Dachshund

Digging for and Securing Against (Non-)Blinded Constants in JIT Code

Giorgi Maisuradze, Michael Backes, Christian Rossow

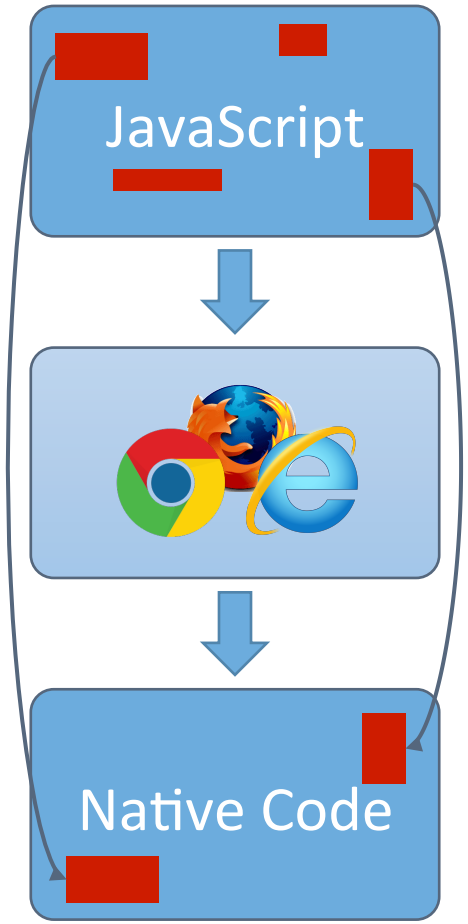
CISPA, Saarland University, Germany

NDSS Symposium 2017

CISPA Center for IT-Security, Privacy and Accountability

Overview

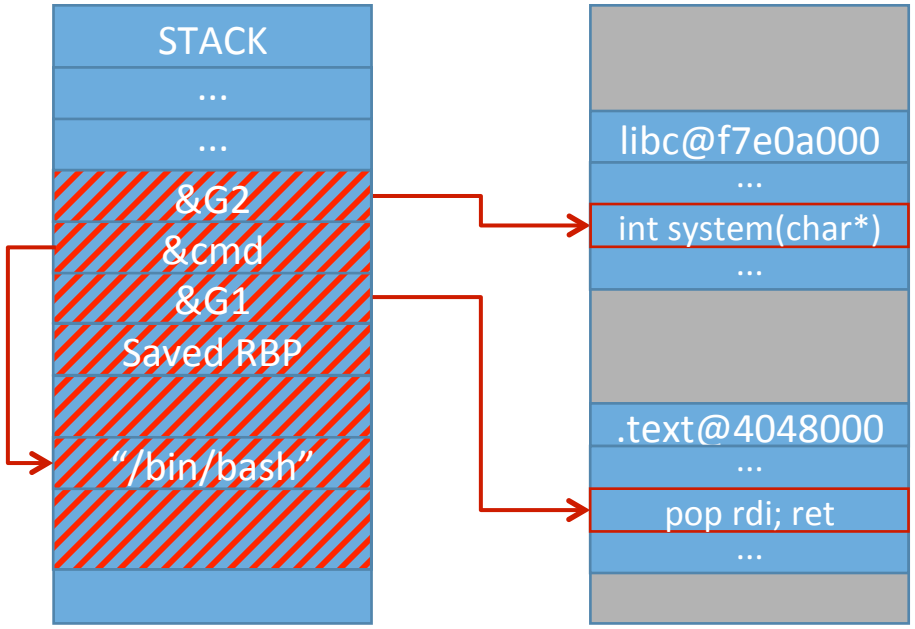
- Code reuse attacks
- Code reuse with JIT code
- Constant (non-)blinding in browsers
- Defending nonblinded cases



Revisiting Code Reuse

Code Reuse Attacks

- Identify gadgets/functions
- Put their addresses on the stack
- Use return instructions to execute them

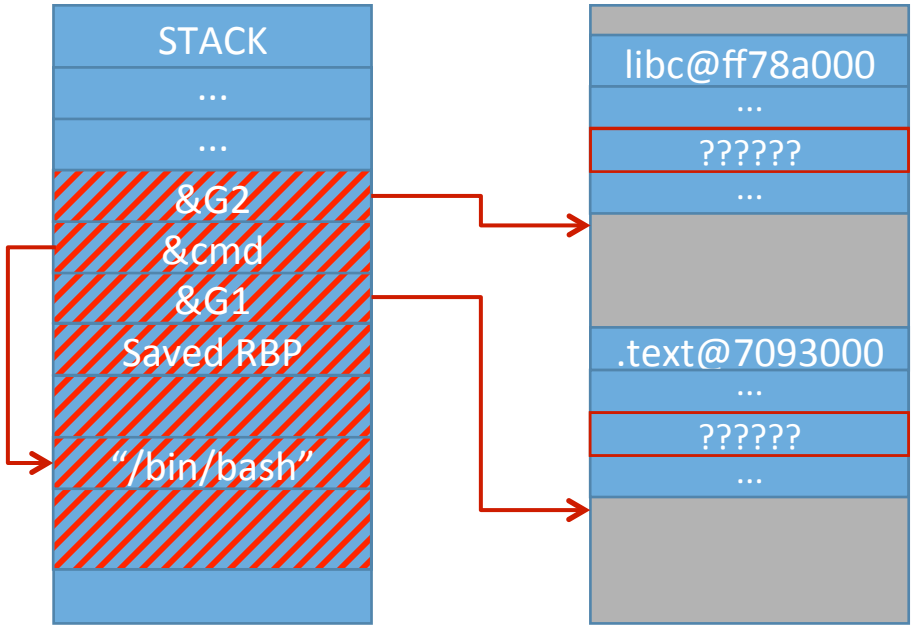


Code Reuse Defenses

- Identify gadgets/functions
- Put their addresses on the stack
- Use return instructions to execute them

Defense:

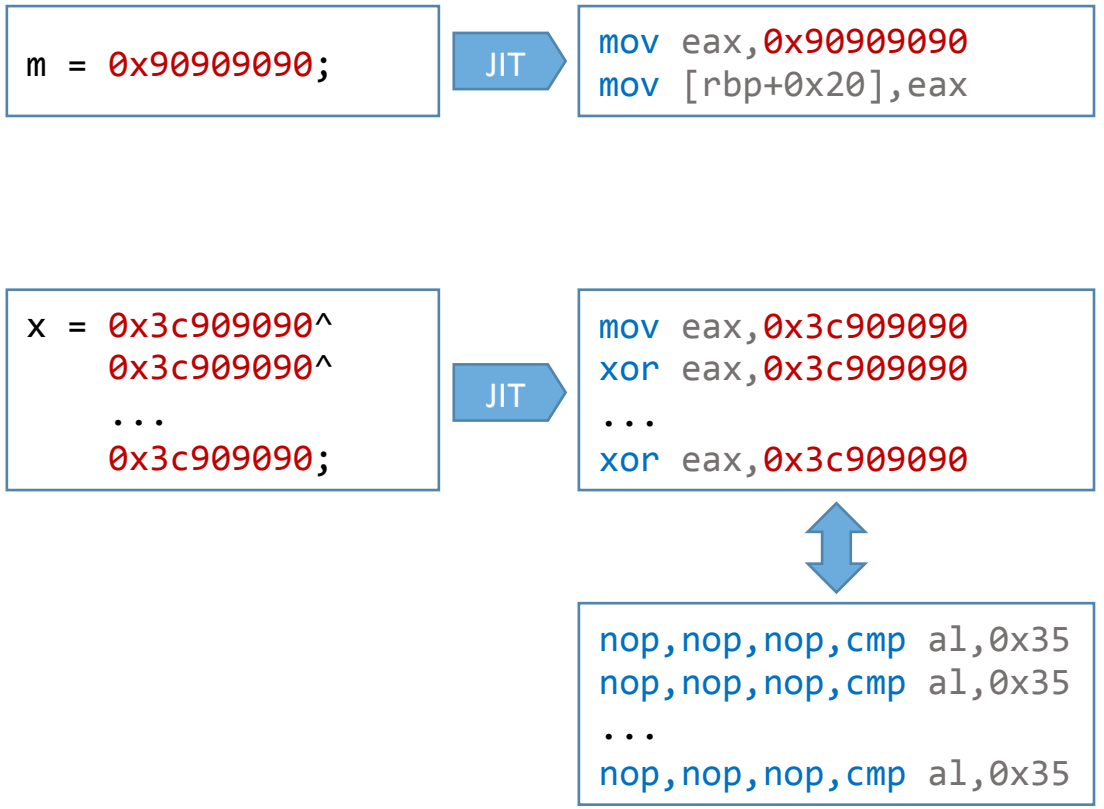
- Randomize memory segments (ASLR)
- Randomize code pages



Code Injection Attacks

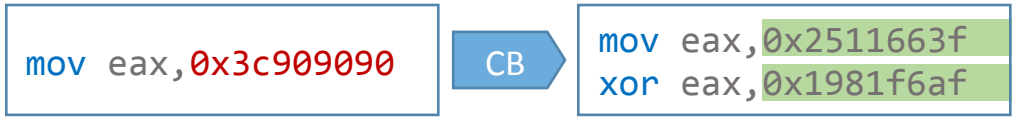
- Use constant values to create controlled gadgets

- JIT Spraying [WOOT'10]
  - Spray code pages with NOP-sled followed by a shellcode



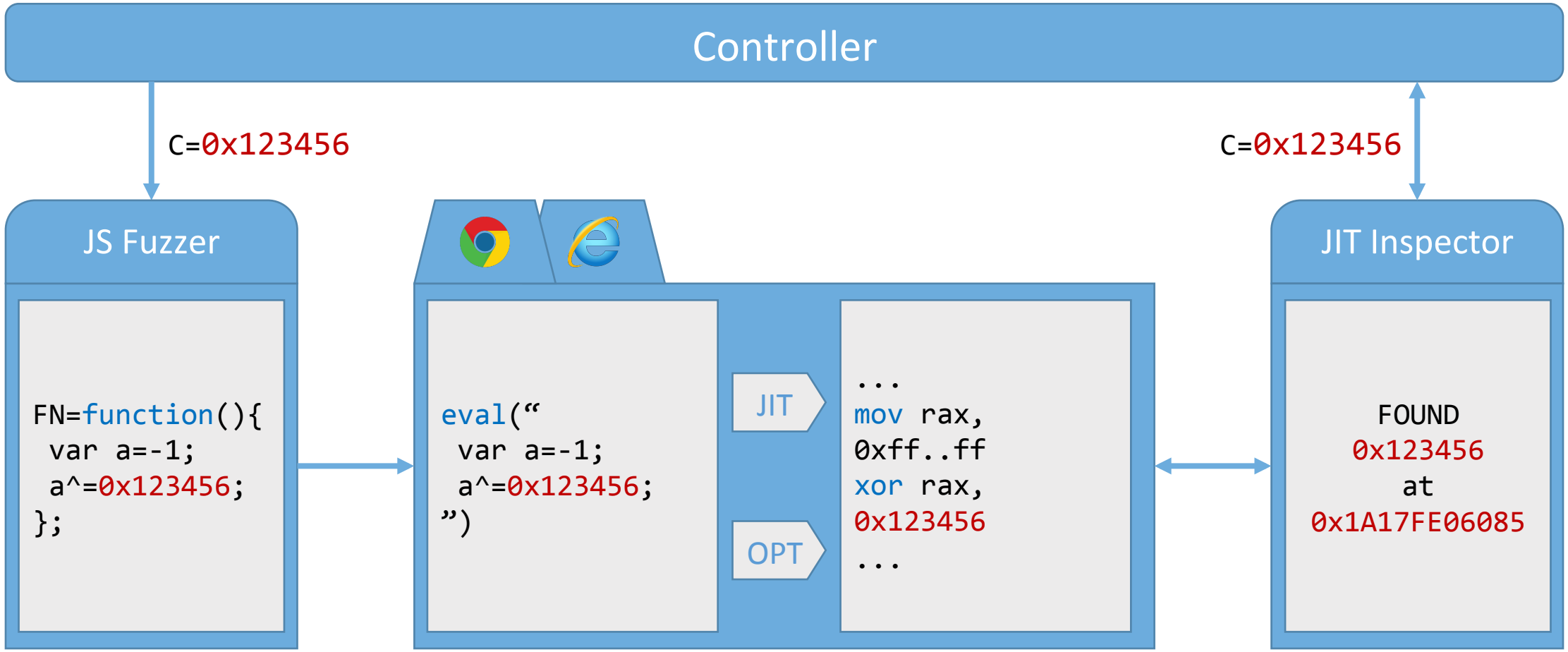
Browser Defenses

- Constant Blinding (Edge, Chrome)
  - Randomize immediate values by XORing them with a random key
  - Add XOR instruction to restore original value
  - Blind large constants (>2B)
- Weaknesses
  - Small constants remain [NDSS'15]
  - Displacement fields [Usenix'16]



Constant Blinding Completeness

Dachshund


Controller
C=0x123456
C=0x123456
JS Fuzzer
FN=function(){
var a=-1;
a^=0x123456;
};
eval("
var a=-1;
a^=0x123456;
")
JIT
OPT
...
mov rax,
0xff..ff
xor rax,
0x123456
...
JIT Inspector
FOUND
0x123456
at
0x1A17FE06085

Found Constants

JavaScript Statements Containing Nonblinded Constants:

- Chrome:
  - Arguments to built-in functions, ternary operators, return statements, bitwise operations, ...
- Edge:
  - Arguments to Math library functions, cases in switch statement, array indexes, global variable accesses, ...

```
function fn() {
 console.log(
        0x12345678);
}
```

```
function fn() {
 return Math.trunc(
        0x12345678);
}
```

```
v1 = b ? 0x12345678:
         0x9abcdef0;
```

```
switch(j) {
 case 0x12345678: m++;
}
```

```
return 0x12345678;
```

```
arr[i] = 0x12345678;
```

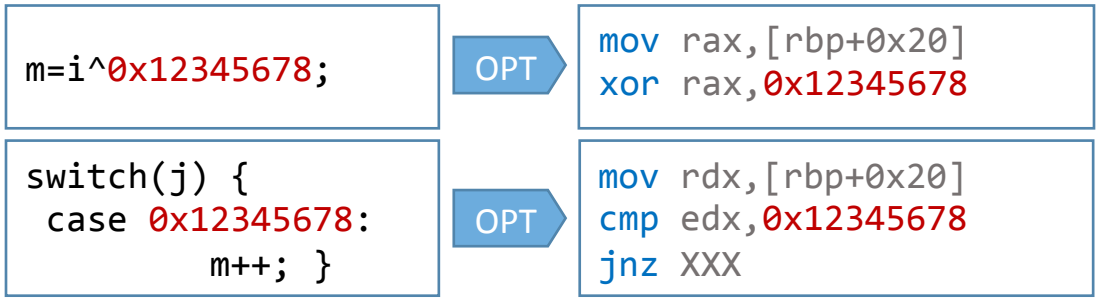
```
v1 = v2 ^ 0x12345678;
```

```
global = 0x12345678;
```

Origins of Constants

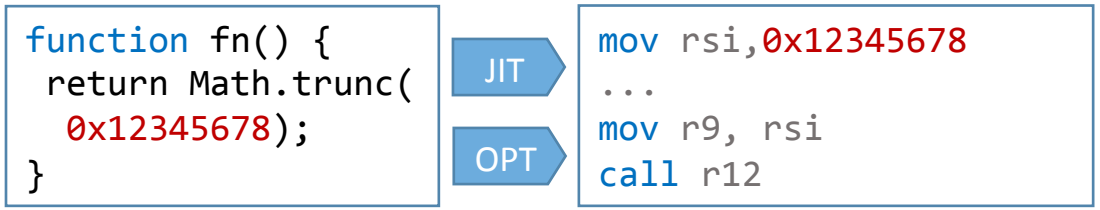
Chrome:

- Non blinded values are coming from the optimizing compiler:
  - conditional *?:*, switch, arithmetic, array indexing, globals,...



Edge:

- Immediate value caching (both in baseline and optimizer)
  - Nonblinded values are stored in a spare register



Generating Gadgets

Required gadgets for setting parameters for *VirtualProtect*:

- Google Chrome 50:
  - Create the function *fn*
  - Trigger optimizing compiler (*>1000 calls*)
- Microsoft Edge 25:
  - Create functions *r8, r9* and *racdx*
  - Trigger baseline compilation for each of them (*>50 calls*)

```
pop  r8; ret  4158c3
pop  r9; ret  4159c3
pop rcx; ret    59c3
pop rdx; ret    5ac3
pop rax; ret    58c3
```

```
function fn(){
 glob[0]= 0xc35841;
 glob[1]= 0xc35941;
 glob[2]=-0x3ca7a5a7;
}
```

```
mov [rbx+0x1b],0x00c35841
mov [rbx+0x23],0x00c35941
mov [rbx+0x2b],0xc3585a59
```

```
function r8(){
 Math.trunc(0xc35841); }
function r9(){
 Math.trunc(0xc35941); }
function racdx(){
 Math.trunc(-0x3ca7a5a7);
}
```

```
mov rsi,0x00c35841
mov rsi,0x00c35941
mov rsi,0xc3585a59
```

Blinding the Constants

Rewriting JavaScript

- Replace all integer constants with global objects

```
function fn(){
 var i=0x1234;
}
```

```
window._c1234=parseInt("0x1234")
function fn(){
 var i=window._c1234;
}
```

- Replace any other literal type that can be interpreted as a number

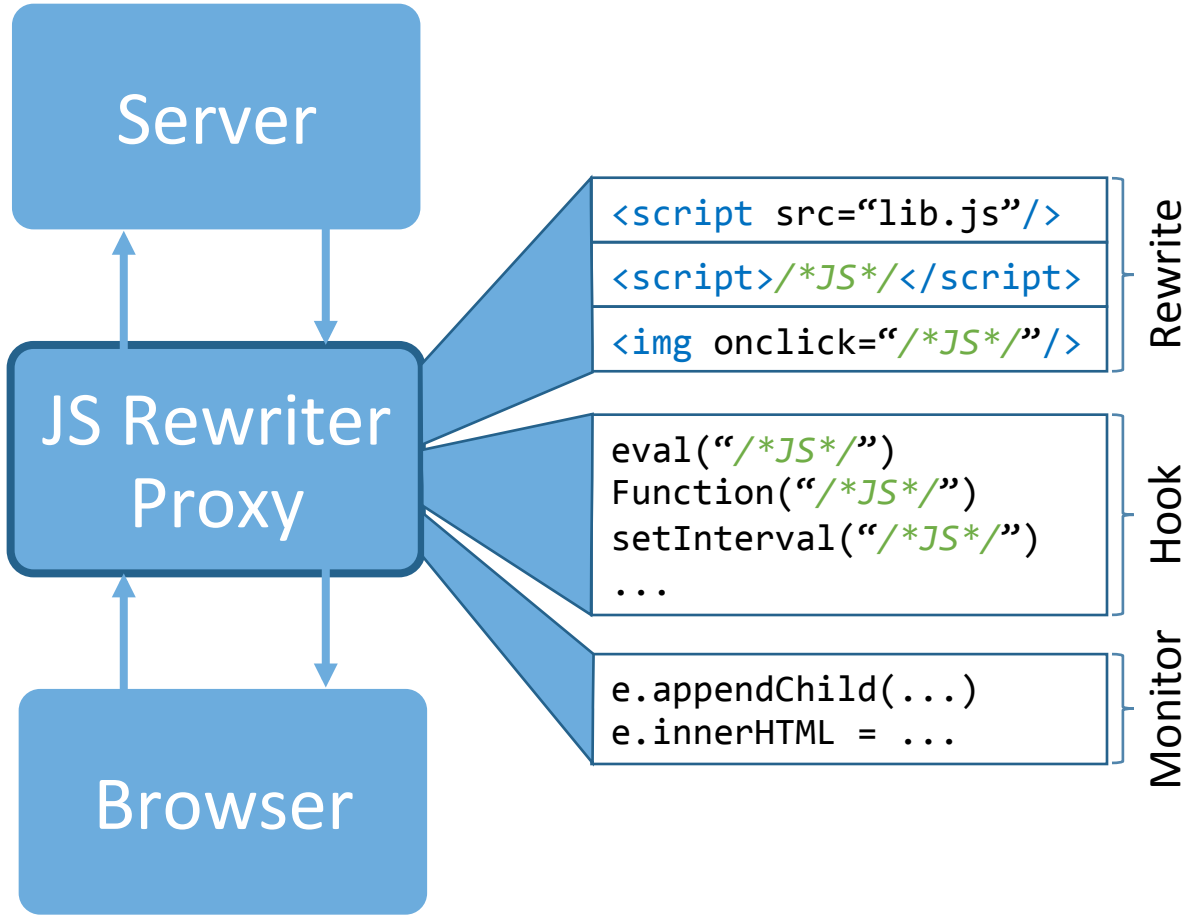
```
function fn(){
 i="1234"&567;
}
```

```
function fn(){
 i=("1234").toString()&567;
}
```

Rewriting JavaScript

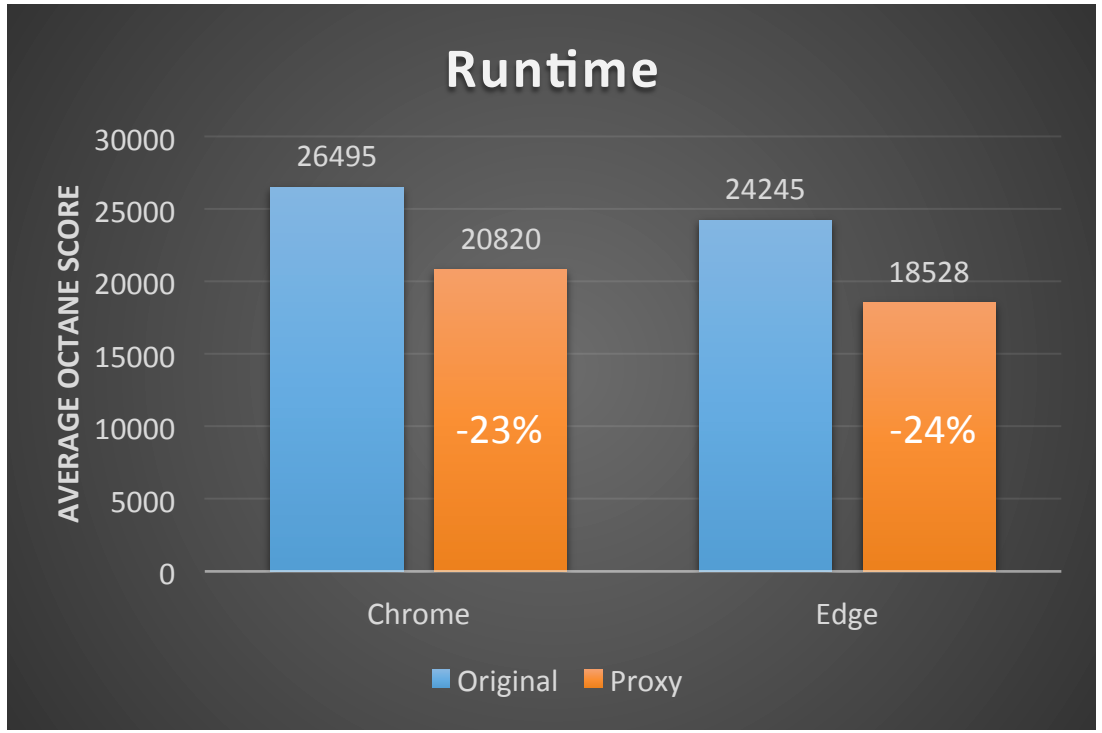
JavaScript rewriter is implemented as a proxy service between the browser and a webserver

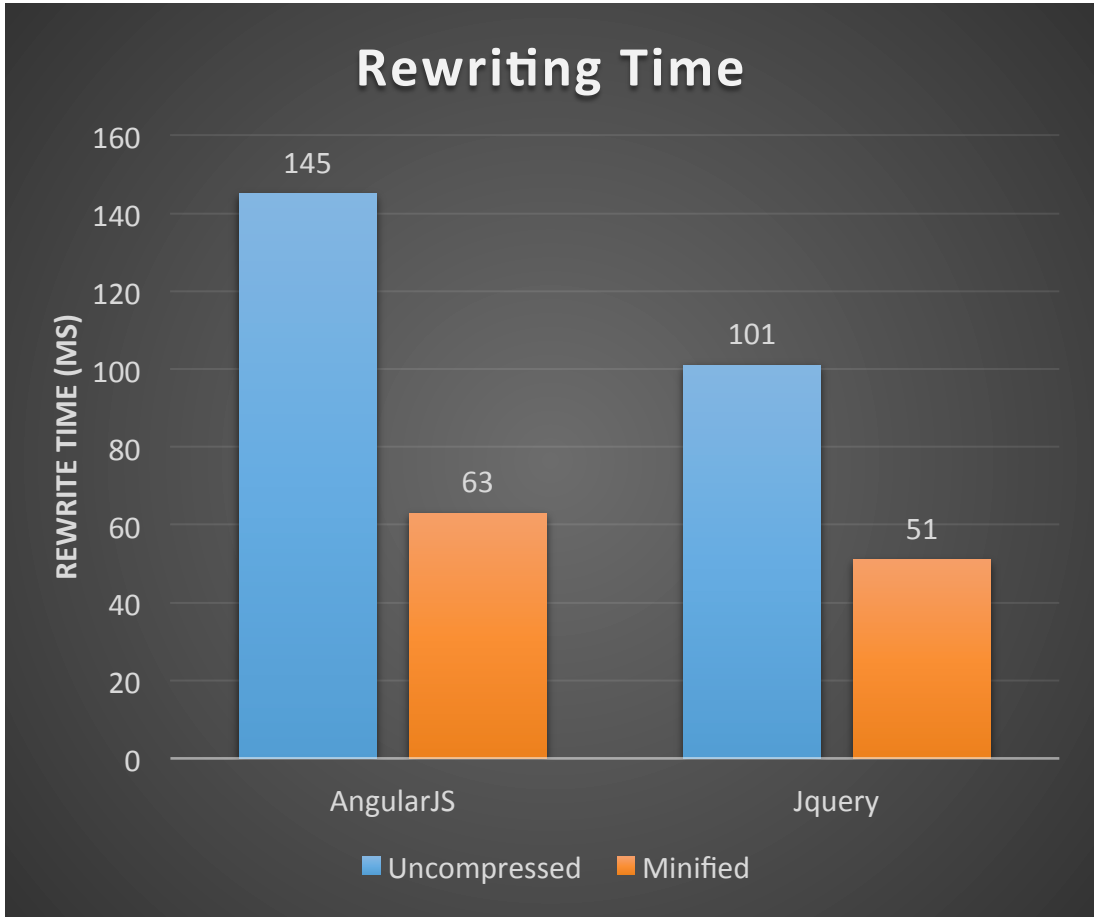
- Rewrite JavaScript in all possible places
- Hook dynamic functions to rewrite new code at runtime
- Rewrite JS in dynamically added DOM nodes



Evaluation

Rewriting got rid of all integer constants found by Dachshund





Summary

- JIT engines are vulnerable to code injection attacks
- Modern browsers do not sufficiently defend against them
- Rewriting JavaScript can get rid of code injection via immediate values

Thank you!