# SERVER-SIDE JAVASCRIPT INJECTION

ATTACKING AND DEFENDING NOSQL AND NODE.JS

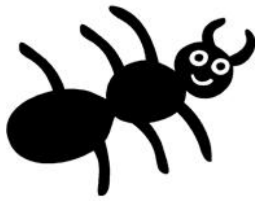
BRYAN SULLIVAN
SENIOR SECURITY RESEARCHER, ADOBE SYSTEMS

BRIEFINGS & TRAINING

USA + 2011

EMBEDDING SECURITY

## POP QUIZ!



## SERVER-SIDE JAVASCRIPT INJECTION VS XSS

» Client-side JavaScript injection (aka XSS)
  - #2 on OWASP Top Ten
  - #4 on 2011 CWE/SANS Top 25

» It's really bad.

» But server-side is much worse.

## BROWSER WAR FALLOUT

## BROWSER WAR FALLOUT

"…despite its deplorable shortcomings, JavaScript is cool and people like it" – Kris Kowal

## JAVASCRIPT DATABASES

CouchDB
relax

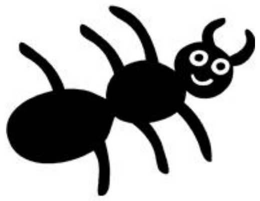
mongoDB

## JAVASCRIPT WEB SERVER

nodeJS

```
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
}).listen(1337, "127.0.0.1");
console.log('Server running at http://127.0.0.1:1337/');
```

## POP QUIZ PART 2…



## COMMONJS

CommonJS javascript: not just for browsers any more!

```
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
}).listen(1337, "127.0.0.1");
console.log('Server running at http://127.0.0.1:1337/');
```

## COMMONJS

CommonJS javascript: not just for browsers any more!

```
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
}).listen(1337, "127.0.0.1");
console.log('Server running at http://127.0.0.1:1337/');
```

## NODE.JS DOCUMENTATION

http://nodejs.org/docs/v0.5.0/api/

- Globals
- STDIO
- Timers
- Modules
- C/C++ Addons
- Process
- Utilities
- Events
- Buffers
- Streams
- Crypto
- TLS/SSL
- String Decoder
- File System
- Path
- Net
- UDP/Datagram
- DNS
- HTTP
- HTTPS
- URL
- Querystrings
- Readline
- REPL
- VM
- Child Processes
- Assertion Testing
- TTY
- OS
- Debugger

# NODE.JS DOCUMENTATION

http://nodejs.org/docs/v0.5.0/api/

- Globals
- STDIO
- Timers
- Modules
- C/C++ Addons
- Process
- Utilities
- Events
- Buffers
- Streams
- Crypto
- TLS/SSL
- String Decoder
- File System
- Path
- Net
- UDP/Datagram
- DNS
- HTTP
- HTTPS
- URL
- Querystrings
- Readline
- REPL
- VM
- Child Processes
- Assertion Testing
- TTY
- OS
- Debugger

## NODE.JS DOCUMENTATION

http://nodejs.org/docs/v0.5.0/api/

- Globals
- STDIO
- Timers
- Modules
- C/C++ Addons
- Process
- Utilities
- Events
- Buffers
- Streams
- Crypto
- TLS/SSL
- String Decoder
- File System
- Path
- Net
- UDP/Datagram
- DNS
- HTTP
- HTTPS
- URL
- Querystrings
- Readline
- REPL
- VM
- Child Processes
- Assertion Testing
- TTY
- OS
- Debugger

## NOSQL

mongoDB

CouchDB
relax

riak

Neo4j
the graph database

## POP QUIZ PART 3…

## NOSQL INJECTION

- Special case: MongoDB and PHP

- MongoDB expects input in JSON array format

```
find( { 'artist' : 'Amy Winehouse' } )
```

- In PHP, you do this with associative arrays

```
$collection->find(array('artist' => 'Amy Winehouse'));
```

## MONGODB AND PHP NOSQL INJECTION

- You also use associative arrays for query criteria

```
find( { 'album_year' : { '$gte' : 2011} } )
find( { 'artist' : { '$ne' : 'Lady Gaga' } } )
```

- But PHP will automatically create associative arrays from querystring inputs with square brackets

```
page.php?param[foo]=bar
param == array('foo' => 'bar');
```

## NOSQL INJECTION DEMO #1

## $WHERE CLAUSES

- Q: What does this have to do with SSJS injection?
- A: The $where clause lets you specify script to filter results

```
find( { '$where' : 'function() { return artist ==
          "Weezer"; }}' )

find ( '$where' : 'function() {
           var len = artist.length;
           for (int i=2; i<len; i++) {
             if (len % i == 0) return false;
           }
           return true; }')
```

## NOSQL INJECTION DEMO #2

mongoDB

## REST APIS AND CSRF

» From the MongoDB documentation
  - "One valid way to run the Mongo database is in a trusted environment, with no security and authentication"
  - This "is the default option and is recommended"

» From the Cassandra Wiki
  - "The default AllowAllAuthenticator approach is essentially pass-through"

» From CouchDB: The Definitive Guide
  - The "Admin Party": Everyone can do everything by default

» Riak
  - No authentication or authorization support

## PORT SCANNING

» If an attacker finds an open port, he's already won…

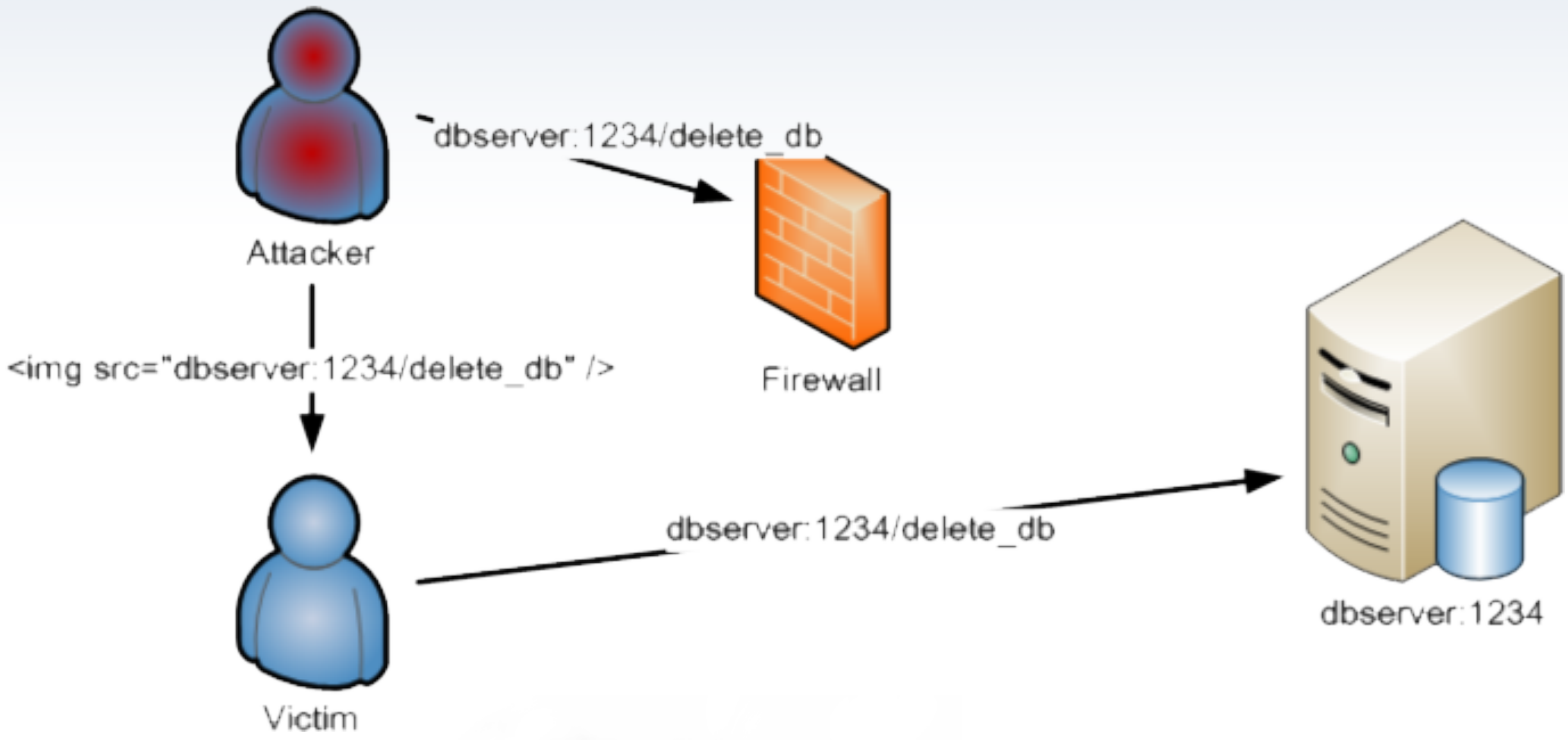
| Database | Default Port(s) |
|---|---|
| MongoDB | 27017<br>28017<br>27080 |
| CouchDB | 5984 |
| Hbase | 9000 |
| Cassandra | 9160 |
| Neo4j | 7474 |
| Riak | 8098 |

## PORT SCANNING

» If an attacker fi n…

| se | lt Port(s) |
|---|---|
| MongoDB | 17<br>8017<br>27080 |
| CouchDB | 5984 |
| Hbase | 9000 |
| Cassandra | 9160 |
| Neo4j | 74 |
| Riak | |

## CSRF FIREWALL BYPASS



## REST API EXAMPLES (COUCHDB)

» Create a document

- POST /mydb/doc_id HTTP/1.0
  {"album" : "Brothers", "artist" : "The Black Keys"}

» Retrieve a document

- GET /mydb/doc_id HTTP/1.0

» Update a document

- PUT /mydb/doc_id HTTP/1.0
  {"album" : "Brothers", "artist" : "The Black Keys"}

» Delete a document

- DELETE /mydb/doc_id HTTP/1.0

## TRADITIONAL GET-BASED CSRF

```
<img src="http://nosql:5984/_all_dbs"/>
```

» Easy to make a potential victim request this URL

» But it doesn’t do the attacker any good

» He needs to get the data back out to himself

## RIA GET-BASED CSRF

```
<script>
    var xhr = new XMLHttpRequest();
    xhr.open('get', 'http://nosql:5984/_all_dbs');
    xhr.send();
</script>
```

» Same-origin policy won’t allow this (usually)

» Same issue for PUT and DELETE

## POST-BASED CSRF

```
<form method=post action='http://nosql:5984/db'>
    <input type='hidden' name='{"data"}' value='' />
</form>
<script>
    // auto-submit the form
</script>
```

» Ok by the same-origin policy!

## CSRF INJECTION DEMOS

## POST IS ALL AN ATTACKER NEEDS

Insert arbitrary data

↓

Insert arbitrary script data

↓

Execute any REST command from inside the firewall

## QUESTIONS?

- » http://blogs.adobe.com/asset
- » brsulliv @ adobe

BRIEFINGS & TRAINING

USA + 2011

EMBEDDING SECURITY