Abertay University

Honours Project Feasibility Study: Artefacts

Isaac Basque-Rice

BSc. (Hons.) Ethical Hacking
Abertay University
Dundee, United Kingdom
1901124@abertay.ac.uk

29th November, 2022

Contents

1 Gantt Chart

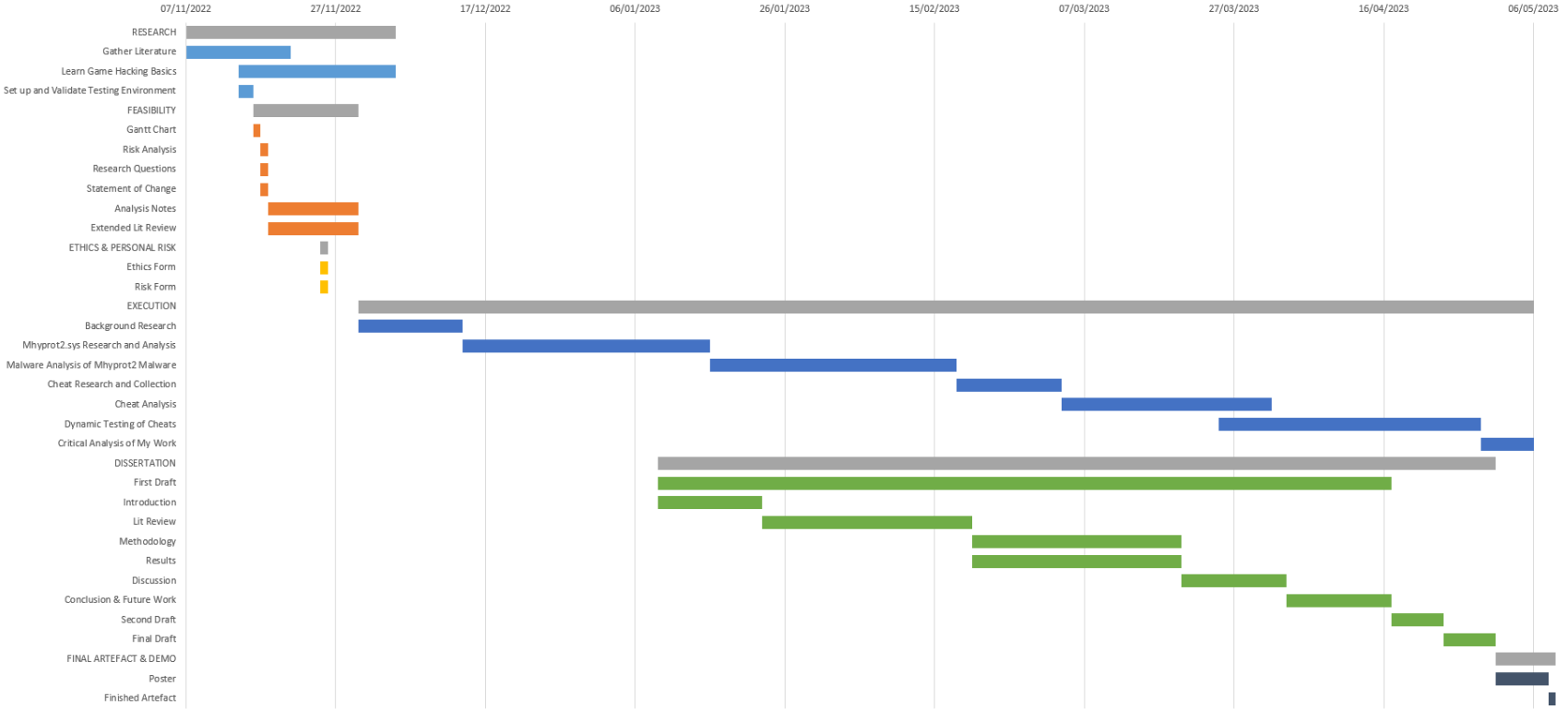

Figure 1: The Gantt Chart for the Video Game Anti-Cheat Security Honours Project

2 Risk Analysis

2.1 Technical Risks

2.1.1 Possible Presence of Malware

Whilst the forums I will be using to download cheat software *do* claim to perform manual malware checks on each uploaded file, there is truthfully no way to be entirely sure that any given file I download will not contain malware. This is especially true as many cheats, in particular process injectors, are often flagged by Anti-Virus software (including Windows Defender) as they interact with process memory in similar ways to genuine malware, as such if true malware is present within the downloaded file, there will be no way to tell at-a-glance, so to speak.

The mitigation for this concern will be to download these files first in a virtual machine, take a hashsum of them for later use, and then make use of a number of malware detection tools I have previously familiarised myself with during my mini-project. The main one herein will be VirusTotal, which first shows how many vendors treat the software in question as a malware, but also shows a community score, which may be more trustworthy in this instance. Failing this, of course, it is possible for me to look at what the process does when running on VirusTotal and validate that myself, as well as checking the forums. If the file is non-malicious I will proceed to install it on my host machine, check the hashsum, and then begin analysis.

2.1.2 Technical Faults

There are several technical points of failure that should be borne in mind during the execution of this project. Failure of the computer or any internal hardware, virtual machine, or USB device is a risk that must be taken into account

The mitigations here are twofold: Firstly, I will employ a number of backup solutions (such as cloud based on Outlook, backing up to both PC and Laptop, using GitHub for code, etc.) in order to prevent any serious loss of time and effort. I also intend to use university devices for some work, which have significant amounts of resilience to data loss built in, particularly in the hacklab and netlab. Should my PC fail I intend to switch to using these and my laptop full-time, or, funding permitting, purchase a new device.

2.2 Non-Technical Risks

2.2.1 Breach of Terms and Conditions

In many cases it may be considered a breach of terms of service or end user license agreement to run cheats against a video game. In such such cases, permission will be sought, either implicitly, i.e. through a bug bounty program that expressly includes anti-cheat software (Epic Games' Easy Anti-Cheat, as a prime example), or explicitly, through attempted contact with the developers. Failing this, a conversation will be had with my supervisor around how to move forward.

2.2.2 Physical Concerns

Although this project may not be physically intensive in the traditional sense, a number of health concerns have been identified with extended periods of work at a computer, such as the work I will be undertaken. These risks include, but are not necessarily limited to, carpal tunnel syndrome, eye strain and headaches (as well as other visual issues), fatigue and stress, as well as problems with posture.

The majority of these issues will be mitigated by ensuring I have enough break time where required. Eye strain and headaches can be mitigated by ensuring I have enough light and that my screens are correctly positioned.

2.2.3 Interaction With Other Players

Some of the games that have anti-cheats that will be tested are multi-player. This naturally means that interactions with other individuals may be possible if the cheats themselves are to be tested, which could result in a leakage of personal details, a breach of the GDPR, and so on.

However, the functionality of the cheats is not the thing that is being tested in this instance, and as a result, playing the games themselves, and hence interacting with others, is relatively unlikely. In the event that both myself and my supervisor deem it necessary to interact with others in the course of my research, I will attempt to go through the proper channels to recruit and gain informed consent from others in order to assist me and prevent unnecessary intrusions into non-participants' experience.

2.2.4 Scope Creep

As is the case with many projects, particularly the ones dictated by a single person or small number of people, scope creep, the tendency towards adding

new aspects to the research project at will, is a concern. The mitigation for this issue is to simply consult the project supervisor before any changes to the substance of the research are made.

2.3 Risk Analysis Matrix

Probability of Risk	**Impact of Risk**		
	Low	Moderate	High
High	Technical Faults		
Moderate		Breach of Terms and Conditions	Presence of Malware
Low	Scope Creep	Interaction with Other Players	Physical Concerns

Table 1: Risk probability impact matrix for the video game anti-cheat security honours project

3 Research Questions

The following is a list of questions my research sets out to answer:

1. How vulnerable are anti-cheat services, particularly the ones that make use of kernel ring 0 permissions, to compromise by a malicious actor?
2. How can the methods used in video game cheat and anti-cheat software be applied in a wider security context?
3. Given anti-cheat platforms' historically opaque approach to their technology, what insight can analysis of cheat software give to vulnerabilities in the anti-cheat software it is attempting to exploit?

4 Statement of Change

There have been no significant changes to my project since submission of my project proposal.

5 Extended Literature Review

5.1 An Overview Of Video Game Security

5.1.1 Security in Online Games: Current Implementations and Challenges

It is of the utmost importance that a holistic view of the security of the video game industry is taken into account. This view is initially provided by Parizi et al. (2019), who correctly identify the link between cheats, hacks, and attacks on video game infrastructure whilst taking care not to conflate the three. Many types of attacks are laid out in this paper, with specific reference to social engineering, infrastructure attacks like Man in the Middle and Distributed Denial of Service attacks, as well as keyloggers and, of course, Trojans. In this way, the paper makes the point that video games are awash with possible vulnerabilities, and remedying them is of the utmost concern.

The authors of this study conclude with a series of recommendations around how to improve the security of video game platforms, with suggestions such as cryptography, anti-cheat (referred to as "software which monitors user's [sic.] actions to prevent any misuse of the software"), and frequent security testing. A shortfall of this paper, however, is its simplicity, in that, a problem is identified in high-level terms, and the solution is prescribed in the same terms, which are functionally unactionable. Additionally, no mention is made throughout the paper of vulnerabilities within the video game software itself, which is undoubtedly a huge oversight.

5.1.2 Information security as a countermeasure against cheating in video games

To contrast with this, Mikkelsen (2017) provides an extensive and technically robust thesis, which likens video game cheating to the concerns of the information security industry, provides in-depth insight into both the causes of and solutions to cheating from an information security perspective. The author's comments, "Most cheating in video games is possible due to information being accessible outside the intended frames of the game developer", are particularly salient given that this may as well be a definition of what information security professionals are trying to avoid.

Mikkelsen separates the areas of concern, in their view, into two general categories, which are networking security and application memory tampering. Much of the network security concerns and remedies regard users' ability to securely place personal details, such as financial information, into in-game or

in-launcher forms. Many of the recommended fixes are already implemented into many popular third-party game engines.

It is the application memory tampering, however, that has achieved more attention from the author, in this case, "as this is the vulnerability that are [sic.] most easily and commonly used by cheaters while also being the area where there are less options available for current game developers to protect their games". The author readily admits that this area is one in which it is particularly difficult to test, and difficult to provide solutions for due to the fact most video game developers don't want to write low-level drivers, however, they imply that a game engine that could implement changes like this as default would enjoy success on the market.

The particular fix the author suggests is the implementation of sandboxing or full virtualisation later between the game and the hardware, outwith the scope of the host operating system. This, naturally once again, would be difficult to implement, but it would allow for greater control over the system on which the game is run from a developer's perspective, and would require little modification to the host device, meaning kernel-level drivers become a functional non-issue.

5.1.3 Video Game Security: The Future Belongs to Machines

In terms of the direction of flow for video game security in the future, Godsey (2017) appears to view Machine Learning as the primary solution given that, firstly, "there are too many players [...] that produce too much data for any team of live humans to comb through", and secondly, the data these players tend to create can be analysed via a number of metrics including behaviour, patterns, IP addresses, and so on, with illegitimate player behaviour being, theoretically, easily detectable in this way. These points together, in the author's view, result in machine learning being the obvious remedy to cheating.

The author categorises the usefulness of ML in the video game context into three areas, these being ML as leverage, as detective, and as watchdog. Based on the descriptions, however, these three offer fundamentally the same thing, that is, the idea that machine learning can detect many examples of suspicious behaviour before their human equivalent could. This is, of course, a possibly accurate observation, however Godsey provides no supporting evidence for their case and no examples of how this has benefited the video game industry to date.

5.2 Current Cheat and Anti-Cheat Techniques

5.2.1 Cheating in video games: The A to Z

In order to learn the methods by which anti-cheat services operate, it is clearly crucial to first know how cheats themselves work. One organisation that prides itself on counteracting video game cheats, and hence has a depth and breadth of knowledge about cheat software, is Irdeto, a Netherlands-based cybersecurity organisation. Irdeto's Blaukovitsch (2022), therefore, provides an excellent overview of what is viewed by them and their organisation to be the primary methods of cheating, particularly in online games, used in the modern day.

These "Common Tricks for Cheating in Video Games" are varied and numerous, from aimbots and triggerbots, which allow cheaters to have perfect aim and automatically fire when they are targeting the enemy, to wallhacks, which allow targets to be seen through walls, and ESP and Radar, which adds numerous elements to the user's HUD to reveal information, such as opponents' locations, ability cooldowns, and so on.

This article, as with a significant number of others within this area, is particularly light on information regarding precisely *how* these cheats work, i.e., what technical methods they make use of in order to facilitate cheating. However, it is possible to gleam rough areas cheats must be used by taking into account recommendations Blaukovitsch, and authors like him, make.

These recommendations include, but are not limited to, prevention of unauthorised patching and memory access, protection of communication protocols video games use, isolating the process (thereby preventing other processes from hooking on), and obfuscation and virtualisation of the game process. From this, we can come to understand that memory access and manipulation are core to what cheats attempt to achieve.

5.2.2 Comparative Study of Anti-cheat Methods in Video Games

Carrying on from this, it is also undoubtedly important in this context to gain a good understanding of what is currently being done to counteract cheating. Whilst many of the specific techniques are, naturally, kept secret to prevent cheat developers from gaining leverage, Lehtonen (2020) provides an excellent comparative analysis of the overall methods they use. The author, much like many of their contemporaries, characterises the relationship between cheat and anti-cheat developers as a 'cat-and-mouse game', as detection is crucial to AC developers and avoiding detection is equally as crucial to cheat developers, thereby creating a race to the bottom, so to speak, between the

two factions. In this sense the author sees it as "largely [resembling] anti-virus development".

In terms of technology used, Lehtonen separates them into four server- and five client-side anti-cheat methods and compares them across 5 separate criteria. Note that due to a number of factors including the ease with which they can be analysed, enhanced security concerns, and increased threat to the end user if they were to be compromised, more focus will be placed herein on client-side anti-cheat software. One of the conclusions they came to was that server-side anti-cheat is much more resistant to tampering, but at the cost of not being accurate enough for primarily client-based cheats, such as wallhacks, as a result, there will likely be much room for client-side anti-cheat software in the present moment, as well as in the near future. This is of significant note as many of the security concerns many people have regarding anti-cheat concerns software installed on their own devices.

The client-side anti-cheat methods described by the author are as follows. Firstly, code encryption, which is frequently done via a process known as 'packing', essentially a form of compression, which makes static analysis of anti-cheat significantly harder if the packer is unknown (but equally trivial if it is known). This technique is suitable to all games and non-invasive, however, is not tamper resistant, can be difficult to implement, and may incur significant overhead (performance costs).

Next, file verification via hashing, wherein all the files required by the game program are hashed and checked on a regular basis to ensure no tampering has occurred. This issue has a functionally non-existent overhead and is easy to implement, but, as the author points out, is extremely easily circumvented and hence not suitable for use on its own, when combined with code encryption, however, it becomes a useful tool.

Detection of known cheat programs can be done in a similar way to how anti-virus works, comparison of hashes or process names on the system to known cheat software, however, this can be easily circumvented by changing the process name and refactoring code to create a different executable. A more complex detection system can, of course, be created, but this decreases the ease with which it can be implemented. Sending data to and from the server as opposed to interfering with game logic means overhead is minimal, however, it is easily tampered with by individuals who possess reverse engineering skills, and it is particularly intrusive, possibly allowing for sensitive information to be revealed through program names, etc.

Memory obfuscation is the process of either relocating, encrypting, or performing both actions on crucial pieces of data within the heap of a program. This can be crucial as, in many video games, important variables such as values relating to the player (health, location, money, etc.) as well as

variables relating to the player's surroundings (enemy locations and so on) are stored in the same location in memory if no intervention is made, and as such can be accessed or even altered in some cases by the user using programs such as CheatEngine. A concern many developers may have is the difficulty with which this is implemented, as well as its susceptibility to tampering on the client side, however, its powerful reach and ability to obfuscate other anti-cheat processes as well as the game itself makes it a powerful tool in anti-cheat developers' arsenal.

Finally, kernel-based anti-cheat drivers are, as the name suggests, components of anti-cheat software that reside in the kernel space. This allows them to "spy on requests for interfacing with the game process memory". This gives a great degree of control of the system over to the anti-cheat process when the game is running. This is an exceptionally tamper-resistant method of client-side anti-cheat as access to the kernel is heavily restricted in Windows for the average user. This does, however, mean that it is a particularly difficult thing to implement, and is, of course, extremely invasive.

5.3 Vulnerable Driver Concerns

5.3.1 What's the Deal With Anti-Cheat Software in Online Games?

Many users have concerns regarding kernel-level anti-cheat drivers being used on their system for reasons of system stability, security, and of course, privacy. The article by Menegus (2022) seeks to calm some of these fears through an interview with Netragard founder Adriel Desautels, wherein she states, amongst other things, "even when we're delivering the most advanced level of that service, we don't need to use attacks that go down [to the kernel level].", implying that vulnerability exploitation at the kernel level is the preserve of state actors exclusively and standard users should not have concerns.

This seems, unfortunately, an incorrect assessment. Kernel-mode rootkits have been known for a long period of time to make use of vulnerable signed kernel drivers (Help Net Security 2022), and, indeed, the anti-cheat driver `mhyprot2.sys` has been used in the wild to disable anti-virus software prior to ransomware deployment (Toulas 2022). The rebuttal to this point could come later in the article, with the reference to a Street Fighter V driver bug that allowed for arbitrary kernel-level execution, however, the speed with which this was noticed and rectified was such that any opportunity for exploitation in the wild was likely missed quickly.

5.3.2 Ransomware Actor Abuses Genshin Impact Anti-Cheat Driver to Kill Antivirus

As mentioned, a vulnerable kernel-level anti-cheat driver, known as `mhyprot2.sys`, has been used in the wild in a ransomware chain of attack. Soliven and Kimura's 2022 research demonstrates that even in an environment with "properly configured endpoint protection" (otherwise known as anti-virus), this driver can be used in the early stages of an attack to disable this protection in (theoretically) any case, independent of whether the game it belonged to, Genshin Impact, is present on the system.

This was achieved by tricking the user into running a process called `avg.msi/avg.exe`, a malware dropper masquerading as AVG Security, which then dropped the driver and another executable onto the desktop. The second process, `kill_svc.exe/HelpPane.exe`, loaded the vulnerable driver using `NtOpenFile`, followed by a list of AV processes that it parses and passes a control code to kill the processes if they are running on the system. This is, naturally, followed by the normal execution of a Ransomware payload.

The authors point out that this driver become well-known in the Genshin Impact community as it would not be removed post-uninstall and allowed for privilege bypass on any system it was installed on. Despite this, it was not accepted by the developers of the game as a vulnerability and, to date, has not been fixed. As a result, it is likely that this is still an issue, especially due to the fact that it is a legitimate driver that cannot trivially be removed once distributed. However, only a limited number of versions of `mhyprot2.sys` versions can be used, therefore hash values can be monitored in order to prevent exploitation in most organisations.

5.3.3 Driver-Based Attacks: Past and Present

The above exploit generally falls into the category of 'Bring Your Own Vulnerable Driver' (BYOVD), the process by which a malicious actor installs a legitimate and vulnerable driver onto a system, exploits the vulnerability therein, gains kernel ring 0 access to the device, and uses that as leverage to hide deeper within the system. Baines's (2021) article on this, contrary to Menegus's, states that this kind of attack is definitionally worthwhile for attackers "because they are seen in the wild", followed by a not insignificant number of examples, including several well-known APT attacks, attacks by larger groups, and so on.

The authors have also written a considerable amount on the use cases, as they see it, for a BYOVD-based attack, the primary reason being to bypass Driver Signature Enforcement, Microsoft's system for ensuring all kernel-level

drivers are legitimate, by allowing for unsigned, malicious drivers to be loaded through vulnerable but legitimate ones. Once this is achieved, performing actions such as arbitrary code execution, overwriting data, unhooking Endpoint Detection Response callbacks, and crashing the entire system, would be trivial.

5.3.4 What Does It Take to Catch a Cheater in 2020?

Naturally, much of the onus on preventing cheat software and malware more generally is expected to fall on the operating system itself, however, Greshishchev and Zuydervelt (2020) demonstrate how this is not to be treated as the case in Windows. Microsoft's security model in Windows, whilst relatively robust, is shown to be nowhere near adequate enough in the average user's case.

The model in question begins with 'Secure Boot', which validates the digital signatures of bootloaders and Operating System files, however, to make use of this requires a Windows Trusted Platform Module (TPM), which does not have support on versions of Windows earlier than Windows 8, thereby discounting approximately 14% of users out of hand. In addition to this, of the many devices that do theoretically support Secure Boot, many older devices do not have full support, and only 10% of machines that meet full Secure Boot requirements have it enabled, despite Microsoft insisting it is enabled by default. Additionally, Trusted Boot, which requires drivers to be signed (and therefore theoretically limits the abilities of cheat devs to make use of kernel-level cheats), cannot be active without Secure Boot.

The authors also identify the fact that anti-cheat detection methods are, thanks to the introduction of several post-boot security features intended to impact kernel-mode execution, subject to the limitation placed on AC developers by Microsoft. These limitations are not present for cheat developers who are naturally able to use illegitimate means (vulnerability exploitation etc.) to develop cheat software.

The primary restriction of note herein is Kernel Patch Protection (KPP), which "monitors whether key resources or kernel code has been modified, if so, it will initiate a shutdown of the system". Prior to the introduction of this system, anti-cheat and anti-virus developers would monitor user-mode Windows API calls by either hooking into the Windows kernel or byte-patching kernel code. Restricting access to this method improved system stability but forced AC and AV developers to use formal methods for monitoring API calls, (`ObRegisterCallbacks`, for example) which allows cheat developers to anticipate how their cheats will be detected and circumvent them.

6 Notes on the Xenos Process Injector

- using Xenos_2.3.2 as this is one of, if not the most popular injector on UnknownCheats (655,245 DLs)
- Can be found here w/ forum thread linked within

6.1 Why Choose Xenos?

- Originally was thinking of dumping VAC modules but decided against as it's not *directly* related
- Instead decided to RE a cheat or two
  - probably from CSGO or GTA V as both are exceptionally popular and easy to hack.
- Looking at the following files as possibilities:
  - Modest Menu
    - Pros:
      - Exceedingly popular
      - Cheat in and of itself
    - Cons:
      - For GTA V, a paid game
  - Xenos
    - Pros:
      - Generic software
      - Works to inject DLLs across a wider variety of games (incl. GTA)
      - Does do Kernel-Level stuff
    - Cons:
      - Detected as malware & because of virtualisation detection software, dynamic testing in a VM is difficult (consult Hubert)
      - Also just an injector, not cheat in and of itself
  - CSGhost
    - Pros:
      - CSGO is f2p so when i create a new steam there will be no issues
    - Cons:
      - Only works with CSGO
      - Also detected as malware
      - Injector
- Will probably go with Xenos for this demo, this is for a few reasons
  - It's generic

        * Discovered later this means most of its interactions are with windows (Windows Hacking Library etc.)
    - Injectors are what cheats use to piggyback off of to bypass anti-cheat and security mechanisms, therefore will likely be more relevant than the actual cheats themselves
    - Kernel level capabilities
    - Have the capacity to reverse it outwith a Windows environment

6.2 Description

- Supports x86 and x64 processes and modules
- Kernel-mode injection feature (driver required)
- Manual map of kernel drivers (driver required)
- Injection of pure managed images without proxy dll
- Windows 7 cross-session and cross-desktop injection
- Injection into native processes (those having only `ntdll` loaded)
- Calling custom initialization routine after injection
- Unlinking module after injection
- Injection using thread hijacking
- Injection of x64 images into WOW64 process
- Image manual mapping
- Injection profiles
- Manual map features:
    - Relocations, import, delayed import, bound import
    - Static TLS and TLS callbacks
    - Security cookie
    - Image manifests and SxS
    - Make module visible to `GetModuleHandle`, `GetProcAddress`, etc.
    - Support for exceptions in private memory under DEP
    - C++/CLI images are supported (use 'Add loader reference' in this case)
- Kernel manual map features are mostly identical to user-mode with few exceptions:
    - No C++ exception handling support for x64 images (only SEH)
    - No static TLS
    - No native loader compatibility - Limited dependency path resolving. Only API set schema, SxS, target executable directory and system directory
- Supported OS: Win7 - Win10 x64
- Changelog: +V2.3.2 - Win10 RS4 update support

6.3 Pre-Decompilation tasks

- Checked the file hash on VirusTotal
  - Hash for the archive is `06b12c6d7ece840a3b805e7a498a4af695f793a053a63b887f6c7395b3138bad`
  - Visible on the download page
  - I already know Windows detects this as a virus *but* VT is great for details and such.
  - VT Link for the Xenos File itself here
  - And for another version here
- Windows Portable Executable
- Compiled with MS Visual Studio C/C++ Compiler
- Is definitely a PE Injector
- Interesting that it's detected as malware by so many services when the community scores it as non-malicious, could be because it can be used for malicious purposes? Unsure
- I have not performed the full pre-decompilation methodology as I am confident both from my research and the above in VT that it is not genuinely malicious, and I also have the benefit of knowing beforehand what it does.

6.4 Notes from README

- Xenos comes with a README.txt file
- Can either
  - Select an existing process
  - Start a new process
  - queue a process and wait for you to start it before injecting
- List of images u want to inject
- Injection Type
  - 2 w/out driver
    - Native inject - `LoadLibraryW/LdrLoadDll` in newly created or existing thread
      - `LoadLibraryW`: WIndows API function, "Loads the specified module into the address space of the calling process. The specified module may cause other modules to be loaded"
      - `LdrLoadDll`: Undocumented function, presume it loads DLLs into process space
    - Manual map - manual copying image data into target process

memory without creating section object
  - 3 w/ driver (Kernel)
    * New Thread - `ZwCreateThreadEx` into `LdrLoadDll`
      · `ZwCreateThreadEx` also undocumented, likely part of `Wdm.h` given the `Zw` prefix
    * APC - async procedure call into `LdrLoadDll`
    * Manual map - same as above but in kernel
- Native Loader options
  - Unlink module from:
    * `PEB_LDR_DATA`
      · Specific variables from this struct
      · `InLoadOrderModuleList`
      · `InMemoryOrderModuleList`
      · `InInitializationOrderModuleList`
    * `HashLinks`
    * `LdrpModuleBaseAddressIndex`
    * all `LIST_ENTRY` structs
  - Erase PE headers
  - Use existing thread instead of creating a new one
- Manual Map options
  - Insert module record into `InMemoryOrderModuleList` /`LdrpModuleBaseAddressIndex` and `HashLinks`
    * Used to make module functions work with manually mapped image
      · `GetModuleHandle`
      · `GetProcAddress`
  - Manually resolve imports
    * Imports will be manually mapped instead of using `LdrLoadDll`
  - Wipe headers after injection
  - Ignore TLS
  - Don't create custom exception handlers
  - Conceal memory
- (Could very easily be used in malware drop if it wasn't already detected by MS, especially because. . .)
- Can be used on the command line

6.5 PE Studio

- 298 blacklisted strings
- 53/71 virustotal score

- 4 embedded files at `0xE0A78`, `0xF0820`, `0xFFBB0`, `0x20F358`
  - Unable to locate these during decompilation
- ran with admin privs
- 58 blacklisted functions
- 1 blacklisted library (`dbghelp`)
  - 1 function, `MiniDumpWriteDump`

6.6 Decompilation

- Due to the fact Xenos is detected as a virus I've decided to use my Remnux VM (that I used for my MiniProject)
  - Remnux is a debian-based operating system designed for Reverse Engineering, similar to Kali but for RE
- Unsure at this stage how to dynamically analyse it, cross that bridge when I come to it
  - Likely at this stage that I will perform this later on, I don't want to run the risk of getting an account/device banned
- User r2's `s main` command to locate main in ghidra
- Confirmed by checking its xrefs and noting that the only reference to this function is `entry`
- Performing string analysis using FLOSS to locate salient strings that may help describe specific functions
- Lots of references to "BlackBone"
  - Windows Memory Hacking Library
- Marking everywhere with the word "inject"
  - "Injection initiated" present, wonder what trips this?
  - Full message: `"Injection initiated. Mode: %d, process type: %d, pid: %d, mmap flags: 0x%X, erasePE: %d, unlink: %d, thread hijack: %d, init routine: \'%s\', init arg: \'%ls\'"`
  - Passed as parameter to separate function
  - Control characters must be referenced somewhere
  - A few functions down I found a reference to `___stdio_common_vsprintf_s`, which is a printf statement and returns the number of characters written
- Decided to start using the imports like I would during a static malware analysis
  - In particular using the ones described in the readme (could not find)
- Switched tack at this stage, decided to go down a research-based route as I felt I wasn't getting anywhere exclusively statically reversing it

- Found this github repository seemingly containing the source code

6.7 Code Review

- Appears to make extensive use of BlackBone
  - Memory hacking library
  - Pretty much handles everything for us, seems like Xenos is functionally just a GUI to interact with BlackBone
  - Lots of references in code to modules and functions that use BlackBone
- main
  - `wWinMain`, documentation here
  - Dumps all memory
  - creates file extension association
  - instantiates object for `mainDlg`
  - logs Operating System Information
  - if the action to perform is not to run the profile
    - Dialog is run as a modeless window, allowing the user to keep working
  - else
    - Load configuration file
    - Inject
- Looking for Undocumented Functions
  - `ZwCreateThreadEx` defined in `Imports.h` of blackbone, returns a pointer handle to a thread
  - Can't find `LdrLoadDll` definition anywhere, just references to it and variables that use that identifier
- Ran out of time at this point, have discovered lots of interesting stuff

7 References

Baines, J. (Dec. 13, 2021). *Driver-Based Attacks: Past and Present — Rapid7 Blog*. Rapid7. URL: https://www.rapid7.com/blog/post/2021/12/13/driver-based-attacks-past-and-present/ (visited on Nov. 21, 2022).

Blaukovitsch, R. (Jan. 31, 2022). *Cheating in Video Games: The A to Z*. Irdeto Insights. URL: https://blog.irdeto.com/video-gaming/cheating-in-games-everything-you-always-wanted-to-know-about-it/ (visited on Nov. 25, 2022).

Godsey, B. (Mar. 6, 2017). *Video Game Security: The Future Belongs to Machines*. Infosecurity Magazine. URL: https://www.infosecurity-magazine.com/opinions/video-game-security-future-machines/ (visited on Nov. 26, 2022).

Greshishchev, M. and Zuydervelt, L. (2020). *What Does It Take to Catch a Cheater in 2020? (Presented by Denuvo by Irdeto)*. URL: https://www.gdcvault.com/play/1026757/What-Does-It-Take-to (visited on Nov. 20, 2022).

Help Net Security (Jan. 13, 2022). *Delivering Vulnerable Signed Kernel Drivers Remains Popular among Attackers*. Help Net Security. URL: https://www.helpnetsecurity.com/2022/01/13/vulnerable-signed-kernel-drivers/ (visited on Nov. 21, 2022).

Lehtonen, S. J. (Mar. 7, 2020). "Comparative Study of Anti-cheat Methods in Video Games". Helsinki, Finland: University of Helsinki, Faculty of Science. 128 pp. URL: http://urn.fi/URN:NBN:fi:hulib-202003241639.

Menegus, B. (Jan. 30, 2022). "What's the Deal With Anti-Cheat Software in Online Games?" In: *Wired*. ISSN: 1059-1028. URL: https://www.wired.com/story/kernel-anti-cheat-online-gaming-vulnerabilities/ (visited on Nov. 20, 2022).

Mikkelsen, K. K. (2017). "Information Security as a Countermeasure against Cheating in Video Games". MA thesis. NTNU. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2448953 (visited on Nov. 21, 2022).

Parizi, R. M. et al. (2019). "Security in Online Games: Current Implementations and Challenges". In: *Handbook of Big Data and IoT Security*. Ed. by A. Dehghantanha and K.-K. R. Choo. Cham: Springer International Publishing, pp. 367–384. ISBN: 978-3-030-10543-3. DOI: 10.1007/978-3-030-10543-3_16. URL: https://doi.org/10.1007/978-3-030-10543-3_16 (visited on Nov. 21, 2022).

Soliven, R. and Kimura, H. (Aug. 24, 2022). *Ransomware Actor Abuses Genshin Impact Anti-Cheat Driver to Kill Antivirus*. Trend Micro. URL:

https://www.trendmicro.com/en_us/research/22/h/ransomware-actor-abuses-genshin-impact-anti-cheat-driver-to-kill-antivirus.html (visited on Oct. 1, 2022).

Toulas, B. (Aug. 25, 2022). *Hackers Abuse Genshin Impact Anti-Cheat System to Disable Antivirus*. BleepingComputer. URL: https://www.bleepingcomputer.com/news/security/hackers-abuse-genshin-impact-anti-cheat-system-to-disable-antivirus/ (visited on Nov. 21, 2022).