

HW #4

PwnAdventure3

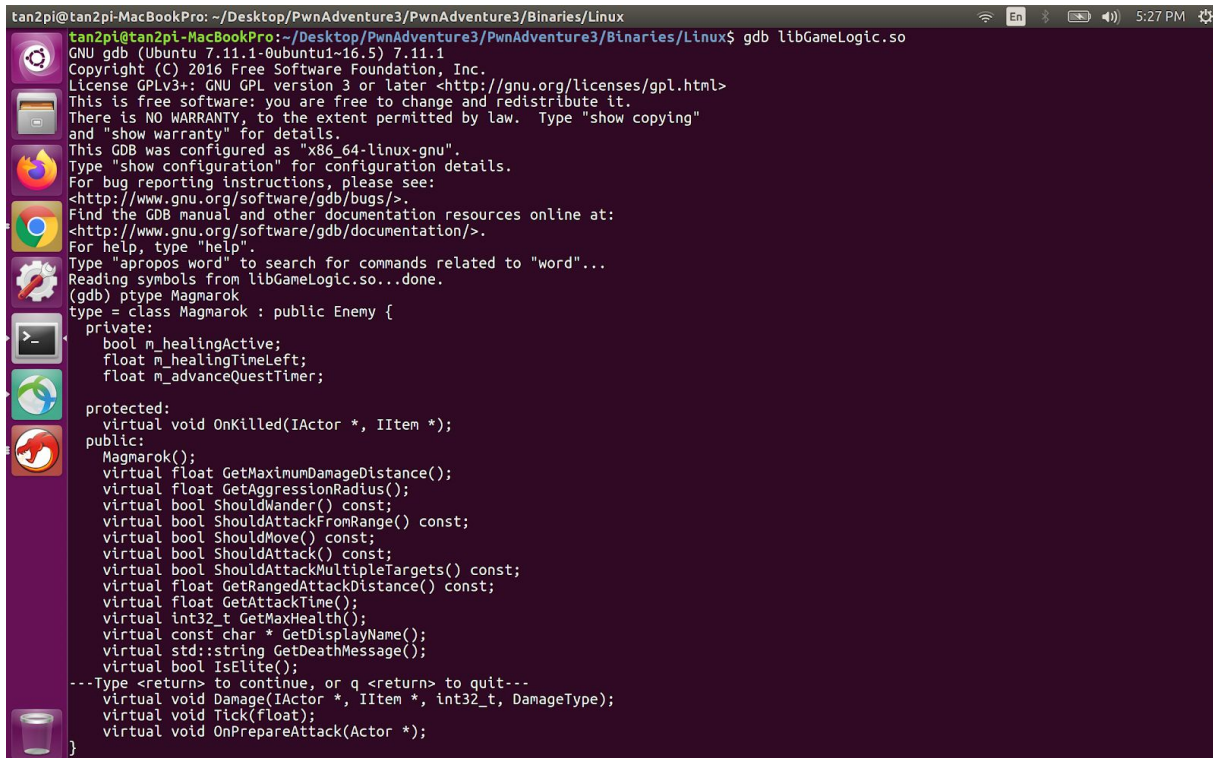
Piyush Tandon, [Github Link](#)

Team Name: BiggertonChungus

Username: tan2pi

Character Name: whoami

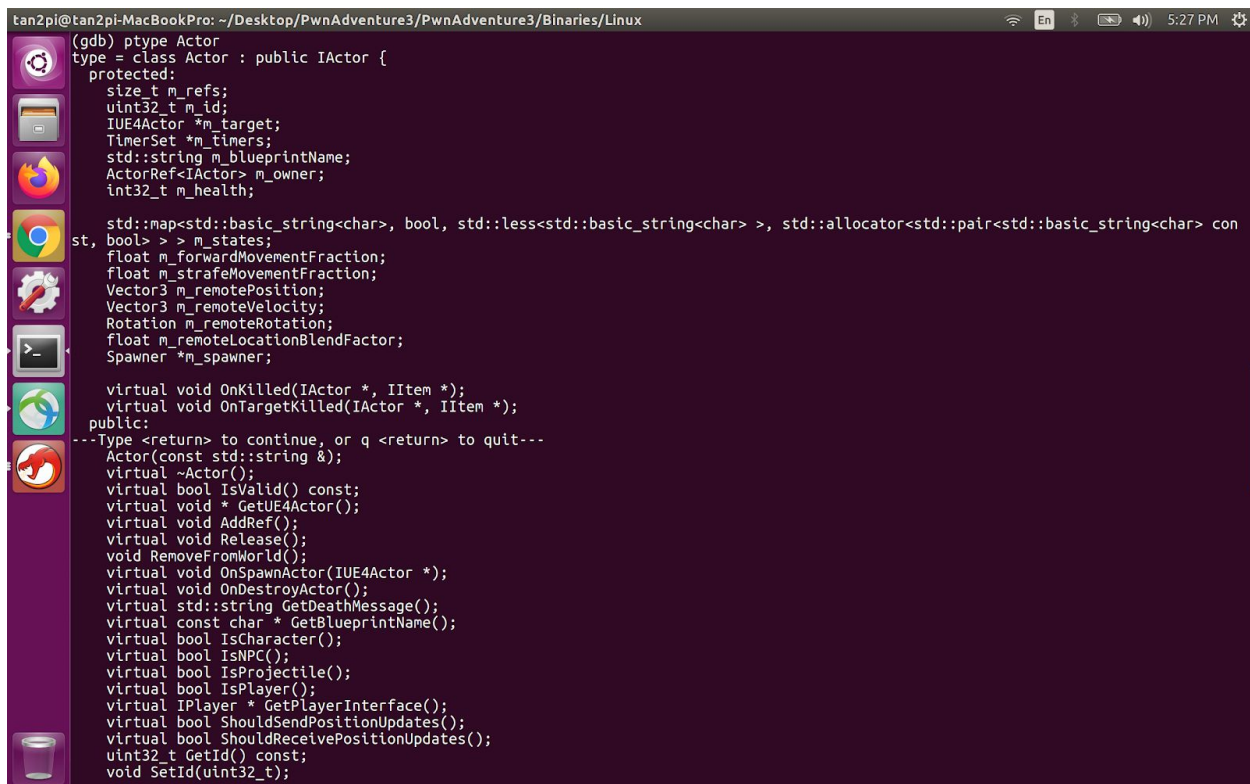
I first noticed I couldn't damage Magmarok at all using the fireball spell, which makes sense. Once I got the ice spell, I could damage him, but only until he hits half health, at which point he heals back to full health. After noticing I couldn't kill Magmarok due to his healing capabilities, I decided to examine the `libGameLogic.so` library to figure out a way to beat it. First, in `gdb`, I loaded the game logic library, and tried to find some pertinent information about the structure of the program. Assuming the writers of the game used OOP, I figured that the Magmarok functionality was encapsulated in a `Magmarok` class. So, using `ptype` Magmarok, I extracted the class members.



```
tan2pi@tan2pi-MacBookPro: ~/Desktop/PwnAdventure3/PwnAdventure3/Binaries/Linux
tan2pi@tan2pi-MacBookPro:~/Desktop/PwnAdventure3/PwnAdventure3/Binaries/Linux$ gdb libGameLogic.so
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from libGameLogic.so...done.
(gdb) ptype Magmarok
type = class Magmarok : public Enemy {
private:
    bool m_healingActive;
    float m_healingTimeLeft;
    float m_advanceQuestTimer;

protected:
    virtual void OnKilled(IActor *, IItem *);
public:
    Magmarok();
    virtual float GetMaximumDamageDistance();
    virtual float GetAggressionRadius();
    virtual bool ShouldWander() const;
    virtual bool ShouldAttackFromRange() const;
    virtual bool ShouldMove() const;
    virtual bool ShouldAttack() const;
    virtual bool ShouldAttackMultipleTargets() const;
    virtual float GetRangedAttackDistance() const;
    virtual float GetAttackTime();
    virtual int32_t GetMaxHealth();
    virtual const char * GetDisplayName();
    virtual std::string GetDeathMessage();
    virtual bool IsElite();
    ---Type <return> to continue, or q <return> to quit---
    virtual void Damage(IActor *, IItem *, int32_t, DamageType);
    virtual void Tick(float);
    virtual void OnPrepareAttack(Actor *);
}
```

The only pertinent variable I could find in the Magmarok class was the `m_healingActive` and `m_timeHealingLeft`. I figured that a current health variable should exist somewhere in a class, otherwise keeping track of health for enemies and players would be painful otherwise. Since Magmarok inherits from the Enemy class, I extracted the Enemy class as Ill. Enemy is also a child class, inheriting from AIActor, and still does not contain any information about current health. After going up the chain of inherited classes, I found the `m_health` variable in the Actor class. So, I know the structure of the Magmarok class encapsulates the Actor, AIActor and Enemy classes. Let's look at some functions.



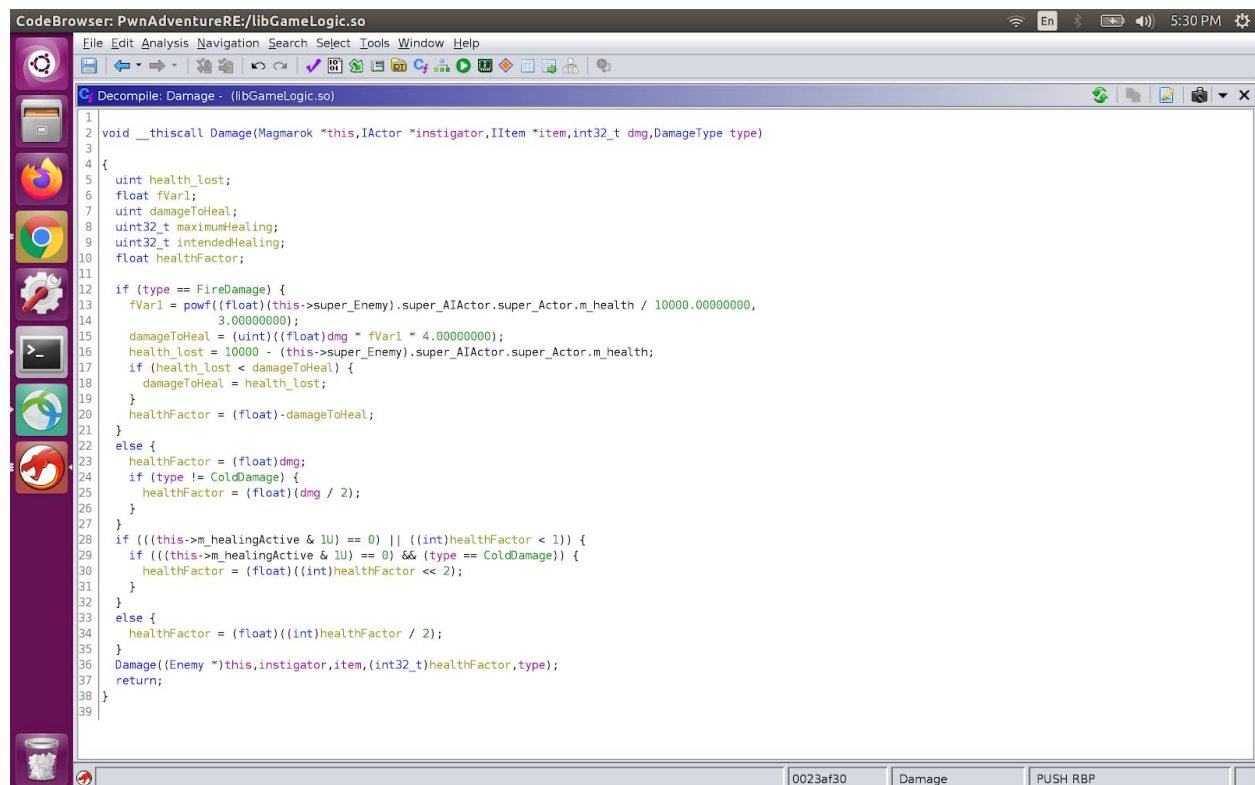
```
(gdb) ptype Actor
type = class Actor : public IActor {
protected:
    size_t m_refs;
    uint32_t m_id;
    IUE4Actor *m_target;
    TimerSet *m_timers;
    std::string m_blueprintName;
    ActorRef<IActor> m_owner;
    int32_t m_health;

    std::map<std::basic_string<char>, bool, std::less<std::basic_string<char> >, std::allocator<std::pair<std::basic_string<char> con
st, bool> > > m_states;
    float m_forwardMovementFraction;
    float m_strafeMovementFraction;
    Vector3 m_remotePosition;
    Vector3 m_remoteVelocity;
    Rotation m_remoteRotation;
    float m_remoteLocationBlendFactor;
    Spawner *m_spawner;

    virtual void OnKilled(IActor *, IItem *);
    virtual void OnTargetKilled(IActor *, IItem *);
public:
    ---Type <return> to continue, or q <return> to quit---
    Actor(const std::string &);
    virtual ~Actor();
    virtual bool IsValid() const;
    virtual void * GetUE4Actor();
    virtual void AddRef();
    virtual void Release();
    void RemoveFromWorld();
    virtual void OnSpawnActor(IUE4Actor *);
    virtual void OnDestroyActor();
    virtual std::string GetDeathMessage();
    virtual const char * GetBlueprintName();
    virtual bool IsCharacter();
    virtual bool IsNPC();
    virtual bool IsProjectile();
    virtual bool IsPlayer();
    virtual IPlayer * GetPlayerInterface();
    virtual bool ShouldSendPositionUpdates();
    virtual bool ShouldReceivePositionUpdates();
    uint32_t GetId() const;
    void SetId(uint32_t);
```

The function that stood out to us was the `Magmarok::Damage()` function, which takes several parameters, the most important being the `dmg` and `damage_type` parameters. Since `gdb` doesn't provide a lot of information about function code, let's open this up in Ghidra and find out some more. Once I load this

into Ghidra, I search for the `Magmarok::Damage()` function, but since Ghidra doesn't demangle these names for us, I have to do a little bit of digging to find it.

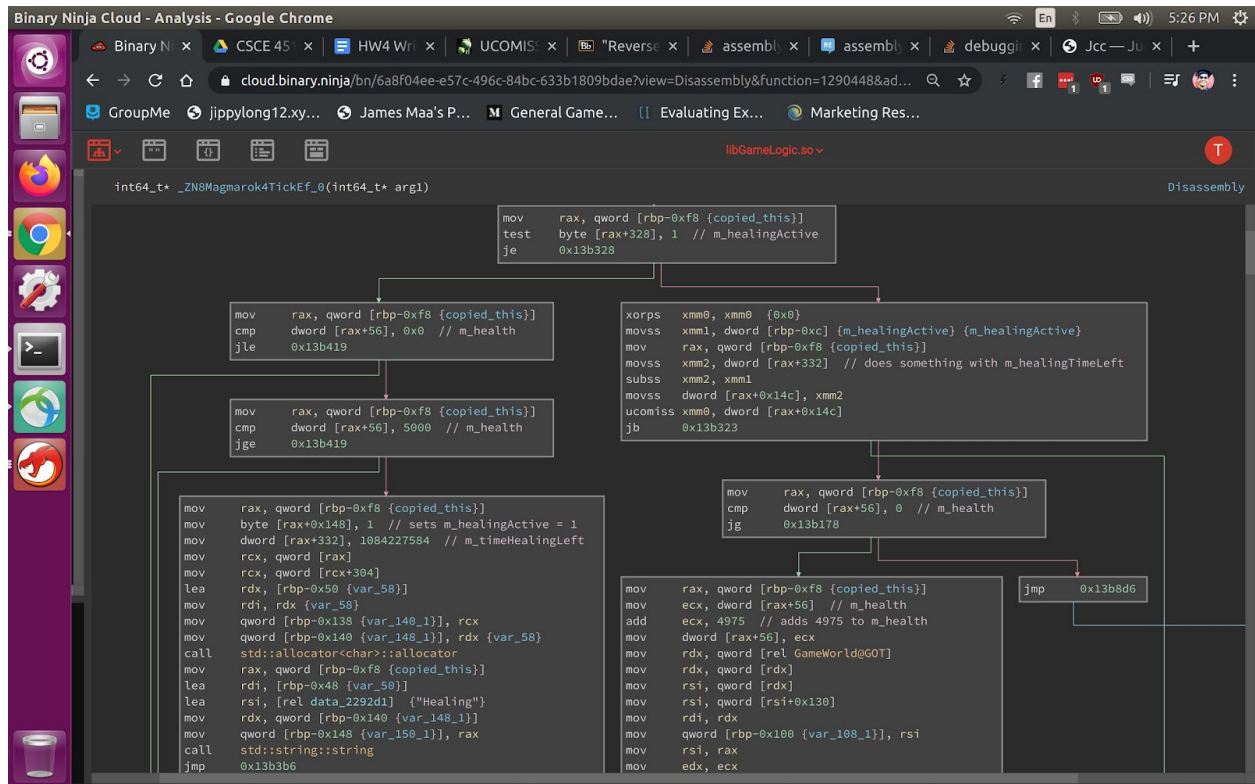


```
1 void __thiscall Damage(Magmarok *this,IActor *instigator,IItem *item,int32_t dmg,DamageType type)
2
3 {
4     uint health_lost;
5     float fVar1;
6     uint damageToHeal;
7     uint32_t maximumHealing;
8     uint32_t intendedHealing;
9     float healthFactor;
10
11     if (type == FireDamage) {
12         fVar1 = powf((float)(this->super_Energy).super_AIActor.super_Actor.m_health / 10000.00000000,
13                     3.000000000);
14         damageToHeal = (uint)((float)dmg * fVar1 * 4.000000000);
15         health_lost = 10000 - (this->super_Energy).super_AIActor.super_Actor.m_health;
16         if (health_lost < damageToHeal) {
17             damageToHeal = health_lost;
18         }
19         healthFactor = (float)-damageToHeal;
20     }
21     else {
22         healthFactor = (float)dmg;
23         if (type != ColdDamage) {
24             healthFactor = (float)(dmg / 2);
25         }
26     }
27     if (((this->m_healingActive & 1U) == 0) || ((int)healthFactor < 1)) {
28         if (((this->m_healingActive & 1U) == 0) && (type == ColdDamage)) {
29             healthFactor = (float)((int)healthFactor << 2);
30         }
31     }
32     else {
33         healthFactor = (float)((int)healthFactor / 2);
34     }
35     Damage((Enemy *)this,instigator,item,(int32_t)healthFactor,type);
36     return;
37 }
38
39
```

Once I view the listing and decompiler for the `Magmarok::Damage()` function, I noticed that Ghidra knows the actual programmatic name for the function, but doesn't label it like that in the symbol tree. But, it seems the decompiler produced a lot of good output for us! So if I hit Magmarok with `FireDamage`, it does some computation and negative damage is dealt to Magmarok, which probably heals him. If I hit him with `ColdDamage`, the damage isn't reduced at all, but other types of damage will have their damage halved against him. And if I try to hit him with `ColdDamage` when he's healing, the `dmg` variable is shifted to the right by 2. This probably reduces the damage dealt to near zero.

So most of the function is pretty standard, but let's look at the healing section of it (line 12), as it seems to be doing something different. The code computes $(m_health / 10,000)^3 \times 4 \times dmg$. The damage amount caused by a `FireDamage` attack is scaled depending on the remaining health of Magmarok,

which, paradoxically, means that Magmarok heals for less if his health is lolr. Perhaps this could be exploited? This scaled damage is then checked against the maximum amount of health that Magmarok can heal, which is `10,000 - m_health`. If the scaled damage is more than this difference, then the healing is set to the difference instead of the scaled damage. The healing variable is negated, which is not done elsewhere. This probably has the functionality of increasing Magmarok's health, when the `Damage()` function is called at the end (the `Magmarok::Damage()` function performs Magmarok specific actions, but the `Damage()` function is used by all enemies). But there's nothing in here about the healing that happens when Magmarok hits half health. The only other function that the healing could be contained in is most likely the `Magmarok::Tick` function. Other functions inside the Magmarok class are either getters or boolean checks, so they likely don't contain the healing functions. A Tick function can also constantly monitor the current health of Magmarok, so the check could be in here. Unfortunately, Ghidra is unable to produce decompiler output for this function. It is able to provide variable names and their corresponding registers, though. But looking at control flow graphs produced by Ghidra is kind of a mess, so I decided to try Binary Ninja Cloud to view the control flow graph (LiveOverflow demonstrated it a bit, and it looked much nicer to use for CFGs). The function prologue is pretty standard here, since BinaryNinja doesn't provide parameter names, I rename them using Ghidra's output in the Listing view for the `Magmarok::Tick` function.



The first check is `test byte [rax+328], 0x1`, followed by a `je` instruction, but I don't know what that value is. Register `rax` contains the pointer `this`, but I don't know the variable pointed to by that offset. In Ghidra, I found the `Magmarok` class in the DataType Manager. In the Structure editor, you can see that at byte 328, the variable `m_healingActive` is stored. So this check determines if healing is already being performed. If healing is not being performed, the next block makes sure that $0 < m_health < 5000$. If the health is less than 5000, then `m_healingActive` is set to true. Additionally, some states are changed (probably). From what I understood from the function, it seems that the healing is actually performed the next time Tick is called. So there's a delay from when Magmarok's health is checked, and when the healing is performed. Going back to the instruction `test byte [rax+328], 0x1`, the block taken if `m_healingActive` is true manipulates `m_healingTimeLeft`, and then adds 4975 to `m_health`. So I now understand how the healing process works, but how can I kill Magmarok?

III, Magmarok's heal doesn't just reset his health, it adds 4975 to his current health. Combined with the fact that there is a delay between checking that Magmarok's health is less than 5000 and actually healing him, maybe there is a way to exploit this. Given this information, I might initially assume that the delay in the healing would allow us to kill him before he healed. But this wouldn't work, because the `Magmarok::Damage()` function likely negates damage taken during healing. But if I look at the `FireDamage` healing calculations, I noticed that there are a lot of casts to different types. The `polr` function requires conversions to `float`, and then the healing is cast to an unsigned integer. But when the `Damage()` function is called, the damage/healing parameter is cast to a 32-bit signed integer. This makes sense, as I know that `m_health` is also a 32-bit signed integer. But due to the differences between unsigned and signed integers, this can be exploited. In `health_lost` variable, for instance, which is unsigned, what if the result of the subtraction is negative? That would mean that `m_health` is greater than 10,000, somehow, but that would result in being able to heal Magmarok above 10,000 HP. This is because the check statement `if (health_lost < damageToHeal)` would always be false if the value in `health_lost` is negative, as the unsigned value would overflow to a super high value. Thus, I gain the ability to heal Magmarok above his maximum health. But what's the benefit of getting his health above 10,000? Since `m_health` is signed, and only 32 bits wide, if I exceed the range of possible values for a signed integer, the integer would overflow back to a negative value, which would effectively kill Magmarok.

That sounds like it could work, but the max value for a signed integer is 2.15 billion, which would take a long time to hit, as our spell would only heal about 50 damage at a time. Luckily, the fire damage is scaled in such a way (formula: $(m_health / 10,000)^3 \times 4 \times dmg$) that if Magmarok's health is more than 10,000, the healing is increased substantially! To put this into practice, I would need to get Magmarok to under 5000 HP using the iceball, and then quickly get him to over

5000 health using the fireball, so that his health now exceeds 10,000. Once his health is over 10,000, I shoot fireballs at him until he dies.

Though it took me a while, I was eventually able to whittle Magmarok down until it died. Just because I was curious, I roughly calculated the number of hits it would take to kill Magmarok, and it ended up being around 40 hits. Forgot to take a screenshot when I got the flag, so here's the next best thing:

