

# VisualBoyAdvance 1.8.0 - stack-based buffer overflow in XPC file parsing results in code execution

jx9716 // TheZZAZZGlitch

## **Abstract**

VisualBoyAdvance is a well-known and widely used emulator for Gameboy and Gameboy Advance systems. Despite its main branch being discontinued since 2004, it still remains a common choice for emulating the aforementioned platforms.

As the emulator's features focus mostly on gaming, the application provides support for importing cheat codes in several formats. Most notably, it supports loading and saving XPC files, which are used by ActionReplay-based cheating devices. However, when importing XPC file data, no bounds checking of any kind is done on the user supplied information. A stack-based buffer overrun vulnerability is present in the file data parsing subroutine, which leads to arbitrary code execution by providing a malicious code list.

## **XPC file format**

XPC is a file format designed to hold cheat code data for multiple games in a single file. The file is required to start with an identifier string, which is normally set to *SharkPortCODES*. After going through several bytes of unknown and irrelevant purpose, we arrive at the file contents - a serialized table of cheat sets to apply, containing a serialized table of cheat lists for each cheat set.

```
uint32_t IdentifierStringLength;
char IdentifierString[IdentifierStringLength];
char Reserved[0xC];
uint32_t NumberOfGames;
[repeat NumberOfGames times] {
    uint32_t GameNameLength;
    char GameName[GameNameLength];
    uint32_t NumberOfCodes;
    [repeat NumberOfCodes times] {
        CodeEntry SingleCodeEntry;
    }
}
```

The knowledge about the structure of a single cheat code entry (marked above as *CodeEntry*) is not necessary for our purposes.

All multibyte integers in the described file format are stored little-endian. Although other identifier strings than *SharkPortCODES* are theoretically possible, the discussed software (VisualBoyAdvance 1.8.0) only supports files with the original identifier, and will reject all files that don't start with this pattern.

## The vulnerability

The emulator in question implements a very simplistic subset of the XPC file format. The file loading process is prefaced by testing whether the first 12 bytes match the traditional signature:

0E 00 00 00 53 68 61 72 6B 50 6F 72 74 43 4F 44 45 53  
. . . . S h a r k P o r t C O D E S

If the check succeeds, the application immediately seeks to the *NumberOfGames* field and proceeds to parse the list of game names, to present a choice dialog to the user.

```
char buffer[1024];
int games = 0;
int len = 0;
// (...)
fread(&games, 1, 4, f);
while(games > 0) {
    fread(&len, 1, 4, f);
    fread(buffer, 1, len, f);
    buffer[len] = 0;
    m_games.AddString(buffer);
    // (...)
}
// (...)
```

GSACodeSelect.cpp:88 - VisualBoyAdvance 1.8.0 source code

The temporary buffer allocated on the stack is only 1024 bytes long, yet the file format parser allows loading name strings of arbitrary length. This is a classic example of a stack-based buffer overflow, which can lead to arbitrary code execution. Since the last mainstream version of original VisualBoyAdvance was compiled back in 2004, none of the standard security mechanisms are present in the executable, making the vulnerability easily exploitable.

## The exploit

Creating an XPC cheat list with a sufficiently long game name is enough to trigger the vulnerability. Additionally, to make the vulnerable code run, the *NumberOfGames* field value has to be greater than 1 - otherwise the emulator will skip processing the game names and will immediately start loading the cheat list.

The process stack is affected by ASLR, but the classic "*jmp esp*" trick is enough to make the exploit very reliable.

```
"\x0e\x00\x00\x00SharkPortCODESxxxxxxxxxxxxx (fake header)
\x02\x00\x00\x00\x00\x05\x00\x00 (the name is 0x500 bytes long)
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA (...) AAAAA (padding)
[offset to jmp esp][payload]"
```

An example proof of concept *calc.exe* exploit should be enclosed to this advisory file.

In order to trigger the vulnerability, the user has to:

- Load any GBA ROM,
- Load the malicious XPS file, most commonly by selecting File » Import » GameShark code file

Although all versions of VBA are vulnerable, the attached exploit will only work with the latest mainstream version, 1.8.0. This is because in the proof of concept payload, the offset to *jmp esp* was chosen to be within the application's image base, so different application versions are incompatible. This can easily be circumvented at a cost of portability between different operating systems - simply choose a different offset within a fixed-base system library. Creating a NOP sled is also a possibility.

## **Affected versions**

All main branch versions of VisualBoyAdvance to this date are affected by this vulnerability.

VBA-ReRecording is a branch of the stable 1.7.2 version of VisualBoyAdvance, and thus, it is also vulnerable. The executables have the NX bit enabled. However, they also contain *call CreateProcess* and *call VirtualProtect* instructions, so constructing a ROP chain circumvents the problem.

VBA-M is the only popular branch of VBA that is actively developed to the current day. All versions of VBA-M still use the same buggy code; however, the downloadable binaries of the emulator are compiled with Microsoft Compiler's */GS* flag, making the attack too hard to perform reliably outside of platforms with low stack cookie entropy, like virtual machines.

## **Vendor status**

Since the mainstream version of VBA has been officially discontinued for more than 10 years, the vendor hasn't been informed about the issue.

## **Disclaimer**

This document, along with all information it contains, is provided "as is", without any warranty. The author is not responsible for the misuse of the information provided in this document. The advisory is provided for educational purposes only. Permission is granted to redistribute this document, as long as its content and copyright notices remain intact.