Web application analysis with OWASP Hatkit
Presentation

- Martin Holst Swende
  - Senior Security Consultant, 2Secure AB
  - ~3 years in IT-security
  - ~5 years as software developer
  - Nmap and Webscarab contributor
  - Project leader of OWASP Hatkit projects
  - Twitter: @mhswende
  - Speaker at Defcon19
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

- Resource intensive
  - All requests/responses get processed/buffered by the proxy regardless if intercepted or not
    - OS security updates, video clips, etc.
  - This usually results in a huge memory footprint

- Static view of request/response data
  - Static GUI, difficult/impossible to customize fields/views
  - Limited filtering and search capabilities

- Limited post-processing capabilities
  - What gets stored in the backend database?
  - Can I export it to other tools?
  - How do I even access the stored data?
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
  - 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 analyze, export and use existing applications.
Based on OWASP Proxy (by Rogan Dawes)
Records traffic to DB
  ▪ parsed object form
  ▪ raw binary data
Syntax highlighting
FQ/NFQ intercept mode (think freedom as in telnet)
Proxy chaining
Reverse proxy mode
TCP interception (early beta)
...This is definitely not your all-in-one proxy!
TCP interception

- Setup interception
  - DNS poisoning
  - /etc/hosts

- Possible to alter packets using
  - Manual editing as hex/string
  - “Processor” – a BeanShell script

- Processors
  - Each TCP session has their own script engine instance
  - Possible to keep state and record info in registry
What is it?
What does it do?
Why use it?
How do I get it?
What does it run on, prerequisites?
What is it?

- A framework to analyse web traffic
- A platform based on MongoDB, 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 (Tableview)
- Traffic and pattern aggregation (Aggregator)
- Traffic analysis via w3af and ratproxy (3pp)
- Export recorded traffic to other proxies (3pp)
- Filter and sort data (filters+tableview)
- Cache proxying (cache-proxy, beta)
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
Traffic is stored as parsed objects in the database.

```json
{{ request:
    { headers: { Host: "server.com", Cookie: ...},
      { uri: { path: "/foobar", params:{ foo: "bar"}...},
        ...} //Parsed request object
    { response : ...} //Parsed response object
    { request-raw} //Binary raw content
    {response-raw} //Binary raw content
}
```
Demo – Traffic overview
Demo – Traffic overview
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
Demo – Using aggregator
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. Useful e.g. to do manual requests edit or use fuzzer.
- W3af greppers
- * Defcon19-release
Demo – Ratproxy analysis
Demo – W3af greppers
Demo – Generic exporter
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 scans (Nikto, ...)
    - Gather e.g. screenshots post mortem without access to target

- Fuzzer integration
  - Send requests directly to a fuzzer.

- Text search
Why use it?

- To better be able to make sense of large bodies of complex information
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
Hatkit Datafiddler

- **What does it run on, prerequisites?**
  - Python
  - Qt4
  - PyQt4 bindings
  - Python MongoDB driver
  - MongoDB
  - (optional: w3af)
  - (optional: ratproxy)

- Tested on Linux and MacOSX
Who should care?

- **Application testers**
  - Hatkit is very useful for analyzing remote servers and applications from a low-level infrastructure point-of-view to high-level application flow.

- **Server administrators**
  - The Hatkit Proxy can be set as a reverse proxy, logging all incoming traffic.
  - The Datafiddler can analyze user interaction, eg. 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?
Some additional screenshots...

- ... for you who weren’t there to see the demos...
Traffic overview

The image shows a screen capture of a software interface with a table and form elements. The table is labeled as `Tabledata settings` and is divided into two sections: `Variables` and `Column`. The `Variables` section lists various variables with their respective labels, such as `_id`, `request.time`, `request.headers`, `request.url`, `response.status`, and `response.headers`. The `Column` section includes columns for `Coloring`, `Enabled`, and `Title`, with entries like `v0`, `date(v1)`, `datetime(v1)`, `paramstring(v3)`, `v4`, `size(v5)`, and `cookies(v2)`. The interface also includes options for `Load`, `Save as`, `Add variable`, and `Add Column`. The bottom of the interface has buttons for `Help`, `Revert`, and `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.
HatKit – Aggregator
HatKit – Aggregator

Pre-defined Aggregates
Here you can aggregate the results of the tool. You can also switch to the Advanced view.

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;
    }
}
```
Traffic analysis via ratproxy

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

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

- **ProxyExporter**
  - 3rd party proxy port: 8080
  - Listening port: 9999
### Traffic analysis via ratproxy

<table>
<thead>
<tr>
<th>row</th>
<th>warn</th>
<th>mod</th>
<th>msg</th>
<th>off_par</th>
<th>res.code</th>
<th>res_payloadlength</th>
<th>res_mimetype</th>
<th>res_sniffedmimeType</th>
<th>res_chargestatus</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/plain</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>