Deducing Minecraft coordinates using footage
Paper for the subject “Computer Security”

Author: Semjon Kravtšenko
Supervisor: Meelis Roos

Tartu 2020
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>3</td>
</tr>
<tr>
<td>2 Idea</td>
<td>4</td>
</tr>
<tr>
<td>3 Creating “exploit”</td>
<td>6</td>
</tr>
<tr>
<td>4 Defense</td>
<td>9</td>
</tr>
<tr>
<td>5 Sources</td>
<td>10</td>
</tr>
</tbody>
</table>
1 Introduction

Minecraft [1] is a popular computer game, created by Mojang. In this game, players can make changes to a large in-game world consisting of blocks. There are both client-side and server-side software, players typically use the client to connect to the server, which stores an in-game world. Under some specific circumstances, users (players) may want to achieve two goals at once: streaming their gaming sessions and keep their in-game location private, so that others could not interfere.

This paper demonstrates that seemingly harmless and unimportant feature of the server software can lead to damages, apparently because of inconsistent threat models.
2 Idea

Minecraft uses a Pseudo Random Number Generator to generate terrain, which is set up by the “seed”. It can be easily tested that two worlds generated with the same seed and the same version of the game will be identical\(^1\). Older versions of Minecraft server software make seed information accessible to clients, but newer versions do not. Luckily, there is a way to obtain seed from specific terrain features, as demonstrated in [2].

Given the seed, we could generate a world with the same formations as original (it will not have modifications made by players, of course), and then a find match with the terrain visible on footage. This process can be automated, leading us to a brute force attack.

Such an attack, however, is not viable, since Minecraft world generation algorithm is rather slow, because it generates many features of the world. Our attack could be optimized, if we choose only one feature of the world for finding a match, and then modify/recreate generation code so that it only contains code necessary for the generation of this feature. This paper considers an attack based on matching bedrock formations (bedrock is a type of block near the world bottom that normally could not be added/removed by players).

In such formation\(^\text{ii}\), approximately 20% of blocks are bedrock (tested on 1.14.4, sample size: >10000 blocks). It can be calculated using the formula in [3], that information about 1 block has entropy value close to \(e = 0.72\), \(n\) blocks will have total entropy value of \(0.72 \cdot n\), since each block does not depend on others. (This is Level 4 bedrock pattern. Bedrock is a darker block.)
confirmed later in this paper. Also, it is evident from the shape of the pattern.) Therefore, each block at the “bedrock level” could be a top-left corner of a false positive formation with some small probability \( P \leq \frac{4}{20.72 \cdot n^3} \), where \( n \) means how many blocks of a formation we need (to find coordinates of it). Constant 4 is needed to account for different rotations. Normally, the Minecraft world has a square shape and is \( 6 \times 10^7 \) blocks across. Thanks to the linearity of expectation, the following formula can be constructed: \( F = P \times B \), where \( B = (6 \times 10^7)^2 = 3.6 \times 10^{15} \) and \( F \) means the expected value of false positives. If \( F = 0.05 \), \( n \approx 81 \). For more practical values \( B = 10^{12} \) and \( F = 0.5 \), \( n \approx 60 \). This value is small enough to expect such a part of bedrock formation to be exposed in some cases.

The described attack is not new. [4] There is evidence that it was used since late 2016. Source code, required for the attack, was posted in early 2017. [5]
3 Creating “exploit”

Since Minecraft is written in Java, it is possible to decompile it to source code (for example, using Procyon). However, such code is obfuscated and will not be easily understandable. Luckily, there are two community projects, aiming to deobfuscate the code: MCP and yarn. On top of that, official obfuscation maps recently became available [6]. A tool called Enigma [7] can be used to apply those obfuscation maps. The code responsible for generating bedrock formations (in 1.14.4) is the following. <...> means that unnecessary code was removed, comments are formatted like this: // Comment.

```java
net.minecraft.world.level.levelgen.WorldgenRandom

public class WorldgenRandom extends Random
    ...
    // As can be seen later, parameters are a position of a chunk of terrain in the world.
    // For some reason, world seed value is actually not used
    public long setBaseChunkSeed(final int integer1, final int integer2) {
        final long long4 = integer1 * 341873128712L + integer2 * 132897987541L;
        this.setSeed(long4);
        return long4;
    }
    ...
```

```java
net.minecraft.world.level.levelgen.NoiseBasedChunkGenerator

public void buildSurfaceAndBedrock(final ChunkAccess bxh) {
    final ChunkPos bhd3 = bxh.getPos();
    final int integer4 = bhd3.x;
    final int integer5 = bhd3.z;
    final WorldgenRandom bzk6 = new WorldgenRandom();
    bzk6.setBaseChunkSeed(integer4, integer5);
    ...
    this.setBedrock(bxh, bzk6);
}
```
protected void setBedrock(final ChunkAccess bxh, final Random random) {
    final BlockPos.MutableBlockPos a4 = new BlockPos.MutableBlockPos();
    final int integer5 = bxh.getPos().getMinBlockX();
    final int integer6 = bxh.getPos().getMinBlockZ();
    final T byv7 = this.getSettings();
    final int integer7 = byv7.getBedrockFloorPosition(); // For overworld, hardcoded as 0
    for (final BlockPos ew11 : BlockPos.betweenClosed(integer5, 0, integer6, 
            integer5 + 15, 0, integer6 + 15)) {
        for (int integer9 = integer7 + 4; integer9 >= integer7; --integer9) {
            if (integer9 <= integer7 + random.nextInt(5)) {
                // We are only interested in level 4 pattern, but we cannot simply remove code for generating other layers, since they call nextInt(), which will change state of Random object.
                bxh.setBlockState(a4.set(ew11.getX(), integer9, ew11.getZ()),
                        Blocks.BEDROCK.defaultBlockState(), false);
            }
        }
    }
}

nextInt is «implemented by class Random as if by» [8]:

    public int nextInt(int bound) {
        if (bound <= 0)
            throw new IllegalArgumentException("bound must be positive");

        if ((bound & -bound) == bound) // i.e., bound is a power of 2
            return (int)((bound * (long)next(31)) >> 31);

        int bits, val;
        do {
            bits = next(31);
            val = bits % bound;
        } while (bits - val + (bound-1) < 0);
        return val;
    }
This can be simplified (given that some possible argument values are not used):

```java
public int nextInt(int bound) {
    return next(31) % bound;
}
```

next updates the seed like this:
```
(seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1)
```
and returns this:
```
(int)(seed >>> (48 - bits))
```

Using this code, it is possible to develop a much faster algorithm for generating pattern, so brute force will become viable for smaller $B$ values. One caveat is that our code should preserve all next(int bits) calls.

However, this can be circumvented. Since Java Random is a congruential pseudorandom number generator, it is possible to create an efficient function, which will return the next random value, as if some values were “skipped”, as explained in [9]. In less detail: since the formula for deriving the next seed is $S_{i+1} = (a \times s_i + b) \mod 2^{48}$, formula to derive second next seed is $S_{i+2} = (a \times ((a \times s_i + b) \mod 2^{48}) + b) \mod 2^{48}$, which can be simplified to $S_{i+2} = (a^2 \times s_i + (b \times a + b)) \mod 2^{48}$. Note that $a_2 = a^2$ and $b_2 = b \times a + b$ can both be precalculated ($mod 2^{48}$), so $S_{i+2} = (a_2 \times s_i + b_2) \mod 2^{48}$. Generally, $S_{i+k} = (a_k \times s_i + b_k) \mod 2^{48}$, where $a_k$, $b_k$ can be precalculated. This optimization leads to a brute force attack, which can find the coordinates of a particular bedrock formation, located hundreds of thousands of blocks away from the search starting point (getting this far in Minecraft would normally require hours). An example of such code can be obtained from [5].

Another optimization idea is to write our code to use GPU, since it has many more cores than CPU. This is used in [10].
4 Defense

No footage of bedrock formations should be available. No world downloads should be available, since it is possible to extract patterns from them.

Since an attacker could choose another feature for finding a match (although this will require knowing the seed), no footage of distinct naturally generated terrain should be available at all.

It seems unlikely that this can be completely fixed without significantly changing generation mechanism. Since there are not many players, who both stream and want to remain undiscovered, it is improbable that Mojang will create such a fix. Even if they will, already generated parts of the world will still be susceptible.

A workaround might be to augment the server to allow players to “claim” a portion of the world, so that it becomes inaccessible to others (or, at least, unmodifiable for others). On most public servers, this is actually being done, for example, with WorldGuard plugin. [11]
5 Sources

https://en.wikipedia.org/wiki/Minecraft/

[2] [1.15] Seed Cracking Mod  
https://www.youtube.com/watch?v=1ChmLi9og8Q


https://www.reddit.com/r/2b2t/comments/5n9zlg/bedrock_finder_source_code/

https://minecraft.gamepedia.com/Obfuscation_map

https://github.com/FabricMC/Enigma/

[8] Random (Java Platform SE 8)  
https://docs.oracle.com/javase/8/docs/api/java/util/Random.html#nextInt-int-

https://youtu.be/rSaZezzK2os?t=611

[10] coolmann24 / BedrockFinderCpp (source code)  
https://github.com/coolmann24/BedrockFinderCpp

https://dev.bukkit.org/projects/worldguard/

All sources worked as of 2020-04-19.

The author appreciates and admires works of the creators of Minecraft, as well as people, who discovered and researched the issue. Without them, this paper could not have been possible.

---

1 Some parts of the world may still be different. One of them is mobs’ positions, since they move after being generated. Their generation depend on players’ positions, so freshly generated world will not have mobs in the same places.

2 Level 4 (height=4) bedrock pattern is analyzed, since other layers containing bedrock are partially covered most of the time.