# 1 Artifacts

This document describes the information for reproducing the gadget insertion mentioned in the paper, and the prototype implementation of JITShield. We will open our project to the community after the anonymous blind review.

## 1.1 Gadget Insertion

In this section, we provide the artifact materials for the gadgets generation. The gadgets can be reproduced when following the instructions below.

1. Get the source code of V8.

```
git clone https://chromium.googlesource.com/chromium/tools/
    depot_tools.git
export PATH=/path/to/depot_tools:$PATH
mkdir ~/v8
cd ~/v8
fetch v8
cd v8
```

2. Checkout the corresponding version and compile it. It takes about 15 minutes for the V8 compilation.

```
git checkout d5662577fe3c847fbd1d61fc7e6871527efe9f06
tools/dev/gm.py x64.release d8
```

3. Running V8 with the template files as input, using the parameter –print-opt-code to show the generated code by v8. (The template files are provided as sibtemp_58c3.js, which can produce the gadget `0x58c3` with *SIB* byte manipulation.)

```
./out/x64.release/d8 --print-opt-code sibtemp_58c3.js > sibres.txt
```

4. The gadgets can be found in the text sibres.txt. The results may look like as follow. We can see *58c3* in a *lea* instruction.

```
0xada00084127    e7  03d1          addl rdx,rcx
0xada00084129    e9  41d1f9        sarl r9, 1
0xada0008412c    ec  4183e013      andl r8,0x13
0xada00084130    f0  03d7          addl rdx,rdi
0xada00084132    f2  41d1fb        sarl r11, 1
0xada00084135    f5  4183e114      andl r9,0x14
0xada00084139    f9  4103d0        addl rdx,r8
0xada0008413c    fc  41d1fc        sarl r12, 1
0xada0008413f    ff  4183e315      andl r11,0x15
0xada00084143   103  4103d1        addl rdx,r9
0xada00084146   106  d1fb          sarl rbx, 1
0xada00084148   108  41d1fe        sarl r14, 1
0xada0008414b   10b  4183e416      andl r12,0x16
0xada0008414f   10f  4103d3        addl rdx,r11
0xada00084152   112  83e320        andl rbx,0x20
0xada00084155   115  d1f8          sarl rax, 1
0xada00084157   117  41d1ff        sarl r15, 1
0xada0008415a   11a  4183e617      andl r14,0x17
0xada0008415e   11e  4401e2        addl rdx,r12
0xada00084161   121  83e019        andl rax,0x19
0xada00084164   124  d1fe          sarl rsi, 1
0xada00084166   126  4183e718      andl r15,0x18
0xada0008416a   12a  4103d6        addl rdx,r14
0xada0008416d   12d  83e621        andl rsi,0x21
0xada00084170   130  4103d7        addl rdx,r15
0xada00084173   133  8d4c58c3      leal rcx,[rax+rbx*2-0x3d]
0xada00084177   137  03d6          addl rdx,rsi
0xada00084179   139  03d1          addl rdx,rcx
0xada0008417b   13b  8d0412        leal rax,[rdx+rdx*1]
0xada0008417e   13e  488b4de8      REX.W movq rcx,[rbp-0x18]
```

## 1.2 JITShield

In this section, we provide the prototype implementation of JITShield based on V8. JITShield is evaluated with JetStream2.

1. Get the source code of V8.

```
git clone https://chromium.googlesource.com/chromium/tools/
    depot_tools.git
export PATH=/path/to/depot_tools:$PATH
mkdir ~/v8
cd ~/v8
fetch v8
cd v8
```

2. Build the native v8 as baseline.

```
git checkout d5662577fe3c847fbd1d61fc7e6871527efe9f06
tools/dev/gm.py x64.release d8
mv out/x64.release out/baseline
```

2. Merge the JITShield patch provided in the artifact materials (i.e. JITShield.patch) and compile it.

```
git apply JITShield.patch
tools/dev/gm.py x64.release d8
mv out/x64.release out/jitshield
```

3. Get the source code of JetStream2 and run the benchmark. The patch is requested for the slightly modification for the benchmark mentioned in the paper, which repeatedly execute the JavaScript code for 100 times after loaded. Please note that it takes about 10 hours to finish the experiments.

```
git clone https://github.com/WebKit/WebKit.git ~/WebKit
cd WebKit
git checkout 4693cb2fa3113a0e5cc91bb8cda3b06a4abff58e
git apply JetStream2.patch
cd PerformanceTests/JetStream2
python3 jetdriver.py ~/v8/v8/out/baseline ~/v8/v8/out/jitshield
python3 process.py
```

4. Screenshot of benchmarking results. The entire experiment results are stored in the file named as "res.txt". By executing the process.py as above, the rough result will be shown as follows.

| benchmark | baseline | JITShield | normalized JITShield |
|---|---|---|---|
| 3d-cube-SP | 1.1923 | 1.2024 | 1.0085 |
| 3d-raytrace-SP | 1.1549 | 1.1734 | 1.0160 |
| acorn-wtb | 1.0256 | 1.0340 | 1.0082 |
| ai-astar | 0.9448 | 0.9852 | 1.0427 |
| ..... | | | |
| typescript | 4.1780 | 4.2221 | 1.0105 |
| uglify-js-wtb | 1.3987 | 1.4108 | 1.0086 |
| UniPoker | 1.6576 | 1.6639 | 1.0038 |
| geomean | 1.1830 | 1.1894 | 1.0054 |