Web application analysis with OWASP Hatkit
Presentation

- Martin Holst Swende
  - @mhswende
- Patrik Karlsson
  - @nevdlull77
Web application testing

- Is very diverse: from a low-level infrastructure point-of-view to high-level application flow
- There are many tools, but a central component is an intercepting proxy
- Usually complex beasts
# Typical proxy features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirement</th>
<th>Must be in proxy?</th>
<th>Possible alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sitemapping</td>
<td>Traffic data</td>
<td>No</td>
<td>Http-level: trivial. Based on html inspection: e.g. in browser DOM–javascript.</td>
</tr>
<tr>
<td>Content analysis</td>
<td>Traffic data</td>
<td>No</td>
<td>W3af, ratproxy, proxmon, webscarab, burp etc</td>
</tr>
<tr>
<td>Fuzzing</td>
<td>Traffic data</td>
<td>No</td>
<td>JBroFuzz</td>
</tr>
<tr>
<td>Spidering</td>
<td>Traffic data</td>
<td>No</td>
<td>Browser-based spiders with DOM-access. Many choices.</td>
</tr>
<tr>
<td>Interception</td>
<td>Live traffic</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td>Manual request</td>
<td>Traffic data + sockets</td>
<td>No</td>
<td>An http/html/json/xml editor + sockets</td>
</tr>
<tr>
<td>Manual inspect</td>
<td>Traffic data</td>
<td>No</td>
<td>An http/html/json/xml editor</td>
</tr>
<tr>
<td>Sess. id analysis</td>
<td>Traffic data</td>
<td>No</td>
<td>Stompy</td>
</tr>
<tr>
<td>Search</td>
<td>Traffic data</td>
<td>No</td>
<td>Wide range: grep to lucene</td>
</tr>
</tbody>
</table>
Typical proxy drawbacks

- It hogs my machine
  - Oh noes: OS updates itself through the proxy
  - They usually don’t perform well after a few thousand requests
- It is not flexible
  - Ok, I see the GET-params in the overview.
    - ...but now I want to see the POST – params
    - ... and now I want to see which of my browsers sent it
    - ... and now I want to see all Server-headers. Ordered by path.
    - ... and now I only want to see responses with content type application/json and the value of the json parameter “foobar”.
  - And what’s with all these cookies eating my screen real estate?
- It is not open
  - I wonder if <tool> would’ve detected that internal ip address?
  - “Let’s chain it: Webscarab, Burp, Paros and Ratproxy”
    - The road to madness...
The Hatkit Project
Http Analysis Toolkit

- Write an intercepting proxy  **Hatkit:Proxy**
  - Lightweight
    - Memory-consumption does not grow with traffic
    - Streams all non-captured traffic to destination asap
  - Recording
    - Saves to database - MongoDB
      - Document store where parsed data is stored as JSON documents
      - Platform independent, Open Source and fast

- Write an analysis engine  **Hatkit:Datafiddler**
  - Flexible
    - Using MongoDB advanced querying facilities
    - Using dynamic views for data
  - And open
    - With several different ways to analyse, export and utilise existing applications.
- Based on Owasp Proxy (by Rogan Dawes)
- Records traffic to DB, both in parsed object form and the raw binary data.
- TCP interception (still in alpha)
- Syntax highlightning
- FQ/NFQ intercept mode (think freedom as in telnet)
- Proxy chaining
- Reverse proxy mode
- ...This is definitely not your all-in-one proxy!
What is it?
What does it do?
Why use it?
How do I get it?
What does it run on, prerequisites?
Hatkit Datafiddler

What is it?

- A MongoDB browser, with additional functionality to extract and display information geared towards web application testing.
- A platform for utilising existing tools on pre-recorded data.
What does it do?
- Displays traffic data as defined by the user
- Traffic and pattern aggregation
- Traffic analysis via w3af and ratproxy
- Export recorded traffic to other proxies
- Filter and sort data
- And more...
Traffic overview

- It is simple to write the kind of view you need for the particular purpose at hand.
- Example scenarios:
  - Analysing user interaction using several accounts with different browsers, you are interested in cookies, user-agent
  - Analysing server infrastructure
    - Server headers, Banner-values, File extensions, Cookie names
  - Searching for potential XSS
    - Use filters to see only the requests where content is reflected
  - Analyzing brute-force attempt
    - Request parameter username, password, Response delay, body size, status code and body hash
### Variables

<table>
<thead>
<tr>
<th>v0</th>
<th>_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>request.time</td>
</tr>
<tr>
<td>v2</td>
<td>request.headers</td>
</tr>
<tr>
<td>v3</td>
<td>request.url</td>
</tr>
<tr>
<td>v4</td>
<td>response.status</td>
</tr>
<tr>
<td>v5</td>
<td>response.headers</td>
</tr>
</tbody>
</table>

### Column

<table>
<thead>
<tr>
<th>Column</th>
<th>Coloring</th>
<th>Enabled</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 v0</td>
<td></td>
<td>✔️</td>
<td>v0</td>
</tr>
<tr>
<td>1 date(v1)</td>
<td></td>
<td>✔️</td>
<td>Date</td>
</tr>
<tr>
<td>2 v1</td>
<td></td>
<td>✔️</td>
<td>Utc</td>
</tr>
<tr>
<td>3 &quot;Time: %s&quot; % v1</td>
<td></td>
<td>✔️</td>
<td>Python</td>
</tr>
<tr>
<td>4 paramstring(v3)</td>
<td></td>
<td>✔️</td>
<td>paramstring(v3)</td>
</tr>
<tr>
<td>5 v4</td>
<td></td>
<td>✔️</td>
<td>v4</td>
</tr>
<tr>
<td>6 size(v5)</td>
<td></td>
<td>✔️</td>
<td>size(v5)</td>
</tr>
<tr>
<td>7 cookies(v2)</td>
<td>✔️</td>
<td>✔️</td>
<td>cookies(v2)</td>
</tr>
</tbody>
</table>

**Add variable**

**Add Column**

**Help**

**Revert**

**Apply**
The vo parameter is the object id. This column uses 'Coloring', which means that the value is not displayed, instead a color is calculated from the hash of the value.
Aggregation

- Aggregation (grouping) is a feature of MongoDB.
  - It is like a specialized Map/Reduce
- You provide the framework with a couple of directives and the database will return the results, which are different kinds of sums.
  - Pass JS right into the DB
- Example scenarios:
  - Generate sitemap
  - Show all http response codes, sorted by host/path
  - Show all unique http header keys, sorted by extension
  - Show all request parameter names, grouped by host
  - Show all unique request parameter values, in grouped by host
AggregatePaths
AggregatePathsSimple
HTTP Status -> path
Host->Server banner
List response headers
Host -> Parameter names
Host->Parameter name->value
function(obj,res)
{
    if(obj.request && obj.request.url && obj.request.url.path)
    {
        var path=obj.request.url.path;
        path=path.split("/");
        var dir=res.count;
        for(x=0;x<path.length;x++)
        {
            if(path[x].length > 0){
                var next = dir[path[x]]; 
                if(!next){dir[path[x]]={};}
                dir=dir[path[x]];
            }
        }
    }
    var p=obj.request.paramstring;
}
Initial {'count': {}}
Key ['request.headers.Host']
Cond {}/
Datafiddler has a mechanism to run selected traffic through third-party plugins. Currently implemented*:

- Ratproxy plugin. Starts ratproxy process, feeds traffic through it, and collects output.
- Generic proxy plugin. Feeds data to a proxy (e.g., Burp) which in turn uses a Datafiddler as forward proxy.
- Webscarab export. Writes traffic data to webscarab save-format. Useful e.g. to do manual requests edit or use fuzzer.

* Defcon19-release
Traffic analysis via ratproxy

- **RatAnalyser**
  - Ratproxy path: /usr/bin/ratproxy

- **WebscarabExporter**
  - Start id: 1
  - Save-path: /home/martin/fiddler-webcarab/2011-06-23_1

- **ProxyExporter**
  - 3rd party proxy port: 8080
  - Listening port: 9999

[Run button]
<table>
<thead>
<tr>
<th>row</th>
<th>warn</th>
<th>mod</th>
<th>mesg</th>
<th>off_par</th>
<th>res.code</th>
<th>res.payloadlength</th>
<th>res.mimetype</th>
<th>res.sniffedmimetype</th>
<th>res.charset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Bad or no charset declared for renderable file</td>
<td>-</td>
<td>200</td>
<td>18183</td>
<td>text/css</td>
<td>text/plain</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>MIME type mismatch on renderable file</td>
<td>-</td>
<td>200</td>
<td>18183</td>
<td>text/css</td>
<td>text/plain</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>XSS candidates (script)</td>
<td>useskin</td>
<td>200</td>
<td>205</td>
<td>text/javascript</td>
<td>text/javascript</td>
<td>utf-8</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Bad or no charset declared for renderable file</td>
<td>-</td>
<td>200</td>
<td>65290</td>
<td>text/javascript</td>
<td>text/javascript</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>Risky Javascript code</td>
<td>innerHTML</td>
<td>200</td>
<td>65290</td>
<td>text/javascript</td>
<td>text/javascript</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>Bad or no charset declared for renderable file</td>
<td>-</td>
<td>200</td>
<td>4777</td>
<td>text/javascript</td>
<td>text/javascript</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>Markup in dynamic Javascript</td>
<td>-</td>
<td>200</td>
<td>4777</td>
<td>text/javascript</td>
<td>text/javascript</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>Risky Javascript code</td>
<td>innerHTML</td>
<td>200</td>
<td>4777</td>
<td>text/javascript</td>
<td>text/javascript</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>Bad or no charset declared for renderable file</td>
<td>-</td>
<td>200</td>
<td>30873</td>
<td>text/javascript</td>
<td>text/javascript</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>Markup in dynamic Javascript</td>
<td>-</td>
<td>200</td>
<td>30873</td>
<td>text/javascript</td>
<td>text/javascript</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>Risky Javascript code</td>
<td>innerHTML</td>
<td>200</td>
<td>30873</td>
<td>text/javascript</td>
<td>text/javascript</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>5</td>
<td>MIME type mismatch on renderable file</td>
<td>-</td>
<td>200</td>
<td>11</td>
<td>text/css</td>
<td>text/plain</td>
<td>utf-8</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>5</td>
<td>Request splitting candidates</td>
<td>ctype</td>
<td>200</td>
<td>11</td>
<td>text/css</td>
<td>text/plain</td>
<td>utf-8</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>Bad or no charset declared for renderable file</td>
<td>-</td>
<td>200</td>
<td>1314</td>
<td>text/css</td>
<td>text/plain</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>MIME type mismatch on renderable file</td>
<td>-</td>
<td>200</td>
<td>1314</td>
<td>text/css</td>
<td>text/plain</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>5</td>
<td>MIME type mismatch on renderable file</td>
<td>-</td>
<td>200</td>
<td>50</td>
<td>text/css</td>
<td>text/plain</td>
<td>utf-8</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>5</td>
<td>Request splitting candidates</td>
<td>ctype</td>
<td>200</td>
<td>50</td>
<td>text/css</td>
<td>text/plain</td>
<td>utf-8</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>5</td>
<td>Request splitting candidates</td>
<td>ctype</td>
<td>200</td>
<td>1256</td>
<td>text/css</td>
<td>text/css</td>
<td>utf-8</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>1</td>
<td>Bad or no charset declared for renderable file</td>
<td>-</td>
<td>200</td>
<td>1634</td>
<td>text/css</td>
<td>text/plain</td>
<td>-</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>1</td>
<td>MIME type mismatch on renderable file</td>
<td>-</td>
<td>200</td>
<td>1634</td>
<td>text/css</td>
<td>text/plain</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>1</td>
<td>Risky Javascript code</td>
<td>document.write</td>
<td>200</td>
<td>59829</td>
<td>text/html</td>
<td>text/html</td>
<td>utf-8</td>
</tr>
</tbody>
</table>
Why use it?

- To better be able to make sense of large bodies of complex information
- To maintain control of your data by not tying it to one single application
How do I get it?

- Download the source
  - https://bitbucket.org/holiman/hatkit-proxy/
  - https://bitbucket.org/holiman/hatkit-datafiddler/

- Or the released binaries
  - https://bitbucket.org/holiman/hatkit-proxy/downloads
  - https://bitbucket.org/holiman/hatkit-datafiddler/downloads

- And check out the documentation
What does it run on, prerequisites?

- Python
- Qt4
- PyQt4 bindings
- Python MongoDB driver
- MongoDB
- (optional: w3af)
- (optional: ratproxy)

Tested on Linux and MacOSX
Upcoming features

- Cache proxy
  - Datafiddler can act as forwarding proxy and use collected traffic as cache. On cache miss, it can either contact remote host or issue 403. This enables:
    - Resume aborted Nikto-scan
    - Gather e.g. screenshots post mortem without access to target

- Fuzzer integration
  - Send requests directly to a fuzzer.

- New release at Defcon19!
For web application testers, the Hatkit combo is very useful for analyzing remote servers and applications, from a low-level infrastructure point-of-view to high-level application flow. For server administrators, The Hatkit Proxy can be set as a reverse proxy, logging all incoming traffic. The combo can then be used as a tool to analyze user interaction, e.g. to detect malicious activity and perform post mortem analysis. The back-end can scale to handle massive amounts of data.
To learn more or join the project, join the mailing lists

- Owasp-hatkit-datafiddler-project@lists.owasp.org
- Owasp-hatkit-proxy-project@lists.owasp.org
Thank you all for listening

- Questions?