Software Assurance Tips

A product of the Software Assurance Tips Team[2]

Generated Friday 7th May, 2021

Andrew Perry

Monday 19th April, 2021

1 Heap Inspection

One of the first findings that I found when I began validating software assurance scans and something that intrigued me to learn more about was *Heap Inspections*.[1] In order to know how to resolve this issue, there are some things to understand about how heap is used in RAM in order to see how this vulnerability plays out and ways to fix it.

1.1 What is Heap?

When we declare a variable in an application, it allocates some memory in RAM. A stack is an array of memory that stores variables and implements a block allocation that is reserved in a LIFO (Last in, First out). When data are written, the stack trends downwards towards the heap. The heap is a location in memory that a program can use to store data in some various amount that may not be known until the program is running. Heap trends upwards when data is passed whereas a stack trends in a downward direction. When creating an object string that points to the stack memory, its value is stored in the heap. Heap is different than stack in that it is dynamically stored in memory which means that it can be stored and removed in any order. If the heap has data that has a fixed length memory block, it will overwrite expired or free values that were originally stored.[5, 3]

1.2 What is Heap Inspection?

In the case of a Heap Inspection, when information is stored in machine memory that is unencrypted and is using a `realloc()` function to increase the size of the block of allocated memory, an attacker can perform a memory dump that could compromise the system. This can cause all kinds of problems such as crash the program, but it can also expose sensitive information because it is not removed from memory properly. There are some ways to help mitigate this issue in C# or .NET applications.

1.3 Heap Inspection Prevention

Using the `String` object should not be used if the information contains sensitive information such as a password, credit card number, etc. It is important to not allow plain text passwords; hash them with a crytographically secure hasing algorithm. An important note is to invoke a Disposable method to use as a garbage collector to free computer memory when an object is no longer needed. One of the ways you can avoid a heap inspection and to protect sensitive data is by using a `SecureString`. Consider the code in Listing 1.[6, 4]

```
SecureString securePwd = new SecureString();
ConsoleKeyInfo key;
Console.Write("Enter password: ");
do {
        key = Console.ReadKey(true);...
```

Listing 1: SecureString Example

1.4 Conclusion

In conclusion, avoiding this issue can be a challenge, if not impossible. However, it's important that if this finding has been found in a static analysis scan, it should not be marked as a false positive, particularly if the code could ever be placed in a shared environment. Doing so may allow the code to go unnoticed while it continues to have unintended results and could allow sensitive information to be obtained by an adversary.

References

[1] CWE Content Team. “CWE-244: Improper Clearing of Heap Memory Before Release (’Heap Inspection’)”. In: (2020). URL: https://cwe.mitre.org/data/definitions/244.html.

[2] Jon Hood, ed. *SwATips*. https://www.SwATips.com/.

[3] Indika. “Difference Between Stack and Heap”. In: (2011). URL: https://www.differencebetween.com/difference-between-stack-and-vs-heap/.

[4] Chittaranjan Swain. “How to fix heap inspection vulnerability in c#”. In: (2019). URL: https://www.c-sharpcorner.com/forums/how-to-fix-heap-inspection-vulnerability-in-c-sharp.

[5] Unknown. “Stack vs Heap: Know the Difference”. In: (). URL: https://www.guru99.com/stack-vs-heap.html.

[6] Unknown. “To Avoid Heap Inspection”. In: (2015). URL: https://www.codeproject.com/Questions/1039781/To-Avoid-Heap-Inspection-I-Used-Securestring-Inste.