

**CEIC 2008**

**Goal**

- What to expect from this session
  - Understanding of web architecture
  - Differences with the "Web 2.0" hype
  - Techniques that Google uses with Gmail
  - Data structures within Gmail
  - Possible artifacts left behind

PAGE 1

**CEIC 2008**

**Overview of Web Architecture**

- Browser and Server
- GET and POST
- Status Codes
- URL String Data
- Cache Control
- Content Encoding
- Client Side Code
- Server Side Code

PAGE 2

**CEIC 2008**

**Overview of Web Architecture**

- Browser and Server
- GET and POST
- Status Codes
- URL String Data
- Cache Control
- Content Encoding
- Client Side Code
- Server Side Code

The diagram shows a central box labeled 'Server'. Below it is a box labeled 'Browser'. An arrow points from the Browser to the Server. Above the Server, there are four boxes: 'Page 1', 'Page 2', 'Image 1', and 'Image 2'. Arrows point from the Server to each of these four boxes, indicating that the server provides these resources to the browser.

Browser – Software on client machines to retrieve and display web sites  
Server – Software on host machine to determine data requested and send stream

PAGE 3

**CEIC 2008**

### Overview of Web Architecture

- Browser and Server
- GET and POST
- Status Codes
- URL String Data
- Cache Control
- Content Encoding
- Client Side Code
- Server Side Code

```

    graph TD
      Browser[Browser] -- GET --> Server[Server]
      Server -- POST --> Browser
  
```

GET – Used by the browser to retrieve information (URL String)  
 POST – Used by the browser to send data to the server (POST Data)

PAGE 4

**CEIC 2008**

### Overview of Web Architecture

- Browser and Server
- GET and POST
- Status Codes
- URL String Data
- Cache Control
- Content Encoding
- Client Side Code
- Server Side Code

```

    graph TD
      Browser[Browser] -- GET --> Server[Server]
      Server -- 200 OK --> Browser
  
```

Status Codes – Sent to the browser from the server to indicate status of request.  
 Example: 200 = OK, 404 = Page Not Found, 500 = Internal Server Error

PAGE 5

**CEIC 2008**

### Overview of Web Architecture

- Browser and Server
- GET and POST
- Status Codes
- URL String Data
- Cache Control
- Content Encoding
- Client Side Code
- Server Side Code

```

    graph TD
      Browser[Browser] -- URL String --> Server[Server]
  
```

URL String – Browser uses the URL string to request certain data within a page. Information is sent in pairs of names and values.

`http://www.myurl.com/pagename?param1=data1&param2=data2&param3=data3`

PAGE 6

**CEIC 2008**

### Overview of Web Architecture

- Browser and Server
- GET and POST
- Status Codes
- URL String Data
- Cache Control
- Content Encoding
- Client Side Code
- Server Side Code

```

    graph TD
      Server[Server] -- Cache Control --> Browser[Browser]
  
```

Cache Control – Sent from server instructing browser to keep (or not) a local copy of data for faster retrieval on next request.

Example: no-cache

PAGE 7

**CEIC 2008**

### Overview of Web Architecture

- Browser and Server
- GET and POST
- Status Codes
- URL String Data
- Cache Control
- Content Encoding
- Client Side Code
- Server Side Code

Content Encoding – Used by the server to save bandwidth. Compression that is allowed by HTTP specifications.

Example: GZIP compression

PAGE 8

**CEIC 2008**

### Overview of Web Architecture

- Browser and Server
- GET and POST
- Status Codes
- URL String Data
- Cache Control
- Content Encoding
- Client Side Code
- Server Side Code

Client Side Code – Code that exists inside web pages instructing the browser to take action with data inside the page.

Example: Javascript, VBscript

PAGE 9

**CEIC 2008**

### Overview of Web Architecture

- Browser and Server
- GET and POST
- Status Codes
- URL String Data
- Cache Control
- Content Encoding
- Client Side Code
- Server Side Code

Client Side Code – Code that exists inside web pages but instructs the server to take action before sending the stream to the browser.

Example: Active Server Pages (ASP), PHP

PAGE 10

**CEIC 2008**

### Traditional Data Requests

- Traditional web requests start off with a GET or POST
- URL string is sometimes used to request specific data within a page
- Server sends status code in response
- Browser receives HTML data which contains links to other data
- Browser looks for links and makes more GET requests to server for additional data such as images, style sheets, include files, etc.
- Browser has built in functionality to structure HTML and additional data for viewing by the user
- Client side code is run to allow web pages to be interactive

PAGE 11

## Web 2.0 Data Requests

CEIC  
2008

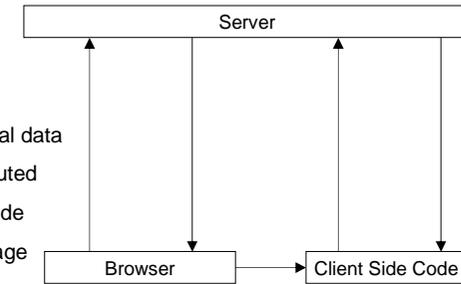
- Web 2.0 data requests start off with the same structure as traditional data requests
- Client side code is used to make additional data requests
- Client side code then modifies web page data with newly requested data
- This technique allows web pages to interact much quicker and smoother since only the required data is requested and sent from the server
- This ability is called Asynchronous Javascript And XML (AJAX)

PAGE 12

## Web 2.0 Data Requests

CEIC  
2008

- Traditional request
- Browser gets additional data
- Client side code executed
- New requests from code
- Code modifies web page



PAGE 13

## Gmail Concepts – One to Many

CEIC  
2008

- Gmail starts with a traditional web request, and then uses AJAX to request much more data.
- The original page requested by the browser is sometimes cached, so there is a javascript function to request data from Gmail about the version of the source code that is currently being executed. If the source code version is out of date, then the javascript can pull down a current version and “reboot”.
- Many more functions are in the source code to request data for all parts of the page. When the user clicks on a link, it calls a function to request data and manipulate it on the browser screen
- An http protocol analyzer can help to demonstrate the process.

PAGE 14

## Gmail Concepts – Data Packets

CEIC  
2008

- Gmail utilizes a data packet structure called Javascript Object Notation (JSON).
- JSON is a native format to javascript which makes parsing the data into an object very simple.
- The packet structure is basically an array of arrays seperated with a comma.
- Square brackets are used to enclose each array, as well as the parent array.
- Values can be string or data in any order. The designer of the code chooses the data contained within.

```
[
  ["value name 1", "other data related to 1", 1],
  ["value name 2", "other data related to 2", 2],
  ["value name 3", "other data related to 3", 3]
]
```

PAGE 15

## Gmail Concepts – Partial Page Updates

CEIC  
2008

- Once the source code retrieves the data from the server, standard javascript objects are used to populate the data within the page.
- Document Object Model (DOM) has been implemented in browsers for many years.
- The DOM gives javascript functions access to pieces of the page to manipulate the data.
- All objects in an HTML page are accessible through the DOM, but common tags used for this purpose are <DIV> and <SPAN>.
- Objects are typically named in an attribute and referred to by that attribute. <DIV id="data1"> </DIV>

PAGE 16

## Gmail Concepts – Protocol Analyzer

CEIC  
2008

- Any protocol analyzer can be used for this inspection
- There are a couple specialized analyzers specifically for HTTP traffic
  - Fiddler (Source Forge – Open Source Software)  
– <http://www.fiddlertool.com/fiddler/>
  - IEInspector - HTTP Analyzer (Commercial Software)  
– <http://www.ieinspector.com/httpanalyzer/>
- There is also a specialized viewer for the JSON data
  - JSON Viewer (Open Source Software)  
– <http://www.codeplex.com/JsonViewer>

PAGE 17

## Gmail Concepts – Review

CEIC  
2008

- The reasons we have discussed are the reasons why Gmail has a very responsive interface.
  - AJAX requests after initial traditional request
  - Javascript only manipulates the part of the page that is required
  - Gmail only requests the data from the server that is needed to make the partial page update
- These same reasons are what makes Gmail difficult for a forensic examiner to recover artifacts.

PAGE 18

## Gmail Artifacts

CEIC  
2008

- There are 3 main artifacts that may be recovered
  - Inbox lists
  - Messages
  - Contacts

PAGE 19

**Gmail Artifacts – Inbox Lists**

- JSON data packets
- The inbox list is loaded once a user authenticates.
- Messages are listed on the mailbox screen
- Each listing contains the first 90 characters of the message
- Listing also contains the related email address
- Other information found indicates dates and attachments

**Gmail Artifacts – Messages**

- JSON data packets
- Most typical email fields such as subject, sender, to, cc, date, etc
- Some fields are found more than once

**Gmail Artifacts – Contacts**

- JSON data packets
- Data found is name and email

**Gmail Artifacts – Common Data**

- Each of these data packets have specific details for their specific purpose.
- There are many common details found in the above packets that are consistent. Data found contains:
  - Name of Gmail account
  - Date and time from server when data was requested
  - Quota (usage) information of the account

## Gmail Artifacts - Keywords

- Each piece of data is identified at the front of the data packet section.
  - `While(1);` - usually at beginning of data packet
  - `["gn"` , - name of the account
  - `["st"` , - time on server when packet was constructed
  - `["qu"` , - quota information for gmail account
  - `["ds"` , - information about folders and unread messages
  - `["t"` , - identifies message list data
  - `["cs"` , - identifies subject of conversation
  - `["mi"` , - identifies information about one email of conversation
  - `["mb"` , - content of the message

## Gmail Artifacts - Caching

- Web developers have always had trouble with browsers caching data that was not intended to be cached. This results in stale data being displayed from cache when the user is making a request from the server.
- Browsers are obeying cache control statements more and more. This means that data is not being cached to the hard drive as much.
- Lack of cache makes a forensic examination of webmail very difficult. This is true of many other webmail services besides Gmail as well.

## What to do

- Reactive investigations may not recover all the data expected so there are some preventative measures available
  - Use of proxy and/or filtering software to monitor or block use of webmail
  - If allowed, install a protocol analyzer to collect all traffic or possibly only specific traffic such as webmail
  - Periodic sweeps of volatile memory

## Contact

- If you have any questions, please email me
  
- [james.habben@encase.com](mailto:james.habben@encase.com)