Malicious Game Client Detection Using Feature Extraction and Machine Learning

# INTRO

The popularity of video games is enormous with games like Grand Theft Auto 5 (GTA 5) hitting 165 million sales by 2022 (Clement, 2022). Another popular game, Minecraft, has sold over 238 million copies through July 2021 (Clement, 2021). User-made extensions for these games are also immensely popular. One popular repository for these types of extensions, GTA5-mods, hosts tens of thousands of modifications for GTA 5 with hundreds of millions of downloads (GTA5-mods, 2022). Similarly, Minecraft has attracted thousands of unpaid developers to make extensions and clients for the game (Curseforge, 2022).

Because of the large player base and evident popularity of user-made extensions, it has already attracted malware authors using video games as a delivery mechanism for malicious software. Cisco Talos (talosintelligence.com) recently discovered a large campaign to push out malware using modding tools for two games, Point Blank and Crossfire. While these tools promised to provide tweaks to the game, they also installed a backdoor trojan known as XtremeRAT (Unterbrink, 2021). A survey from July 2020 to June 2021 found that Minecraft was the most infected game with game-related malware (kaspersky, 2021). This data from the Kaspersky Security Network found over 184,000 infected users across 3 million detections for Minecraft-related malware.

Many studies have looked at key indicators for malware on infected machines (An et al., ; Canzanese et al., b; Liang et al., 2016). This study aims to add to malware research by finding key indicators for Minecraft clients, or launchers. Game clients present unique challenges from other forms of malware. They may, for example, only affect game performance (e.g., randomly teleporting the character) rather than affecting data confidentiality, integrity, or availability. Additionally, clients often advertise that they may trigger anti-virus notifications, especially clients that purport to give players hacks. This research will be beneficial for companies and institutions trying to catch video game related malware in user uploaded content. For my research, I intend to find what key indicators are indicative of malware in Minecraft clients.

**Research Question:  What are the prominent indicators of malware in game clients?**

To answer the research question, I will look at current and past research to identify valid indicators of malware infection. I will then apply these indicators to video game mods to develop a set of prominent indicators to help identify infected video game mods. Finally, I will utilize machine learning algorithms to develop a model for detecting malware in the mods. This method was used by Smutz and Stavrou to detect malware in PDFs (Smutz & Stavrou, ).

# LITERATURE REVIEW

## Overview of Clients

### Defining Clients

A client is a method for accessing a video game and is usually required for modern games. They are important because they allow for licensing and user-based authentication. Minecraft ships with its own launcher developed by the publisher Mojang. Often users want to extend the functionality of these launchers for means of modifying game mechanics, making routine tasks easier, or providing player hacks to give advantages in multiplayer servers.

Developers that alter the original game code have not always been in good favor with the game publisher. In 2017, TakeTwo, the publisher behind Rockstar’s GTA V, delivered a cease and desist to the largest tool for modifying in-game API behavior, OpenIV (Livingston, 2017). Although TakeTwo eventually withdrew their legal complaint, it sent shockwaves through the community. Not as lucky, a group of developers who called themselves Apeiron had been developing a total conversion remake of the 2003 Knights of the Old Republic (KOTOR) for two years (Grayson, 2018). In 2018, Lucas Film ordered a cease and desist for the remake. Despite making no profits from the conversion and with no plans to generate revenue, the team had no choice but to accept the order and cease production.

Although game studios have a right to protect their content, developers of game enhancing clients are fearful of repercussions often seek anonymity. Most clients are made by unknown authors that only go by a username unlike the OpenIV tools and the KOTOR remake. Anonymity is great for malware attackers because it limits the ability of researchers to identify where a malicious software may have come from and how it spread. When obtaining relevant video game malware, it will be necessary to try and document where it came from, but this can be made difficult due to the anonymity of those who publish altered game clients.

### Malware in clients

Understanding why clients are created is important in understanding what attack vectors malware authors may use when writing malicious game clients. These developers are usually not employed by game studios and work on their software without pay (Poor, 2014). The disconnect between developers of game extensions and the publishing developers leaves room open for malicious app development. Any verification of game clients will have to be done at the repository level for a website hosting them.

## Cheat Detection

Video game cheats that give an unfair advantage to players by modifying the game code are essentially unauthorized modifications and share similarities with malware. Many companies employ cheat detection software that looks for cheats and functions like anti-virus applications. Many methods used in cheat detection are applicable to this study for finding prominent indicators of malware in clients. Authors of cheat detection software are looking for process injection, unusual network communication from the game, and abnormal software behavior.

A literature review of cheat detection found five major types of detection methods: behavioral, verification, result, reputation, and hardware (Björkskog, 2019). Behavioral detection compares known cheaters with current players and looks for similarities in play style and behavior. Verification detection looks at the game state for illegal actions such as having a larger viewable area than allowed. Result detection assumes that players should be predictable and when dramatic changes in ability are noted over a short time, it is flagged. A newer method of detection is reputation based and has trusted clients act as referees for other clients. Fake data may be sent to clients, such as an invisible player. Cheat software may be able to detect this invisible player and react to it while trustworthy players will not. Lastly, hardware detection is perhaps the most common method and relies on monitoring CPU state and running processes. It is the most intrusive detection method.

Out of the above method detection methods, hardware detection is the most useful when looking at malware in game clients. A study looking at methods for detecting video game injector exchange notes that the “cornerstone” to making cheats Is using a stealthy memory injector to insert the code into the running game process (Karkallis et al., ). Given the similarities between game cheats and malicious clients, looking for process injection will be of paramount importance when finding indicators for malware in Minecraft clients.

Cheating in video games is a constant game of hide and seek between cheaters and developers. As the developers get better at detecting running processes and injections, writers of cheat software get better at hiding them. Kanervisto et al. (Hautamaki, Anssi Kanervisto and Tomi Kinnunen and Ville, 2022) looked at methods for detecting cheats using machine learning instead of relying solely on memory analysis. This approach offers some insights into research about prominent indicators of malware in clients. Instead of only relying on what can be seen while the client is running, developing a wholistic machine learning model for malware activity will likely be more effective.

Cheat detection has also caused security problems, for example, Capcom rolled out measures to detect cheating in Street Fighter V by implementing a kernel level driver for monitoring running processes and injected code (Williams, 2016). This resulted in a severe rootkit due to poor programming and would allow attackers to inject any arbitrary code in the game’s now kernel-level software. Although no known attacks were used during the period this driver was running, it shows how games can be leveraged to escalate access.

## Anomaly Detection

Building a machine learning model for detecting video game malware will require looking for anomalies in infected systems. A 2018 study analyzing malware for the Echo device by Amazon found differences in the Linux kernel system calls using a one-class SVM-based algorithm (An et al., ). There were many unique system calls in the infected malware that did not show up in the clean systems. Additionally, they generated a pattern of calls based on the frequency and call number. This allowed their algorithm to quickly and accurately predict which samples contained malware.

An older study looking at Windows XP was able to show that malware infected systems have a higher number of NTQuery system calls than uninfected systems (Canzanese et al., a). Based on the IoT system call analysis and this Windows system call analysis, this is an excellent indicator to look at when researching malware.

A similar research study looking at IoT devices used a machine learning algorithm for detecting differences in network activity to find Mirai malware researchers aggregated packets captured across the network (Nguyen et al., ). They assigned each packet a symbol based on the characteristics and were able to detect malware with 95.6% accuracy and no false positives. This method of using machine learning to look at network packets will a useful indicator to investigate clients that may contain malware.

## Non-signature Malware Detection

Signature based detection for malware has limited effectiveness when encountering novel malware. If the malware author changes the code enough that the signature changes, it will be more difficult to detect by routine scanners such as VirusTotal. One method for doing this involves extracting strings and analyzing them. A study of Windows malware extracted DLL calls from binary files as well as the function calls and number of calls that these DLLs appeared to make (Schultz et al., ). They found a significant difference in the number and type of DLL calls in infected systems compared to clean systems. Looking at DLL calls in client binaries could be an indicator of malware.

## Live Detection Summary

### Network traffic

Analyzing packets individually is a complex task in modern malware analyses because of the extreme volume of network traffic on most modern devices. Based on research for a tool called MalAlert, some aggregate features can be extracted to detect the presence of malware. In this 2019 study, they showed that the quantity of bytes sent was an important feature in identifying malware compared to a baseline. (Piskozub et al., 2019) Additionally, they showed that port features were an important and aggregated two different sets of ports, those between 1 and 49152 and 49153 through 65535. These were compared against the 10 most common ports for abnormalities. Malware is more likely to be present when there is non-HTTP traffic being set over ports 80 and 443 and HTTP traffic is being sent over non-HTTP ports. (Zhao et al., 2015)

Additionally, it has been shown that DNS requests and geographical regions can establish a trust measure that aids in detecting malware. Requests that are reaching untrusted domains more infrequently may indicate malware. Zou et al. showed that malware was more likely to be transmitted over DNS requests to untrusted sources. (Zou et al., 2015) Although their model is more complicated than will be used in this thesis, it provides evidence that looking at trusted and untrusted regions for DNS requests is an important aspect of identifying potential malware.

### System Calls

Malware has grown increasingly complex and many forms of malware have detection capabilities that detect analysis engines and can terminate itself. Although on the surface this can appear detrimental for analysis, this can be useful as application that terminate a significant number of processes may indicate malware presence. (Oyama, 2018) This research article shows that the number of API calls involved in process termination can be an indicator of malware presence. Additionally, the number of processes started is also a potential indicator of malware. This can be a tool that malware uses to hide itself by starting child processes, especially ones that normally look benign, such as explorer.exe.

Furthermore, researched have shown that the context behind malware behavior in process generation is important. (Wang et al., 2020) They explained that the number of processes started as well as the type can indicate malware presence. Looking at the process tree is an important tool to identify potential malware. For example, a word editing program called texteditor.exe would appear less suspicious if it started a subprocess for an email application such as Outlook. It would appear more suspicious if the texteditor.exe application started a command terminal of Powershell, suggesting that it might be executing code.

A 2018 study on ransomware behavior showed that a large portion, 6 of 9, of the malicious API calls were directed at filesystem operations (Hampton et al., 2018). These operations included directory scans, requesting file types and sizes, and read and write operations. The malware significantly exceeded call rates of normal system operations. Because of this potential for malware to make an exaggerated number of file system API calls, the frequency will be an important metric for malware detection.

In addition to frequency of file changes, the locations of changes are important because changes to some areas of the file system may be more likely to indicate the presence of malware. For instance, a study looking at the file operation location and noted some areas, such as start up directory modifications, were more likely to indicate malware (Aslan & Erdal, 2022). Additionally, they showed that the frequency of registry changes and the location of registry changes, particularly those that relate to DLL (linked file) locations was indicative of malware. Given this information, it will be important to look at both the frequency and location of both file system and registry API calls.

### Persistence

The goal of malware is usually beyond a one-time code execution. Attackers usually want to be able to maintain some sort of access or control over a system. Because of this, malware authors try to create persistent code. A previously mentioned 2022 study looked at changes to the Windows startup directories and registry location for startup applications (Aslan & Erdal, 2022). They found that a particular piece of software was more likely to be classified as malware if it was making modifications to these areas.

Additionally, it is useful to look at executables accessed by a suspicious software to determine if it is trying to remain persistent. A recent article on persistence methods highlighted that access to the binaries Reg.exe, Nslookup.exe, Regasm.exe, Runas.exe, Schtasks.exe, and Sc.exe indicated a strongly likelihood of malware trying to become persistent on a Windows machine (Barr-Smith et al., 2021). For this research, access to startup file locations and calls to these binaries will be monitored.

### Memory

Looking at how an application behaves in memory is crucial to understanding if it contains potential malware. One of the most proven methods of looking at in-memory behavior for Windows machines is looking at DLL calls. Normal system usage generally has uniform DLL access, which means that no one DLL is called significantly more than another (Matsuda et al., 2020). This same article found that malware typically shows a significant amount of DLL calls to one particularly linked library, such as user32.dll. While the specific DLL called is not always the most reliable indicator of malware, the frequency can be a feature for malware. This thesis will look at DLL frequency to aid in malware detection.

### Anomaly Detection

High level overviews of system resources have been shown to be an indicator malware. A recent study looking that the Colonial Pipeline ransomware hack was able to show a method for detecting malware based on irregularities in system resource utilization (Kim & Park, 2022). Compared to baseline CPU utilization, malware showed more CPU usage over the same interval of time. Additionally, they showed that the patterns of RAM usage were different, although not necessarily higher due to both the control sample and the malware sample hitting the maximum RAM allocation at times.

A similar conclusion was reached when investigating cryptojacking, stealing resources for crypto mining, since it involved irregular CPU and RAM usage (Naseem et al., 2021). Particularly, researched noted that malware could throttle the CPU to avoid detection for high CPU usage. While this may work if only examining for high CPU usage, they suggest an approach that looks at CPU usage patterns compared to baseline. For this research, it will be important to look at CPU patterns, including high or low frequencies and usage, rather than just focusing on one aspect.

All the features that will be used for live analysis are summarized in the figure below.

A picture containing text, screenshot, font, number

Description automatically generated

The table below gives an overview of the types of data that will be collected. Numerical quantity describes a total sum of the variable. Country or region describes the geo-location of requests. For startup modification the data is taken in as a list of string values. CPU and RAM usage are lists of numbers where each number describes the percent utilization of the resource for each second of run time.

|  |  |
| --- | --- |
| Network Traffic | |
| DNS requests | Numerical quantity |
| Network connections | Numerical quantity |
| HTTP requests | Numerical quantity |
| DNS requests locations | Country/Region |
| Network connections locations | Country/Region |
| HTTP requests locations | Country/Region |
| System Calls | |
| Processes instantiated | Numerical quantity |
| Modified Files | Numerical quantity |
| Registry Reads | Numerical quantity |
| Registry Writes | Numerical quantity |
| Registry Deletes | Numerical quantity |
| Memory | |
| DLL Calls | Numerical quantity |
| Persistence | |
| Startup behavior modifications | List of values |
| Suspicious binary file accesses | Numerical quantity |
| Anomaly Detection | |
| CPU usage | Numerical list of CPU usage |
| RAM usage | Numerical list of RAM usage |

## Static Detection Summary

|  |  |
| --- | --- |
| Strings | |
| IP Addresses/Domains | Numerical quantity |
| File Paths | List of file paths |
| DLL Calls | List of calls |
| Exeinfo PE | |
| Packer signature | String value |
| Modified Files | Numerical quantity |
| Registry Reads | Numerical quantity |
| Registry Writes | Numerical quantity |
| Registry Deletes | Numerical quantity |
| Memory | |
| DLL Calls | Numerical quantity |
| Persistence | |
| Startup behavior modifications | List of values |
| Suspicious binary file accesses | Numerical quantity |
| Anomaly Detection | |
| CPU usage | Numerical list of CPU usage |
| RAM usage | Numerical list of RAM usage |

# METHODS

There are three major parts to this study shown in Figure 1. I will obtain a set of video games clients that are malicious as well as a set that are non-malicious. Using these two different types of clients I will extract a set of features from them using both static and live analysis methods. From this data that I will collect I will use machine learning classification techniques (e.g., KNN, SVM, and ANN) to discover which features are most prominent and create a model to recognize malicious and non-malicious clients.

## STEP 1: Obtain Malware Samples

This study will use the java version of the game Minecraft to look at malicious clients given the available quantity of clients for this game and that it is one of the largest attack vectors for video game malware. I will need to obtain malware samples of infected clients for the game Minecraft, usually in the format of jar (java archive) files. I will try to find as many known infected Minecraft clients for Minecraft as well as uninfected counter parts to those clients.

Malware samples were obtained using VirusTotal advanced search and through manual search methods using Google. On VirusTotal advanced search, I filtered relevant samples by first using the “type” search modifier, which allows a specific filetype to be specified. For these searches, “jar” files, or java extensions, are how Minecraft clients are typically packaged. All files were also filtered with the “positives” search modifier, with searches conducted looking for at least one or five positives. The “metadata” search modifier looks for any file metadata that contains a particular word or phrase. The “content” search modifier looks for any word or phrase matches anywhere in the sample information page.

Minecraft clients may also purport to be able to automatically install mods for the game, so results matching patterns for typical mod platforms, such as forge, bukkit, spigot, and fabric are included.

The table below shows the queries made and number of samples collected for each query.

|  |  |
| --- | --- |
| **Search Query** | **Number of samples collected** |
| type:jar metadata:"minecraft" positives:5+ | 44 |
| type:jar metadata:"minecraft" positives:1+ | 11 |
| type:jar content:"minecraft" positives:5+ | 8 |
| type:jar content:"forge" positives:5+ | 37 |
| type:jar name:"forge" positives:5+ | 5 |
| type:jar name:"bukkit" positives:5+ | 3 |
| type:jar name:"spigot" positives:5+ | 3 |
| type:jar name:"fabric" positives:5+ | 2 |
| type:jar name:"optifine" positives:5+ | 1 |
| type:jar content:"fabric" positives:5+ | 7 |
| type:jar content:"bukkit" positives:5+ | 13 |
| type:jar content:"spigot" positives:5+ | 10 |
| type:jar content:"paper" content:"minecraft" positives:5+ | 3 |
| Similar to queries | 1 |
| **Total** | **148** |

Relevant samples were collected based on the following criteria. Files were first categorized based on their function by searching for the original source or a content creator show casing the software. All obtained software in the sample set can be classified into two major categories, clients and mods. Clients can further be broken down into normal clients and cheat clients, the distinction being cheat clients are meant to provide in-game hacks. Mods are not being used for analysis in this research. Each used software was manually researched through a combination of Google, Youtube, SpigotMC, CurseForge, and other Minecraft software repositories.

In addition to VirusTotal, client samples were found manually using common Minecraft software repositories, such as 9minecraft and cheatermad. Additionally, forums and discussion posts were used as reference points to locate clients individually. For these types of clients they were uploaded to VirusTotal to get information on the number of submissions and submission dates.

Some potential client samples could not be identified and those samples were not used in this analysis to ensure that the data collected does not go out of scope.

|  |  |
| --- | --- |
| Software Category | Count |
| Clients | 96 |
| Mods | 31 |
| Unidentified | 85 |

The name, content, and metadata search modifiers help narrow down specific types of of Minecraft software. These include Forge, Bukkit, Spigot, Fabric, and Paper, which each have their own APIs for interacting with the game.

Samples were obtained from submissions starting in July 2019 and ending in February 2023. Most samples were uploaded to VirusTotal between 2022 and 2023

|  |  |
| --- | --- |
| Year | Number of Samples |
| 2019 | 1 |
| 2020 | 3 |
| 2021 | 12 |
| 2022 | 70 |
| 2023 | 62 |

The samples averaged a file size of 15.04 MB and a median file size of 5.06 MB, with the largest sample at 128.11 MB and the smallest at just 0.03 MB. The average amount of user submissions was 5.42 submissions per file and the median was 2, with the highest being 160 and the lowest 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Mean | Median | Max | Min |
| File Size | 15.044 MB | 5.059 MB | 128.11 MB | 0.030 MB |
| Total Submissions (Number of unique user uploads) | 5.507 | 2 | 160 | 1 |
| Detections (VirusTotal unique vendor flags) | 11.182 | 9 | 36 | 2 |

In an effort to maintain closely matched samples, the non-malware samples were found within a similar timeframe to the malware samples, ranging from 2019 to 2023, with many more samples being included from 2022 and 2023 than the other years. The purpose of closely matching samples is to ensure that both the clean sample sets and malware sample sets look similar. Based on the useable samples, the non-malware clients are aggregated to generalize the activity of non-malicious clients.

## STEP 2: Malware Sandbox

A sandbox environment allows testing of live malware samples and is the key element that will provide much of the data needed for the analysis in this thesis. According to StatCounter, Windows 10 is the most popular operating system for Windows platforms, covering over 71% of the Windows marketshare (StatCounter, 2023). Because of this, Windows 10 was chosen for the malware sandbox operating system. The sandbox environment also needed to run interactively, meaning the user can interact with software samples. The environment also needed Java 8 available to run executable jar files. The service Any.run met all the requirements for data collection as outlined in table X as well as the ability to run on a Windows 10 64-bit operating system interactively.

The general process of running samples involved using a template configuration profile for the analysis VM so that each run was kept consistent. The behavior of launchers can vary and require manual interaction. The configuration for Any.run was set to start the object from the Desktop directory with a maximum run time of 660 seconds. Network use was enabled because many launchers require downloading additional binaries. The operating system was set to 64-bit Windows 10 with auto confirm UAC. The chosen pre-installed software set was “Complete” because it was the only software set that included Java 8, which is needed for a majority of the samples.

The following uniform protocol was used. If the launcher was an installer, an attempt was made to install the software and launch it if it did not automatically do so. For launchers that did not require installation and those that did, an attempt was made to sign into Minecraft using an offline account and launch the game. Whether or not a software launched visually, once I was not able to make more progress in starting the game or installing the software, one minute was given for additional background activity to be analyzed. Once there appeared to be no more activity and at least one minute had passed, the sandbox was terminated.

For installers or launchers in other languages, the best attempt was made to follow the prompts in the application. If the application closed unexpectedly, the processes was restarted until I successfully reached an end to input that I can give the client or client installer.

The PID for the client process was recorded for each sandbox analysis in order to ensure in the later data processing phase that the process being analyzed is the sample and not another process. For clients that spawned more processes, a list of PIDs was kept instead of a single PID.

It is difficult to discern between a successful client launch and an unsuccessful startup. It is possible that malware may be hiding in software that appears non-functional to the user visible. To account for this, analyses where the software did not appear visibly were still used in the data pool.

Due to file size limitations for the sandbox environment, files over 100 MB in size were uploaded to DropBox where I created an automatic download link for the file. Any.run allows for using a download link to analyze a file. These download links were used to analyze the file. Public DropBox links are created by first creating a link for the uploaded file in the right click context menu. Secondly, the URL requires a change from “dl=0” to “dl=1” to make it an automatic download.

Clients that were retrieved as directories were uploaded as a zip file so that the binaries could have all the necessary application extension files needed for using the client. These were unzipped and then run manually following the processes described in this section.

At the end of each run, after the sandbox had been stopped, the data needed for the run was downloaded in json file format for later processing. These were saved with the filename being the hash of the program that was being analyzed so they could easily be referenced later.

## STEP 3: Feature Extraction

For this proposal, I give a few examples of feature extraction. I will augment this list as I perform a more thorough literature review.

### Static Analysis

I will use strings program in Linux to look for calls to DLL files (application extensions), especially user32.dll. I will compare this against the quantity of DLL calls in clean mods to see if malicious mods make more references to DLLs and which DLLs they are calling. Additionally, I can compare the strings output of a large known set of malware against both the mod malware and the clean mods to see if there is a broader correlation that indicates malware in mods.

### Live Analysis

I will capture the network activity using Wireshark for both the clean mods and malicious mods. My goal is to capture the quantity and type of network packets that are going in and out of the system while the game is running with the different mods. I will put these captures through multiple network traffic analyzers (such as Snort) to extract features.

I will capture the overall quantity of system calls for both the clean mods and malicious mods. I want to capture the amount of calls to see if there is a difference in how many calls malicious mods are making compared to clean mods.

I will take XML reports of API calls from the environments running clean and malicious mods and look for E9 bytes that have been added to the beginning of functions. This is a JUMP instruction in assembly and indicates a rootkit has taken over that API call. I will compare the quantity of hooked APIs in malicious mods to those of clean mods.

## STEP 3: Data Analysis and Classification

The data that I collect with the extracted features mentioned in the previous section will vary greatly and not be useful without proper analysis. To make sense of this data and look at it wholistically I will use be using several different classification methods, for example: random forest, SVM, neural network, and k-nearest neighbors. This classification algorithms will help identify prominent indicators, or features, from the mods, thus answering the research question to find what prominent indicators of malware are in video game mods.

Malicious mods and non-malicious mods will have a different set of prominent features and will allow me to predict how likely a mod contains malware using a score from each of the classification methods. Using multiple classification methods is important because it will indicate if the results are replicable in many data sets or only a few data sets. Knowing which classification method works the best for finding indicators of malware will help me to make a model for predicting if a mod contains malware. I will report the accuracy of the classification models. I will also discuss the robustness of the models to adversaries.

# CITATIONS

References

An, N., Duff, A., Noorani, M., Weber, S., & Mancoridis, S.Malware anomaly detection on virtual assistants. Paper presented at the *2018 13th International Conference on Malicious and Unwanted Software (MALWARE),* 124-131.

Aslan, Ö, & Erdal, A. (2022). Malware detection method based on file and registry operations using machine learning. *Sakarya University Journal of Computer and Information Sciences, 5*(2), 134-146.

Barr-Smith, F., Ugarte-Pedrero, X., Graziano, M., Spolaor, R., & Martinovic, I. (2021). (2021). Survivalism: Systematic analysis of windows malware living-off-the-land. Paper presented at the *2021 IEEE Symposium on Security and Privacy (SP),* 1557-1574.

Björkskog, G. (2019). Detecting cheaters who utilise third-party software to gain an advantage in multiplayer video games. <https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1322397&dswid=4210>

Canzanese, R., Kam, M., & Mancoridis, S. (a). (a). Inoculation against malware infection using kernel-level software sensors. Paper presented at the *Proceedings of the 8th ACM International Conference on Autonomic Computing,* 101-110.

Canzanese, R., Kam, M., & Mancoridis, S. (b). (b). Toward an automatic, online behavioral malware classification system. Paper presented at the *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems,* 111-120. <https://ieeexplore.ieee.org/abstract/document/6676498>

Clement, J. (2021). *Cumulative number of copies of minecraft sold worldwide as of april 2021*. Statista.

Clement, J. (2022). *Lifetime unit sales generated by grand theft auto V worldwide as of may 2022*. Statista.

Curseforge. (2022). Curesforge minecraft mods. <https://www.curseforge.com/minecraft/mc-mods>

Grayson, N. (2018). Star wars: KOTOR fan remake shutting down after cease and desist from lucasfilm. <https://kotaku.com/star-wars-kotor-fan-remake-shutting-down-after-cease-a-1829720602>

GTA5-mods. (2022). *5Mods*. GTA5-mods.

Hampton, N., Baig, Z., & Zeadally, S. (2018). Ransomware behavioural analysis on windows platforms. *Journal of Information Security and Applications, 40*, 44-51.

Hautamaki, Anssi Kanervisto and Tomi Kinnunen and Ville. (2022). GAN-aimbots: Using machine learning for cheating in first person shooters. *IEEE Transactions on Games,* <https://doi.org/https://doi.org/10.48550/arXiv.2205.07060> Focus to learn more

Karkallis, P., Blasco, J., Suarez-Tangil, G., & Pastrana, S.Detecting video-game injectors exchanged in game cheating communities. Paper presented at the 305-324.

kaspersky. (2021). *Minecraft most malware-infected game on the market with 228k users affected*. kaspersky.

Kim, J., & Park, K. (2022). Ransomware classification framework using the behavioral performance visualization of execution objects. *Computers, Materials & Continua, 72*(2), 3401-3424.

Liang, G., Pang, J., & Dai, C. (2016). A behavior-based malware variant classification technique. *International Journal of Information and Education Technology, 6*(4), 291.

Livingston, C. (2017). GTA modding tool OpenIV shuts down due to cease and desist from take-two (updated). <https://www.pcgamer.com/gta-modding-tool-openiv-shuts-down-claiming-cease-and-desist-from-take-two/#:~:text=News-,GTA%20modding%20tool%20OpenIV%20shuts%20down%20due%20to%20cease,from%20Take%2DTwo%20(Updated)&text=The%20tool%20has%20been%20essential,due%20to%20a%20legal%20notice.>

Matsuda, W., Fujimoto, M., & Mitsunaga, T. (2020). Detection of malicious tools by monitoring DLL using deep learning. *Journal of Information Processing, 28*, 1052-1064.

Naseem, F. N., Aris, A., Babun, L., Tekiner, E., & Uluagac, A. S. (2021). (2021). MINOS: A lightweight real-time cryptojacking detection system. Paper presented at the *Ndss,*

Nguyen, T. D., Marchal, S., Miettinen, M., Fereidooni, H., Asokan, N., & Sadeghi, A.DÏoT: A federated self-learning anomaly detection system for IoT. Paper presented at the *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS),* 756-767.

Oyama, Y. (2018). Trends of anti-analysis operations of malwares observed in API call logs. *Journal of Computer Virology and Hacking Techniques, 14*(1), 69-85.

Piskozub, M., Spolaor, R., & Martinovic, I. (2019). MalAlert. *Acm Sigmetrics Performance Evaluation Review,* <https://doi.org/10.1145/3308897.3308961>

Poor, N. (2014). Computer game modders’ motivations and sense of community: A mixed-methods approach. *New Media & Society, 16*(8), 1249-1267. <https://doi.org/10.1177/1461444813504266>

Schultz, M. G., Eskin, E., Zadok, F., & Stolfo, S. J.Data mining methods for detection of new malicious executables. Paper presented at the *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001,* 38-49.

Smutz, C., & Stavrou, A.Malicious PDF detection using metadata and structural features. Paper presented at the *Proceedings of the 28th Annual Computer Security Applications Conference,* 239-248.

StatCounter. (2023). *Desktop windows version market share worldwide .* <https://gs.statcounter.com/windows-version-market-share/desktop/worldwide/#monthly-202307-202307-bar>

Unterbrink, N. L. a. H. (2021). *Cheating the cheater: How adversaries are using backdoored video game cheat engines and modding tools*. Cisco.

Wang, Q., Hassan, W. U., Li, D., Jee, K., Yu, X., Zou, K., Rhee, J., Chen, Z., Cheng, W., & Gunter, C. A. (2020). (2020). You are what you do: Hunting stealthy malware via data provenance analysis. Paper presented at the *Ndss,*

Williams, C. (2016). *Double KO! capcom's street fighter V installs hidden rootkit on PCs*. The Register.

Zhao, G., Xu, K., Xu, L., & Wu, B. (2015). Detecting APT malware infections based on malicious DNS and traffic analysis. *IEEE Access, 3*, 1132-1142.

Zou, F., Zhang, S., Rao, W., & Yi, P. (2015). Detecting malware based on DNS graph mining. *International Journal of Distributed Sensor Networks, 11*(10), 102687.

stylefix