buffer provided by the parsing function for this packet. This causes stack cookie corruption which leads to a crash of the application.

This can be addressed by either:

- Not packing the data, so that it does not need to be unpacked into the stack with this function or
- Allocating appropriate stack space on the calling function or
- Fixing the faulting function to prevent it going outside of it’s normal bounds.

Unfair boss / NPC deaths

Another “cheat” with this packet, is that any player can force death states on any enemy with this packet. For example, a malicious player could join a world and send an enemy action packet that puts the main boss of the area into it’s dying state which will end the play session and harm the host save file. This can equally be done to critical NPCs that are loaded into the area.

<u>Miscellaneous</u>

Executable static packet buffers**

In this game, memory that can only be read or written to also has permission to be executed. This is not dangerous by itself, but can be chained together with other exploits to run malicious, remote code on player's machines via the game. An example of this is in packet 23 (sub_0x8AAD18) which uses a buffer of data_0x44767470, however there are more in memory.

I seriously recommend that this be addressed, with the proper memory protection procedure.

RequestNotifyDisconnectSession

This is a packet that is sent to the matchmaking server when a player improperly disconnects from a session. It has one key field:

```
struct frpgRequestNotifyDisconnectSession {
      int iPlayerID;
};
```

Once the server sees a specific player ID has received a certain amount of these packets, that player will be severely restricted from matchmaking for the next few days.

The function which communicates this is sub_C740F0. Therefore a malicious player can grab player IDs (as they are unique per-Steam account) and spoof them to the server to get massive amounts of players unfairly restricted with little to no effort at all. For example:

```
sub_C740F0(*data_4777FA8, victim_player_id);
```

I recommend that this functionality be removed, as it is too easily abusable by malicious cheaters, and has a very large potential amount of harm to do to the community.

WandingGhost/DeadingGhost objects**

Please note that the potential RCE described in packet 1/24 can be sent as WandingGhost/DeadingGhost data! This means that the code redirection can be sent via the matchmaking server to players who aren't in a multiplayer session!

Conclusion

I apologise if any details seem vague, as this has been researched over many years and I have neglected to properly document these until now.

I have had an opportunity to see the code of Elden Ring’s closed network test, and can confirm that from static analysis - **a lot of these issues are STILL present in Elden Ring** as of writing this document, I really hope this changes in the future. **Many of the same issues have also existed since Dark Souls: Prepare to Die Edition.**

This document only covers the issues that I have found with the game. All p2p traffic received should have at least some degree of sanity / bounds checking on the data to ensure it can be trusted, and the more the better.