SYNACKTIV

Diablo I

RUMPS A RENNES

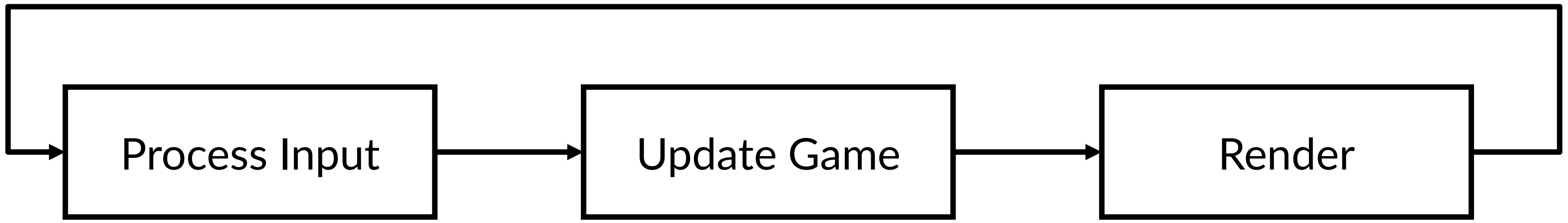
About me

- **Thomas Dubier**
- **@Tomtombinary**
- **Security Engineer at Synacktiv**
  - Offensive security company
  - Pentest, Reverse engineering, Development, Incident response
  - Offices at Paris, Lyon, Rennes, Toulouse, Lille

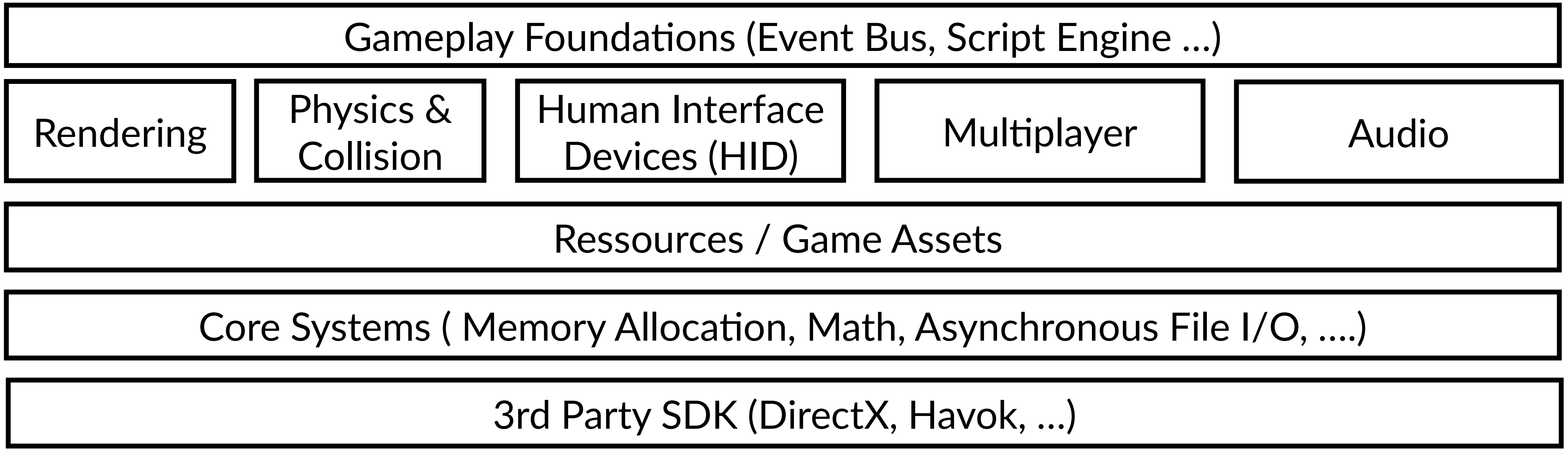
Research Motivations

- **Why look for vulnerabilities in old video games?**
  - To have fun
  - To recycle my old video game collections
  - Interesting when old games are re-released
  - There are always bugs, but sometimes complicated to exploit
- **Focus on RCE (no cheating technique)**

What is a video game ?

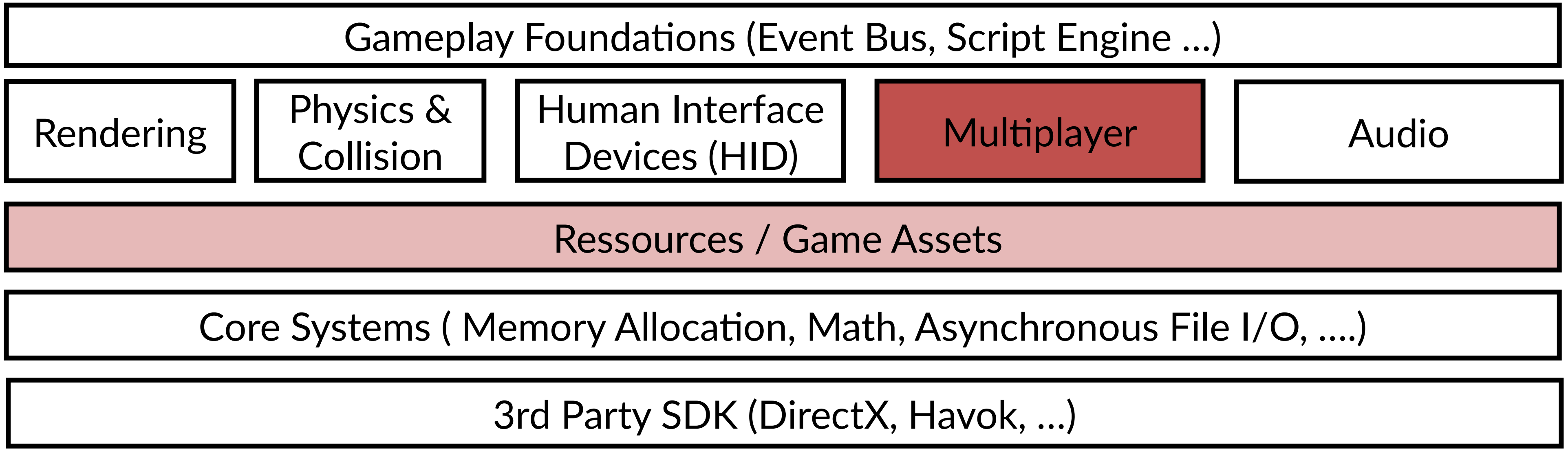


What is a game engine ?


Gameplay Foundations (Event Bus, Script Engine ...)
Rendering
Physics & Collision
Human Interface Devices (HID)
Multiplayer
Audio
Ressources / Game Assets
Core Systems (Memory Allocation, Math, Asynchronous File I/O, ...)
3rd Party SDK (DirectX, Havok, ...)

Source: https://www.gameenginebook.com/
« Game Engine Architecture » by Jason Gregory

Where to Focus ?


Gameplay Foundations (Event Bus, Script Engine ...)
Rendering
Physics & Collision
Human Interface Devices (HID)
Multiplayer
Audio
Ressources / Game Assets
Core Systems (Memory Allocation, Math, Asynchronous File I/O, ...)
3rd Party SDK (DirectX, Havok, ...)

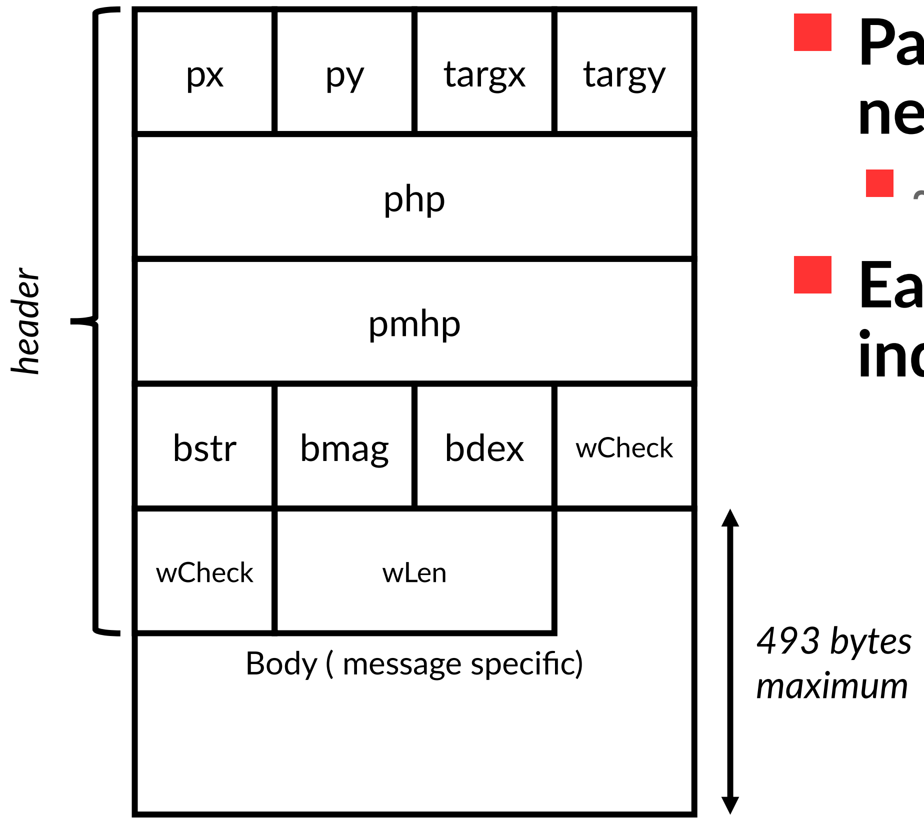
Diablo I

- Developed in 1996
- Re-released version from GOG
  - Windows 10
  - Multiplayer support
- Source code issued from original game reverse-engineering
- https://github.com/diasurgical/devilution

Network Stack

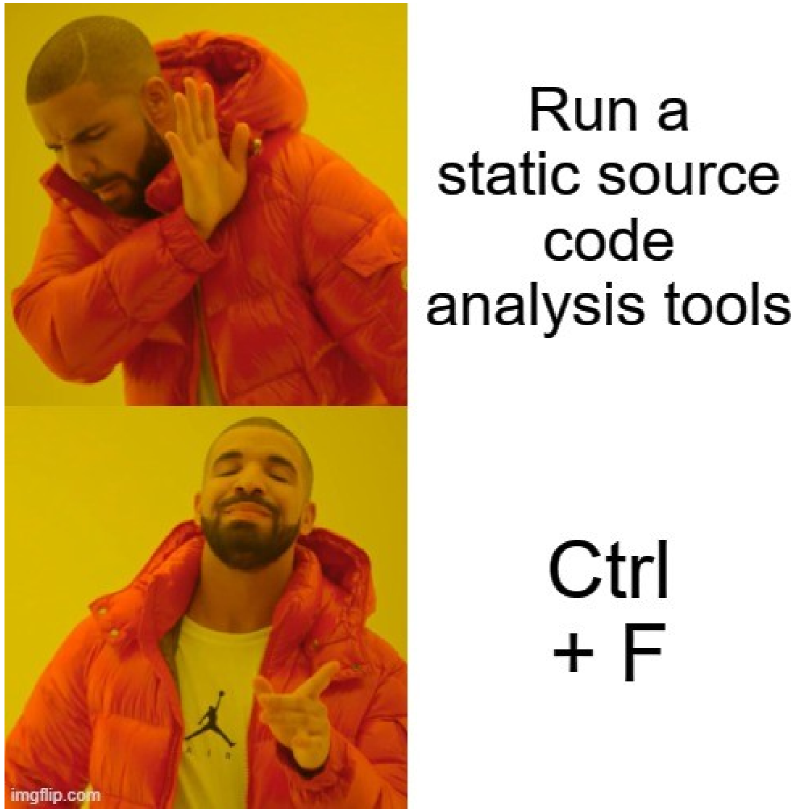
Diablo.exe	*Handle game specific messages*
Storm.dll	*Closed-source, shared between Diablo I, Warcraft II, Starcraft (partially documented)*
ipxwrapper.dll	*Open-source*
Network UDP Stack	*OS implementation*

Attack surface



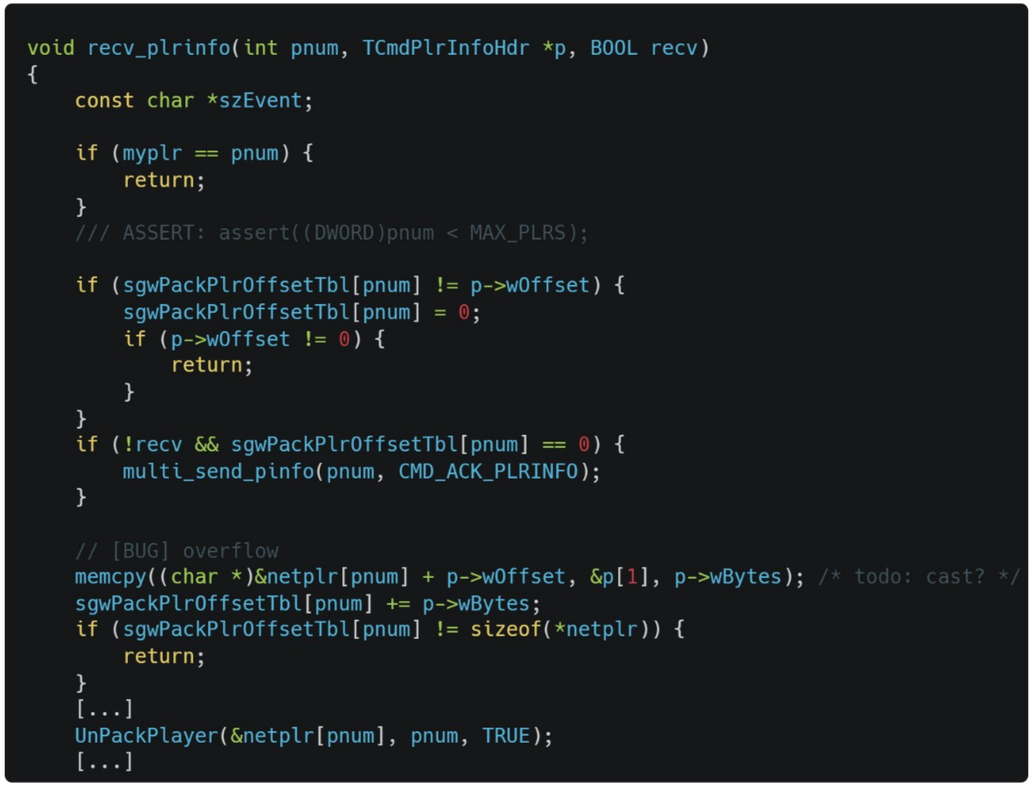
- **ParseCmd handles messages from the network**
  - ~76 different messages
- **Each message starts with 1 byte which indicates its type**

Looking for vulnerabilities



- Old source code
- Search for
  - memcpy
  - strcpy
  - sprintf

Vulnerability



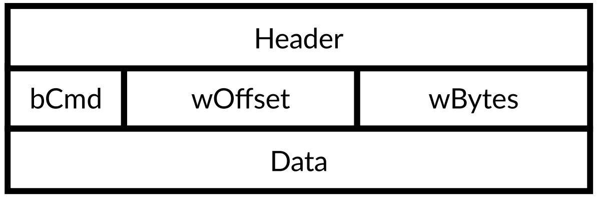
```
void recv_plrinfo(int pnum, TCmdPlrInfoHdr *p, BOOL recv)
{
    const char *szEvent;

    if (myplr == pnum) {
        return;
    }
    /// ASSERT: assert((DWORD)pnum < MAX_PLRS);

    if (sgwPackPlrOffsetTbl[pnum] != p->wOffset) {
        sgwPackPlrOffsetTbl[pnum] = 0;
        if (p->wOffset != 0) {
            return;
        }
    }
    if (!recv && sgwPackPlrOffsetTbl[pnum] == 0) {
        multi_send_pinfo(pnum, CMD_ACK_PLRINFO);
    }

    // [BUG] overflow
    memcpy((char *)&netplr[pnum] + p->wOffset, &p[1], p->wBytes); /* todo: cast? */
    sgwPackPlrOffsetTbl[pnum] += p->wBytes;
    if (sgwPackPlrOffsetTbl[pnum] != sizeof(*netplr)) {
        return;
    }
    [...]
    UnPackPlayer(&netplr[pnum], pnum, TRUE);
    [...]
```

- **CMD_SEND_PLRINFO receives and unpacks information about player**
- **Player information is fragmented in multiple messages**



CMD_SEND_PLRINFO message

Vulnerability

- **Write 0xFFFF arbitrary bytes from &netplr[1]**
- **netplr in .bss (segment containing uninitialized static variables)**
- **No code pointer / vtable ⏸**
- **Can corrupt player array (plr)**
  - Represent states of each players

Vulnerability

- CMD_DLEVEL receives and unpacks level information
- Level information is fragmented in multiple messages
- Same kind of vulnerability

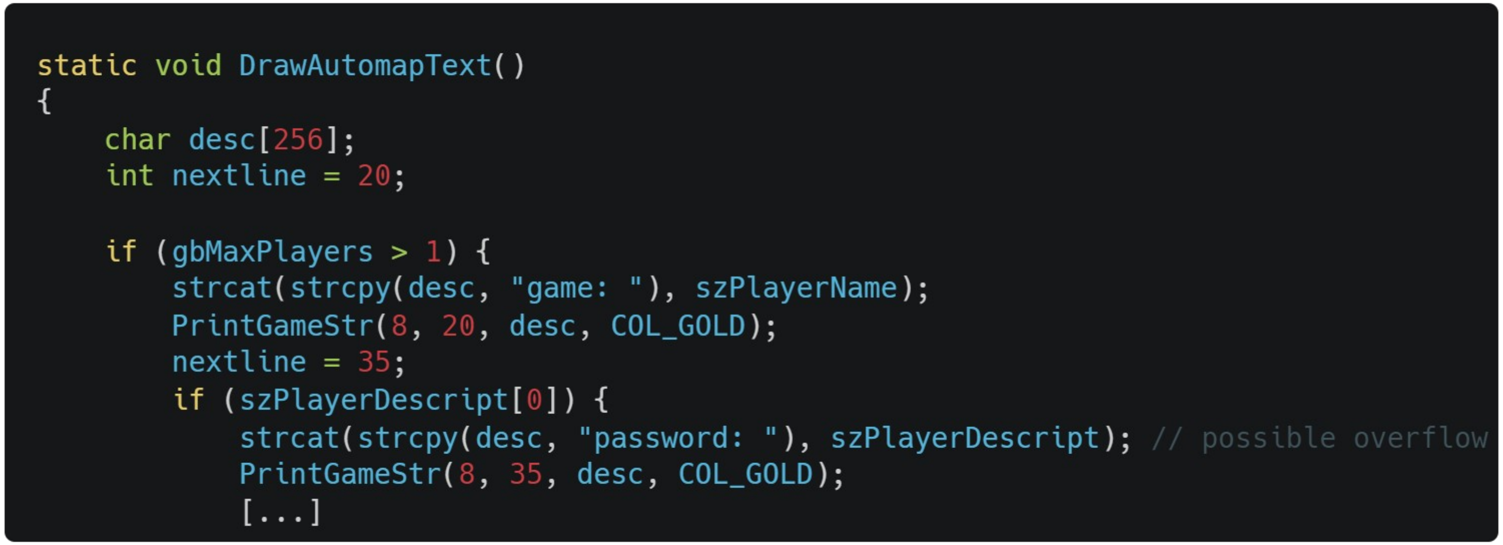
```
static DWORD On_DLEVEL(int pnum, TCmd *pCmd)
{
    TCmdPlrInfoHdr *p = (TCmdPlrInfoHdr *)pCmd;
    [...]
    /// ASSERT: assert(p->wOffset == sgdwRecvOffset);
    memcpy(&sgRecvBuf[p->wOffset], &p[1], p->wBytes); // [BUG] overflow
    sgdwRecvOffset += p->wBytes;
    return p->wBytes + sizeof(*p);
```

Vulnerability

- sgRecvBuf is 4722 bytes length in .bss
- Write 0xFFFF arbitrary bytes from &sgRecvBuf[1]
- No code pointer / vtable ▮▮
- Can corrupt szPlayerDescript
- Can corrupt sgwPackPlrOffsetTlb array
  - Used to receive netplr information

Indirect vulnerability

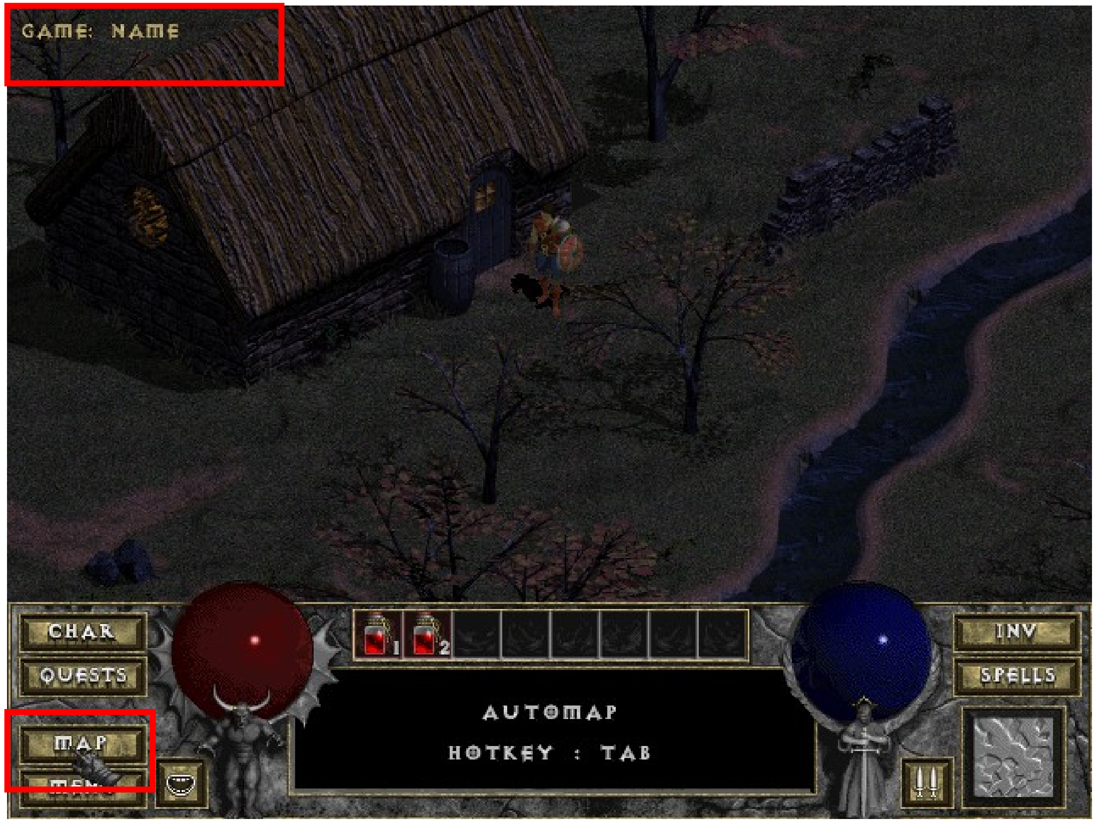
- *szPlayerName* and *szPlayerDescript* can’t be controlled directly
- Each buffer is 128 bytes length



```c
static void DrawAutomapText()
{
    char desc[256];
    int nextline = 20;

    if (gbMaxPlayers > 1) {
        strcat(strcpy(desc, "game: "), szPlayerName);
        PrintGameStr(8, 20, desc, COL_GOLD);
        nextline = 35;
        if (szPlayerDescript[0]) {
            strcat(strcpy(desc, "password: "), szPlayerDescript); // possible overflow
            PrintGameStr(8, 35, desc, COL_GOLD);
            [...]
```

How to trigger stack buffer overflow ?



- Toggle button map
- Display is determined by a boolean variable automapflag
- automapflag is located before .bss ⏸
- Can't be corrupted directly

Take advantage of game loop

- *ProcessPlayers* is called in loop
- plr[0] can be corrupted
- Achieve arbitrary memory OR with *_px* and *_py*

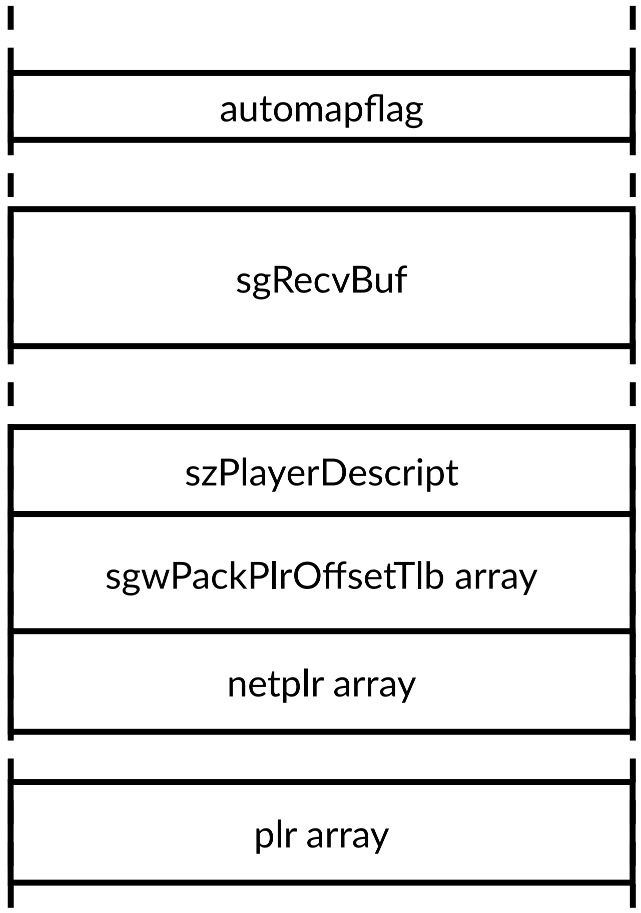
```c
void ProcessPlayers()
{
    [...]
            do {
                switch (plr[pnum]._pmode) {
                [...]
                case PM_DEATH:
                    tplayer = PM_DoDeath(pnum);
                    break;
                [...]
```

```c
BOOL PM_DoDeath(int pnum)
{
    if ((DWORD)pnum >= MAX_PLRS) {
        app_fatal("PM_DoDeath: illegal player %d", pnum);
    }

    if (plr[pnum]._pVar8 >= 2 * plr[pnum]._pDFrames) {
        if (deathdelay > 1 && pnum == myplr) {
            deathdelay--;
            if (deathdelay == 1) {
                deathflag = TRUE;
                if (gbMaxPlayers == 1) {
                    gamemenu_on();
                }
            }
        }

        plr[pnum]._pAnimDelay = 10000;
        plr[pnum]._pAnimFrame = plr[pnum]._pAnimLen;
        dFlags[plr[pnum]._px][plr[pnum]._py] |= BFLAG_DEAD_PLAYER; // arbitrary OR
    }
    [...]
```

Bring all together



- Send CMD_DLEVEL to corrupt szPlayerDescript
- Send CMD_SEND_PLRINFO to corrupt
  - plr[0]._pmode
  - plr[0]._px
  - plr[0]._py
- Player 0 dies and a arbitrary OR is made on automapflag
- automapflag != 0 DrawAutomapText is called
- szPlayerDescript is not null byte terminated => Stack Buffer Overflow

PoC

SYNACKTIV

https://www.linkedin.com/company/synacktiv
https://twitter.com/synacktiv
Nos publications sur : https://synacktiv.com