Technical cheating prevention in location-based games

Conference Paper · June 2021

DOI: 10.1145/3472410.3472449

4 authors:

Samuli Laato
Tampere University
78 PUBLICATIONS **1,309** CITATIONS
SEE PROFILE

Sampsa Rauti
University of Turku
67 PUBLICATIONS **390** CITATIONS
SEE PROFILE

Lauri Koivunen
University of Turku
10 PUBLICATIONS **32** CITATIONS
SEE PROFILE

Jouni Smed
University of Turku
140 PUBLICATIONS **1,376** CITATIONS
SEE PROFILE

Technical cheating prevention in location-based games

SAMULI LAATO, University of Turku, Finland
LAURI KOIVUNEN, University of Turku, Finland
SAMPSA RAUTI, University of Turku, Finland
JOUNI SMED, University of Turku, Finland

Online multiplayer game developers have plenty of motivation to prevent technical cheating in their games. In the case of location-based games (LBGs), in addition to the usual anti-cheat measures, developers need to be able to verify players' location sensor data. This is a challenge, as mobile devices' sensor data is easy to manipulate, and its verification goes beyond the permissions granted to individual applications such as LBGs. In this work we focus on the most popular LBG, Pokémon GO, and investigate via a technical analysis and gray literature search what countermeasures the game has implemented to prevent technical cheating. We obtain a large diverse set of data which we synthesize using the Gioia method into three main clusters: (1) preemptive measures; (2) ad hoc analysis; and (3) player communication. Our work demonstrates that a wide range of technical tools, both self-developed and 3rd party provided, are used to enhance the multi-layered cheating prevention of LBGs. Our findings can be considered an exemplar look into the industry leaders' solutions to technical cheating issues that LBGs face.

CCS Concepts: • **Security and privacy** → *Software reverse engineering*; **Software security engineering**.

Additional Key Words and Phrases: location-based games, video games, technical cheating, location spoofing, cheating prevention

ACM Reference Format:
Samuli Laato, Lauri Koivunen, Sampsa Rauti, and Jouni Smed. 2021. Technical cheating prevention in location-based games. In *International Conference on Computer Systems and Technologies '21 (CompSysTech '21), June 18–19, 2021, Ruse, Bulgaria.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3472410.3472449

1 INTRODUCTION

"We don't talk much about our anticheat efforts, partly because we don't want to provide cheaters with information that can help them understand our detection mechanisms better. But, rest assured our reluctance to share is not due to a lack of effort or intent from our side." -Niantic[1]

There are various forms of cheating in video games. In her seminal book *Cheating: Gaining Advantage in Videogames*, Consalvo describes its various forms, intertwined with the gaming culture and ethical conduct [5]. Cheating in general could compromise things such as reading from an external source (such as the Nintendo power magazine) about games' secrets [5], using *cheat codes* that the developers have voluntarily entered into the game, and purposefully deceiving other players in a competitive setting. In this study, we depart from these three forms of cheating, and focus solely on technical cheating. While cheating can exist in single-player games [18], we focus on it in online multiplayer games. The reason for this is that the consequences of technical cheating when other players are involved are much more dire and far-reaching.

[1] https://nianticlabs.com/blog/cheatingupdate-022321

This is a pre-print version of the manuscript "Technical cheating prevention in location-based games" published in the proceedings of the CompSysTech conference in 2021 by ACM.

Examples of technical cheating include packet tampering, use of modified user interfaces, look-ahead cheating and cracking, and while there exists counter-measures which developers can employ [21, pp. 290–297], technical cheating remains a major problem in today's online gaming landscape [10]. As new types of games are developed, new forms of cheating also emerge. This is particularly true for games relying on sensor data. For example, in fitness games that use accelerometers to measure movement, players may calibrate the sensor to provide false data, or use it in a way that it is not supposed to be used. This way players are able to make the game easier for them. As location-based games (LBGs) are heavily reliant on location data, ensuring its validity is paramount to make sure LBGs are played as they were meant to be played. This can be seen as a requirement especially for competitive LBGs.

In addition to the validity of location data, LBGs contain the same set of technical cheating issues as other multiplayer online games. The various forms of possible technical exploits and cheating opportunities that exist in LBGs require developers to constantly pay attention to anticheat measures to make the playing experience fun for others and also to avoid monetary loss. In this work, we investigate how various forms of technical cheating have been taken into account in the most popular LBG, Pokémon GO. As the game is the most popular LBG and has been in operation for five years, findings related to it can be considered valuable for practitioners in the gaming industry, but also outside it, as location-based applications and other applications relying on valid sensor data are becoming increasingly popular.

2 BACKGROUND

2.1 Motivations to cheat

Common motivations for cheating are [21, p. 289]:

- *lack of skill or time*: the player cheats to overcome a part of the game which is too difficult or time consuming
- *money*: the player cheats to win prize money or to create virtual assets to trade with real-world money
- *boredom*: the player wants to skip uninteresting content
- *fun*: the player gets entertainment from using a cheat
- *causing havoc*: the player cheats to ruin the other players' game experience
- *experimentation and exploration*: the player wants to find out all the content and secret areas
- *extending the life span of the game*: the player wants to continue to play after the original content is finished
- *creativity*: the player uses cheats to create new experiences such as mods or total conversions
- *non-conformity*: the player seeks to do forbidden things or rebel against the authority
- *fame*: the player seeks recognition and prestige by cheating to win the game or from inventing an cheat

Among these, the primary motivation for technical cheating in the online multiplayer video gaming landscape can be considered to be monetary gain [21, p. 289]. As games increasingly sell digital goods, and these can also sometimes be traded between players, the ability to obtain those goods for free is worth of real money. Furthermore, there exists markets outside the immediate game context where players may sell and purchase, for example, high level characters or accounts from others. Having the ability to obtain high level characters or accounts may thus be a desirable goal and worth spending time to find technical exploits.

Players may also resort to cheating for social gain. According to Chen and Ong [2], one of the core rationalization processes of online cheating relates to social ties to the gaming communities. In LBGs players have been found to be motivated to grind onward and progress to gain social capital [19]. Furthermore, as players may be motivated to control virtual territory in LBGs if afforded by the game mechanics, reserving it for their team or away from opponent [12], social competition may be one more motivation for players to resort to cheating. This is further supported by findings which indicate that when faced with social consequences of cheating, players are significantly less likely to cheat [3]. In other words, when playing anonymously, players cheat more frequently [3].

Especially with regards to technical exploits, simple curiosity may be involved. Players may wish to know how a specific game is built and whether anti-cheating measures have been put in place. Reverse engineering systems may be a fun technical challenge, and a simple enjoyable activity for some. Surprisingly, players have been found to be accepting towards cheating in single-player concepts [18]. This could indicate that if for some reason the social dismay caused by cheating is removed, the positive drivers such as curiosity and the positive outcomes such as stress relief could make technical cheating acceptable in the eyes of players [18].

2.2 Forms of technical cheating in video games

In a networked multiplayer game, a cheater can target the clients, the servers, or the network connecting them [8]. On the client side, the attacker can go *over* the client (e.g. using macros or triggering keyboard events for control and reading pixel values from the user interface), *under* the client (e.g. hacking a driver to access the video memory or tampering with the network traffic), or *in* the client (e.g. altering the code in the client's memory). Game servers themselves are vulnerable to network attacks as well as physical attacks such as theft or vandalism. Third-party attacks on clients or servers include IP spoofing and denial-of-service (DoS) attacks.

In first-person shooter games, a usual way to cheat is to enhance the player's reactions with reflex augmentation (e.g. an aiming proxy monitoring the opponents' positions and improving the cheater's aim). Reversely, in packet interception the proxy prevents certain packets from reaching the cheating player (e.g. suppressing packets containing damage information). In a packet replay attack, the same packet is sent repeatedly, which can be used to boost the firing rate of a slow weapon (e.g. a bazooka). A common method for breaking the control protocol is to change bytes in a packet and observe the effects. However, these kind of attacks can be easily countered by using checksums.

In a peer-to-peer architecture, where all nodes uphold the game state and the players' time-stamped actions must be conveyed to all nodes, an attacker can use look-ahead cheating. Here, the attacker gains an unfair advantage by delaying their actions to see what the other players do before choosing their action. The attacker then forges the time-stamped packets so that they seem to be issued before they actually were. To prevent this we can use, for example, a lockstep protocol that requires each player to announce first a commitment to an action. Only when everyone has received the commitments, the players reveal their actions, which can be then checked against the original commitments [1]. Nowadays most games utilize a client-server architecture, making such forms of cheating obsolete. However, there are still some games such as Starcraft2 which use a peer-to-peer network for multiplayer.

Networking is not the only target for attacks. A cracked client software may allow the attacker to gain access to the replicated, hidden game data (e.g. the status of other players). On the surface, this kind of passive cheating does not tamper with the network traffic, but the attacker can base their decisions on more accurate knowledge than they are supposed to have. For example, typical exposed data in real-time strategy games are the variables controlling the visible area on the screen. This problem is common also in first-person shooters where, for instance, a compromised graphics rendering driver may allow the player to see through walls. Strictly speaking, these information exposure problems stem from the software and cannot be prevented with networking alone. Clearly, the sensitive data should be encoded and its location in the memory should be hard to detect. Nevertheless, it is always susceptible to ingenious hackers and, therefore, requires some additional countermeasures. For instance, hardware-based methods or behavior analysis can be used in ensuring that a client is not compromised [6, 16].

Network traffic and software are not the only vulnerable places but design defects can create loop-holes, which can be exploited. Also, the heterogeneity of networked environment can be a source for unexpected behavior. For instance, there may be features that become eminent only when the latency is extremely high or when the server is under a DoS attack. More often than not the player themself can be the weakest link against attacks like collusion or social engineering, which are hard prevent with technical solutions alone.

Finally, botting refers to the unauthorized access of the games' API to automatically play the game [17]. Botting can also be done via legitimate client software, and in these cases the client software functions are automated. Botting through legitimate client software is more difficult to detect. LBGs may also contain bugs or glitches which players may abuse to gain advantage in-game, however, if these can be utilized without technical intervention, we do not consider them technical cheating.

2.3 Goals of technical cheating prevention

As evident from the above description, there exists various ways for players to cheat in online multiplayer games. As there is a development race between multiplayer game networking programmers and technical exploiters, new ways to cheat also constantly emerge. Thus, it is important to review the goals of cheating prevention. On a general level, cheating prevention has three distinct goals [22, 24]:

- protect the sensitive information,
- provide a fair playing field, and
- uphold a sense of justice inside the game world.

Each of these goals can be viewed from a technical or social perspective: Sensitive information (e.g. players' accounts) can be gained, for instance, by cracking the passwords or by pretending to be an administrator and asking the players to give their passwords. A fair playing field can be compromised, for instance, by tampering with the network traffic or by colluding with other players. The sense of justice can be violated, for instance, by abusing inexperienced and ill-equipped players or by ganging up and controlling parts of the game world.

To investigate how these three goals [22, 24] are reached in LBGs, what the biggest issues with regards to technical cheating in the games are and what cheating prevention measures exist, we turn to our empirical case study.

3 MATERIALS AND METHODS

3.1 Case game

In this work, we provide an in-depth analysis of a single game rather than a surface-level analysis of several games. Among the various commercially available LBGs, Pokémon GO is by far the most popular in terms of both number of active players and revenue [13]. In addition, a large body of academic literature focuses on the game, which means that our findings can be compared with and understood in the context of previous work.

Pokémon GO is developed by Niantic, but relies on the intellectual property owned by The Pokémon Company, Game Freak and Nintendo. Initially released in summer 2016, the game immediately became a global hit and was downloaded for more than 500 million times. Its popularity and financial success have earned it the reputation of the wayfarer of the LBG genre. Pokémon GO is a multiplayer online game, where players move around to find pokémon which they capture and collect, move around to find gyms which they can control and move around to hatch eggs from which they can receive rare pokémon [13]. As a multiplayer online game, it has to deal with technical cheating problems present typically in online video games. However, it also contains the added challenge of being heavily reliant on sensor data (in particular, location data), which adds another technical cheating issue that the developers need to resolve.

3.2 Cheating in Pokémon GO

To discover the most rampant forms of cheating, we conducted ethnographic observations among Pokémon GO players. We actively participate in the game's social communities and read online forums. We also drew from previous literature on the topic [17] who had investigate cheating in Pokémon GO from a broader perspective, not just technical cheating. Through discussions with players, reading chats, following cheating-focused reddit pages such as r/pokemongospoofing and r/pokemongodev and drawing from our technical background, we obtained insights into the main forms of cheating in Pokémon GO. These can be summarized as follows.

- Location spoofing. In one way or another, falsifying the GPS data to enable players to travel around the game map at will, whereas legitimate players need to physically move to travel.
- Botting. Using automated means to gain levels and powerful pokémon in the game. Automatically deploying pokémon to gyms and controlling them, preventing legitimate players from taking them over.
- Assistance tools. There exists a wide range of assistance tools players may use. These include maps where players may see information that the game does not otherwise disclose and automatic catching devices. It is worth noting, that Niantic has also released an official automatic catch device, but the interface for that device has been utilized for other purposes as well, all of which may not be considered entirely legitimate.

In addition, non-technical forms of cheating exists, such as playing with multiple accounts which is against the developer's official rules [17]. However, here we focus in particular on the technical cheating and its prevention.

3.3 Ethical considerations of technical analysis

In Niantic terms of service[2], Niantic states: *"you may not (–) reverse engineer, decompile or disassemble the Apps"*. While violating the application publishers terms of service is not a crime[3], there are legal and technical limitations to what kind of analysis can be performed on freely available apps. In essence, there are two issues that limit the extent of the technical analysis of applications:

- Ethical guidelines, research ethics and legislation prevent more aggressive forms of technical analysis and reverse engineering attempts.
- Much of anticheat measures take place on the server side, and obtaining information about them is difficult or technically not possible. Through second-hand information some factors can be recognized, but quickly this start to get speculative.

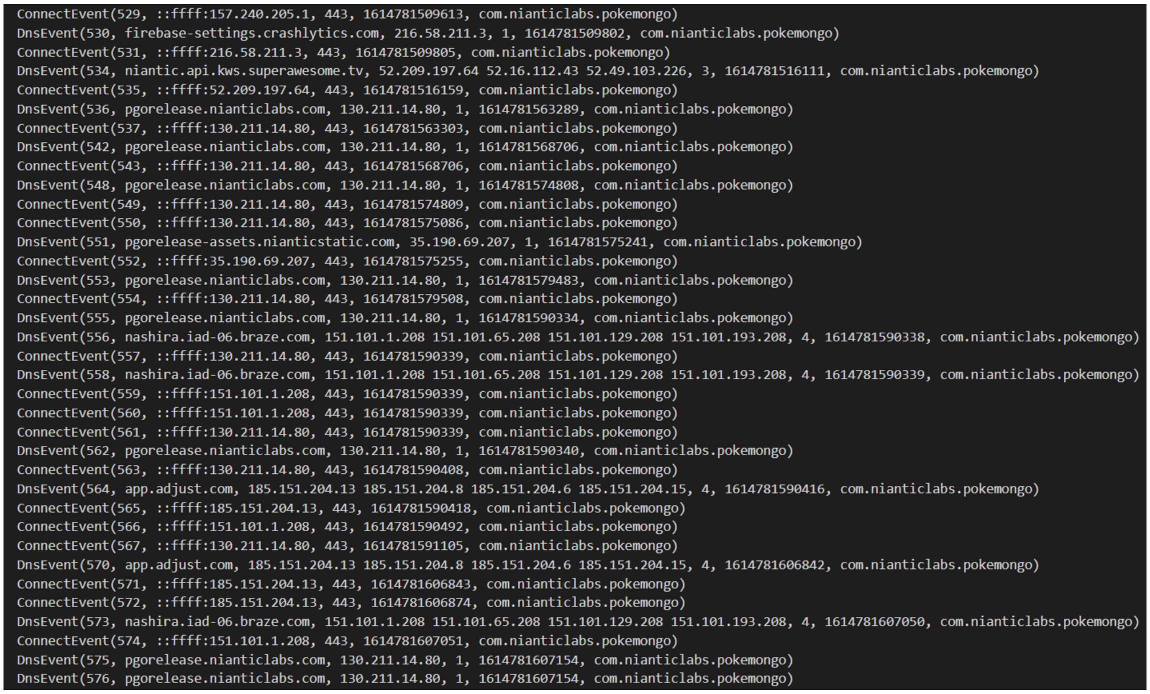
A difference needs to be made between static analysis and observation of how an application behaves, and active tinkering and tampering with the app functionality and network traffic. In our case, we remained at the level of static analysis and at no point sent any falsified data to Niantic servers or tried to reverse engineer server-side cheating counter measures. This was done to abide by ethical conduct and research ethics.

[2] https://nianticlabs.com/terms/en/, updated May 15, 2019

[3] https://www.eff.org/deeplinks/2010/07/court-violating-terms-service-not-crime-bypassing

3.4 Data sources

3.4.1 Used tools. The version of Pokémon GO we investigated was downloaded from apkmirror[4] which should correspond to the actual apk downloaded to the Android devices themselves.



```
ConnectEvent(529, ::ffff:157.240.205.1, 443, 1614781509613, com.nianticlabs.pokemongo)
DnsEvent(530, firebase-settings.crashlytics.com, 216.58.211.3, 1, 1614781509802, com.nianticlabs.pokemongo)
ConnectEvent(531, ::ffff:216.58.211.3, 443, 1614781509805, com.nianticlabs.pokemongo)
DnsEvent(534, niantic.api.kws.superawesome.tv, 52.209.197.64 52.16.112.43 52.49.103.226, 3, 1614781516111, com.nianticlabs.pokemongo)
ConnectEvent(535, ::ffff:52.209.197.64, 443, 1614781516159, com.nianticlabs.pokemongo)
DnsEvent(536, pgorelease.nianticlabs.com, 130.211.14.80, 1, 1614781563289, com.nianticlabs.pokemongo)
ConnectEvent(537, ::ffff:130.211.14.80, 443, 1614781563303, com.nianticlabs.pokemongo)
DnsEvent(542, pgorelease.nianticlabs.com, 130.211.14.80, 1, 1614781568706, com.nianticlabs.pokemongo)
ConnectEvent(543, ::ffff:130.211.14.80, 443, 1614781568706, com.nianticlabs.pokemongo)
DnsEvent(548, pgorelease.nianticlabs.com, 130.211.14.80, 1, 1614781574808, com.nianticlabs.pokemongo)
ConnectEvent(549, ::ffff:130.211.14.80, 443, 1614781574809, com.nianticlabs.pokemongo)
ConnectEvent(550, ::ffff:130.211.14.80, 443, 1614781575086, com.nianticlabs.pokemongo)
DnsEvent(551, pgorelease-assets.nianticstatic.com, 35.190.69.207, 1, 1614781575241, com.nianticlabs.pokemongo)
ConnectEvent(552, ::ffff:35.190.69.207, 443, 1614781575255, com.nianticlabs.pokemongo)
DnsEvent(553, pgorelease.nianticlabs.com, 130.211.14.80, 1, 1614781579483, com.nianticlabs.pokemongo)
ConnectEvent(554, ::ffff:130.211.14.80, 443, 1614781579508, com.nianticlabs.pokemongo)
DnsEvent(555, pgorelease.nianticlabs.com, 130.211.14.80, 1, 1614781590334, com.nianticlabs.pokemongo)
DnsEvent(556, nashira.iad-06.braze.com, 151.101.1.208 151.101.65.208 151.101.129.208 151.101.193.208, 4, 1614781590338, com.nianticlabs.pokemongo)
ConnectEvent(557, ::ffff:130.211.14.80, 443, 1614781590339, com.nianticlabs.pokemongo)
DnsEvent(558, nashira.iad-06.braze.com, 151.101.1.208 151.101.65.208 151.101.129.208 151.101.193.208, 4, 1614781590339, com.nianticlabs.pokemongo)
ConnectEvent(559, ::ffff:151.101.1.208, 443, 1614781590339, com.nianticlabs.pokemongo)
ConnectEvent(560, ::ffff:151.101.1.208, 443, 1614781590339, com.nianticlabs.pokemongo)
ConnectEvent(561, ::ffff:130.211.14.80, 443, 1614781590339, com.nianticlabs.pokemongo)
DnsEvent(562, pgorelease.nianticlabs.com, 130.211.14.80, 1, 1614781590340, com.nianticlabs.pokemongo)
ConnectEvent(563, ::ffff:130.211.14.80, 443, 1614781590408, com.nianticlabs.pokemongo)
DnsEvent(564, app.adjust.com, 185.151.204.13 185.151.204.8 185.151.204.6 185.151.204.15, 4, 1614781590416, com.nianticlabs.pokemongo)
ConnectEvent(565, ::ffff:185.151.204.13, 443, 1614781590418, com.nianticlabs.pokemongo)
ConnectEvent(566, ::ffff:151.101.1.208, 443, 1614781590492, com.nianticlabs.pokemongo)
ConnectEvent(567, ::ffff:130.211.14.80, 443, 1614781591105, com.nianticlabs.pokemongo)
DnsEvent(570, app.adjust.com, 185.151.204.13 185.151.204.8 185.151.204.6 185.151.204.15, 4, 1614781606842, com.nianticlabs.pokemongo)
ConnectEvent(571, ::ffff:185.151.204.13, 443, 1614781606843, com.nianticlabs.pokemongo)
ConnectEvent(572, ::ffff:185.151.204.13, 443, 1614781606874, com.nianticlabs.pokemongo)
DnsEvent(573, nashira.iad-06.braze.com, 151.101.1.208 151.101.65.208 151.101.129.208 151.101.193.208, 4, 1614781607050, com.nianticlabs.pokemongo)
ConnectEvent(574, ::ffff:151.101.1.208, 443, 1614781607051, com.nianticlabs.pokemongo)
DnsEvent(575, pgorelease.nianticlabs.com, 130.211.14.80, 1, 1614781607154, com.nianticlabs.pokemongo)
DnsEvent(576, pgorelease.nianticlabs.com, 130.211.14.80, 1, 1614781607154, com.nianticlabs.pokemongo)
```

Fig. 1. A screenshot of the data that the DNS and Connect event listener captured when booting up Pokémon GO

The look into the Pokémon GO .apk was performed with the popular tool for decompiling .apks called Apktool[5]. Packet sniffing was performed by using Android's device policy app which can also log DNS queries per application[6]. In addition, we used tcpdump and dnsmasq to validate the findings. The traffic is encrypted so any inspection cannot be done without tampering with the application itself, since certificate pinning is used. The raw output of the packet sniffing is depicted in Figure 1. This was filtered by comparing phone static networking with that of the Pokémon GO application. Each unique domain was obtained and further investigated. The summary of this investigation is visualized in Table 1.

3.4.2 Gray literature and other sources. In addition to our own technical analysis, we used our own experience of Pokémon GO, read Niantic's blogposts about technical cheating and technical reddit forums dedicated to the analysis of Pokémon GO[7].

[4] https://www.apkmirror.com/apk/niantic-inc/pokemon-go/pokemon-go-0-203-1-release/pokemon-go-0-203-1-android-apk-download/download/?forcebaseapk

[5] https://ibotpeaches.github.io/Apktool/install/

[6] https://github.com/googlesamples/android-testdpc

[7] As one important source we used the following subreddit pages: (1) reddit/r/pokemongodev; and reddit/r/pokemongospoofing

Table 1. Domain names called by the Pokémon GO client app

Domain name	Explanation
adjust.com	3rd party service. Botting fraud detection and data collection management.
nianticlabs.com	The developer.
crashlytics.com	3rd party service. Crash tracking and fixing.
facebook.com	3rd party service. Provides a login service for players.
fastly.net	3rd party service. Provides security checks, DDoS protection and more.
braze.com	3rd party service. Focused on data-driven increasing customer engagement and retention.
superawesome.tv	3rd party service. Aims to make "the internet safer for kids".
helpshift.com	3rd party service. An in-app customer feedback tool.
nianticstatic.com	The developer.
googleapis.com	3rd party service. API for communicating with Google Services.

Unsurprisingly, due to the huge popularity of Pokemon GO, several forum and blog posts on technical analysis of the game have surfaced. Many of the details we discuss in this article is only available in these sources and not in academic publications. Therefore, we make use of this gray literature to support our technical analysis and observations.

3.5 Data analysis

The Gioia method for qualitative analysis and synthesis typically uses in-depth interviews as the primary source of data, but works with any source of qualitative data [7]. The method is particularly suitable for our case as we have various qualitative data sources (technical analysis results + gray literature) that need to be synthesized. Following Gioia et al. [7], we clustered our findings together based on the type of anticheat measure that we detected. We then connected the raw findings into 2nd order themes based on the type of analysis, which we further linked to three aggregate dimensions: (1) post hoc analyses; (2) verbal measures; and (3) preemptive measures. The results of this analysis are shown in Fig 2.

4 RESULTS: CHEATING COUNTERMEASURES IN LBGS

In this section we go through our results in more detail. We are discussing the findings based on the discovered 2nd order themes: (1) anomaly detection; (2) anticheat measure communication; (3) obfuscation; and (4) client verification.

4.1 Anomaly detection

Typically with anomaly detection, a machine learning model is trained with data collected from verified legitimate players. The more diverse and high quality this data set is, the less false positives it will yield when used for anomaly detection. Due to the various ways Pokémon GO is played and the various devices players use, anomaly detection may

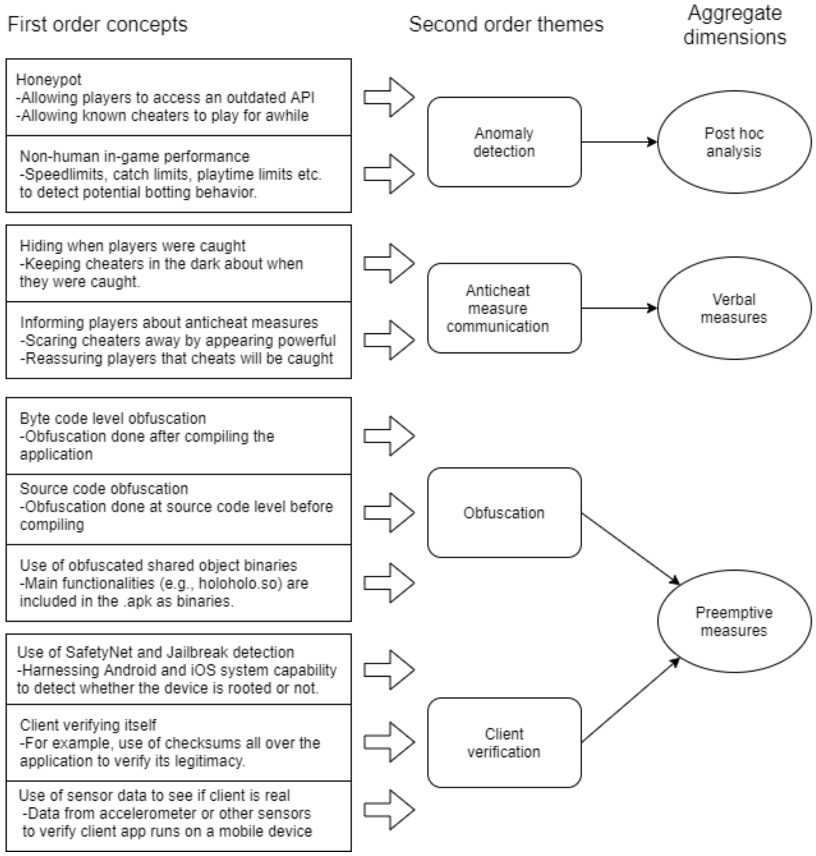

First order concepts
Second order themes
Aggregate dimensions
Honeypot
-Allowing players to access an outdated API
-Allowing known cheaters to play for awhile
Non-human in-game performance
-Speedlimits, catch limits, playtime limits etc. to detect potential botting behavior.
Hiding when players were caught
-Keeping cheaters in the dark about when they were caught.
Informing players about anticheat measures
-Scaring cheaters away by appearing powerful
-Reassuring players that cheats will be caught
Byte code level obfuscation
-Obfuscation done after compiling the application
Source code obfuscation
-Obfuscation done at source code level before compiling
Use of obfuscated shared object binaries
-Main functionalities (e.g., holoholo.so) are included in the .apk as binaries.
Use of SafetyNet and Jailbreak detection
-Harnessing Android and iOS system capability to detect whether the device is rooted or not.
Client verifying itself
-For example, use of checksums all over the application to verify its legitimacy.
Use of sensor data to see if client is real
-Data from accelerometer or other sensors to verify client app runs on a mobile device
Anomaly detection
Anticheat measure communication
Obfuscation
Client verification
Post hoc analysis
Verbal measures
Preemptive measures

Fig. 2. Findings of the study clustered according to Gioia et al. [7]. These measures are not exhaustive, but are a reflection of anticheat efforts in LBGs and Pokémon GO.

be used only as one tool among many for detecting potential cheaters. Rule-based systems can also be used as part of anomaly detection with limits set for certain types of in-game behaviors.

4.1.1 Honeypot: accessing an outdated API version. For quite some time, Niantic allowed players to access their Pokémon GO server API[8] with an old outdated version which legitimate clients did not use. One possible reason for this is to gather a dataset of illegitimate playing, which could then be used to train an ML model for anomaly detection purposes. Furthermore, allowing known cheaters to play could serve the function of wasting their time, as they would spend time and resources in gathering resources which they would eventually just lose when banned. Another reason for allowing players to play with an outdated API is that there existed various assistant tools for Pokémon GO, many of which would not be considered cheating by other players. These included the *PokeNurse* tool which would allow players to mass transfer Pokémon and see details about their stats more easily, features that at the time (in 2016-2017) were not yet added as features to the main Pokémon GO client.

[8] https://pgorelease.nianticlabs.com/plfe/

4.1.2 Non-human in-game performance. Rule-based systems can be set server side to limit playing to what can be viewed as humanly possible. For example, Pokémon GO used to include limits to how many Pokémon could be caught per day and per week, and some legitimate players also ran into these limitations. Limits could also be set to how fast you can travel, reaching a place faster than an airplane would could trigger such a check. These types of checks are performed server side to detect possible botting or spoofing activity. If players capture an unrealistic number of rare Pokémon in a short time, they may be using a 3rd party map assistance tool.

Here we observe a race between cheaters and anticheat measures, where cheaters can add human-like characteristics to bots to make them behave as they were humans, while the cheating prevention operations try to catch as many bot accounts as possible while avoiding false positives. In the end, cheating detection based on purely non-human characteristics can be viewed as an impossibility due to the enormous variance in devices and play styles that exist globally in a game such as Pokémon GO, and the ability of cheaters to tune their bots to (almost) fully resemble the playing of human players. Despite these shortcomings, non-human in-game performance checking can be considered a relatively easy and a necessary anticheat precaution to be implemented for any multiplayer game, including LBGs and Pokémon GO.

4.2 Anticheat measure communication

4.2.1 Hiding when players were caught. As is typical in professional cybersecurity business, secrecy is important also in technical cheating prevention. Revealing too much about protection measures may enable adversaries to better avoid getting caught, and even assist them in reverse engineering the entire protection scheme. Hiding when players have been caught has been a common policy in multiplayer online games for some time, and Niantic has also adopted this approach. Bans are issued to players in so-called "ban waves", not when players were initially caught. This makes it difficult for cheaters or bot account managers to know what they did wrong and when.

4.2.2 Informing players about anticheat measures. To date, Niantic has released three statements to Pokémon GO players about the anticheat measures. These releases are also aligned with the principle of secrecy, as they give away very little about technical cheating prevention measures while reassuring players that a lot is being done. More transparent communication of anticheat measures takes place in the form of banning players. However, for some technical cheaters, ban waves may be reassuring as they discover that they have not been detected.

4.3 Obfuscation

Reverse-engineering the source code and tampering with mobile games can be made significantly more difficult by employing obfuscation [20]. Obfuscation means rewriting the program code in a more complex form in order to make it hard for humans to understand and for automatic tools to de-obfuscate. Even though the meaning of the code is hidden, the functionality of the obfuscated programs remains the same [4].

Obfuscation can take place on source code level (before compilation) or on the bytecode or binary level (after compilation). By making use of dynamic obfuscation and regularly changing how the code is obfuscated, the game developer can decrease the time window available for de-obfuscating the code.

4.4 Client verification

4.4.1 Root detection to prevent sensor data falsification. According to the Android developer guide[9], SafetyNet Attestation API enables service providers (in this case, the Niantic Pokémon GO servers) to verify whether the device that runs the Pokémon GO client side application passes a set of Android compatibility tests. SafetyNet is therefore able to detect most rooted devices. Here, rooted devices would allow players to modify phone sensor data at will, enabling easy technical cheating. Thus, using SafetyNet to make using rooted Android phones more difficult can be considered a technical cheating prevention measure. The downside is of course that legitimate players may also have rooted devices, forcing them to opt for non-rooted devices when they want to play, or find a technical way to circumvent SafetyNet. Similarly, for iOS devices there exists Jailbreak detection that essentially does the same as SafetyNet does for Android.

Root detection coupled with an operating system that does not allow location spoofing can be considered quite a secure counter measure for spoofing. Despite this, rampant spoofing problems still persist in Pokémon GO. For example, r/pokemongospoofing subreddit as of March, 2021 lists numerous ways to falsify location data for both Android and iOS, for both rooted and non-rooted devices. This goes to demonstrate, that sensor data falsification remains a huge unsolved issue in the LBG and other sensor data-based games industries.

4.4.2 Use of Additional Sensor data. Mobile phones contain a wide range of sensors from luminosity sensors to location sensors and accelerometers. In normal use, the data provided by these sensors is constantly changing as the phone is being carried around. This data may be used to determine the likelihood of a player being legitimate. There exists evidence that Niantic is collecting a lot of data from players. As an example, the *"Adventure Sync"* feature of Pokémon GO transmits constant location data to Niantic and furthermore, requests full access to data collected by Google Fit. With such data at their use, it may be possible to distinguish patterns, separate legitimate players from non-legitimate players and so forth. However, similarly to the use of anomaly detection, the use of additional sensor data to detect cheaters is speculative.

4.5 Other forms of cheating prevention

In our network analysis, we noticed that the Pokémon GO client app connects to various services such as d2.shared.global.fastly.net, which[10] provides security checks, API protection and DDoS protection among others. We also saw the client app connect to app.adjust.com which provides cheating prevention related services[11] such as:

- Detect in-app bot fraud at scale
- Prevent ad fraud from compromising your budgets and data

In addition to these 3rd party services there likely exists many more solutions purely at the developers' end which aim to prevent technical cheating.

[9] https://developer.android.com/training/safetynet
[10] according to https://www.fastly.com
[11] According to https://www.adjust.com

5 DISCUSSION

5.1 Key Findings

Through technical analysis of Pokémon GO and additional work, we detected anticheat measures in two paramount technical categories: preemptive measures and post hoc measures, and a third category which related to the communication of these measures and when to issue technical punishment (bans). Based on these findings, the following anticheat measures for LBGs can be presented:

- Root detection to make sensor data falsification harder.
- Server side checks to rule out super human in-game performance.
- Use of machine learning approaches such as anomaly detection to catch cheaters. Usually pending further investigation due to the unreliability of the approach.
- Allowing cheaters to access an outdated API for some time to collect data on verified bot activity.
- Camouflaging from players the information when they were caught.
- Use of additional sensor data from the playing device such as accelerometer telemetry to calculate the probability of the player being a real person.
- Obfuscation of the source code or byte code to make reverse engineering of the application more difficult.
- Use of checksums and constantly changing client side security to make reverse engineering the networking more difficult.

In addition, various other measures exist and can be implemented. Secrecy is a major part of anticheat actions according to Niantic, and it would not be possible to technically objectively demystify all these measures. In the end, technical cheating prevention remains an arms race between developers and cheaters.

5.2 Theoretical Implications

Technical cheating prevention shares many similarities with cybersecurity. For example, keeping an outdated API operational or intentionally accepting API calls or requests with erroneous parameters has a lot in common with honeypots and proactive cybersecurity measures in general [15, 23]. Accepting requests a legitimate client should not make, can be used to catch both technical cheaters in video games and malicious adversaries whom cybersecurity professionals deal with. This causes cheaters and adversaries to waste their time while their tactics and behavioral patterns can be observed and recorded. Finally, when the time is right, they can be banned and removed from the system. In both cybersecurity and cheating prevention, however, one has to consider how much harm the cheater or adversary is allowed to cause. In a honeypot environment, an adversary can be observed for a long time and often does not cause any real damage, but cheaters in multiplayer games cause immediate harm to other players. From a theoretical standpoint, we therefore propose that technical cheating prevention literature could draw from cybersecurity research, and even contribute to it, and vice versa.

As an implication, many methods and concepts from cybersecurity could be used in cheating prevention and anti-cracking efforts even more effectively, also in the mobile environment. For example, Moving Target Defense (MTD) could be applied to protect applications from analysis and tampering [9]. MTD refers to approaches that are diverse and that continually and dynamically change in order to add complexity and cost for adversaries. This way, the analysis of the exact mechanics of a mobile game and opportunities for cheating is made more difficult, and the application's resiliency against reverse engineering is increased. As one example of MTD, each update of the application could be differently obfuscated to impede analysis.

Our findings also contribute to the extant literature on cheating in games in general. In particular, we extend the work of Paay et al. [17] who studied cheating in Pokémon GO via player interviews, and looked at the situation from the players' perspective. In this work, we respond to the many concerns raised by the players in that study [17] by showing what technical counter measures have been implemented by the developer.

5.3 Implications for Game Design

According to Niantic[12], 90% of players who received a warning due to cheating changed their playing habits. However, it is not clear to what data this is based on, and what it implies in practise. As players can create new accounts at will in Pokémon GO, it may be possible that players who receive warnings on some of their accounts just move to cheat on next ones.

Cheating is in many ways a social phenomenon [17]. Mostly it relates to competition between players (such as the desire to dominate others), but there exists other social aspects to it as well. Players in the LBGs Pokémon GO and Ingress have been found to be stratified based on their team [11], and teams may be more vigorous in detecting and condemning cheating among their opponents than among their own.

Interestingly, some forms of technical cheating were respected by players. For instance, some people who created city-wide maps (operated by a large bot army) and provided this information to other players to use were celebrated as heroes[13]. Drawing from this experience, game developers could cooperate with 3rd party tool creators or make notes of which features players like and possible implement them into their game officially.

In addition to the measures discussed in this study, various other solutions exist. One interesting example of how game developers can combat cheaters and cracked versions of games comes from Spyro: Year of the Dragon, a platform game developed by Insomniac Games. After a pirated version of the game has been detected, the gameplay still continues normally, but the gaming experience keeps slowly degrading. For example, items start to disappear from the levels and from the inventory, and the game starts randomly crashing. These effects only get worse as the game proceeds, ultimately making it unplayable. Moreover, the anti-piracy mechanisms were protected using checksums and obfuscation, which made removing them harder [14].

5.4 Limitations and Future Work

In this study, we focused on a single LBG, Pokémon GO. Being by far the most popular LBG, studying 3rd party tools and cheats created for the game, as well as the game's cheating prevention measures is a feasible way to provide insights into the current state of cheating prevention in LBGs. In particular, it enabled us to look at how the industry leader has attempted to solve the crucial issue of verifying the location sensor data from players' devices. This being said, our analysis is not exhaustive and is limited by a set of factors, including the study sample and lack of access to fully disclosed cheating countermeasures.

While the academic study of cheating countermeasures and prevention is important, it is also a double edged sword. The general tools for cheating prevention are to be shared, but more specific solutions should not be disclosed to not encourage and enable the reverse engineering of the specific systems and their protection schemes. For studying technical solutions more in-depth, ad hoc studies need to be conducted instead of studying existing applications.

[12] https://nianticlabs.com/blog/cheatingupdate-022321

[13] One example of this comes from Finland, where the creator of a city-wide Pokémon GO map received donations and respect from other players according to a local newspaper: https://www.hs.fi/kaupunki/art-2000005308047.html

[14] https://www.gamasutra.com/view/feature/131439/keeping_the_pirates_at_bay.php

As future research directions, one particularly promising avenue is the utilization of machine learning, for example, anomaly detection for capturing technical cheaters. Here we identified that the collection of a dataset comprising of purely legitimate players may be a challenge, and solutions for the dataset collection are needed. On the flip side, machine learning with human data could be used to train more powerful bots that automatically play the game, and are much more difficult to detect. Similarly to the field of cybersecurity, we could see an arms race here between technical cheaters and cheating prevention. Players can find playing itself meaningful, or then they can find the outcomes of playing meaningful [14]. One hypothesis for future research is that that when the outcome is more important than playing, players are more likely to cheat.

6 CONCLUSION

As long as online video games continue to intertwine monetary value and amount of playing, bot creation will remain a potentially profitable business. And as long as online games keep having a inter-human competitive element, some players will be motivated to seek tool -assistance (technical cheats) to help them play the game at a higher level. Additionally, providing technical tools that make playing easier, but not necessarily in a game-breaking way, have demand. These may include tools such as the already mentioned *PokéNurse* that help players manage their inventory. It is typically unclear whether these types of tools are technical cheating in the same way as, for example, botting and spoofing in LBGs.

In this work we identified various forms of cheating and counter measures in the most popular LBG, Pokémon GO. Our findings show that the endeavor of technical cheating prevention is parallel to cybersecurity protection. We saw Pokémon GO use many of the same methods that, for example, online banking apps use including utilizing root detection and verifying client side inputs. We saw evidence of setting up honeypots to catch cheaters and the use of external anti-cheat services. Despite all this, it is clear there exists multi-layered and various kinds of security measures, all of which are not and should not be disclosed to the general public, even in the form of academic research. Technical cheating prevention, similarly to cybersecurity protection, is enhanced when keeping adversaries in the dark.

REFERENCES

[1] Nathaniel E. Baughman and Brian Neil Levine. 2001. Cheat-Proof Playout for Centralized and Distributed Online Games. In *Proceedings of the Twentieth IEEE Computer and Communication Society INFOCOM Conference*. Anchorage, AK, USA.

[2] Vivian Hsueh Hua Chen and Jeremy Ong. 2018. The rationalization process of online game cheating behaviors. *Information, Communication & Society* 21, 2 (2018), 273–287. https://doi.org/10.1080/1369118X.2016.1271898

[3] Vivian Hsueh Hua Chen and Yuehua Wu. 2015. Group identification as a mediator of the effect of players' anonymity on cheating in online games. *Behaviour & Information Technology* 34, 7 (2015), 658–667. https://doi.org/10.1080/0144929X.2013.843721

[4] Christian Collberg, Clark Thomborson, and Douglas Low. 1997. *A taxonomy of obfuscating transformations*. Technical Report. Department of Computer Science, The University of Auckland, New Zealand.

[5] Mia Consalvo. 2009. *Cheating: Gaining Advantage in Videogames*. MIT Press, Cambridge, MA, USA.

[6] Wu-Chang Feng, Ed Kaiser, and Travis Schluessler. 2008. Stealth Measurements for Cheat Detection in On-line Games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*. Worcester, MA, USA, 15–20.

[7] Dennis A Gioia, Kevin G Corley, and Aimee L Hamilton. 2013. Seeking qualitative rigor in inductive research: Notes on the Gioia methodology. *Organizational research methods* 16, 1 (2013), 15–31. https://doi.org/10.1177/1094428112452151

[8] Greg Hoglund and Gary McCraw. 2007. *Exploiting Online Games: Cheating Massively Distributes Systems*. Addison-Wesley, Reading, MA, USA.

[9] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. 2011. *Moving target defense: creating asymmetric uncertainty for cyber threats*. Vol. 54. Springer Science & Business Media.

[10] Michal Kedziora, Aleksander Gorka, Aleksander Marianski, and Ireneusz Jozwiak. 2020. Anti-Cheat tool for detecting unauthorized user interference in the unity engine using Blockchain. In *Data-Centric Business and Applications*. Springer, 191–209. https://doi.org/10.1007/978-3-030-34706-2_10

[11] Samuli Laato, Nobufumi Inaba, Mauri Paloheimo, and Teemu Daniel Laajala. 2021. Group polarisation among location-based game players: an analysis of use and attitudes towards game slang. *Internet Research* (2021).

[12] Samuli Laato, Bastian Kordyaka, AKM Islam, and Konstantinos Papangelis. 2021. Landlords of the Digital World: How Territoriality and Social Identity Predict Playing Intensity in Location-based Games. In *Proceedings of the 54th Hawaii International Conference on System Sciences*. ScholarSpace at University of Hawaii, Wailea, Maui, Hawaii, 744. https://doi.org/10.24251/HICSS.2021.091

[13] Samuli Laato, Tarja Pietarinen, Sampsa Rauti, and Erkki Sutinen. 2020. Potential Benefits of Playing Location-Based Games: An Analysis of Game Mechanics. In *Selected revised works from the International Conference on Computer Supported Education*. Springer, Springer, 557–581. https://doi.org/10.1007/978-3-030-58459-7_27

[14] Samuli Laato, Sampsa Rauti, AKM Najmul Islam, and Erkki Sutinen. 2021. Why playing augmented reality games feels meaningful to players? The roles of imagination and social experience. *Computers in Human Behavior* 121 (2021), 106816.

[15] Samuel Laurén, Sampsa Rauti, and Ville Leppänen. 2016. An interface diversified honeypot for malware analysis. In *Proccedings of the 10th European Conference on Software Architecture Workshops*. 1–6.

[16] Peter Laurens, Richard F. Paige, Phillip J. Brooke, and Howard Chivers. 2007. A Novel Approach to the Detection of Cheating in Multiplayer Online Games. In *12th IEEE International Conference on Engineering Complex Computer Systems*. 97–106.

[17] Jeni Paay, Jesper Kjeldskov, Daniele Internicola, and Mikkel Thomasen. 2018. Motivations and practices for cheating in Pokémon Go. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*. 1–13. https://doi.org/10.1145/3229434.3229466

[18] Cale J Passmore, Mathew K Miller, Jun Liu, Cody J Phillips, and Regan L Mandryk. 2020. A Cheating Mood: The Emotional and Psychological Benefits of Cheating in Single-Player Games. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*. 58–70. https://doi.org/10.1145/3410404.3414252

[19] Sampsa Rauti, Samuli Laato, and Tarja Pietarinen. 2020. Learning social skills and accruing social capital through pervasive gaming. In *CEUR Workshop Proceedings*, Vol. 2685. CEUR, Heidelberg, Germany, Online.

[20] Sampsa Rauti, Samuel Laurén, Petteri Mäki, Joni Uitto, Samuli Laato, and Ville Leppänen. 2020. Internal interface diversification as a method against malware. *Journal of Cyber Security Technology* (2020), 1–26. https://doi.org/10.1080/23742917.2020.1813397

[21] Jouni Smed and Harri Hakonen. 2017. *Algorithms and Networking for Computer Games* (second ed.). John Wiley & Sons, Chichester, UK.

[22] Jouni Smed, Timo Kaukoranta, and Harri Hakonen. 2002. Aspects of Networking in Multiplayer Computer Games. *The Electronic Library* 20, 2 (2002), 87–97.

[23] Lance Spitzner. 2003. Honeypots: Catching the insider threat. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE, 170–179.

[24] Jianxin Jeff Yan and Hyun-Jin Choi. 2002. Security Issues in Online Games. *The Electronic Library* 20, 2 (2002), 125–133.