Johnny Zhong
CS373 – Defense Against The Dark Arts
Week4 - Hacking and How It's Done

Using a program in a way that the programmer did not original intend for.
2 Primary Methods:

- Manipulating software through vulnerability.
- Configuration related vulnerability. (weak password)

Bug Bounty Programs – get compensated by companies by finding vulnerabilities and informing them.

DMZ – Hard outer perimeter to prevent intrusion.
Current focus is on user vulnerability, via social engineering, phishing, etc.
Browser scripts are commonly used.

WinDbg – Debugger
Similar to GDB, freeze program.
All numbers in WinDbg is in hex – change for decimal via "format"

lm – list modules
lm f <search_regex> – list modules and find modules matching the search string
bp <progname!location> – break point
bl – breakpoint list
g - go
.hh <command> for help with that command.
da – decode ascii
du – decode Unicode
t – single step into
p – step over
g with address – go to address
pt – return out of function

Register Notes (that we care about):
EAX – Often stores return value of a function
EIP – memory instruction pointer
EBP/EDP – set start and end of stack frame

!teb – stack (thread)
!peb – heap (process)
!address – provide address and will inform you where the address is

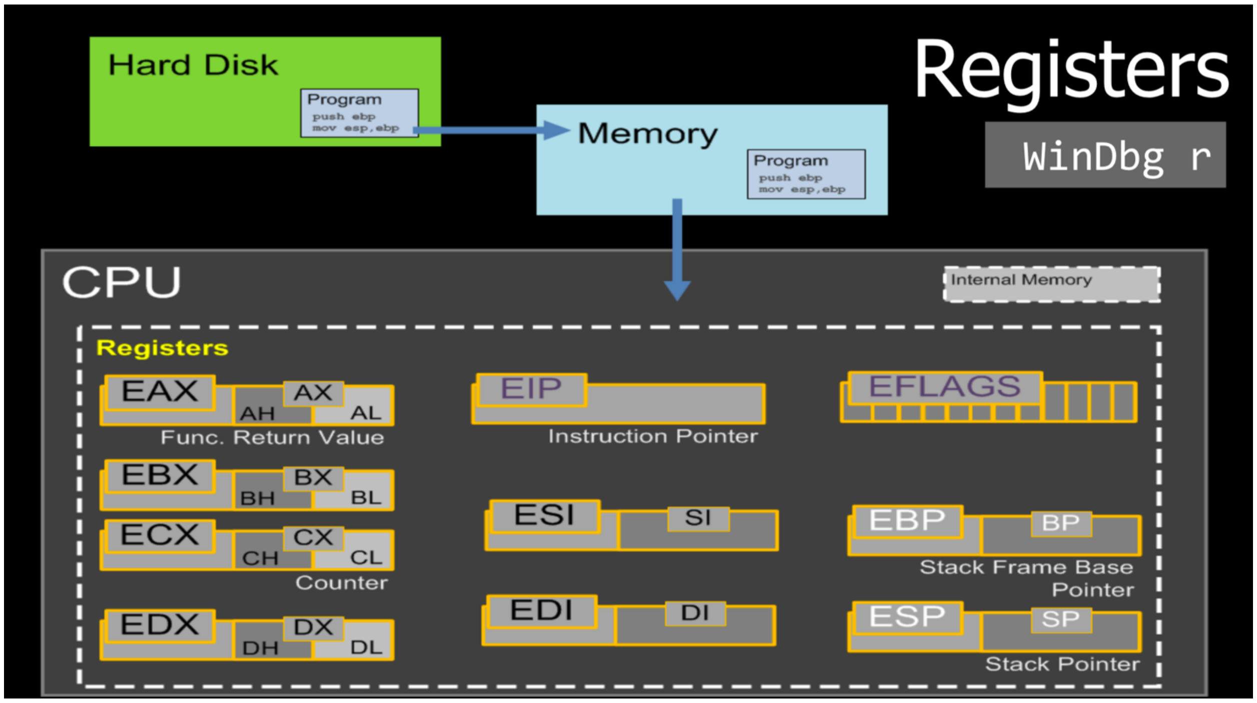
Memory Corruption – Accessing memory in invalid way which results in undefined behavior
Program expecting input from user.
Buffer Length Calculation Vulnerability

Exploitation – taking advantage of a vulnerability, via an exploit. (code, program, or input to a program)

Vulnerability Trigger –
invokes software bug to obtain control of the program, usually into an unstable state, usually a crash that the hacker can then use to their advantage
Payload –
aka "shell code". Action to be performed when control is obtained.

Metasploit – a method to combine exploits and payloads.

Stack Recap – method for program to go back to initially called functions.
Function params and return addresses and saved EBP are stored before EBP is moved to ESP.



Init:
EBP | Func Param | Ret Addr| ESP
Need to save local variables, so need to move the frame. First start by saving the EBP:
EBP | Func Param | Ret Addr| Saved EBP | ESP
Move the EBP to ESP, so we can start a new frame:
Func Param | Ret Addr| Saved EBP | ESP/EBP
Store local variables:

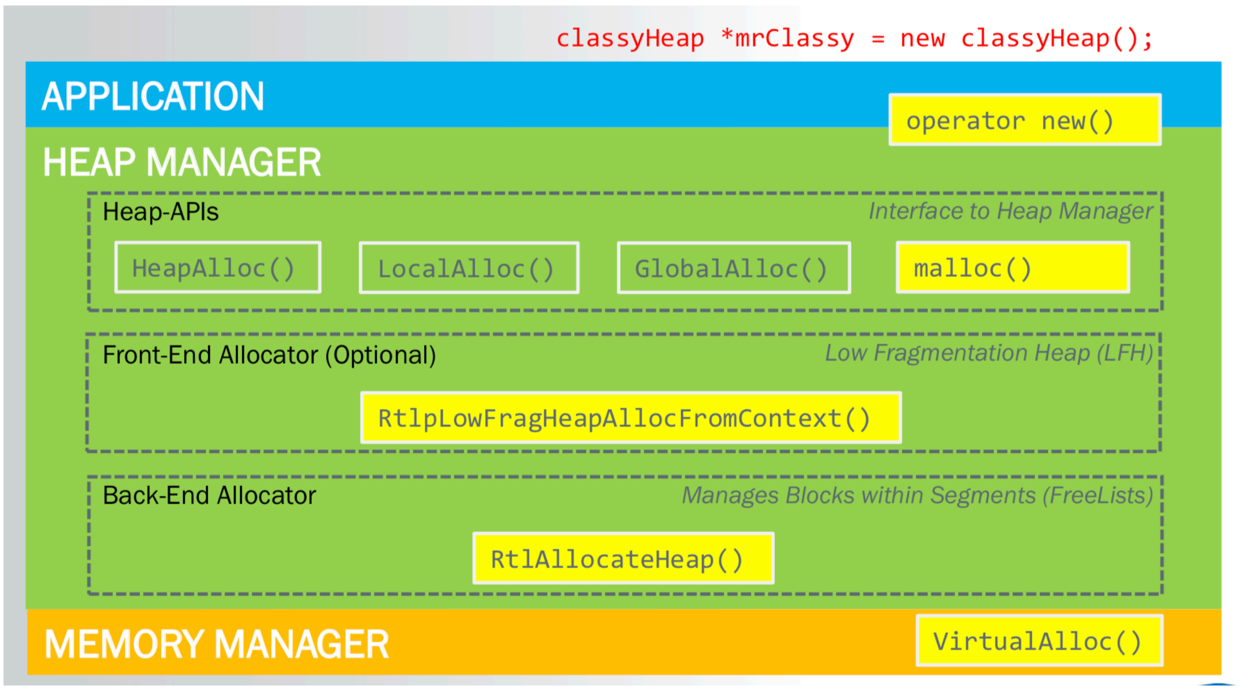
Func Param | Ret Addr| Saved EBP (pointer) | EBP | Local Vars | ESP

Overriding a program (Lab):
Because we can take advantage of stack overflow, we're able to overwrite the return address of the function and have the function execute code at a given address. That address may be code that we injected into memory.
The lab seemed to be a mix of memory exploration and execution. It relied on the user being able to parse registers and the disassembly information present in WinDBG.

Windows Debugging
Stomp/smash – interchangeable
Stack Canary

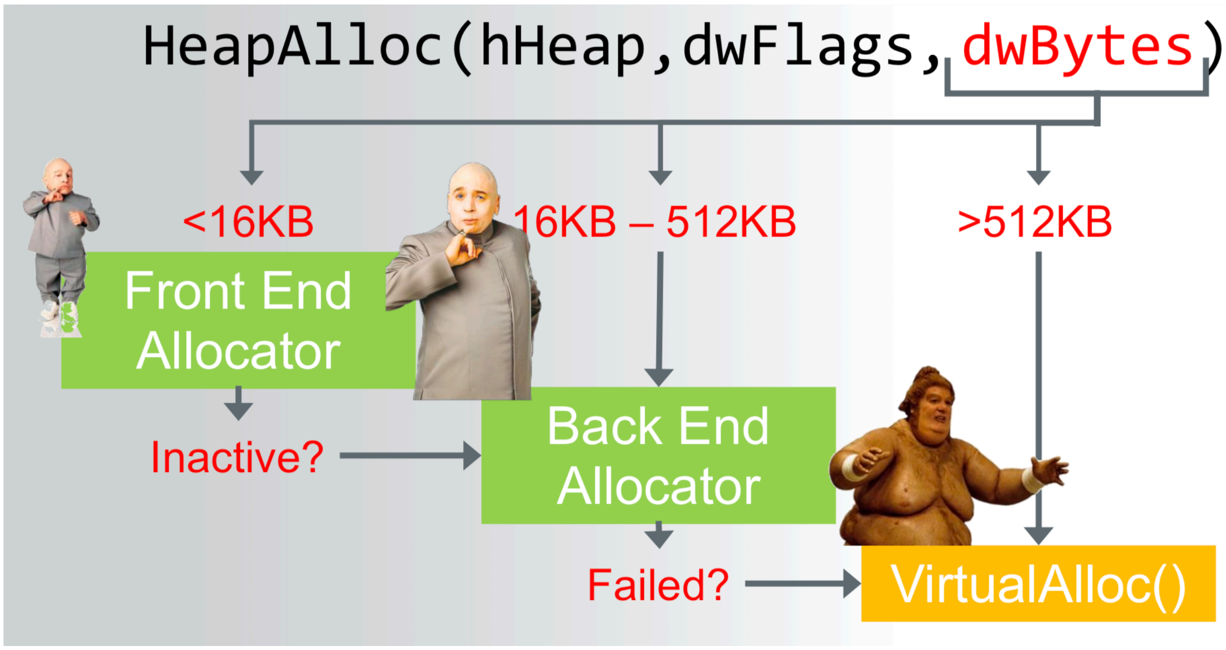
- an object that saves the return address of a function to determine if the function has altered/safe to return.
- Can mitigate this by running into an error elsewhere (other than return)

Use After Free
Using memory after the memory has been freed in browsers. Was popular as an exploit for a while.
Free object, put something in the same spot in memory.
Put shellcode in memory, then use the shellcode in memory.


classyHeap *mrClassy = new classyHeap();
APPLICATION
operator new()
HEAP MANAGER
Heap-APIs
Interface to Heap Manager
HeapAlloc()
LocalAlloc()
GlobalAlloc()
malloc()
Front-End Allocator (Optional)
Low Fragmentation Heap (LFH)
RtlpLowFragHeapAllocFromContext()
Back-End Allocator
Manages Blocks within Segments (FreeLists)
RtlAllocateHeap()
MEMORY MANAGER
VirtualAlloc()

Heap
Heap APIs – methods to access heap memory

FrontEnd Allocator – Accesses Low Fragmentation Heap (small size 16kb), enabled dynamically by program
BackEnd Allocator – Called by FrontEnd Allocator if available, calls VirtualAlloc() if too large.



Programs have access to different heap allocators.
In our given instance, IE requests 1mb, which is provided by VirtualAlloc.
All malloc methods create a bucket of a given size if there have been a repeated number of requests of the same size of memory.
Low Fragmentation heap is exploitable as it does not coalesce same size requests into a continuous block of memory upon detection that a bucket is appropriate.
This can be turned on via JS and can be used in JS.

Manipulate Heap in the Browser
Allocate to the heap via JS.
To activate low fragmentation heap, make string for allocation to attribute tag, loop string n number of times.

Create a class, instantiate it so that it goes to low frag memory, then delete the instantiated object, but keep the pointer. Insert a pointer another block of code to that pointer location, and that allows for us to inject code.

Debugging
Page Heap – special heap that gets enabled dynamically. Provides debugging information.
Automatically initiates all memory on heap with values.
Needs to be manually enabled for IE.

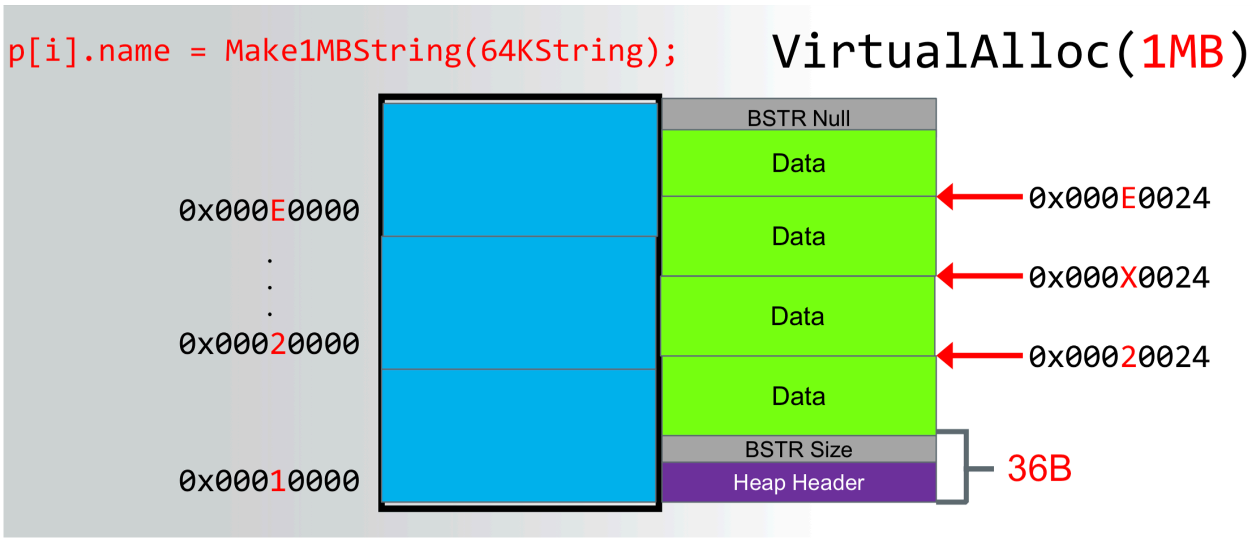
Crash information without page heap is not very useful.
After enabling, more information is provided by the crash.

!heap -p -a eax> Inspect page heap and at given address (in this case eax)

Need to know the size of the freed memory so we know how much to place.
Freed allocation cannot give size. Use debugger to track back to what called it. Stop the program that freed the memory block in order to know how much was freed and where.

Look for "HeapFree", insert breakpoint at call to HeapFree.
Look at the arguments provided to HeapFree. Takes 3 params.
In this case, it's the 1st thing that's provided to the stack.
dd esp - allows us to inspect the arguments
!heap -p -a poi(esp+8) provides information about the heap block.

Make allocations of the same size:
After determining size, disable page heap.
In JS, provide a block the same size into the given memory address. This will occupy the freed space.



Position Shellcode:
Given that we're using JS in a browser and ready to load shellcode, the exploit allows the user to reach outside of the browser and load anything they want into that freed space to

Make a string. Issue is that the string can live anywhere. Solution: make a huge allocation. Put shellcode in after making huge allocation. Since VirtualAlloc provides memory blocks in predictable chunks of 64kb (0xn0000), we can assume that our data is in a super high address. We find where the heap starts, make huge request of a known size, then we know where our data lives via arithmetic.