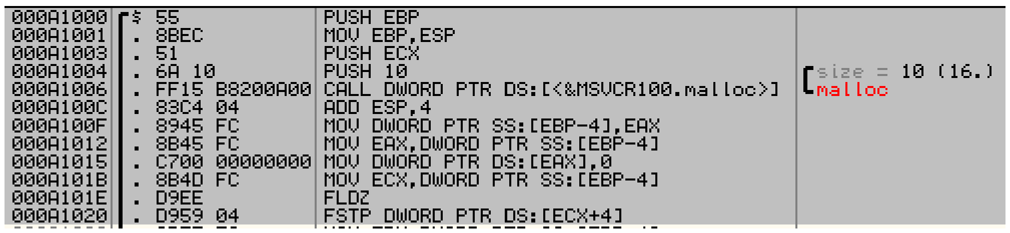
One common problem with developing hacks or external modifications for games/applications is when the target application gets modified through patches, new versions, or so on. This might render offsets, structures, functions, or anything important that is used in the hack as useless if it is hardcoded. For example, assume that the hack puts a hook on a function at 0x1234ABCD. One day, a new version of the application is released and the new compiled version no longer has this function at 0x1234ABCD, but it's at some different address, 0×12345678. Now the hack no longer works, and in the worst case, even crashes the application when used. This becomes annoying because some applications are frequently updated, which in turn requires frequent updates on the part of the hack developer. Even if the updates aren't too frequent, it can be unnecessarily inconvenient to hunt down where the structures, functions, and so on ended up. One solution to this is called signature scanning. This technique is nothing new or special and has been used by both hack developers and anti-virus programs for many years (anti-viruses probably much longer than in hacks). It relies on finding parts of a program through scanning for certain byte patterns. For example, anti-virus programs rely partially on signature scanning when they scan files since each virus or variant can be identified with a sequence of bytes unique to it. Byte strings from scanned programs are taken and hashed. This hash is compared with known virus hashes in a database and if there is a match then there is a good chance that the application is a virus or has been infected. This of course ignores additional heuristics and scanning methods incorporated into anti-virus programs, but is still at a very basic level a key component of how they all work. This same methodology can be applied to developing game hacks or external modifications to applications in general since functions, structures, and so on also have unique byte patterns identifying them.



```
000A1000 ┌$ 55            PUSH EBP
000A1001 │. 8BEC          MOV EBP,ESP
000A1003 │. 51            PUSH ECX
000A1004 │. 6A 10         PUSH 10                                  ┌size = 10 (16.)
000A1006 │. FF15 B8200A00 CALL DWORD PTR DS:[<&MSVCR100.malloc>]   └malloc
000A100C │. 83C4 04       ADD ESP,4
000A100F │. 8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
000A1012 │. 8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
000A1015 │. C700 00000000 MOV DWORD PTR DS:[EAX],0
000A101B │. 8B4D FC       MOV ECX,DWORD PTR SS:[EBP-4]
000A101E │. D9EE          FLDZ
000A1020 │. D959 04       FSTP DWORD PTR DS:[ECX+4]
```

Shown above is part of a function that could serve as a signature. No other function in the application performs this unique set of instructions so assuming that this function does not change (the actual code within it is modified or things like new optimization settings or compilers being used) then it can always be identified with `\x55\x8B\xEC\x51\x6A\x10\...\xD9\x59\x04` regardless of any updates of patches to the application. However, this technique is not without its downsides: scanning an entire file or image is costly in terms of speed. Thus, it is a pretty bad idea to develop a hack that scans an image for a signature each time it is loaded since that can slow things down a lot. What I personally do is keep an external config file that holds signatures and the offset (RVA) into the image at which they're located. Then when a hack is loaded it can read in the config file and check that the signature exists where it's supposed to. If it doesn't then the hack will perform a scan on the whole image and write back into the config file where the new signature exists. This is only one way of doing it though so to each their own. Since the implementation is just searching for a substring, I feel that there's really no need to put one here. Important things to note though for developing signature scanners:

- Signature scanners should have some wildcard usage built in. Whether `EAX`, `EBX` and so on is used to hold a temporary value is irrelevant. For example, `MOV EAX, 123` as a byte string is `B823010000` and `MOV EBX, 123` is `BB23010000`. The important part of those instructions is the 123 immediate value, so the `B8` or `BB` byte is irrelevant. The signature can then be `\x??\x23\x01\x00\x00`. How `\x??` is treated is implementation dependent.
- Usually the most important parts of a signature are any references to other code, structures, local variables, etc. Getting a signature containing these will increase the chance of it being found. However, references to other code is a bit dangerous since relative distances can change between new versions of a target application.
- A signature, by definition, should be unique. Using `PUSH EBP` ; `MOV EBP, ESP` is a bad idea.