ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

GOOGLE NATIVE CLIENT: IL COMPUTER NEL BROWSER, ANALISI E UTILIZZO DELLO STRUMENTO

Relazione finale in:

Reti di Calcolatori

Relatore:

D'Angelo Gabriele

Presentata da:

Lucchini Davide

Sessione II

Anno Accademico 2009/2010

Indice

Introduzione....4
CAPITOLO 1....8
La tecnologia Google Native Client....8
  Introduzione alla Tecnologia....8
    L'utilizzo del Native Client nei browser....9
  Portabilità....11
    Configurazioni testate....13
    Integrazione con Google Chrome....13
    Porting dei linguaggi di programmazione....14
    Esempio di porting....14
  Sicurezza....15
    La sandbox interna....16
    Superficie di Attacco....18
  Confronto con altre tecnologie....19
    System request moderation....20
    Fault Isolation....21
    Trust With Authentication....24
CAPITOLO 2....26
Funzionamento e Performance....26
  Funzionamento del Native Client....26
    Integrazione con Chrome....29
  Comunicazioni....30
    IMC....31
    PPAPI....32
  Performance....34
    Test delle performance computazionali/grafiche....36
    Bullet....38
    Quake....39
CAPITOLO 3....40
Utilizzo e Valutazione di Google Native Client....40
  Strumenti per l'utilizzo del Native Client....40
    Native Client SDK....41
  Esempi....43
    Hello World....45
    Pi Generator (Pepper 2D and threading)....45
    Sine Wave Synthesizer (Pepper Audio)....46
    Tumbler (Pepper 3D)....47
  Valutazione del Native Client....48
    Misurazioni Effettuate....56
Conclusioni....58
Bibliografia....61

Introduzione

Secondo Google, in futuro, si potrebbe avere bisogno solo di un browser sul proprio sistema. Eppure si potrà continuare a fare tutto ciò per cui di solito si utilizza il computer senza dover installare strumenti aggiuntivi: elaborare immagini, scrivere testi, masterizzare dischi o divertirsi con giochi ad alta risoluzione grafica.

Ciò è possibile grazie ad una semplice estensione del browser. Il Native Client (abbreviato in NaCl, una allusione al cloruro di sodio) esegue il codice di un programma caricato dal browser attraverso il web sul computer di casa. Il clou è che il codice del programma può essere scritto in C o C++, di solito i linguaggi più utilizzati per le applicazioni standalone.

Il Native Client è un progetto open-source in fase di sviluppo da parte di Google. Esso è cominciato nel 2008 e una prima versione dimostrativa del plugin è stata rilasciata nel Giugno 2009. Nel novembre dello stesso anno Google ha fatto richiesta per un brevetto relativo al Native Client [34]. Da allora sul sito del NaCl [7] si sono susseguiti vari aggiornamenti, comprensivi di documenti ufficiali [1] e varie documentazioni.

Il Native Client è rilasciato in licenza sotto la New BSD License.

Le licenze BSD [5] sono una famiglia di licenze permissive per software. Esse risultano più aperte e libere di altre, ad esempio della licenza GNU General Public License (GNU GPL) [44], non avendo fra i propri obiettivi quello di proteggere la libertà del software cui sono applicate ma semplicemente di rendere per tutti il software completamente libero, accessibile e modificabile.

Lo scopo del Native Client è coniugare i vantaggi forniti dalle applicazioni web, quali neutralità, portabilità e sicurezza, con i vantaggi tipici delle applicazioni desktop, primo fra tutti la capacità di sfruttare appieno la potenza di calcolo del computer. "I PC di oggi possono eseguire miliardi di istruzioni per secondo, ma le attuali applicazioni web possono accedere solo a una piccola frazione di questa potenza di calcolo" [45], ha affermato Brad Chen, del team di sviluppo di Google.

Il Native Client è uno strumento portabile fra i vari sistemi operativi e browser e fornisce caratteristiche orientate alle performance generalmente assenti dagli ambienti di programmazione per applicazioni web, come ad esempio il supporto ai thread. Sotto questo punto di vista il Native Client assume una notevole importanza nel panorama informatico delle tecnologie: se gli obiettivi promessi saranno raggiunti il NaCl potrebbe rivoluzionare tutto il web.

Il team di Google sostiene che il Native Client non è stato pensato per sostituire le tecnologie web esistenti, ma per essere utilizzato dagli sviluppatori insieme ad esse per creare applicazioni che offrano un' esperienza più ricca e dinamica di quanto fosse possibile fino ad ora.

L'intento di questa tesi è illustrare il progetto del Google Native Client descrivendolo e analizzandone i vari componenti. Partendo da un punto di vista teorico, supportato dai documenti rilasciati dagli stessi autori, si passerà ad un' analisi dei vari aspetti che lo caratterizzano, per arrivare infine ad una valutazione dello stato attuale del NaCl.

Tutto questo per provare a verificare se le intenzioni iniziali degli sviluppatori sono state raggiunte o meno. In particolare provare a dimostrare se il Native Client ha prospettive di utilizzo future migliori rispetto alle altre tecnologie web perché effettivamente gestisce meglio gli aspetti fondamentali di sicurezza e prestazioni.

Nei capitoli della tesi verrà prima di tutto descritta questa tecnologia definendo gli obiettivi e elencando i componenti da cui è composta. In seguito verranno descritti specificatamente questi componenti e infine valutati in base allo stato attuale del Native Client.

Nel primo capitolo viene prima di tutto introdotta la tecnologia del NaCl e il suo utilizzo nei browser web. In seguito si pone l'attenzione su due degli obiettivi fondamentali proposti da Google: portabilità e sicurezza. Entrambi gli argomenti verranno trattati per descrivere come il Native Client li implementa. Per finire si farà un confronto con le altre tecnologie web.

Nel secondo capitolo si da enfasi al funzionamento e alle performance. Verrà spiegato come un' applicazione NaCl viene eseguita nel browser e come vengono gestite le comunicazioni fra browser e Native Client. Si discuteranno poi le performance considerando i dati dei test effettuati dagli sviluppatori del NaCl per potersi fare un' idea della differenza che intercorre tra le prestazioni di una normale applicazione e quelle di un' applicazione di tipo Native Client.

Infine nel terzo e ultimo capitolo l'argomento principale sarà la valutazione degli strumenti precedentemente analizzati. Ciò avverrà tramite la descrizione di un' esperienza diretta avuta con gli esempi e l'esposizione dell'implementazione effettuata di una semplice applicazione NaCl.

La tesi si concluderà con una valutazione basata su questa esperienza.

CAPITOLO 1

La tecnologia Google Native Client

Introduzione alla Tecnologia

Il Native Client è una sandbox per codice nativo x86 untrusted [1].
Per codice nativo si intende codice che è compilato per essere eseguito su un particolare processore, nel nostro caso codice nativo x86. Il termine x86 [5] si riferisce ad una famiglia di Instruction Set Architecture (ISA) basata sull'Intel 8086. L'ISA [5] è la parte dell'architettura del computer legata alla programmazione, compresi i tipi di dati nativi, istruzioni, registri, modalità di indirizzamento, architettura della memoria, interrupt e gestione delle eccezioni, I/O esterno. Un ISA include una specifica del set di opcodes (operation code), la porzione di un istruzione in linguaggio macchina che specifica il tipo di operazione che deve essere eseguita, e i comandi nativi implementati da un particolare processore.
Per quanto riguarda il termine “untrusted” [5], esso viene usato nel campo delle sicurezza informatica per indicare un codice proveniente da una sorgente sconosciuta che potrebbe potenzialmente danneggiare un sistema. A tale codice è generalmente permessa l'esecuzione in un sistema informatico, ma con alcune protezioni. Ad esempio, dovrebbe essere eseguito in una sandbox [5].
Una sandbox è un meccanismo di sicurezza per separare i programmi in esecuzione. È spesso usata per eseguire codice non testato, o programmi untrusted provenienti da terze parti non verificate o da fornitori e utenti non affidabili. La sandbox tipicamente fornisce un insieme strettamente controllato di risorse, uno spazio sul disco e sulla memoria dedicato ad una memorizzazione temporanea dei dati (scratch space), per eseguire al suo interno programmi “ospiti”. L'accesso alla rete, l'abilità di ispezionare il sistema host o di leggere da dispositivi di input, sono solitamente disabilitati o pesantemente limitati. In questo senso, le sandbox sono un esempio specifico di virtualizzazione.

L'utilizzo del Native Client nei browser

Come piattaforma applicativa, il moderno browser web fornisce insieme una considerevole combinazione di risorse: pieno accesso alle risorse di Internet, linguaggi di programmazione ad alta produttività come JavaScript, e il contributo del Document Object Model [9] per le presentazioni grafiche e l'interazione degli utenti.
Il DOM è lo standard ufficiale del W3C per la rappresentazione di documenti strutturati in maniera che siano neutrali sia per la lingua che per la piattaforma.
Mentre queste forze pongono il browser all'avanguardia come mezzo per lo sviluppo di nuove applicazioni, esso rimane arcaico in una dimensione critica: prestazione computazionale.
Grazie alla legge di Moore (**"**Le prestazioni dei processori, e il numero di transistor ad esso relativo, raddoppiano ogni 18 mesi**"**) e allo zelo con cui essa è stata osservata dalla comunità hardware, molte applicazioni di interesse hanno adeguate performance in un browser nonostante il suo "handicap".
Eppure alcuni set di computazioni rimangono generalmente irrealizzabili per le applicazioni basate su browser a causa delle restrizioni dovute alle prestazioni. Ad esempio: simulazione della fisica Newtoniana, computational fluid-dynamics (un settore della meccanica dei fluidi che utilizza metodi numerici e algoritmi per risolvere problemi che coinvolgono flussi di liquidi) e rendering di scenari ad alta risoluzione.
Inoltre, l'ambiente attuale tende a precludere l'uso di grandi corpi di codice ad alta qualità sviluppati in linguaggi diversi da JavaScript.
La soluzione di questi problemi è legata alla possibilità di eseguire il codice nativo come applicazione web [1].
Riuscendo nell'intento, teoricamente si potrebbero raggiungere le stesse prestazioni ottenute con l'esecuzione del programma in locale, sul computer dell'utente.
Attualmente, per ovviare a questo problema, i moderni browser web prevedono meccanismi di estensione come ActiveX e NPAPI [5].

Queste architetture permettono ai plugin di aggirare i meccanismi di sicurezza altrimenti applicati al contenuto web, dando a questi allo stesso tempo un pieno accesso alle prestazioni native.
Un plugin è un programma non autonomo che interagisce con un altro per ampliarne le funzioni. Ad esempio, un plugin per un software di grafica permette l'utilizzo di nuove funzioni non presenti nel software principale.
Data questa organizzazione, e l'assenza di provvedimenti tecnici per limitare questi plugin, le applicazioni web che desiderano usare il codice nativo devono affidarsi a provvedimenti alternativi per la sicurezza. Ad esempio, costringere l'utente a stabilire manualmente la sicurezza di un componente attraverso le finestre di pop-up (come per ActiveX), o l'installazione manuale di un'applicazione che viene eseguita da console.
Storicamente, questi provvedimenti sono stati inadeguati per prevenire l'esecuzione di codice nativo maligno, portando a disagi e danni economici.
Come conseguenza, c'è un pregiudizio tra gli esperti verso le estensioni di codice nativo per le applicazioni basate su browser e sfiducia tra la maggior parte degli utenti dei computer.
Mentre si riconosce la non sicurezza dei sistemi correnti per incorporare codice nativo nelle applicazioni web, si può anche osservare che non c'è una ragione fondamentale del perché il codice nativo dovrebbe essere non sicuro.
Nel Native Client [1], viene separato il problema della sicurezza dell'esecuzione nativa, da quello di estendere la fiducia, gestendo entrambi i problemi separatamente.
Concettualmente, il Native Client è organizzato in due parti: un ambiente di esecuzione limitato per il codice nativo con lo scopo di prevenire side effect involontari, e un insieme di servizi software (run-time environment) per ospitare queste estensioni di codice nativo attraverso le quali i side effect ammissibili possono avvenire in sicurezza.
Per side effect [1] si intendono comportamenti non previsti che avvengono durante l' esecuzione del codice nativo, come ad esempio un accesso non moderato all'interfaccia delle system call del sistema operativo nativo.

Il contributi principali del Native Client sono:

- una infrastruttura per il sistema operativo e moduli binari x86 eseguiti in sandbox e portabili attraverso i diversi browser, supporto per costrutti di prestazioni avanzate come thread, istruzioni SSE, compiler instrinsics e hand-coded assembler
- un open system (sistema informatico che fornisce combinazioni di interoperabilità, portabilità e open software standard) progettato per il facile inserimento di nuovi compilatori e linguaggi
- perfezionamenti al CISC software fault isolation [3], usando segmenti x86 per una maggiore semplicità e per ridurre il tempo di overhead (parametro fondamentale per lo studio delle prestazioni di un sistema operativo: esso rappresenta il tempo medio di CPU necessario per eseguire i moduli del kernel).

Queste funzionalità sono combinate in una infrastruttura che garantisce una gestione sicura dei side effect e supporta la comunicazione locale.
Complessivamente, il Native Client fornisce un esecuzione sandboxed del codice nativo e la portabilità tra vari sistemi operativi, promettendo di mantenere le prestazioni del codice nativo per il browser.

Portabilità

La portabilità [5] è la caratteristica del codice di base di un software di poter essere riutilizzato quando si sposta il programma da un ambiente ad un altro, senza dover riscrivere il codice da zero.
L'operazione di *porting*, cioè la scrittura di un port, è solitamente necessaria a causa di: differenze tra le CPU, diverse interfacce (Application Programming Interface API) dei sistemi operativi, diversità dell'hardware o incompatibilità nell'implementazione del linguaggio di programmazione sull'ambiente per cui deve essere compilato il programma.
Un componente software è portabile se il costo del porting è minore di quello necessario per riscriverlo da zero. Minore è il costo del porting, più portabile sarà il software; in questo senso, la portabilità si può considerare un caso particolare di riusabilità del software.

La portabilità può essere raggiunta su tre livelli diversi:

- i file di installazione del programma possono essere trasferiti su un altro computer
- il programma può essere reinstallato su un altro computer
- i file sorgenti possono essere ricompilati per un altro computer.

Per il Native Client si tratta del secondo caso. Infatti, l'installazione del plugin del NaCl è specifica per sistema operativo e browser, quindi volendo passare da un sistema ad un altro, occorrerà reinstallare il plugin. Possiamo perciò affermare che il NaCl in generale è portabile [1].

La portabilità è stato un punto cruciale dello sviluppo, visti gli obiettivi, per fare in modo che il Native Client possa essere utilizzato da tutti. Una pagina web si deve presentare e deve funzionare allo stesso modo, indipendentemente dal browser, dal sistema operativo, o dal tipo di hardware su cui viene eseguita [8]. Mentre la portabilità riguardante il browser e il sistema operativo è già verificata, quella hardware necessita di un supporto specifico.

Inizialmente, al primo rilascio del Native Client, venivano supportati i sistemi operativi più comuni: Windows, Mac Os X, e Linux; ma solo su macchine con processori x86 a 32 bit.

Recentemente il supporto è stato esteso alle altre architetture: x86-64 e ARM [10].
Nonostante la possibilità di eseguire il Native Client sulle architetture più comuni, i responsabili del NaCl riconoscono che ciò non sia sufficiente: se si afferma una nuova architettura, essa deve essere in grado di eseguire tutti i moduli NaCl già rilasciati, senza il bisogno di sviluppatori per ricompilare il codice. Per questo motivo si sta studiando una tecnologia [2] che permetterà agli sviluppatori di distribuire una rappresentazione portabile dei programmi NaCl. Usando questa tecnologia, un browser in esecuzione su qualsiasi tipo di processore potrà tradurre la rappresentazione portabile in un codice oggetto nativo senza accedere al codice sorgente del programma.

Configurazioni testate

Per utilizzare il Native Client sono necessari un browser e un sistema operativo supportati. Nonostante l'obiettivo della portabilità sia tenuto in grande considerazione, il Native Client è ancora in fase di sviluppo, per cui al momento il NaCl non è utilizzabile con ogni configurazione.
Per quanto riguarda Windows i sistemi operativi testati sono: Windows Xp, Windows Vista e Windows 7. I browser: Firefox [12] (dalla versione 3 in poi), Chrome [11], Safari [14] e Opera [13], mentre Internet Explorer non è supportato, anche se è in progetto per il futuro.
Per Linux sono stati testati Firefox e Chrome con i sistemi operativi Ubuntu 8.04 (Hardy Heron), Ubuntu 9.04 (Jaunty Jackalope) e Ubuntu 9.10 (Karmic Koala), sia a 32 che a 64 bit.
Per Mac sono stati testati Firefox e parzialmente Camino [15] e Safari su sistema operativo Mac OS X 10.5.
Per quanto riguarda Windows e Mac attualmente il supporto è disponibile solo per le versioni a 32 bit, entro la fine del 2010 dovrebbe essere esteso alle versioni a 64 bit.

Integrazione con Google Chrome

Per il browser di Google, Chrome, viene utilizzato un approccio particolare [6].
È in fase di sviluppo un' integrazione vera e propria che si differenzia dagli altri browser per il funzionamento del NaCl. Invece di essere installato come un plugin e lavorare in un processo separato è intenzione degli sviluppatori di Google integrare all'interno di Chrome il Native Client, in modo che sfrutti la sua architettura per funzionare al meglio. I benefici principali di questa operazione saranno tre: incremento delle performance, maggiore sicurezza, riutilizzo della infrastruttura di Chrome.
Le performance sono incrementate in quanto il Native Client viene eseguito all'interno del renderer process (il processo associato alla pagina visualizzata) eliminando la necessità di comunicazioni tra il processo e il codice del plugin NaCl.
Maggiore sicurezza perché eseguendo il codice NaCl nella sandbox di Chrome viene aggiunto uno strato ulteriore di protezione (la sandbox esterna).

Riutilizzo della infrastruttura, affidandosi a Chrome per nuove build, aggiornamenti automatici, crash report ecc. E per l'uso di ChromeFrame [17], un plug-in open-source per fornire agli utenti che utilizzano Internet Explorer la possibilità di usare le tecnologie web di Google Chrome, e quindi, vista l'integrazione, anche di usare il Native Client.

Porting dei linguaggi di programmazione

È in previsione per il futuro, anche se non è uno degli obiettivi primari, la possibilità per i programmatori di usare il proprio linguaggio di programmazione nel browser, senza essere costretti ad usare JavaScript. Per questo motivo è in sviluppo il porting di diversi compilatori e interpreti comuni.
A questo proposito sul sito del Native Client [7] è stata messa a disposizione una procedura generale per il porting di codice. Anche se l'attenzione principale è posta su progetti che utilizzano Autoconf, la maggior parte delle informazioni è rilevante anche per gli altri.
GNU Autoconf [18] è uno strumento per generare script di configurazione per programmi C,C++,Fortran 77, Fortran, Erlang, Objective C su sistemi operativi unix-like. Gli script analizzano il computer di un utente e configurano il pacchetto software prima dell'installazione.
L'idea è di generare un makefile accettabile, editando a mano il file di configurazione, se necessario, per includere i parametri e i percorsi del Native Client che permettano di creare un makefile compatibile.

Esempio di porting

È stato eseguito il porting di un implementazione interna del decoder video H.264 [5] per valutare la difficoltà dello sforzo del porting.
L'applicazione originale convertiva il video H.264 in un file di formato raw, implementato in circa 11000 righe di C per l'ambiente standard GCC [19] su Linux.
Il porting ha richiesto circa venti linee di codice C aggiuntive, più della metà utilizzate per codice di controllo sugli errori. Oltre a riscrivere il makefile, nessun altra modifica è stata necessaria. Questa esperienza è coerente con quella generale avuta con il Native Client: librerie ereditate da Linux che non richiedono intrinsecamente la rete e il disco fisso generalmente possono essere portate con uno sforzo minimo.

Sicurezza

Il Native Client funziona come una specie di microkernel [4] che assume il ruolo di sistema operativo per le sue applicazioni, accedendo direttamente alle risorse hardware. Il codice da eseguire deve essere strettamente sorvegliato dal NaCl affinché il programma eseguito non tenti in qualsiasi modo di compromettere il sistema. L'aspetto della sicurezza è essenziale anche perché l'utente non si accorge dell'attivazione del NaCl: l'estensione di Google si avvia in background non appena la pagina web visitata avvia il programma.
Un' applicazione NaCl è composta da una collezione di componenti trusted e untrusted.
Il Native Client deve essere in grado di gestire i moduli untrusted da qualsiasi sito web con sicurezza comparabile a quella di sistemi indiscutibili come JavaScript; deve quindi verificare l'attendibilità e la pericolosità del codice.
Quando si presenta al sistema, un modulo untrusted potrebbe contenere codice e dati di qualsiasi tipo. Una conseguenza è che il runtime environment del NaCl deve essere in grado di confermare che il modulo è conforme ad una serie di regole di validità. Quelli non conformi sono rifiutati dal sistema.
Una volta che un modulo NaCl conforme viene accettato per l'esecuzione, il runtime del NaCl deve vincolare la sua attività per prevenire side effect involontari.
Il modulo NaCl potrebbe arbitrariamente combinare l'intera varietà dei comportamenti permessi dall'ambiente di esecuzione con l'intento di compromettere il sistema.
Ad esempio potrebbe inviare dati arbitrari tramite l'interfaccia di comunicazione tra moduli (InterModule Communication [1]).
Per evitare tutto questo, i moduli NaCl devono essere vincolati, dall'architettura e dalle regole di validità per il codice untrusted, all'interno di una sandbox, o meglio, di due: sandbox interna (inner sandbox [1]), e sandbox esterna (outer sandbox [1]).

Queste due sandbox garantiscono due strati di protezione e agiscono nel seguente modo.
La sandbox interna decompila il codice. Un validatore [1] analizza i componenti poco sicuri verificando l'affidabilità dell'esecuzione del codice e controllando che il programma da eseguire contenga solo comandi sicuri. Mentre la sandbox interna è sempre uguale, quella esterna si adegua al sistema operativo e verifica le chiamate di sistema in esecuzione, bloccando quelle non consentite.
Ad esempio per prevenire l'accesso involontario alla rete, le network system call (come connect( ) e accept( )) vengono bloccate e il modulo NaCl può accedere alla rete solo con JavaScript (quindi tramite browser). Questo tipo di accesso è soggetto agli stessi vincoli che sono applicati agli altri accessi JavaScript, senza effetti sulla sicurezza di rete.
Il nucleo della sicurezza del NaCl è la sandbox interna che viene considerata abbastanza robusta da non aver bisogno di altri meccanismi di sicurezza. Nonostante ciò per fornire una difesa più profonda, è stata sviluppata la sandbox esterna.
Essa è sostanzialmente simile a strutture come systrace [5], un utility di sicurezza che limita gli accessi di un applicazione al sistema, utilizzando criteri di accesso per le system call. In questo modo si possono mitigare gli effetti di buffer overflow e altre vulnerabilità della sicurezza.

La sandbox interna

La sandbox interna utilizza analisi statiche per trovare difetti di sicurezza nel codice x86 untrusted.
Suddette analisi non sono considerate generalmente attendibili, su codice x86 arbitrario, a causa di pratiche che non permettono di effettuare una corretto disassemblaggio, come self-modifying code (codice auto-modificante).
Nel Native Client queste pratiche non sono permesse tramite un set di allineamenti e regole strutturali [1] che, quando osservate, assicurano che il modulo di codice nativo possa essere disassemblato in modo affidabile, così che tutte le istruzioni accessibili vengano identificate durante il disassembly. Per disassembly si intende l'operazione di tradurre il codice da linguaggio macchina a linguaggio assembly.

I vincoli del codice compilato (oggetto) del NaCl sono i seguenti:

1: Una volta caricato in memoria il codice oggetto non è scrivibile, vincolato, durante l'esecuzione, dai meccanismi di protezione a livelli del SO.

2: Il codice oggetto è staticamente collegato ad un indirizzo di partenza pari a 0, con il primo byte del testo a 64k.

3: Tutti i trasferimenti indiretti di controllo utilizzano una pseudo-istruzione *nacljmp* [1] (composta da un' istruzione *and* a tre byte e da un' istruzione *jump* a due byte)

4 : Il codice oggetto è inserito nella pagina più vicina con almeno un' istruzione *htl(0xf4)*

5: Il codice oggetto non contiene istruzioni o pseudo-istruzioni che superano il limite di 32-byte.

6: Tutti gli indirizzi relativi a istruzioni valide sono raggiungibili da un disassembly fall-trough che parte all'indirizzo di caricamento(indirizzo base).

7: Tutti i trasferimenti diretti di controllo fanno riferimento ad istruzioni valide.

Insieme, il vincolo 1 e 6 rendono il disassembly affidabile.

Con uno strumento di disassembly di questo tipo, il validatore del NaCl può assicurare che l'eseguibile includa solamente un sottoinsieme di istruzioni lecite, disabilitando le istruzioni macchina pericolose.

Per eliminare i side effect il validatore deve verificare 4 sotto problemi:

– Integrità dei dati: nessun caricamento o salvataggio al di fuori dei dati della sandbox

– Disassembly affidabile (verificato dai vincoli 1 e 6)

– No istruzioni untrusted

– Controllo dell'integrità del flusso di esecuzione (Control-Flow Integrity)

Per risolvere questi problemi, il NaCl utilizza il software fault isolation CISC [3], un metodo per effettuare il sandboxing del codice binario untrusted. Il sistema combina l'utilizzo di memoria segmentata con le precedenti tecniche software fault isolation CISC [20].

La segmentazione [5] è una comune tecnica di gestione della memoria che suddivide la memoria fisica disponibile in blocchi di lunghezza fissa o variabile detti segmenti. Il Native Client utilizza segmenti di 32 byte, e le singole istruzioni non possono superare questo limite. Ad un segmento può essere associata una combinazione di permessi in base ai quali si determina a quali processi è consentito o negato un certo tipo di accesso. In questo modo ad esempio è possibile distinguere segmenti di programma, di dati e di stack. Gestendo la memoria si può così fare in modo che da un segmento di programma vengano caricate solo istruzioni e non, ad esempio, dati. Questo permette di implementare una sandbox per i dati senza essere costretti ad utilizzarne una per il caricamento e la memorizzazione delle istruzioni.

Sfruttando l'hardware esistente per implementare questa serie di controlli, vengono semplificati enormemente quelli richiesti a runtime per vincolare i riferimenti alla memoria, riducendo l'impatto sulle prestazioni dei meccanismi di sicurezza.
La sandbox interna è usata per creare un sotto-dominio di sicurezza all'interno di un processo del sistema operativo nativo.
Con questa organizzazione è possibile collocare un servizio affidabile di sottosistema a runtime, all'interno dello stesso processo del modulo untrusted dell'applicazione, che include un meccanismo a trampolino [1] per permettere un trasferimento di controllo sicuro dal codice trusted a quello untrusted e viceversa.
Generalmente un sistema operativo non è privo di difetti: la sandbox interna, non solo isola il sistema operativo dal modulo nativo, ma aiuta anche ad isolare il modulo nativo dal sistema operativo.

Superficie di Attacco

I componenti del NaCl che un aspirante “aggressore” potrebbe cercare di sfruttare sono i seguenti:

- sandbox interna: binary validation
- sandbox esterna: intercettazione delle chiamate del sistema operativo
- service runtime: moduli di caricamento binari
- service runtime: interfacce del trampolino
- interfaccia di comunicazioni IMC
- interfaccia NPAPI

Per aumentare ulteriormente la sicurezza, in aggiunta alla sandbox interna e a quella esterna, il progetto di sistema include anche black-list[5] di CPU e di moduli NaCl. Una black-list è un meccanismo solitamente utilizzato per il controllo degli accessi: a tutti gli utenti è permesso l'accesso tranne a quelli segnati sulla lista nera, per contrasto la white-list funziona al contrario. Nel caso della CPU black-list sono prese in considerazione, al posto degli utenti, le istruzioni che la CPU dovrà eseguire, mentre per la NaCl module black-list i moduli NaCl. Questi meccanismi permettono di incorporare strati di protezione basati sulla fiducia degli sviluppatori del Native Client nella robustezza dei vari componenti e nella comprensione di come raggiungere il miglior bilanciamento tra performance, flessibilità e sicurezza. In futuro potrebbero essere rilasciate anche delle white-list se saranno necessarie per difendere il sistema.

Confronto con altre tecnologie

Esistono altre tecnologie che permettono di eseguire il codice nativo nel browser, soluzioni simili al Native Client che però presentano delle differenze.
Un esempio comune può essere la tecnologia NPAPI (Netscape Plugin Application Programming Interface), un' architettura multi-piattaforma utilizzata da molti browser.
NPAPI associa alcuni tipi di dati (più precisamente alcuni tipi di MIME, es. "audio/mp3") ad un plugin. Quando il browser visita una pagina contenente quel tipo di contenuti carica il plugin associato assegnandogli uno spazio delimitato della pagina e affidandogli il compito di gestire i dati. Il problema di questa tecnologia è la sicurezza, eseguendo le istruzioni native con gli stessi privilegi del processo ospitante, un plugin malevolo potrebbe danneggiare il sistema.
Il Native Client utilizza una particolare implementazione NPAPI per comunicare con il browser, ma le misure di sicurezza adottate sono in grado di risolvere il problema.

Le altre tecnologie che utilizzano tecniche per eseguire in sicurezza il codice di terze parti, generalmente hanno problemi in tre categorie (che verranno discusse in seguito): system request moderation (il sistema necessita di moderazione), fault isolation (isolazione degli errori), e trust with authentication (fiducia con autenticazione).

System request moderation

Per system request moderation si intende la necessità da parte della tecnologia che esegue il codice nativo, di controllare alcuni aspetti del sistema operativo che potrebbero creare problemi nell'esecuzione del codice. Tra queste tecnologie ci sono Android [21] di Google e Xax [22] di Microsoft Research. Android è un software per dispositivi mobili che comprende varie applicazioni e un sistema operativo.
Esso utilizza una sandbox per eseguire applicazioni di terze parti. Ogni applicazione di Android viene eseguita come un differente utente Linux, e un sistema di contenimento partiziona l'attività di system call in gruppi con permessi come "Network communication" e "Your personal information". Il riconoscimento da parte dell'utente dei permessi richiesti è necessario prima di installare una applicazione di terze parti.
La separazione degli utenti nega di per sé intercomunicazioni potenzialmente utili. Per ovviare a questo problema, Android ha costituito un modello di permessi, chiamato "Intent system and ContentProvider data access model", all'interno del Binder, un meccanismo di comunicazioni interprocess (fra più processi).
Xax invece è un modello per i plugin dei browser che consente agli sviluppatori di sfruttare gli strumenti esistenti, librerie e interi programmi per rilasciare applicazioni per il web ricche di funzionalità. Xax è forse il progetto più simile al Native Client in termini di obiettivi, nonostante il suo approccio implementativo sia piuttosto differente. Esso utilizza l'intercettazione delle system call basandosi su ptrace su Linux e su un "kernel device driver" su Windows. Gli sviluppatori del NaCl respingono questo approccio kernel-based considerandolo impraticabile a causa di problemi riguardanti il supporto.

In particolare si nota che l'implementazione Windows dello Xax richiede un driver di dispositivo kernel-mode che deve essere aggiornato per ogni build supportata da Windows, un metodo oneroso anche se implementato dal venditore stesso del sistema operativo.
Inoltre, ci sono difetti conosciuti in ptrace che Xax non considera. Nonostante gli autori riconoscano uno di questi problemi nel loro documento [22], sul web sono elencati quarantuno problemi relativi a ptrace [29].
Per merito della sua sandbox interna, puramente user-space, il Native Client è meno vulnerabile a questi complicati problemi del kernel. Xax è inoltre esposto ad attacchi denial-of-service che possono causare un riavvio o un hang (il sistema non rispondere più agli input) della macchina. Il denial of service [5] (letteralmente negazione del servizio*)* è un attacco in cui si cerca di portare il funzionamento di un sistema informatico che fornisce un servizio, ad esempio un sito web, al limite delle prestazioni, fino a renderlo non più in grado di erogare il servizio.
Visto che il Native Client esamina ogni istruzione e rifiuta i moduli con istruzioni risultanti sospette, esso riduce significativamente la superficie di attacco e include meccanismi adeguati per la difesa contro nuovi exploit che potrebbero essere scoperti.
A parte queste differenze relative alla sicurezza, Xax non supporta il threading, un aspetto ormai essenziale vista la tendenza verso le CPU multi-core [5].

Fault Isolation

Dopo che la nozione di software fault isolation è diventata comune [20], i ricercatori hanno descritto sistemi complementari e alternativi. Alcuni lavorano direttamente sul codice macchina x86. Altri sono basati sulle rappresentazioni di programmi intermediari, come linguaggi type-safe, macchine virtuali astratte o rappresentazioni di compilatori intermediari. Essi utilizzano una rappresentazione portabile, permettendo la portabilità dell'ISA ma creando un ostacolo alle performance. Problema che nel NaCl viene risolto lavorando direttamente sul codice macchina.

Un vantaggio del Native Client, infatti, è quello di esprimere il sandboxing direttamente in codice macchina in modo da non richiedere un compilatore affidabile. Questo riduce notevolmente la dimensione dei componenti necessari alla sicurezza del NaCl semplificandone l'implementazione. Inoltre ha l'ulteriore beneficio dell'apertura del sistema a tool chain di terze parti.

Tra le tecnologie che presentano problemi nell'isolazione degli errori ci sono i sistemi CFI [24], VINO [26] e Nooks[27].

Il CFI (Control Flow Integrity [24]) è una tecnica di sicurezza per evitare che attacchi al software possano arbitrariamente controllare il funzionamento del programma. Confrontato al Native Client, il CFI fornisce un più fine controllo dell'integrità del flusso di esecuzione. Mentre il NaCl ha la sola garanzia che il controllo del flusso di esecuzione indiretto mirerà ad un indirizzo allineato nel segmento di testo, il CFI può limitare un trasferimento di controllo specifico ad un sottoinsieme equamente arbitrario di obiettivi conosciuti. Questo controllo più preciso può essere utile in alcuni scenari, come nell'assicurazione dell'integrità di una interpretazione da un linguaggio ad alto livello, ma esso non è vantaggioso per il Native Client, dato che l'intenzione è quella di permettere un controllo del flusso di esecuzione del tutto arbitrario, anche hand-coded assembler, fin tanto che l'esecuzione riguarda codice conosciuto e gli obiettivi sono allineati. Inoltre la complessità del controllo utilizzato dal CFI incide notevolmente sulle prestazioni, infatti, l'overhead medio del CFI è di tre volte superiore rispetto a quello del Native Client (rispettivamente 15% e 5%).

VINO [26] è un sistema che si basa sull'isolamento delle estensioni untrusted del kernel. Il suo scopo è fornire ai programmi la possibilità di riutilizzare le primitive dei sistemi operativi e di poter accedere ad un' unica interfaccia "universale" per l'utilizzo delle risorse. VINO presenta alcune similitudini con il Native Client. Prima di tutto utilizza lo stesso metodo del NaCl per implementare la sicurezza: software fault isolation basata sul sandboxing delle istruzioni macchina. Inoltre utilizza anch'esso una tool chain per la compilazione per aggiungere istruzioni sandboxing al codice macchina x86, usando il C++ per implementare le estensioni.

Al contrario del Native Client però, VINO non ha una validatore binario: esso si basa su un compilatore affidabile. Un validatore per VINO sarebbe più difficile da implementare di quello del Native Client, visto che il suo validatore dovrebbe forzare l'integrità dei dati, raggiunta nel Native Client con la memoria segmentata.

Il sistema Nooks [27] migliora l'affidabilità del kernel del sistema operativo, isolando il codice trusted del kernel dai moduli untrusted dei driver delle periferiche, usando uno strato trasparente del SO chiamato Nooks Isolation Manager (NIM). Come il Native Client, il NIM utilizza la protezione della memoria per isolare i moduli untrusted, ma visto che opera nel kernel, i segmenti x86 non sono disponibili. Quindi, al posto della segmentazione, utilizza un' altra tecnica di protezione della memoria: il paging. Gli algoritmi di paging [5] dividono la memoria in parti di dimensioni minori, e la allocano usando pagine come blocco minimo di lavoro. Alla memoria fisica della RAM si somma la memoria virtuale del file di paging, che è un file creato per simulare una quantità maggiore di memoria RAM. Una page table è la struttura dati usata da questa memoria virtuale per memorizzare le associazioni tra indirizzi virtuali e indirizzi fisici. Il NIM usa una page table privata per ogni modulo di estensione. Per cambiare i domini di protezione, aggiorna l'indirizzo base della page table, un' operazione che provoca un side effect: lo scorrimento del Translation Lookaside Buffer (TLB) x86. Il Translation Lookaside Buffer [5] è un buffer usato per velocizzare la traduzione degli indirizzi virtuali in indirizzi fisici. In questo modo, la gestione del NIM delle page table indica un' alternativa alla protezione fornita dalla segmentazione della memoria che viene utilizzata dal Native Client. Mentre un' analisi delle performance di questi due approcci probabilmente esporrebbe differenze interessanti, il confronto è discutibile sull'x86 visto che un meccanismo è disponibile solo all'interno del kernel e l'altro solo all'esterno. Una distinzione critica tra il Nooks e il Native Client è che il primo è progettato solo per la protezione per errori (bug) involontari, non abusi. In contrapposizione il Native Client deve essere resistente a tentati abusi intenzionali, compito che assolve grazie ai suo meccanismi per un disassembly x86 affidabile e per il controllo dell'integrità del flusso di esecuzione. Nooks non ha meccanismi analoghi.

Trust With Authentication

Per trust with authentication si intende la necessità per il programma di avere il consenso dell'utente per eseguire codice nativo di terze parti.

L'esempio più comune di questo tipo nel contenuto web è Microsoft ActiveX [23]. ActiveX è un estensione (utilizzabile in sistemi operativi Windows) che, integrata in un'applicazione predisposta all'utilizzo di questa tecnologia, permette di aggiungere nuove funzionalità, comandi ed eventualmente semplificare alcuni processi. I controlli ActiveX si basano su un modello affidabile per fornire sicurezza, con controlli firmati crittograficamente usando il Microsoft proprietary Authenticode system, e permettendo solo l'esecuzione una volta che l'utente ha indicato la propria fiducia nell'autore. In Internet Explorer possono essere eseguiti controlli ActiveX che si trovano all'interno di pagine web. Una volta caricati, essi hanno accesso completo al sistema operativo e all'applicazione stessa. Il problema principale di questa soluzione è la sicurezza, infatti, le misure preventive introdotte dalla Microsoft non sono efficaci. Anche se l'installazione di un controllo ActiveX non è affidabile e potrebbe portare all'installazione di software maligno, è l'utente a dover effettuare la scelta finale e nonostante gli avvertimenti potrebbe essere raggirato da contenuto malevolo. Infatti, questa dipendenza dall'utente che prende decisioni indicando la propria fiducia è comunemente sfruttata.

Gli ActiveX non forniscono garanzie che un controllo affidabile sia sicuro, e anche quando il controllo in sé non è intrinsecamente doloso, i suoi difetti possono essere sfruttati, spesso permettendo l'esecuzione di codice arbitrario. Per attenuare questi problemi, Microsoft mantiene una blacklist di controlli considerati non sicuri. In contrapposizione, il Native Client è progettato per prevenire questi exploit, anche per moduli NaCl difettosi.

CAPITOLO 2

Funzionamento e Performance

Funzionamento del Native Client

Il Native Client [34] è un sistema che esegue, in modo sicuro, un modulo di codice nativo su un dispositivo di computazione. Durante le operazioni il sistema riceve il modulo di codice nativo, che è composto da codice untrusted, e lo carica in un ambiente di esecuzione sicuro per poi eseguirlo. L'ambiente di esecuzione garantisce l'integrità del codice, il controllo dell'integrità del flusso di esecuzione, e integrità dei dati per il modulo di codice nativo. Inoltre decide quali risorse possono essere utilizzate dal modulo sul dispositivo di computazione e/o come queste risorse possono essere accedute. Eseguendo il modulo di codice nativo nell'ambiente di esecuzione sicuro, i servizi del sistema raggiungono le performance del codice nativo senza rischio di side effect indesiderati.

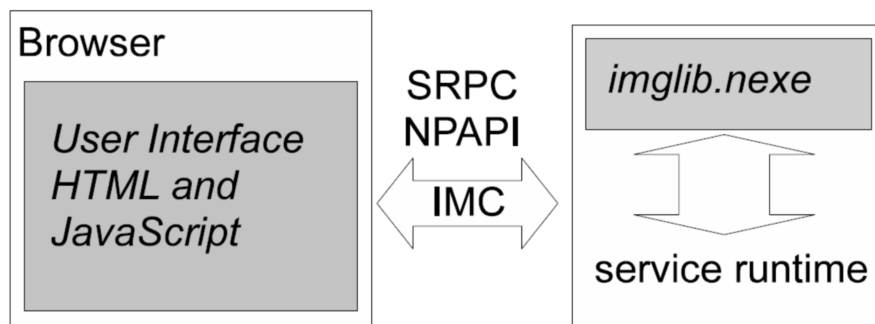

Figura 1: Ipotetica applicazione basata sul Native Client per la modifica e la condivisione di foto. Su sfondo grigio i componenti untrusted.

È possibile descrivere il funzionamento di un' ipotetica applicazione basata sul NaCl per la gestione e la condivisione di fotografie (Figura 1) per capirne meglio il funzionamento.
Essa si divide principalmente in due componenti: un' interfaccia utente, implementata in JavaScript ed eseguita nel browser web, e una libreria di gestione delle immagini (imglib.nexe), implementata come modulo NaCl.
Quest'ultimo sarà incluso nella pagina web tramite JavaScript e HTML usando il tag “embed” con il MIME type “application/x-nacl-srpc” [7]. In questo ipotetico scenario, l’interfaccia utente e la libreria di processione delle immagini sono parte dell’applicazione e quindi untrusted. Il componente browser è vincolato dal suo ambiente di esecuzione e la libreria di gestione delle immagini è vincolata dal contenitore NaCl. Entrambi i componenti sono portabili sui diversi sistemi operativi e browser, con la portabilità del codice nativo garantita dal Native Client.

Prima di eseguire l’applicazione infatti, l’utente ha installato il Native Client come plugin del browser che è specifico per sistema operativo e per browser ed ha pieno accesso all’interfaccia delle system call del sistema operativo.
Quando un utente naviga nel sito web che ospita l’applicazione, il browser carica ed esegue il componente dell’applicazione JavaScript. JavaScript in risposta invoca il plugin NaCl del browser per caricare la libreria di gestione delle immagini in un contenitore NaCl.
Il modulo di codice nativo è caricato “silenziosamente”, nessuna finestra di pop-up viene visualizzata per chiedere il permesso. Il Native Client è responsabile per vincolare il comportamento del modulo untrusted. Infatti, per garantire sicurezza, l'eseguibile vero e proprio (il modulo .nexe) è vincolato dalle due sandbox, quella interna, che verifica che l'eseguibile non contenga codice dannoso, e quella esterna, che verifica i comandi di sistema in esecuzione.
Ogni componente viene eseguito nel proprio spazio di indirizzi privato.
La comunicazione fra componenti interni è basata sull’affidabile datagram service del Native Client, l’IMC (Inter-Module Communications) [1].
Per le comunicazioni tra il browser e il modulo NaCl, il Native Client fornisce due opzioni: Simple RPC (SRPC), e il Netscape Plugin Application Programming Interface (NPAPI).
Il modulo NaCl inoltre ha accesso ad un’interfaccia chiamata “service runtime” [1], che fornisce operazioni di gestione della memoria, creazione di thread, e altri servizi di sistema. Questa interfaccia è analoga a quella delle system call di un sistema operativo convenzionale.
Una volta che il modulo è stato caricato in memoria ed inizializzato, è pronto per rispondere alle chiamate di JavaScript, che, potendo comunicare tramite IMC con il modulo NaCl, ne richiama le funzioni.

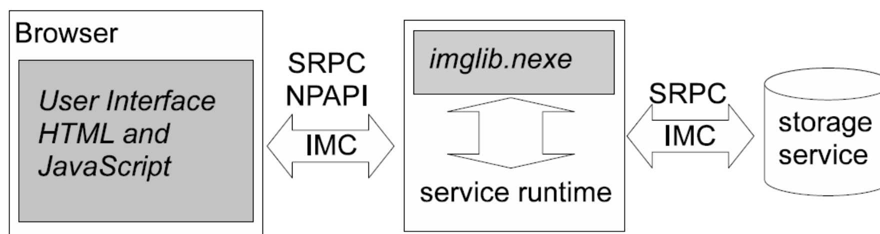

Figura 2: L'ipotetica applicazione per la gestione delle fotografie con un servizio affidabile per la memorizzazione locale (storage service).

Le applicazioni possono usare più moduli NaCl e entrambi i componenti, trusted e untrusted, possono usare l'IMC. Ad esempio, l'utente che utilizza l'applicazione delle fotografie potrebbe opzionalmente essere in grado di usare un (ipotetico) servizio NaCl affidabile per la memorizzazione locale delle immagini (Figura 2).
Poiché ha accesso al disco locale, il servizio di memorizzazione deve essere installato come plugin nativo del browser; non può essere implementato come modulo NaCl.
Supponendo che l'applicazione delle fotografie sia stata progettata per usare opzionalmente il servizio di memorizzazione, l'interfaccia utente proverà a controllare il plugin di quest'ultimo durante l'inizializzazione. Se il plugin è rilevato, l'interfaccia utente proverà a stabilire un canale di comunicazione IMC con esso e a passare un descrittore alla libreria di immagini per abilitare la comunicazione tra la libreria di immagini e il servizio di memorizzazione direttamente tramite servizi basati su ICM (SRPC, memoria condivisa, ecc.).
In questo caso il modulo NaCl dovrà essere "linkato" staticamente ad una libreria che fornisce un interfaccia procedurale per accedere al servizio di memorizzazione. In questo modo si nascondono i dettagli della comunicazione a livello IMC in quanto non si è in grado di stabilire se ad esempio venga utilizzato SRPC o memoria condivisa.
Notare che il servizio di memorizzazione deve supporre che la libreria di immagini sia untrusted, quindi esso è responsabile per assicurare che solo le richieste coerenti con il contratto implicito denotato con l'utente siano soddisfatte.
Ad esempio, dovrebbe imporre un limite sul totale del disco usato dall'applicazione delle fotografie e dovrebbe in seguito limitare le operazioni riferendosi solo ad una specifica directory.
Il Native Client è ideale per componenti di un applicazione che richiedono pura computazione.
Non è appropriato per moduli che richiedono la creazione di processi, accesso diretto al file system, o accesso non vincolato alla rete.

Servizi affidabili come la memorizzazione di dati dovrebbero generalmente essere implementate all'esterno del Native Client, incoraggiando la semplicità e la robustezza dei componenti individuali e imponendo severamente l'isolamento e un esame minuzioso di tutti i componenti.
Queste scelte di progettazione rimandano alla progettazione del micro-kernel [5] del sistema operativo.

Integrazione con Chrome

Una prima versione dell'integrazione del Native Client in Goggle Chrome [6] è attualmente utilizzata per testare moduli programmati tramite una versione provvisoria dell'SDK del NaCl [7]. Per questo motivo sono noti aspetti del funzionamento più specifici (Figura 3):

– Invece di essere eseguito come processo separato come i normali plugin di Chrome, il Native Client viene eseguito all'interno del renderer process. In questo modo si elimina la necessità di una comunicazione IMC tra il renderer process e il plugin NaCl. Inoltre il suo codice viene eseguito in una sandbox mentre solitamente per gli altri plugin ciò non viene fatto
– Il plugin NaCl si affida a PPAPI (che sarà discusso nel prossimo paragrafo) per funzionalità audio e video. I moduli NaCl hanno quindi accesso alla interfaccia PPAPI (oltre a quella SRPC).
– I moduli NaCl sono caricati ed eseguiti in processi separati. La separazione è necessaria per l'isolazione degli errori, un' eccezione causata da codice untrusted su piattaforma x86 causa la terminazione del processo. Inoltre i processi NaCl vengono eseguiti nella sandbox di Chrome, aggiungendo un ulteriore strato di sicurezza, la "sandbox esterna"
– Come ogni altro processo di Chrome, il processo di caricamento del NaCl (chiamato sel_ldr) è eseguito dal processo del browser e stabilisce un canale di comunicazione IPC (inter-process communications) con esso durante l'inizializzazione. Mente il processo NaCl comunica con il NaCl plugin (eseguito nel renderer process) usando l'IMC.

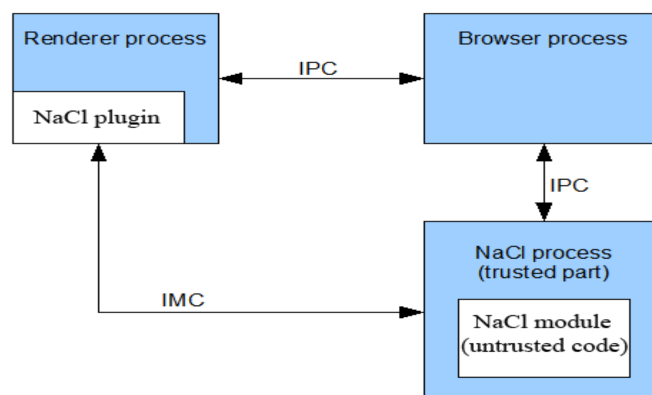

Figura 3: L'integrazione del Native Client in Google Chrome.

Comunicazioni

Per le comunicazioni tra processi, il Native Client fornisce un affidabile datagram service, l' “Inter-Module Communications” o IMC. Il datagram service [5] è un servizio fornito dall'Internet Protocol (IP) [5] per la distribuzione di messaggi ed è alla base dei protocolli TCP [31] e UDP [32]. L'IMC permette a moduli trusted e non, di inviare o ricevere datagrammi costituiti da array di byte non tipizzati con opzionalmente un “NaCl Resource Descriptors” per facilitare la condivisione di file, oggetti sulla memoria condivisa, canali di comunicazione, ecc.
L'IMC è la base per due astrazioni ad alto livello.
La prima, il servizio Simple Remote Procedure Call (SRPC) [1], fornisce una sintassi adatta per definire e utilizzare subroutine all'esterno del modulo NaCl, comprese le chiamate al codice da JavaScript nel browser.
La seconda, Netscape Plugin Application Programming Interface (NPAPI) fornisce un' interfaccia familiare per interagire con lo stato del browser, compresa la capacità di aprire URL e accedere al DOM (Document Object Model), che è conforme ai vincoli esistenti per la sicurezza dei contenuti.
Entrambi i meccanismi possono essere utilizzati per interazioni generali con il contenuto convenzionale del browser, compresa la modifica dei contenuti, la gestione dell'attività di mouse e tastiera, e l'accesso a contenuti addizionali di un sito: sostanzialmente tutte le risorse comunemente disponibili a JavaScript.
L’IMC inoltre fornisce segmenti di memoria condivisa e sincronizzazione di oggetti condivisa che vengono utilizzati per evitare overhead di messaggi in comunicazioni ad alta capacità (high-volume) o ad alta frequenza (high-frequency).
Il service runtime è responsabile per fornire il contenitore attraverso il quale i moduli NaCl interagiscono gli uni con gli altri nel browser.

Esso fornisce un insieme di servizi di sistema comunemente associati all'ambiente di sviluppo di un applicazione. Mette a disposizione le chiamate di sistema [5] sysbrk() (imposta il break di sistema ad un determinato indirizzo e ritorna l'indirizzo dopo l'aggiornamento) e mmap() (associa ad un file o ad un dispositivo uno spazio di dimensione finita in memoria), primitive per supportare interfacce malloc()/free() (allocazione dinamica della memoria e liberazione della memoria utilizzata) o altre astrazioni per l'allocazione di memoria.

Inoltre il service runtime fornisce un sottoinsieme dell'interfaccia thread POSIX, con alcune estensioni del NaCl, per la creazione e la distruzione di thread, variabili di condizione, mutex, semafori, e la memorizzazione locale di thread. POSIX [5] è il nome che indica la famiglia degli standard definiti dall'IEEE [33] denominati formalmente IEEE 1003. Esso specifica l'interfaccia comune del sistema operativo (Application Programming Interface, API) all'utente ed al software.

Tale servizio mette anche a disposizione l'interfaccia I/O comune dei file POSIX, utilizzati per operazioni sui canali di comunicazione così come contenuti web-based di sola lettura. Visto che lo spazio dei nomi del file system locale non è accessibile da queste interfacce, i side effect non sono possibili.

IMC

L'IMC è la base delle comunicazioni all'interno e all'esterno dei moduli del NaCl. La sua implementazione è costruita attorno ad un *NaCl socket*, fornendo un bi-direzionale e affidabile datagram service simile agli “Unix domain socket” [5]. Un Unix domain socket è un punto per lo scambio di dati tra processi eseguiti all'interno dello stesso sistema operativo host.
Un modulo NaCl untrusted ottiene il suo primo socket NaCl alla creazione, accessibile da JavaScript tramite l'oggetto di tipo Document-Object Model usato per crearlo.
JavaScript utilizza il socket per inviare messaggi al modulo NaCl, e può inoltre condividerlo con altri moduli.

JavaScript può anche scegliere di connettere il modulo ad altri servizi ad esso disponibili aprendo e condividendo socket NaCl come descrittori. I descrittori NaCl possono essere anche usati per creare segmenti di memoria condivisa.
Utilizzando messaggi NaCl, l'astrazione SRPC del Native Client è implementata interamente in codice untrusted.
L'SRPC fornisce una sintassi conveniente per dichiarare interfacce procedurali tra JavaScript e i moduli NaCl, o tra due moduli NaCl, supportando alcuni tipi di dati base (int, float, char) oltre agli array per i descrittori NaCl. Tipi di dati più complessi e i puntatori non sono supportati. Strategie esterne per la rappresentazione dei dati possono essere facilmente stratificate all'inizio dei messaggi NaCl o SRPC.
L'implementazione NPAPI del Native Client supporta un sottoinsieme delle interfacce comuni NPAPI. Gli specifici requisiti che hanno modellato l'implementazione corrente sono le abilità di leggere, modificare e richiamare proprietà e metodi degli oggetti scriptati nel browser, il supporto per semplice grafica raster, il metodo createArray() e la capacità di aprire e utilizzare un URL come un descrittore di file (file descriptor). Il file descriptor [5] è un numero intero non negativo che rappresenta un file o un socket aperto da un processo e sul quale il processo può effettuare operazioni di input/output. Tale numero è un indice in un array che fa parte del PCB. Il Process Control Block [5] è la struttura dati che contiene le informazioni essenziali per la gestione di un processo. L'implementazione del sottoinsieme NPAPI è stata scelta dagli sviluppatori del NaCl principalmente per convenienza, infatti, è nelle loro intenzioni un miglioramento per vincolarla maggiormente ed estenderla. Necessità per cui si è arrivati ad utilizzare l'API Pepper per i plugin(PPAPI o "Pepper2")[30].

PPAPI

PPAPI è un API, utilizzabile attraverso i diversi sistemi operativi, per plugin dei browser. Esso viene utilizzato in modo sperimentale da Google Chrome come alternativa a NPAPI, per risolvere problemi di portabilità e performance riscontrati con quest'ultimo. In futuro i plugin Pepper saranno supportati solo all'interno del Native Client.

Sono stati discussi alcuni dei problemi del NPAPI come framework per plugin del browser indipendenti dalla piattaforma. Per prima cosa, vengono ritenuti necessari alcuni cambiamenti al NPAPI per renderlo adatto ai recenti sviluppi nella tecnologia dei browser, come l'esecuzione dei plugin out of process.
Out of process (fuori dal processo) [30] è usato per indicare la possibilità di eseguire un plugin in un processo diverso da quello del browser, quindi al di fuori del suo processo.
In secondo luogo, mentre NPAPI fornisce un framework per scrivere plugin, molti di essi finiscono con il basarsi sul sistema operativo o sulle caratteristiche specifiche per piattaforma di un browser. Questo specialmente per plugin grafici che implementano grafica 2d o 3d, dove quasi tutto il plugin può consistere in eventi API o grafica specifica per un particolare sistema operativo.
Infine, sia i plugin grafici che non, sono difficili da integrare correttamente con altri strati, rendendo difficile per un autore di pagine web raggiungere lo stesso aspetto e funzionamento attraverso browser e sistemi operativi differenti.
Gli obiettivi del PPAPI sono quattro:

1 – Dare una chiara semantica al NPAPI attraverso i browser, che implementi i plugin in un processo separato da quello del browser.

2 – Integrare il rendering dei plugin con la composizione del processo del browser, permettendo overlay HTML (un modo per includere qualsiasi tipo di documento in una pagina web), ecc., per lavorare correttamente con i plugin.

3 – Definire eventi, rasterizzazione 2d [5], e un passo iniziale per l'accesso a grafica 3d, in un modo comune attraverso sistemi operativi e browser.

4 – Determinare quali plugin sono disponibili per un browser senza caricare il plugin.

Un problema in qualche modo collegato, ma che per il momento è stato accantonato per il futuro, è come estendere NPAPI per fornire accessi indipendenti dalla piattaforma alle nuove caratteristiche hardware o software come fotocamere digitali, memorie SD, lettori di codici a barre, webcam, microfoni, ecc.

Performance

Un obiettivo primario del Native Client è fornire le piene performance dell'esecuzione del codice nativo. Le performance di un modulo NaCl dipendono dai vincoli di allineamento, dalle istruzioni aggiuntive per i trasferimenti indiretti del controllo del flusso di esecuzione e dal costo incrementale delle comunicazioni NaCl.
Si parla di alignment (allineamento) o meglio di memory alignment [36], per indicare la proprietà di un dato in memoria di essere collocato ad un indirizzo multiplo di una certa quantità di bit. Una variabile, ad esempio, può essere allineata a 8, 16 o 32 bit. Questo significa che l'indirizzo di memoria in cui la variabile è collocata è un multiplo di 8,16 o 32 bit.
Il modo più semplice per verificare se una variabile è allineata o meno ad un dato valore è usare il suo indirizzo con l'operatore di modulo (% in C e C++) e il valore di bit desiderato. Se il risultato è zero, vuol dire che l'indirizzo è un multiplo esatto di quel valore e quindi la variabile risulta allineata.
L'allineamento è necessario per le prestazioni: il computer può accedere alla memoria solo per blocchi, e l'allineamento garantisce che le variabili non inizino a metà blocco, evitando al calcolatore laboriose operazioni per il recupero dell'informazione.
Prima di tutto consideriamo l'overhead necessario per rendere il codice nativo esente da side effect. L'overhead delle sandbox dipende dal numero di message-passing e dall'attività di service runtime che l'applicazione richiede. Per isolare l'incidenza dei vincoli binari del NaCl, gli sviluppatori hanno eseguito il build di uno "SPEC2000 CPU" benchmark [1] usando il compilatore del NaCl, e hanno eseguito un "linking" apposito per avviarlo come standard Linux binary.
Con il termine benchmark [5] si intende un insieme di test software volti a determinare la capacità di un software di svolgere più o meno velocemente, precisamente o accuratamente, un particolare compito per cui è stato progettato.

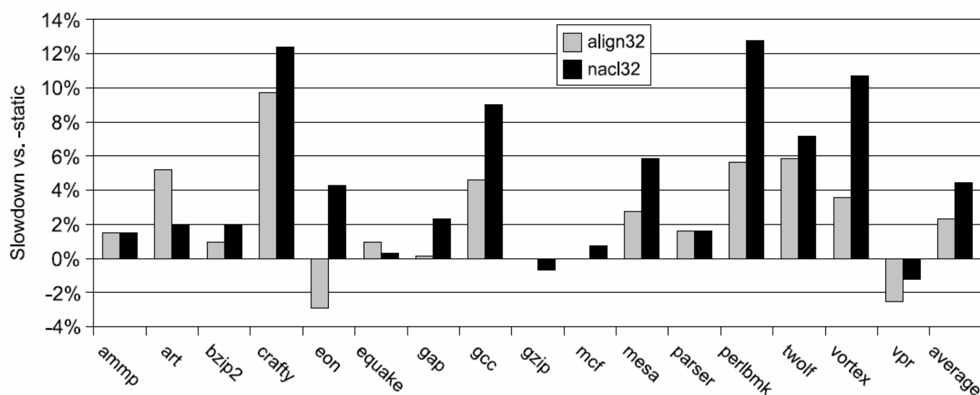

Figura 4: Performance ottenute con il benchmark SPEC2000. I risultati "Static" sono per codice oggetto "linkato" staticamente; quelli "align32" per codice oggetto allineato in blocchi di 32 byte, e quelli "nacl32" per codice oggetto NaCl.

Il peggior caso di tempo di overhead del NaCl si riscontra con applicazioni CPU bound perché esse hanno la maggior densità di alignment e sandboxing overhead.
Si definiscono Cpu-bound [5] i processi che sfruttano pesantemente le risorse computazionali del processore, ma non richiedono operazioni di input/output al sistema operativo in quantità rilevanti (al contrario di IO-bound).

Test	Static	Aligned	NaCl	Incremento
Ammp	200	203	203	1,5%
art	46,3	48,7	47,2	1,9%
bzip2	103	104	104	1,9%
crafty	113	124	127	12%
eon	79,2	76,9	82,6	4,3%
equake	62,3	62,9	62,5	0,3%
gap	63,9	64	65,4	2,4%
gcc	52,3	54,7	57,0	9,0%
gzip	149	149	148	-0,7%
mcf	65,7	65,7	66,2	0,8%
mesa	87,4	89,8	92,5	5,8%
parser	126	128	128	1,6%
perlbmk	94	99,3	106	13%
twolf	154	163	165	7,1%
vortex	112	116	124	11%
vpr	90,7	88,4	89,6	-1,2%

Tabella 1: Performance ottenute con il benchmark SPEC2000. Il tempo di esecuzione è in secondi. I codici oggetto dei vari test sono tutti stati "linkati" staticamente.

La figura 4 e la tabella 1 mostrano l'overhead della compilazione NaCl di un insieme di benchmark SPEC2000. Le misurazioni sono fatte senza la sandbox esterna del NaCl. Il peggior caso di performance di overhead si verifica con il tool crafty, ed è di circa 12%, con una media globale per gli altri benchmark del 5%. I contatori per le misurazioni delle performance hardware indicano che i maggiori rallentamenti sono dovuti a istruzioni di cache mancanti. Per il tool crafty, l'unità di fetch dell'istruzione è andata in fase di stallo durante l'83% dei cicli per il build del NaCl, comparata al 49% per la build di default. Anche su gcc e vortex hanno un' incidenza significativa le istruzioni di cache mancanti.
Visto che l'implementazione dell'alignment è di tipo standard, gli sviluppatori sperano di effettuare miglioramenti incrementali alla dimensione del codice con il perfezionamento dell'implementazione.
Le misurazioni "NaCl" si riferiscono a codice oggetto (binaries) linkato staticamente, allineato a blocchi di 32 byte, e che utilizza l'istruzione nacljmp per i trasferimenti indiretti del controllo del flusso di esecuzione. Per isolare l'incidenza di questi tre vincoli, la figura 4 mostra le performance per il linking statico e per il linking statico e l'alignment. Questi confronti mettono in chiaro che l'alignment è il fattore principale in casi in cui l'overhead è significativo.

Per confronto, l'incidenza dell'overhead del linking statico e delle istruzioni di sandboxing è poca.
L'incidenza dell'aligment non è costante in tutta la suite di benchmark. In alcuni casi, l'alignment sembra migliorare le performance, e in altri sembra peggiorare la situazione. In casi in cui l'alignment rende le performance peggiori, un possibile fattore è la dimensione del codice. La tabella 2 mostra che l'incremento della dimensione del codice NaCl a causa dell'alignment può essere significativa, specialmente in benchmark come gcc con un gran numero di chiamate a funzioni statiche. Allo stesso modo, i benchmark con un gran numero di trasferimenti di controllo (es. crafty, vortex) hanno una dimensione del codice maggiore, aumentata a causa dell'alignment necessario sui target dei salti (jump). L'incremento della dimensione del codice del sandboxing con l'istruzione nacljmp è esiguo. Globalmente, l'incidenza delle performance del Native Client su questi benchmark è su una media inferiore al 5%. A questo livello, il tempo di overhead si può paragonare in modo favorevole all'esecuzione di codice nativo untrusted.

Test	Static	Aligned	NaCl	Incremento
Ammp	657	759	766	16,7%
art	469	485	485	3,3%
bzip2	492	525	526	7%
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	614	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%

Tabella 2: Dimensione del codice con il benchmark SPEC2000, le misurazioni sono in kilobytes.

Test delle performance computazionali/grafiche

Sono stati implementati dagli sviluppatori del Native Client tre semplici benchmark composti da computazione+animazione per testare e valutare le performance della CPU per il codice a thread.

I tre benchmark sono:

– Earth: un carico di lavoro su un tracciato a semiretta, proiettando un immagine piatta della Terra su un globo rotante.
– Voronoi: tassellatura di Voronoi. In matematica, un diagramma di Voronoi [5], è un particolare tipo di decomposizione, in uno spazio metrico, determinata dalle distanze rispetto ad uno specifico insieme discreto di elementi dello spazio (ad esempio un insieme finito di punti).
– Life: simulazione di automi cellulari del Conway's Game of Life.

Il "Conway's Game of Life" [5] è l'esempio più famoso di automa cellulare: il suo scopo è quello di mostrare come comportamenti simili alla vita possano emergere da regole semplici e interazioni tra molti corpi.

Questi carichi di lavoro hanno aiutato gli sviluppatori del NaCl a perfezionare e valutare l'implementazione dei thread e a fornire un benchmark di confronto per la compilazione nativa standard.
Gli sviluppatori del NaCl hanno usato usato il comando Linux *time* per avviare e cronometrare le release di eseguibili standalone (un programma indipendente) al confronto con quelle NaCl. Le misurazioni sono state effettuate con un sistema Ubuntu Dapper Drake Linux con un processore quad core Intel Q6600 a 2.4GHz con VSYNC (video system's vertical sync) disabilitato. È importante disabilitarlo durante il benchmark di un applicazione di questo tipo perchè il thread di rendering potrebbe essere messo in coda fino al vertical sync successivo sullo schermo. Gli eseguibili normali sono stati "buildati" usando g++ 4.0.3 [19], la versione NaCl con nacl-g++ 4.2.2 [7].
Voronoi e Earth utilizzano 4 thread di lavoro e vengono eseguiti a 1000 frame. Life utilizza un solo thread, per 5000 frame.
La tabella 3 mostra la media delle tre esecuzioni consecutive.
Voronoi viene eseguito più velocemente come applicazione NaCl rispetto ad un normale eseguibile. Gli altri due test, al contrario, vengono eseguiti più velocemente come normali eseguibili rispetto alle loro controparti del Native Client.

Esempio	Native Client	Eseguibile Linux
Voronoi	12,4	13,9
Earth	14,4	12,6
Life	21,9	19,4

Tabella 3: Test delle performance computazionali/grafiche. Le misurazioni sono in secondi.

Globalmente queste misure preliminari indicano che, per questi semplici esempi di test, l'implementazione a thread del NaCl può essere comparata in modo ragionevole a Linux. La tabella 4 mostra una comparazione delle performance sui thread tra il Native Client e un normale eseguibile Linux, usando la demo di Voronoi. Confrontando il Native Client a Linux, le performance crescono con l'aumentare del numero di thread.

Eseguibile	1 thread	2 thread	4 thread
Native Client	42,16	22,04	12,4
Linux Binary	46,29	24,53	13,9

Tabella 4: Performance dei thread con Voronoi. Le misurazioni sono in secondi.

Bullet

Bullet [37] è un sistema open-source per la simulazione fisica. Esso ha caratteristiche accurate e di modellazione che lo rendono appropriato per applicazioni real-time come i giochi per computer. È la rappresentazione di un tipo di sistema che gli autori del NaCl vorrebbero supportare.
Per i test è stato usato Bullet 2.66 che è configurabili tramite autotools, permettendo un uso specifico del compilatore NaCl. Alcune #define sono state adattate per eliminare system call profiling non supportate e altro codice specifico di SO. Il profiling [5] è l'analisi del funzionamento di un programma durante la sua esecuzione, con l'obiettivo di determinare quali parti di questo programma debbano essere ottimizzate incrementando la sua velocità e/o riducendo i suoi requisiti relativi alla memoria.
Globalmente occorrono un paio d'ore di sforzi per ottenere la libreria per il Native Client.
I test per le performance utilizzano il programma dimostrativo “HelloWorld” preso dalla distribuzione sorgente del Bullet, una simulazione di una gran quantità di sfere che cadono e collidono su una superficie piana.

Sono state confrontate due build utilizzando GCC 4.2.2 che è in grado di generare codice oggetto NaCl. Misurando 100000 iterazioni, sono stati rilevati 36,5 secondi per il build di base (-static) e 36,1 secondi per quello con allineamento in blocchi di 32-byte (come richiesto dal Native Client), circa l'1% di incremento della velocità. Includendo il codice operativo addizionale per i vincoli richiesti dal Native Client il risultato a tempo di esecuzione è di 37,3 secondi, circa un rallentamento globale del 2%. Queste misure sono state prese usando un dual-core Opteron 8214 a due processori con 8 GB di memoria.

Quake

È stato effettuato il profiling di sdlquake-1.0.9 [35]. Quake è stato eseguito ad una risoluzione di 640X480 su un Ubuntu Dapper Drake Linux box con una CPU INTEL Q6600 quad core a 2,4GHZ. VSYNC disabilitato. E' stato effettuato il build dell'eseguibile Linux utilizzando gcc 4.0.3, e il Native Client con nacl-gcc 4.2.2, entrambi con l'ottimizzazione *-O2*.
Con Quake, le differenze tra l'eseguibile normale e quello del Native Client sono, per scopi pratici, indistinguibili (vedere la tabella 5 per il confronto). Il test rappresenta la stessa sequenza di eventi indipendentemente dal frame-rate. Una lieve oscillazione del frame rate può continuare a verificarsi a causa dello scheduler dei thread del sistema operativo e della pressione effettuata alle cache di memoria condivise da parte di altri processi. Nonostante Quake utilizzi software di rendering, le performance del trasferimento finale di immagini bitmap al desktop dell'utente potrebbe dipendere dal carico di lavoro del dispositivo video. Il Rendering [5] è un termine usato nell'ambito della computer grafica; identifica il processo di "resa" ovvero di generazione di immagine a partire da una descrizione matematica di una scena tridimensionale interpretata da algoritmi che definiscono il colore di ogni punto dell'immagine.

N. esecuzione	Native Client	Eseguibile Linux
1	143,2	142,9
2	143,6	143,4
3	144,2	143,5
Media	143,7	143,3

Tabella 5: Comparazione delle performance di Quake. Le misurazioni rappresentano il numero di frame al secondo.

CAPITOLO 3

Utilizzo e Valutazione di Google Native Client

Strumenti per l'utilizzo del Native Client

Come già detto in precedenza, uno degli obiettivi che il Native Client si prefigge è raggiungere la portabilità: la possibilità per qualsiasi utente di utilizzare questa tecnologia indipendentemente dal sistema operativo, browser o architettura che utilizza. Lo strumento necessario per utilizzare il Native Client dovrà essere solo uno: un plugin specifico per sistema operativo e browser che una volta installato gestirà automaticamente le applicazioni NaCl contenute nelle pagine web. In questo modo l'obiettivo sopracitato sarà raggiunto e l'utente non avrà bisogno di ulteriori programmi per l'uso del NaCl.

Il Native Client però è ancora in fase di sviluppo, ogni giorno vengono documentati e risolti vari problemi e un po' alla volta gli sviluppatori stanno procedendo con l'aggiornamento e l'inserimento di documentazioni e informazioni per mettere a disposizione della comunità di internet nuovi esempi e la possibilità di testare e valutare la tecnologia.

Per questi motivi non è ancora pronto un plugin definitivo che si possa occupare dei moduli NaCl garantendo la portabilità. Inizialmente era stata rilasciata una prima versione del plugin installabile su alcuni dei browser più comuni, ma a causa dei numerosi problemi riscontrati è stata rimpiazzata dall'integrazione del NaCl in Google Chrome.

In questo momento l'unico modo per utilizzare il Native Client ed eseguire degli esempi è l'utilizzo del browser di Google che a partire dalla versione 6.0.472.53 ne incorpora direttamente una particolare implementazione. Il plugin NaCl non viene attivato per default, è necessario avviare Chrome con un flag specifico: *--enable-nacl*. A questo punto è già possibile avviare degli esempi online forniti da Google: è sufficiente visitare la pagina web che li ospita.

Ricapitolando per provare il Native Client bastano tre semplici operazioni:

- Installare Google Chrome (versione almeno 6.0.472.53)
- Avviare Google Chrome con il flag --enable-nacl
- Visitare una pagina web contente un modulo NaCl.

Native Client SDK

Oltre alla possibilità di eseguire esempi online, Google mette a disposizione un SDK [7] per effettuare il build ed eseguire moduli NaCl in locale.

Software Development Kit [5] (SDK) è un termine che in italiano si può tradurre come "pacchetto di sviluppo per applicazioni", e sta ad indicare un insieme di strumenti per lo sviluppo e la documentazione di software. Gli SDK possono variare considerevolmente in quanto a dimensioni e tecnologie utilizzate, ma tutti possiedono alcuni strumenti fondamentali:

- un compilatore, per tradurre il codice sorgente in un eseguibile
- librerie standard dotate di interfacce pubbliche dette API
- documentazione sul linguaggio di programmazione per il quale l'SDK è stato sviluppato e sugli strumenti a disposizione nell'SDK stesso
- informazioni sulle licenze da utilizzare per distribuire programmi creati con l'SDK

Questo corredo di base può essere esteso con strumenti di vario tipo.

Per gli sviluppatori di Google l'SDK del Native Client dovrà avere una serie di caratteristiche e funzionalità:

- Strumenti per rendere il Native Client e i port per il Native Client più facili da usare per gli sviluppatori
- Librerie per aiutare a rendere l'integrazione tra codice nativo compilato dal Native Client e il codice del browser più semplice possibile
- Supporto per ambienti di sviluppo come Eclipse [43] e Microsoft Visual Studio [42]
- Documentazione: esempi di come incorporare codice binario per una varietà di applicazioni, tutorial su come far funzionare un semplice sito NaCl, documentazione per codice basato sul web per ogni libreria disponibile che uno sviluppatore potrebbe utilizzare per programmare moduli NaCl.

Inoltre si prefiggono anche una serie di obiettivi per l'SDK:

- Rendere lo sviluppatore di applicazioni NaCl indipendente, liberandolo dalla necessità di seguire il canale degli sviluppatori di Chrome
- Fornire un' alternativa alle OpenGl [40] per il 3d
- Fornire uno strumento per il debugging a linea di comando
- Fornire un' interfaccia comune per il debugging che superi i limiti tra JavaScript e il codice NaCl. In pratica, la possibilità di utilizzare qualcosa di simile alla console per sviluppatori di Google Chrome per effettuare il debug dei moduli NaCl all'interno del browser.

Per effettuare il build di applicazioni NaCl, oltre all'SDK scaricabile (come archivio auto-estraente) dal sito del Native Client [7], è necessario anche installare Python [16] (dalla versione 2.4 in poi). Una volta eseguite queste operazioni è sufficiente estrarre i dati dell'SDK per poter eseguire alcuni esempi in locale (forniti con l'SDK). Come per gli esempi online occorre avviare Google Chrome con il flag --enable-nacl ma in questo caso bisogna anche lanciare un server web che possa servire i contenuti della directory degli esempi. A questo scopo un semplice server HTTP basato su Python è già disponibile con l'SDK e si può attivare tramite il comando Python httpd.py 5103 per metterlo in ascolto sulla porta 5103. In questo modo si possono eseguire gli esempi locali con un indirizzo del tipo http://localhost:5103/nome_esempio/nome_esempio.html.

Per effettuare il build degli esempi, sono disponibili vari *makefile* (uno per ogni esempio) che seguendo le regole dell'SDK genera i file oggetto e i file eseguibili .nexe. Ogni file creato è in duplice versione, una per sistemi operativi a 32 bit e una per quelli a 64bit. Questi file (.nexe) rappresentano il modulo NaCl vero e proprio che viene incluso dalla pagina web apposita.

Il make [5] è un utility, sviluppata sui sistemi operativi della famiglia UNIX ma disponibile su un'ampia gamma di sistemi, che automatizza il processo di creazione di file che dipendono da altri file, risolvendo le dipendenze e invocando programmi esterni per il lavoro necessario.

L'utility è usata soprattutto per la compilazione di codice sorgente in codice oggetto, unendo e poi linkando il codice oggetto in programmi eseguibili o in librerie. Il make usa file chiamati makefile per determinare il grafo delle dipendenze per un particolare output, e gli script necessari per la compilazione da passare al terminale (shell).

```
# Copyright 2010, The Native Client SDK Authors. All rights reserved.
# Use of this source code is governed by a BSD-style license that can
# be found in the LICENSE file.

# Makefile for the Hello World example.

.PHONY: all clean

CCFILES = hello_world_module.cc hello_world.cc npp_gate.cc

OBJECTS_X86_32 = $(CCFILES:%.cc=%_x86_32.o)
OBJECTS_X86_64 = $(CCFILES:%.cc=%_x86_64.o)

NACL_SDK_ROOT = ../..

CFLAGS = -Wall -Wno-long-long -pthread -DXP_UNIX -Werror
INCLUDES =
LDFLAGS = -lgoogle_nacl_imc \
          -lgoogle_nacl_npruntime \
          -lpthread \
          -lsrpc \
          $(ARCH_FLAGS)
OPT_FLAGS = -O2

all: check_variables hello_world_x86_32.nexe hello_world_x86_64.nexe

# common.mk has rules to build .o files from .cc files.
-include ../common.mk

hello_world_x86_32.nexe: $(OBJECTS_X86_32)
	$(CPP) $^ $(LDFLAGS) -m32 -o $@

hello_world_x86_64.nexe: $(OBJECTS_X86_64)
	$(CPP) $^ $(LDFLAGS) -m64 -o $@

clean:
	-$(RM) $(OBJECTS_X86_32) $(OBJECTS_X86_64) \
	    hello_world_x86_32.nexe hello_world_x86_64.nexe
```

Figura 5: Un makefile utilizzato per effettuare il build degli esempi del Native Client. In questo caso si tratta di quello dell'esempio "Hello World".

Come la versione del Native Client usata per eseguire i moduli anche la versione dell'SDK è in una fase preliminare. È intenzione degli sviluppatori semplificarlo e migliorarlo di pari passo con i progressi del Native Client.

Esempi

Gli esempi forniti con l'SDK del Native Client sono strutturati in un modo particolare (figura 6). Ogni esempio è in una cartella specifica ed è costituito da un insieme di file che si possono dividere in tre categorie:

– **JavaScript application**: Fornisce l'interfaccia utente e il meccanismo di gestione degli eventi, così come la pagina HTML; esegue alcuni calcoli

– **JavaScript bridge**: Invia le chiamate alle funzioni da parte di

JavaScript al modulo del Native Client per l'elaborazione; decomprime gli argomenti delle funzioni JavaScript e li re-comprime come argomenti per i metodi del C++ per essere gestiti dal NaCl

– **Native Client module:** Esegue i calcoli numerici dell'applicazione e fornisce il motore di rendering grafico. Regola il rendering nell'area di visualizzazione che viene impostata nel documento HTML.

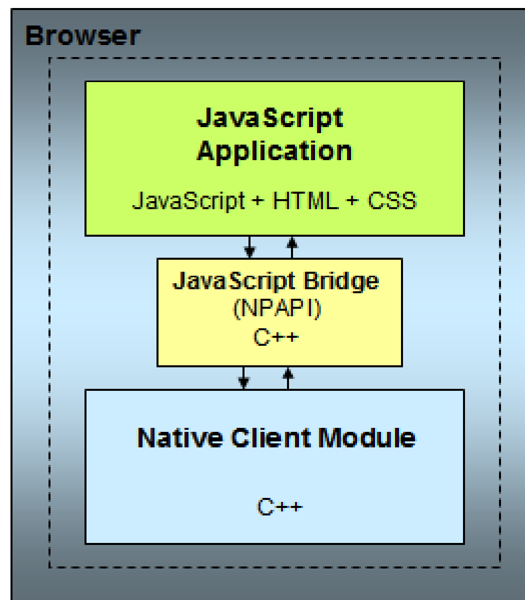

Figura 6: Struttura degli esempi dell'SDK del Native Client.

I file che fanno parte della categoria "JavaScript Application" sono:

– la pagina HTML che include il modulo NaCl

– eventuali file JavaScript(.js) o fogli di stile(.css).

Quelli che fanno parte della categoria "JavaScript Bridge" sono:

– scripting bridge.cc (e scripting bridge.h): Chiamano i metodi C++ nel modulo del Native Client che corrispondono alle funzioni invocate nella "Javascript Application"; decomprime le funzioni e gli argomenti e li re-comprime per l'utilizzo da parte dei metodi C++ del Native Client corrispondenti

– npp_gate.cc: Funzioni NPAPI che danno la possibilità al browser di effettuare chiamate al modulo NaCl

– npn_bridge.cc: Funzioni NPAPI che danno la possibilità al Native Client di effettuare chiamate al browser

– nome_esempio_module.cc: Contenitore del modulo NaCl che è chiamato dal browser per eseguire le procedure di inizializzazione e di chiusura.

Infine, quelli che fanno parte della categoria “Native Client module” sono:

– nome_esempio.cc (e nome_esempio.h): l'applicazione di controllo del modulo NaCl. Imposta il contesto dell'applicazione e l'eventuale contesto grafico.

– Eventuali altri file sorgente .cc e altri file di intestazione .h per funzionalità o applicazioni aggiuntive.

Hello World

Questo è l'esempio più semplice dell'utilizzo del Native Client.

A differenza della struttura generale degli esempi, questa applicazione è così semplice che il modulo NaCl non ha neanche bisogno di interpellare il browser, risponde semplicemente alla sue chiamate. Per questo motivo il Javascript Bridge è composto solo dai file npp_gate.cc e hello_world_module.cc.

Gli elementi rilevanti della pagina HTML di questo esempio sono due pulsanti e uno “status”. Ogni pulsante è associato ad una funzione JavaScript che richiama un metodo del modulo NaCl incluso nella pagina. Il primo pulsante richiede al modulo NaCl il metodo fortytwo() che restituirà al browser il valore numerico 42. Il secondo chiama il metodo helloworld() che restituisce al browser la stringa “hello, world”. I valori restituiti saranno poi visualizzati tramite finestre di popup (funzione alert() di JavaScript). Lo “status” informa l'utente sul successo o meno del caricamento del plugin NaCl.

Figura 7: Esecuzione dell'esempio “Hello World”.

Pi Generator (Pepper 2D and threading)

In questo caso la struttura dei file dell'esempio è come quella generale

descritta in precedenza, senza file aggiuntivi. Questo è il primo esempio con una parte grafica in 2d.

Il modulo NaCl eseguito nella pagina web crea un thread che effettua una stima del pi greco(π) usando il metodo Monte Carlo [5].

Il thread disegna un miliardo di punti in modo random all'interno di un quadrato che condivide due lati con un quarto di cerchio (un quadrante). Visto che l'area del quadrante è $r^2\pi/4$ e l'area del quadrato è r^2, dividendo il numero di punti all'interno del quadrante per il numero di punti all'interno del quadrato si ottiene una stima di $\pi/4$. Sotto alla parte grafica una casella di testo viene aggiornata con la stima corrente di π. La pagina HTML durante il caricamento del modulo NaCl richiama una funzione JavaScript che chiama il metodo paint() del modulo NaCl che si occupa della parte grafica. Il quadrato 2d è generato usando l'API 2D Pepper, che è un estensione di NPAPI. Inoltre il "JavaScript Bridge" una volta gestita la comunicazione tra JavaScript e il modulo NaCl per la gestione della parte grafica richiama un altro metodo che restituisce al browser il valore aggiornato di π.

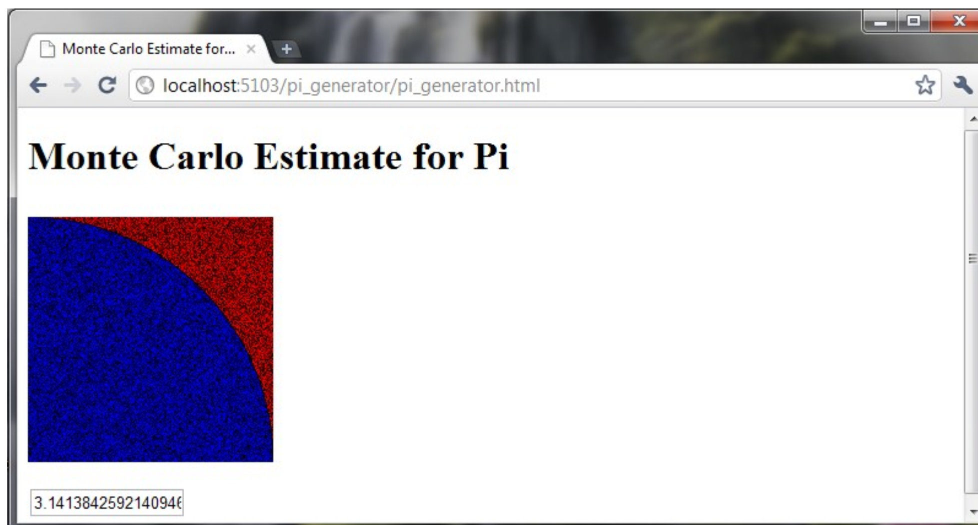

Figura 8: Esecuzione dell'esempio "Pi Generator".

Sine Wave Synthesizer (Pepper Audio)

Questo esempio non presenta differenze rispetto al precedente per quanto riguarda la struttura dei file, ma invece di una parte grafica esegue un contenuto audio. In questo esempio infatti, è possibile inserire una frequenza e tramite due pulsanti "play" e "stop" richiamare una metodo del modulo NaCl per sintetizzare e riprodurre un' onda sonora sinusoidale. Per eseguire il tono viene utilizzata l'API Audio Pepper.

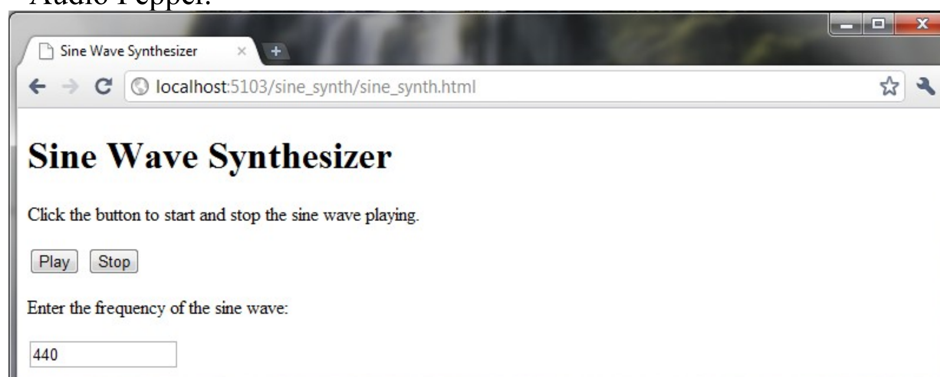

Figura 9: Esecuzione dell'esempio “Sine Wave Synthsizer”.

Tumbler (Pepper 3D)

L'esempio più complicato tra quelli forniti dagli sviluppatori di Google è il Tumbler. Rispetto agli altri esempi esso presenta differenze per quanto riguarda la struttura dei file, infatti sono parecchi quelli che vengono aggiunti.

Per quanto riguarda i file della categoria “JavaScript Application” oltre alla pagina HTML ci sono quattro file .js aggiuntivi:

– tumbler.js: Definisce l'oggetto applicazione che carica il modulo del Native Client, inizializza l'applicazione e la esegue

– dragger.js: Esegue elaborazioni degli eventi standard JavaScript. Gestisce gli eventi generati del mouse e chiama le funzioni appropriate

– trackball.js: Associa gli eventi 2D di trascinamento del mouse ad una rotazione 3D simulando una trackball che ruota con il movimento del mouse

– vector.js: Una semplice libreria vettoriale.

Per la categoria "JavaScript Bridge" non ci sono differenze. Invece per la categoria "Native Client module" ci sono quattro file sorgenti aggiuntivi:

– cube.cc (e cube.h): Crea e disegna il cubo.

– shader_util.cc (e shader_util.h): Semplici funzioni di help che caricano gli shaders. La parola inglese shader [5] indica uno strumento della computer grafica 3D che generalmente è utilizzato per determinare l'aspetto finale della superficie di un oggetto.

– transforms.cc (e transforms.h): Un semplice insieme di routine di matrice 4x4.

– basicmacros.h: Macro C++ di uso generico.

In questo esempio il modulo NaCl eseguito nella pagina web disegna un cubo 3D tramite l'API 3D Pepper e permette all'utente di ruotarlo con il mouse come se stesse usando una trackball.

Per la programmazione di grafica in 3D ci sono più scelte in Google Chrome. Per JavaScript, si possono usare le WebGL graphics library [5]. WebGL fornisce un API per grafica 3D implementata in un browser senza bisogno di plugin aggiuntivi. È basato sulle OpenGL Es 2.0 [41] e utilizza il linguaggio di shading delle OpenGL, GLSL. Per C/C++ il modulo NaCl può utilizzare la libreria standard OpenGL Es 2.0.

Tumbler è un esempio C++ che dimostra lo standard OpenGL.

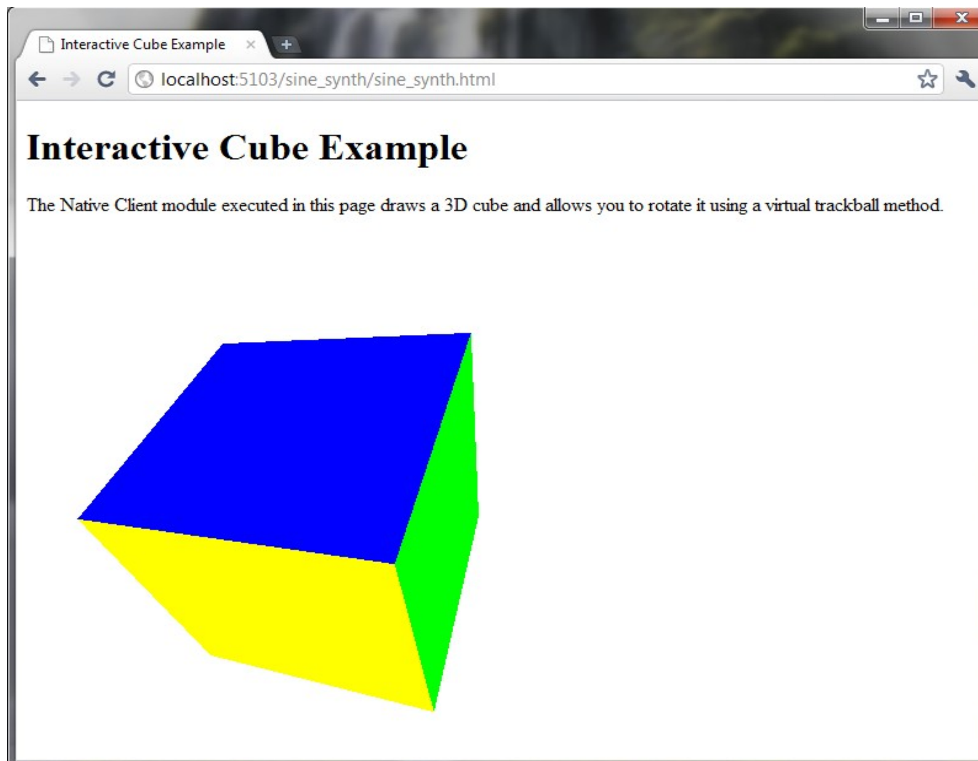

Figura 10: Esecuzione dell'esempio "Tumbler".

Valutazione del Native Client

Vedere e comprendere gli esempi messi a disposizione dal team di Google può aiutare a capire che tipo di contenuti e di applicazioni possono essere realizzate sul web con l'aiuto di Google Native Client e del suo SDK.

Per valutare il Native Client è però necessario effettuare dei test di persona, per provare l'utilità dello strumento e per capire che tipo di differenza ci sia dal punto di vista delle prestazioni tra un normale eseguibile e un applicazione NaCl

Ho effettuato quindi varie prove per capire tramite codice la struttura di un programma NaCl, partendo da un esempio semplice (ma anche incompleto) come può essere "Hello World" per poi arrivare ad un esempio più completo la cui struttura è come quella generale descritta precedentemente.

Prima di proseguire occorre notare che non tutti i test possono essere effettuati.

Questo a causa dei vincoli imposti dal Native Client o della complessità dell'attuale versione dell'SDK. Ad esempio, uno dei vincoli dettati dalle regole del NaCl riguarda l'accesso alla rete, infatti le chiamate di sistema relative non sono consentite: può essere usato solo JavaScript. Questa limitazione chiaramente rende inefficaci i test

di confronto riguardanti l'accesso alla rete, visto che le operazioni che si potrebbero effettuare da un normale eseguibile non si possono portare sul modulo NaCl.

Ad esempio volendo scaricare un file in rete, con il Native Client sarà tutto affidato al browser e non al modulo NaCl, mentre con un normale eseguibile le operazioni si possono gestire manualmente.

Un' altra limitazione riguarda l'accesso al filesystem: non è possibile accedere ai file locali. Questa è una misura aggiuntiva di sicurezza visto che il Native Client dovrebbe essere usato solo per pura computazione software. Per l'accesso al filesystem dovrebbero essere utilizzati strumenti alternativi.

Visti questi vincoli ho sviluppato un semplice programma per valutare la differenza di prestazioni tra un normale eseguibile programmato in C++ e un applicazione NaCl e per capire come il codice possa essere adattato per funzionare tramite l'SDK del Native Client. L'esempio che ho creato riguarda la gestione della memoria tramite hash table. In informatica una hash table [5] è una struttura dati usata per mettere in corrispondenza una data *chiave* con un dato *valore*. Può usare qualsiasi tipo di dato come indice e tutte le operazioni si possono fare in tempo costante. L'hash table è molto utilizzata nei metodi di ricerca chiamati Hashing. L'hashing è un'estensione della ricerca indicizzata da chiavi che gestisce problemi di ricerca nei quali le chiavi di ricerca non presentano queste proprietà. Una ricerca basata su hashing è completamente diversa da una basata su confronti: invece che muoversi nella struttura data in funzione dell'esito dei confronti tra chiavi, si cerca di accedere agli elementi nella tabella in modo diretto tramite operazioni aritmetiche che trasformano le chiavi in indirizzi della tabella. Il primo passo per realizzare algoritmi di ricerca tramite hashing è quello di determinare la *funzione di hash*: il dato da indicizzare viene trasformato da un'apposita funzione in un indirizzo.

Idealmente, chiavi diverse dovrebbero essere trasformate in indirizzi differenti, ma poiché non esiste la funzione di hash perfetta, è possibile che due o più chiavi diverse siano convertite nello stesso indirizzo. Il caso in cui la funzione hash applicata a due chiavi diverse genera un medesimo indirizzo viene chiamato collisione e può essere gestito in vari modi.

Per la gestione dell'hash table ho utilizzato una libreria [38] che ne fornisce una semplice implementazione. Essa utilizza chiavi di tipo stringa e le associa ad un puntatore di tipo void che rappresenta l'elemento. La dimensione della tabella e la funzione di hash sono specificate alla sua creazione. Se non si specifica una funzione di hash viene usata quella di default, una semplice funzione che restituisce come hash il valore ASCII [39] della chiave. Per gestire le collisioni, ogni elemento della tabella è un puntatore ad una lista collegata (linked list), in questo modo i valori multipli con lo stesso hash possono essere memorizzati. Le strutture dati utilizzate sono due, una per la hash table e una per la lista collegata.

Le funzioni implementate sono:

– `hashtbl_create(dimensione, funzione di hash)`: per creare la hash table

– `hashtbl_destroy(puntatore alla hash table)`: per liberare la memoria occupata dalla hash table

– `hashtbl_insert(puntatore alla hash table, chiave, valore)`: per inserire un elemento costituito da una chiave ed un valore nell'hash table

– `hashtbl_remove(puntatore alla hash table, chiave)`: per rimuovere un elemento dall'hash table

– `hashtbl_get(puntatore alla hash table, chiave)`: per ricercare una chiave nella hash table e ottenere il valore corrispondente.

Utilizzando queste funzioni il programma crea un hash table e inserisce un milione di elementi, ognuno con la chiave costituita dal numero dell'elemento (da 0 a 999.999) e con il valore costituito da una stringa costante: “HashTableValue”. Una volta terminato l'inserimento vengono cancellati 500.000 elementi in modo “random”.

Per farlo ho sviluppato una funzione (chiamata appunto random) che genera un numero “casuale” tra 0 e 999.999, questo numero sarà usato come chiave per cancellare l'elemento con la chiave corrispondente. Prima di effettuare l'eliminazione è necessaria anche una ricerca dell'elemento per evitare di provare ad eliminare più volte la stessa chiave, visto che la funzione random() potrebbe generare più volte lo stesso numero, e di conseguenza la stessa chiave.

In seguito la memoria viene liberata tramite la funzione apposita.

Per calcolare il tempo di esecuzione ho utilizzato la funzione clock() che restituisce il numero di "tick" della CPU utilizzati dal processo da quando è partito. Dividendo il numero di cicli per CLOKS_PER_SEC, una costante che definisce il numero di "tick" della CPU che trascorrono in un secondo, si ottiene il tempo di esecuzione effettivo in secondi.

La funzione clock() e la costante CLOKS_PER_SEC sono definite dalla libreria standard time.h.

Per calcolare il tempo di esecuzione nella versione NaCl dell'esempio utilizzo le funzioni time() e difftime() (sempre definite in time.h) a causa di un errore dell'SDK NaCl che non permette l'uso di clock().

Di seguito le parti salienti del codice.

```
//Creazione della hash table
if(!(hashtbl=hashtbl_create(1000000, NULL))) {
      return 0;
}
//Inserimento degli elementi nella hash table

for (i=0;i<1000000;i++){

      chiave=i;

      controllo=

      hashtbl_insert(hashtbl,(char*)&chiave,&valore);

      if (controllo!=0){

            return false;

      }

}
```

```
//Cancellazione degli elementi dalla hash table
for (i=0;i<500000;i++){
	while(searchValue==NULL){
		chiave=random(1000000);
		searchValue=
		hashtbl_get(hashtbl,char*)&chiave);
		if (searchValue!=NULL){
			controllo=
			hashtbl_remove(hashtbl,(char*)&chiave);
			if (controllo!=0){
				return false;
			}
		}
	}
}
//Liberazione della memoria
hashtbl_destroy(hashtbl);


//Calcolo del tempo impiegato per l'esecuzione
numeroCicli=clock();
numeroSecondi=((float)numeroCicli)/
((float)CLOCKS_PER_SEC);
```

Per le operazioni di build ho utilizzato Microsoft Visual Studio.

Portare il codice sull'applicazione NaCl comporta una serie di modifiche rispetto ai file che compongono un normale esempio, non tanto per il codice in sé, che non ha richiesto modifiche sostanziali, ma per permettere a JavaScript di poter richiamare la funzione che esegue questo codice. Per farlo occorrono delle modifiche al "JavaScript Bridge", oltre chiaramente ad includere il codice sopracitato nel modulo NaCl e ad adattare la pagina HTML.

Sono partito con la modifiche di un esempio già fatto, il Sine Wave Synthesizer. Questo esempio, descritto in precedenza, è composto da tutti i file necessari per l'esecuzione di un modulo NaCl. Ho modificato l'esempio togliendo tutte le funzioni (come playSound() e stopSound) e attributi (come la frequenza) e riducendo l'insieme di file alla sola struttura utile per l'esecuzione di un esempio.

A questo punto ho inserito le funzioni e ho effettuato le modifiche necessarie per inserire il mio codice di test. I file che non necessitano di modifiche sono i file npp_gate.cc, npn_bridge.cc e tabella_hash_module.cc (nome_esempio_module.cc). Infatti questi tre file sono uguali per tutti gli esempi. Le modifiche sono da effettuare nei file scripting_bridge.cc (e .h) e tabella_hash.cc (e .h), oltre chiaramente alla pagina HTML.

Il "bridge" fa da ponte tra JavaScript e il modulo NaCl, quindi fornisce un interfaccia alla pagina web per richiamare funzioni dell'applicazione NaCl. Per questo motivo è stata necessaria la dichiarazione di un "identificatore" per associare al metodo che eseguirà le operazioni di gestione della memoria una stringa con cui essere richiamato da JavaScript. Il "bridge" ha anche il compito di richiamare questo metodo del modulo NaCl quando JavaScript invoca la stringa a cui è stato associato. In questo caso l'identificatore "id_esegui" viene associato alla stringa "Go" e alla funzione del bridge "Esegui" che a sua volta richiama la funzione del modulo NaCl corrispondente.

Così quando JavaScript invoca la funzione Go() il bridge richiamerà automaticamente la funzione Esegui() dal modulo NaCl.

Di seguito il codice.

```
//Dichiarazione dell'identificatore "id_esegui" e
associazione alla stringa "Go"

NPIdentifier id_esegui = NPN_GetStringIdentifier("Go");

//Associazione dell'identificatore "id_esegui" alla
funzione "Esegui"

method_table->insert(std::pair<NPIdentifier,
MethodSelector>(id_esegui,&ScriptingBridge::Esegui));
```

```
//Funzione dello Scripting Bridge che richiama la
funzione di esecuzione del codice.

Bool ScriptingBridge::Esegui(const NPVariant*
args,uint32_t,arg_count,NPVariant* result)
{
      TableHash* table_hash =
      static_cast<TableHash*>(npp_->pdata);
      if (table_hash) {
            table_hash->Esegui();
            return true;
      }
      return false;
}
```

Allo stesso modo per rendere disponibile alla pagina HTML il tempo di esecuzione finale e una variabile per il controllo degli errori, occorrono altri due identificatori, “id_tempo” e “id_controllo” che saranno associati ognuno ad una funzione che restituisce a JavaScript i valori dei due attributi corrispondenti della classe Table_Hash del modulo NaCl.

Con queste modifiche è stato effettuato un collegamento tra JavaScript e il modulo NaCl.

Occorre quindi modificare il file tabella_hash.cc per inserire il metodo Esegui() e i due attributi descritti sopra.

Prima di tutto bisogna includere la libreria usata per la gestione della hash table: `#include "examples/progetto/hashtable.h"`.

Quindi nella dichiarazione della classe Table_Hash ho aggiunto le dichiarazioni del metodo Esegui(), del metodo Random() per il calcolo della chiave in modo casuale e dei due attributi, uno di tipo float per il tempo di esecuzione e un intero per il controllo degli errori. A questo punto dopo aver aggiunto l'inizializzazione dei due attributi nel costruttore della classe sarà sufficiente definire la classe Esegui() riportando lo stesso codice utilizzato per il normale eseguibile. Occorreranno poi solo alcune modifiche a suddetto codice: aggiornare l'attributo “controllo” nel caso di errori e salvare il tempo di esecuzione nell'attributo “tempo”. Inoltre bisognerà definire il metodo Random() riportando anche qui lo stesso codice utilizzato nel normale eseguibile.

Ora non resta che modificare la pagina HTML per richiamare la funzione Esegui(). Ho inserito un pulsante che richiama una funzione JavaScript che a sua volta richiamando la funzione Go() invoca tramite il "bridge" la funzione Esegui() del modulo NaCl. In modo che cliccando il pulsante con il mouse vengano lanciate le operazioni definite in precedenza di gestione della memoria tramite hash table. Una volta terminate, una finestra di popup avviserà l'utente dell'esito delle operazioni e in caso di successo il tempo di esecuzione impiegato. Di seguito il codice della pagina HTML.

```
//Il pulsante che richiama la funzione JavaScript

<button onclick="go()">Esegui</button>

//La funzione Javascript che invoca il metodo del modulo
NaCl

Function go() {

try {

      table_hash.Go();

      if (table_hash.controllo==0){

            alert("Esecuzione terminata con successo,
tempo                  impiegato: "+table_hash.tempo);

      }

      else{

            alert("Esecuzione fallita! Errore:
                  "+table_hash.controllo+".");

      }

} catch(e) {

      alert(e.message);

            }

}
```

Occorrerà poi una piccola modifica al makefile standard degli esempi per aggiungere ai file da compilare l'implementazione dell'hash table utilizzata.

```
CCFILES = npn_bridge.cc \ npp_gate.cc
\scripting_bridge.cc \ table_hash.cc \
table_hash_module.cc \ hashtable.cc
```

A questo punto sia il normale eseguibile che l'esempio corrispondente del Native Client sono utilizzabili e possono essere confrontati.

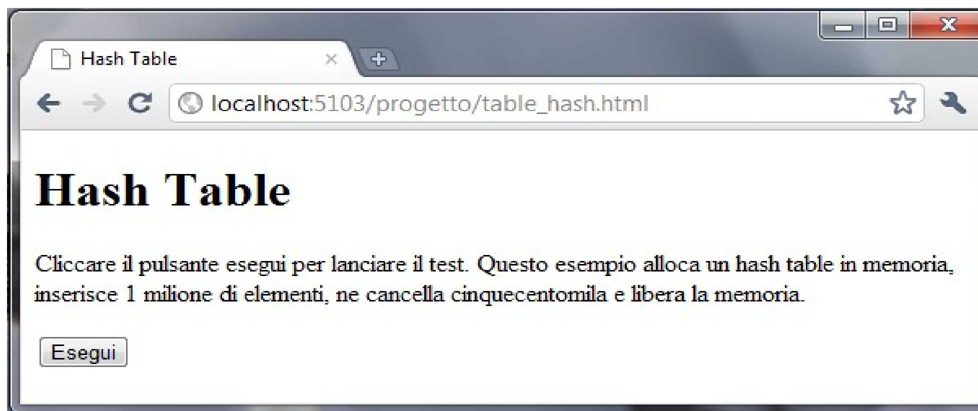

Figura 11: Esecuzione dell'esempio "Hash Table" tramite il modulo NaCl.

Figura 12: Esecuzione dell'esempio "Hash Table" tramite il normale eseguibile.

Misurazioni Effettuate

Ho effettuato alcune misurazioni per verificare le prestazioni.

I test sono stati effettuati con un sistema Windows 7 a 32bit con un processore dual core AMD Phenom II 550 3,10 GHz e 4 GB di memoria RAM. Nella tabella 6 si possono osservare i risultati di dieci esecuzioni consecutive dell'eseguibile normale e della versione Native Client. Purtroppo per un errore della versione corrente dell'SDK il tempo di esecuzione dell'esempio NaCl ha una precisione limitata ai secondi. Nonostante ciò dalle misurazioni ottenute è palese che il tempo medio di esecuzione per la versione normale dell'eseguibile è inferiore di circa 28 secondi rispetto al tempo medio di esecuzione della versione Native Client. Certo questo semplice test non può essere considerato esaustivo, ma il risultato non si avvicina per niente alle misurazioni effettuate dagli sviluppatori del NaCl. Infatti, la diminuzione media di performance stimata del Native Client rispetto ad un normale eseguibile è di circa il 5%, mentre in questo caso precipita addirittura a più del 44%. Che la versione di questo esempio del Native Client non raggiungesse le prestazioni di quella del normale eseguibile era prevedibile; un aumento del tempo di esecuzione di queste proporzioni un po' meno, viste le premesse. Si può ipotizzare, anche se è difficile dare una spiegazione, che ciò sia dovuto al diverso metodo di gestione della memoria del Native Client (segmentazione) rispetto a quello dei sistemi operativi attuali (paginazione).

In particolare, proprio perchè l'esempio in questione è basato sull'allocazione, l'inserimento, la ricerca e l'eliminazione di dati in memoria, invece di riguardare pura computazione. Sicuramente sono necessari test più complessi e approfonditi, ma questi risultati fanno venire alcuni dubbi visto che uno degli obiettivi fondamentali del Native Client è appunto il mantenimento delle prestazioni del codice nativo nel browser.

Numero Esecuzione	Normale Eseguibile	Native Client	Incremento
1	65,13	93	42,8%
2	64,69	92	42,2%
3	64,60	94	45,5%
4	64,85	93	43,4%
5	64,65	93	43,8%
6	64,57	94	45,5%
7	64,55	93	44%
8	64,68	93	43,7%
9	64,49	94	45,7%
10	64,54	94	45,6%
Media	64,68	93,3	44,24%

Tabella 6: Performance ottenute con l'esempio "Hash Table". Le misurazioni sono in secondi.

Conclusioni

Il Native Client è un sistema per incorporare codice nativo x86 untrusted in un' applicazione che viene eseguita in un browser web. Questa tecnologia promette di raggiungere generalmente tre obiettivi: la portabilità, la sicurezza e le prestazioni del codice nativo.

Giudicando lo stato attuale di sviluppo del Native Client non si può ancora affermare che gli obiettivi siano stati raggiunti pienamente. Per quanto riguarda la portabilità, nonostante il team di Google abbia testato il Native Client su vari sistemi operativi e browser non si è ancora al punto in cui poter utilizzare questa tecnologia con qualsiasi configurazione. Attualmente ci sono delle limitazioni significative, infatti, solo con l'integrazione del NaCl in Google Chrome è possibile utilizzare lo strumento e quindi per raggiungere la portabilità relativa ai browser ci sarà ancora da lavorare. Questo aspetto provoca anche una certa perplessità: in teoria la natura open-source del NaCl potrebbe consentirne il porting per altri browser, ma in pratica anche l'SDK del Native Client, fino a questo momento, è intimamente legato al framework su cui poggia Chrome. Anche se in futuro Google dovesse svilupparne una versione "neutrale", resta da vedere se i suoi rivali saranno disposti a seguirne le orme e trasformare il Native Client in uno standard vero e proprio. Diverso è il discorso per i sistemi operativi, i più comuni sono supportati. Mentre non vale lo stesso per le architetture x86 a 64bit e ARM per cui il Native Client è utilizzabile solo in minima parte. Un altro aspetto della portabilità è quello del porting del codice esistente. In questo caso ci si può ritenere più soddisfatti vista la semplicità con cui è possibile portare il codice: anche se ciò chiaramente dipende dalla complessità e dal tipo di codice, in generale il porting è lineare.

Per quanto riguarda la sicurezza il Native Client fornisce un insieme consistente di strumenti per proteggere il sistema da codice untrusted. Oltre alla sandbox interna, che già da sola rappresenta un dispositivo di sicurezza molto robusto, vengono forniti altri meccanismi ridondanti, come un ulteriore strato di protezione fornito dalla sandbox esterna, CPU black-list e module black-list.

Dare una valutazione della sicurezza attualmente è difficile: sicuramente ancora l'obiettivo non è raggiunto visto che periodicamente sul sito del Native Client vengono portati alla luce possibili problemi che la riguardano.

Naturalmente servirebbero dei test aggiuntivi per valutare meglio questo aspetto. Considerando però i numerosi provvedimenti relativi alla sicurezza e l'importanza che il team di Google da alle discussioni pubbliche che la riguardano per poter fornire dei meccanismi migliori, si può pensare che una volta che il Native Client sarà rilasciato raggiungerà sicuramente un livello superiore di sicurezza rispetto alle tecnologie web per codice nativo fin'ora rilasciate.

Ponendo invece l'attenzione sulle prestazioni, la promessa di mantenere le stesse prestazioni del codice nativo nel browser può essere valutata positivamente solo in parte. In base ai dati dei vari test effettuati dagli sviluppatori del Native Client si può osservare che la penalità di performance rilevata con l'esecuzione di un' applicazione NaCl è bassa, particolarmente in scenari CPU-bound per i quali il sistema è progettato. Tuttavia i risultati che ho ottenuto con i miei test non corrispondono a queste valutazioni, visto che la penalità di performance è molto più elevata. C'è da considerare che l'utilizzo del Native Client è destinato ad applicazioni per siti web e quindi teoricamente operazioni di gestione della memoria come quelle del mio esempio non dovrebbero essere comuni. Ad ogni modo i risultati ottenuti non sono certo soddisfacenti. Sarebbero necessari test più complessi per approfondire la questione, come ad esempio quelli non realizzabili a causa della versione incompleta dell'SDK del Native Client, oppure dei test per approfondire le applicazioni grafiche. Con i dati attualmente disponibili si può affermare che l'obiettivo del mantenimento delle stesse prestazioni del codice nativo nel browser è quasi stato raggiunto, visto che i test eseguiti dagli sviluppatori del NaCl sono certamente più rilevanti dei miei, ma qualche dubbio rimane e non si può ancora dire con certezza che sotto questo punto di vista il NaCl sia superiore a tutte le altre soluzioni.

Questa è chiaramente una valutazione preliminare riguardante lo stato attuale del Native Client, una valutazione più completa potrà sicuramente essere fatta una volta che lo sviluppo giungerà al termine e il Native Client verrà rilasciato ufficialmente.

In quel momento si apriranno anche una serie di spunti futuri per l'utilizzo di questo software. Gli sviluppatori stessi del Native Client hanno ammesso che ci sono campi di applicazione per questa tecnologia al di fuori del browser ma che per il momento non fanno parte dei loro programmi.

Sul web c'è chi ipotizza che il Native Client rappresenti il nucleo di un sistema operativo. Google smentisce tali voci e sottolinea il fatto che il NaCl sfrutta le risorse di CPU e della Ram, ma non può scrivere sul disco fisso. Anche se il software non è stato sviluppato per questo fine sicuramente potrebbe rivoluzionare l'utilizzo del web se gli obiettivi dichiarati saranno raggiunti. E se in futuro salveremo i nostri dati solo su internet, allora il sistema operativo non avrà neanche più bisogno di un disco fisso.

Bibliografia

[1] Bennet Yee, David Sehr, Gregory Dardyk, J. Bradley Chen, Robert Muth, Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar.
Google Inc. "Native Client: A Sandbox for Portable, Untrusted x86 Native Code". IEEE Symposium on Security and Privacy, Maggio 2009.
http://nativeclient.googlecode.com/svn/data/docs_tarball/nacl/googleclient/native_client/documentation/nacl_paper.pdf

[2] Alan Donovan, Robert Muth, Brad Chen, David Sehr.
"PNaCl:Portable *Native Client* Executables". Febbraio 2010.
http://nativeclient.googlecode.com/svn/data/site/pnacl.pdf

[3] David Sehr, Robert Muth, Cliff Biffle, Victor Khimenko, Egor Pasko,
Karl Schimpf, Bennet Yee, Brad Chen. "Adapting Software Fault Isolation to Contemporary CPU Architectures". The 19th USENIX Security Symposium, Agosto 2010.
http://nativeclient.googlecode.com/svn/data/site/NaCl_SFI.pdf

[4] Markus Mandau. "Google Native Client: il computer nel browser". Chip http://www.chip.it/ , 2009.

[5] http://www.wikipedia.org , 2010.

[6] http://www.chromium.org/nativeclient/design-documents/native-client-integration-with-chrome , 2010.

[7] http://code.google.com/p/nativeclient/ , 2010.

[8] http://blog.chromium.org/2010/03/native-client-and-web-portability.html , 2010.

[9] http://www.w3.org/DOM/ , 2010.

[10] http://www.arm.com/ , 2010.

[11] http://www.google.it/chrome/ , 2010.

[12] http://www.mozilla-europe.org/it/firefox/ , 2010.

[13] http://www.opera.com/ , 2010.

[14] http://www.apple.com/it/safari/ , 2010.

[15] http://caminobrowser.org/ , 2010.

[16] http://www.python.it/ , 2010.

[17] http://code.google.com/intl/it-IT/chrome/chromeframe/ , 2010.

[18] http://www.gnu.org/software/autoconf/ , 2010.

[19] http://gcc.gnu.org/ , 2010.

[20] R. Wahbe, S. Lucco, T. E. Anderson, S.L. Graham. "Efficient software-based fault isolation". ACM SIGOPS Operating System Review, Dicembre 1993.
http://www.dsi.uniroma1.it/~vamd/TSL/sfi.pdf

[21] Google Inc. "Android: an open handset alliance project"
http://code.google.com/android/ , 2007.

[22] John R. Douceur, Jeremy Elson, Jon Howell, Jacob R. Lorch. "Leveraging Legacy Code to Deploy Desktop Applications on the Web". *Microsoft Research*, Dicembre 2008.
http://research.microsoft.com/pubs/72878/xax-osdi08.pdf

[23] A. Denning. "ActiveX Controls Inside Out." Microsoft Press, Maggio 1997.

[24] M. Abadi, M. Budiu, U. Erlingsson, J. Ligatti. "ControlFlow Integrity: Principles, Implementations, and Applications". Novembre 2005.
http://research.microsoft.com/pubs/69217/ccs05-cfi.pdf

[25] B. N. Bershad, C. Chambers, S. Eggers, C. Maeda, D. McNamee, P. Pardyak, S. Savage, E. G. Sirer. "Spin: An Extensible Microkernel for Application-specific Operating System Services". Febbraio 1994.
http://www.cs.cornell.edu/People/egs/papers/spin-sigops94.ps

[26] Y. Endo, M. Seltzer,J. Gwertzman, C. Small, K. A. Smith, D. Tang. "VINO: The 1994 Fall Harvest". Dicembre 1994.
http://citeseerx.ist.psu.edu/viewdoc/download?
doi=10.1.1.36.7215&rep=rep1&type=ps

[27] M. Swift, M. Annamalai, B. Bershad, H. Levy. "Recovering device drivers". December 2004.
http://nooks.cs.washington.edu/recovering-drivers.pdf

[28] G. Nelson. "System programming in Modula-3". Prentice-Hall, 1991.

[29] Mitre’s Common Vulnerabilities and Exposures site
http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=ptrace , 2010.

[30] https://wiki.mozilla.org/NPAPI:Pepper , 2010.

[31] Information Sciences Institute, University of Southern California, Marina del Rey, California. "TRANSMISSION CONTROL PROTOCOL, DARPA INTERNET PROGRAM, PROTOCOL SPECIFICATION". September 1981.

[32] J. Postel, ISI. "User Datagram Protocol". Agosto 1980.

[33] http://www.ieee.org/index.html , 2010.

[34] Google Inc., Mountain View, Canada. "Method for safely executing an untrusted native code module on a computing device". United States Patent Application *20090282474.* Novembre 2009. http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PG01&p=1&u=/netahtml/PTO/srchnum.html&r=1&f=G&l=50&s1=20090282474.PGNR.&OS=DN/20090282474&RS=DN/20090282474

[35] www.libsdl.org, 2010.

[36] http://www.eetimes.com/design/memory-design/4024961/Optimizing-Memcpy-improves-speed , 2010.

[37] Bullet physics SDK. http://www.bulletphysics.com , 2010.

[38] http://en.literateprograms.org/Hash_table_%28C%29 , 2010.

[39] http://www.asciitable.com/ , 2010.

[40] http://www.opengl.org/ , 2010.

[41] http://www.khronos.org/opengles/ , 2010.

[42] http://www.microsoft.com/visualstudio/ , 2010.

[43] http://www.eclipse.org/ , 2010.

[44] http://www.gnu.org/licenses/gpl.html , 2010.

[45] http://googlecode.blogspot.com/2008/12/native-client-technology-for-running.html , Dicembre 2008.