JavaScript Pitfalls

Ming Chow (mchow@cs.tufts.edu)

SOURCE Boston 2013

Overview

- JavaScript is now a first-class citizen and it has never been more important
- What has pushed JavaScript to being a first-class citizen: client-side web applications, AJAX, jQuery, mobile, V8 engine, more powerful web browsers, HTML5, server-side development
- More powers, more potentials for abuse (and to be misunderstood)

Scope of Presentation

- JavaScript in:
  - HTML5 APIs
  - Node.js
  - Database systems such as MongoDB

Review: Same-Origin Policy

- Restricts how a document or script loaded from one origin can interact with a resource from another origin
- Same-origin = same protocol, same port, and same host
- Documentation: https://developer.mozilla.org/en-US/docs/JavaScript/Same_origin_policy_for_JavaScript
- Bypass traditional same-origin policy by:
  - `<script src="..."></script>`
  - `<link rel="stylesheet" href="...">`
  - `<img… />`

Cross-Origin JavaScript Requests (or Cross-Origin Resource Sharing)

- We are slowing moving away from the same-domain policy
- Not directly part of HTML5 but introduced by W3C
- `XDomainRequest()` created by Microsoft in Internet Explorer 8
- `XMLHttpRequest()` is starting to allow cross-domain requests (Firefox 3.5+ and Safari 4+)
- Caveat: consent between web page and the server is required.

Cross Origin Resource Sharing (CORS)

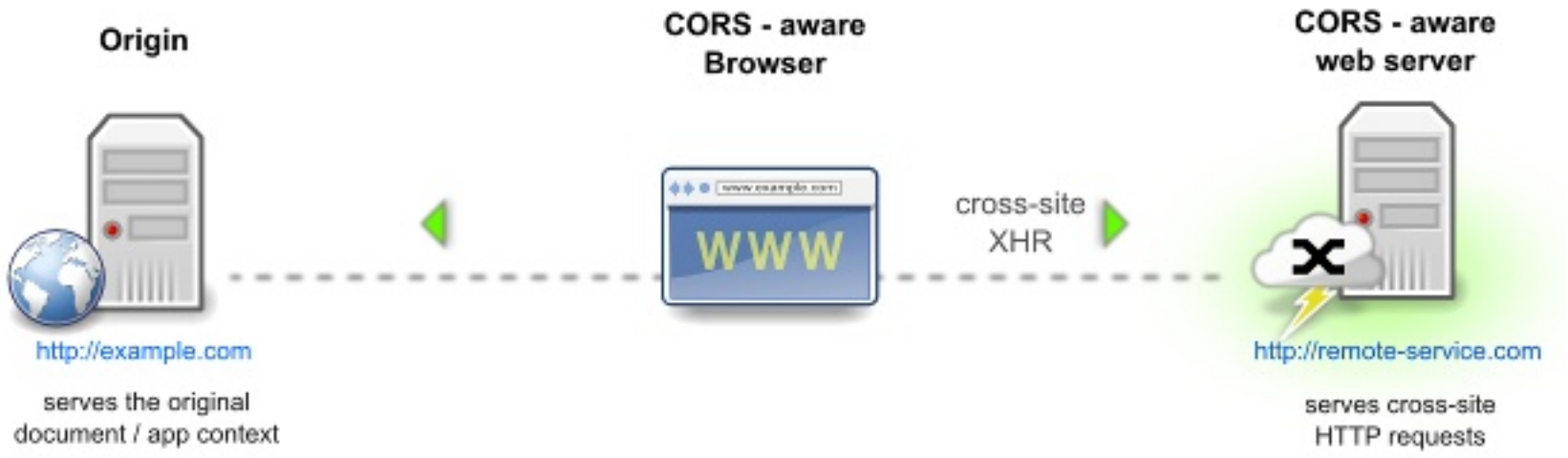

Image source: http://software.dzhuvinov.com/cors-filter.html

CORS (continued)

- Server must respond with an `Access-Control-Allow-Origin` header of either `*` (a.k.a., universal allow) or the exact URL of the requesting page (site-level; white-list)
  - Example 1 (BAD!): `header('Access-Control-Allow-Origin: *');`
  - Example 2 (BAD!): `Access-Control-Allow-Origin: http://allowed.origin/page?cors=other.allowed.origin%20malicious.spoof`
- Resolutions:
  - Validate response
  - Add some form of authentication / credentials checking via `withCredentials` property on `XMLHttpRequest` (e.g., cookie)

Web GL

- Impressive technology on the web browser
  - http://chrome.angrybirds.com/
  - http://alteredqualia.com/three/examples/webgl_cars.html
- Issue is simple: WebGL requires browser having direct access to graphics hardware
- Known exploits due to bugs in WebGL implementation:
  - Cross-domain image stealing (from the memory of the graphics hardware)
  - Denial of Service
- Resolution: disable Web GL on your web browser (e.g., in Chrome, run `chrome://flags/` and go to "Disable WebGL")

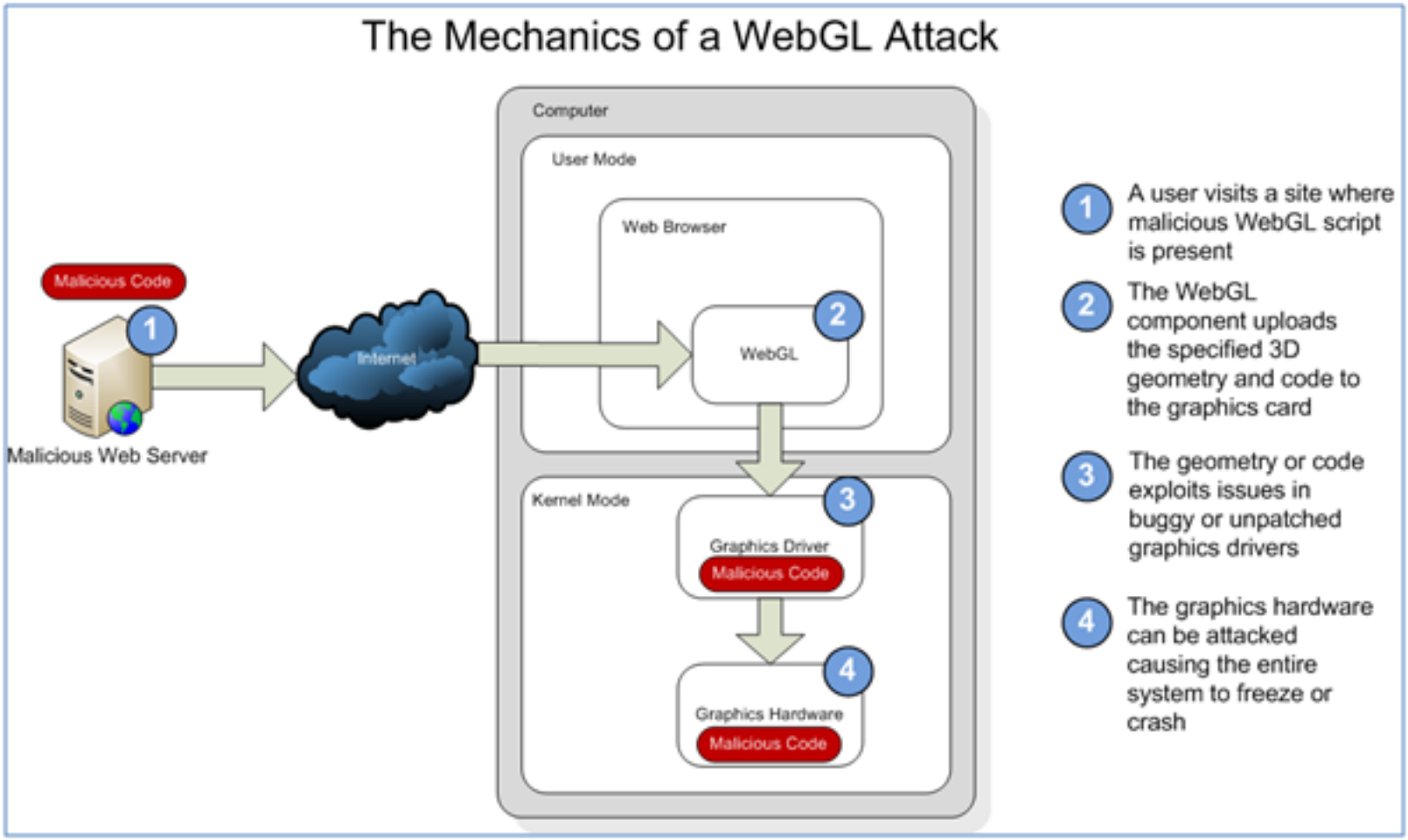

Image source:
http://www.contextis.com/research/blog/webgl-new-dimension-browser-exploitation/

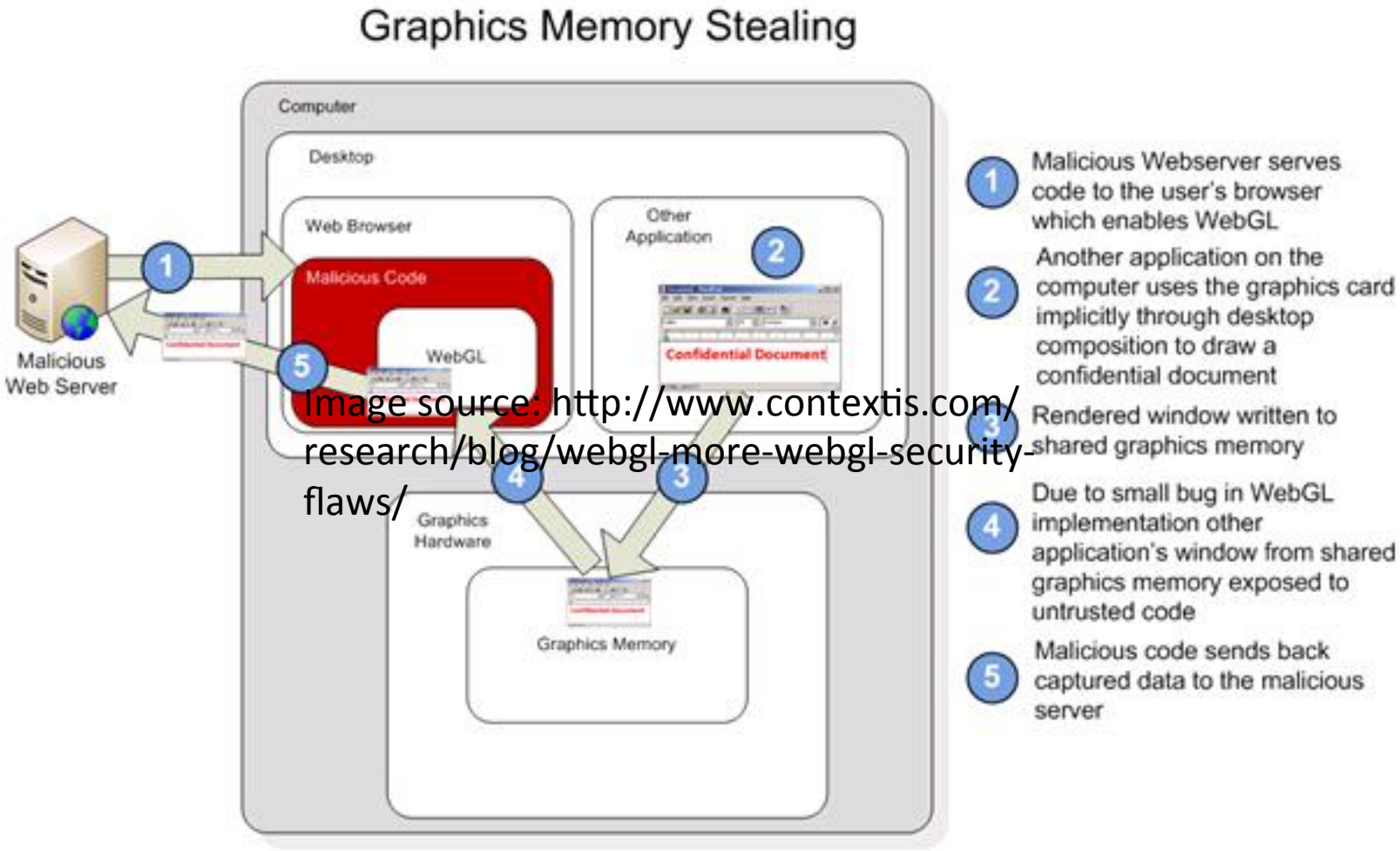

Image source: http://www.contextis.com/research/blog/webgl-more-webgl-security-flaws/

Image source:
http://www.contextis.com/research/blog/webgl-more-webgl-security-flaws/

Web Workers

- CPU hog
- If a page uses a worker, you cannot stop it with the browser's stop button (i.e., crash browser).
- Unlike threads, cannot set the priority of a worker.

Web Sockets

- Allows for multiplexing bi-directional, full-duplex communications channels over a single TCP connection (between client and server)
- Communications done over port 80

Web Sockets (continued)

- Another unfinished standard. The original design was riddled with flaws.
  - So bad that Mozilla had to turn it off.
- Data is always in plaintext
- Trusting clients (you know the drill by now, I hope)
- MitM
- Checking the origin

Local Storage

- Storage: ~5 MB of data per origin on the client-side (i.e., browser)
- According to the official spec, http://www.w3.org/TR/webstorage/:
  - *"User agents should guard against sites storing data under the origins other affiliated sites, e.g. storing up to the limit in a1.example.com, a2.example.com, a3.example.com, etc, circumventing the main example.com storage limit. A mostly arbitrary limit of five megabytes per origin is recommended."*
- How it works: no guard against affiliated sites / subdomains (e.g., 1.filldisk.com, 2.filldisk.com, 3.filldisk.com)
- Still an open issue on Chrome, IE, Safari. Not an issue on Firefox as Firefox has a 10 MB local storage cap for *any domain*. Try http://feross.org/fill-disk/

Node.js

- Server-side JavaScript
- Non-locking
- Great for creating fast, scalable, and lightweight networking software such as web servers
- A lot less bloated, less administration, and faster than Apache
- You still have to verify responses

Node.js: Server-Side Injection (whatcouldpossiblygowrong)

```
var http = require('http');
http.createServer(function (request, response) {
    if (request.method === 'POST') {
          var data = '';
          request.addListener('data', function(chunk) {
                    data += chunk;
          });
          request.addListener('end', function() {
                    try {
                              eval("(" + data + ")");
                    }
                    catch (exception) {}
          });
    }
}).listen(3000);
console.log('Listening on port 3000...');
```

Review: JavaScript

`eval(`*`code_as_string`*`)`

- The idea: executes string argument to function as JavaScript code
- Example usage (client-side): parsing JSON with `eval()`
  - Remember, JSON is a subset of JavaScript
  - The better option: use `JSON.parse()`
    - Built-in to some browsers such as Chrome
    - `JSON.parse()` will throw an exception if the text contains anything dangerous.
- Even more dangerous now on server-side

Mongo DB

- NoSQL; JSON-like document-storage
- Very fast
- No admin user or authentication enabled by default (must modify `/etc/mongodb.conf`). Alas, anyone can connect to your database from mongo
- Information gathering by simply looking at the `startup_log` collection in the `local` db
- Injection attacks a bit harder to do but still very doable via `$where` and `db.eval()`

In Summary: The Common Themes

1. This is no longer your father's JavaScript.
2. Even more important now to verify the responses and origins.
3. Some principles we have known for years are fading away.
4. Old security still matters.
5. Too common for people using technology naively.
6. Too common for organizations roll out APIs and technology naively.

References

- Chow, M. Abusing HTML5. DEF CON 19 Hacking Conference, The Rio All Suite Hotel and Casino, Las Vegas, NV, August 5 - 7, 2011.
- Crockford, D. JavaScript: The Good Parts, O'Reilly Media, 2008.
- Sullivan, B. Server-Side JavaScript Injection, BlackHat USA, 2011.
- http://www.contextis.com/research/blog/webgl-new-dimension-browser-exploitation/
- http://www.contextis.com/research/blog/webgl-more-webgl-security-flaws/
- http://news.cnet.com/8301-30685_3-20025272-264.html
- http://bishankochher.blogspot.com/2011/12/nodejs-security-good-bad-and-ugly.html
- http://www.slideshare.net/ASF-WS/asfws-2012-nodejs-security-old-vulnerabilities-in-new-dresses-par-sven-vetsch
- http://feross.org/fill-disk/
- https://community.qualys.com/blogs/securitylabs/2012/08/15/would-you-let-your-grandma-use-websockets