MalloryWorker

Stealthy Computation and Covert Channels using Web Workers

Michael Rushanan, David Russell, Aviel D. Rubin
Johns Hopkins University

Introduction

- The HTML5 standard:
  - Eliminates third-party plugins
  - Necessitates complex JavaScript (JS):
    - Computationally expensive
    - Long-running
- Long-running JS scripts terminated

Introduction

- Web Workers API
  - Spawn background workers in parallel to main page
- Encrypting medical images (e.g., x-rays)
  - Client side AES-GCM
- MacBook got loud and *OS X crashed*

Problem

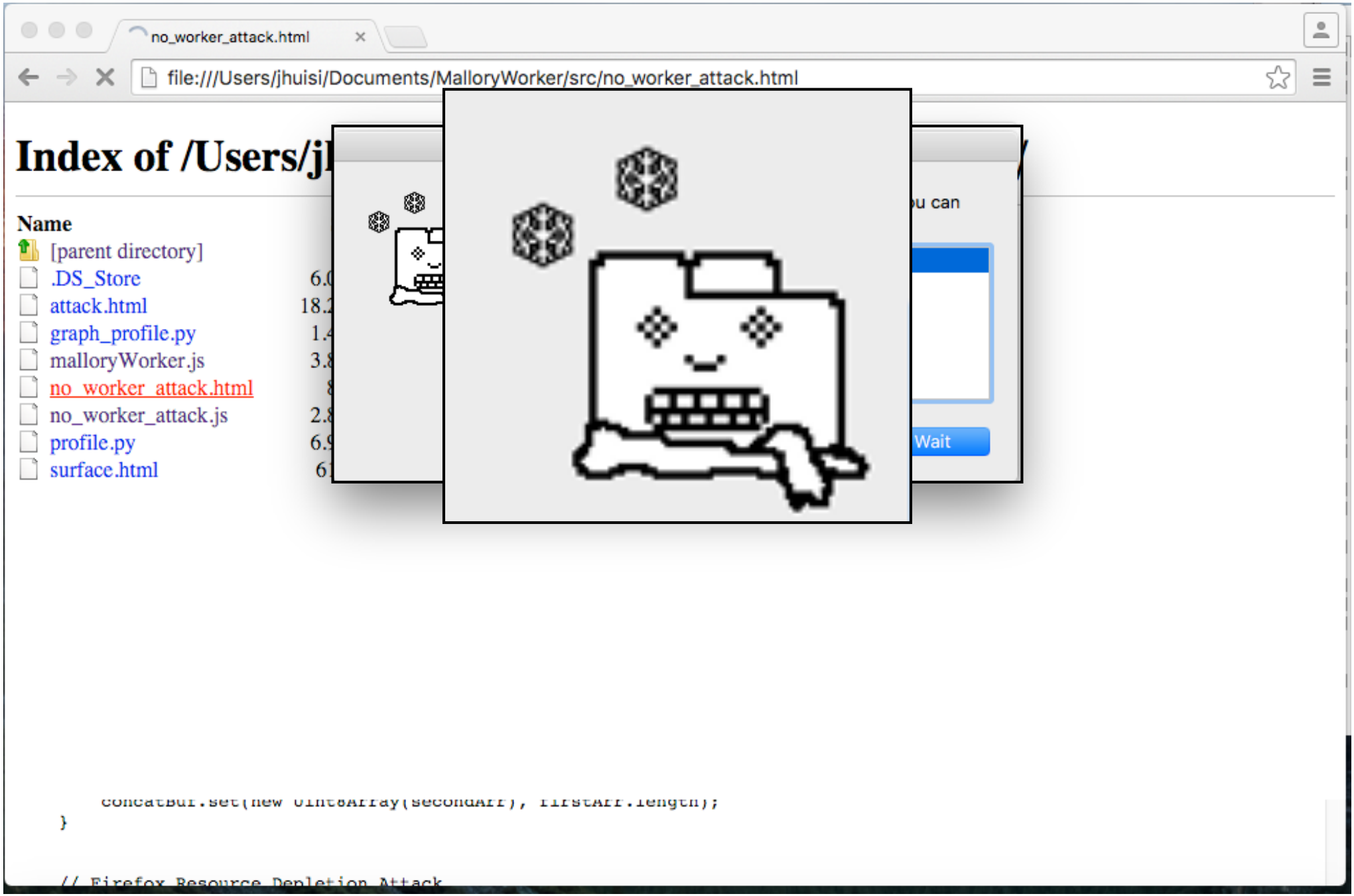
- *Infinite loop*
- Simple bug allowed excessive memory usage
- Workers instantiated unbeknownst to users
- Web applications perform *any* computation
  - Bitcoin mining
  - Password cracking

Questions

- How pervasive are Web Workers?
- Who is the Attacker?
  - What are her goals/capabilities?
    - What kind of computation can she achieve?
    - DoS? Resource depletion? Other?
- What are the defenses?

Background

- JavaScript (JS) and UI rendering share a thread
  - UI updates block JS execution
  - Shared task queue
  - Long-running JS scripts terminated
    - Little context; UI Freezes


no_worker_attack.html
file:///Users/jhuisi/Documents/MalloryWorker/src/no_worker_attack.html
Index of /Users/jl
Name
[parent directory]
.DS_Store
attack.html
graph_profile.py
malloryWorker.js
no_worker_attack.html
no_worker_attack.js
profile.py
surface.html
u can
Wait
}
// Firefox Resource Depletion Attack

Web Workers

- Web Workers API
  - Spawn background workers in parallel to main page
- Unable to access:
  - The Dynamic Object Model (DOM)
  - Callers (i.e., parent Object):
    - Variables and Functions

Web Workers

- Shared workers are accessed by multiple web applications
- Dedicated workers are accessed by one application
- Constructor takes a file or inline string description
- `onMessage()` listens for messages
- `postMessage()` sends a message
- `terminate()` stops worker and calls garbage collection

Web Workers

```
<script id="mw" type="javascript/worker">
    self.onmessage = function(event) {
    self.postMessage({’msg’: ’hello.’,}); }
</script>

<script language="javascript">
    var blob = new Blob([document.querySelector(’#mw’). textContext]);
    var m_worker = new Worker(window.URL.createObjectURL(blob)) ;

m_worker.onmessage = function(event) {
    console.log(event.data);
};

m_worker.postMessage({
    'name': ‘Mike’,
    ‘msg': ‘Hi’,
});
</script>
```

Pervasiveness

- Scanned 7000 webpages from Alexa's Top Sites
- 1.2% of webpages used Web Workers
- yahoo.com
  - rapidworker_1_2_0.0.3.js (base64 encoding and compression)
- usbank.com
  - foresee-worker.js (compress session event logs)

Pervasiveness

- Published in October 2014
- APIs available to Workers:
  - Cache: control cache storage with current origin
  - Channel Messaging: Separate contexts attached to the same context communicate
  - OffscreenCanvas
  - WebSocket

Threat Model

- Attacker malicious misuses workers
- Akhawe et al.'s [1] definition of attacker:
  - Web attacker
  - Gadget attacker

[1] Akhawe, D., Barth, A., Lam, P.E., Mitchell, J., Song, D.: Towards a formal foundation of web security. In: Proceedings of the 2010 23rd IEEE Computer Security Foundations Symposium. pp. 290–304. IEEE Computer Society (2010), http://dx.doi.org/10.1109/CSF.2010.27

Threat Model

- Web and gadget attacker goals:
  - **Stealthy computation**
    - Attacker controlled code execution
  - DoS
  - Resource depletion
  - Covert channel

Threat Model

- Web Attacker *[insider]*
  - Unsophisticated
    - No visibility or control into network
  - Operates malicious web application
    - E.g., social engineering page

Threat Model

- Gadget Attacker *[outsider]*
  - Unsophisticated
    - No visibility or control into network
  - Inject content into legitimate web applications
    - Purchase advertisement space
    - Cross-site Scripting (XSS)

Web Worker Primitives

- Stealthy computation is simple:
  - Write f(x), run f(x)
- Wasteful computation
  - Intensive operations
  - Abuse garbage collection

Wasteful Computation

- Three primitives:
  - Infinite loop sequence
  - CPU throttling
  - Memory throttling

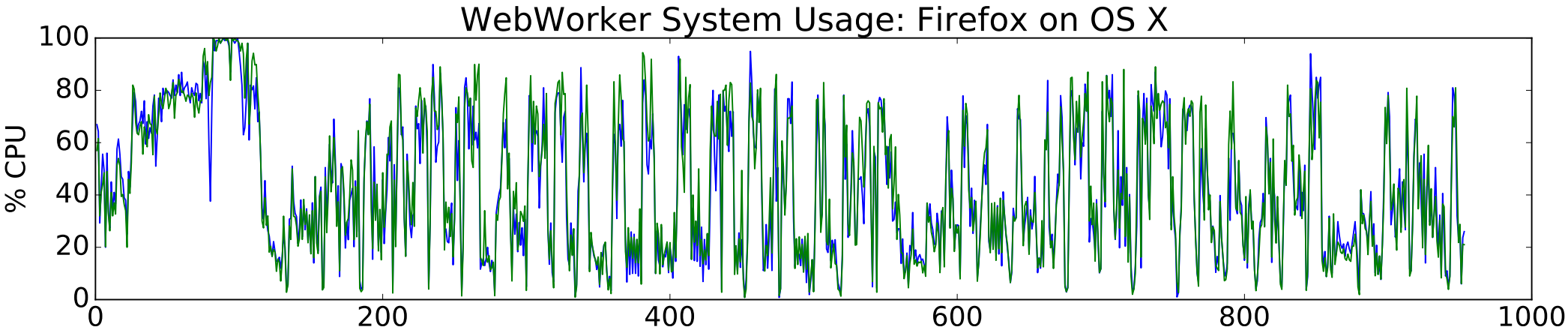
Infinite Loop Sequence

```
while(true) {
    // Loop forever.
}
```

CPU Throttling

```
var cpu_work = function ()
{
  var scratch = [];
  // Fill the ArrayBuffer with random values.
  for( var j = 0; j < 1024; j++ )
  {
   scratch.push(Math.random());
  }

  var firstArr = new Uint8Array(scratch);
  var secondArr = new Uint8Array(scratch);

  // ArrayBuffer concatenation.
  var concatBuf = new Uint8Array(firstArr.byteLength +
   secondArr.byteLength);

  concatBuf.set(new Uint8Array(firstArr), 0);
  concatBuf.set(new Uint8Array(secondArr), firstArr.length);
}
```

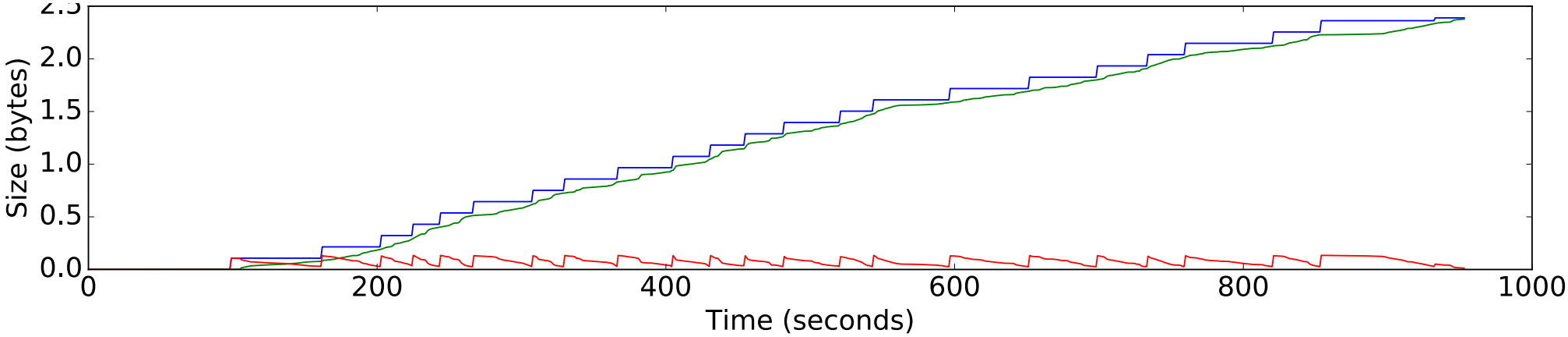
Quick Look


WebWorker System Usage: Firefox on OS X
% CPU
100
80
60
40
20
0
0
200
400
600
800
1000

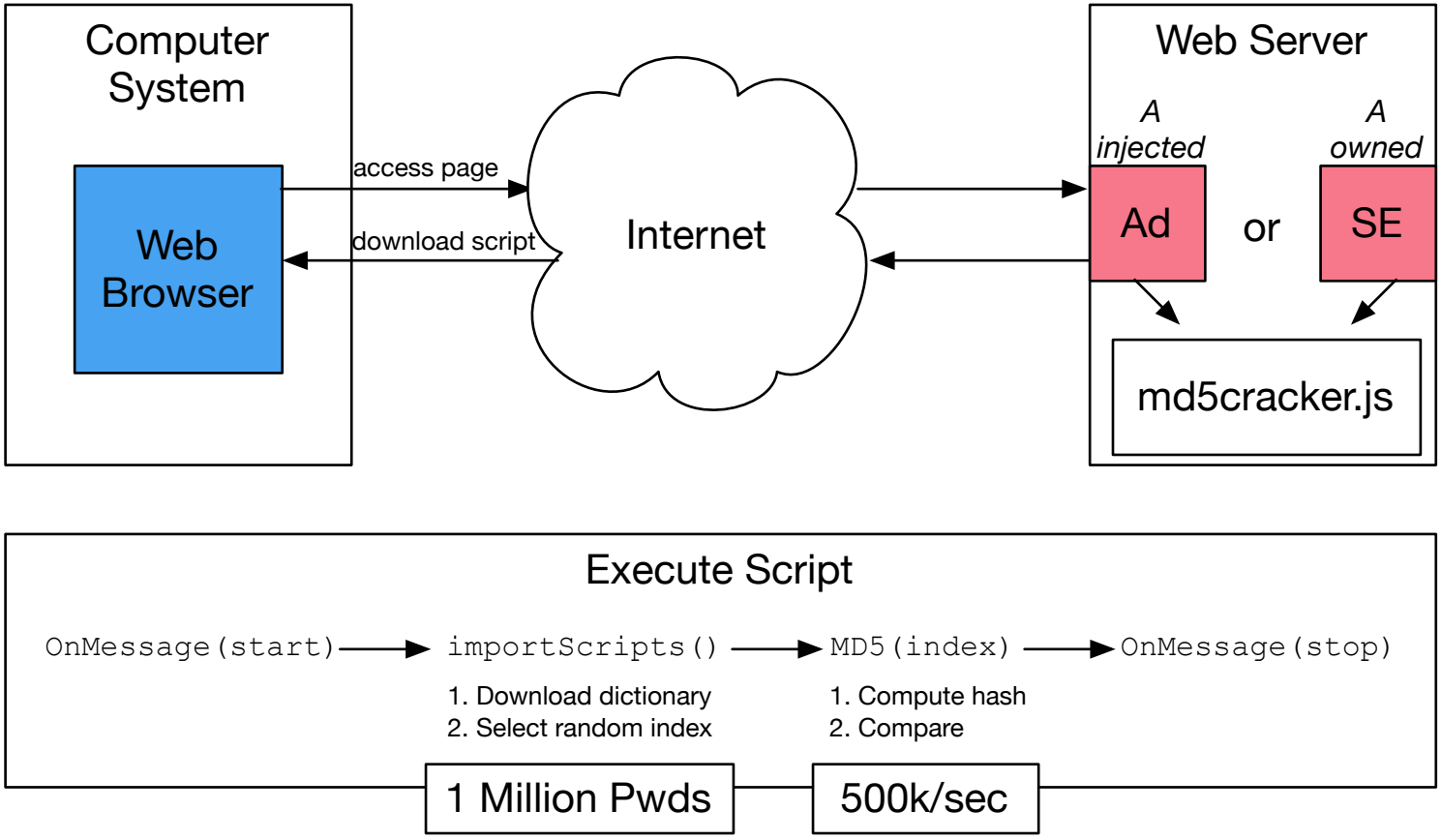
Memory Throttling

```
var bucket = null;

var RD_ATTACK_FIREFOX_SAFARI = function ()
{
   var leak = bucket;
   var unused = function ()
   {
    if (leak)
    {
       var hole_in_bucket = 1;
    }
   };

   bucket = {
    longStr: new Array(10000000).join(Math.random()), someMethod: function
() {
    var hole_in_bucket = 2; }
   };

   // Placeholder for doing some repetitive operation.
   cpu_work();
};
```

Quick Look



Password Cracker



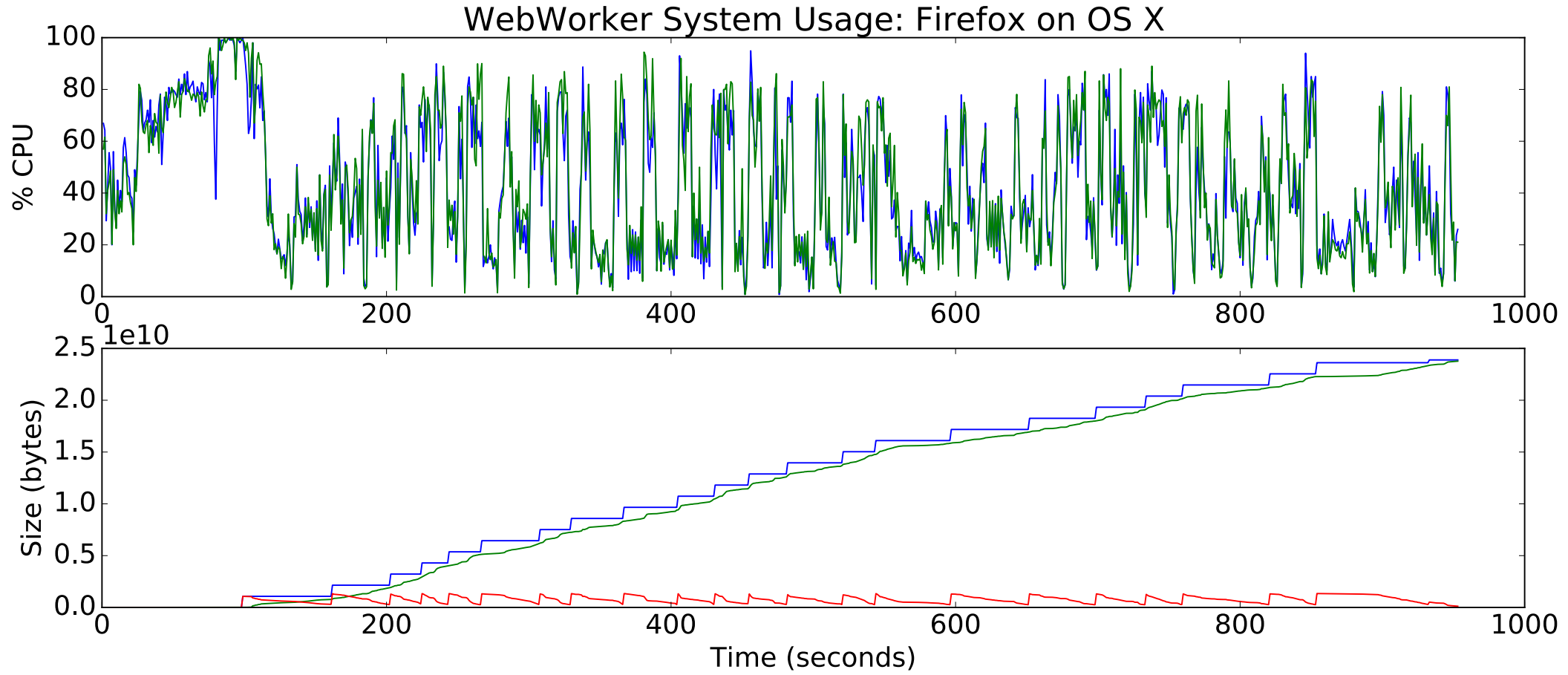
DoS Attack

- Works against any 64-bit OS X device
- Use loop and memory throttle primitives
- Unique to OS x because of virtual memory
  - Grow swap to maximum size of backing store
  - Portion of disk stores virtual memory pages
- Exploit garbage collection for an **extended period**

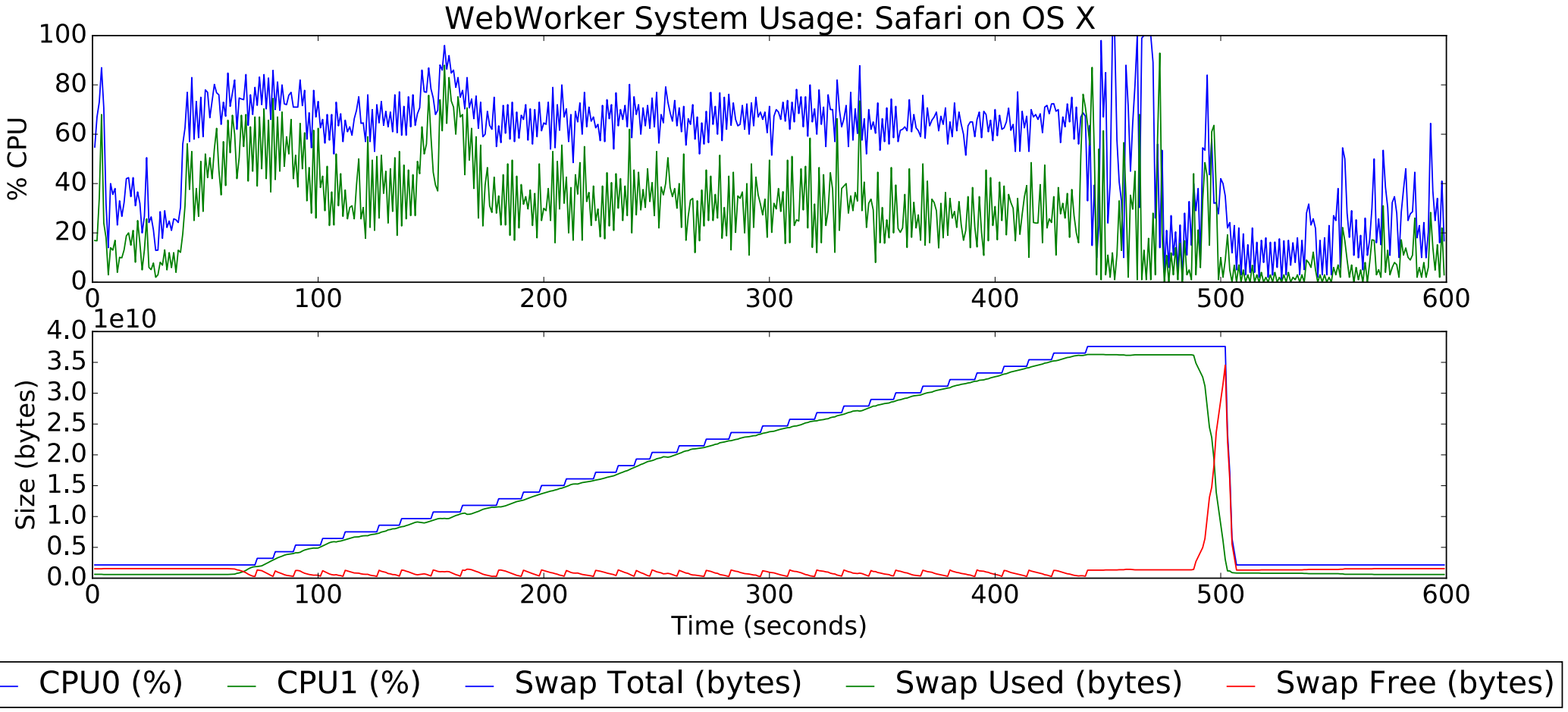
Experiment

- Mid-2013 MacBook Air
- 50 GB free hard disk space
- 10 MB was leaked each loop iteration
- Deadlock in 15 minutes

DoS Attack

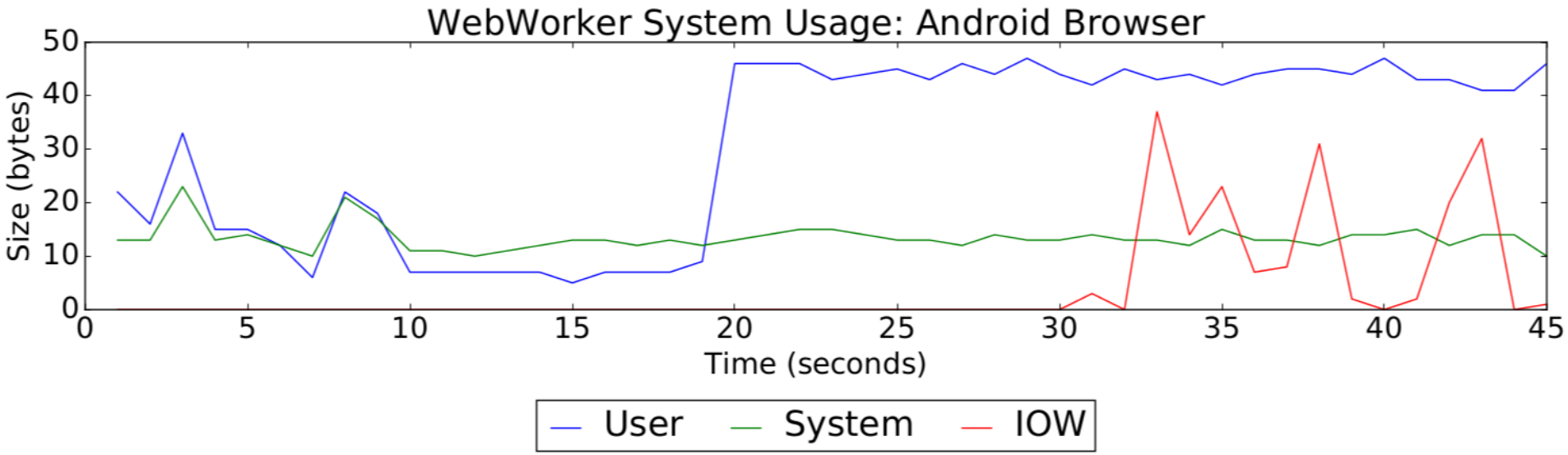

WebWorker System Usage: Firefox on OS X
% CPU
100
80
60
40
20
0
0
200
400
600
800
1000
1e10
2.5
2.0
1.5
1.0
0.5
0.0
Size (bytes)
0
200
400
600
800
1000
Time (seconds)

DoS Attack

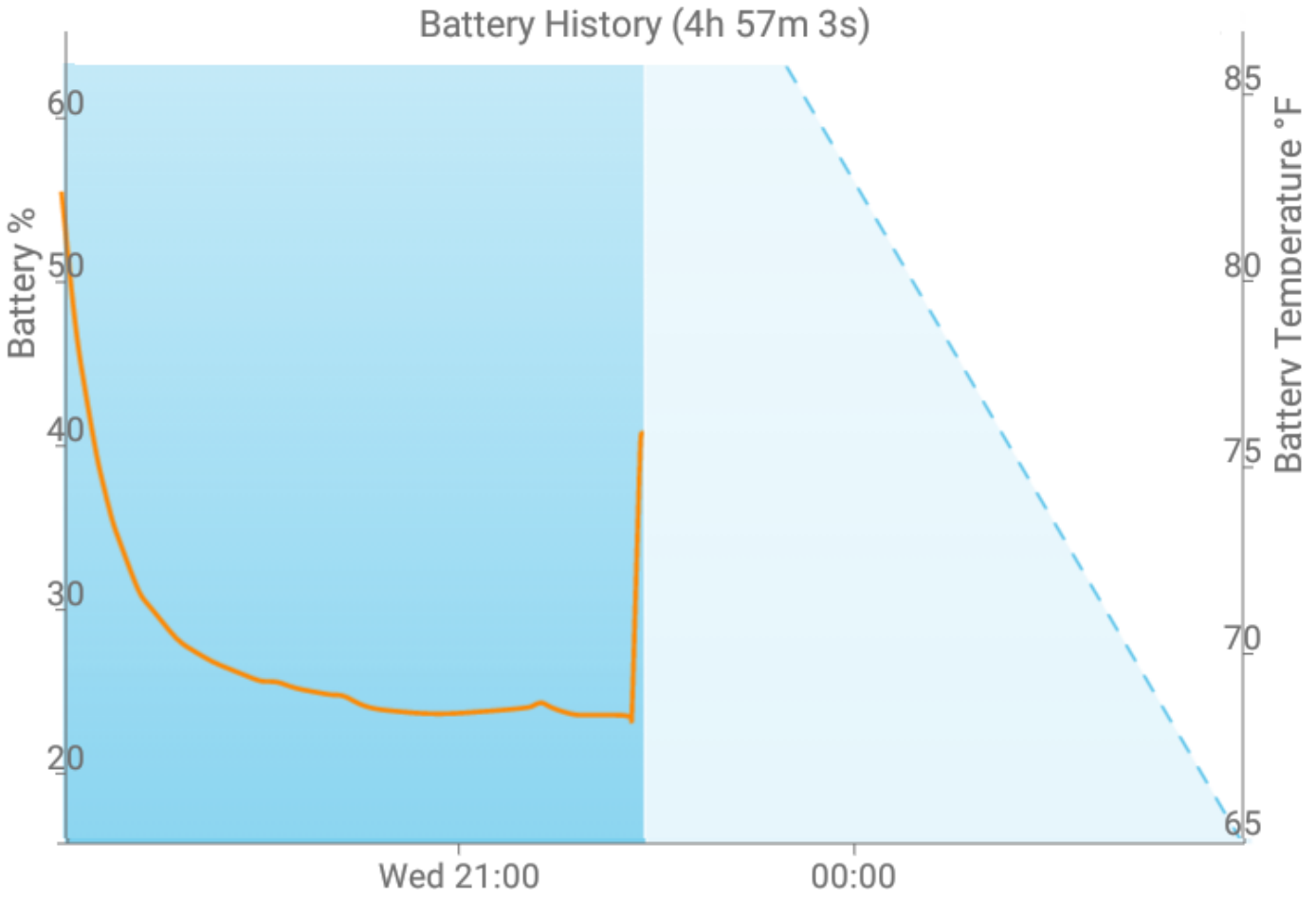

WebWorker System Usage: Safari on OS X
% CPU
100
80
60
40
20
0
0
100
200
300
400
500
600
1e10
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Size (bytes)
0
100
200
300
400
500
600
Time (seconds)
CPU0 (%)
CPU1 (%)
Swap Total (bytes)
Swap Used (bytes)
Swap Free (bytes)

Resource Depletion

- Mobile Chrome supports Web Workers API



Resource Depletion



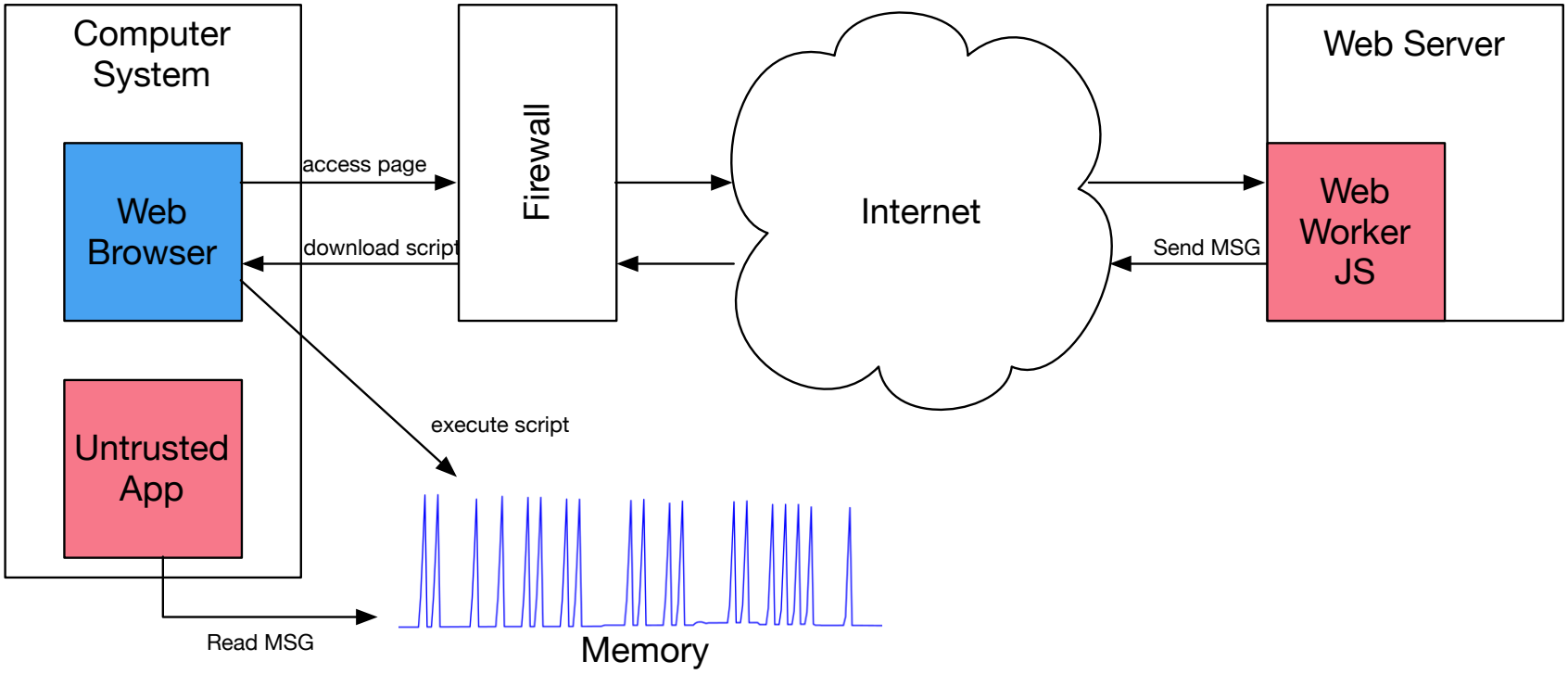
Covert Channel

- Communication mechanism between two processes that are:
  - Not supposed to
  - Not allowed to
- System resource usage can be a channel
- Timing channels (i.e., modulate system resources)

Covert Channel

- Web Application <-> Untrusted PC Application
- CPU and memory throttling primitives
- Unprivileged operation to monitor both
  - Try it using python!
- Firewalls and anti-virus won’t catch it
- Not blacklisted or anomalous

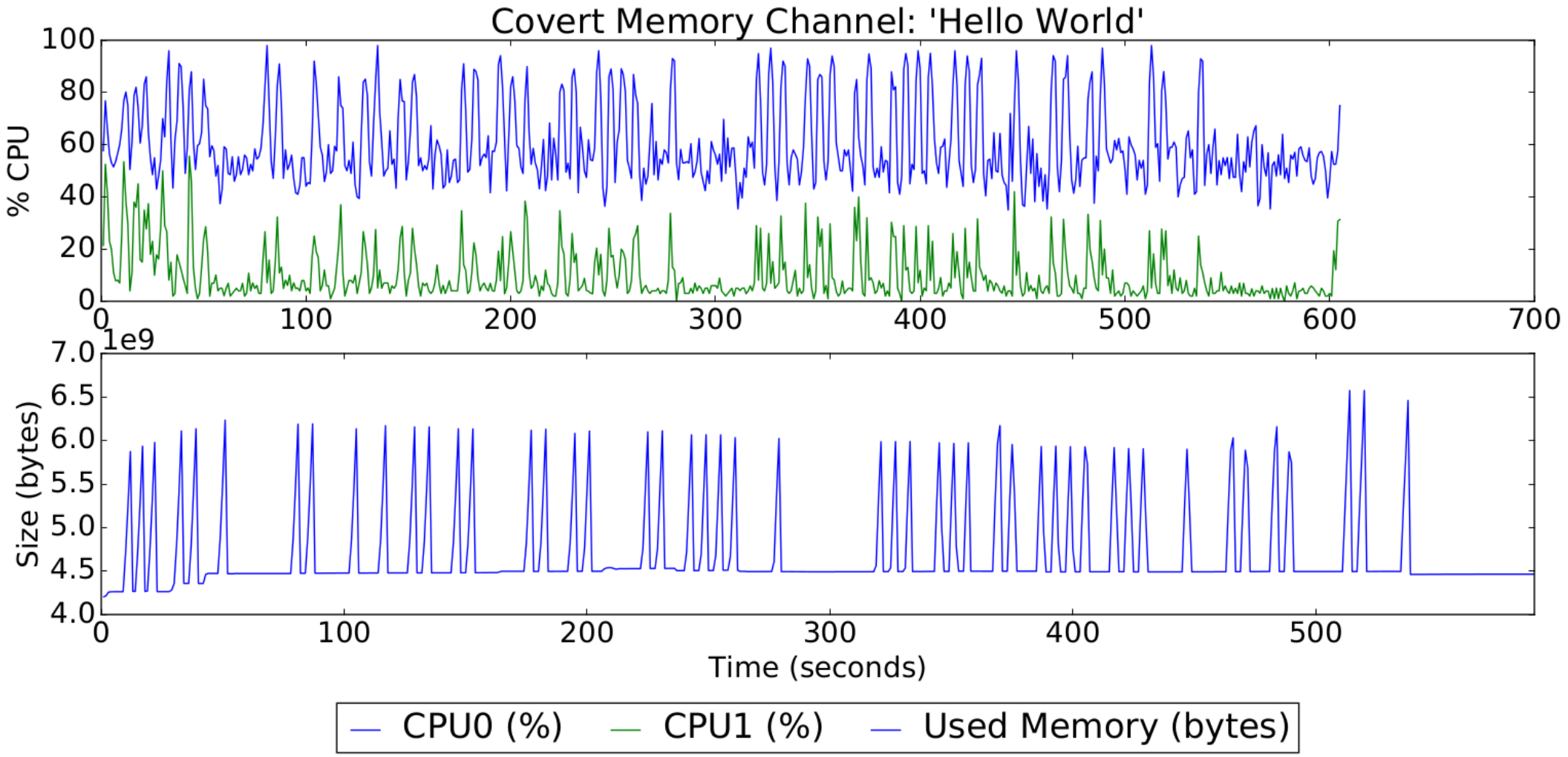
Covert Channel



Memory Channel

- Memory is more determinist than CPU
  - CPU was too noisy
  - Usable when streaming video
- Send 1 bit per 5 seconds:
  - Fill 40MB array
  - Clear memory with a call to `terminate`

Memory Channel



Criticism

- Does not require web worker
  - True, but the browser is noticeable slow
  - Can not use `terminate`
- Implement using setTimeout — fails
- Bit rate is a proof-of-concept

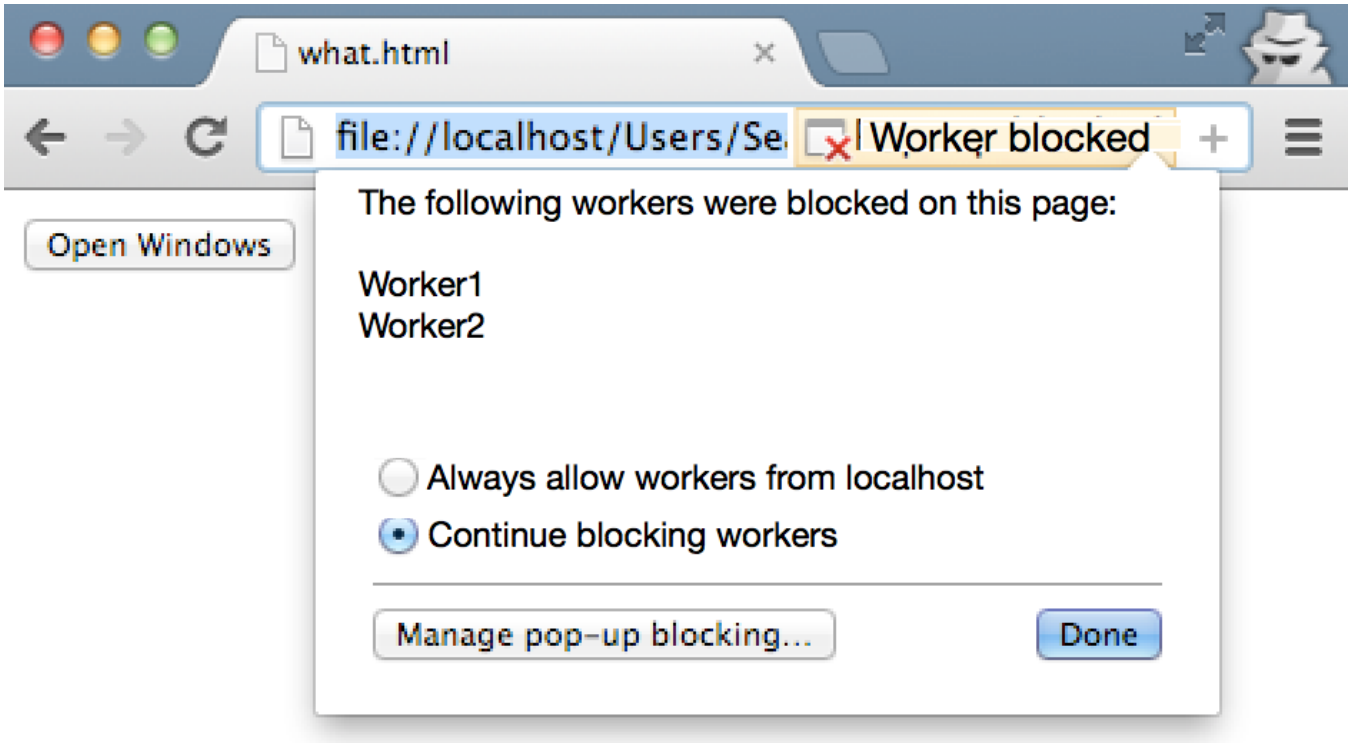
Potential Mitigations

- Modify Web Workers API:
  - Inform users a worker is executing
  - Intent of execution
  - Execution impacting on resources

Potential Mitigations

- Dialog box:
  - Do not allow any site to execute computationally intensive scripts
  - Inform me when any site executes computationally intensive scripts (Recommended)
  - Allow all sites to execute computationally intensive scripts

Dialog Box


what.html
file://localhost/Users/Se
Worker blocked
Open Windows
The following workers were blocked on this page:
Worker1
Worker2
Always allow workers from localhost
Continue blocking workers
Manage pop-up blocking...
Done

wAudit

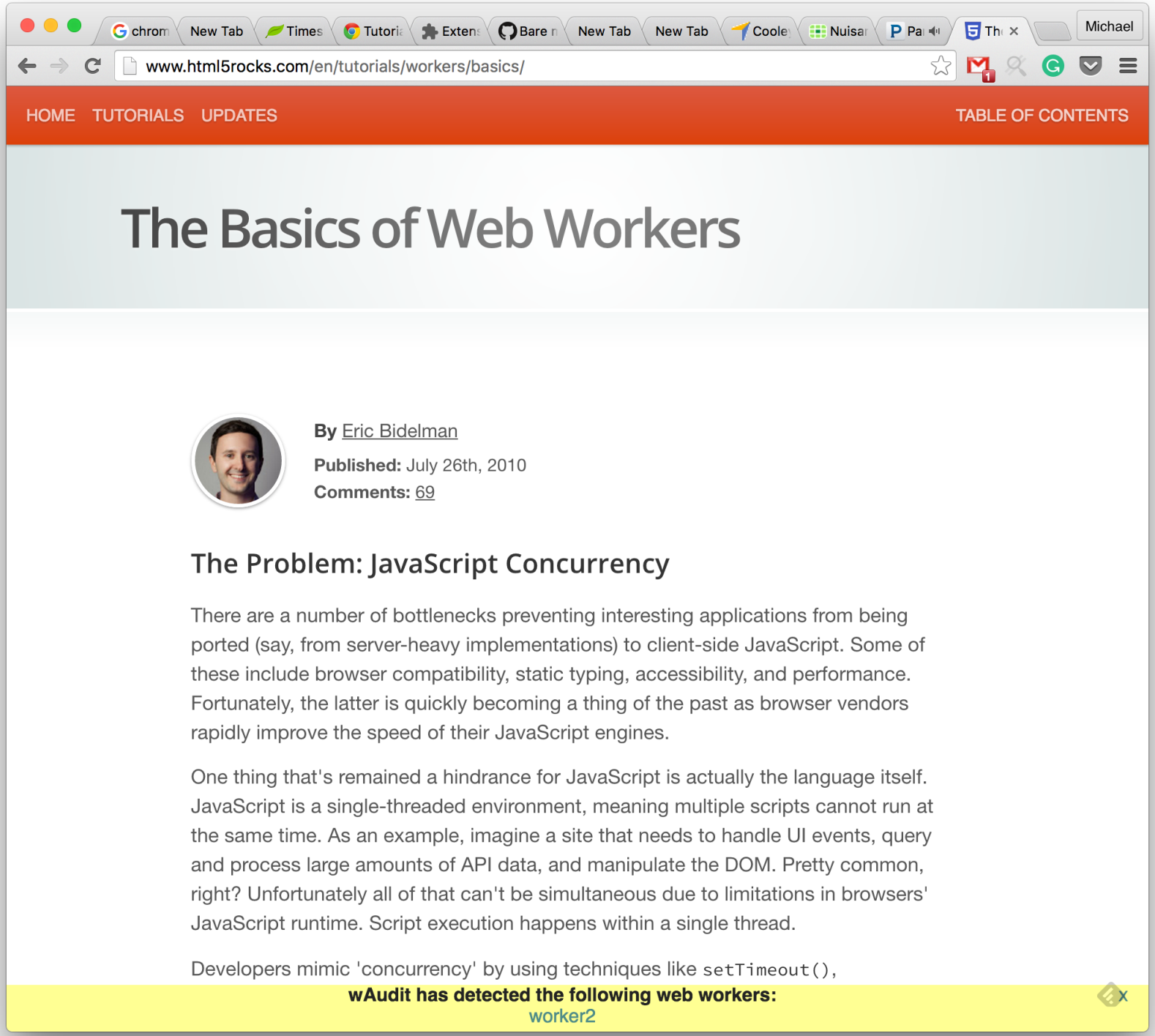
- wAudit is a browser extension
- Inject wAudit as a script into visited page
- Creates HTML script element appended to document object's root element
- Injected script recursively searches DOM objects to identify [object Worker]
- Alert the user and allow termination

wAudit

```
function getChildren(object) {
    var keys = Object.keys(object);
    keys = keys.slice(6,keys.length-1);
    keys.forEach(function(key) {
                var value = object[key];
                if(value && typeof value === 'object') {
                if(_objects.indexOf(value) < 0) {
                _objects.push(value);
                _names.push(key);
                getChildren(value);
```

```
getChildren(this);

for(var i=0;i<_objects.length;i++) {
    try {
        if(_objects[i].toLocaleString() === '[object Worker]') {
```

```
var text = "<center><strong>wAudit has detected the following web workers:</strong><p
style='float:right'><sup><a onclick='document.getElementbyID(\'worker_info
\').style.display=none;'>X</sup>    </p><br/>";
for(var i=0;i<_workers.length;i++) {
    evalstring = _workers[i]+".terminate();alert('Terminated');";
    text += "<a onclick="+ evalstring +">"+
    _workers[i] +"</a>";
```


Michael
www.html5rocks.com/en/tutorials/workers/basics/
HOME TUTORIALS UPDATES
TABLE OF CONTENTS
The Basics of Web Workers
By Eric Bidelman
Published: July 26th, 2010
Comments: 69
The Problem: JavaScript Concurrency
There are a number of bottlenecks preventing interesting applications from being ported (say, from server-heavy implementations) to client-side JavaScript. Some of these include browser compatibility, static typing, accessibility, and performance. Fortunately, the latter is quickly becoming a thing of the past as browser vendors rapidly improve the speed of their JavaScript engines.
One thing that's remained a hindrance for JavaScript is actually the language itself. JavaScript is a single-threaded environment, meaning multiple scripts cannot run at the same time. As an example, imagine a site that needs to handle UI events, query and process large amounts of API data, and manipulate the DOM. Pretty common, right? Unfortunately all of that can't be simultaneous due to limitations in browsers' JavaScript runtime. Script execution happens within a single thread.
Developers mimic 'concurrency' by using techniques like setTimeout(),
wAudit has detected the following web workers:
worker2

Questions

- Thank you for attending my talk!

Backup Slides

Related Work

- What other stealthy computations exist?
  - Anibal Sacco et al. use workers to optimize heap-spray attacks
  - Kuppan describes HTML5 botnets