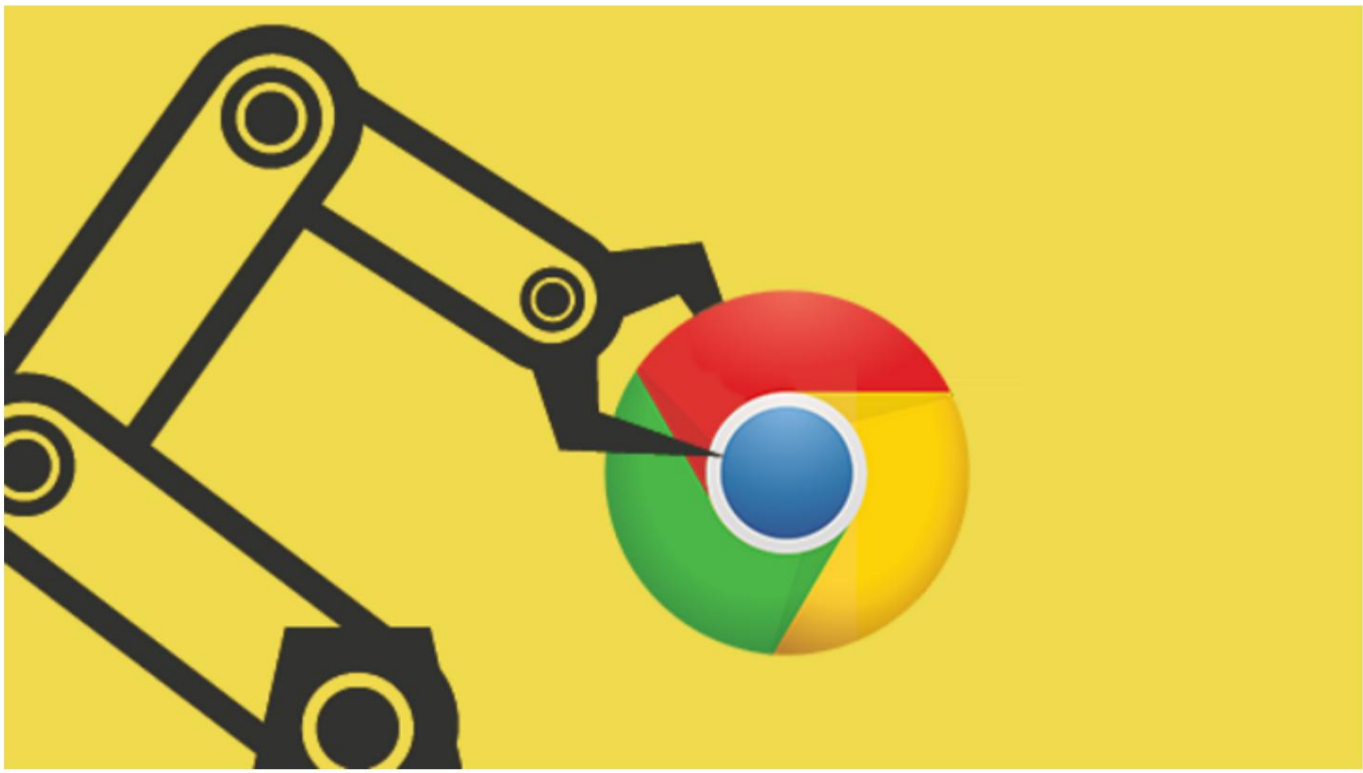
# "Chrome V8 Source Code" 34. Ultimate optimization technology Turbofan



## 1 abstract

Turbofan is an optimized compilation based on the Sea of Nodes theory. It is the last node among the three nodes of the V8 Compiler Pipeline. In addition There are also Ingintion and Sparkplug. Turbofan makes JavaScript execute faster, but it also requires more compilation time, so V8 only works with hot Point functions use Turbofan. This article will introduce the opening method, workflow and important data structure of Turbofan.

## 2 Turbofan

The test sample code is given below:

```
function addstring(a,b) {
        return a+b;
}
addstring("hello ","Turbofan!");
//separator line.....
Bytecode Age: 0
                00000043637A1DDE @    0:13 00 _              LdaConstant [0]
                00000043637A1DE0 @    2 :c2                  Star1
                00000043637A1DE1 @    3 : 19 fe f8 6 :       Mov <closure>, r2
                00000043637A1DE4 @    64 51 01 f9 02         CallRuntime[DeclareGlobals],
r1-r2

                00000043637A1DE9 @ 11 : 21 01 00             LdaGlobal [1], [0]
                00000043637A1DEC @ 14 : c2                   Star1
                00000043637A1DED @ 15 : 13 02                LdaConstant [2]
```

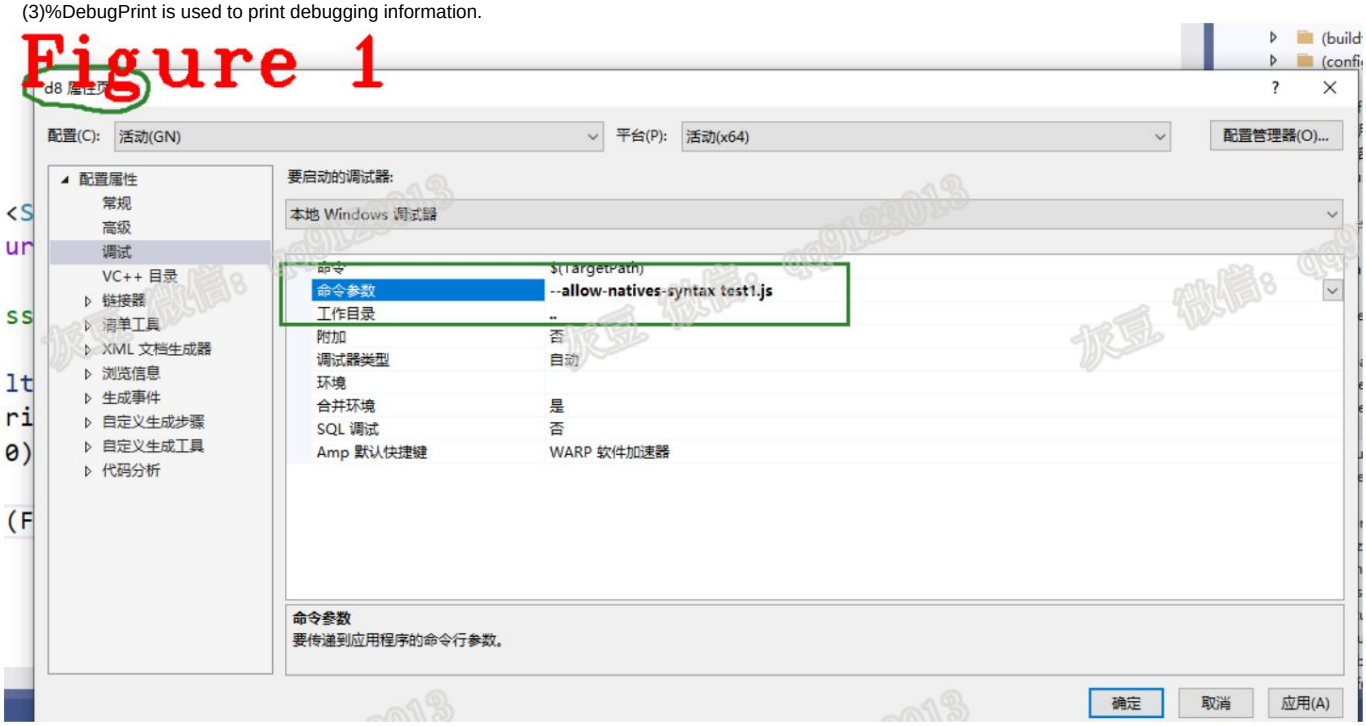
```
                00000043637A1DEF @ 17 : c1          Star2
                00000043637A1DF0 @ 18 : 13 03       LdaConstant [3]
                00000043637A1DF2 @ 20 : c0          Star3
                00000043637A1DF3 @ 21 : 62 f9 f8 f7 02  CallUndefinedReceiver2 r1,
r2, r3, [2]
                00000043637A1DF8 @ 26 : c3          Star0
                00000043637A1DF9 @ 27 : a8          Return
Constant pool (size = 4)
00000043637A1D79: [FixedArray] in OldSpace - map:
  0x01d0222812c1 <Map> - length: 4
  0:
     0x0043637a1d01 <FixedArray[2]> 1: 0x0043637a1c09
                <String[9]: #addstring> 2: 0x004363 7a1c29 <String[6]:
                #hello > 3: 0x0043637a1c41 <String[9]: #Turbofan!>
```

In the above code, the three instructions LdaGlobal [1], [0], LdaConstant [2] and LdaConstant [3] obtain the addstring() function and two parameters respectively; CallUndefinedReceiver2 calls the addstring() function to complete the string addition. Only hot functions will start Turbofan. In order to facilitate learning, you need to use the %OptimizeFunctionOnNextCall() and --allow-natives-syntax instructions to actively trigger Turbofan.

(1) %OptimizeFunctionOnNextCall() is used to start optimization compilation the next time it is called; (2) --allow-natives-syntax allows d8 to accept natives instructions. Figure 1 shows how to use it.

(3)%DebugPrint is used to print debugging information.

Figure 1




Make some changes to the test code, the source code is as follows:

```
1. function addstring(a,b) { return a+b; 2.

3. } 4.
console.log(addstring("hello ","Turbofan!")); 5. %DebugPrint(addstring);
6. %OptimizeFunctionOnNextCall(addstring);
7. addstring("Speculative ", "Optimization") ; 8.
%DebugPrint(addstring);
```

The above 4th line of code will generate Feedback after execution; the 5th line of code uses Turbofan to optimize addstring; the core of Turbofan optimization mechanism The idea is speculation, that is, it will save a lot of unnecessary operations to improve performance. In short, Turbofan believes that addstring is only used for adding strings. method operation, it is expected that the two parameters passed in the next time addstring is called will still be strings, so that many unnecessary operations can be saved. implement The information printed after the 5th line of code is as follows:

```
1. hello Turbofan!
2. DebugPrint: 000000F3196E1FA1: [Function] in OldSpace
3.      - map: 0x00ae91d413a1 <Map(HOLEY_ELEMENTS)> [FastProperties]
4.      - prototype: 0x00f3196c3ee9 <JSFunction (sfi = 000001CFDCE91269)>
5.      - elements: 0x03e492c81309 <FixedArray[0]> [HOLEY_ELEMENTS]
6.      - function prototype:
7.      - initial_map:
8.      - shared_info: 0x00f3196e1db1 <SharedFunctionInfo addstring>
9.      - name: 0x00f3196e1c09 <String[9]: #addstring>
10.       - builtin: InterpreterEntryTrampoline
11.       - formal_parameter_count: 2
12.       - kind: NormalFunction
13.       - context: 0x00f3196c34a9 <NativeContext[262]>
14.       - code: 0x021668546441 <Code BUILTIN InterpreterEntryTrampoline>
15.       - interpreted
16.       - bytecode: 0x00f3196e20c1 <BytecodeArray[6]>
17.       - source code: (a,b) {
18.           return a+b;
19. }
20. //.....omitted.....
twenty one.       - feedback vector: No feedback vector, but we have a closure feedback cell
array
22. 000003E492C82FB9: [ClosureFeedbackCellArray] in ReadOnlySpace
twenty three.       - map: 0x03e492c81f19 <Map>
twenty four.       - length: 0
25. 000000AE91D413A1: [Map]
26. - type: JS_FUNCTION_TYPE
27.       - instance size: 64
28.       - inobject properties: 0
29.       - elements kind: HOLEY_ELEMENTS
30.       - unused property fields: 0
31.       - enum length: invalid
32.       -stable_map
33.       - callable
34.       - constructor
35.       - has_prototype_slot
36.       - back pointer: 0x03e492c81599 <undefined>
37.       - prototype_validity cell: 0x01cfdce84a09 <Cell value= 1>
38.       - instance descriptors (own) #5: 0x00f3196c4041 <DescriptorArray[5]>
39.       - prototype: 0x00f3196c3ee9 <JSFunction (sfi = 000001CFDCE91269)>
40.       - constructor: 0x00f3196c3fd9 <JSFunction Function (sfi = 000001CFDCE91409)>
41.       - dependent code: 0x03e492c81239 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
42.       - construction counter: 0
```

The above information is the debug output before executing Turbofan. Line 1 is the running result; line 8 is SharedFunctionInfo; line 10

The explanation startup mode is InterpreterEntryTrampoline, that is, interpretation execution; line 14 is the entrance to InterpreterEntryTrampoline

Address; lines 23-42 give the map information of addstring.

The following is the debug information after executing Turbofan:

```
1. Speculative Optimization
2. DebugPrint: 000000F3196E1FA1: [Function] in OldSpace
3.      - map: 0x00ae91d413a1 <Map(HOLEY_ELEMENTS)> [FastProperties]
4.      - prototype: 0x00f3196c3ee9 <JSFunction (sfi = 000001CFDCE91269)>
5.      - elements: 0x03e492c81309 <FixedArray[0]> [HOLEY_ELEMENTS]
6.      - function prototype:
7.      - initial_map:
8.      - shared_info: 0x00f3196e1db1 <SharedFunctionInfo addstring>
9.      - name: 0x00f3196e1c09 <String[9]: #addstring>
10.       - formal_parameter_count: 2
11.       - kind: NormalFunction
12.       - context: 0x00f3196c34a9 <NativeContext[262]>
13.       - code: 0x021668583001 <Code TURBOFAN>
14.       - interpreted
15.       - bytecode: 0x00f3196e20c1 <BytecodeArray[6]>
16.       - source code: (a,b) {
17.             return a+b;
18. }
19. - properties: 0x03e492c81309 <FixedArray[0]>
20.       - feedback vector: 000000F3196E2119: [FeedbackVector] in OldSpace
twenty one.       - map: 0x03e492c81b69 <Map>
twenty two.       - length: 1
twenty three.       - shared function info: 0x00f3196e1db1 <SharedFunctionInfo addstring>
twenty four.       - no optimized code
25.       - optimization marker: OptimizationMarker::kNone
26.       - optimization tier: OptimizationTier::kNone
27.       - invocation count: 0
28.       - profiler ticks: 0
29.       - closure feedback cell array: 000003E492C82FB9: [ClosureFeedbackCellArray]
in ReadOnlySpace
30. - map: 0x03e492c81f19 <Map>
31.       - length: 0
32.       - slot #0 BinaryOp BinaryOp:String {
33.             [0]: 16
34.         }
35. 000000AE91D413A1: [Map]
36.       - type: JS_FUNCTION_TYPE
37.       - instance size: 64
38.       - inobject properties: 0
39.       - elements kind: HOLEY_ELEMENTS
40.       - unused property fields: 0
41.       - enum length: invalid
42.       -stable_map
43.       - callable
44.       - constructor
45.       - has_prototype_slot
46.       - back pointer: 0x03e492c81599 <undefined>
```

```
47.     - prototype_validity cell: 0x01cfdce84a09 <Cell value= 1>
48.     - instance descriptors (own) #5: 0x00f3196c4041 <DescriptorArray[5]>
49.     - prototype: 0x00f3196c3ee9 <JSFunction (sfi = 000001CFDCE91269)>
50.     - constructor: 0x00f3196c3fd9 <JSFunction Function (sfi = 000001CFDCE91409)>
51.     - dependent code: 0x03e492c81239 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
52.     - construction counter: 0
```

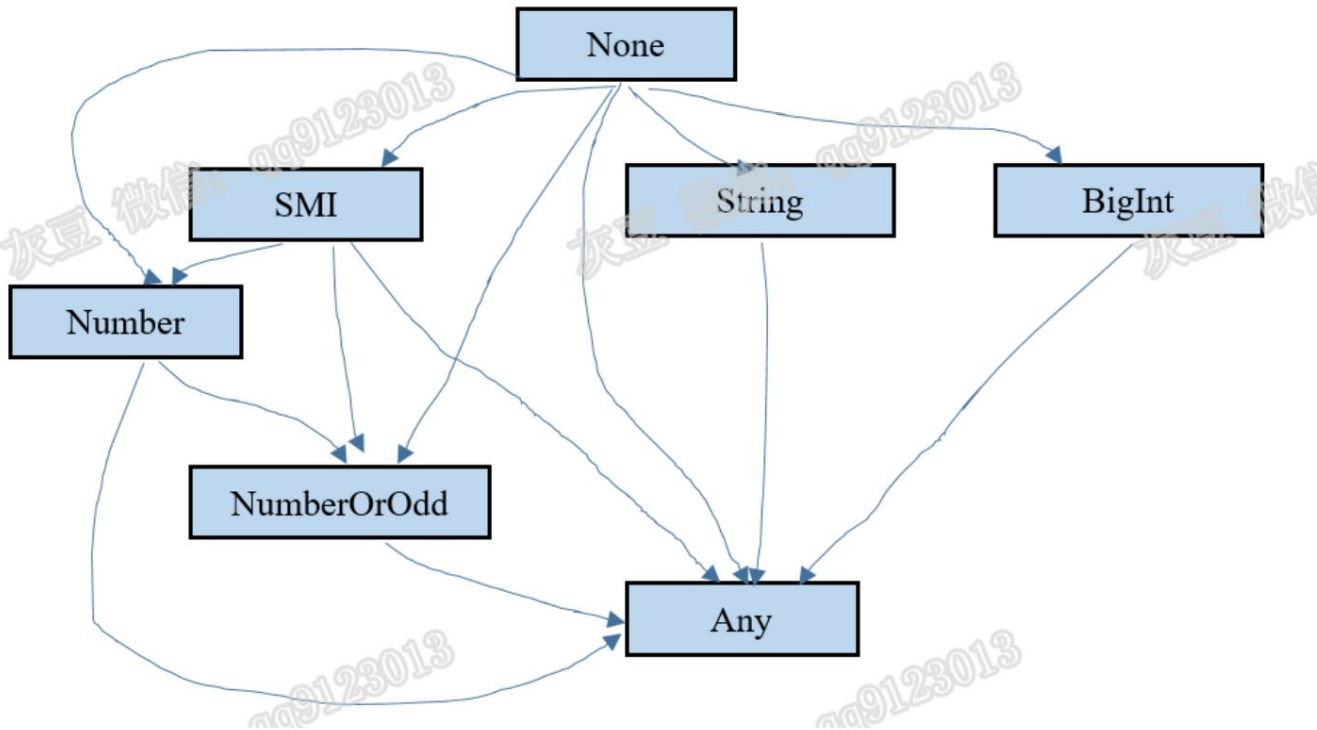
The above information is the debug output after executing Turbofan. The first line is the output result; the difference from the debug output before executing Turbofan yes:

**(1)** The information on line 13 indicates that the operating mode is Code TURBOFAN, not InterpreterEntryTrampoline.

**(2)** The information in line 32 indicates that the current feedback vector slot operates on strings, that is, optimized for string operations.

Feedback is a mechanism for collecting runtime information. It is called vector because it is implemented using C++ vector. feedback is available It can be used to guide the inline cache to store relevant information, and can also be used to guide Turbofan work. According to JavaScript data types, we can The feedback status can be roughly calculated as shown in Figure 2.

Figure 2




When V8 starts executing, feedback is None, because we have nothing at this time and cannot see any information. Since addstring is often used for characters String addition, so after running for a period of time, the feedback becomes String. At this time, Turbofan can be used to optimize string addition. along with The program's running feedback becomes Any again, which means addstring is often used to add any data. Pay attention to the status of feedback Generally, it is converted from top to bottom. If the conversion is reversed, de-optimization may occur.

This article uses %OptimizeFunctionOnNextCall() to actively evoke Turbofan. The source code is as follows:

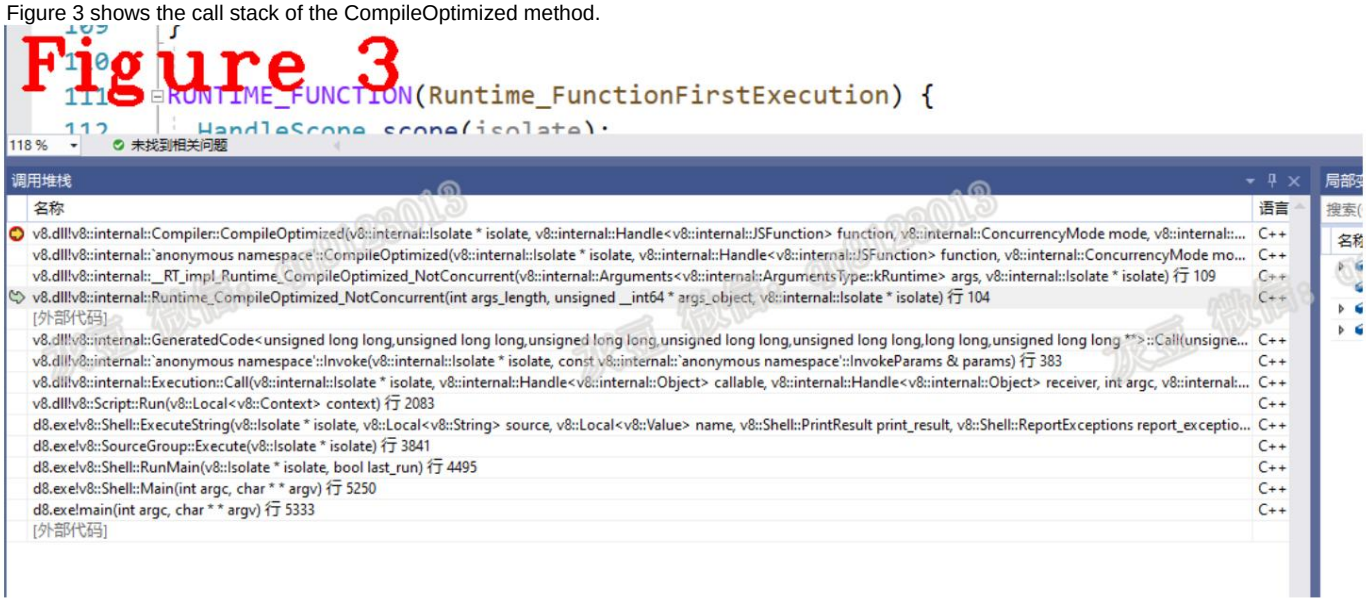
```
1. RUNTIME_FUNCTION(Runtime_OptimizeFunctionOnNextCall) {
2.       HandleScope scope(isolate);
3.       return OptimizeFunctionOnNextCall(args, isolate,
TierupKind::kTierupBytecode);
4. }
5. //.....separator line.....
6. bool Compiler::CompileOptimized(Isolate* isolate, Handle<JSFunction> function,
7.                                  ConcurrencyMode mode, CodeKind code_kind) {
```

```
8.      DCHECK(CodeKindIsOptimizedJSFunction(code_kind));
9.      DCHECK(AllowCompilation::IsAllowed(isolate));
10.       if (FLAG_stress_concurrent_inlining &&
11.             isolate->concurrent_recompilation_enabled() &&
12.             mode == ConcurrencyMode::kNotConcurrent &&
13.             isolate->node_observer() == nullptr) {
14.           SpawnDuplicateConcurrentJobForStressTesting(isolate, function, mode,
15.                                                       code_kind);
16.       }
17.       Handle<Code> code;
18.       if (!GetOptimizedCode(isolate, function, mode, code_kind).ToHandle(&code))
{
19.           // Optimization failed, get the existing code. We could have optimized
code
20.           // from a lower tier here. Unoptimized code must exist already if we are
twenty one.   // optimizing.
twenty two.   DCHECK(!isolate->has_pending_exception());
twenty three. DCHECK(function->shared().is_compiled());
twenty four.  DCHECK(function->shared().IsInterpreted());
25.           code = ContinuationForConcurrentOptimization(isolate, function);
26.       }
27.       function->set_code(*code, kReleaseStore);
28.       // Check postconditions on success.
29.       return true;
30. }
```

The above is two parts of code: Runtime_OptimizeFunctionOnNextCall sets the enable flag of Turbofan to true. After setting the enable flag

The Runtime_CompileOptimized_NotConcurrent method will be called, in which CompileOptimized in the above code is called.

Figure 3 shows the call stack of the CompileOptimized method.




**Technical summary**

**(1)** Native instructions can be used in JavaScript only when the --allow-natives-syntax option is used;

**(2)** Reverse transformation of feedback status may lead to de-optimization.

Okay, that's it for today, see you next time.

**Personal abilities are limited and there are shortcomings and mistakes. Criticisms and corrections are welcome.**

**WeChat: qq9123013 Note: v8 Exchange Zhihu: https://www.zhihu.com/people/v8blink**

This article was originally

published by Huidou and reprinted from: https://www.anquanke.com/post/id/

262579 Anquanke - Thoughtful new security media