sensepost

WADI FUZZER

Saif El-Sherei & Etienne Stalmans

What is Fuzzing??

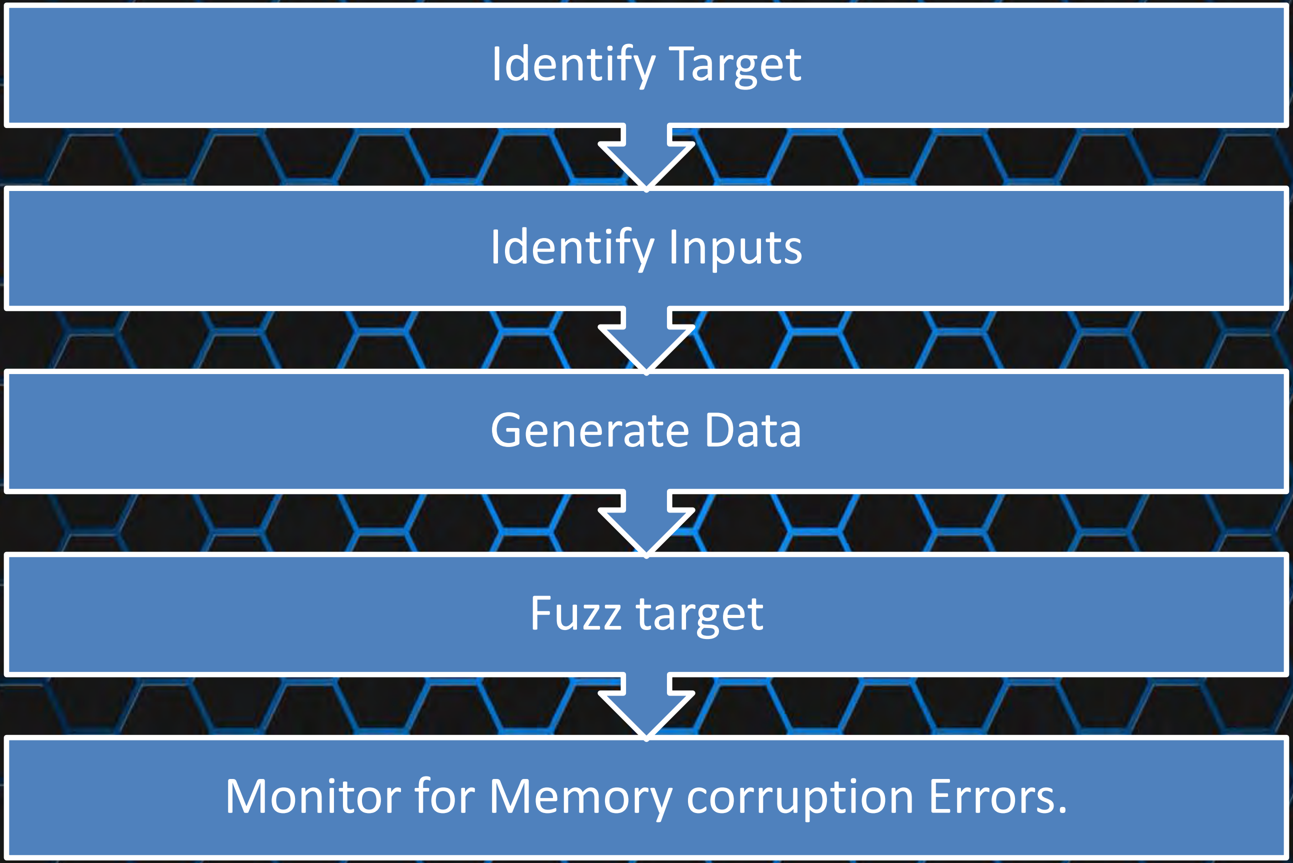
Fuzzing is feeding an Application with malformed input in hope to find errors and faults in the application code and with a bit of luck these faults can lead to exploitable vulnerabilities

Basic History of Fuzzing

- 1988-199:
  - Boris Beizer Syntax Testing.
  - Barton Miller Fuzz: An Emprical Study of Robustness.
- 1999 – 2001 OUSPG PROTOS SNMP, HTTP, SIP, H.323, LDAP, etc …
- 2002 Dave Aitel SPIKE block based fuzzing. Codnemicon first commercial fuzzer.
- 2004 Browser Fuzzing start lcamtuf's MangleMe.
- 2005 FileFuzz , SPIKEfile NotSPIKEfile File format fuzzing.
- 2006 Month of Browser Bugs (MoBB) HD Moore relases a browser bug every day for a month release of CSSDIE, COMRaider and Axman and hamachi.
- 2011 Lcamtuf decided to revolutionize Browser fuzzing in 2011 by releasing cross_fuzz. In his own words "a surprisingly effective but notoriously annoying cross-document DOM binding fuzzer that helped identify about one hundred bugs in all browsers on the market - many of said bugs exploitable - and is still finding more." it is based on ref_fuzz which he developed in 2008.
- 2014 lcamtuf's introduces American Fuzzy lop (Afl) Evolutionary fuzzer.

Fuzzing Types And Techniques

Fuzzing methodology



Fuzzing Types

- **Mutation/Non-Intelligent Fuzzing**

Randomly apply mutation algorithms to the supplied input to generate several test cases without any concern to the target format.

- **Generation/Intelligent Fuzzing**

Utilize grammar to model a certain format specification and randomly generate semi-valid test cases. to minimize fault conditions and generate test cases that are accepted by the target.

Fuzzing Types Contd..

- **Evolutionary Fuzzing**

  Combining either types of fuzzing with code and binary instrumentation tools, to monitor code paths and generate test cases based on the results of the instrumentation to achieve least number of test cases with highest amount of code coverage and branches explored.

  In Fewer words lcamtuf's American fuzzy lop.

Tools of the Trade - Memory Error Detectors

Memory Error Detectors poisons memory areas after memory allocations and after the memory is free-ed. It monitors access to these parts in memory and returns detailed error information.

Windows:

- PageHeap which is part of Gflags part of windows debugging toolkit can be applied on some windows processes.

Linux and OSX:

- Google's Address Sanitizer (Asan) is a clang compiler plugin that can be implemented during compilation time of any linux or OSX application.

Tools of the Trade - Fuzzing Harnesses

Fuzzing harnesses are not themselves fuzzers but they are tools that run the target process feed it the generated test case and monitor the process for crashes.

Windows:

- Grinder by Stephen Fewer.

Linux and Mac OSX:

- Node Fuzz by Atte Kettunen of OUSPG.

Introducing

WADI FUZZER

Wadi – Fuzzing Harness

Atte Kettunen's of OUSPG NodeFuzz:

- Nodefuzz is a fuzzing framework that works on Linux, And Mac OSX.
- It is coded by Atte Kettunen of OUSPG using Nodejs.
- It works by instrumenting the browser using ASan. and a test case generation module to feed the browser the test case through web sockets.
- The modules are not provided as part of NodeFuzz you can code your own. it is a pretty simple process.
- NodeFuzz is what we are currently using. with our own custom modules.

Wadi – Memory Error Detector

Google's AddressSanitizer (ASan):

- AddressSanitizer (ASan) is a clang compiler plugin Developed By Google which allows fast memory error detection.
- The run-time library replaces the malloc and free functions. The memory around malloc-ed regions (red zones) is poisoned. The free-ed memory is placed in quarantine and also poisoned. Every memory access is monitored and if address is poisoned a detailed error is returned.
- It Helps find use-after-free and heap,stack,global}-buffer overflow bugs in C/C++ programs. For Linux and mac OSX
- Google and Mozilla both releases ASan pre built binaries for testing.

What is Wadi?

- Exploring new tributaries in browser fuzzing.
- Grammars are used to describe how browsers should process web content, Wadi turns that around and uses grammars to break browsers.
- Wadi already responsible for a handful of high severity bugs in browsers.

Why Wadi?

- From Chrome, to IE, Wadi identifies exploitable bugs in new and existing web APIs.
- The talk introduces Wadi and walks the audience through the steps taken to go from LL(1) grammar to fuzz test cases and browser crashes

A simple Intro to The DOM

Interfaces are types of objects that allow web applications and web browsers to programmatically access and interact with them to access their members.

The Document Object Model (DOM):

The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them.

Web API:

When writing code for the Web using JavaScript, there are great many APIs available. that you may be able to use developing Web applications. ex: speech, webaudio, gamepad, canvas, webgl, animation, etc..

Wadi Architecture

- Wadi is 3538 lines of code.
- 2932 of these are grammar for test case generation.
- Wadi works as a NodeFuzz Module and it is used to fuzz Chromium and Firefox Asan builds.
- Already responsible for a number of bugs.

What is Grammar ?

Grammar in English explains how a sentence is constructed. The same can be said about Grammars in compilers it describes how the language syntax is constructed the input is parsed based on a set rules “Productions” and tokens defined in the grammar definitions. In fuzzing grammar is used to help the fuzzer generate valid test cases.

w3c have provided us with a nice interface definition language(IDL) that defines browser technology interfaces this is utilized by the fuzzer to be able to create and fuzz all attributes and methods used by them.

Grammar – IDL Interface

According to the IDL definitions an interface is an object with a set of interface members these can be constants, attributes, or functions. Each interface has a unique identifier and inherits from a parent interface if needed.

- **LL(1) Interface Definition Grammar:**

```
"interface" identifier Inheritance "{" InterfaceMembers "}" ";“
```

Grammar – IDL Interface Example

- **Identifier:** Text
- **Inheritance:** CharacterData

```
interface Text : CharacterData {
            Text            splitText(in unsigned long offset)
                                                 raises(DOMException);
            Text            replaceWholeText(in DOMString content)
                                                 raises(DOMException);
            readonly  attribute boolean        isElementContentWhitespace;
                      attribute DOMString   wholeText;
};
```

Grammar – IDL Interface Members

An interface member can be a constant, attribute or function as mentioned before. What we are interested in are the attributes and functions of an interface object. Taking a closer look at how interface members are defined in the IDL specification.

- **InterfaceMembers:**

Attributes: isElementContentWhitespace, wholeText

Functions: splitText(), replaceWholeText()

LL(1) Grammar Interface Member Definition:

```
InterfaceMember    →          Const
                   | AttributeOrOperation

AttributeOrOperation      →         "stringifier" StringifierAttributeOrOperation
                          | Attribute
                          | Operation
```

Grammar – IDL Attributes

An attribute is a declared interface member with an identifier whose value can be retrieved and in some cases changed.

LL(1) Grammar Attribute Definition:

```
Attribute →        Inherit ReadOnly "attribute" Type identifier ";"
```

Attributes of Example Interface:

```
readonly attribute boolean      isElementContentWhitespace;

         attribute DOMString   wholeText;
```

Example Interface Attributes:

Identifier	Input Type	ReadOnly
isElementContentWhitespace	boolean	True
wholeText	DOMString	False

Grammar – IDL Functions

Functions in the IDL specification is referred to as operations. A function is an interface member that defines behavior that can be invoked on objects implementing the interface.

LL(1) Grammar Attribute Definition:

Operation	→	Qualifiers OperationRest
OperationRest	→	ReturnType OptionalIdentifier "(" ArgumentList ")" ";"

Functions of Example Interface:

Text	replaceWholeText(in DOMString content)
Text	splitText(in unsigned long offset)

Example Interface Functions:

Identifier	Number of Args	Arg Identifier	Arg Type	Return Value
replaceWholeText	One	content	DOMString	Text
splitText	One	offset	unsigned long	Text

Grammar – Fuzzing Grammar

Using the gathered information we can parse the Interface objects in any given IDL to a JavaScript object containing information about the interface members to be used by the fuzzer.

There are some main pieces of information gathered from the example interface mainly the interface identifier (Name), inherited parent interface, interface members (methods, attributes).

For Attributes and Methods an array of arrays is created with each child array containing information about a single Attribute or method members of the respective interface.

A single Attribute array will have three members:

[The Attribute Identifier, [function(s) to generate the expected value type],’readOnly flag’]

A single Method array will have three members:

[The Method Identifier, [function(s) to generate method parameters and values],’high flag’]

Grammar – Fuzzing Grammar Contd.

You can code your own functions to generate attribute values and method parameters. Or use the already available helper functions in NodeFuzz ‘*randoms.js*’ file.

The main Attributes and Methods arrays are concatenated to the parent interface object attributes and methods arrays to simulate inheritance.

Example Interface as defined in the Fuzzing grammar:

```
TextInterface = {
        'Name': 'text',
        'Attributes': [
                        ['isElementContentWhitespace',[GenerateExpectedValue()],'readonly'],
                        ['wholeText',[ GenerateExpectedValue()],’’]
        ].concat(CharacterDataInterface.Attributes),
        'Methods':[
                        ['replaceWholeText',[ GenerateExpectedParameters()],'high'],
                        ['splitText',[ GenerateExpectedParameters()],'high']
        ].concat(CharacterDataInterface.Methods),
        'tagName':'Text',
        'style':CSS2PropertiesInterface
};
```

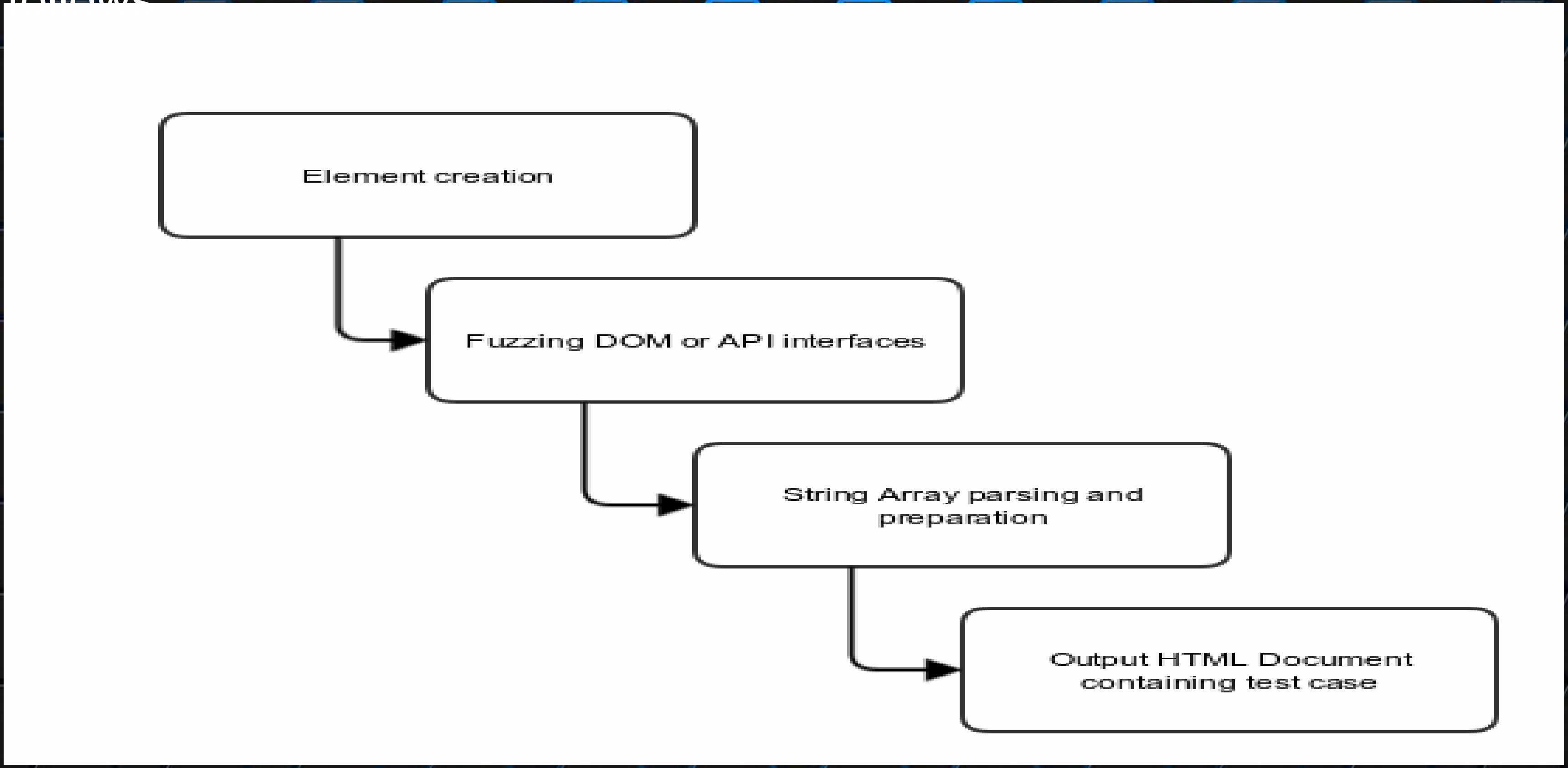
Helper Functions

These are some of the helper functions that are implemented in NodeFuzz '*randoms.js*' we can implement our own or tweak the ones already available to our needs these functions are used in generating attributes values, method parameters and all through the test case generation.

Name	**Description**
randoms()	return random number, float value or hex number
rint(num)	return a random number limited by parameter.
ra(array)	return a random element an array.
arrayWalk(array)	return a random element from array if it is a function execute it and retrurn the return value. if it is a string or an int return the value.
string(num)	return a random length string with length based on a random number limited by the input parameter.
randbool()	returns random Boolean.
floatValue(num)	return a random float value.
getRandomColor()	returns a random color in either hex, hsl, rgbint formats
distanceValue()	return a random number or float with distance suffixes like px,%,cm, etc ...
retURI(num)	return a random length URL
returnRandomElement()	returns a random element from the list of created elements and either try to reference a near by object like 'firstChild','nextSibling',etc ... or just return the element object itself.

Fuzzing Module

Wadi works on grammar created from the IDL that mapped interfaces to javascript objects using these objects to be able to generate valid JS statements into a string array. Then output an HTML document with a script containing the generated test case. The flow of the Wadi is as follows


Element creation
Fuzzing DOM or API interfaces
String Array parsing and preparation
Output HTML Document containing test case

Test Case generation – Element Creation

These are the main and first functions executed by Wadi. It generates JS statements to randomly create elements from the available HTML interfaces and inserts random child text nodes.

Name	Description
createElement()	creates a random element from the list of interfaces and saves a reference to the object both in fuzzer space and browser space
createTextNode()	creates random length text nodes and attach them to random elements in the DOM.
mangleElements()	randomly mangles element positions within the document.

Wadi Output:

```
try { HTML0=document.createElement("EMBED")} catch(e) {}
try { HTML0.id="HTML0"} catch(e) {}
try { createdElements['HTML0']=HTML0} catch(e) {}
try { document.body.appendChild(HTML0)} catch(e) {}
```

Test Case generation – Element Creation Contd.

The creation functions will save a references to the created Element objects to the local fuzzer space objects array '***CreatedElements***' to be able to access properties and methods of the created element. As well as save a reference to the created object to the browser space to be able to manipulate the saved references.

The saved object in fuzzer space will have the following structure for the previous example:

```
{   'objName':HTML0,
    'type':'object name',
    'object':Embed interface object reference
};
```

Test Case generation – Fuzzing Interfaces Functions

Wadi will then call the function ***fuzz(num)*** num being the number of rounds to execute fuzzer functions. Simply the ***fuzz()*** function picks a random function name from the below list and executes it and return the output JavaScript statement to our string array.

Function	**Description**
fuzzWindowAttribs,	randomly set the 'window' interface object attributes.
fuzzWindowMethods,	randomly call the 'window' interface object methods.
fuzzStyle,	Pick a random element and set a random style property using element.style.
fuzzStyle1,	pick a random style sheet with random reference to element and set random style properties using insertRule.
fuzzDocumentAttribs,	randomly set the 'document' interface object attributes.
fuzzDocumentMethods,	randomly call the 'window' interface object methods.
deleteRandomKey,	deletes a random refence to the created objects saved in the 'createdElements' object in browser space.
fuzzPLayerMethods,	if no animation player found call the createPlayer() function to create a new animation player and add reference to it in the createElements object array. if a player exist call a random method from the animation interface object.
fuzzPlayerAttribs,	if no animation player found call the createPlayer() function to create a new animation player and add reference to it in the createElements object array. if a player exist set a random attribute from the animation interface object.

Test Case generation – Wadi Interfaces contd.

Function	**Description**
MutationObserve,	creates a random mutation observer and add reference to it in the createElements object array.
fuzzMutationObserve,	if no mutation observer have been created, call () function if one exists call or set random method or attribute from the Mutation observer interface object.
createRangeTraversal,	creates a random treeWalker or nodeIterator and add reference to them in the createElements object array.
fuzzRangeTraversal,	if no range has been created call createRangeTraversal() function. else call or set random method or attribute from the respective NodeIterator Interface or TreeWalker Interface objects.
fuzzElementsMethods,	randomly set the attributes of a randomly selected element from the list of created elements.
fuzzElementsAttribs,	randomly call the methods of a randomly selected element from the list of created elements.
addLoop,	add a random loop function around js block loops are (for, while, setTimeout, setInterval)
crossRef,	try to set object references to a random other ex: HTML0 = HTML1.firstChild
AddEvent,	attach a random event to one of the created elements. creates an event object using createEvent directive and add reference to the created event object to a list for later use.
dispatchEvt,	randomly fires one of the created events.
intfuzz	return random function name for use ass callbacks for certain operations
GarColl,	force garbage collection.

Test Case generation – preparing the output script

- Creating Internal callbacks based on the number of function names returned by intfuzz()
- Insert the element creation JS block.
- Insert all other object create JS statements.
- Randomly insert JS statements returned by fuzz() function.

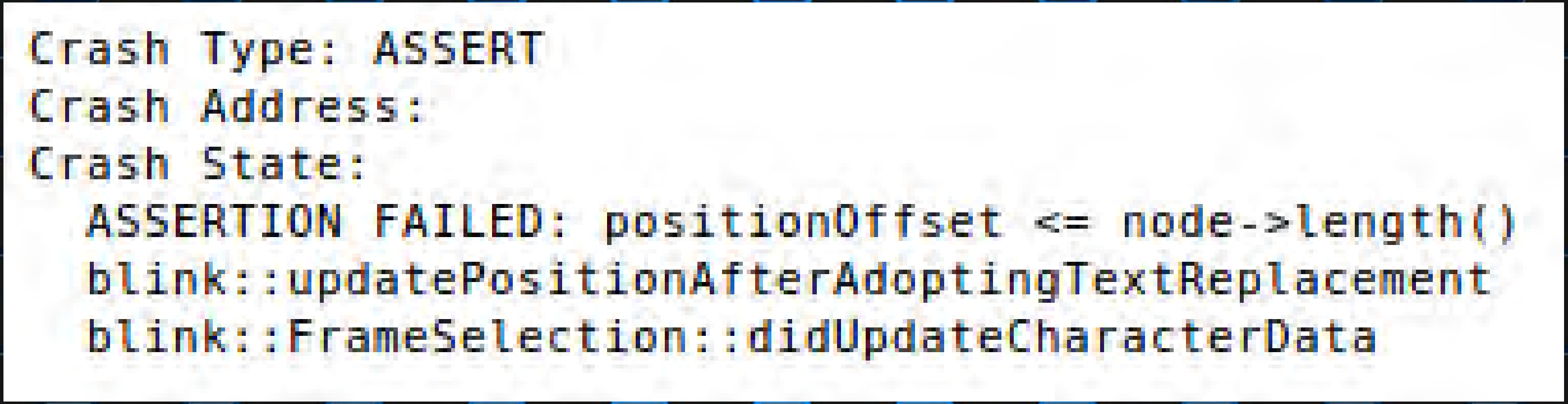
Sample Wadi Output

```
<html><head><style></style></head><body></body>
<script>
var createdElements={}
var createdElements={}
function func0(arg) {
try {window.addEventListener("select",func0,0)} catch(e) {}
try {document.styleSheets[0].insertRule("FONT,FONT {outline-offset: initial; }",0)} catch(e) {}
try {delete HTML0} catch(e) {}
try {window.status=""} catch(e) {}
}
try { HTML0=document.createElement("FONT")} catch(e) {}
try { HTML0.id="HTML0"} catch(e) {}
try { createdElements['HTML0']=HTML0} catch(e) {}
try { document.body.appendChild(HTML0)} catch(e) {}
try { TXT0=document.createTextNode('ä¤䇏4ᦵB£x□]□00fcihF‰f回{½-嘖j踏a2z2㡔6')} catch(e) {}
try { createdElements['TXT0']=TXT0} catch(e) {}
try { HTML0.appendChild(TXT0)} catch(e) {}
try { var EVT0= new Event("select",{"bubbles":0, "cacelable":0})} catch(e) {}
try { var ni0 = document.createNodeIterator(HTML0.firstChild,32,NodeFilter.FILTER_REJECT,1); createdElements['ni0']=ni0}
catch(e) {}
try {TXT0.lastChild=""} catch(e) {}
try {gc()} catch(e) {}
try {createdElements[HTML0].style.imageRendering="pixelated"} catch(e) {}
try {delete HTML0} catch(e) {}
try {window.status=""} catch(e) {}
</script> </html>
```

Results

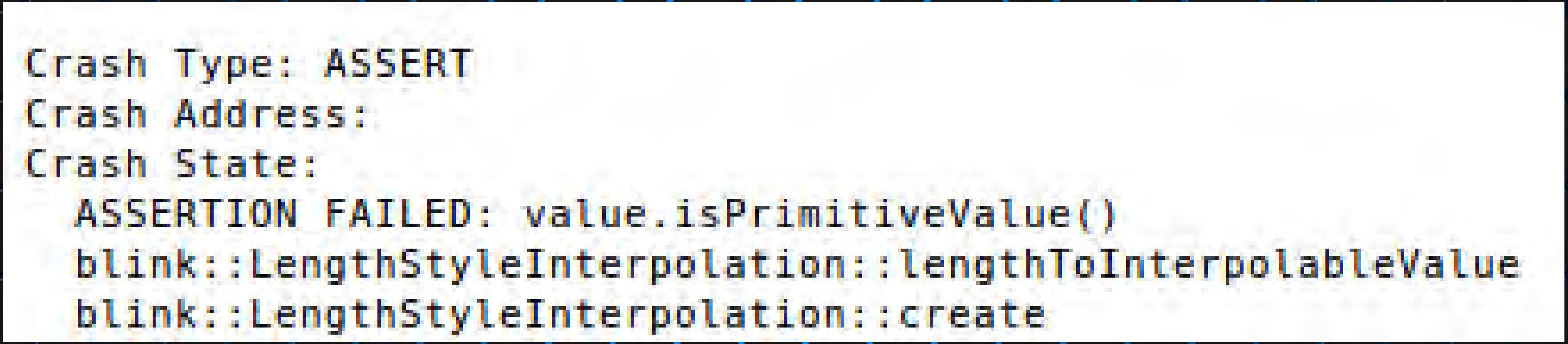
- Using Wadi we were able to find And report 4 confirmed bugs in latest Chromium ASan. version
- 2 Were Duplicates.
  - Issue No: 446517 Duplicate With issue 383777
  - Issue No: 445772 Duplicate with Issue: 445638
- 2 were confirmed with security severity high and affecting all OS. Fixed awaiting Release and hopefully reward :D
  - Issue No: 445332
  - Issue No: 453279

BUG#1 Issue No:446517



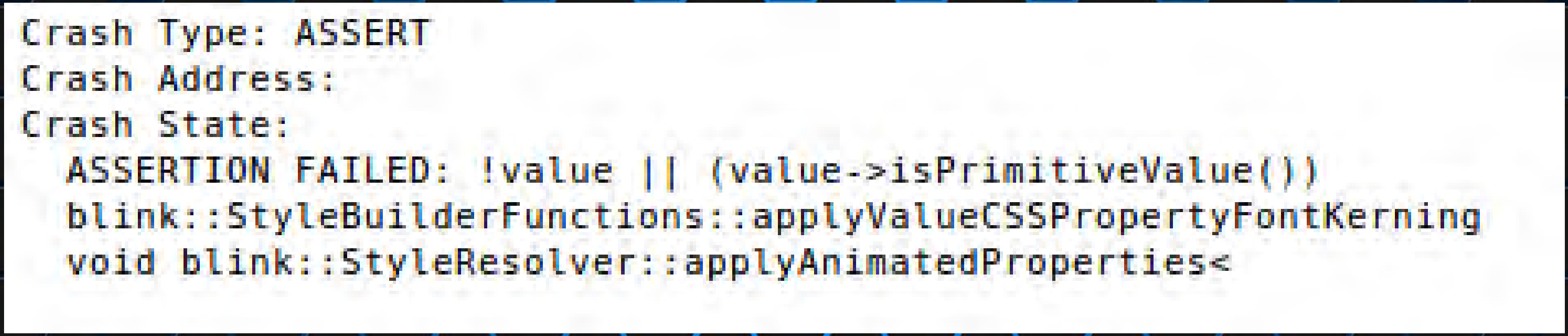
```
Crash Type: ASSERT
Crash Address:
Crash State:
  ASSERTION FAILED: positionOffset <= node->length()
  blink::updatePositionAfterAdoptingTextReplacement
  blink::FrameSelection::didUpdateCharacterData
```

BUG#2 Issue No:445772



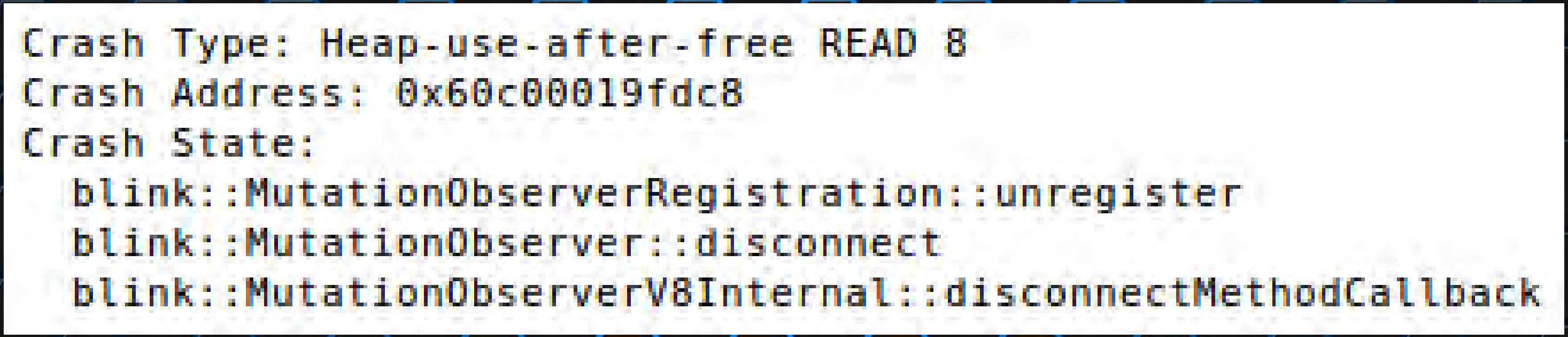
```
Crash Type: ASSERT
Crash Address:
Crash State:
  ASSERTION FAILED: value.isPrimitiveValue()
  blink::LengthStyleInterpolation::lengthToInterpolableValue
  blink::LengthStyleInterpolation::create
```

BUG#3 Issue No:445332



```
Crash Type: ASSERT
Crash Address:
Crash State:
  ASSERTION FAILED: !value || (value->isPrimitiveValue())
  blink::StyleBuilderFunctions::applyValueCSSPropertyFontKerning
  void blink::StyleResolver::applyAnimatedProperties<
```

BUG#4 Issue No:453279



```
Crash Type: Heap-use-after-free READ 8
Crash Address: 0x60c00019fdc8
Crash State:
  blink::MutationObserverRegistration::unregister
  blink::MutationObserver::disconnect
  blink::MutationObserverV8Internal::disconnectMethodCallback
```

References

- http://www.w3.org/DOM/
- http://www.w3.org/TR/WebIDL/
- Fuzzing Brute Force Vulnerability Discovery: Michael Sutton, Adam Greene, Michael Pedram Amini.
- Fuzzing for Software Security Testing and Quality Assurance: Ari Takanen, Jared D. Demott, Charlie Miller.
- Browser bug hunting - Memoirs of a last man standing: Atte Kettunen 44con talk.

Q&A