Paweł Hajdan

```
#include <limits.h>         // So we can set the bounds of our types
#include <stddef.h>         // For size_t
#include <string.h>         // for memcpy

#include "base/port.h"    // Types that only need exist on certain systems

#ifndef COMPILER_MSVC
// stdint.h is part of C99 but MSVC doesn't have it.
#include <stdint.h>         // For intptr_t.
#endif

typedef signed char         schar;
typedef signed char         int8;
typedef short               int16;
// TODO: Remove these type guards.  These are to avoid conflicts with
// obsolete/protypes.h in the Gecko SDK.
#ifndef _INT32
#define _INT32
typedef int                 int32;
#endif
```

Test czytelności

Paweł Hajdan

i takie tam

Paweł Hajdan

1.5 Design notes

This section is non-normative.

It must be admitted that many aspects of HTML appear at first glance to be nonsensical and inconsistent.

HTML, its supporting DOM APIs, as well as many of its supporting technologies, have been developed over a period of several decades by a wide array of people with different priorities who, in many cases, did not know of each other's existence.

Features have thus arisen from many sources, and have not always been designed in especially consistent ways. Furthermore, because of the unique characteristics of the Web, implementation bugs have often become de-facto, and now de-jure, standards, as content is often unintentionally written in ways that rely on them before they can be fixed.

Despite all this, efforts have been made to adhere to certain design goals. These are described in the next few subsections.

W3C Working Draft

HTML Design Principles

W3C Working Draft 26 May 2009

This Version:
http://www.w3.org/TR/2009/WD-html-design-principles-20090526/
Latest Version:
http://www.w3.org/TR/html-design-principles/
Editors:
Anne van Kesteren (Opera Software ASA) <annevk@opera.com>
Maciej Stachowiak (Apple Inc) <mjs@apple.com>

Kompatybilność

- Obsługa istniejących dokumentów
- Kompatybilność wstecz
- Nie rewolucja, lecz ewolucja

Wielość implementacji

- Dobrze zdefiniowana semantyka
- Prostota
- Wyspecyfikowana obsługa błędów

Web Sockets

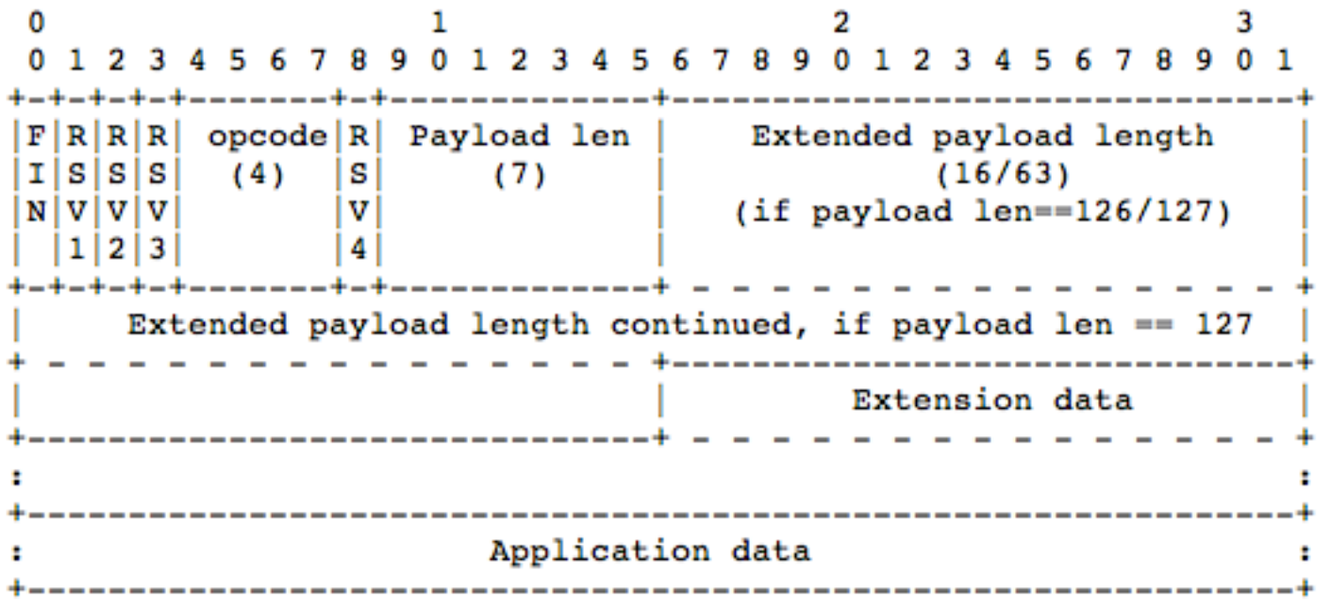
Przykład użycia

```
var ws = new WebSocket("ws://example.com/wsserver");

ws.onopen = function() {
    alert('onopen!');
}

ws.onmessage = function(messageEvent) {
    alert(messageEvent.data);
}

ws.send('Hello, World!');
```

Narzut HTTP

```
     HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

Ramka Web Sockets



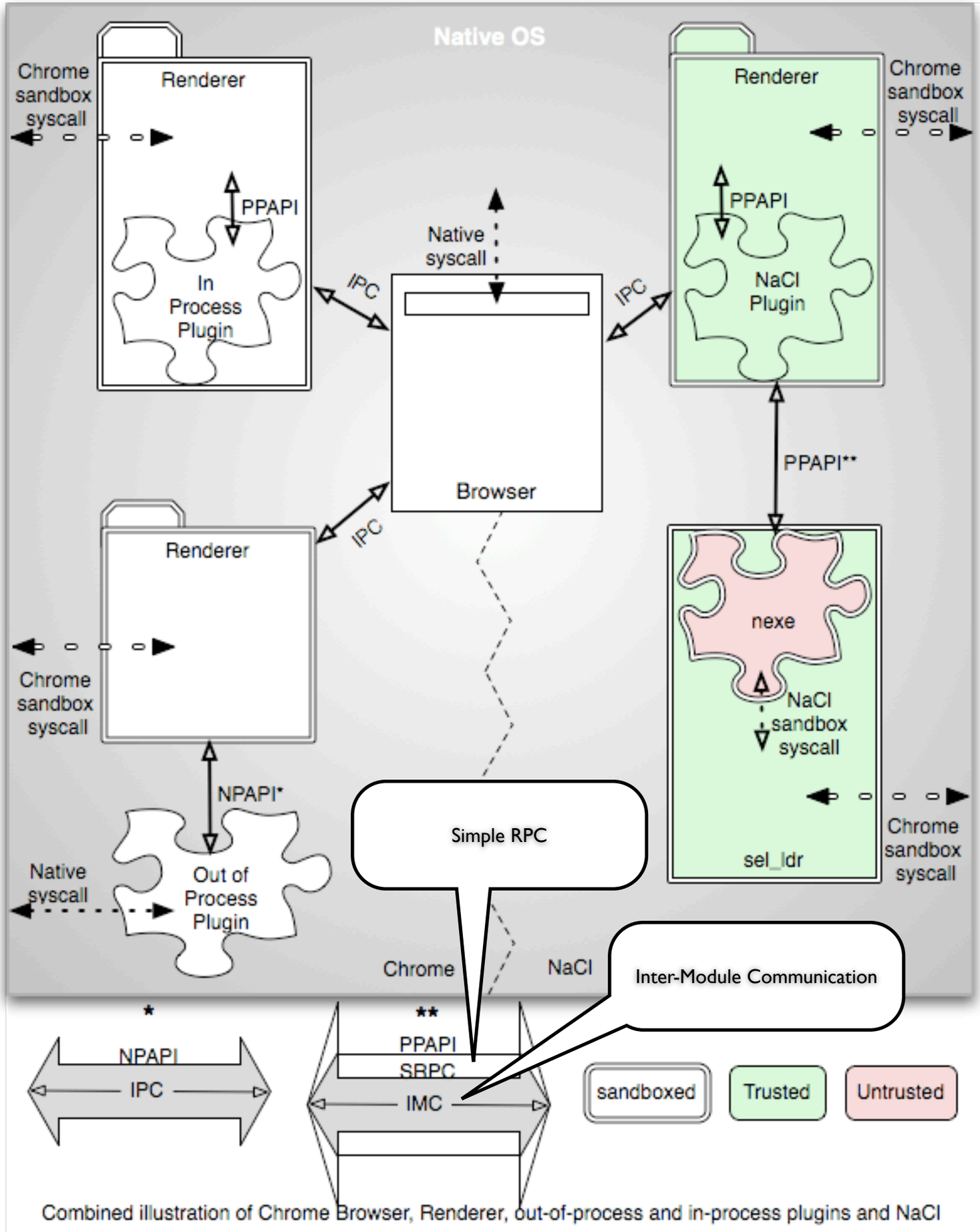
Native Client: A Sandbox for Portable, Untrusted x86 Native Code

Bennet Yee, David Sehr, Gregory Dardyk, J. Bradley Chen, Robert Muth,
Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar
Google Inc.

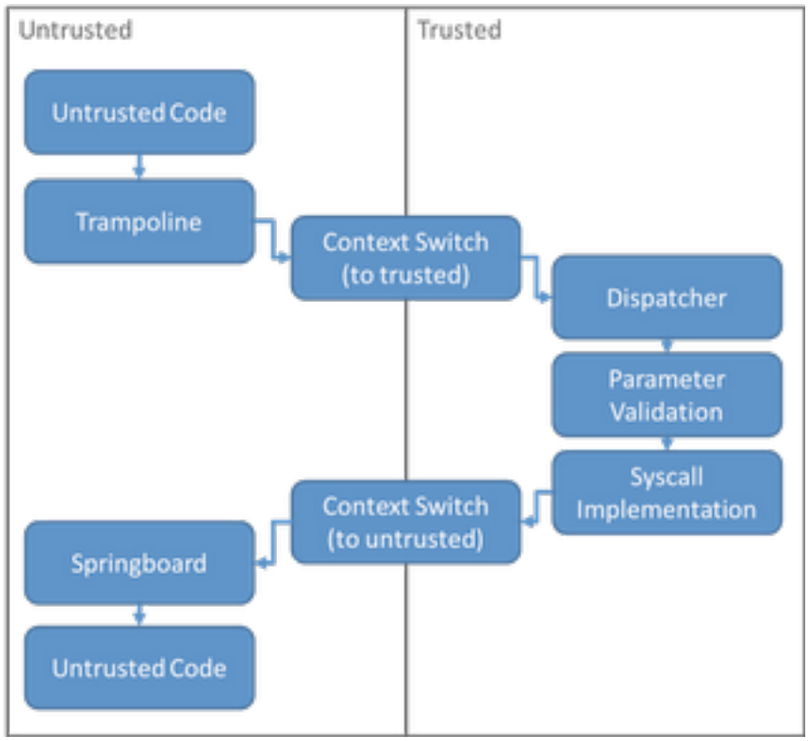
NaCl

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

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



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


Untrusted
Trusted
Untrusted Code
Trampoline
Context Switch
(to trusted)
Dispatcher
Parameter
Validation
Syscall
Implementation
Context Switch
(to untrusted)
Springboard
Untrusted Code

Atak #1

1. Zakończenie walidacji
2. Aplikacja wykonuje unmap sekcji kodu
3. Na to miejsce wstawiany jest dowolny kod

Atak #2

1. Aplikacja zmienia wartość rejestru EFLAGS
2. Zmieniony zostaje bit DF (direction flag)
3. Aplikacja żąda wywołania systemowego
4. NaCl nie czyści EFLAGS

Atak #3

1. Niepełne sprawdzanie instrukcji skoku
2. Skok w dowolne miejsce

Atak #4

1. Aplikacja wywołuje jedną z funkcji SRPC
2. Wejście nie jest poprawnie sprawdzane
3. Ujemna liczba całkowita trafia do malloc

Atak #5

1. Aplikacja może czytać zmienne środowiska
2. Wyciek informacji

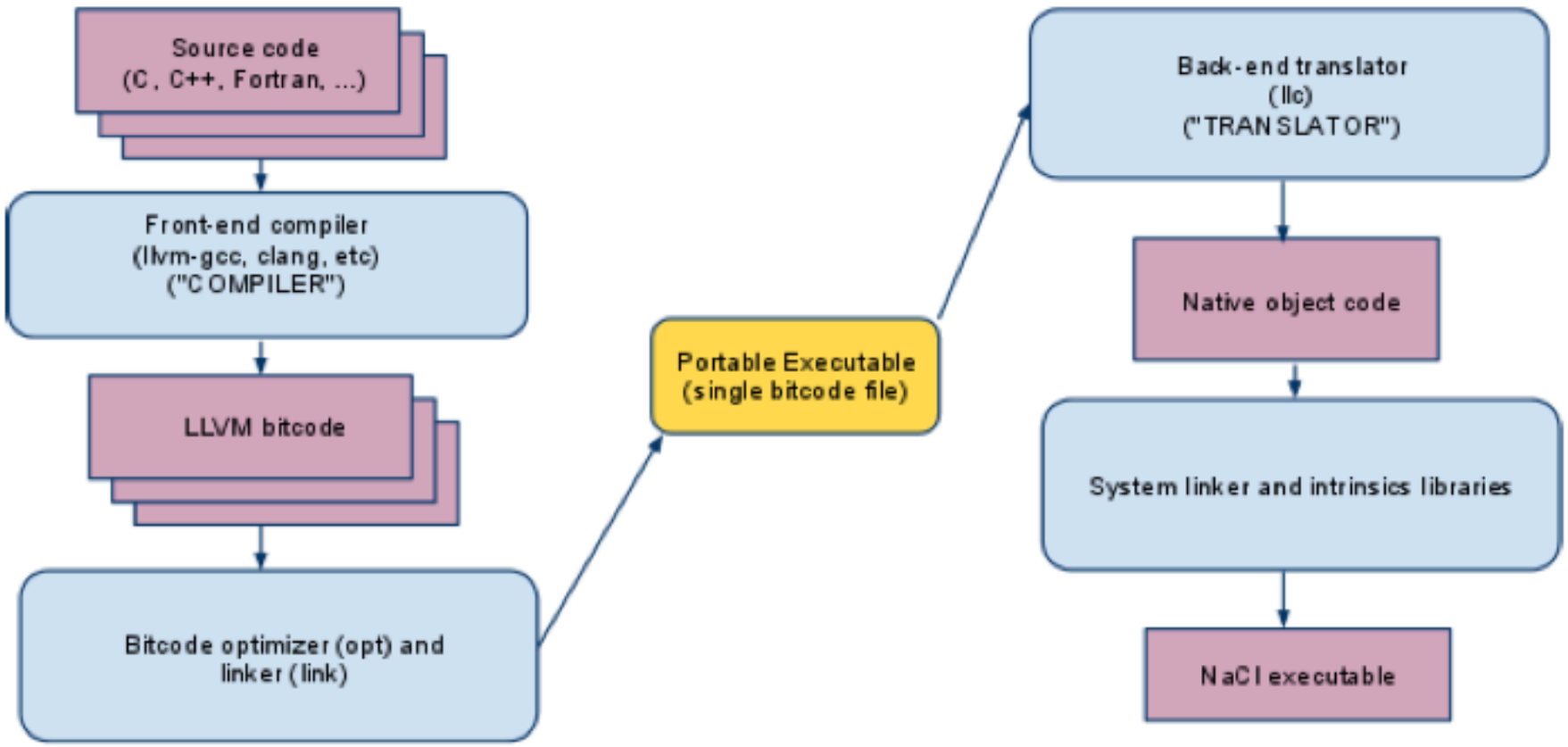
Atak #6

1. Dane związane z renderowaniem są zapisywane w katalogu tymczasowym
2. Mogą one być wczytane przez inny proces
3. Naruszenie polityki same-origin

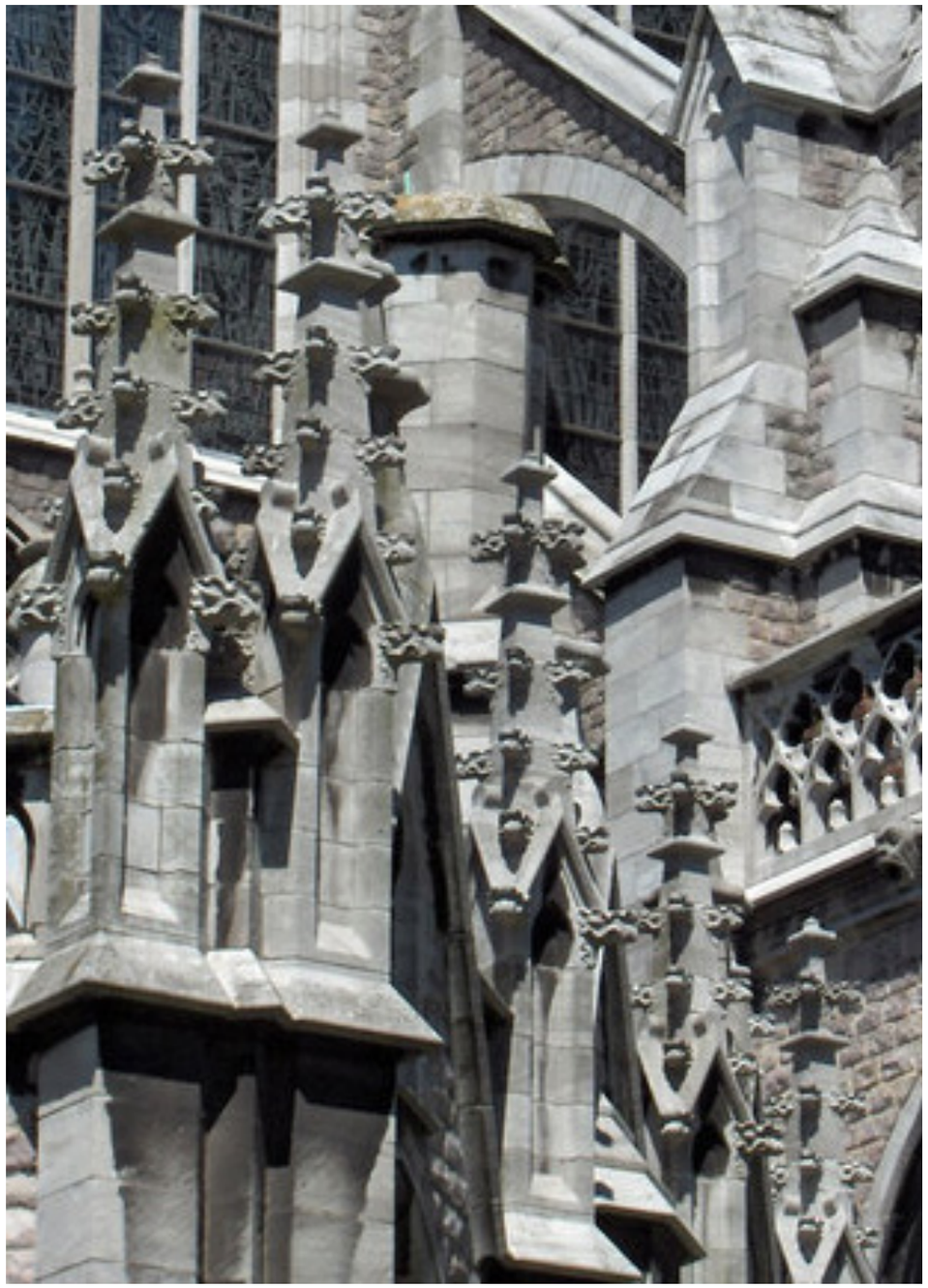
Atak #7

1. Aplikacja wysyła spreparowaną odpowiedź SRPC
2. Podwójne zwolnienie pamięci
3. Uszkodzenie sterty

Sandbox zewnętrzny



Portable NaCl



pinnacle

Baza danych

w przeglądarce

Baza danych

w przeglądarce

Baza danych

w przeglądarce

```
using namespace WebCore;

// Defined in Chromium's codebase in third_party/sqlite/src/os_unix.c
extern "C" {
void chromium_sqlite3_initialize_unix_sqlite3_file(sqlite3_file* file);
int chromium_sqlite3_fill_in_unix_sqlite3_file(sqlite3_vfs* vfs, int fd, int dirfd, sqlite3_file* file, const char* fileName, int noLock);
int chromium_sqlite3_get_reusable_file_handle(sqlite3_file* file, const char* fileName, int flags, int* fd);
void chromium_sqlite3_update_reusable_file_handle(sqlite3_file* file, int fd, int flags);
void chromium_sqlite3_destroy_reusable_file_handle(sqlite3_file* file);
}

// Chromium's Posix implementation of SQLite VFS
namespace {
```

```
// Opens a file.
//
// vfs - pointer to the sqlite3_vfs object.
// fileName - the name of the file.
// id - the structure that will manipulate the newly opened file.
// desiredFlags - the desired open mode flags.
// usedFlags - the actual open mode flags that were used.
int chromiumOpen(sqlite3_vfs* vfs, const char* fileName,
                 sqlite3_file* id, int desiredFlags, int* usedFlags)
{
    chromium_sqlite3_initialize_unix_sqlite3_file(id);
    int fd = -1;
    int result = chromium_sqlite3_get_reusable_file_handle(id, fileName, desiredFlags, &fd);
    if (result != SQLITE_OK)
        return result;

    if (fd < 0) {
        fd = PlatformBridge::databaseOpenFile(fileName, desiredFlags);
        if ((fd < 0) && (desiredFlags & SQLITE_OPEN_READWRITE)) {
            int newFlags = (desiredFlags & ~(SQLITE_OPEN_READWRITE | SQLITE_OPEN_CREATE)) | SQLITE_OPEN_READONLY;
            fd = PlatformBridge::databaseOpenFile(fileName, newFlags);
        }
    }
    if (fd < 0) {
        chromium_sqlite3_destroy_reusable_file_handle(id);
        return SQLITE_CANTOPEN;
    }

    if (usedFlags)
        *usedFlags = desiredFlags;
    chromium_sqlite3_update_reusable_file_handle(id, fd, desiredFlags);

    fcntl(fd, F_SETFD, fcntl(fd, F_GETFD) | FD_CLOEXEC);

    // The mask 0x00007F00 gives us the 7 bits that determine the type of the file SQLite is trying to open.
    int fileType = desiredFlags & 0x00007F00;
    int noLock = (fileType != SQLITE_OPEN_MAIN_DB);
    result = chromium_sqlite3_fill_in_unix_sqlite3_file(vfs, fd, -1, id, fileName, noLock);
    if (result != SQLITE_OK)
        chromium_sqlite3_destroy_reusable_file_handle(id);
    return result;
}
```

```
// Databases ------------------------------------------------------------------

PlatformFileHandle PlatformBridge::databaseOpenFile(const String& vfsFileName, int desiredFlags)
{
    return webKitClient()->databaseOpenFile(WebString(vfsFileName), desiredFlags);
}

int PlatformBridge::databaseDeleteFile(const String& vfsFileName, bool syncDir)
{
    return webKitClient()->databaseDeleteFile(WebString(vfsFileName), syncDir);
}

long PlatformBridge::databaseGetFileAttributes(const String& vfsFileName)
{
    return webKitClient()->databaseGetFileAttributes(WebString(vfsFileName));
}

long long PlatformBridge::databaseGetFileSize(const String& vfsFileName)
{
    return webKitClient()->databaseGetFileSize(WebString(vfsFileName));
}
```

```
void DatabaseMessageFilter::OnDatabaseOpenFile(const string16& vfs_file_name,
                                               int desired_flags,
                                               IPC::Message* reply_msg) {
  DCHECK(BrowserThread::CurrentlyOn(BrowserThread::FILE));
  base::PlatformFile file_handle = base::kInvalidPlatformFileValue;
  base::PlatformFile target_handle = base::kInvalidPlatformFileValue;
  string16 origin_identifier;
  string16 database_name;

  // When in incognito mode, we want to make sure that all DB files are
  // removed when the incognito profile goes away, so we add the
  // SQLITE_OPEN_DELETEONCLOSE flag when opening all files, and keep
  // open handles to them in the database tracker to make sure they're
  // around for as long as needed.
  if (vfs_file_name.empty()) {
    VfsBackend::OpenTempFileInDirectory(db_tracker_->DatabaseDirectory(),
                                        desired_flags, &file_handle);
  } else if (DatabaseUtil::CrackVfsFileName(vfs_file_name, &origin_identifier,
                                            &database_name, NULL) &&
             !db_tracker_->IsDatabaseScheduledForDeletion(origin_identifier,
                                                          database_name)) {
    FilePath db_file =
        DatabaseUtil::GetFullFilePathForVfsFile(db_tracker_, vfs_file_name);
    if (!db_file.empty()) {
      if (db_tracker_->IsIncognitoProfile()) {
        db_tracker_->GetIncognitoFileHandle(vfs_file_name, &file_handle);
        if (file_handle == base::kInvalidPlatformFileValue) {
          VfsBackend::OpenFile(db_file,
                               desired_flags | SQLITE_OPEN_DELETEONCLOSE,
                               &file_handle);
          if (VfsBackend::FileTypeIsMainDB(desired_flags) ||
              VfsBackend::FileTypeIsJournal(desired_flags))
            db_tracker_->SaveIncognitoFileHandle(vfs_file_name, file_handle);
        }
      } else {
        VfsBackend::OpenFile(db_file, desired_flags, &file_handle);
      }
    }
  }
}
```

```
// Then we duplicate the file handle to make it useable in the renderer
// process. The original handle is closed, unless we saved it in the
// database tracker.
bool auto_close = !db_tracker_->HasSavedIncognitoFileHandle(vfs_file_name);
VfsBackend::GetFileHandleForProcess(peer_handle(), file_handle,
                                    &target_handle, auto_close);

DatabaseHostMsg_OpenFile::WriteReplyParams(
      reply_msg,
#if defined(OS_WIN)
      target_handle
#elif defined(OS_POSIX)
      base::FileDescriptor(target_handle, auto_close)
#endif
      );
  Send(reply_msg);
}
```

```cpp
namespace base {

// ------------------------------------------------------------------------
// We introduct a special structure for file descriptors in order that we are
// able to use template specialisation to special-case their handling.
//
// WARNING: (Chromium only) There are subtleties to consider if serialising
// these objects over IPC. See comments in ipc/ipc_message_utils.h
// above the template specialisation for this structure.
// ------------------------------------------------------------------------
struct FileDescriptor {
  FileDescriptor()
      : fd(-1),
        auto_close(false) { }

  FileDescriptor(int ifd, bool iauto_close)
      : fd(ifd),
        auto_close(iauto_close) { }

  bool operator==(const FileDescriptor& other) const {
    return (fd == other.fd && auto_close == other.auto_close);
  }

  // A comparison operator so that we can use these as keys in a std::map.
  bool operator<(const FileDescriptor& other) const {
    return other.fd < fd;
  }

  int fd;
  // If true, this file descriptor should be closed after it has been used. For
  // example an IPC system might interpret this flag as indicating that the
  // file descriptor it has been given should be closed after use.
  bool auto_close;
};

}  // namespace base
```

```cpp
#if defined(OS_POSIX)
void ParamTraits<base::FileDescriptor>::Write(Message* m, const param_type& p) {
  const bool valid = p.fd >= 0;
  WriteParam(m, valid);

  if (valid) {
    if (!m->WriteFileDescriptor(p))
      NOTREACHED();
  }
}

bool ParamTraits<base::FileDescriptor>::Read(const Message* m, void** iter,
                                             param_type* r) {
  bool valid;
  if (!ReadParam(m, iter, &valid))
    return false;

  if (!valid) {
    r->fd = -1;
    r->auto_close = false;
    return true;
  }

  return m->ReadFileDescriptor(iter, r);
}

void ParamTraits<base::FileDescriptor>::Log(const param_type& p,
                                            std::string* l) {
  if (p.auto_close) {
    l->append(StringPrintf("FD(%d auto-close)", p.fd));
  } else {
    l->append(StringPrintf("FD(%d)", p.fd));
  }
}
#endif  // defined(OS_POSIX)
```

```
#if defined(OS_POSIX)
bool Message::WriteFileDescriptor(const base::FileDescriptor& descriptor) {
  // We write the index of the descriptor so that we don't have to
  // keep the current descriptor as extra decoding state when deserialising.
  WriteInt(file_descriptor_set()->size());
  if (descriptor.auto_close) {
    return file_descriptor_set()->AddAndAutoClose(descriptor.fd);
  } else {
    return file_descriptor_set()->Add(descriptor.fd);
  }
}

bool Message::ReadFileDescriptor(void** iter,
                                 base::FileDescriptor* descriptor) const {
  int descriptor_index;
  if (!ReadInt(iter, &descriptor_index))
    return false;

  FileDescriptorSet* file_descriptor_set = file_descriptor_set_.get();
  if (!file_descriptor_set)
    return false;

  descriptor->fd = file_descriptor_set->GetDescriptorAt(descriptor_index);
  descriptor->auto_close = true;

  return descriptor->fd >= 0;
}

void Message::EnsureFileDescriptorSet() {
  if (file_descriptor_set_.get() == NULL)
    file_descriptor_set_ = new FileDescriptorSet;
}

#endif
```

```cpp
bool Channel::ChannelImpl::ProcessOutgoingMessages() {
  DCHECK(!waiting_connect_);  // Why are we trying to send messages if there's
                              // no connection?
  if (output_queue_.empty())
    return true;

  if (pipe_ == -1)
    return false;

  // Write out all the messages we can till the write blocks or there are no
  // more outgoing messages.
  while (!output_queue_.empty()) {
    Message* msg = output_queue_.front();

    size_t amt_to_write = msg->size() - message_send_bytes_written_;
    DCHECK(amt_to_write != 0);
    const char* out_bytes = reinterpret_cast<const char*>(msg->data()) +
        message_send_bytes_written_;

    struct msghdr msgh = {0};
    struct iovec iov = {const_cast<char*>(out_bytes), amt_to_write};
    msgh.msg_iov = &iov;
    msgh.msg_iovlen = 1;
    char buf[CMSG_SPACE(
        sizeof(int[FileDescriptorSet::MAX_DESCRIPTORS_PER_MESSAGE]))];

    ssize_t bytes_written = 1;
    int fd_written = -1;

    if (message_send_bytes_written_ == 0 &&
        !msg->file_descriptor_set()->empty()) {
      // This is the first chunk of a message which has descriptors to send
      struct cmsghdr *cmsg;
      const unsigned num_fds = msg->file_descriptor_set()->size();
```

<sys/socket.h>

man socket.h

<sys/socket.h>

man cmsg

```
DCHECK_LE(num_fds, FileDescriptorSet::MAX_DESCRIPTORS_PER_MESSAGE);
if (msg->file_descriptor_set()->ContainsDirectoryDescriptor()) {
  LOG(FATAL) << "Panic: attempting to transport directory descriptor over"
                " IPC. Aborting to maintain sandbox isolation.";
  // If you have hit this then something tried to send a file descriptor
  // to a directory over an IPC channel. Since IPC channels span
  // sandboxes this is very bad: the receiving process can use openat
  // with ".." elements in the path in order to reach the real
  // filesystem.
}

msgh.msg_control = buf;
msgh.msg_controllen = CMSG_SPACE(sizeof(int) * num_fds);
cmsg = CMSG_FIRSTHDR(&msgh);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SCM_RIGHTS;
cmsg->cmsg_len = CMSG_LEN(sizeof(int) * num_fds);
msg->file_descriptor_set()->GetDescriptors(
    reinterpret_cast<int*>(CMSG_DATA(cmsg)));
msgh.msg_controllen = cmsg->cmsg_len;

// DCHECK_LE above already checks that
// num_fds < MAX_DESCRIPTORS_PER_MESSAGE so no danger of overflow.
msg->header()->num_fds = static_cast<uint16>(num_fds);
```

```
    fd_written = pipe_;
    bytes_written = HANDLE_EINTR(sendmsg(pipe_, &msgh, MSG_DONTWAIT));
    if (bytes_written > 0)
      msg->file_descriptor_set()->CommitAll();

    if (bytes_written < 0 && !SocketWriteErrorIsRecoverable()) {
#if defined(OS_MACOSX)
      // On OSX writing to a pipe with no listener returns EPERM.
      if (errno == EPERM) {
        Close();
        return false;
      }
#endif  // OS_MACOSX
      if (errno == EPIPE) {
        Close();
        return false;
      }
      PLOG(ERROR) << "pipe error on "
                  << fd_written
                  << " Currently writing message of size: "
                  << msg->size();
      return false;
    }
```

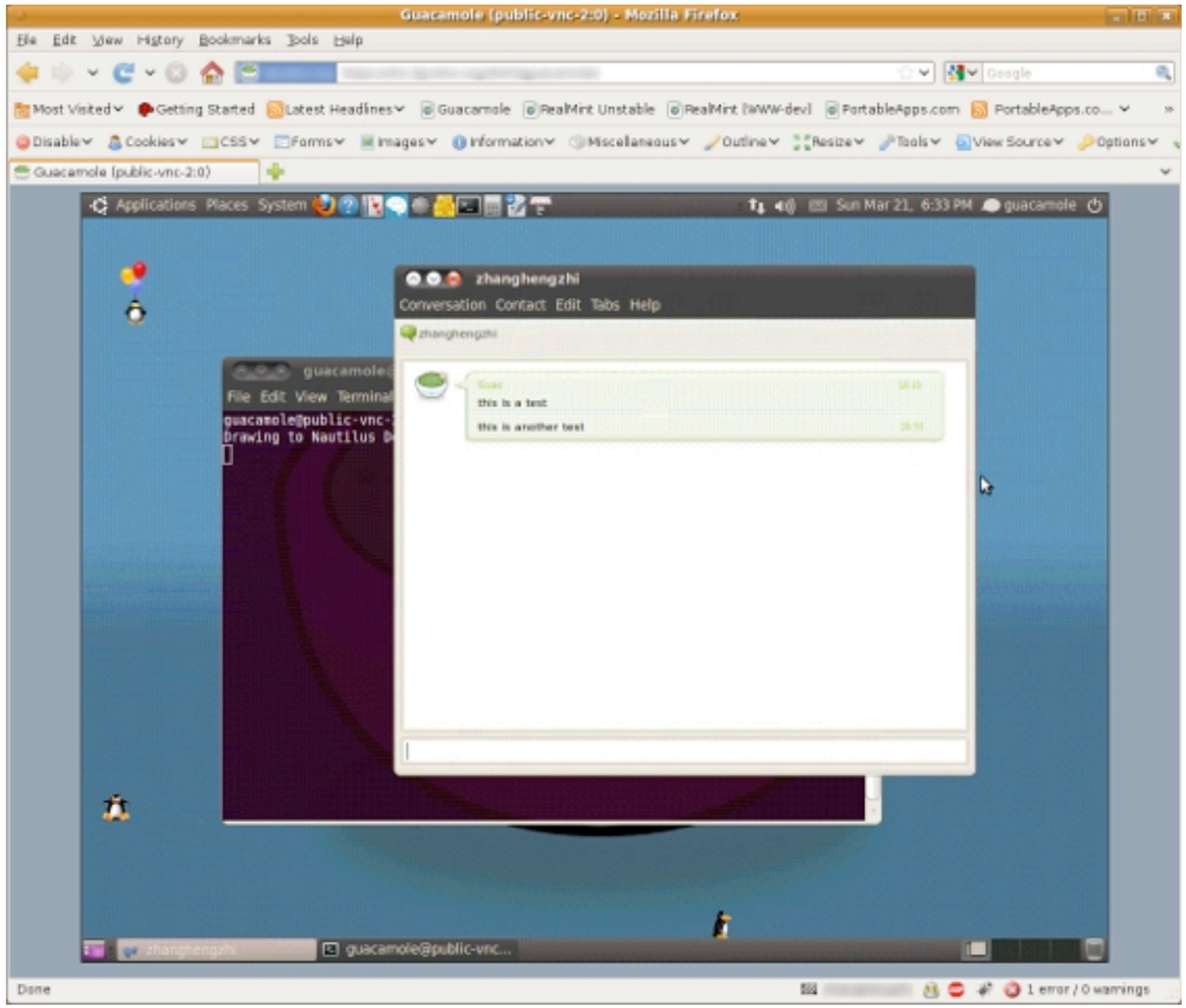
```
if (static_cast<size_t>(bytes_written) != amt_to_write) {
  if (bytes_written > 0) {
    // If write() fails with EAGAIN then bytes_written will be -1.
    message_send_bytes_written_ += bytes_written;
  }

  // Tell libevent to call us back once things are unblocked.
  is_blocked_on_write_ = true;
  MessageLoopForIO::current()->WatchFileDescriptor(
      pipe_,
      false,  // One shot
      MessageLoopForIO::WATCH_WRITE,
      &write_watcher_,
      this);
  return true;
    } else {
      message_send_bytes_written_ = 0;

      // Message sent OK!
      DVLOG(2) << "sent message @" << msg << " on channel @" << this
               << " with type " << msg->type() << " on fd " << pipe_;
      delete output_queue_.front();
      output_queue_.pop();
    }
  }
  return true;
}
```

to już było ...

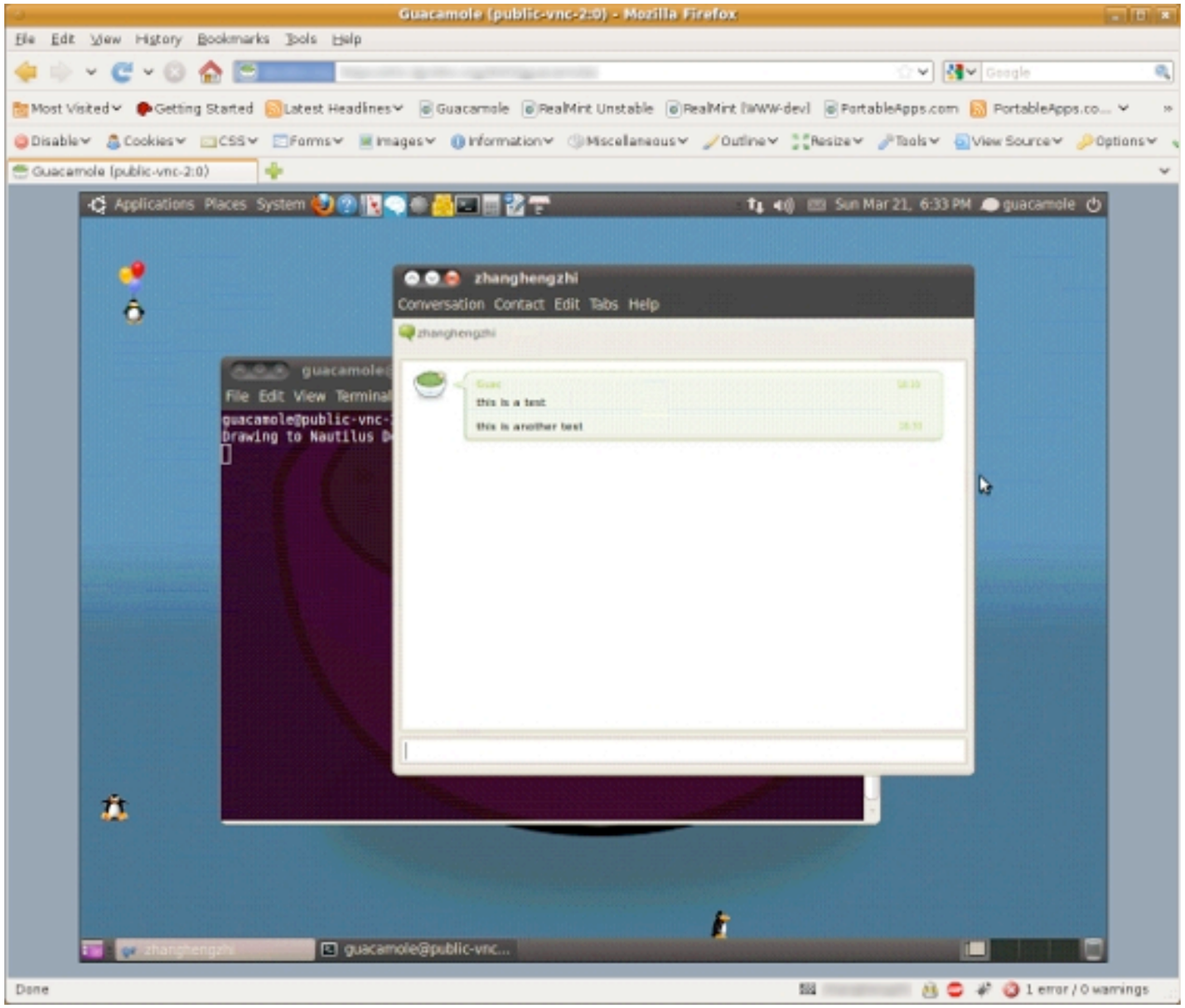
```
void DatabaseMessageFilter::OnDatabaseOpenFile(const string16& vfs_file_name,
                                               int desired_flags,
                                               IPC::Message* reply_msg) {
  DCHECK(BrowserThread::CurrentlyOn(BrowserThread::FILE));
  base::PlatformFile file_handle = base::kInvalidPlatformFileValue;
  base::PlatformFile target_handle = base::kInvalidPlatformFileValue;
  string16 origin_identifier;
  string16 database_name;

  // When in incognito mode, we want to make sure that all DB files are
  // removed when the incognito profile goes away, so we add the
  // SQLITE_OPEN_DELETEONCLOSE flag when opening all files, and keep
  // open handles to them in the database tracker to make sure they're
  // around for as long as needed.
  if (vfs_file_name.empty()) {
    VfsBackend::OpenTempFileInDirectory(db_tracker_->DatabaseDirectory(),
                                        desired_flags, &file_handle);
  } else if (DatabaseUtil::CrackVfsFileName(vfs_file_name, &origin_identifier,
                                            &database_name, NULL) &&
             !db_tracker_->IsDatabaseScheduledForDeletion(origin_identifier,
                                                          database_name)) {
    FilePath db_file =
        DatabaseUtil::GetFullFilePathForVfsFile(db_tracker_, vfs_file_name);
    if (!db_file.empty()) {
      if (db_tracker_->IsIncognitoProfile()) {
        db_tracker_->GetIncognitoFileHandle(vfs_file_name, &file_handle);
        if (file_handle == base::kInvalidPlatformFileValue) {
          VfsBackend::OpenFile(db_file,
                               desired_flags | SQLITE_OPEN_DELETEONCLOSE,
                               &file_handle);
          if (VfsBackend::FileTypeIsMainDB(desired_flags) ||
              VfsBackend::FileTypeIsJournal(desired_flags))
            db_tracker_->SaveIncognitoFileHandle(vfs_file_name, file_handle);
        }
      } else {
        VfsBackend::OpenFile(db_file, desired_flags, &file_handle);
      }
    }
  }
}
```

```
// static
void VfsBackend::OpenFile(const FilePath& file_path,
                          int desired_flags,
                          base::PlatformFile* file_handle) {
  DCHECK(!file_path.empty());

  // Verify the flags for consistency and create the database
  // directory if it doesn't exist.
  if (!OpenFileFlagsAreConsistent(desired_flags) ||
      !file_util::CreateDirectory(file_path.DirName()))
    return;

  int flags = 0;
  flags |= base::PLATFORM_FILE_READ;
  if (desired_flags & SQLITE_OPEN_READWRITE)
    flags |= base::PLATFORM_FILE_WRITE;

  if (!(desired_flags & SQLITE_OPEN_MAIN_DB)) {
    flags |= base::PLATFORM_FILE_EXCLUSIVE_READ |
             base::PLATFORM_FILE_EXCLUSIVE_WRITE;
  }

  flags |= ((desired_flags & SQLITE_OPEN_CREATE) ?
      base::PLATFORM_FILE_OPEN_ALWAYS : base::PLATFORM_FILE_OPEN);

  if (desired_flags & SQLITE_OPEN_EXCLUSIVE) {
    flags |= base::PLATFORM_FILE_EXCLUSIVE_READ |
             base::PLATFORM_FILE_EXCLUSIVE_WRITE;
  }

  if (desired_flags & SQLITE_OPEN_DELETEONCLOSE) {
    flags |= base::PLATFORM_FILE_TEMPORARY | base::PLATFORM_FILE_HIDDEN |
             base::PLATFORM_FILE_DELETE_ON_CLOSE;
  }

  // Try to open/create the DB file.
  *file_handle =
      base::CreatePlatformFile(file_path, flags, NULL, NULL);
}
```

Canvas + Web Sockets = ?



guacamole.sf.net



sos na bazie awokado

guacamole.sf.net

noVNC is a full VNC client implemented using HTML5 Canvas and WebSockets. You will need to either use a VNC server with WebSockets support or use websockify to proxy the raw TCP socket data to the WebSockets API of your browser. See the noVNC README and website for more information.

Host: Port: Password: Connect

noVNC ready: native WebSockets, createImageData rendering Settings Send CtrlAltDel

Klient	Obciążenie sieci
Microsoft Remote Desktop	225 kbps
UltraVNC	512 kbps
Microsoft SharedView	729 kbps

Klient	Obciążenie sieci
	A
Microsoft Remote Desktop	225 kbps
	B
UltraVNC	512 kbps
	C
Microsoft SharedView	729 kbps
	D

Klient	Obciążenie sieci
Microsoft Remote Desktop	225 kbps
WebNC	280 kbps
UltraVNC	512 kbps
Microsoft SharedView	729 kbps

Microdata

Microdata

które naprawdę działa!

Drooling Dog Bar B Q - Colfax, CA

★★★★☆ 15 reviews - Price range: $$

Drooling Dog has some really good **BBQ**. I had the pulled pork sandwich, **.... Drooling Dog BBQ** is a great place to stop at on your way up the hill to Tahoe **...**

www.yelp.com/biz/**drooling-dog**-bar-b-q-colfax - 75k - Cached - Similar pages

```
<div class="hreview-aggregate">
  <div class="item vcard">
    <h1 class="fn org">Drooling Dog Bar B Q</h1>
    ...
    <img class="stars_4 rating average" src="stars_map.png" alt="4 star rating" />
    <em>based on <span class="count">15</span> reviews</em>
    ...
    <strong>Price range:</strong> <span class="pricerange">$$</span>
  </div>
</div>
```

Rich Snippets Testing Tool Beta

Rich Snippets allows you to enhance your Google search results by marking up web pages with Microformats, RDFa or Microdata.

Test your website

Enter a web page URL to see how it may appear in search results:

http://www.urbanspoon.com/r/6/765421/restaurant/Pizza-My-Heart-S Preview

Examples: Urbanspoon, LinkedIn

Google search preview

Pizza My Heart - Santa Cruz | Urbanspoon

★★★★★ 10 reviews - Price range: Under $10 per entree

Excerpt from the page will show up here. Excerpt from the page will show up here.
Excerpt from the page will show up here. Excerpt from the page will show up here.

www.urbanspoon.com/r/6/765421/restaurant/Pizza-My-Heart-Santa-Cruz - Cached - Similar pages

Note that there is no guarantee that a Rich Snippet will be shown for this page on actual search results. For more details, see the FAQ.

Extracted Rich Snippet data from the page

```
hreview-aggregate
   item hcard
      fn = Pizza My Heart
      org
         organization-name = Pizza My Heart
      adr
         street-address = 1116 Pacific Ave Ste B
         locality = Santa Cruz
         region = CA
         postal-code = 95060
      tel
         value = (831) 426-2511
      url = http://www.pizzamyheart.com/
   rating
```

Adoption: pages with markup

One million web pages sampled from the Internet

~ 4%

	Microformats	RDFa
Total pages	40,091	2,514
hCard / People	33,675 (13%)	1,160
Reviews	1,950 (88%)	872 (66%)
Recipe	152 (53%)	--
hCalendar / Event	126 (41%)	--
Products	519	77

Data from June 2010. Percentages: actually used data.

Microdata

które naprawdę działa - od 2009!

Web Workers

```html
<!DOCTYPE HTML>
<html>
 <head>
  <title>Worker example: One-core computation</title>
 </head>
 <body>
  <p>The highest prime number discovered so far is: <output id="result"></output></p>
  <script>
   var worker = new Worker('worker.js');
   worker.onmessage = function (event) {
     document.getElementById('result').textContent = event.data;
   };
  </script>
 </body>
</html>
```

```
var n = 1;
search: while (true) {
  n += 1;
  for (var i = 2; i <= Math.sqrt(n); i += 1)
    if (n % i == 0)
      continue search;
  // found a prime!
  postMessage(n);
}
```

SPDY

Table 1: Average page load times for top 25 websites

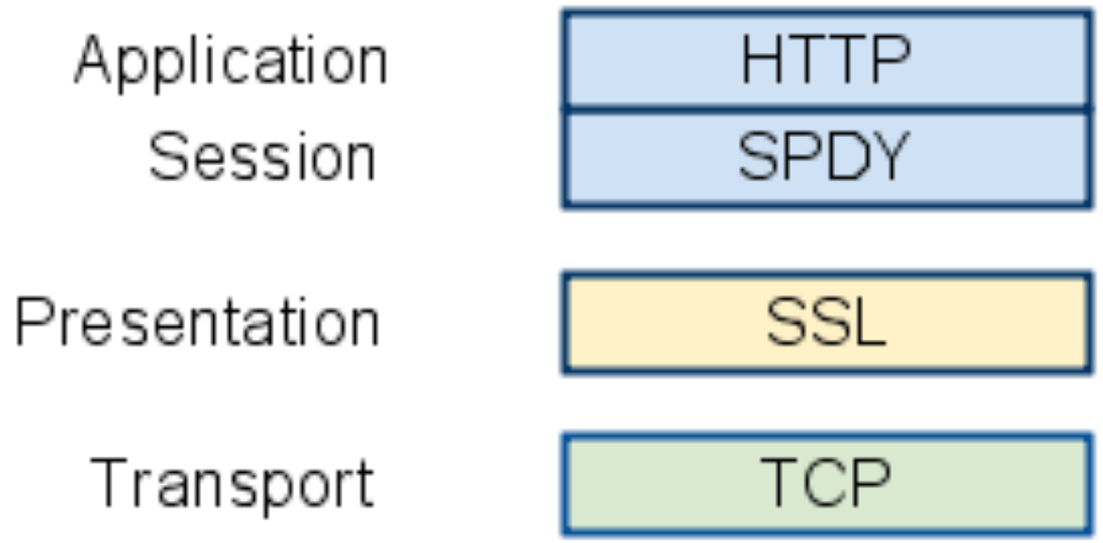
	DSL 2 Mbps downlink, 375 kbps uplink		Cable 4 Mbps downlink, 1 Mbps uplink	
	Average ms	**Speedup**	**Average ms**	**Speedup**
HTTP	3111.916		2348.188	
SPDY basic multi-domain* connection / TCP	2242.756	27.93%	1325.46	43.55%
SPDY basic single-domain* connection / TCP	1695.72	45.51%	933.836	60.23%
SPDY single-domain + server push / TCP	1671.28	46.29%	950.764	59.51%
SPDY single-domain + server hint / TCP	1608.928	48.30%	856.356	63.53%
SPDY basic single-domain / SSL	1899.744	38.95%	1099.444	53.18
SPDY single-domain + client prefetch / SSL	1781.864	42.74%	1047.308	55.40%

Table 2: Average page load times for top 25 websites by packet loss rate

	Average ms		Speedup
Packet loss rate	HTTP	SPDY basic (TCP)	
0%	1152	1016	11.81%
0.5%	1638	1105	32.54%
1%	2060	1200	41.75%
1.5%	2372	1394	41.23%
2%	2904	1537	47.7%
2.5%	3028	1707	43.63%

Table 3: Average page load times for top 25 websites by RTT

	Average ms		**Speedup**
RTT in ms	**HTTP**	**SPDY basic (TCP)**	
20	1240	1087	12.34%
40	1571	1279	18.59%
60	1909	1526	20.06%
80	2268	1727	23.85%
120	2927	2240	23.47%
160	3650	2772	24.05%
200	4498	3293	26.79%


Application
HTTP
Session
SPDY
Presentation
SSL
Transport
TCP

- wiele zasobów w jednym połączeniu TCP
- priorytety zasobów
- kompresja nagłówków (redukcja ~80%)
- wysyłanie inicjowane przez serwer

- implementacja w C++ (Chromium)
- implementacja w Pythonie
- implementacja w Javie (Tomcat)
- mod_spdy

HTML

pytania?