Google Native Client

Analysis Of A Secure Browser Plugin Sandbox

Chris Rohlf
Leaf SR

Me

- Chris Rohlf
- Founder: Leaf SR
- BlackHat Review Board & Past Speaker 2009/2011
- @chrisrohlf
- Chris.Rohlf@gmail.com
- leafsr.com

Google Native Client

browsers are the new platform for applications

Google Native Client

browsers are the new platform for applications

sandboxes are the future of application security

Google Native Client

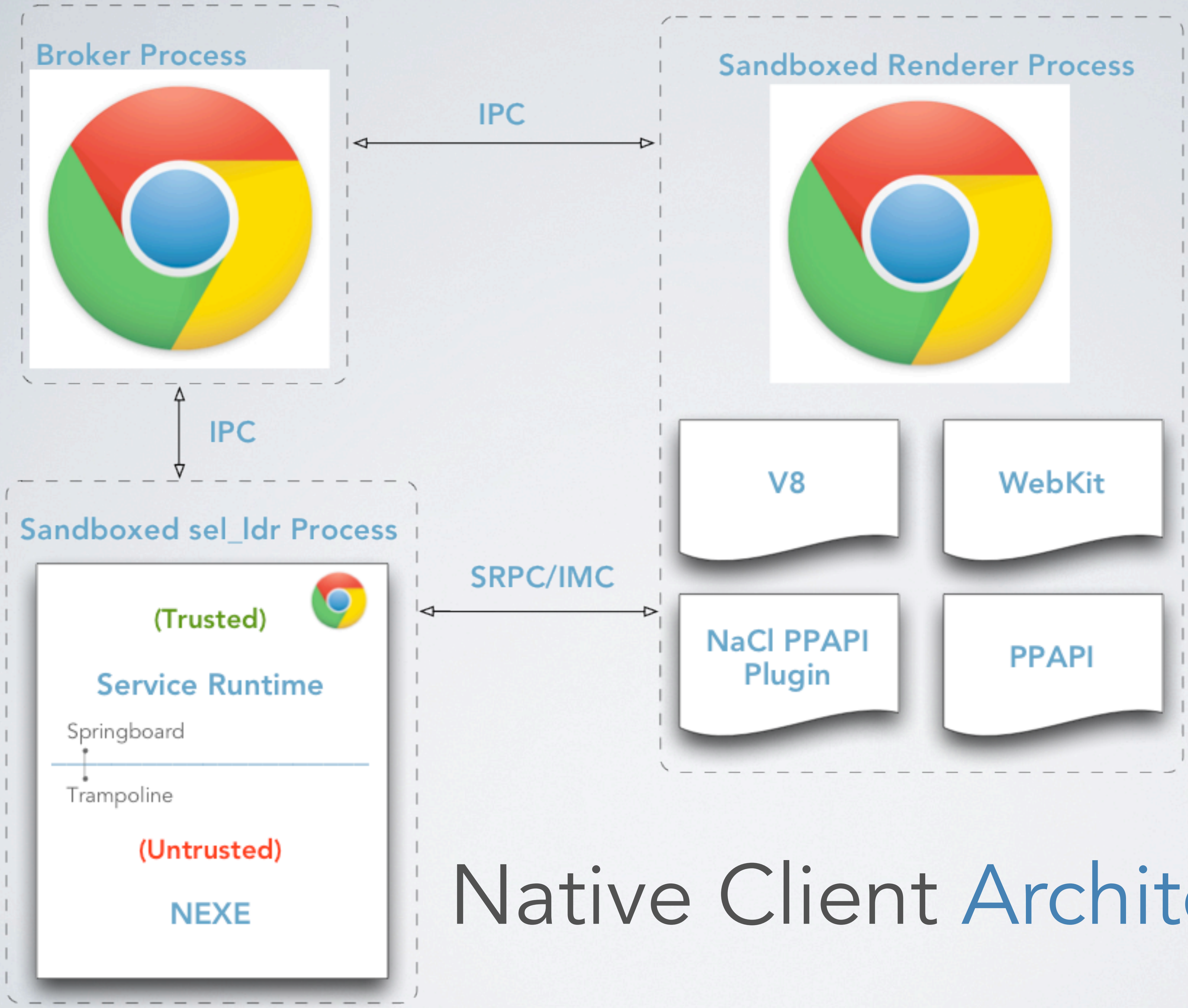
browsers are the new platform for applications

sandboxes are the future of application security

breaking easy targets is boring!

Google Native Client

- A Chrome plugin that allows the execution of native untrusted code in your browser (Win32, OS X, Linux)
- Chrome 14 shipped with NaCl by default
- Large and complex architecture
  - Modified Compiler Toolchain
  - Secure ELF loader
  - Disassembler and Code Validator
  - Service Runtime
  - Inner and Outer Sandbox
  - SRPC (Simple Remote Procedure Call)
  - IMC (Inter-Module Communication)
  - PPAPI (Pepper)

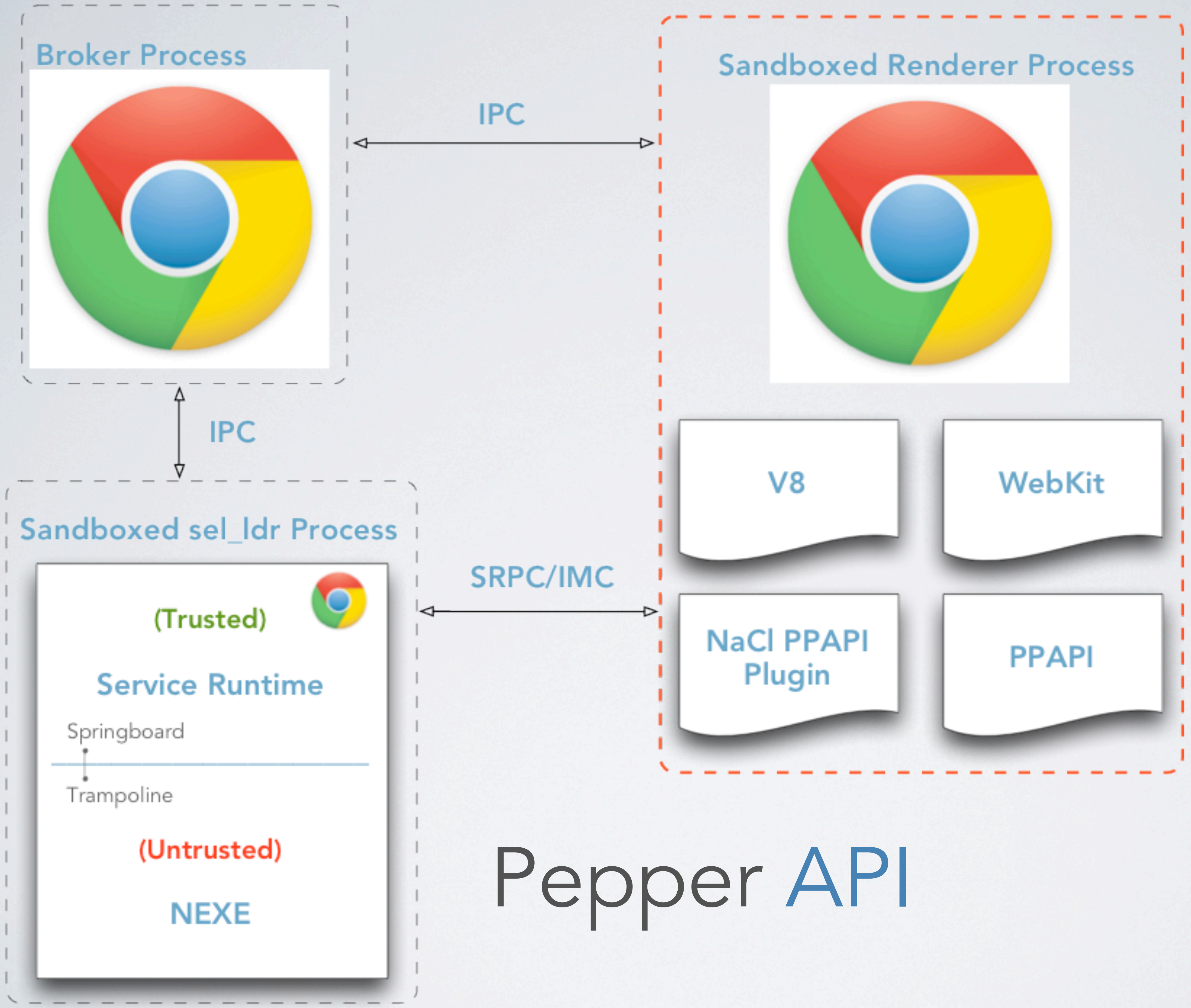


Native Client Architecture

Levels Of Trust

- Outer Sandbox
  - Chrome Broker - Trusted
  - Chrome Renderer - Untrusted
  - NaCl Service Runtime - Untrusted
  - NEXE Module - Untrusted

- Inner Sandbox
  - Chrome Broker - Trusted
  - Chrome Renderer - Trusted
  - NaCl Service Runtime - Trusted
  - NEXE Module - Untrusted


Broker Process
IPC
Sandboxed Renderer Process
IPC
V8
WebKit
Sandboxed sel_ldr Process
SRPC/IMC
(Trusted)
Service Runtime
Springboard
Trampoline
(Untrusted)
NEXE
NaCl PPAPI Plugin
PPAPI

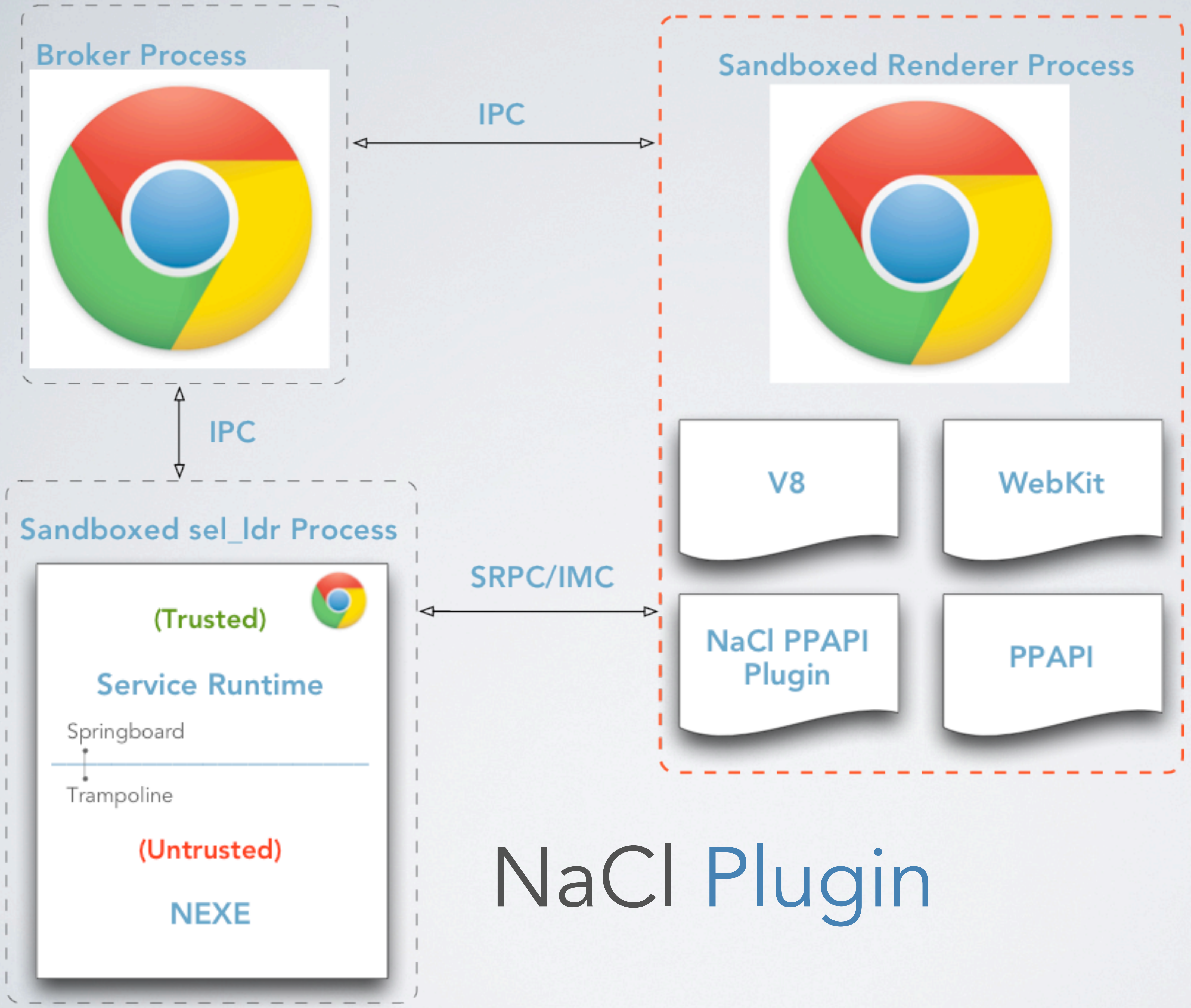
Pepper API

Pepper API

- PPAPI (Pepper Plugin Application Programming Interface)
  - Replaces NPAPI (Netscape Plugin API)
- New APIs for Audio, 3D, Input devices and more
- Interfaces for privileged actions (FileIO...)
- Not scriptable via JavaScript
- Out of process plugins are supported by the standard

PPAPI Plugins

- Trusted PPAPI plugins run in a sandboxed Chrome renderer process or a separate sandboxed plugin process
  - Native Client is a trusted plugin
  - Adobe Pepper Flash is a trusted out of process plugin
- Untrusted PPAPI plugins in Chrome run as a NEXE module
  - NEXE's communicate with PPAPI layer using a proxy


Broker Process
IPC
Sandboxed Renderer Process
IPC
V8
WebKit
Sandboxed sel_ldr Process
SRPC/IMC
(Trusted)
Service Runtime
Springboard
Trampoline
(Untrusted)
NEXE
NaCl PPAPI Plugin
PPAPI

NaCl Plugin

NaCl Plugin

- The NaCl plugin itself is a trusted PPAPI plugin
- Lives inside the Chrome renderer process as a DLL
- Invoked by HTML embed tag

```
<embed name="NaCl_module"
    id="hello_world"
    width=200 height=200
    src="hello_world.nmf"
    type="application/x-NaCl" />
```

NaCl Plugin

- The .nmf file is a ‘NaCl Manifest File’
- JSON that specifies NEXE and .so libraries

```
{ "files": {
  "libgcc_s.so.1": { "x86-32": { "url": "lib32/libgcc_s.so.1" } },

  "main.nexe": { "x86-32": { "url": "hw.nexe" } },

  "libc.so.3c8d1f2e": { "x86-32": { "url": "lib32/libc.so.3c8d1f2e" } },

  "libpthread.so.3c8d1f2e": { "x86-32": { "url": "lib32/libpthread.so.3c8d1f2e" } } },

  "program": { "x86-32": { "url": "lib32/runnable-ld.so" } }

}
```

- The plugin parses this with *jsoncpp*
- NaCl downloads the NEXE using the PPAPI URLLoader and FileIO interfaces

NaCl Plugin

- NaCl must expose part of itself to the browser DOM
- However PPAPI is not scriptable like NPAPI
- Only a few properties and functions are exposed
  - readyState, lastError, exitStatus
  - postMessage

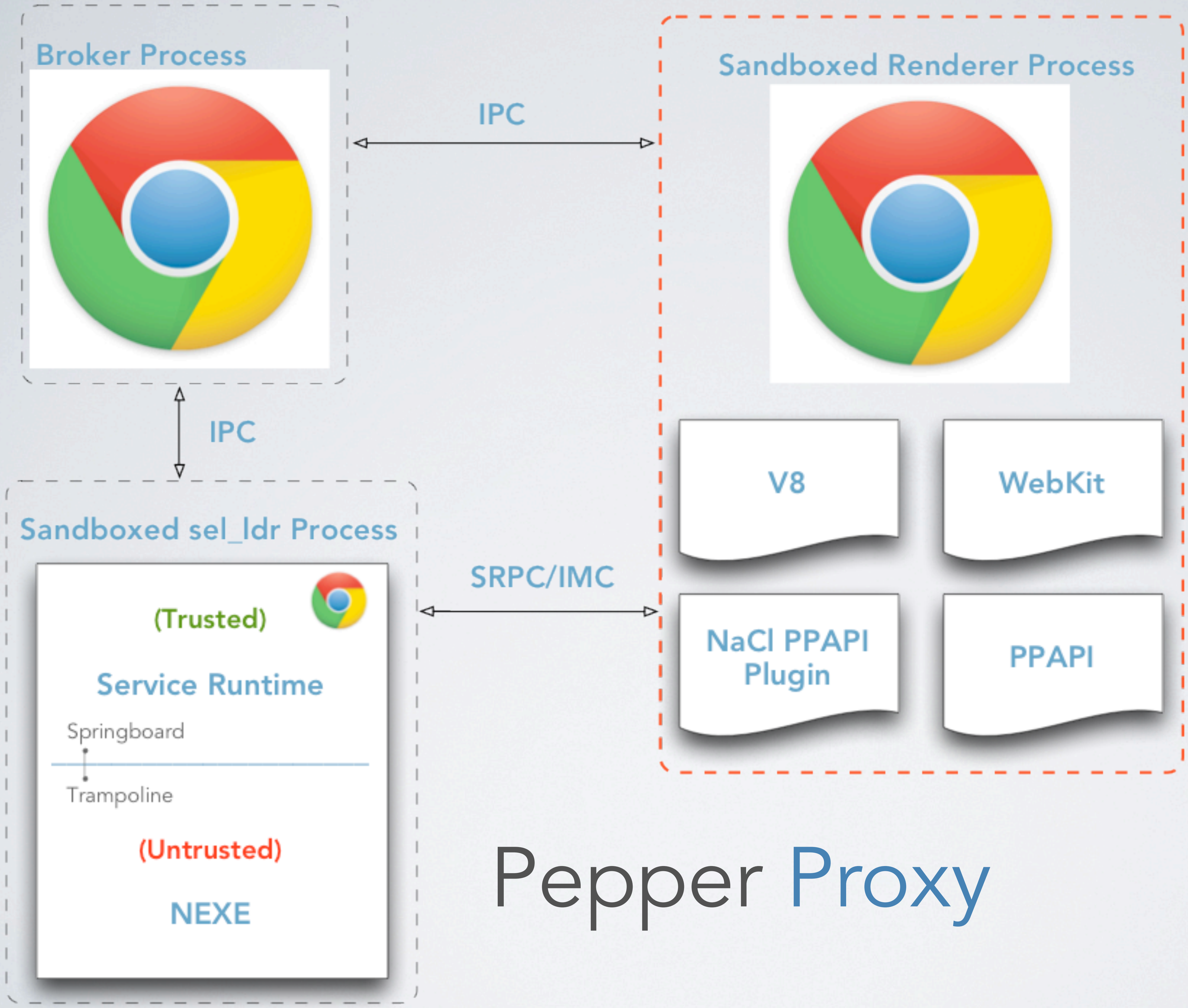
NaCl Plugin

- PostMessage is how JavaScript talks to the NEXE module
- The NEXE must implement pp::Instance::HandleMessage to receive these messages
- The PPB_Messaging::PostMessage C++ interface is used to send messages back to JavaScript
- JavaScript gets alerted to new messages using an event listener and a callback
- These messages have no format and can be binary or ASCII

NaCl Plugin

- NaCl asks the broker to start the service runtime process
- Once the service runtime has started they establish an SRPC connection with each other
    - This is the administrative SRPC channel
    - Individual SRPC channels will be created for each untrusted NEXE module that is launched

Pepper Proxy



Pepper Proxy

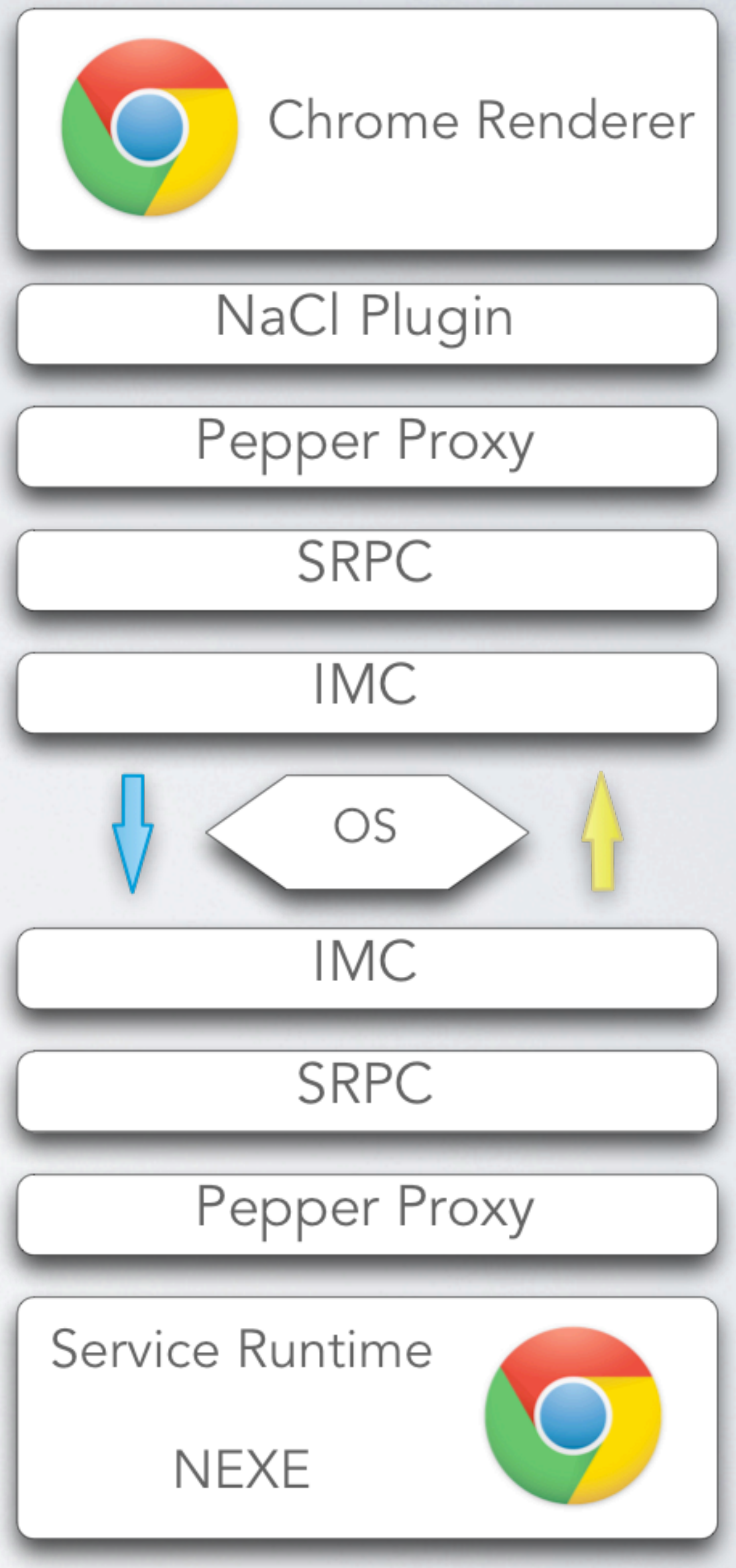
- NEXE modules need to talk to the browser in order to do anything useful
- PPAPI provides plugins the ability to access privileged browser resources
- The NaCl plugin implements the pepper proxy
- This is the bridge between PPAPI and the untrusted NEXE

Pepper Proxy

- NaCl uses IDL files to describe these interfaces
  - Pepper proxy code is auto generated using them
- Both sides can act as server and/or client
  - Shared code between trusted/untrusted sides
- A simple rule to help differentiate the two
  - Interfaces prefixed with PPP are on the untrusted side
  - Interfaces prefixed with PPB are on the trusted side

Pepper Proxy

- The protocol stack allows untrusted NEXE modules to invoke trusted PPAPI interfaces
- Serialized PPAPI arguments, over SRPC, over IMC
- Most of this is binary data packaged up as PP_Var or basic data types

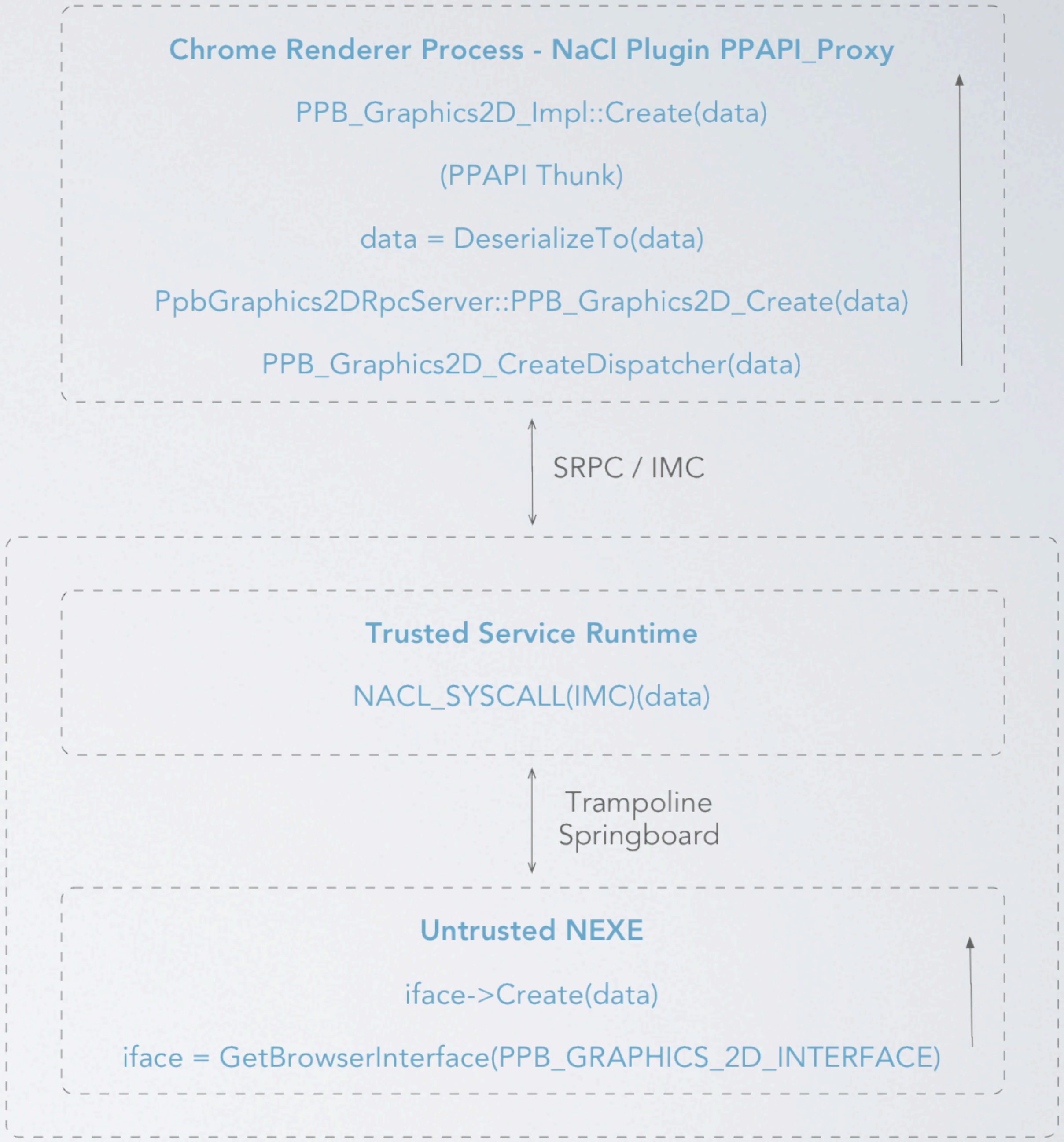


Pepper Proxy

- Data sent over the proxy is serialized
  - PP_VarType - enum specifying type of PP_Var
  - PP_VarValue - union holding simple data types
  - PP_Var - structure that holds PP_VarType and PP_VarValue

Pepper Proxy

1. Creates a PP_Var and other args
2. Calls a remote pepper interface
3. SRPC message gets created
4. SRPC message sent over IMC
5. Received by remote pepper proxy
6. Data deserialized
7. Passed off to PPAPI

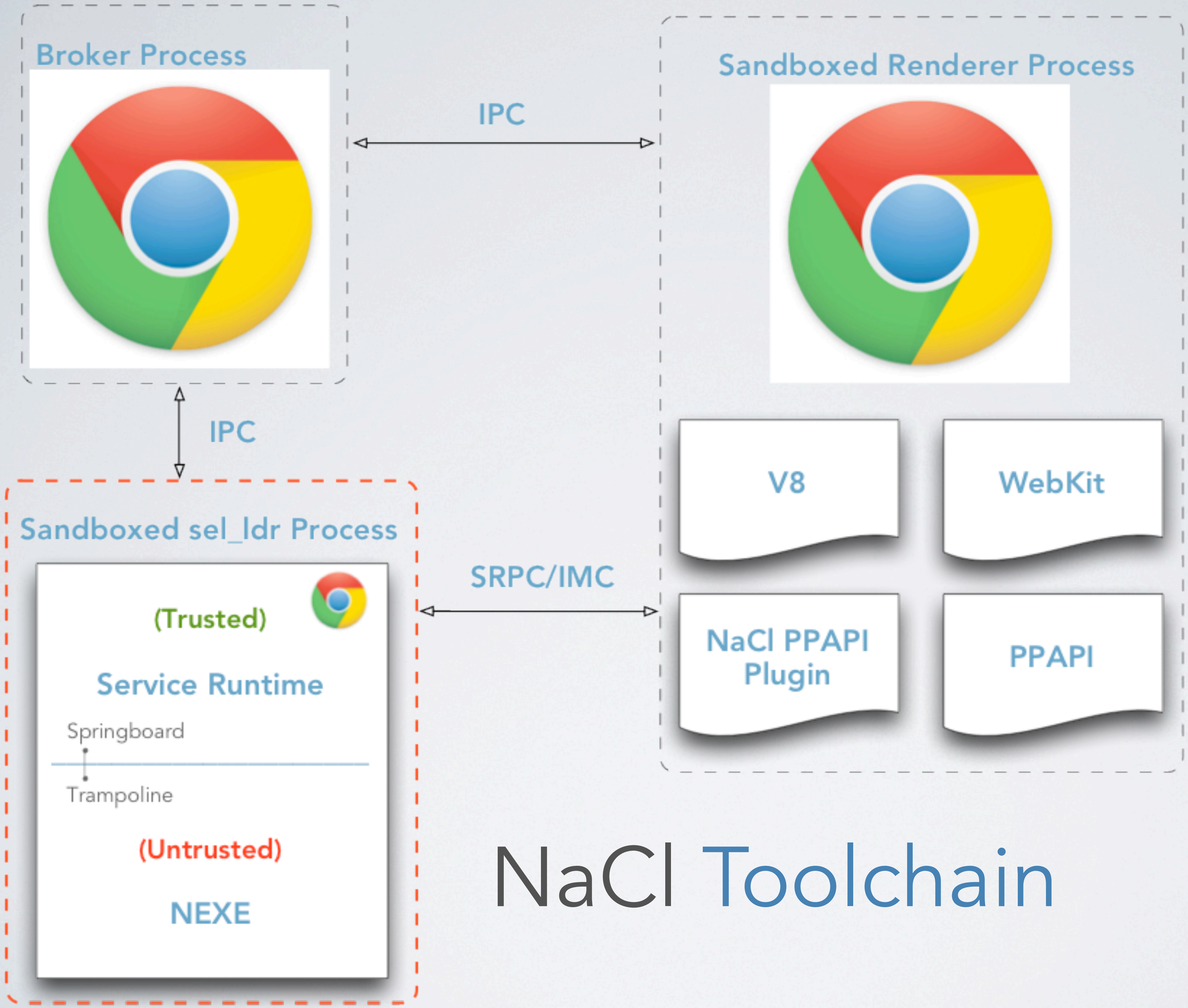

Chrome Renderer Process - NaCl Plugin PPAPI_Proxy
PPB_Graphics2D_Impl::Create(data)
(PPAPI Thunk)
data = DeserializeTo(data)
PpbGraphics2DRpcServer::PPB_Graphics2D_Create(data)
PPB_Graphics2D_CreateDispatcher(data)
SRPC / IMC
Trusted Service Runtime
NACL_SYSCALL(IMC)(data)
Trampoline
Springboard
Untrusted NEXE
iface->Create(data)
iface = GetBrowserInterface(PPB_GRAPHICS_2D_INTERFACE)

Pepper Proxy

- Manages callbacks and related SRPC data structures
  - Callbacks exist to inform a remote NEXE when the renderer has completed an operation
  - Callbacks are bound to the SRPC channel
  - Callbacks can only be invoked on the main thread

Pepper Proxy

- The pepper proxy is <u>not</u> a security boundary
- It performs little to no validation of untrusted data
- It is exists only to proxy data between an untrusted NEXE and the PPAPI implementation



NaCl Toolchain

NaCl Toolchain

- Modified GCC toolchain ships with NaCl SDK
- Only the SDK compiler can be used to produce a NEXE
- SDK can produce code for:
  - 32-bit x86
  - 64-bit x86_64

NaCl Toolchain

- NEXE modules are compiled and linked as ELF
  - All of the typical structures are present
    - ELF Header, Program Headers, Dynamic Segment, Section Headers, Symbol Tables, Relocation Entries
- Standard tools can be used to disassemble a NEXE

NaCl Toolchain

- 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>
 1000ad1:       90                  nop
 1000ad2:       90                  nop
```

NaCl Toolchain

- No instructions may 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>
 1000ad1:       90                  nop
 1000ad2:       90                  nop
```

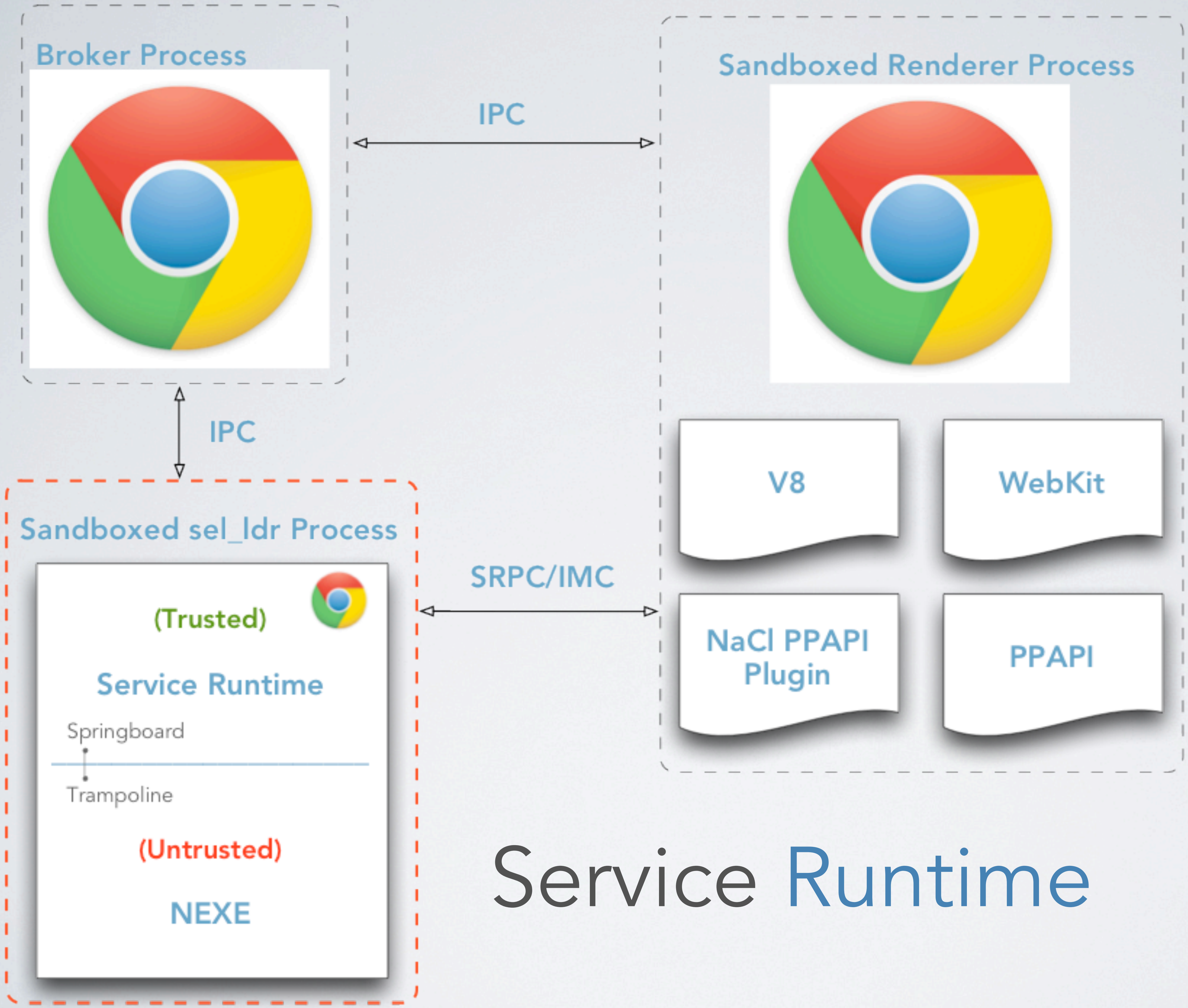
NaCl Toolchain

- 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

```
0x100057b:   83 e0 e0   and    $0xffffffe0,%eax ; eax = 0x100057a
0x100057e:   ff d0      call   *%eax            ; eax = 0x1000560
```

NaCl Toolchain

- NaCl toolchain
  - Ensures all ELF structures are properly formed
  - Only safe instructions are emitted
  - All branch instructions are preceded by alignment mask
- But an attacker can modify the NEXE binary...



Service Runtime

Service Runtime

- Stand alone process, launched from Chrome
- Runs in the same outer sandbox as Chrome renderer
- Shares its virtual address space with a NEXE module
  - Service Runtime memory is <u>trusted</u>
  - NEXE memory is <u>untrusted</u>
  - Separation between the two is the ‘Inner Sandbox’

ELF Loader

- Multiple segments refer to other segments and so on
- Validates ELF header and Program headers
- ELF loader enforces:
  - Only one executable segment allowed
  - Segment load addresses and sizes

Instruction Validator

- Disassembles all NEXE instructions
- Starts at trusted 32 byte aligned entry point
- Exits on any blacklisted instructions
  - Privileged instructions
  - Instructions that modify segment registers
  - ret
  - sysenter
  - prefix bytes

Instruction Validator

- The validator only performs static analysis on the code
- Static analysis will not know register values at runtime
- As long as the branch target is mod 32 aligned it knows whatever the target is it has been validated
- Example:
  - 0x1000560 not known when the validator runs
    - Guaranteed to have been properly validated due to proper 32 byte alignment

```
0x100057b:   83 e0 e0   and    $0xffffffe0,%eax ; eax = 0x100057a
0x100057e:   ff d0      call   *%eax            ; eax = 0x1000560
```

Instruction Validator

- The NaCl SDK contains a standalone NEXE validator

```
$ ./ncval_x86_32 ../examples/hello_world_glibc/hw.nexe
segment[0] p_type 3 p_offset ee0 vaddr 11000ee0 paddr 11000ee0 align 1 filesz 19 memsz 19 flags 4
segment[1] p_type 1 p_offset 140 vaddr 1000140 paddr 1000140 align 65536 filesz da0 memsz da0 flags 5
parsing segment 1
VALIDATOR: 10008e6: ret instruction (not allowed)
VALIDATOR: 10008e6: Illegal instruction
segment[2] p_type 1 p_offset ee0 vaddr 11000ee0 paddr 11000ee0 align 65536 filesz 478 memsz 478 flags 4
segment[3] p_type 1 p_offset 1358 vaddr 11011358 paddr 11011358 align 65536 filesz 13c memsz 13c flags 6
segment[4] p_type 1 p_offset 10000 vaddr 11020000 paddr 11020000 align 65536 filesz 0 memsz 2c flags 6
segment[5] p_type 2 p_offset 136c vaddr 1101136c paddr 1101136c align 4 filesz d8 memsz d8 flags 6
segment[6] p_type 1685382481 p_offset 0 vaddr 0 paddr 0 align 4 filesz 0 memsz 0 flags 6
segment[7] p_type 7 p_offset 0 vaddr 0 paddr 0 align 4 filesz 0 memsz 0 flags 4
*** ../examples/hello_world_glibc/hw.nexe IS UNSAFE ***
Validated ../examples/hello_world_glibc/hw.nexe
*** ../examples/hello_world_glibc/hw.nexe IS UNSAFE ***
```

Service Runtime

- Inner Sandbox
  - Enforced by loading the untrusted NEXE into the trusted service runtime address space

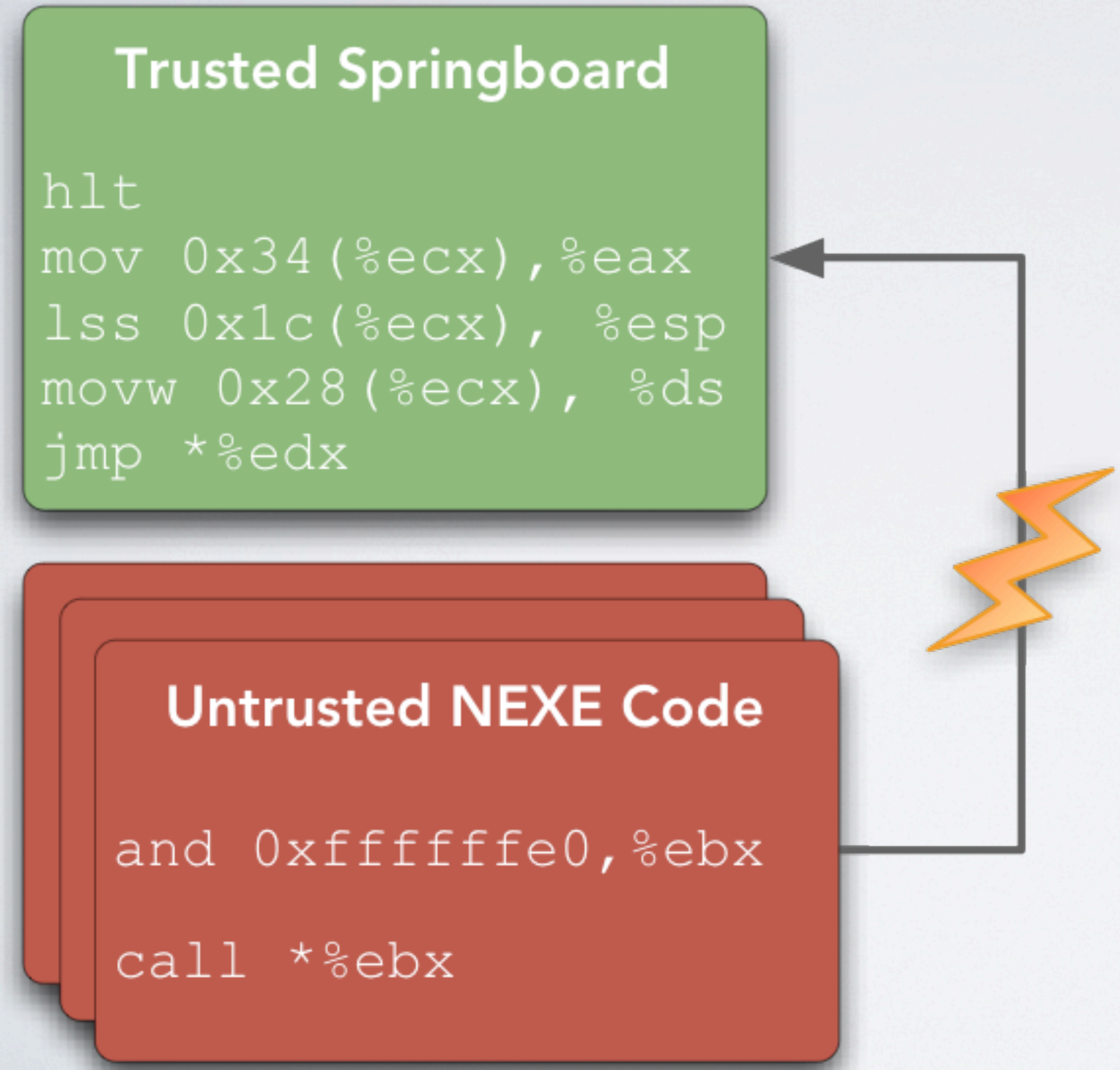
- Separating untrusted code and data
  - x86 memory segmentation model
  - x86_64 mov/branch alignment, guard pages
  - ARM load/store/branch alignment, guard pages
  - Trusted TLS segments store data/registers between context switches

Service Runtime

- 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 all set to %ds

- 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 contains privileged instructions that manipulate the segment registers

Service Runtime



```
hlt
mov 0x34(%ecx),%eax
lss 0x1c(%ecx), %esp
movw 0x28(%ecx), %ds
jmp *%edx
```

```
and 0xffffffe0,%ebx

call *%ebx
```

The untrusted NEXE code is unable to transfer control to an arbitrary location in the trusted Springboard due to the alignment mask before the call. It may only execute the first instruction, which is a *hlt*

Service Runtime

- No existing or new memory allocations may be marked as executable at runtime
- This guarantees only validated code pages have executable permissions

NACL Syscalls

- A NEXE cannot make normal syscalls
    - sysenter / int 0x80 not allowed by the validator
- 30~ NACL_SYSCALLS
- Basic operations such as open, close, read, write, ioctl, mmap, munmap, stat, _exit and a few others
    - NACL specific IMC accept, connect, send, recv
- These are dispatched via Springboard/Trampoline code

Service Runtime IMC

- Inter-Module Communication (IMC) is a protocol implemented by NaCl
- IMC uses basic socket types that are initialized by the imc_makeboundsock and imc_socketpair NACL_SYSCALL's
- IMC is built on top of platform supplied UNIX sockets, named pipes and shared memory
- IMC is too low level to be used by application developers

Service Runtime SRPC

- Simple Remote Procedure Call (SRPC)
- Is encapsulated by IMC
- SRPC allows for the encapsulation of serialized data between NEXE modules and the NaCl plugin
- SRPC endpoints are invoked via SRPC signatures

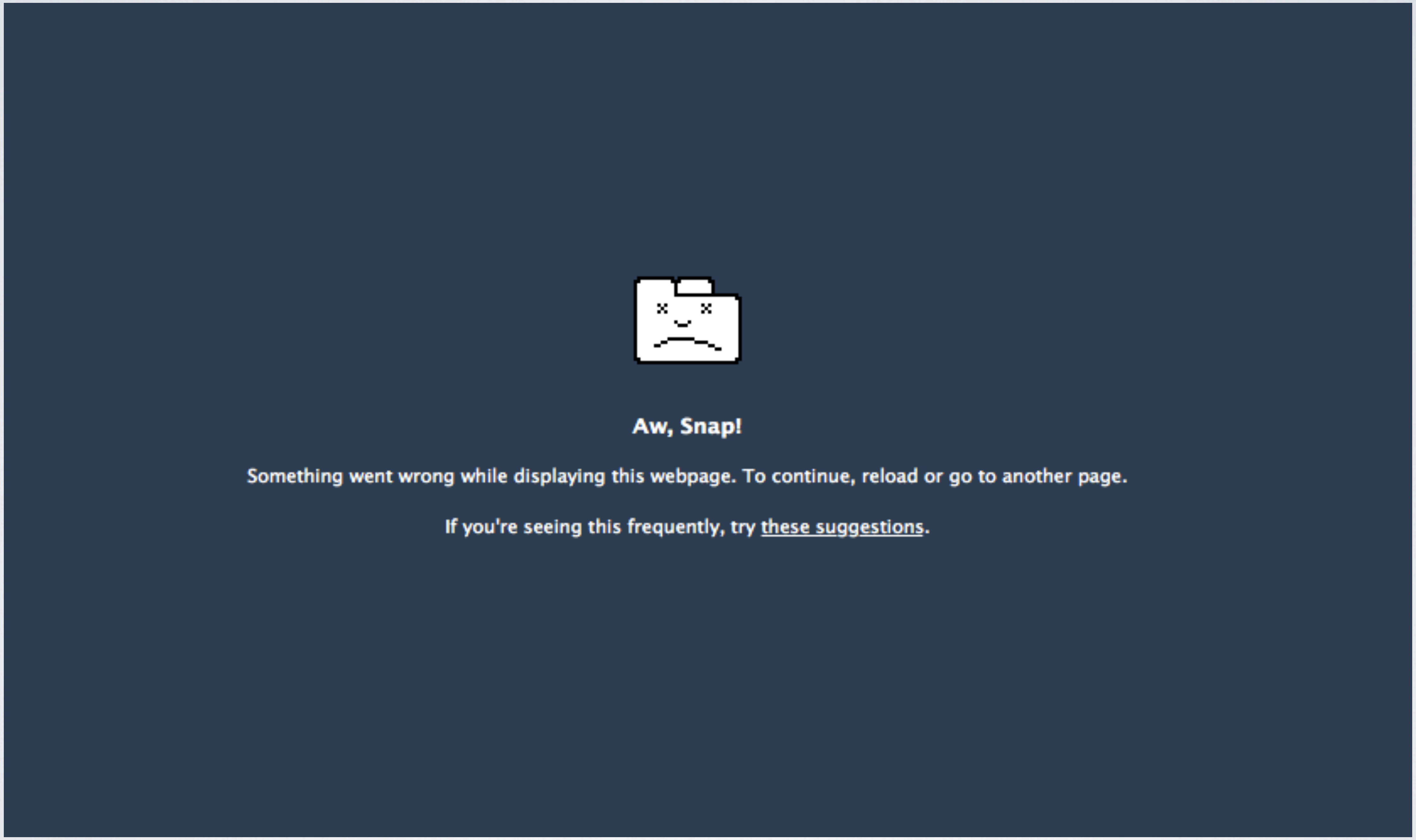
```
NaClSrpcInvokeBySignature(channel, "MyMethod:i:i", resource, bool)
```

- Although SRPC is at a higher level than IMC its not intended to be used by application developers directly

Portable NaCl

- PNaCl (Pinnacle)
- Web site hosts LLVM IR produced by NaCl SDK toolchain
- PNaCl transforms LLVM IR to a NEXE for the users architecture using AOT compiler
- The inner sandbox and pepper proxy remain as-is
- 32-bit ARM Support
- Google has projected a 2012 release

NaCl Attack Surface



Aw, Snap!

Something went wrong while displaying this webpage. To continue, reload or go to another page.

If you're seeing this frequently, try these suggestions.

NaCl Security

If a NaCl module can execute instructions that were not validated by the service runtime then the security provided by Native Client is broken

NaCl Attack Surface

- Outer sandbox escape requires
  - Vulnerabilities in the broker process
  - No/Weakly sandboxed processes (Flash, GPU)
  - Kernel vulnerabilities the sandbox can't fully prevent

- Inner sandbox restrictions
  - Cannot reach the broker process directly
  - You cannot make syscalls or talk to the kernel directly
  - Instruction validation

NaCl Attack Surface

- NaCl raises the bar for exploitation
- But you need trusted components to do useful things
  - These reside in the outer sandbox
- We find attack surface anywhere untrusted code can influence the execution of trusted code

NaCl Attack Surface

- Vulnerable NEXE modules are not an issue
  - Validation of all direct/indirect execution branches
  - Why attack a NEXE module when you can load one directly instead?

- Malicious NEXE modules
  - Attempt to find and exploit vulnerabilities in various trusted NaCl components
  - **Find vulnerabilities in the Chrome renderer and go after them with an untrusted NEXE module**

NaCl Attack Surface

- Service Runtime
  - Inner Sandbox
    - ELF loader
    - Instruction validator disassembler
    - NACL_SYSCALL implementations

NaCl Attack Surface

- NaCl PPAPI Plugin
  - IMC
  - SRPC
  - JavaScript DOM interfaces
  - JSON parser

NaCl Attack Surface

- PPAPI - Pepper Proxy
  - Server interfaces
  - Client interfaces
- PPAPI
  - Interface implementations reached via the proxy
- GPU
  - Direct interfaces to the GPU
  - Pinky Pie exploited an integer overflow here at Pwnium

First Inner Sandbox Breakout

- Call instruction memory dereference

```
andl $0xffffffe0, %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 Security Contest

- Uncovered 20 new security vulnerabilities in NaCl
- Nothing that significantly broke the inner sandbox design
- 1st place - Mark Dowd, Ben Hawkes
  - 2nd place - Chris Rohlf, Eric Monti, Jason Carpenter
    - 3rd place - Gabriel Campana
      - 4th place - Daiki Fukumori
        - 5th place - Alex Radocea
- Some of these vulnerabilities are still relevant to NaCl

2009 Security Contest

- Two byte jmp prefixes
  - Unchecked prefix bytes on branch instruction
- EFLAGS direction flag modification
  - Untrusted NEXE code can change the direction flag
- Validated code unmapping
  - An untrusted NEXE can unmap valid code and replace it
- Uninitialized vtable
  - An uninitialized vtable can lead to code execution
- Double delete operator
  - NPAPI Object life cycle management confusion

2009 Security Contest

- The architecture has changed significantly since 2009
- NPAPI is no longer used but the same type of issues could be found in the PPAPI replacement
- Proved the inner sandbox was relatively strong
  - Provided a good look at the future of NaCl vulnerabilities
  - Trusted components that handle untrusted data are more likely to contain vulnerabilities than the inner sandbox

Google NaCl Vulnerabilities

- Win64 inner sandbox escape via KiUserExceptionDispatcher
  - Exceptions transfer execution to this function which is not aware of the trusted vs untrusted stack
- bsf Instruction inner sandbox escape
  - Cannot properly validate alignment of conditional value
- Trampoline address space leak
  - Does not use PIC code, leaks a .text instruction address
- Address space leak via JavaScript error
  - A JavaScript error message contained a memory address

Pepper Proxy Source Audit

- June 2011 Pepper Proxy Source Audit
  - Performed under contract to Google while at Matasano
- 3 week project
- 1 Person
- 10 vulnerabilities discovered
- Manual source audit of C/C++
  - Many thousands of lines
  - No scanners
  - Mostly grep and reading source

Pepper Proxy Source Audit

- Are there vulnerabilities in the pepper proxy that would allow a malicious NEXE to escape the inner sandbox?
  - Many interfaces accept and deserialize untrusted data
  - Lots of opportunity for vulnerable code

Pepper Proxy Source Audit

- PPB_URLLoader_Open CORS Header Injection

```
set_method = request_interface_->SetProperty(request_,
  PP_URLREQUESTPROPERTY_METHOD,
  Module::StrToVar("POST\x0d\x0ax-csrf-token:\x20test1234"));
```

- PPB_Audio_Create SRPC Channel Use After Free
  - The pepper proxy can be tricked into using a stale SRPC channel pointer when a callback is completed
  - Proof of concept:
    - Add `exit(-1)` after requesting PPAPI fill the audio buffer
    - Pepper proxy SRPC invalidates the channel but the callback structure is left with a stale pointer

Pepper Proxy Source Audit

- PPB_Graphics2D_Create Shared Memory Overflow
  - Check for overflow in (width * height)

```
 if (static_cast<int64>(width) * static_cast<int64>(height) >=
    std::numeric_limits<int32>::max())

  return false;  // Prevent overflow of signed 32-bit ints.
```

  - Perform multiplication after the check but * 4

```
PepperPluginDelegateImpl::CreateImage2D(int width, int height) {
  uint32 buffer_size = width * height * 4;
}
```

  - Create a new shared memory segment

```
TransportDIB::Create() {
  const int shmkey = shmget(IPC_PRIVATE, size, 0666);
}
```

Pepper Proxy Source Audit

- Integer/Buffer Overflows
  - PPB_Graphics2D_Create Shared Memory Integer Overflow
  - PPB_Context3DTrusted_CreateTransferBuffer Shared Memory Integer Overflow
  - PPB_URLLoader_ReadResponseBody Heap Overflow
  - PPB_FileIO_Dev_Read Heap Overflow
  - PPB_PDF_SearchString Potential Heap Overflow
  - MessageChannelEnumerate Heap Overflow
- Use After Free
  - PPB_Audio_Create SRPC Channel Use After Free
- Information Leak
  - PPB_FileIO_Write Out Of Bounds Read Information Leak
- Other
  - PPB_URLLoader_Open CORS Request Allows For Header Injection
  - PPB_FileRef_Create Potential Directory Traversal

Pepper Proxy Source Audit

- Length calculations are difficult with serialized binary data
- Callbacks and channel related structures create complexities on top of application logic
- Confusion over whether PPAPI or the proxy should validate data can lead to vulnerabilities

Chrome Shaker

- A pepper proxy fuzzer
- Developed under contract to Google January 2012
- Joint project with Matasano Security (Cody Brocious)
- Google deployed it in their fuzzing farm
- https://code.google.com/p/chrome-shaker/

Chrome Shaker

- Simple NEXE template in C++
  - NEXE template code, random numbers, memory etc...
- Python tool that parses pepper proxy IDL files
  - Generates C++ into the basic template
  - Sets up each PPAPI interface in the ‘C’ style
  - Calls interfaces in random order with random arguments

Chrome Shaker

- Fuzzing from a NEXE is not easy
- You can't log to disk
  - How do you write to disk from within two sandboxes?
    - Use the interface (FileIO) that you're currently fuzzing?
    - We had to resort to STDOUT
- You need a constant source of random data
  - We cheated and call into JavaScript for window.crypto
- Some code paths require calling interfaces in order
  - We have an API dependency file in yaml for this

NaCl Exploitation

- Exploitation of a vulnerability in a trusted component allows arbitrary code execution in the outer sandbox
  - Inner sandbox has been defeated
  - Chrome renderer sandbox is still enforced
  - This is not equivalent to a WebKit/V8 bug
    - The pepper proxy exposes interfaces at a lower level that JavaScript cannot reach

The Conclusion

- NaCl is trying to solve a difficult problem
- NaCl is not ActiveX or NPAPI
- NaCl research into SFI and the inner sandbox will influence future sandbox designs with similar goals
- As the trusted code attack surface grows more implementation vulnerabilities will be found
  - The NaCl design helps to mitigate their impact
- Enable Chrome's Click-To-Play

The End

Questions?

leafsr.com
Chris.Rohlf@gmail.com

BlackHat Survey

Please fill out the BlackHat survey!