

Dynamic

Reverse

Engineering

NYU:Po1y

AARON PORTNOY



AARON PORTNOY



AARON PORTNOY



AARON PORTNOY

TippingPoint DV Labs



AARON PORTNOY

 **TippingPoint** DV Labs

WHAT WE'RE COVERING:

- ❑ Debugging with WinDBG
- ❑ Runtime Reversing Techniques
- ❑ Dynamic Memory
- ❑ Vulnerability Analysis

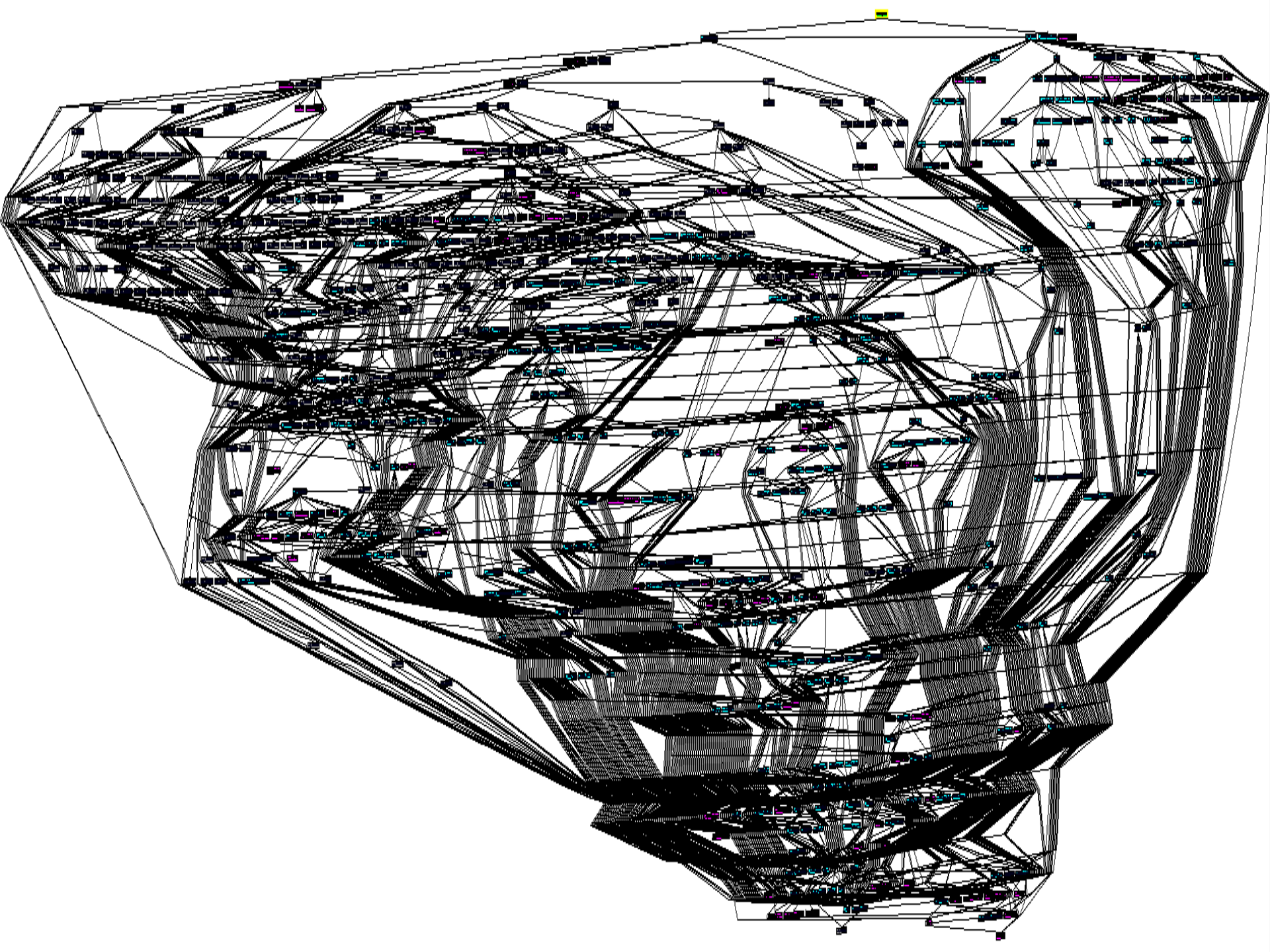
Reverse Engineering Goals

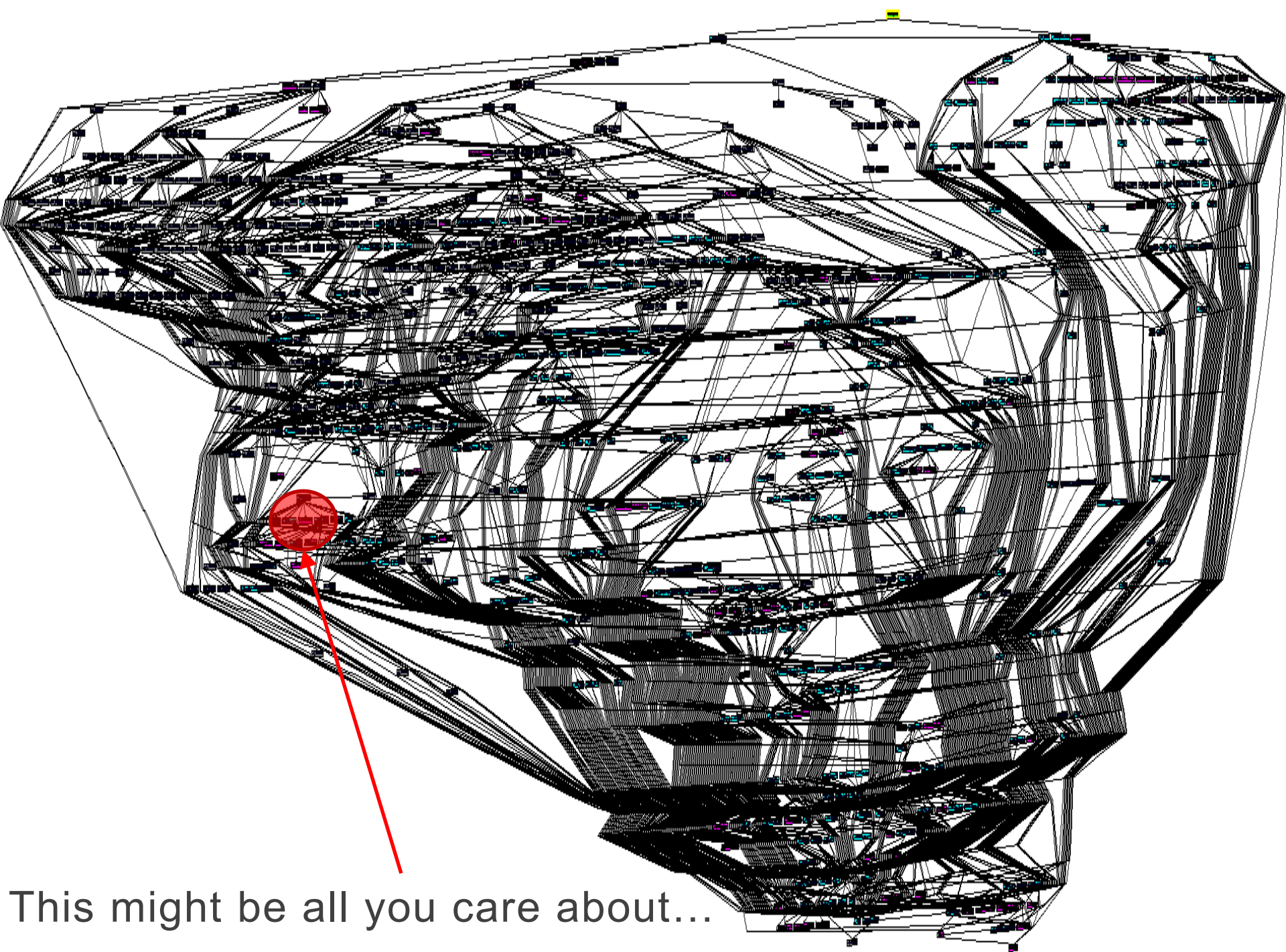
Why are you reversing? What do you hope to achieve?

- Vulnerability Discovery
- Vulnerability Analysis
- Software/DRM Cracking



All of these require just a **limited** understanding of the target as a whole





This might be all you care about...

STATIC REVERSING RECAP

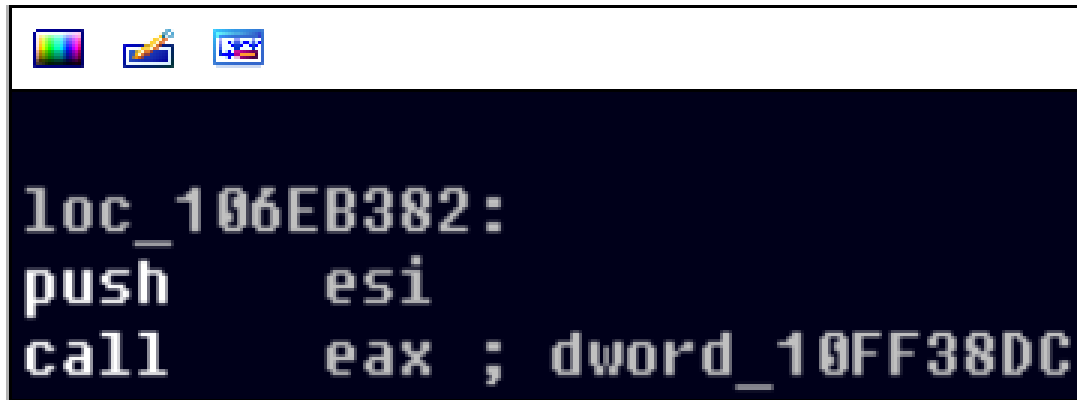
With static RE you're looking at a limited view (usually a single module at a time)

How many are familiar with **linkers** and **loaders**?

Lots of "stuff" goes on when you load a module into a process

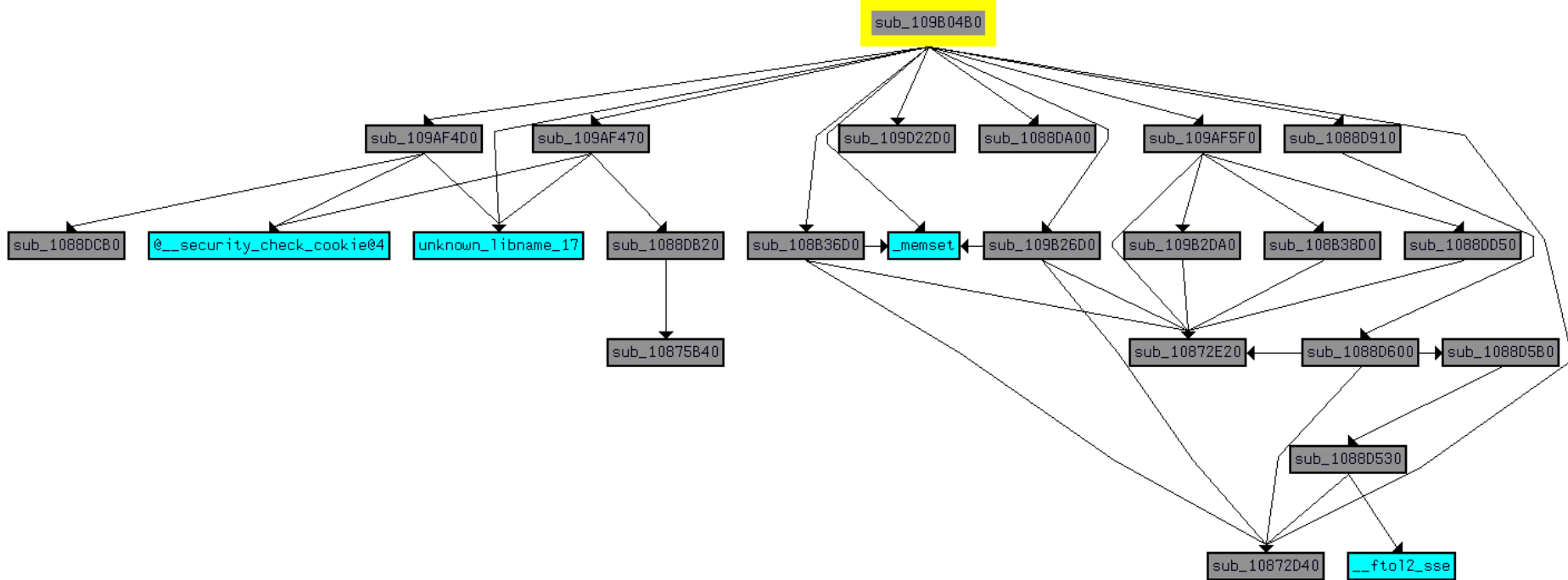
There is a lot of information missing when Reing statically...

Like dynamic call targets

A screenshot of a debugger window with a dark background and white text. The window title bar contains three icons: a color palette, a pencil, and a document. The assembly code displayed is:

```
loc_106EB382:  
push    esi  
call    eax ; dword_10FF38DC
```

.data:10FF38DC dword_10FF38DC dd ?



Static RE Challenges

Reversing anything reasonably complex statically is
painful

Pick your favorite lexer/parser/renderer for
example

(quicktime, shockwave, windows media player, flash, ...)

- ❑ Read input file into a buffer
- ❑ Split contiguous data into streams
 - ❑ or chunks, or sections, or ...
- ❑ Save all that info into application-specific data structures
- ❑ Pass of certain bits of that data to a lexer
 - ❑ Store tokens and corresponding data into other, perhaps unrelated app-specific data structures
- ❑ Parse the data (using metadata gathered at some point during all of this)
- ❑ Render media to the screen/file/output device

❑ Read input file into a buffer

Potential bugs

❑ Split contiguous data into streams

❑ or chunks, or sections, or ...

Potential bugs

❑ Save all that info into application-specific data structure

Potential bugs

❑ Pass of certain bits of that data to a lexer

Potential bugs

❑ Store tokens and corresponding data into other, perhaps unrelated app-specific data structures

❑ Parse the data (using metadata gathered at some point during all of

Potential bugs

❑ Render media to the screen/file/output device

Potential bugs

- ❑ Read input file into a buffer
- ❑ Split contiguous data into streams
 - ❑ or chunks, or sections, or ...
- ❑ Save all that info into application-specific data structures
- ❑ Pass of certain bits of that data to a lexer
 - ❑ Store tokens and corresponding data into other, perhaps unrelated app-specific data structures
- ❑ Parse the data (using metadata gathered at some point during all of this)
- ❑ Render media to the screen/file/output device

How would **you** write this code?

I bet your approach is different than your neighbor's...

- ❑ Read input file into a buffer
- ❑ Split contiguous data into streams
 - ❑ or chunks, or sections, or ...
- ❑ Save all that info into application-specific data structures
- ❑ Pass off certain bits of that data to a lexer
 - ❑ or parser, or compiler, or interpreter, or ...
 - ❑ perhaps unrelated application-specific data structures
- ❑ Parse the data (using metadata gathered at some point during all of this)
- ❑ Render media to the screen/file/output device

Good luck reversing an implementation
statically

How would **you** write this code?

I bet your approach is different than your neighbor's...

Static RE Challenges

Many things happen in the scope of a
process

...and you miss this information when reversing statically

- ❑ Exception handler registration and relationships

- ❑ Linking and relocations

 - ❑ Module dependencies

- ❑ Dynamic memory allocations (more on this later)

- ❑ Global variable values

- ❑ Out of band data that can influence the code

QUESTIONS?

DYNAMIC REVERSE ENGINEERING

Intro to Dynamic Reversing

Dynamic RE is focused on performing analysis during runtime

It introduces a new dimension: **state**

Dynamic RE is intended to supplement your static efforts
...by filling in missing information

More importantly, dynamic reversing lets you **validate** or **disprove** your suspicions

Intro to Dynamic Reversing

To dispel some preconceptions, you don't always need a debugger

...although it is the ideal tool to aid in the process

Debugger preferences, anyone?

Drilling Down

The first thing we want to do when reversing an application is figure out which **modules** it has that we care about

...and then we can peruse them statically

Module **dependencies** can help us here

If a module imports `ws2_32.dll`, its going to be doing “network stuff”

Dependency Walker - [dbgeng.dll]

File Edit View Options Profile Window Help

DBGENG.DLL

- MSVCRT.DLL
- DBGHELP.DLL
- VERSION.DLL
- ADVAPI32.DLL
- KERNEL32.DLL
- NTDLL.DLL
- WS2_32.DLL
- USER32.DLL
 - NTDLL.DLL
 - GDI32.DLL
 - KERNEL32.DLL
 - ADVAPI32.DLL
 - CFGMGR32.DLL
 - MSIMG32.DLL

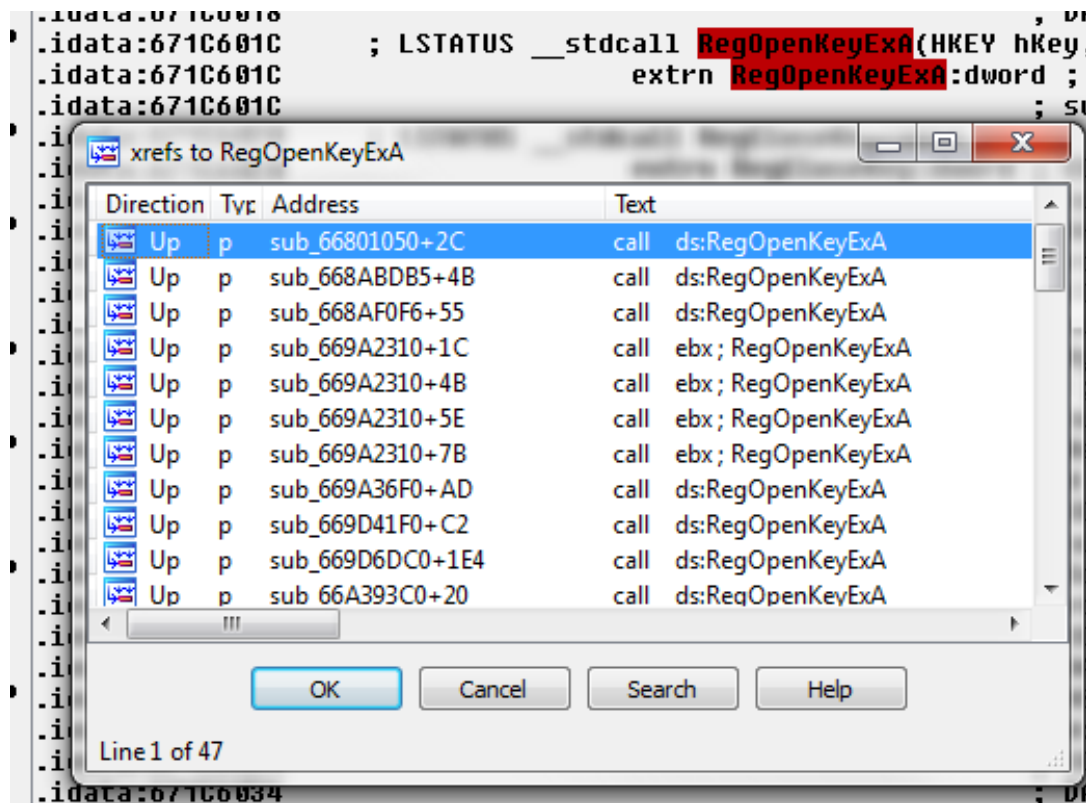
PI	Ordinal ^	Hint	Function	Entry Point
E	Ordinal ^	Hint	Function	Entry Point
<input checked="" type="checkbox"/>	1 (0x0001)	0 (0x0000)	DebugConnect	0x000E6F35
<input checked="" type="checkbox"/>	2 (0x0002)	1 (0x0001)	DebugConnectWide	0x000E62FA
<input checked="" type="checkbox"/>	3 (0x0003)	2 (0x0002)	DebugCreate	0x000E6344

Module	File Time Stamp	Link Time Stamp	File Size	Attr.	Link Checksum	Real Checksum
<input checked="" type="checkbox"/> ? GPSVC.DLL	Error opening file. The system cannot find the file specified (2).					
<input checked="" type="checkbox"/> ? IESHIMS.DLL	Error opening file. The system cannot find the file specified (2).					
<input checked="" type="checkbox"/> IEFRAE.DLL	05/26/2011 11:20a	03/08/2011 8:50a	9,702,400	A	0x00941037	0x00941037
<input type="checkbox"/> ADVAPI32.DLL	11/20/2010 8:18a	11/20/2010 7:54a	640,512	A	0x000A1449	0x000A1449
<input type="checkbox"/> API-MS-WIN-CORE-CONSOLE-L1-1-0.DLL	07/13/2009 9:03p	07/13/2009 9:04p	3,072	HA	0x0000FC51	0x0000FC51
<input type="checkbox"/> API-MS-WIN-CORE-DATETIME-L1-1-0.DLL	07/13/2009 9:03p	07/13/2009 9:04p	3,072	HA	0x000021E9	0x000021E9
<input type="checkbox"/> API-MS-WIN-CORE-DEBUG-L1-1-0.DLL	07/13/2009 9:03p	07/13/2009 9:04p	3,072	HA	0x0000BC69	0x0000BC69
<input type="checkbox"/> API-MS-WIN-CORE-DELAYLOAD-L1-1-0.DLL	07/13/2009 9:03p	07/13/2009 9:04p	3,072	HA	0x0000D654	0x0000D654
<input type="checkbox"/> API-MS-WIN-CORE-ERRORHANDLING-L1-1-0.DLL	07/13/2009 9:03p	07/13/2009 9:04p	3,072	HA	0x00005A00	0x00005A00
<input type="checkbox"/> API-MS-WIN-CORE-FIBERS-L1-1-0.DLL	07/13/2009 9:03p	07/13/2009 9:04p	3,072	HA	0x00009419	0x00009419
<input type="checkbox"/> API-MS-WIN-CORE-FILE-L1-1-0.DLL	07/13/2009 9:03p	07/13/2009 9:04p	5,120	HA	0x0000EB9B	0x0000EB9B
<input type="checkbox"/> API-MS-WIN-CORE-HANDLE-L1-1-0.DLL	07/13/2009 9:03p	07/13/2009 9:04p	3,072	HA	0x0000E589	0x0000E589
<input type="checkbox"/> API-MS-WIN-CORE-HFAP-L1-1-0.DLL	07/13/2009 9:03p	07/13/2009 9:04p	3,584	HA	0x00001FFA	0x00001FFA

Warning: At least one delay-load dependency module was not found.
Warning: At least one module has an unresolved import due to a missing export function in a delay-load dependent module.
For Help, press F1

Drilling Down

After we've found an interesting module, we can dig at it in IDA



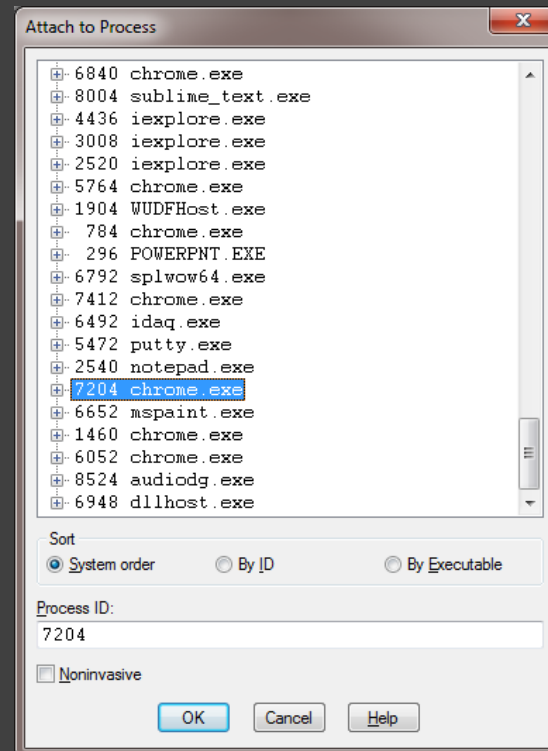
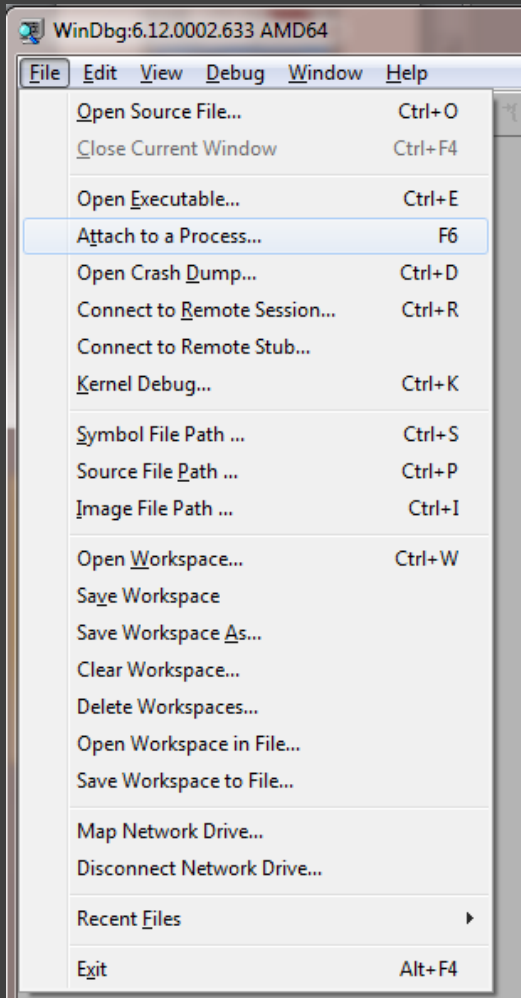
Drilling Down

Now, what would be **really** useful, is if we could determine if any of the code we find is actually **reached** during execution...

Enter our debugger...

Introduction to WinDBG

The first thing you'll want to do when using WinDBG is to attach to (or load) a process



Introduction to WinDBG

```
Pid 5984 - WinDbg.6.11.0001.404 X86
File Edit View Debug Window Help
[Icons]
Command
Microsoft (R) Windows Debugger Version 6.11.0001.404 X86
Copyright (c) Microsoft Corporation. All rights reserved.

*** wait with pending attach
Symbol search path is: *** Invalid ***
*****
* Symbol loading may be unreliable without a symbol search path. *
* Use .symfix to have the debugger choose a symbol path. *
* After setting your symbol path, use .reload to refresh symbol locations. *
*****
Executable search path is:
ModLoad: 00400000 0047d000 C:\Program Files (x86)\PuTTY\putty.exe
ModLoad: 779d0000 77b50000 C:\Windows\SysWOW64\ntdll.dll
ModLoad: 75530000 75640000 C:\Windows\system32\kernel32.dll
ModLoad: 752b0000 752f6000 C:\Windows\system32\KERNELBASE.dll
ModLoad: 751c0000 75260000 C:\Windows\system32\ADVAPI32.dll
ModLoad: 75680000 7572c000 C:\Windows\system32\msvcrt.dll
ModLoad: 757c0000 757d9000 C:\Windows\SysWOW64\sechost.dll
ModLoad: 75d10000 75e00000 C:\Windows\system32\RPCRT4.dll
ModLoad: 750c0000 75120000 C:\Windows\system32\Spapi.dll
ModLoad: 750b0000 750bc000 C:\Windows\system32\CRYPTBASE.dll
ModLoad: 72d90000 72f2e000 C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f2b2\COMCTL32.dll
ModLoad: 76170000 76200000 C:\Windows\system32\GDI32.dll
ModLoad: 75870000 75970000 C:\Windows\system32\USER32.dll
ModLoad: 751b0000 751ba000 C:\Windows\system32\LPK.dll
ModLoad: 76200000 7629d000 C:\Windows\system32\USP10.dll
ModLoad: 76460000 764b7000 C:\Windows\system32\SHLWAPI.dll
ModLoad: 75130000 751ab000 C:\Windows\system32\comdlg32.dll
ModLoad: 764c0000 7710a000 C:\Windows\system32\SHELL32.dll
ModLoad: 762b0000 76310000 C:\Windows\system32\INM32.dll
ModLoad: 75e00000 75ecc000 C:\Windows\system32\MGCTF.dll
ModLoad: 76010000 7616c000 C:\Windows\system32\ole32.dll
ModLoad: 73280000 732b2000 C:\Windows\system32\WINMM.dll
ModLoad: 73200000 73251000 C:\Windows\system32\WINSPOOL.DRV
ModLoad: 75640000 75675000 C:\Windows\system32\ws2_32.dll
ModLoad: 75970000 75976000 C:\Windows\system32\NGI.dll
ModLoad: 748e0000 74960000 C:\Windows\system32\uxtheme.dll
ModLoad: 10100000 1010e000 C:\Program Files\Logitech\SetPoint\x86\lgscroll1.dll
ModLoad: 733f0000 7348b000 C:\Windows\WinSxS\x86_microsoft.vc80.crt_lfc8b3b9a1e18e3b_8.0.50727.6195_none_d09154e044272b9a\MSVCR80.dll
ModLoad: 74b90000 74bb1000 C:\Windows\system32\ntmarta.dll
ModLoad: 75260000 752a5000 C:\Windows\system32\WLDAP32.dll
ModLoad: 734c0000 734d3000 C:\Windows\system32\dmapi.dll
ModLoad: 75730000 757b3000 C:\Windows\system32\CLBCatQ.DLL
ModLoad: 75980000 75a0f000 C:\Windows\system32\OLEAUT32.dll
ModLoad: 61800000 61884000 C:\Windows\system32\hhctrl.ocx
ModLoad: 748d0000 748d8000 C:\Windows\system32\secur32.dll
ModLoad: 10000000 10012000 C:\Program Files\LENOVO\HOTKEY\hkvolvekey.DLL
(1760.1d38): Break instruction exception - code 80000003 (first chance)
eax=7efda000 ebx=00000000 ecx=00000000 edx=77a6f7ea esi=00000000 edi=00000000
eip=779e000c esp=0268ff5c ebp=0268ff88 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Windows\SysWOW64\ntdll.dll -
ntdll!DbgBreakPoint:
779e000c cc             int     3

0:001>
```

Introduction to WinDBG

```
(1760.1d38): Break instruction exception - code 80000003 (first chance)
eax=7efda000 ebx=00000000 ecx=00000000 edx=77a6f7ea esi=00000000 edi=00000000
eip=779e000c esp=0268ff5c ebp=0268ff88 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Windows\SysWOW64\ntdll.dll -
ntdll!DbgBreakPoint:
779e000c cc                int     3
```

Current register values

Introduction to WinDBG

```
(1760.1d38): Break instruction exception - code 80000003 (first chance)
eax=7efda000 ebx=00000000 ecx=00000000 edx=77a6f7ea esi=00000000 edi=00000000
eip=779e000c esp=0268ff5c ebp=0268ff88 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Windows\SysWOW64\ntdll.dll -
ntdll!DbgBreakPoint:
779e000c cc                int     3
```

Current flags value

Introduction to WinDBG

```
(1760.1d38): Break instruction exception - code 80000003 (first chance)
eax=7efda000 ebx=00000000 ecx=00000000 edx=77a6f7ea esi=00000000 edi=00000000
eip=779e000c esp=0268ff5c ebp=0268ff88 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Windows\SysWOW64\ntdll.dll -
ntdll!DbgBreakPoint:
779e000c cc                int     3
```

Current address (eip) and instruction

Introduction to WinDBG

Some basic commands to get you started...

- ❑ `g` : the go command, lets the process run
 - ❑ You can interrupt a process by Ctrl+Break
 - ❑ Or via the menu (Debug->Break)

- ❑ `r` : shows the current register contents and disassembly at the current eip

- ❑ `.hh` : brings up the WinDBG help (awesome resource)
 - ❑ `.hh g`

Introduction to WinDBG

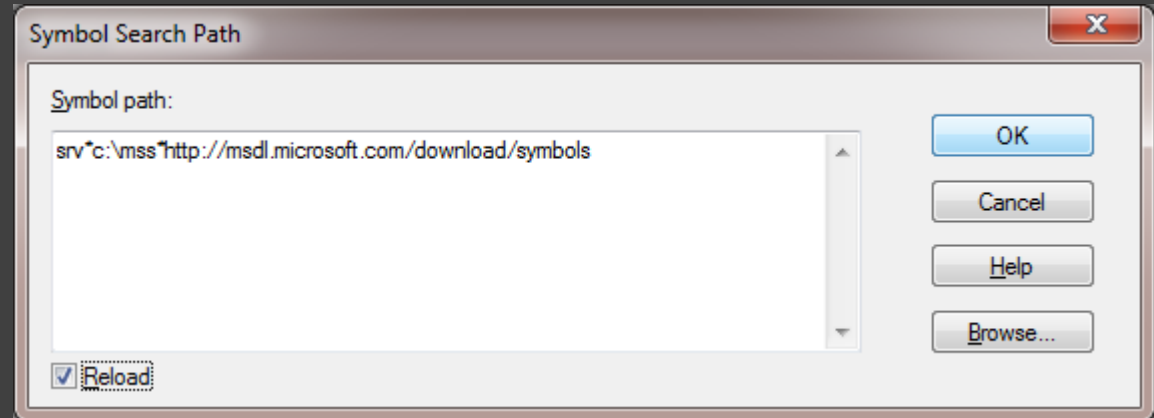
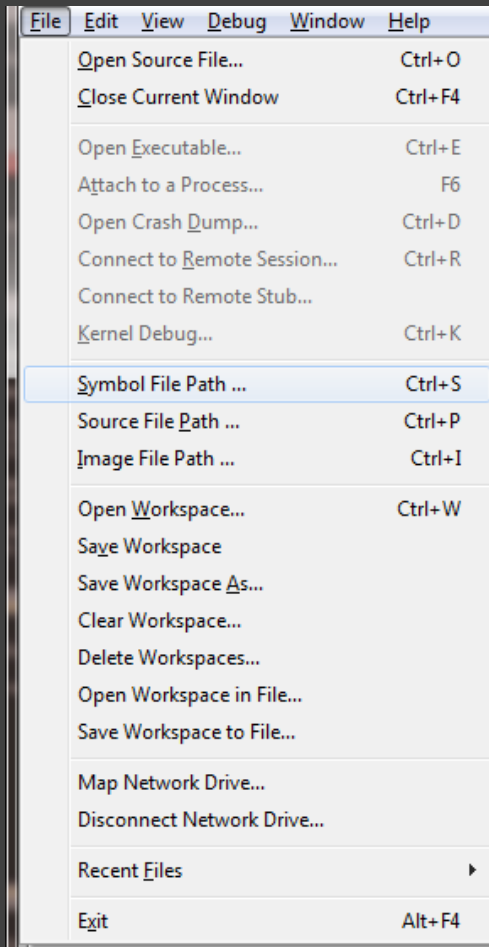
Some basic commands to get you started...

- ❑ t : trace steps one instruction
 - ❑ and will step **into** functions
 - ❑ Also useful is the tc command: step until call

- ❑ p : single-steps one instruction
 - ❑ and will step **over** functions
 - ❑ Also useful is the pt command: step until return

Introduction to WinDBG

Symbol support in WinDBG is generally very good



```
0:001> .reload
Reloading current modules
.....
0:001> .symfix
```

Microsoft/Apple/Chrome/Citrix all have symbol servers

Introduction to WinDBG

S Y M B O L S

```
0:000> k
ChildEBP RetAddr
WARNING: Stack unwind information not available. Following frames may be wrong.
0018f30c 76188061 ntdll!memcpy
0018f36c 76188326 GDI32!GetTextAlign+0x8c
0018f390 75891751 GDI32!GetTextMetricsW+0x74
0018f400 7589150f USER32!DrawTextExW+0x2b3
0018f4a4 758914bc USER32!DrawTextExW+0x71
0018f4c8 748f8b07 USER32!DrawTextExW+0x1e
0018f510 748f8a86 uxtheme!GetThemeTextExtent+0x109
0018f550 748f93b1 uxtheme!GetThemeTextExtent+0x88
0018f59c 748f932b uxtheme!GetThemeTextExtent+0x9b3
0018f67c 748f828d uxtheme!GetThemeTextExtent+0x92d
0018f760 748f885f uxtheme!GetThemeBool+0x1c3c
0018f784 748f0b0d uxtheme!GetThemeBool+0x220e
0018f7d8 748f22c3 uxtheme!GetThemeBool+0x10b0d
0018f7f4 7588d09b uxtheme!Ordinal49+0x271
0018f83c 75897953 USER32!WCSToMBCEx+0x17e
0018f854 758afaa5 USER32!CallWindowProcA+0x24
0018f89c 7589afac USER32!GetCursor+0x3c5
0018f8bc 758862fa USER32!DrawTextExA+0xd4
0018f8e8 75887316 USER32!MapfnScSendMessage+0x332
0018f960 75886de8 USER32!GetDC+0x52
WARNING: Whitespace at end of path element
0:000> .reload
Reloading current modules
.....
0:000> k
ChildEBP RetAddr
0018f30c 76188061 ntdll!memcpy
0018f36c 76188326 GDI32!bGetTextMetricsWInternal+0x32
0018f390 75891751 GDI32!GetTextMetricsW+0x74
0018f400 7589150f USER32!DT_InitDrawTextInfo+0x1ba
0018f4a4 758914bc USER32!DrawTextExWorker+0x72
0018f4c8 748f8b07 USER32!DrawTextExW+0x1e
0018f510 748f8a86 uxtheme!CTextDraw::GetTextExtent+0xbe
0018f550 748f93b1 uxtheme!GetThemeTextExtent+0x69
0018f59c 748f932b uxtheme!_GetNcCaptionTextSize+0x62
0018f67c 748f828d uxtheme!CThemeWnd::GetNcWindowMetrics+0xd8
0018f760 748f885f uxtheme!CThemeWnd::NcPaint+0x61
0018f784 748f0b0d uxtheme!OnDwpNcPaint+0x61
0018f7d8 748f22c3 uxtheme!_ThemeDefWindowProc+0x13c
0018f7f4 7588d09b uxtheme!_ThemeDefWindowProcA+0x18
0018f83c 75897953 USER32!DefWindowProcA+0x68
0018f854 758afaa5 USER32!DefWindowProcWorker+0x27
0018f89c 7589afac USER32!DefDlgProcWorker+0x802
0018f8bc 758862fa USER32!DefDlgProcA+0x29
0018f8e8 75887316 USER32!InternalCallWinProc+0x23
0018f960 75886de8 USER32!UserCallWinProcCheckWow+0xd8
```

Introduction to WinDBG

❑ x : examine symbols

```
0:000> x oleaut32!*funcand*
759b08c2 OLEAUT32!GEN_DTINFO::SetFuncAndParamNames = <no type information>
759f0701 OLEAUT32!STL_TYPEINFO::SetFuncAndParamNames = <no type information>
759b0910 OLEAUT32!TYPE_DATA::SetFuncAndParamNames = <no type information>
759b094d OLEAUT32!TYPE_DATA::SetFuncAndParamNamesOfHdefn = <no type information>
7599af04 OLEAUT32!CTypeInfo2::SetFuncAndParamNames = <no type information>
```

❑ ln : list nearest symbol

```
0:000> ln @eip
(779f2340) ntdll!memcpy | (779f267e) ntdll! ?? ::FNODOBFM::`string'
Exact matches:
    ntdll!memcpy = <no type information>
```

❑ dt : display type

```
0:000> dt _RTL_CRITICAL_SECTION
ntdll!_RTL_CRITICAL_SECTION
+0x000 DebugInfo      : Ptr32 _RTL_CRITICAL_SECTION_DEBUG
+0x004 LockCount      : Int4B
+0x008 RecursionCount : Int4B
+0x00c OwningThread   : Ptr32 Void
+0x010 LockSemaphore  : Ptr32 Void
+0x014 SpinCount      : Uint4B
```

Introduction to WinDBG

- ❑ `!m` : shows listed modules
 - ❑ You can search with `!mm`

```
0:001> !m m*ole*
start      end          module name
75980000 75a0f000  OLEAUT32   (deferred)
76010000 7616c000  ole32      (deferred)
```

- ❑ `!lmi` : gives you more detailed info

```
0:001> !lmi oleaut32
Loaded Module Info: [oleaut32]
  Module: OLEAUT32
  Base Address: 75980000
  Image Name: C:\Windows\syswow64\OLEAUT32.dll
  Machine Type: 332 (I386)
  Time Stamp: 4d673de9 Fri Feb 25 00:28:09 2011
  Size: 8f000
  CheckSum: 8c067
  Characteristics: 2102
  Debug Data Dirs:
    Type      Size      VA      Pointer
    CODEVIEW  25, 84678, 83a78 RSDS - GUID:
    Age: 2, Pdb: oleaut32.pdb
    CLSID    4, 84674, 83a74 [Data not mapped]
  Symbol Type: DEFERRED - No error - symbol load deferred
  Load Report: no symbols loaded
```

```
0:001> !m
start      end          module name
00400000 0047d000  putty      (deferred)
10000000 10012000  hkvolkey   (deferred)
10100000 1010e000  lgscroll   (deferred)
61800000 61884000  hhctrl     (deferred)
72d90000 72f2e000  COMCTL32   (deferred)
73200000 73251000  WINSPOOL   (deferred)
73280000 732b2000  WINMM      (deferred)
733f0000 7348b000  MSVCR80    (deferred)
734c0000 734d3000  dwmapi     (deferred)
748d0000 748d8000  secur32    (deferred)
748e0000 74960000  uxtheme    (deferred)
74b90000 74bb1000  ntmarta    (deferred)
750b0000 750bc000  CRYPTBASE  (deferred)
750c0000 75120000  SspiCli    (deferred)
75130000 751ab000  comdlg32   (deferred)
751b0000 751ba000  LPK        (deferred)
751c0000 75260000  ADVAPI32   (deferred)
75260000 752a5000  WLDAP32    (deferred)
752b0000 752f6000  KERNELBASE (deferred)
75530000 75640000  kernel32   (export symbols)
75640000 75675000  ws2_32     (deferred)
75680000 7572c000  msvcrt     (deferred)
75730000 757b3000  CLBCatQ    (deferred)
757c0000 757d9000  sechost    (deferred)
75870000 75970000  USER32    (deferred)
75970000 75976000  NSI        (deferred)
75980000 75a0f000  OLEAUT32   (deferred)
75d10000 75e00000  RPCRT4     (deferred)
75e00000 75ecc000  MSCTF      (deferred)
76010000 7616c000  ole32      (deferred)
76170000 76200000  GDI32      (deferred)
76200000 7629d000  USP10      (deferred)
762b0000 76310000  IMM32      (deferred)
76460000 764b7000  SHLWAPI    (deferred)
764c0000 7710a000  SHELL32    (deferred)
779d0000 77b50000  ntdll      (export symbols)
```

Introduction to WinDBG

Avoid STDs

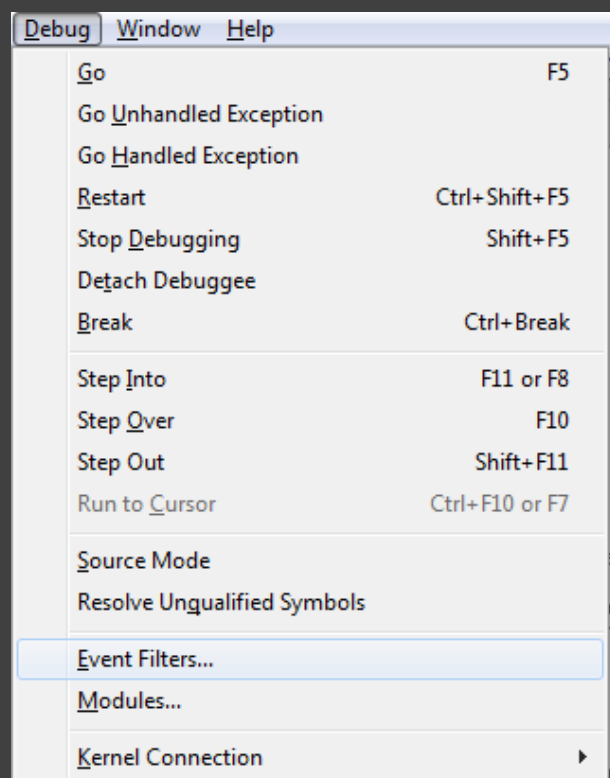
...stdlibs, that is (standard libraries)

```
0:001> lmf
start      end      module name
00000000`00400000 00000000`0047d000  putty     C:\Program Files (x86)\PuTTY\putty.exe
00000000`10000000 00000000`10012000  hkvolkey  hkvolkey.dll
00000000`10100000 00000000`1010e000  lgscroll  lgscroll.dll
00000000`61800000 00000000`61884000  HHCTRL    HHCTRL.OCX
00000000`72d90000 00000000`72f2e000  COMCTL32  COMCTL32.dll
00000000`73200000 00000000`73251000  WINSPOOL  WINSPOOL.DRV
00000000`73280000 00000000`732b2000  WINMM     WINMM.dll
00000000`733f0000 00000000`7348b000  MSVCR80   MSVCR80.dll
00000000`734c0000 00000000`734d3000  dwmapi    dwmapi.dll
00000000`748d0000 00000000`748d8000  Secur32   Secur32.dll
00000000`748e0000 00000000`74960000  UxTheme   UxTheme.dll
00000000`74b90000 00000000`74bb1000  NTMARTA   NTMARTA.dll
00000000`74c00000 00000000`74c08000  wow64cpu  C:\Windows\SYSTEM32\wow64cpu.dll
00000000`74c10000 00000000`74c6c000  wow64win  C:\Windows\SYSTEM32\wow64win.dll
00000000`74c70000 00000000`74caf000  wow64     C:\Windows\SYSTEM32\wow64.dll
00000000`750b0000 00000000`750bc000  CRYPTBASE CRYPTBASE.dll
00000000`750c0000 00000000`75120000  SspiCli   SspiCli.dll
00000000`75130000 00000000`751ab000  COMDLG32  COMDLG32.dll
00000000`751b0000 00000000`751ba000  LPK       LPK.dll
00000000`751c0000 00000000`75260000  ADVAPI32  ADVAPI32.dll
00000000`75260000 00000000`752a5000  WLDAP32   WLDAP32.dll
00000000`752b0000 00000000`752f6000  KERNELBA SE KERNELBASE.dll
00000000`75530000 00000000`75640000  KERNEL32  KERNEL32.dll
00000000`75640000 00000000`75675000  WS2_32    WS2_32.dll
00000000`75680000 00000000`7572c000  msvcrt    msvcrt.dll
00000000`75730000 00000000`757b3000  CLBCatQ   CLBCatQ.DLL
00000000`757c0000 00000000`757d9000  SECHOST   SECHOST.dll
00000000`75870000 00000000`75970000  USER32    USER32.dll
00000000`75970000 00000000`75976000  NSI       NSI.dll
00000000`75980000 00000000`75a0f000  OLEAUT32  OLEAUT32.dll
00000000`75d10000 00000000`75e00000  RPCRT4    RPCRT4.dll
00000000`75e00000 00000000`75ecc000  MSCTF     MSCTF.dll
00000000`76010000 00000000`7616c000  ole32     ole32.dll
00000000`76170000 00000000`76200000  GDI32     GDI32.dll
00000000`76200000 00000000`7629d000  USP10     USP10.dll
00000000`762b0000 00000000`76310000  IMM32     IMM32.dll
00000000`76460000 00000000`764b7000  SHLWAPI   SHLWAPI.dll
00000000`764c0000 00000000`7710a000  SHELL32   SHELL32.dll
00000000`777f0000 00000000`77999000  ntdll     C:\Windows\SYSTEM32\ntdll.dll
00000000`779d0000 00000000`77b50000  ntdll_779d0000 ntdll.dll
```

Introduction to WinDBG

□ `sx` : set exceptions

- `sxe` `ld:modulename`
- Also available via GUI



```
0:001> sx
  ct - Create thread - ignore
  et - Exit thread - ignore
 cpr - Create process - ignore
 epr - Exit process - break
  ld - Load module - output
  ud - Unload module - ignore
 ser - System error - ignore
 ibp - Initial breakpoint - break
 iml - Initial module load - ignore
 out - Debuggee output - output

  av - Access violation - break - not handled
 asrt - Assertion failure - break - not handled
 aph - Application hang - break - not handled
 bpe - Break instruction exception - break
 bpec - Break instruction exception continue - handled
 eh - C++ EH exception - second-chance break - not handled
 clr - CLR exception - second-chance break - not handled
 clrn - CLR notification exception - second-chance break - handled
 cce - Control-Break exception - break
  cc - Control-Break exception continue - handled
 cce - Control-C exception - break
  cc - Control-C exception continue - handled
 dm - Data misaligned - break - not handled
 dbce - Debugger command exception - ignore - handled
 gp - Guard page violation - break - not handled
 ii - Illegal instruction - second-chance break - not handled
 ip - In-page I/O error - break - not handled
 dz - Integer divide-by-zero - break - not handled
 iov - Integer overflow - break - not handled
 ch - Invalid handle - break
 hc - Invalid handle continue - not handled
 lsq - Invalid lock sequence - break - not handled
 isc - Invalid system call - break - not handled
 3c - Port disconnected - second-chance break - not handled
 svh - Service hang - break - not handled
 sse - Single step exception - break
 ssec - Single step exception continue - handled
 sbo - Stack buffer overflow - break - not handled
 sov - Stack overflow - break - not handled
 vs - Verifier stop - break - not handled
 vcpp - Visual C++ exception - ignore - handled
 wkd - Wake debugger - break - not handled
 wob - WOW64 breakpoint - break - handled
 wos - WOW64 single step exception - break - handled

 * - Other exception - second-chance break - not handled
0:001> sxe ld:aaronsdll
```

Introduction to WinDBG

Some usage notes...

To prevent symbol lookup, prefix registers with @
e.g. `r @eax`

You can reference addresses multiple ways...

- Address (explicit base-16 number)
- Module name (e.g. `OLEAUT32`)
- Symbol within a module (e.g. `OLEAUT32!DispCallFunc`)
- Offsets (e.g. `OLEAUT32!DispCallFunc+0x202`)

Introduction to WinDBG

Viewing Memory

- ❑ dd target : dump double words
- ❑ dw target : dump words
- ❑ db target : dump bytes
- ❑ dc target : dump data and ASCII printables
- ❑ da target : dump ASCII string
- ❑ du target : dump UNICODE string

```
0:000> dc @esp
0018acf8 00000014 0225be68 0225be68 0018fdbc  ....h%.h%.
0018ad08 0041d58d 0018ad38 00000014 00000000  ..A.8.
0018ad18 004454d0 0225be68 00000000 0018ad38  .TD.h%.8.
0018ad28 00000014 75895f74 00000000 00000000  ....t_u.
0018ad38 2d485353 2d302e32 6e65704f 5f485353  SSH-2.0-OpenSSH_
0018ad48 0a332e34 00000000 00000000 00000000  4.3.
0018ad58 00000000 00000000 00000000 00000000  .....
0018ad68 00000000 00000000 00000000 00000000  .....
0:000> da 0018ad38
0018ad38  "SSH-2.0-OpenSSH_4.3."
```

Introduction to WinDBG

Call Stacks

- ❑ k : view call stack
 - ❑ Variations on this can give you arguments (kv)
 - ❑ Sometimes this can fail if there are non-ebp based frames

```
0:000> k
ChildEBP RetAddr
0018f30c 76188061 ntdll!memcpy
0018f36c 76188326 GDI32!bGetTextMetricsWInternal+0x32
0018f390 75891751 GDI32!GetTextMetricsW+0x74
0018f400 7589150f USER32!DT_InitDrawTextInfo+0x1ba
0018f4a4 758914bc USER32!DrawTextExWorker+0x72
0018f4c8 748f8b07 USER32!DrawTextExW+0x1e
0018f510 748f8a86 uxtheme!CTextDraw::GetTextExtent+0xbe
0018f550 748f93b1 uxtheme!GetThemeTextExtent+0x69
0018f59c 748f932b uxtheme!_GetNcCaptionTextSize+0x62
0018f67c 748f828d uxtheme!CThemeWnd::GetNcWindowMetrics+0xd8
0018f760 748f885f uxtheme!CThemeWnd::NcPaint+0x61
0018f784 748f0b0d uxtheme!OnDwpNcPaint+0x61
0018f7d8 748f22c3 uxtheme!_ThemeDefWindowProc+0x13c
0018f7f4 7588d09b uxtheme!ThemeDefWindowProcA+0x18
0018f83c 75897953 USER32!DefWindowProcA+0x68
0018f854 758afaa5 USER32!DefWindowProcWorker+0x27
0018f89c 7589afac USER32!DefDlgProcWorker+0x802
0018f8bc 758862fa USER32!DefDlgProcA+0x29
0018f8e8 75887316 USER32!InternalCallWinProc+0x23
0018f960 75886de8 USER32!UserCallWinProcCheckWow+0xd8
```

Introduction to WinDBG

Breakpoints—the single most useful tool at your disposal

There are two main types of breakpoints:

Software

bp target

bm target

```
0:000> bp sprintf
0:000> bp 0x12345678
0:000> bm /a oleaut32!*dispcall*
           3: 75993dcf @!"OLEAUT32!DispCallFunc"
0:000> ba r1 0x22222222
```

Hardware

ba r1 target

ba e1 target

ba w4 target

Introduction to WinDBG

Breakpoints—the single most useful tool at your disposal

And then there are conditionals:

❑ bp address “expression”

```
0:000> bp NTDLL+0x300 "if (@eax < 0x20) {} .else {g}"
```

Introduction to WinDBG

Breakpoints—the single most useful tool at your disposal

Various operations on breakpoints:

- ❑ bc : clear a breakpoint (or all of them with *)
- ❑ bd : disable “
- ❑ be : enable “
- ❑ bl : list breakpoints

```
0:000> bl
0 e 77aa53c3      0001 (0001)  0:**** ntdll!sprintf
1 e 12345678      0001 (0001)  0:****
2 e 22222222 r 1 0001 (0001)  0:****
3 e 75993dcf      0001 (0001)  0:**** OLEAUT32!DispCallFunc
0:000> bc 1
0:000> bd 3
0:000> bl
0 e 77aa53c3      0001 (0001)  0:**** ntdll!sprintf
2 e 22222222 r 1 0001 (0001)  0:****
3 d 75993dcf      0001 (0001)  0:**** OLEAUT32!DispCallFunc
```

Breakpoints on API Functions

You can glean runtime information by setting breakpoints on interesting functions

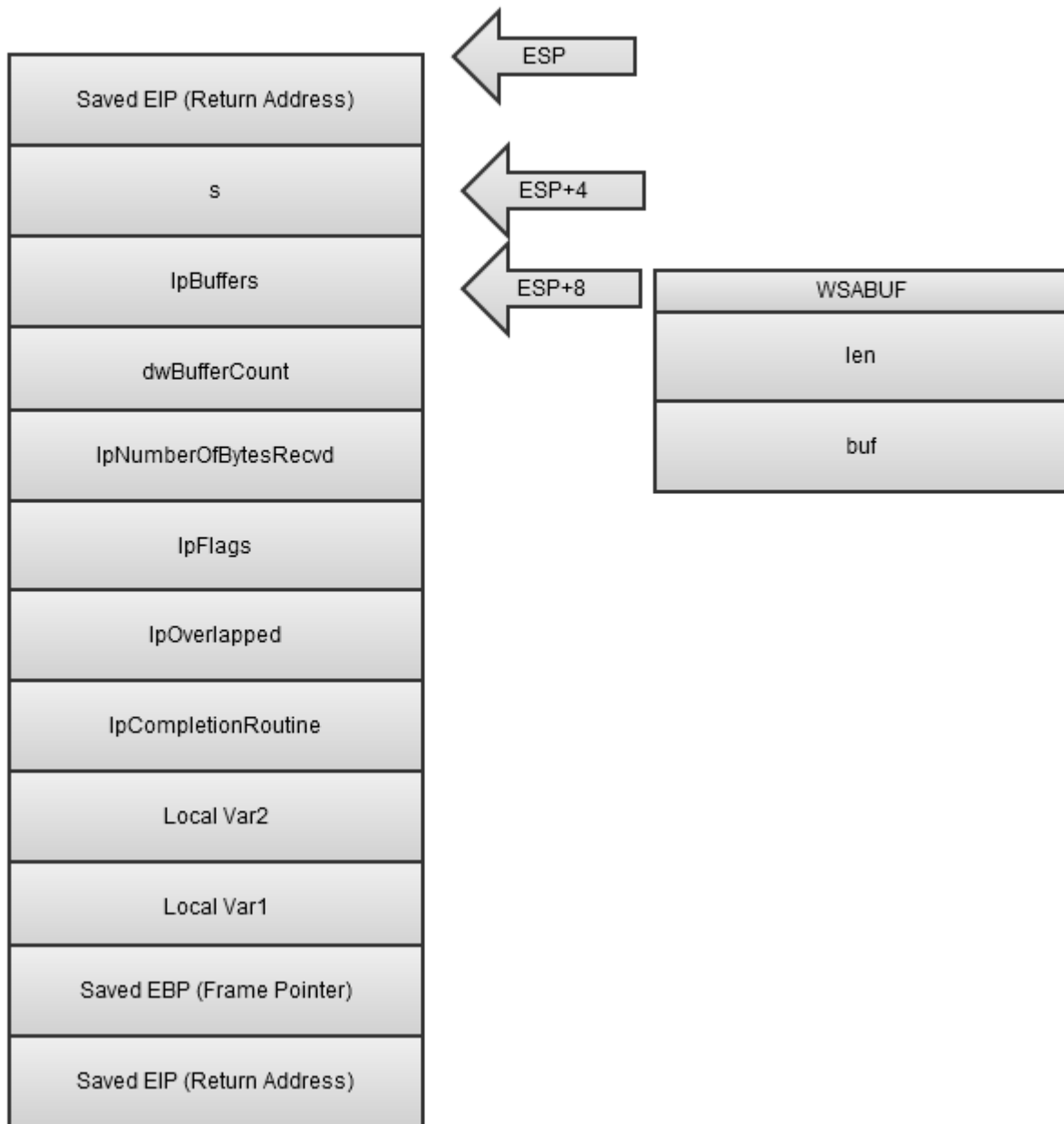
API documentation can help you set “intelligent” breakpoints

WSARecv function

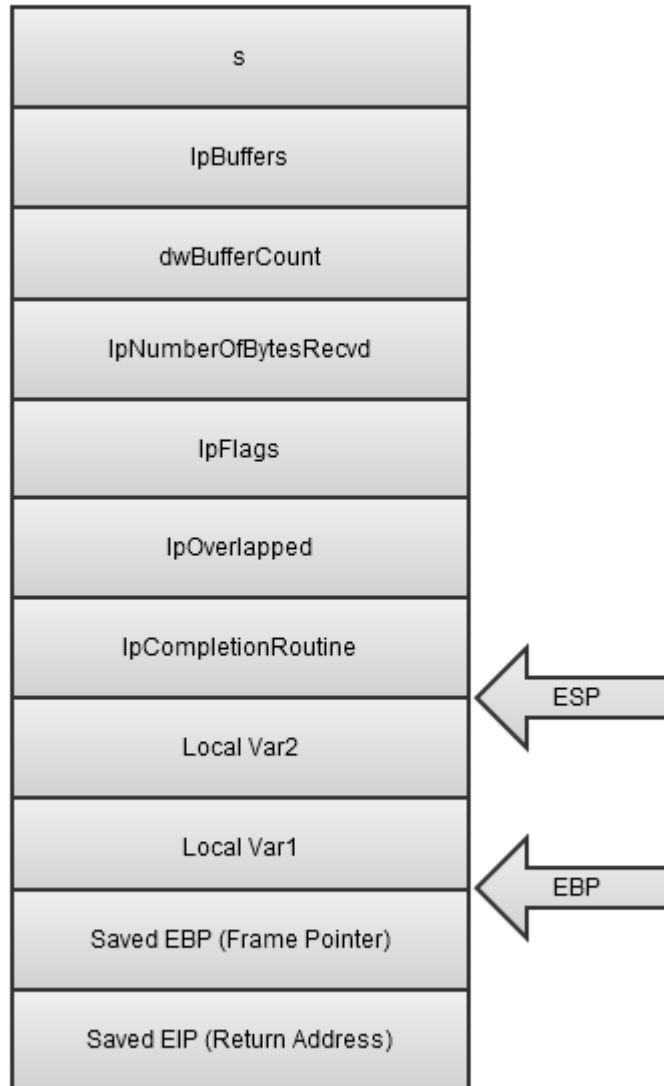
The **WSARecv** function receives data from a connected socket or a bound connectionless socket.

Syntax

```
int WSARecv(  
    __in     SOCKET s,  
    __inout  LPWSABUF lpBuffers,  
    __in     DWORD dwBufferCount,  
    __out    LPDWORD lpNumberOfBytesRecvd,  
    __inout  LPDWORD lpFlags,  
    __in     LPWSAOVERLAPPED lpOverlapped,  
    __in     LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine  
);
```







Breakpoints on API Functions

These breakpoints let you see what's going on as you use the application you're auditing

```
[*] recv called with size 10: AAAABBBBCC
[*] Call stack is:
ChildEBP RetAddr  Args to Child
0018ad14 00445471 000001f4 0018ad38 00005000 ws2_32!recv
```

```
[*] Setting memory breakpoint on buffer at 0x18ad38
```

Breakpoint 1 hit

```
putty+0x1d4f4:
0041d4f4 50                push     eax
0:000> ub @eip L2
putty+0x1d4ee:
0041d4ee 8b4508           mov     eax,dword ptr [ebp+8]
0041d4f1 0fb600           movzx  eax,byte ptr [eax]
0:000> dd @ebp+8 L1
0018ad0c  0018ad38
```

Breakpoints on Implicit Functionality

Some instruction sequences are interesting to vulnerability hunters...

Like implied string copy operations, or loops,
...

```
.text:1000C74E loc_1000C74E:      ; CODE XREF: sub_1000C700+33j
.text:1000C74E      mov     eax, [esi+2760h]
.text:1000C754      mov     ecx, [eax+4]
.text:1000C757      mov     ecx, [eax+8]
.text:1000C75A      mov     eax, [esi+48h]
.text:1000C75D      push   ecx
.text:1000C75E      push   edx          ; ArgList
.text:1000C75F      push   offset aInNew_messageS ; "In NEW_MESSAGE state\n\t-> current messag"...
.text:1000C764      push   5            ; int
.text:1000C766      push   eax          ; int
.text:1000C767      call  j_logfunc
.text:1000C76C      mov     ecx, [esi+2760h]
```

Some common (implicit) register usage

eax : return value

ecx : counter in various *rep* operations. 'this' pointer in C

esi : source pointer in various *rep* operations

edi : destination pointer

ebp : sometimes the frame pointer, sometimes general use

```
.text:1000C786      jz     short loc_1000C7AB
.text:1000C788      mov     eax, [edx+14h]
.text:1000C78C      call  eax          ; module+0x99b2 - this seeks into our
.text:1000C78C      ; buffer by our controlled dword amount. it saves the
.text:1000C78E      mov     ecx, [esi+48h]
.text:1000C791      push   0            ; ArgList
.text:1000C793      push   5            ; int
.text:1000C798      push   ecx          ; int
.text:1000C79B      mov     [esi+2Ch], edi
.text:1000C79E      call  j_logfunc
.text:1000C7A3      add     esp, 10h
.text:1000C7A6      mov     eax, edi
.text:1000C7A8      pop     edi
.text:1000C7A9      pop     esi
.text:1000C7AA      retn
```

```
.text:1000C74E loc_1000C74E:      ; CODE XREF: sub_1000C700+33j
.text:1000C74E      mov     eax, [esi+2760h]
.text:1000C754      mov     ecx, [eax+4]
.text:1000C757      mov     edx, [eax+8]
.text:1000C75A      mov     eax, [esi+48h]
.text:1000C75D      push   ecx
.text:1000C75E      push   edx                ; ArgList
.text:1000C75F      push   offset aInNew_messageS ; "In NEW_MESSAGE state\n\t-> current messag"...
```

x86 RECAP

While we're on the topic of registers...

```
.text:1000C764      push   5                  ; int
.text:1000C767      call   j_logfunc
.text:1000C76C      mov     ecx, [esi+2760h]
.text:1000C772      mov     edx, [ecx]
.text:1000C774      mov     eax, [edx+1Ch]
.text:1000C777      add     esp, 14h
.text:1000C77A      call   eax
.text:1000C77C      test   eax, eax
.text:1000C784      mov     edx, [ecx]
.text:1000C786      jz     short loc_1000C7AB
```

Some registers can be split into different bit-widths

```
.text:1000C788      if_signed 0:
.text:1000C788      mov     eax, [edx+14h]
.text:1000C78B      push   edi
.text:1000C78C      call   eax                ; module+0x99b2 - this seeks into our
.text:1000C78C      ; buffer by our controlled dword amount. it saves the
.text:1000C78C      ; faulty pointer
.text:1000C78E      mov     ecx, [esi+48h]
.text:1000C793      push   offset aStillInNew_mes ; "Still in NEW_MESSAGE state\n"
.text:1000C798      push   5                  ; int
.text:1000C79A      push   4                  ; int
.text:1000C79E      call   j_logfunc
.text:1000C7A3      add     esp, 10h
.text:1000C7A6      mov     eax, edi
.text:1000C7A8      pop     edi
.text:1000C7A9      pop     esi
.text:1000C7AA      retn
```

eax : 32-bits

ax : 16-bits

al : low 8 bits

ah : high 8 bits

x86 RECAP

Examples:

rep movsd : *rep movsd [edi], [esi]* : `eax = memcpy(edi, esi, ecx)`

rep stosd : *rep stosd [edi], eax* : `eax = memset(edi, eax, ecx)`

rep scasd : *rep scasd [edi]* : `eax = strchr(edi, eax)`

push 0x0D

call malloc : `eax = malloc(0x0D)`

```
.text:1000C74E loc_1000C74E:          ; CODE XREF: sub_1000C700+33j
.text:1000C74E      mov     eax, [esi+2760h]
.text:1000C754      mov     ecx, [eax+4]
.text:1000C758      mov     ecx, [eax+8]
.text:1000C75A      mov     ecx, [esi+48h]
.text:1000C75D      push   ecx
.text:1000C75E      push   edx          ; ArgList
.text:1000C75F      push   offset aInNew_messageS ; "In NEW_MESSAGE state\n\t-> current messag"...
.text:1000C764      push   5            ; int
.text:1000C766      push   eax          ; int
.text:1000C768      call   _logfunc
.text:1000C76C      mov     ecx, [esi+2760h]
.text:1000C772      mov     edx, [ecx]
.text:1000C774      mov     eax, [edx+1Ch]
.text:1000C777      add     esp, 14h
.text:1000C77A      call   eax
.text:1000C77C      rep   movsd [edi], [esi]
.text:1000C784      mov     edx, [ecx]
.text:1000C786      jz     short loc_1000C7AB
.text:1000C788      rep   stosd [edi], eax
.text:1000C78B      mov     eax, [edx+14h]
.text:1000C78E      call   rep_scasd [edi]
.text:1000C793      push   0            ; ArgList
.text:1000C795      push   offset aStillInNew_mes ; "Still in NEW_MESSAGE state\n"
.text:1000C79A      push   ecx          ; int
.text:1000C79B      mov     [esi+2Ch], edi
.text:1000C79E      call   j_logfunc
.text:1000C7A3      add     esp, 10h
.text:1000C7A6      mov     eax, edi
.text:1000C7A8      pop     edi
.text:1000C7A9      pop     esi
.text:1000C7AA      retn
```

Breakpoints on Implicit Functionality

You can breakpoint on these and dump out a string representation of the C code that is being executed

```
[*] rep movsd hit, memcpy(0x12345678, 0x87654321, 0x20)
```

```
[*] rep scasd hit, code is searching 0x44444444 for value "FOO"
```

And so on... get creative

Now, lets move on to some more interesting uses for breakpoints

QUESTIONS?

Memory Access Breakpoints

You can set breakpoints when an address is “touched”

...remember the *ba* command?

Combine this ability with the earlier breakpoints on `recv`

This can provide you with useful code locations

...to reverse statically

Memory Access Breakpoints

These breakpoints are most useful when dealing with
dynamic memory

How many of you are familiar with how a
memory manager works (heap)?

Dynamic Memory

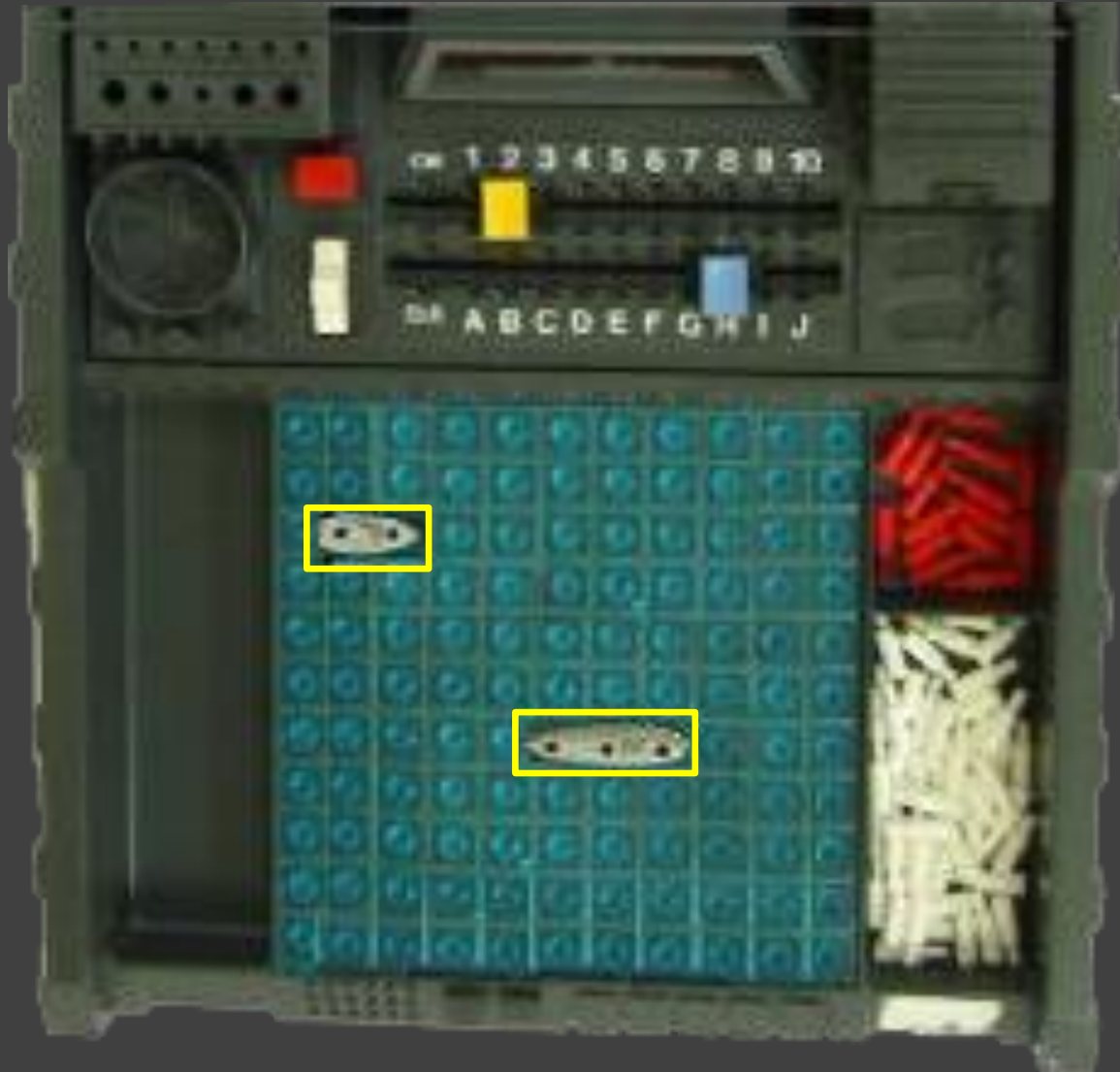


Dynamic Memory



```
A = malloc(10);
```

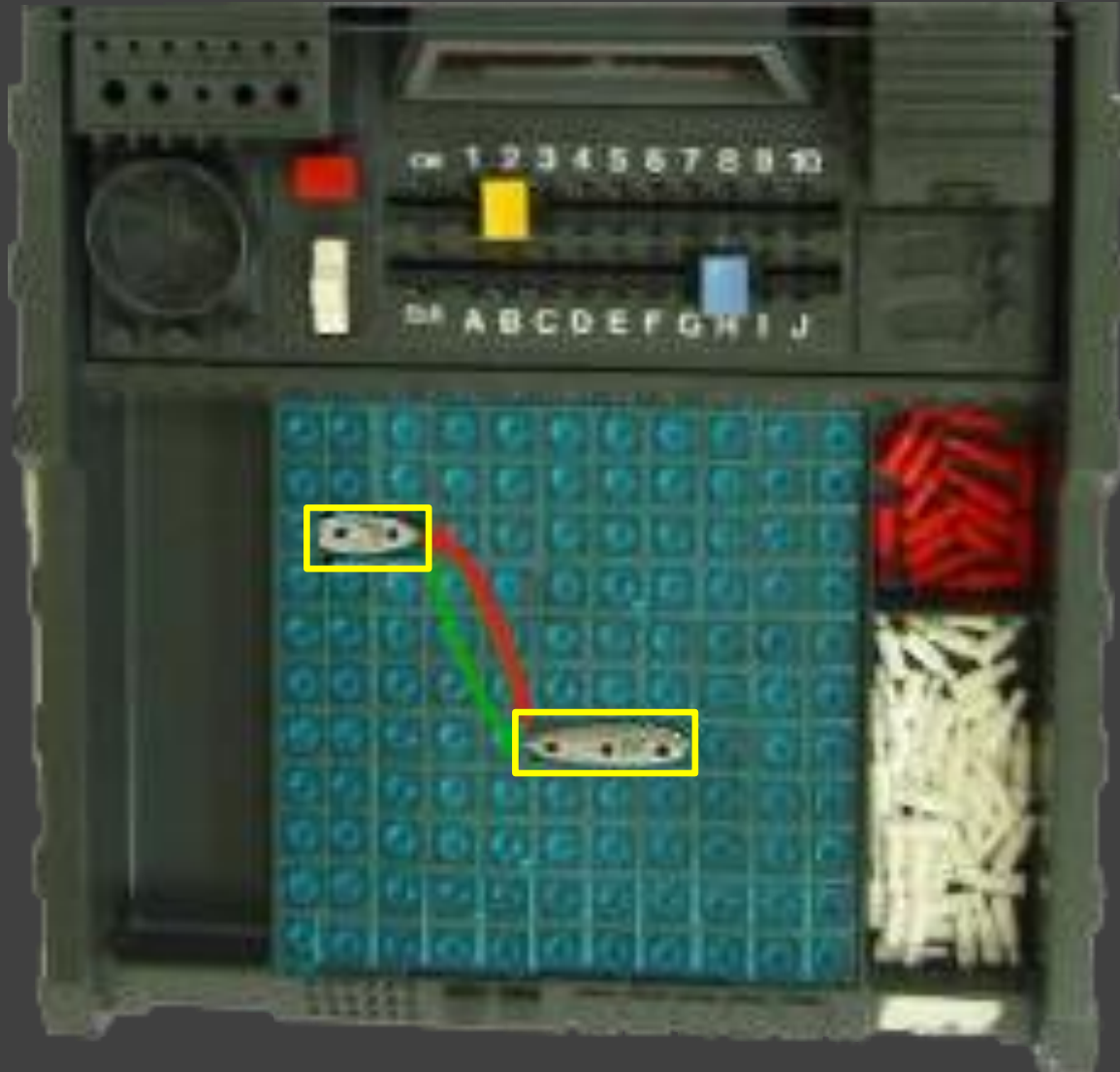
Dynamic Memory



```
A = malloc(10);
```

```
B = malloc(20);
```

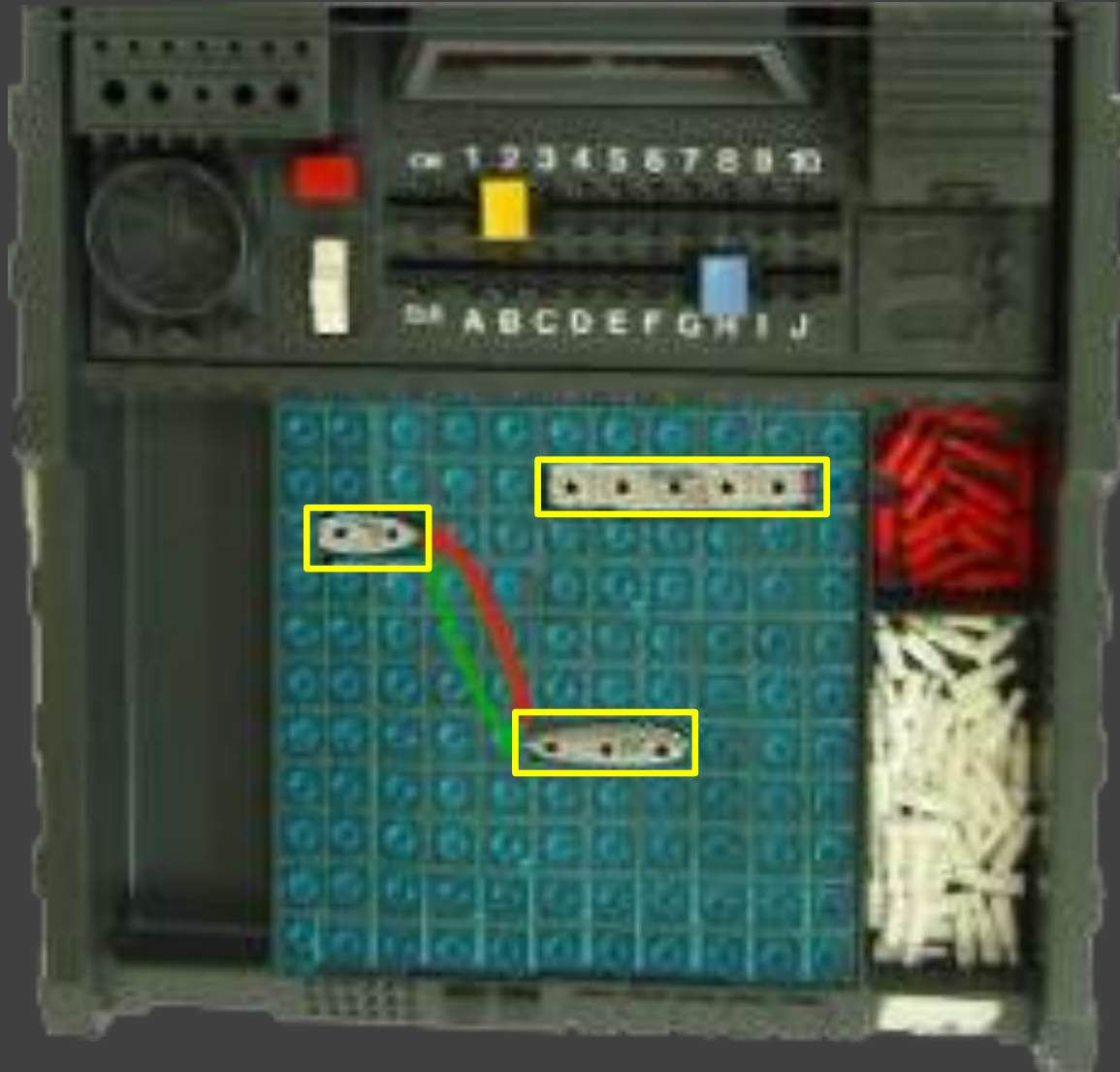
Dynamic Memory



```
A = malloc(10);
```

```
B = malloc(20);
```

Dynamic Memory

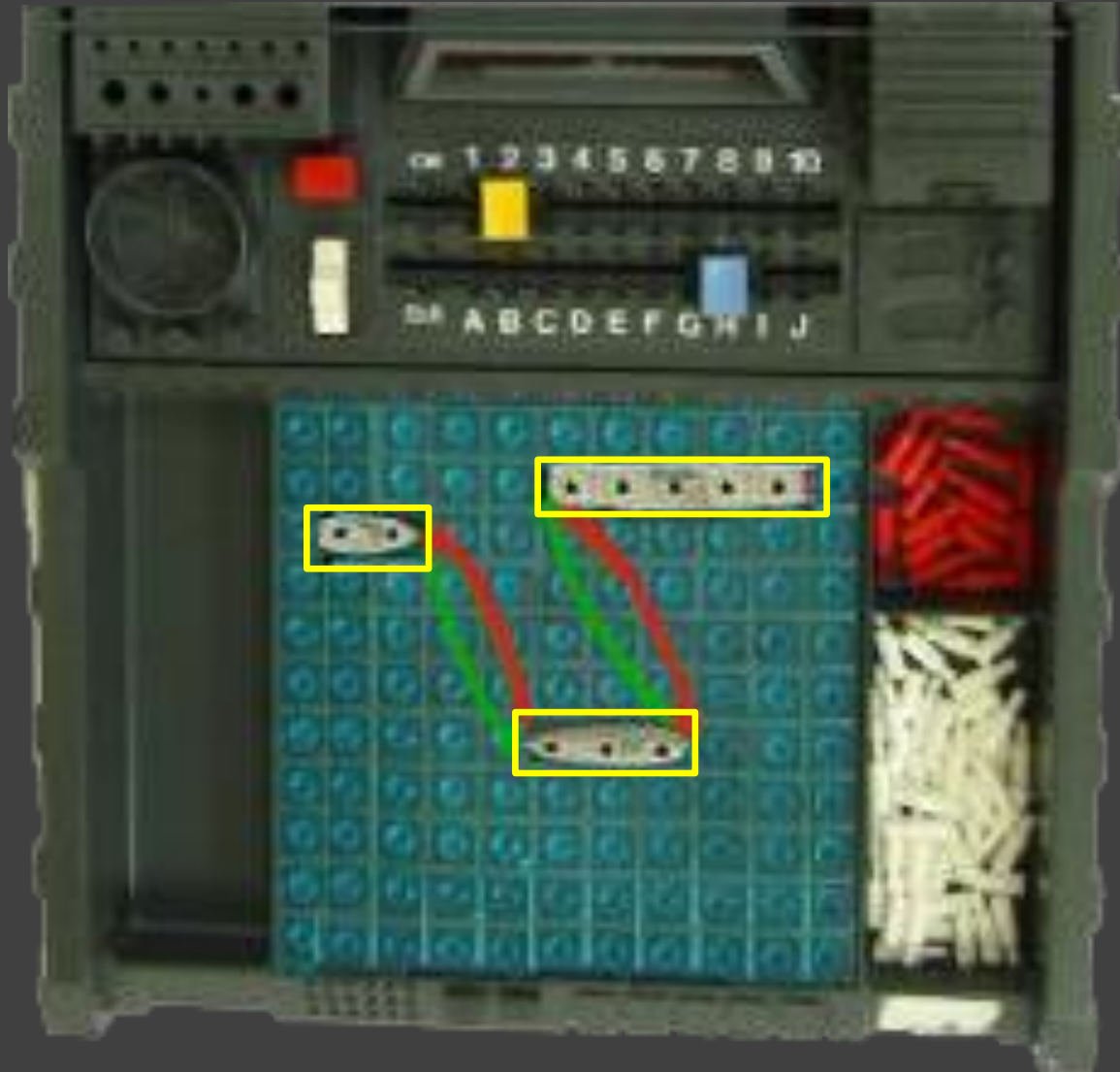


```
A = malloc(10);
```

```
B = malloc(20);
```

```
C = malloc(64);
```

Dynamic Memory

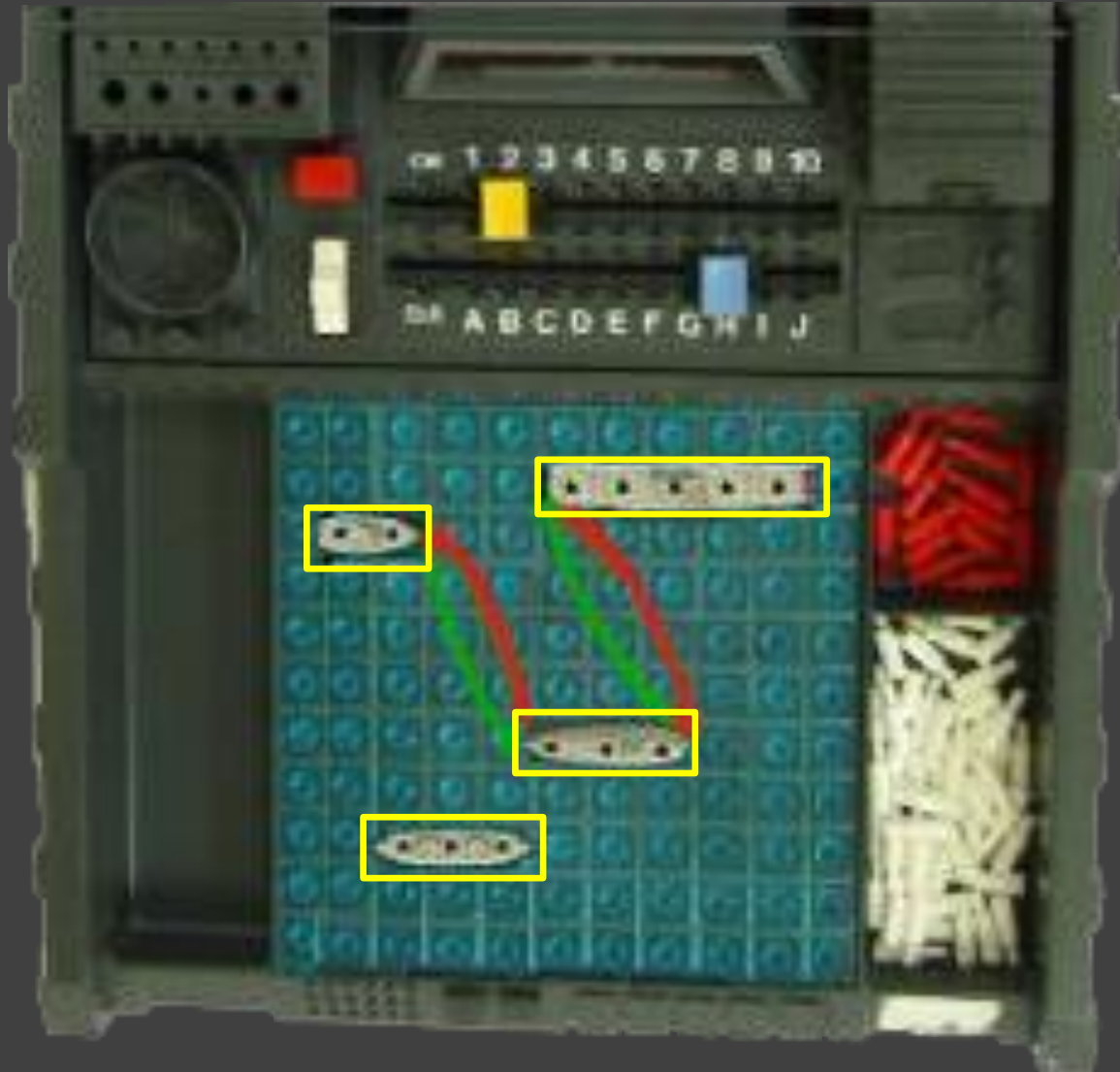


```
A = malloc(10);
```

```
B = malloc(20);
```

```
C = malloc(64);
```

Dynamic Memory



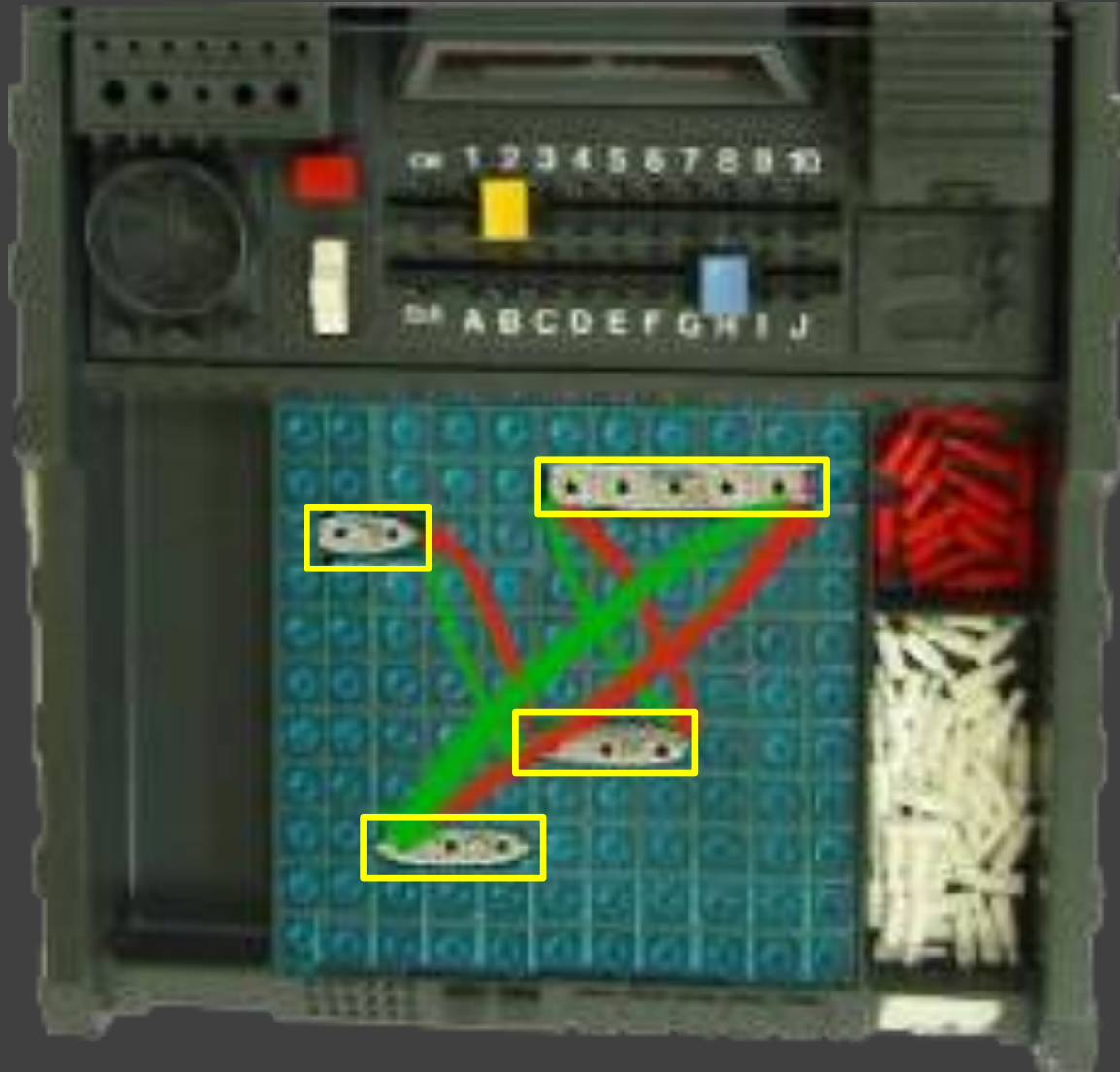
```
A = malloc(10);
```

```
B = malloc(20);
```

```
C = malloc(64);
```

```
D = malloc(20);
```

Dynamic Memory



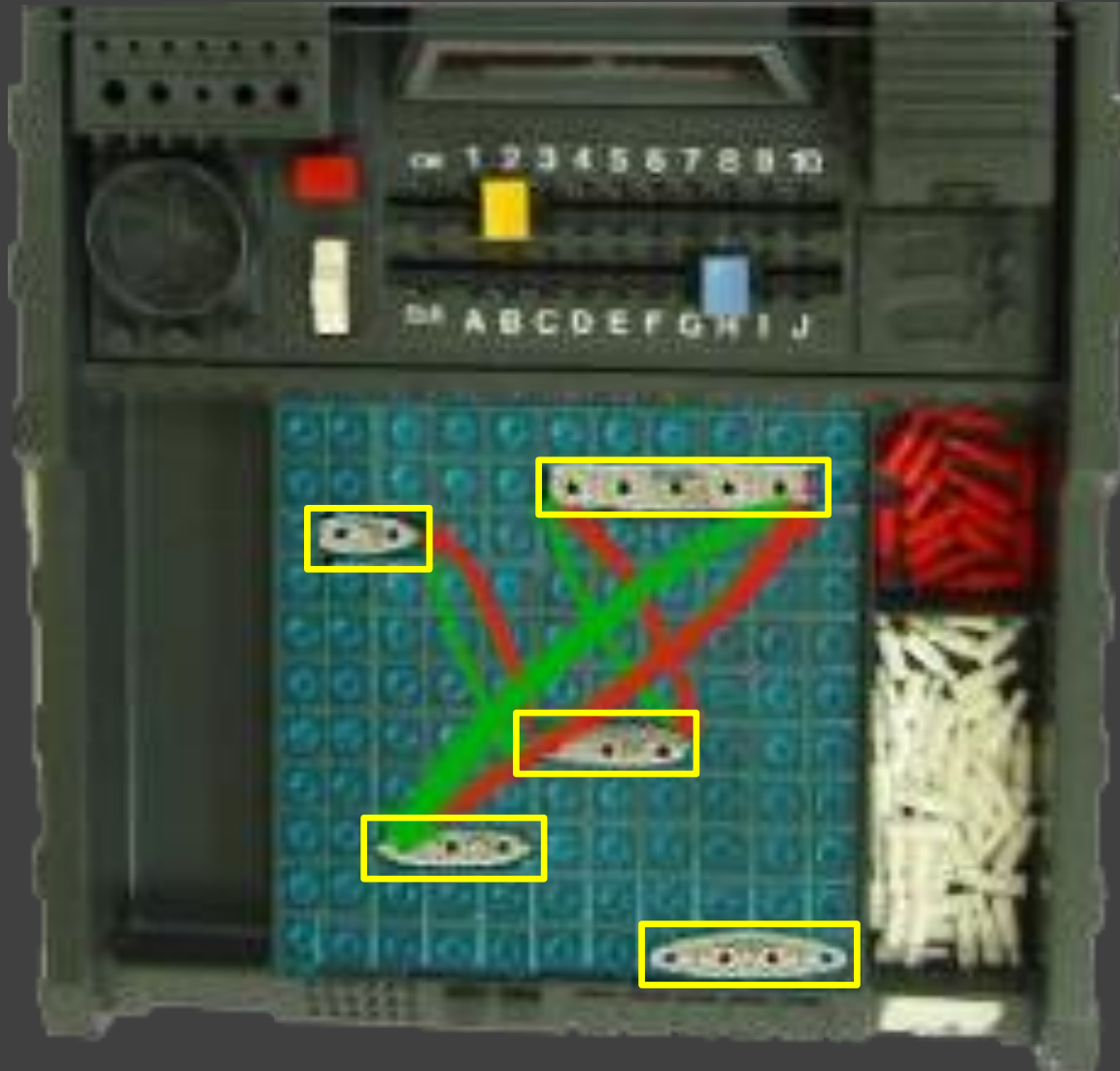
```
A = malloc(10);
```

```
B = malloc(20);
```

```
C = malloc(64);
```

```
D = malloc(20);
```

Dynamic Memory



```
A = malloc(10);
```

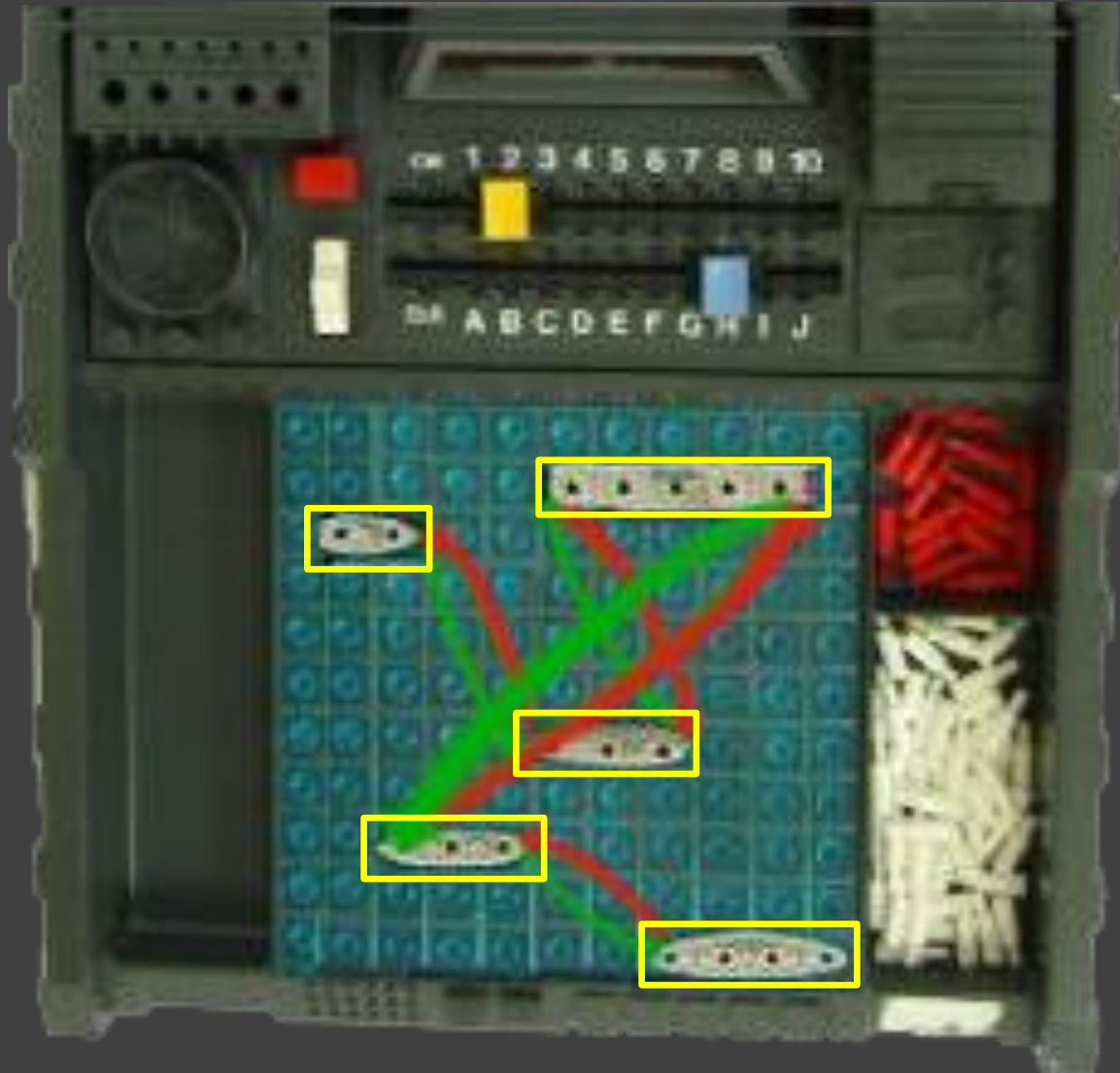
```
B = malloc(20);
```

```
C = malloc(64);
```

```
D = malloc(20);
```

```
E = malloc(30);
```

Dynamic Memory



```
A = malloc(10);
```

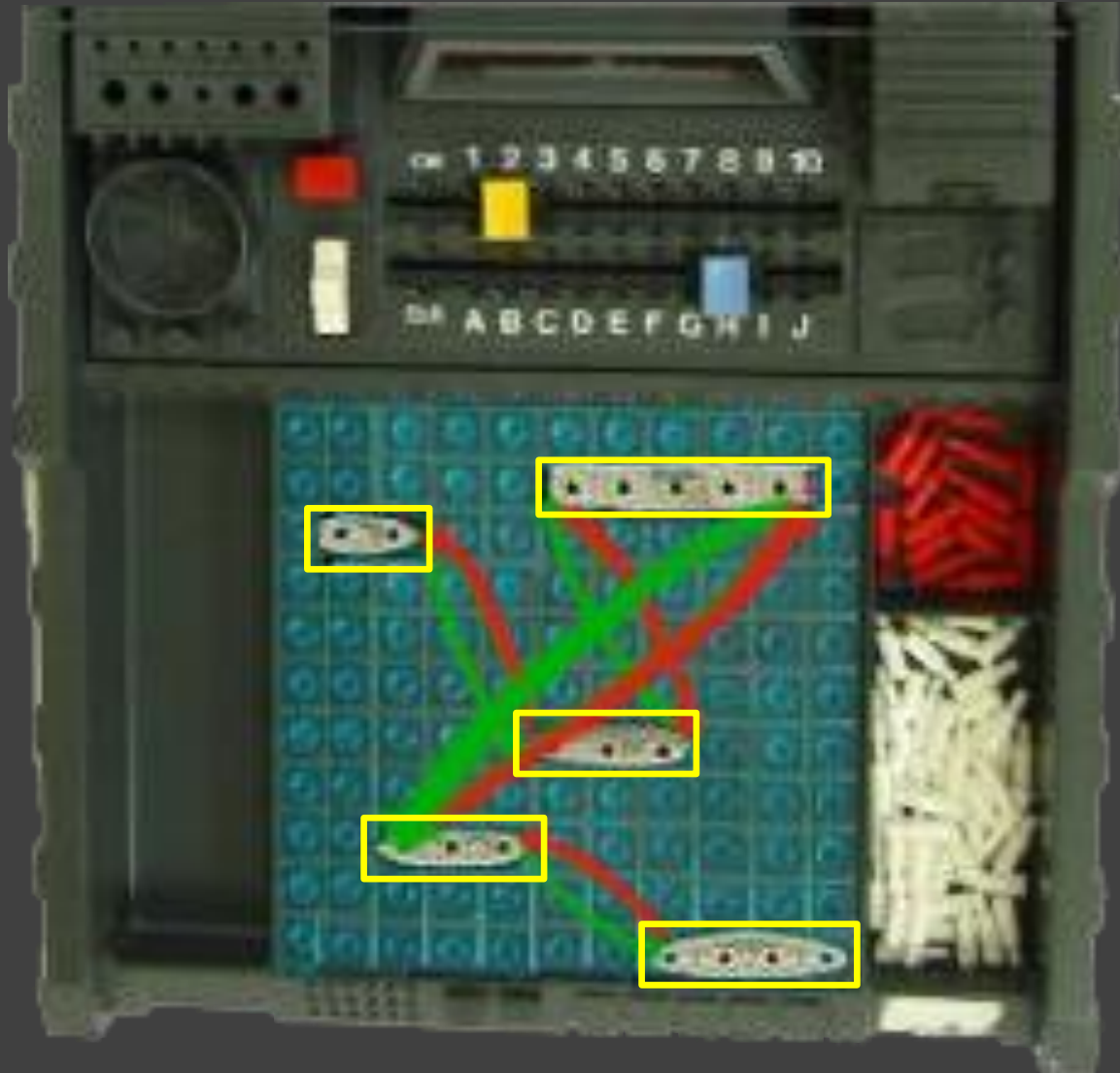
```
B = malloc(20);
```

```
C = malloc(64);
```

```
D = malloc(20);
```

```
E = malloc(30);
```

Dynamic Memory



```
A = malloc(10);
```

```
B = malloc(20);
```

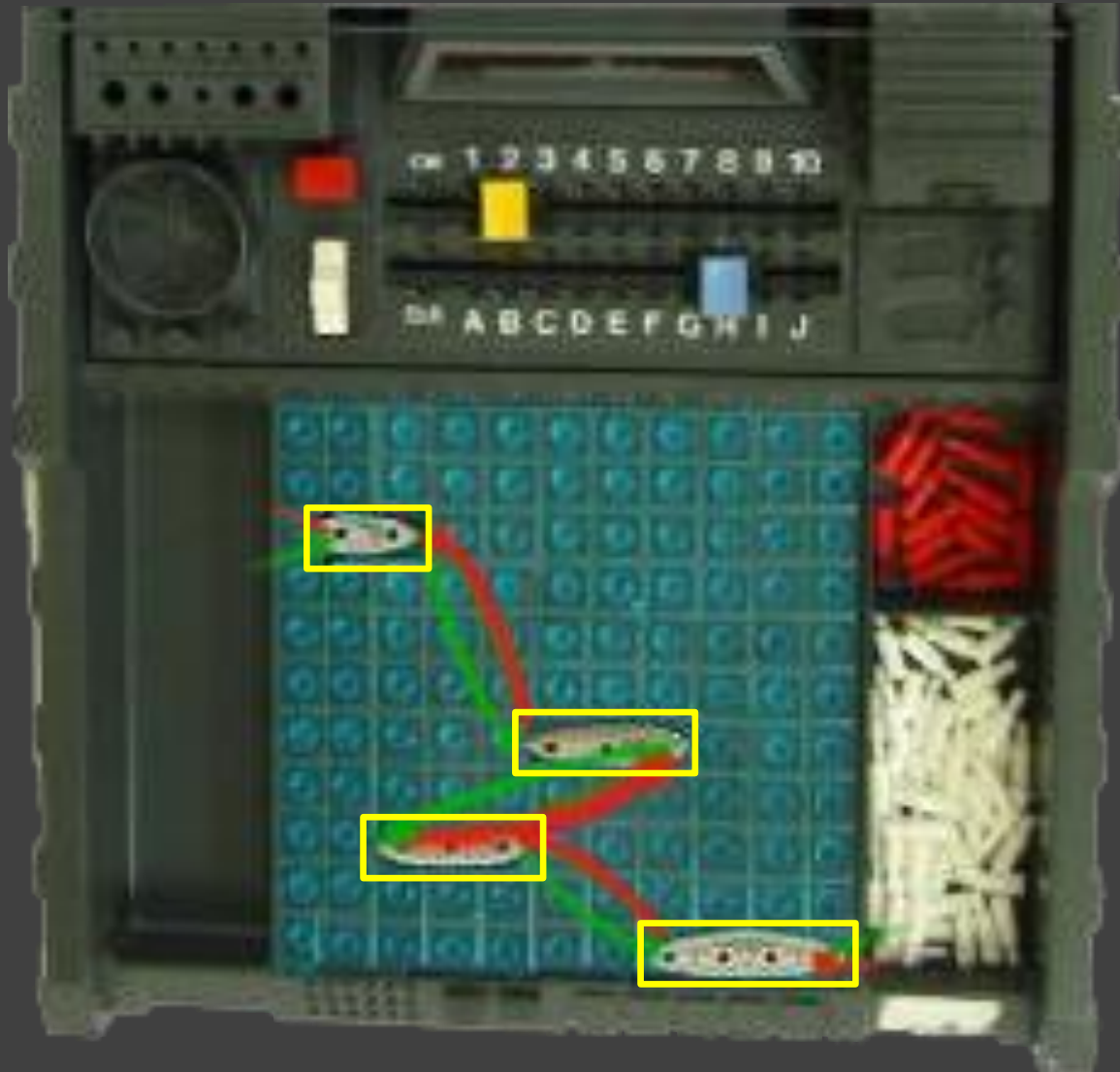
```
C = malloc(64);
```

```
D = malloc(20);
```

```
E = malloc(30);
```

```
free(C);
```

Dynamic Memory



```
A = malloc(10);
```

```
B = malloc(20);
```

```
C = malloc(64);
```

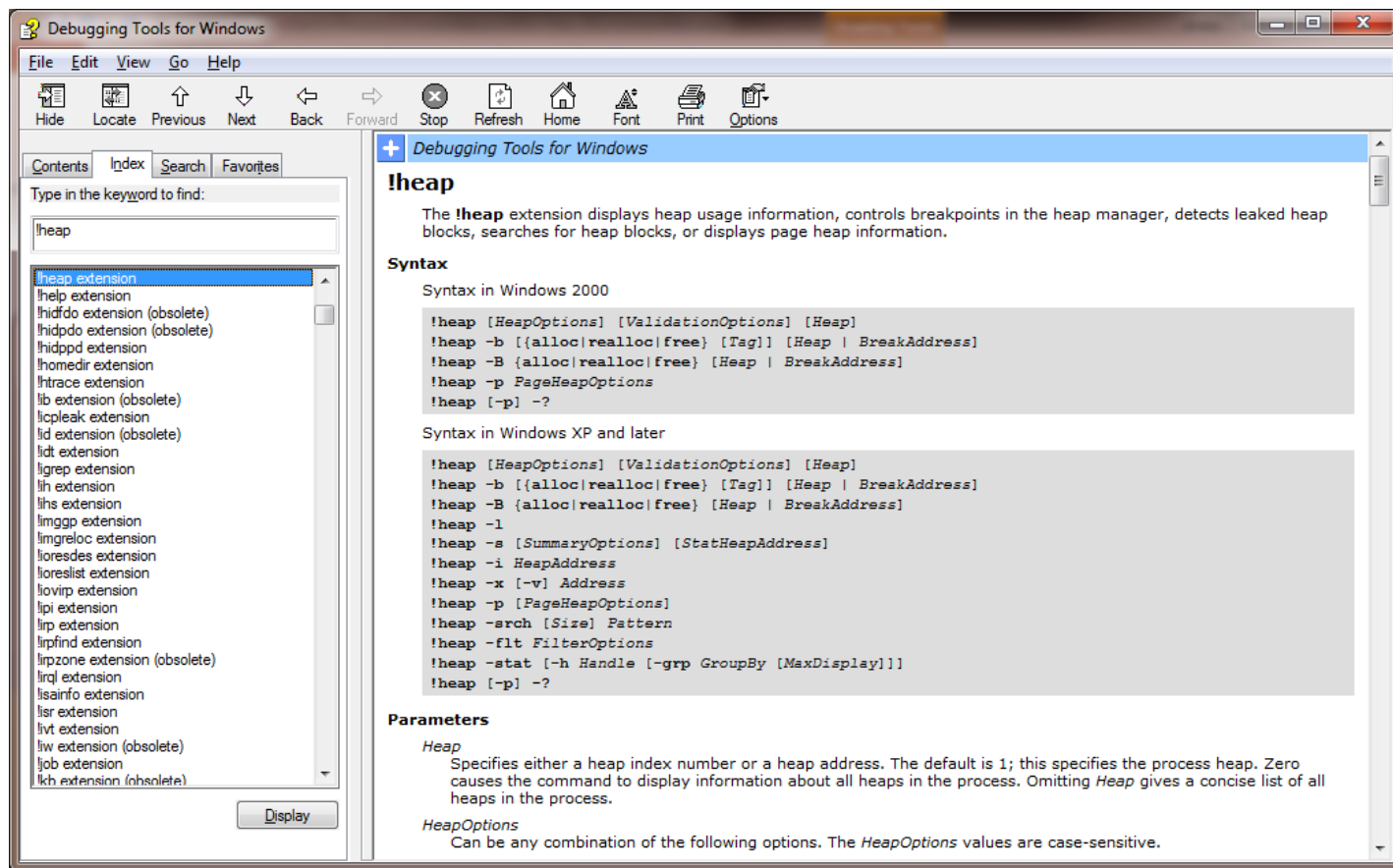
```
D = malloc(20);
```

```
E = malloc(30);
```

```
free(C);
```

!heap Extension

WinDBG allows you to inspect the heap in a few different ways



The screenshot shows the 'Debugging Tools for Windows' help window. The left pane contains a search bar with 'heap' entered and a list of extensions. The right pane displays the documentation for the '!heap' extension, including its description, syntax for Windows 2000 and Windows XP and later, and a list of parameters.

!heap

The **!heap** extension displays heap usage information, controls breakpoints in the heap manager, detects leaked heap blocks, searches for heap blocks, or displays page heap information.

Syntax

Syntax in Windows 2000

```
!heap [HeapOptions] [ValidationOptions] [Heap]
!heap -b [{alloc|realloc|free} [Tag]] [Heap | BreakAddress]
!heap -B {alloc|realloc|free} [Heap | BreakAddress]
!heap -p PageHeapOptions
!heap [-p] -?
```

Syntax in Windows XP and later

```
!heap [HeapOptions] [ValidationOptions] [Heap]
!heap -b [{alloc|realloc|free} [Tag]] [Heap | BreakAddress]
!heap -B {alloc|realloc|free} [Heap | BreakAddress]
!heap -l
!heap -s [SummaryOptions] [StatHeapAddress]
!heap -i HeapAddress
!heap -x [-v] Address
!heap -p [PageHeapOptions]
!heap -srch [Size] Pattern
!heap -flt FilterOptions
!heap -stat [-h Handle [-grp GroupBy [MaxDisplay]]]
!heap [-p] -?
```

Parameters

Heap
Specifies either a heap index number or a heap address. The default is 1; this specifies the process heap. Zero causes the command to display information about all heaps in the process. Omitting *Heap* gives a concise list of all heaps in the process.

HeapOptions
Can be any combination of the following options. The *HeapOptions* values are case-sensitive.

!heap Extension

Some useful !heap commands

- ❑ !heap 0 -v
 - ❑ This command walks the main heap, verifying its structure

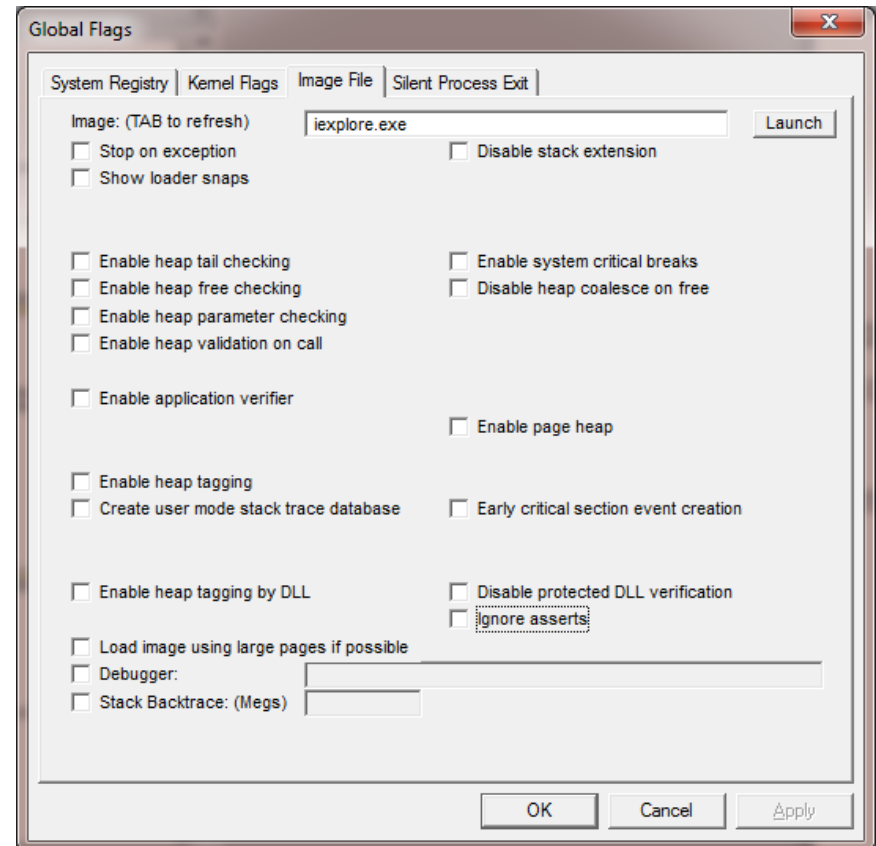
- ❑ !heap -p -a target
 - ❑ User-mode Stack Trace Database
 - ❑ This command will give you a call stack for an allocation pointed to by target

Global Flags

Some of the heap commands require extra information to be stored while an application is running.

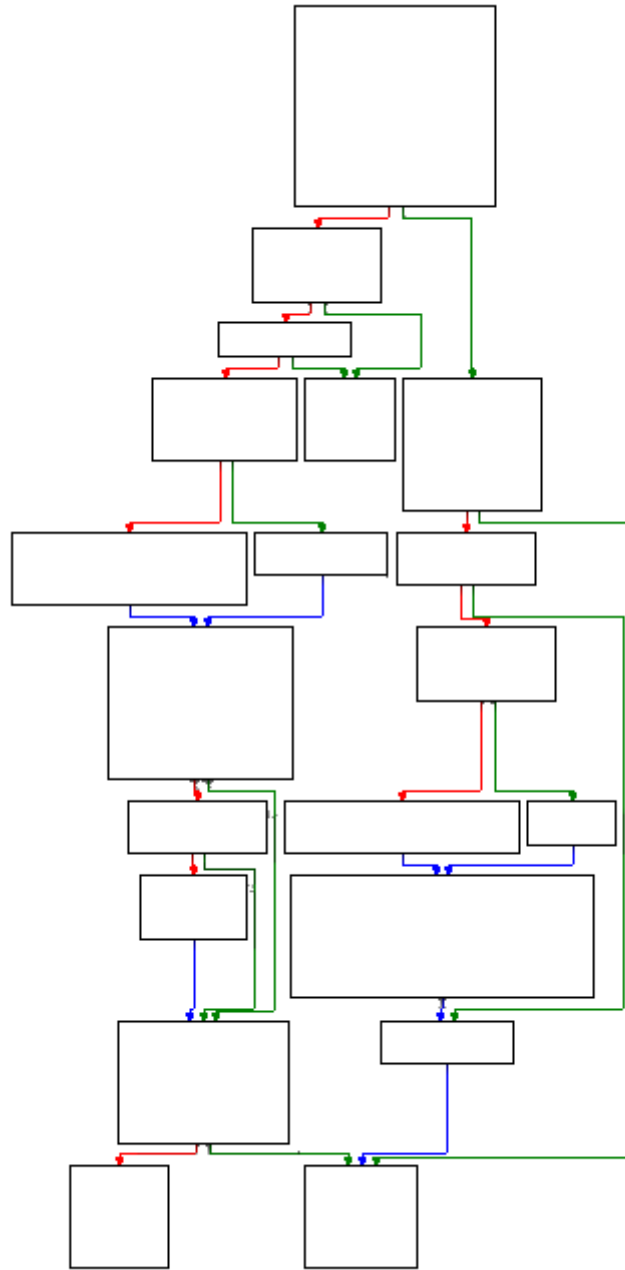
Enter Global Flags...

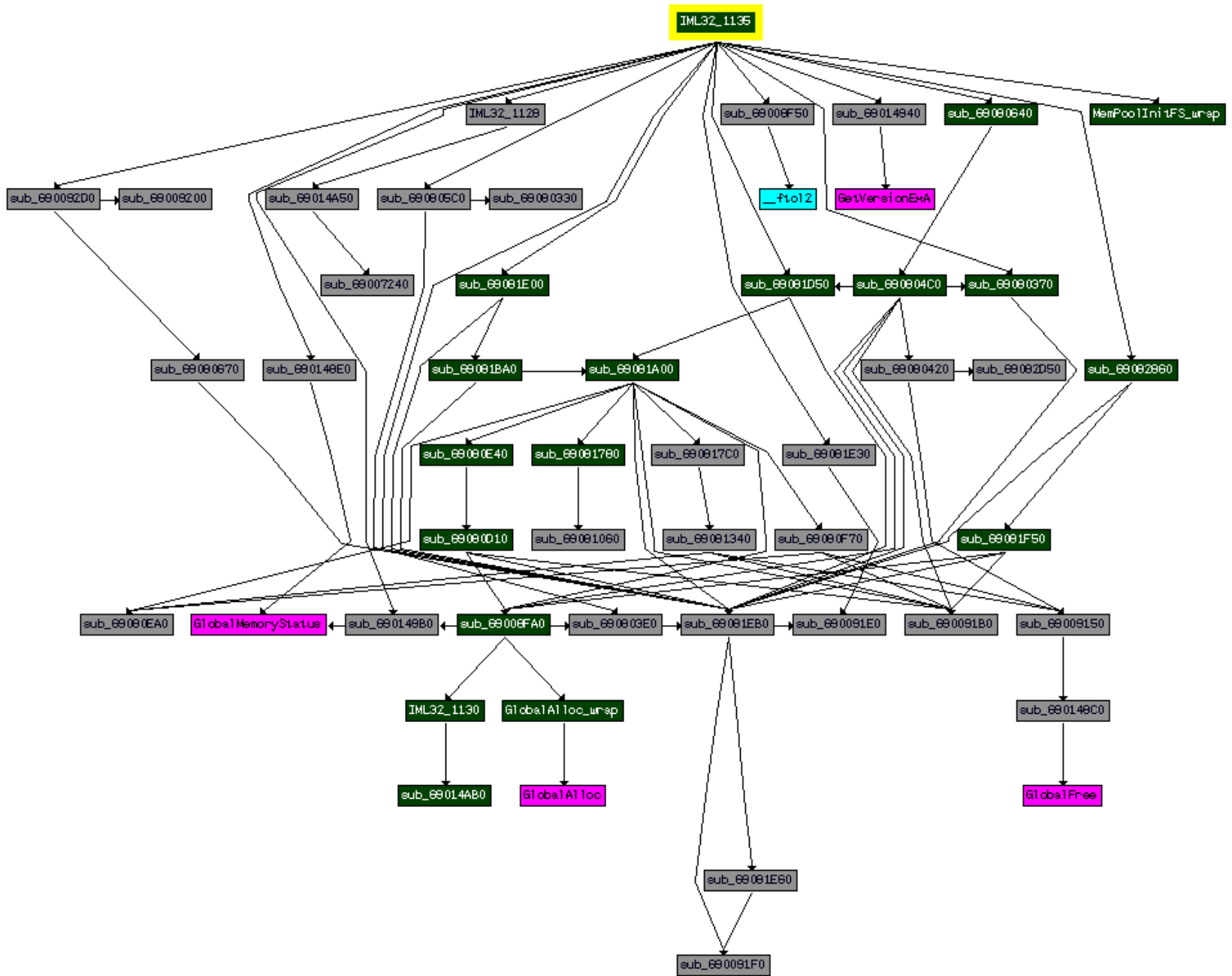
gflags.exe is distributed with the debugging tools for Windows (as is WinDBG)



QUESTIONS?

Graphs and Paths





Hit Tracing

Determining basic block or function-level flow is called **hit tracing**

Imagine having your entire IDA Pro database marked up with every location that was reached in a debugger

Useful, right?

QUESTIONS?