Native Client

A Sandbox for Portable, Untrusted x86 Native Code

Outline

- Problem
- What is NaCl (Demo)
- Related work
- Solution
- Attack Surface(First Attack Demo)
- Evaluation
- TUCAO

Before NaCl

Sum all numbers between 1 and 10000000

C

Golang

JS + V8

python

```
(venv)Karlheinzs-MacBook-Pro:benchmarks Karl$ gcc -O3 loop_sum.c -o loop_sum -march=native
(venv)Karlheinzs-MacBook-Pro:benchmarks Karl$ time ./loop_sum
sum: 49999995000000

real    0m0.006s
user    0m0.001s
sys     0m0.002s
(venv)Karlheinzs-MacBook-Pro:benchmarks Karl$
```

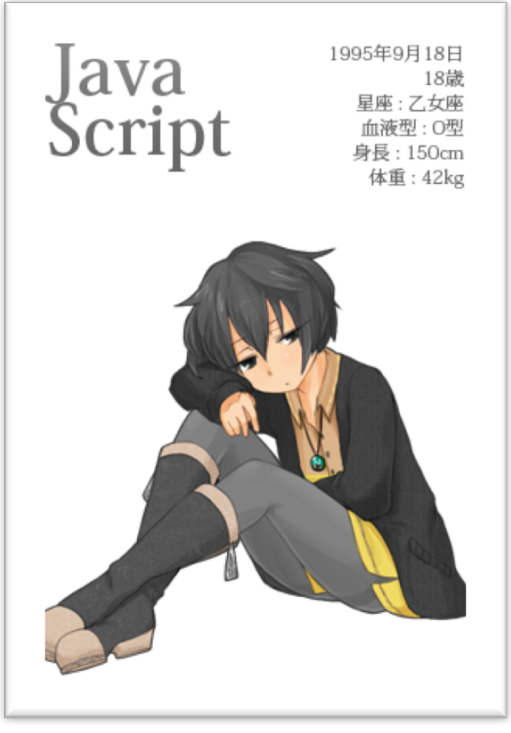
```
Karlheinzs-MacBook-Pro:test Karl$ go build test.go
Karlheinzs-MacBook-Pro:test Karl$ time ./test
49999995000000

real    0m0.010s
user    0m0.005s
sys     0m0.004s
```

```
(venv)Karlheinzs-MacBook-Pro:benchmarks Karl$ time node loop_sum.js
49999995000000

real    0m0.077s
user    0m0.058s
sys     0m0.017s
```

```
(venv)Karlheinzs-MacBook-Pro:benchmarks Karl$ time python sum.py
49999995000000

real    0m1.758s
user    0m1.394s
sys     0m0.204s
```

What is NaCl

+



= CP

What is NaCl(Demo)

```
function moduleDidLoad() {
  CounterModule = document.getElementById( 'counter' );
  var inc = document.getElementById( 'inc' );
  inc.onclick = function() { CounterModule.postMessage(1); };
}

function handleMessage(message_event) {
  updateStatus(message_event.data);
}
```

```
void CounterInstance::HandleMessage( const pp::Var& var_message ) {
  if ( !var_message.is_number() )
    return; //Early exit
  auto x = var_message.AsInt();
  count += x;
  const pp::Var reply( count );
  PostMessage( reply );
}
```

Counter using NaCl

Status **OK**

+

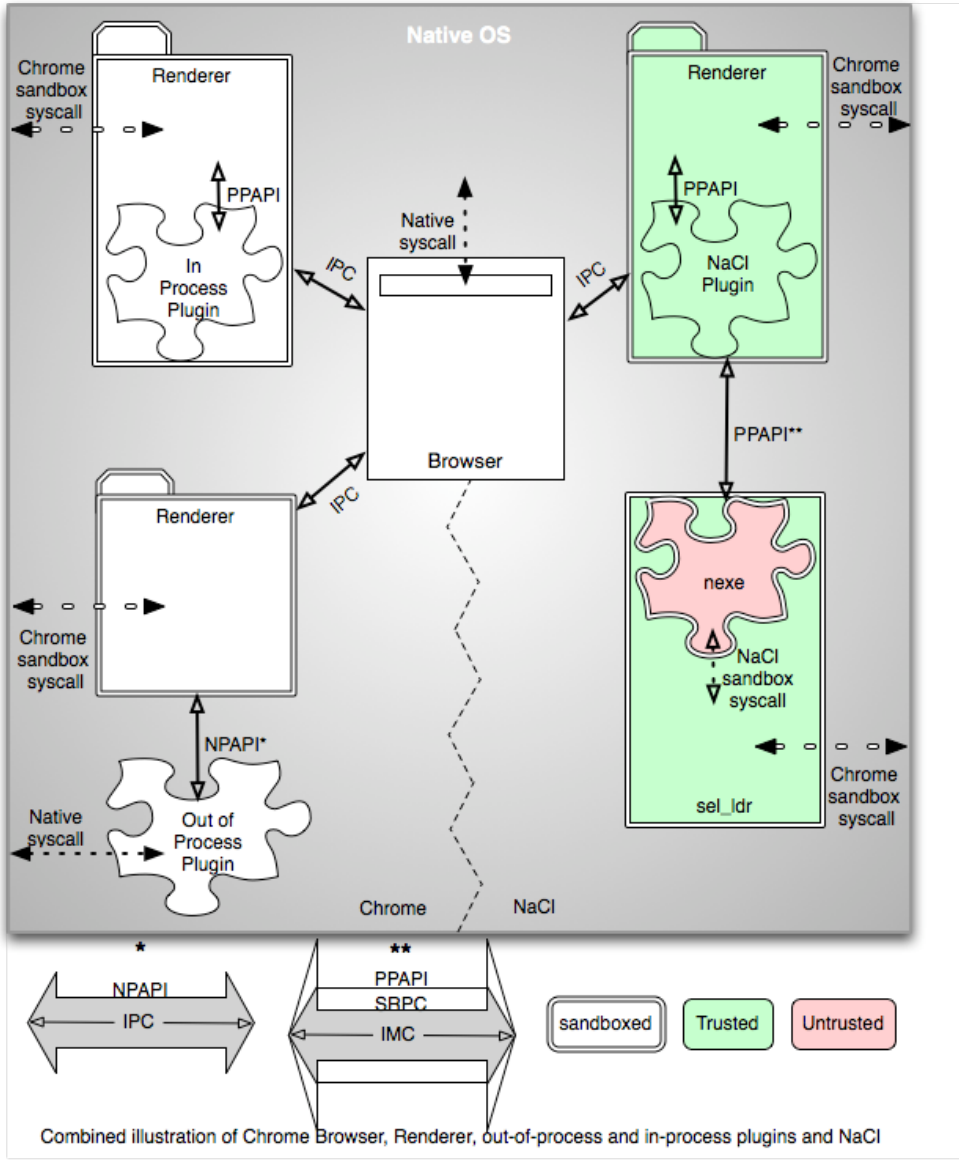
Counter using NaCl

Status **2**

+

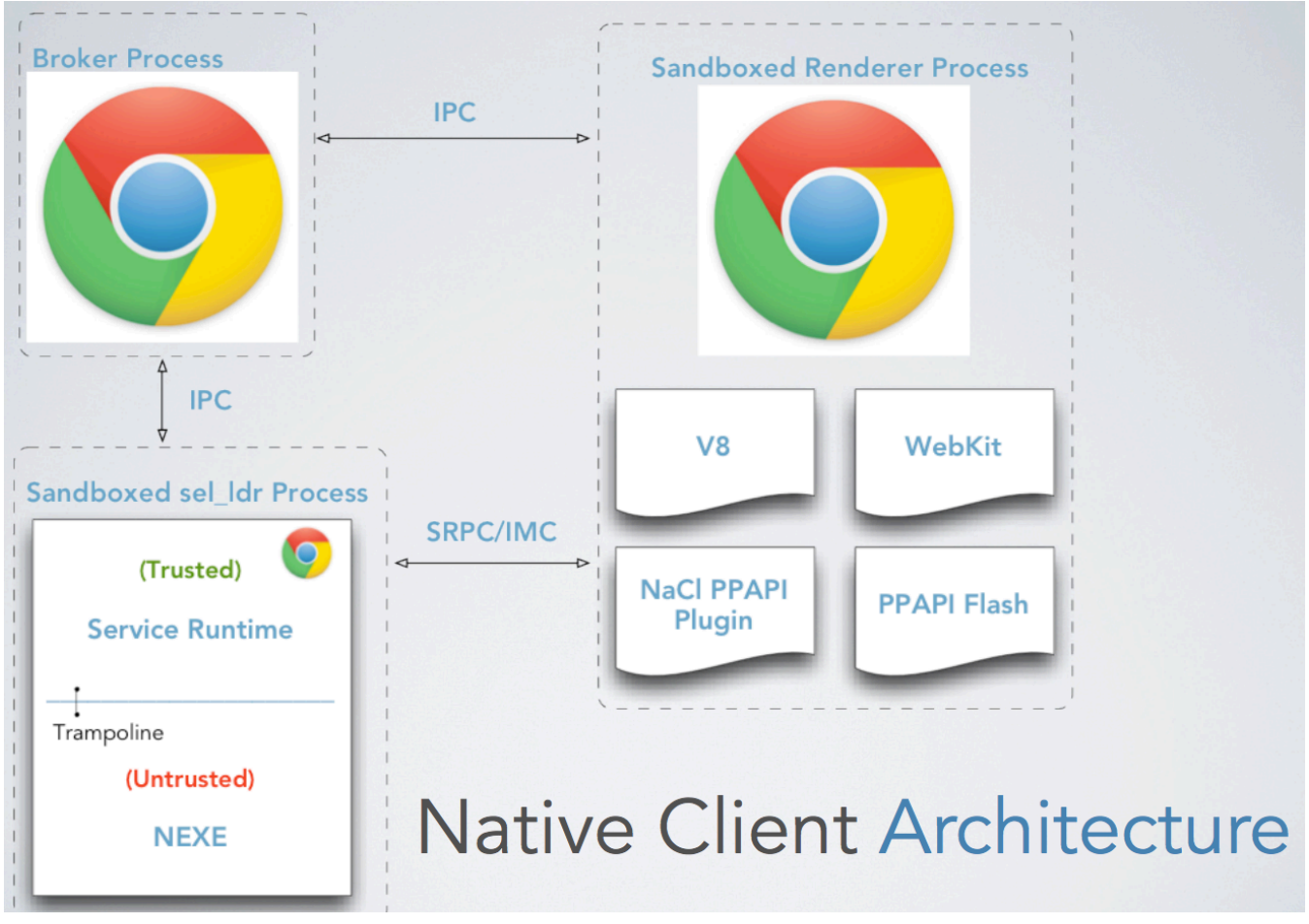
What is NaCl

- A sandbox for running compiled C and C++ code in the browser efficiently and **securely**



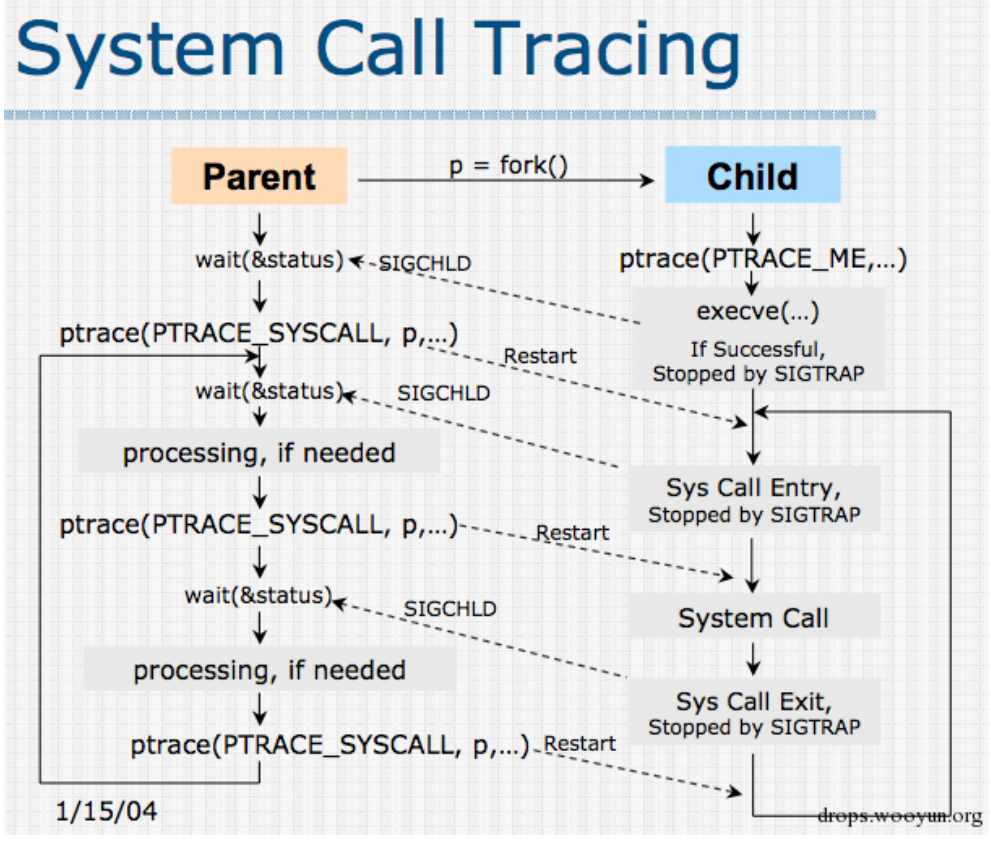
Combined illustration of Chrome Browser, Renderer, out-of-process and in-process plugins and NaCl

System Architecture



RW-System Request Moderation

- ptrace
- systrace

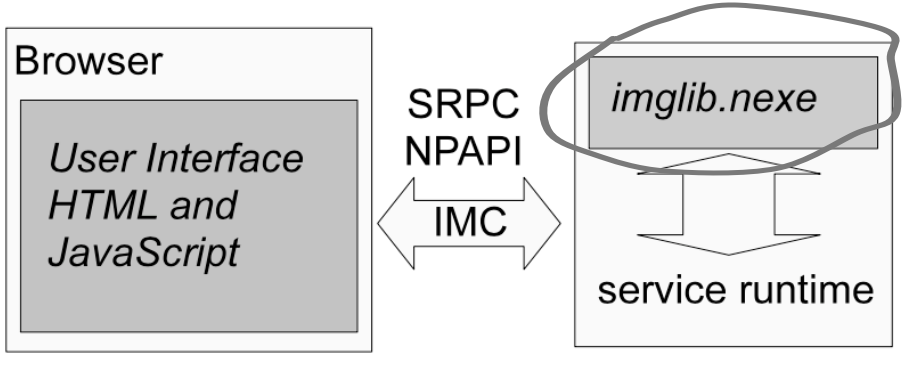


RW-ActiveX

- BS is better, browser is platform.
- Only support IE in Windows~~(The world should use Windows, so no need to cross-platform)~~
- No sandbox
- security ≈ 0

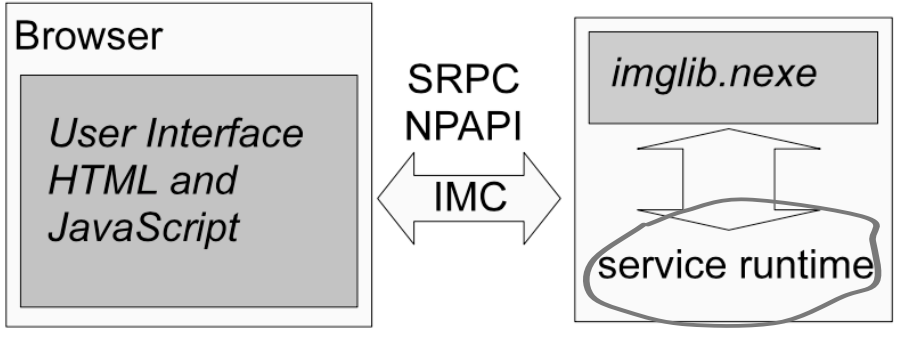
Threat Model-Sandbox

- execute any reachable instruction block in the validated text segment
- exercise the NaCl application binary interface to access runtime services



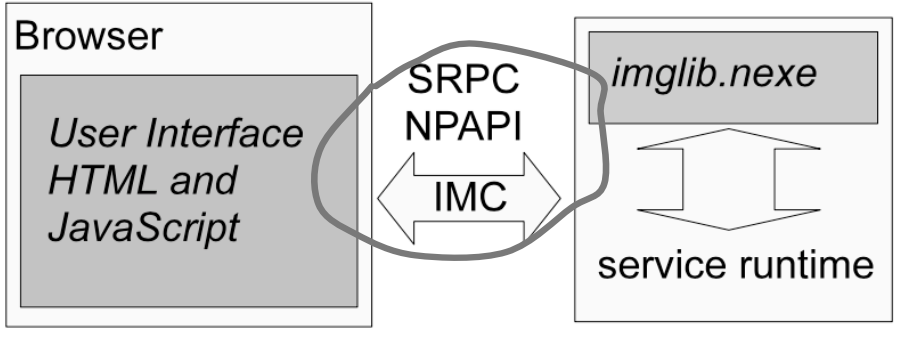
Threat Model-Service Runtime

- unmoderated access to the native operating system's system call interface
- such as process creation, direct disk access



Threat Model-IMC

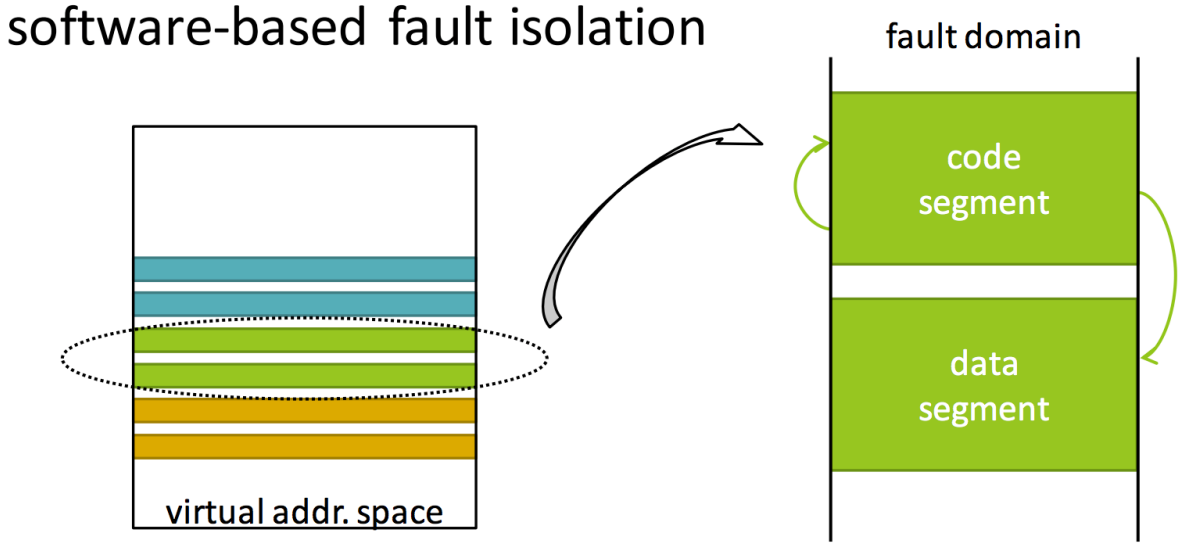
- send arbitrary data via our intermodule communication interface



Solution-Inner Sandbox

- Problem
  - **Data integrity**: no loads or stores outside of data sandbox
  - Reliable disassembly
  - No unsafe instructions
  - **Control flow integrity**
- uses **static analysis** to detect security defects in untrusted x86 code

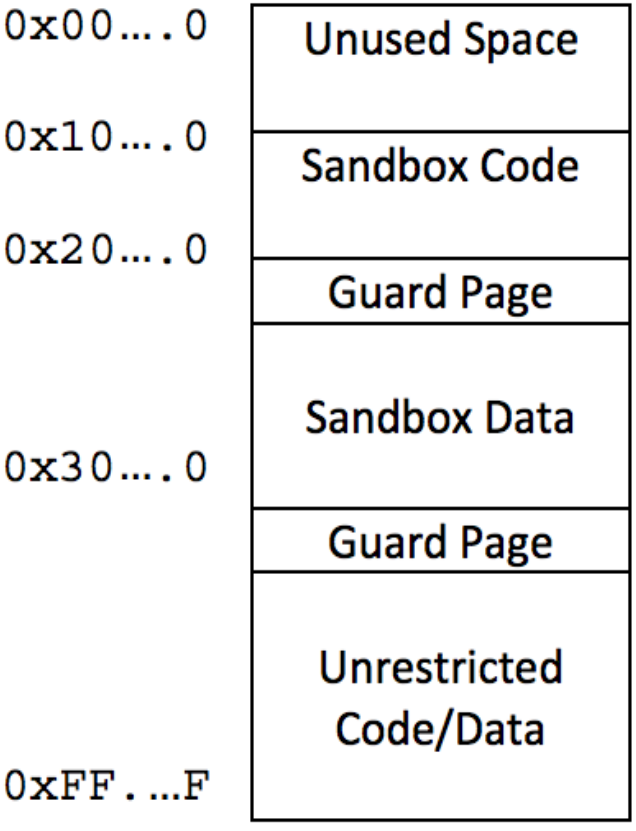
RW-Software Fault Isolation



- load extension codes and their data into their own *fault domain*
  - fault domain = code segment + data segment
- enforce security policies that
  - a distrusted module is prohibited from writing or jumping outside its fault domain.
  - i.e. those distrusted modules cannot modify/execute each other's data/code.
  - the only way to do is to use explicit cross fault-domain communication.

From cs.umd.edu

RW-Software Fault Isolation



Segment Matching

- pseudo code

```
dedicated-reg ← target address
scratch-reg ← (dedicated-reg >> shift-reg)
compare scratch-reg and segment-reg
trap if not equal
store/jump using dedicated-reg
```

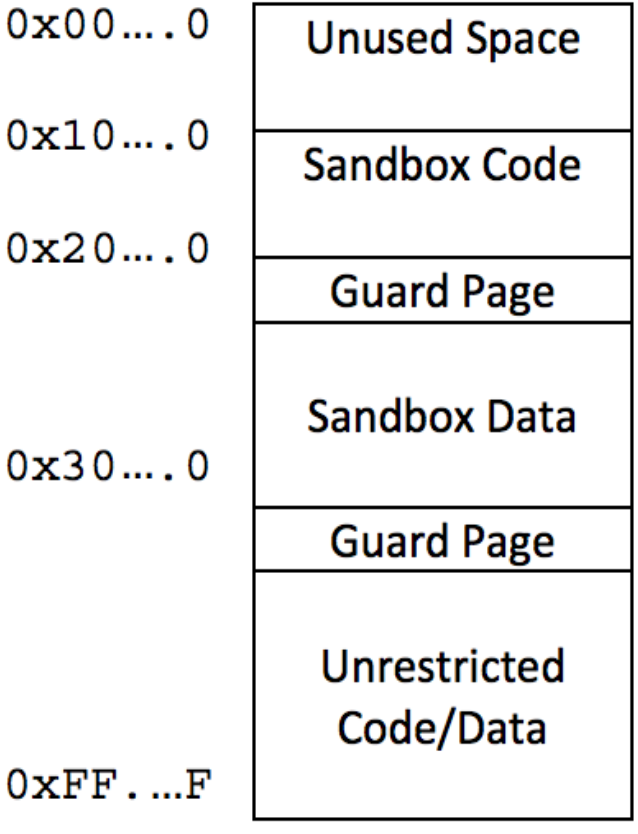
Address Sandboxing

- pseudo code

```
dedicated-reg^x2 ← target-reg & and-mask-reg
dedicated-reg ← dedicated-reg | segment-reg^x2
store/jump using dedicated-reg
```

From cs.umd.edu

RW-Software Fault Isolation



Address Sandboxing

- pseudo code

```
dedicated-reg^x2 ← target-reg & and-mask-reg
dedicated-reg ← dedicated-reg | segment-reg^x2
store/jump using dedicated-reg
```

```
; jmp %eax
; segment ID bit = 32 - 4, for example
and %eax, $0x0000000f ; clear segment ID bits in Rg
or %eax, %cs          ; set segment ID to correct value
jmp %eax              ; do jump to safe target address
```

Solution-Inner Sandbox

- NEXE instructions must be aligned to 32 byte boundary
  - This is required by the inner sandbox
- Blacklisted instructions are never emitted

```
01000ac0 <PpapiPluginStart>: ; 32 byte aligned
 1000ac0: 53                 push %ebx
 1000ac1: 83 ec 28           sub $0x28,%esp
 1000ac4: a1 00 00 02 11     mov 0x11020000,%eax
 1000ac9: 8b 08              mov (%eax),%ecx
 1000acb: 85 c9              test %ecx,%ecx
 1000acd: 74 31              je 1000b00 <PpapiPluginStart+0x40>
 1000acf: eb 0f              jmp 1000ae0 <PpapiPluginStart+0x20>
```

Solution-Inner Sandbox

- No instructions can straddle the 32 byte boundary
- Branches are used to transfer control across boundaries
- No ret instructions, the stack is manually modified

```
01000ac0 <PpapiPluginStart>: ; 32 byte aligned
 1000ac0: 53                 push %ebx
 1000ac1: 83 ec 28           sub $0x28,%esp
 1000ac4: a1 00 00 02 11     mov 0x11020000,%eax
 1000ac9: 8b 08              mov (%eax),%ecx
 1000acb: 85 c9              test %ecx,%ecx
 1000acd: 74 31              je 1000b00 <PpapiPluginStart+0x40>
 1000acf: eb 0f              jmp 1000ae0 <PpapiPluginStart+0x20>
```

Solution-Inner Sandbox

- Branch instructions are properly aligned to validated code
- call instructions are subject to a simple AND operation
  - Alignment masking on the destination register ensures a 32 byte alignment which guarantees the destination has been run through the validator
- Prevents over written NEXE function pointers and modified registers from resulting in arbitrary code execution
- No OR operation compared with former research

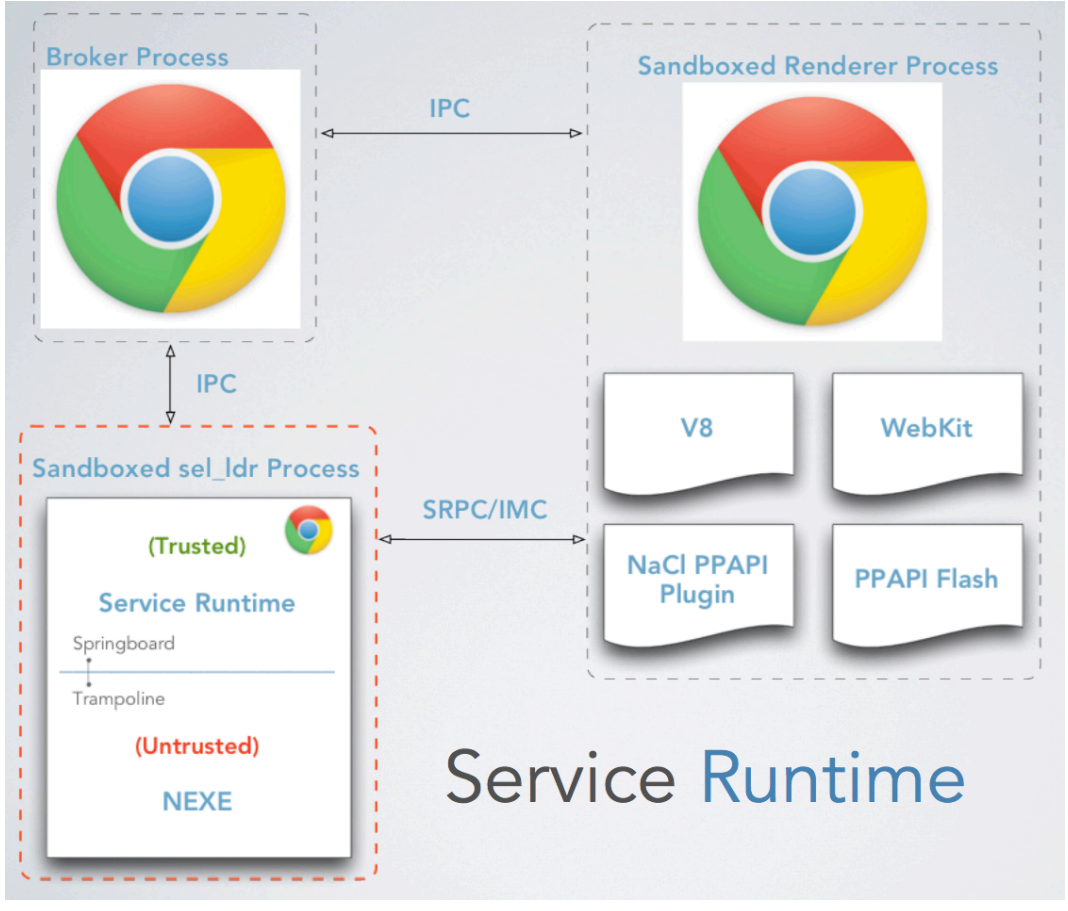
```
0x100057b: 83 e0 e0     and $0xfffffffe0,%eax ; eax = 0x100057a
0x100057e: ff d0        call *%eax            ; eax = 0x1000560
```

Static Analysis

- Pros
  - No need to trust compiler, a small TCB(600 : 73000000)
  - No runtime overhead
- Cons
  - Not accurate
  - **A hacker could modify binary**

Solution-Service Runtime

- Stand alone process, launched from Chrome
- Runs in the same sandbox as Chrome renderer Outer Sandbox

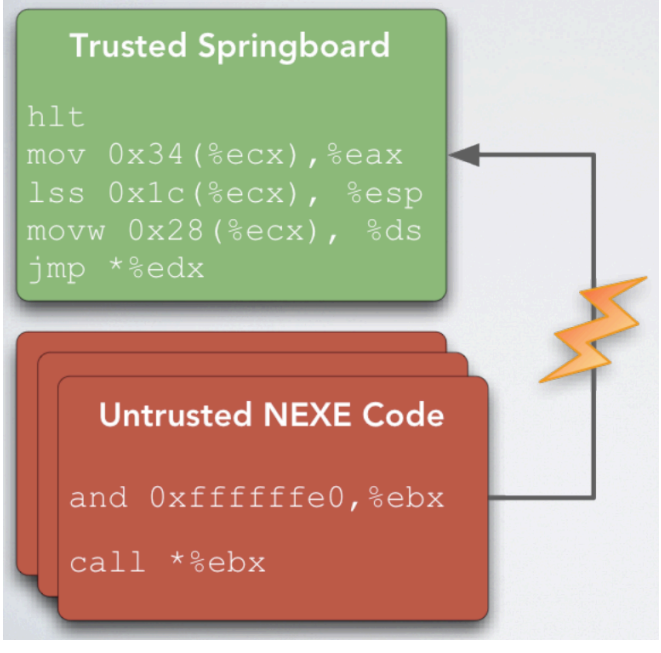


Solution-Service Runtime

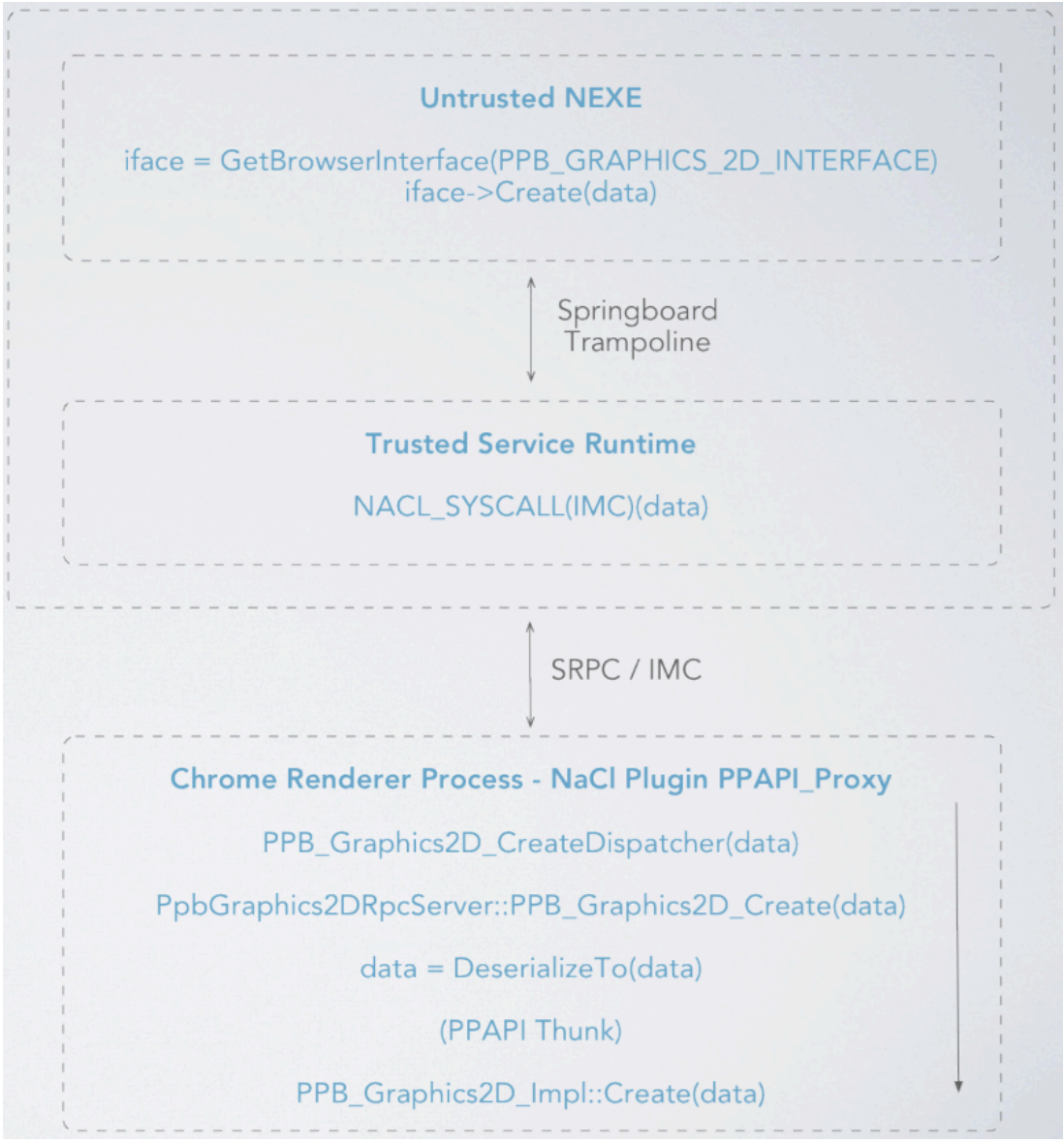
- Trusted instruction blocks are mapped at runtime to enable a context switch between trusted and untrusted
  - Springboard enable trusted to untrusted
  - Trampolines enable untrusted to trusted
  - Each contain privileged instructions that manipulate the segment registers
- x86 Segment Registers
  - Used to separate trusted from untrusted code/data
  - Modified when switching between trusted/untrusted
  - %cs code
  - %ds data
  - %gs thread local storage
  - %ss %es %fs

Solution-Service Runtime

- Untrusted Code couldn’t jump into trusted code because of hlt and 32-byte align.



Solution-IMC



Attack Surface

- inner sandbox: binary validation
- outer sandbox: OS system-call interception
- service runtime binary module loader
- service runtime trampoline interfaces
- IMC communications interface
- NPAPI interface

First Attack

```
andl $0xfffffffe0, %edx
call *(%edx)
```

- The validator and branch alignment ensure the value in the register is 32 byte aligned but not the value it references
- Results in execution of a non-validated instruction
- Discovered by Alex Radocea

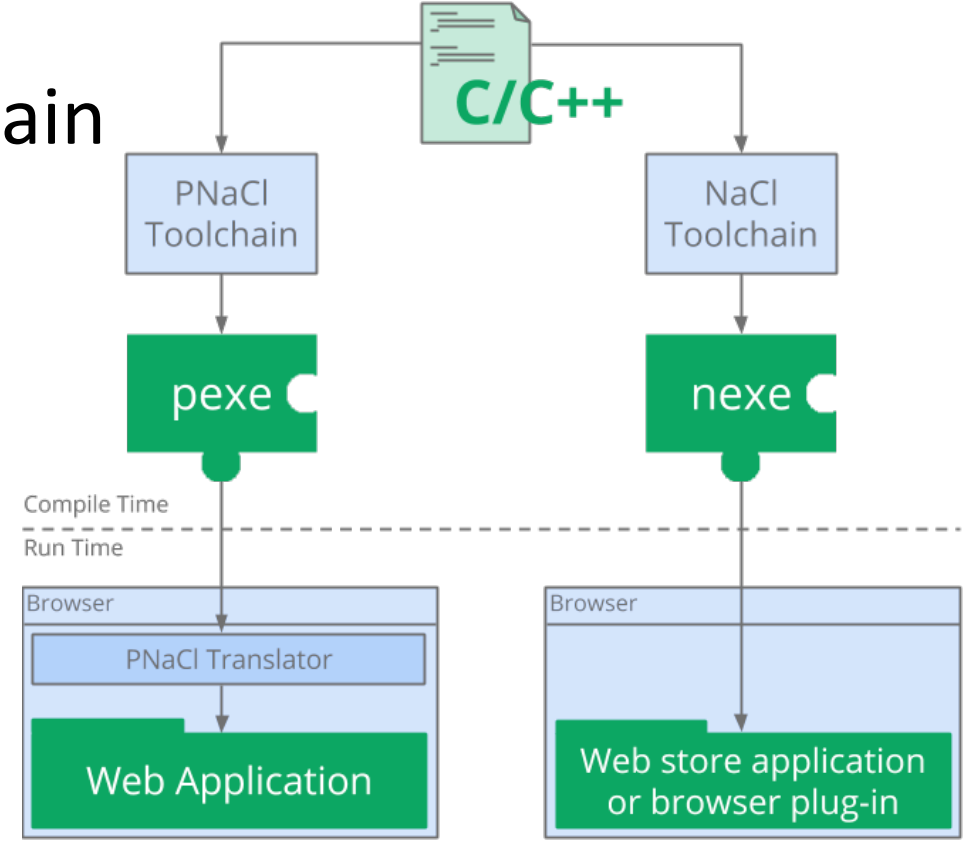
2009 Native Client Security Contest

- Native Client Security Contest: The results are in!
- Uncovered 20 new security vulnerabilities in NaCl
- Nothing that significantly broke the inner sandbox design

Winners	
1st Place	Team "Beached As"
2nd Place	Team "CJETM"
3rd Place	Team "0xdead"
4th and 5th places (tie)	Teams "teamfkmr" and "Alex Rad"

Additional Work

- Modify GNU toolchain



Evaluation

- SPEC2000
- Compute/Graphics Performance Tests
- H.264 Decoder
- Bullet

SPEC2000

	static	aligned	NaCl	increase
ammp	657	759	766	16.7%
art	469	485	485	3.3%
bzip2	492	525	526	7.0%
crafty	756	885	885	17.5%
eon	1820	2016	2017	10.8%
equake	465	475	475	2.3%
gap	1298	1836	1882	45.1%
gcc	2316	3644	3646	57.5%
gzip	492	537	537	9.2%
mcf	439	452	451	2.8%
mesa	1337	1758	1769	32.3%
parser	641	804	802	25.2%
perlbmk	1167	1752	1753	50.2%
twolf	773	937	936	21.2%
vortex	1019	1364	1351	32.6%
vpr	668	780	780	16.8%

Table 5: Code size for SPEC2000, in kilobytes.

QUAKE

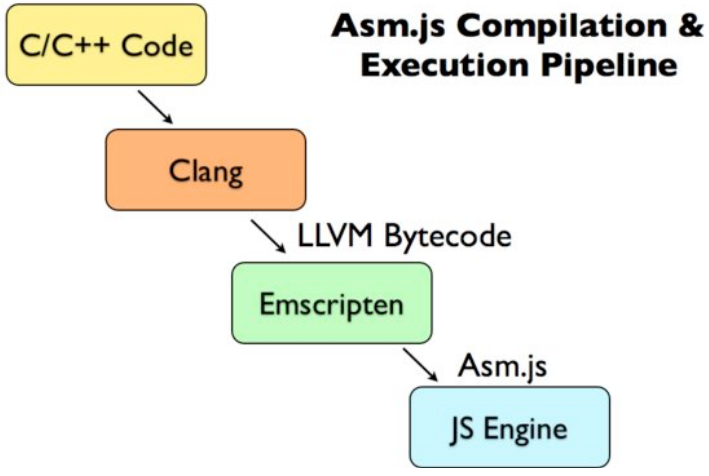
Run #	Native Client	Linux Executable
1	143.2	142.9
2	143.6	143.4
3	144.2	143.5
Average	143.7	143.3

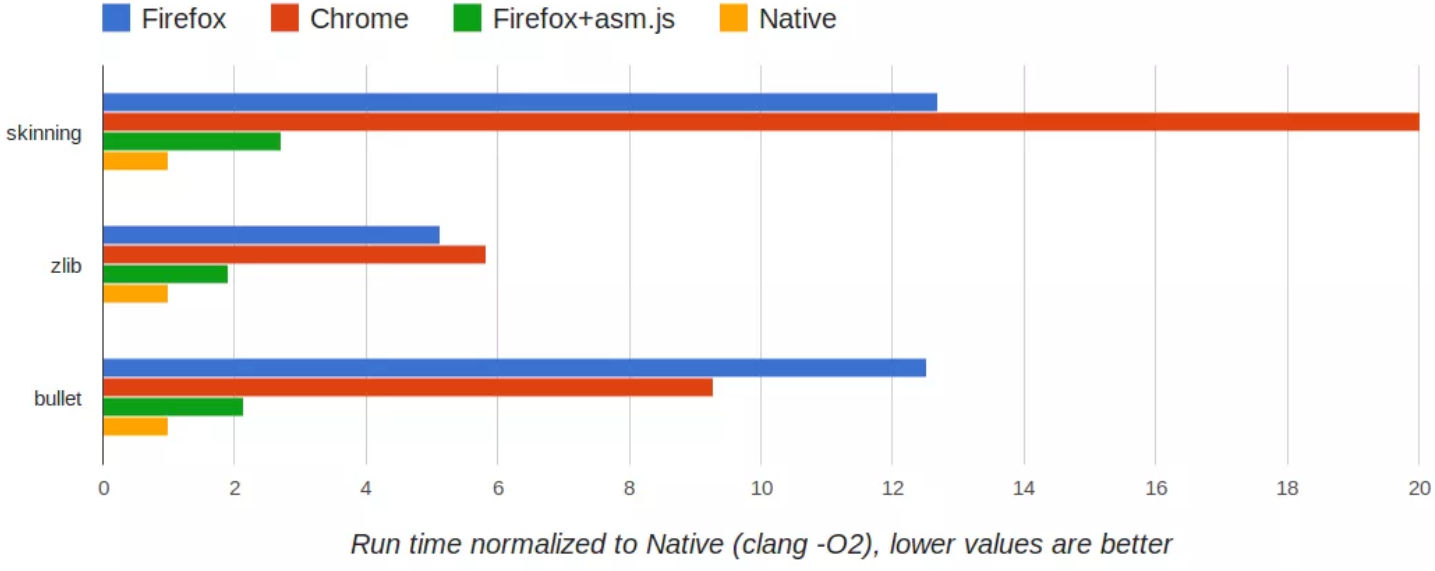
Table 8: Quake performance comparison. Numbers are in frames per second.

TUCAO

- JavaScript is eating the world now.
- ChromeOS + Native Client
- Asm.js is an alternative

Asm.js


Asm.js Compilation & Execution Pipeline
C/C++ Code
Clang
LLVM Bytecode
Emscripten
Asm.js
JS Engine


Firefox
Chrome
Firefox+asm.js
Native
skinning
zlib
bullet
0
2
4
6
8
10
12
14
16
18
20
Run time normalized to Native (clang -O2), lower values are better