Web application analysis with OWASP Hatkit

OWASP
The Open Web Application Security Project

Presentation

- Martin Holst Swende
  - @mhswende
- Patrik Karlsson
  - @nevdull77

2Secure

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

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

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.

Hatkit:Proxy

the intercepting recording proxy

- 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!

Hatkit:Datafiddler

The analysis engine

- What is it?
- What does it do?
- Why use it?
- How do I get it?
- What does it run on, prerequisites?

Hatkit Datafiddler

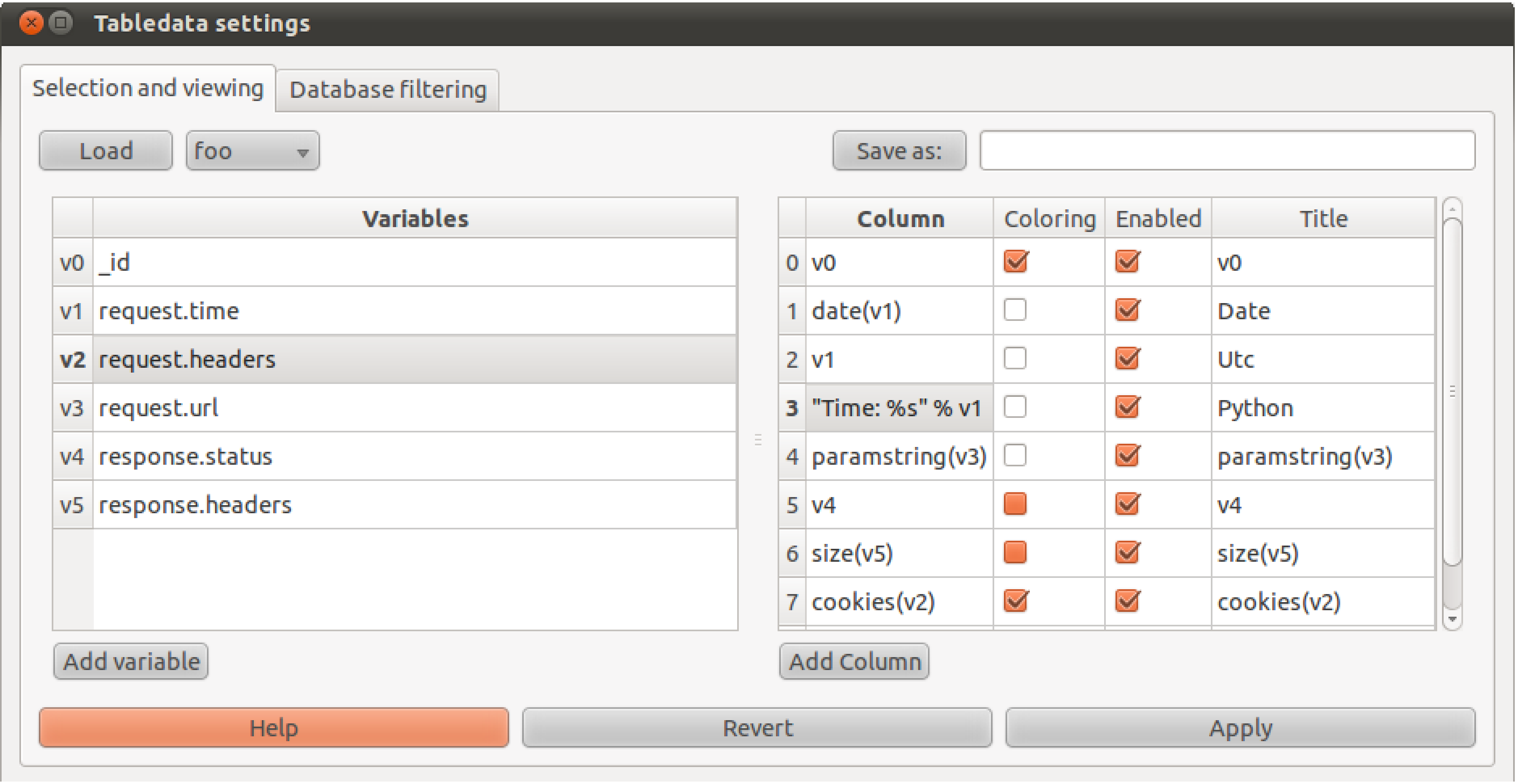
- What is it?
  - A MongoDB browser, with additional functionality to extract and display information geared towards web application testing.
  - A platform for utilising existing tools on pre-recorded data.

Hatkit Datafiddler

- 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…

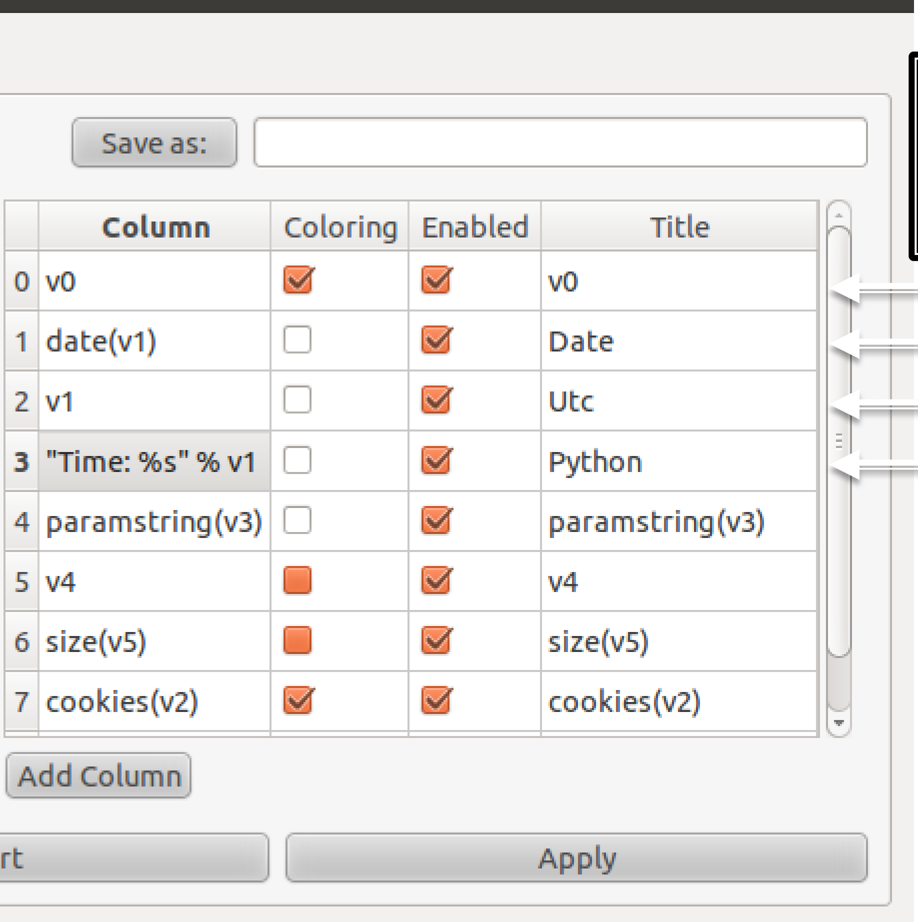
Traffic overview

- 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
v0	_id
v1	request.time
v2	request.headers
v3	request.url
v4	response.status
v5	response.headers

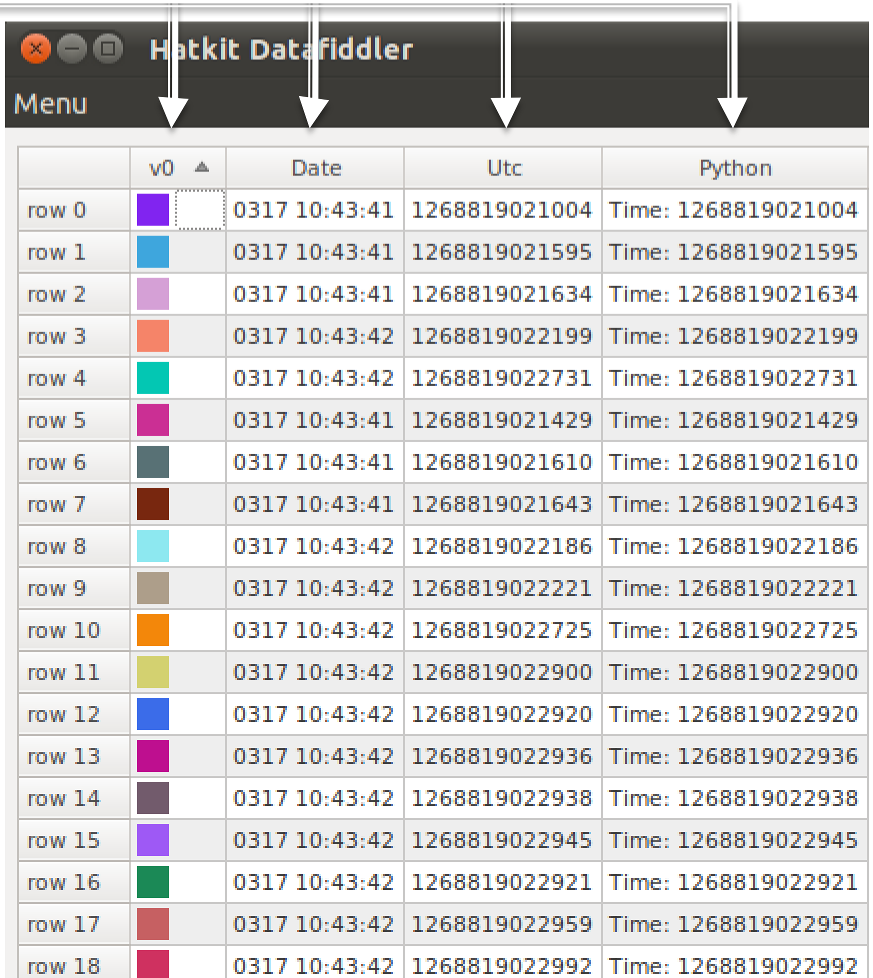
	Column	Coloring	Enabled	Title
0	v0	☑	☑	v0
1	date(v1)	☐	☑	Date
2	v1	☐	☑	Utc
3	"Time: %s" % v1	☐	☑	Python
4	paramstring(v3)	☐	☑	paramstring(v3)
5	v4	■	☑	v4
6	size(v5)	■	☑	size(v5)
7	cookies(v2)	☑	☑	cookies(v2)



Save as:

	Column	Coloring	Enabled	Title
0	v0	☑	☑	v0
1	date(v1)	□	☑	Date
2	v1	□	☑	Utc
3	"Time: %s" % v1	□	☑	Python
4	paramstring(v3)	□	☑	paramstring(v3)
5	v4	■	☑	v4
6	size(v5)	■	☑	size(v5)
7	cookies(v2)	☑	☑	cookies(v2)

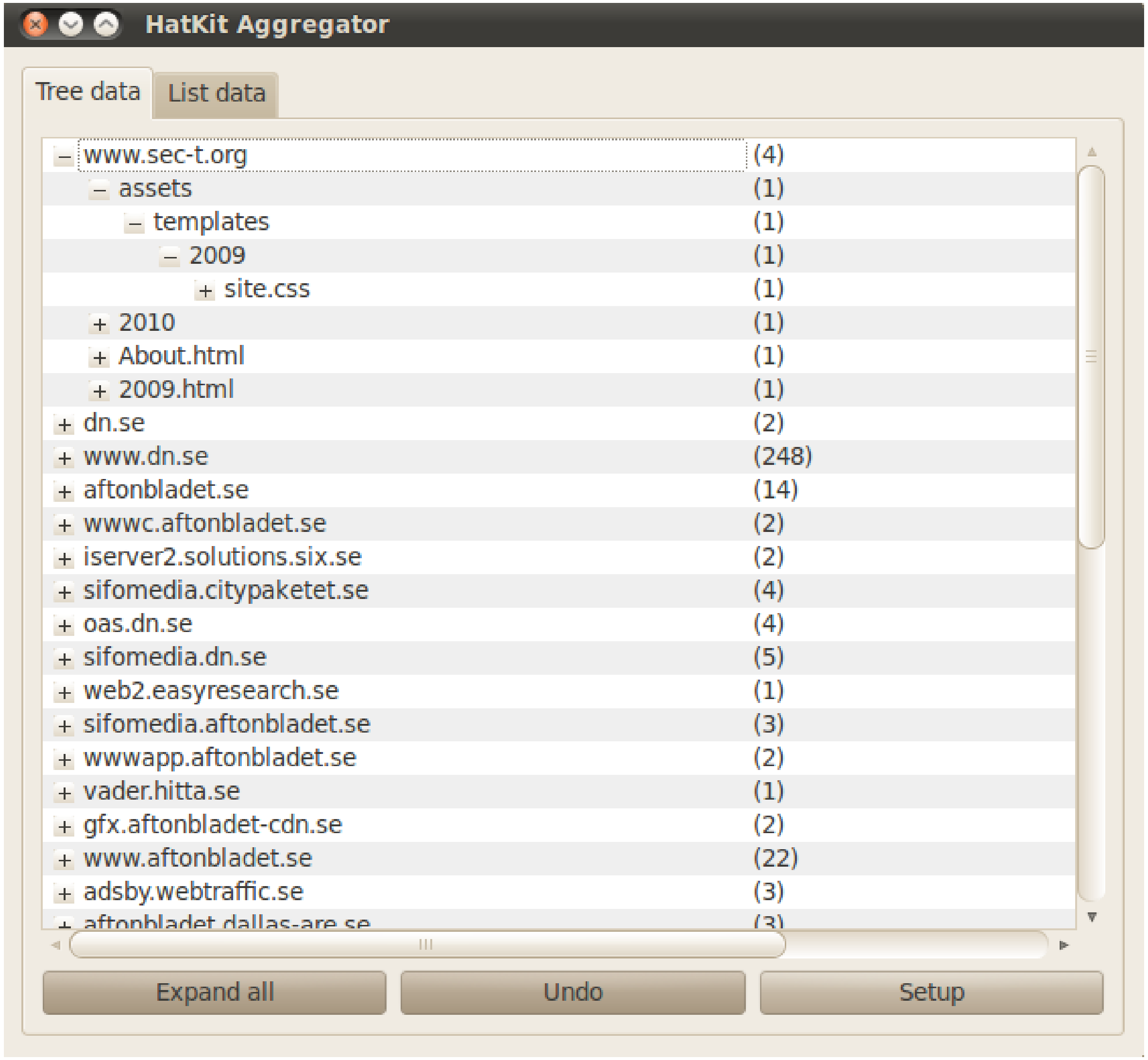
The vo parameter is the object id. This column uses 'Coloring', which means that the value is not displayed, instead a color is calculated from the hash of the value.



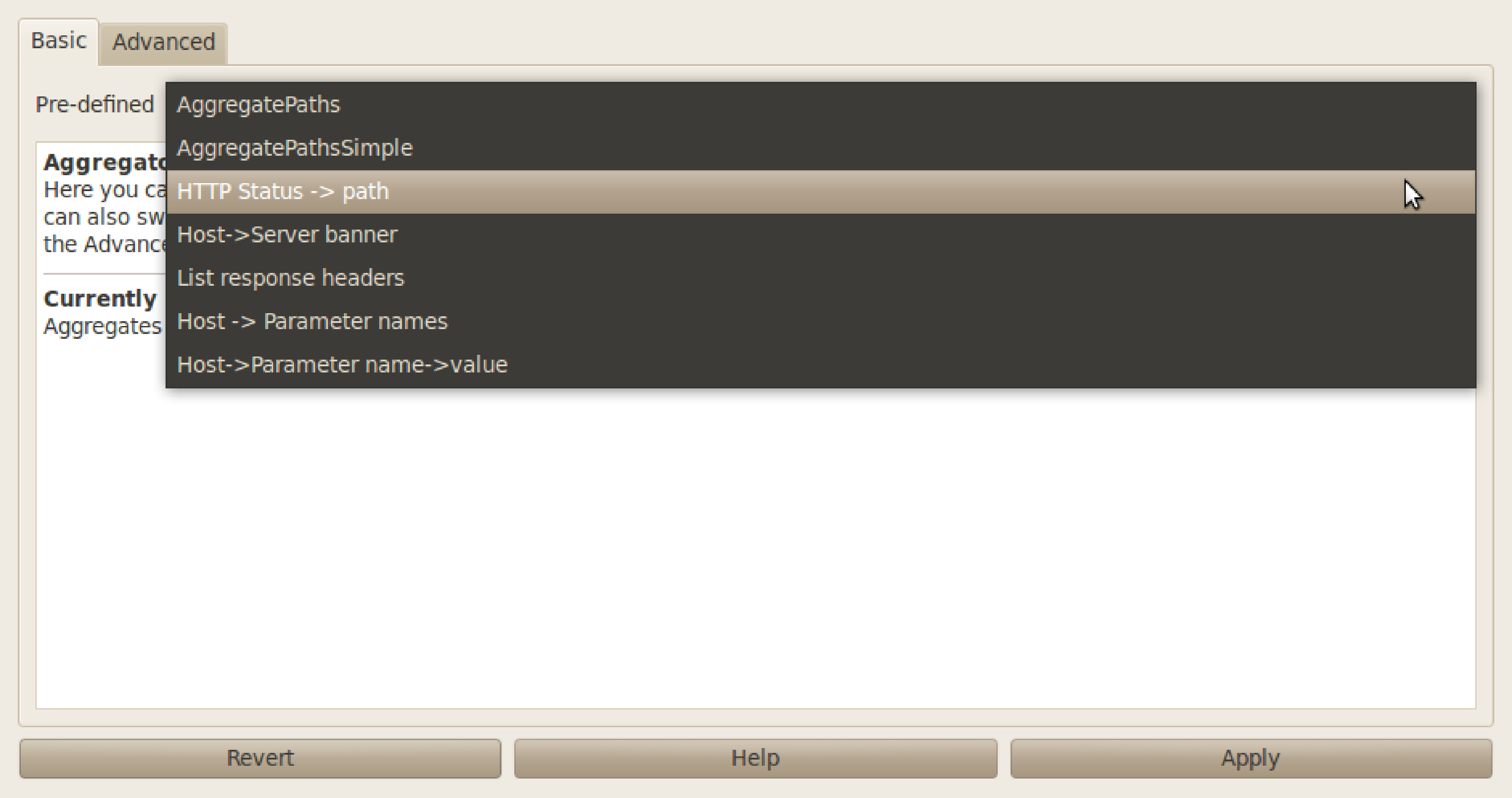
	v0 ▲	Date	Utc	Python
row 0	■	0317 10:43:41	1268819021004	Time: 1268819021004
row 1	■	0317 10:43:41	1268819021595	Time: 1268819021595
row 2	■	0317 10:43:41	1268819021634	Time: 1268819021634
row 3	■	0317 10:43:42	1268819022199	Time: 1268819022199
row 4	■	0317 10:43:42	1268819022731	Time: 1268819022731
row 5	■	0317 10:43:41	1268819021429	Time: 1268819021429
row 6	■	0317 10:43:41	1268819021610	Time: 1268819021610
row 7	■	0317 10:43:41	1268819021643	Time: 1268819021643
row 8	■	0317 10:43:42	1268819022186	Time: 1268819022186
row 9	■	0317 10:43:42	1268819022221	Time: 1268819022221
row 10	■	0317 10:43:42	1268819022725	Time: 1268819022725
row 11	■	0317 10:43:42	1268819022900	Time: 1268819022900
row 12	■	0317 10:43:42	1268819022920	Time: 1268819022920
row 13	■	0317 10:43:42	1268819022936	Time: 1268819022936
row 14	■	0317 10:43:42	1268819022938	Time: 1268819022938
row 15	■	0317 10:43:42	1268819022945	Time: 1268819022945
row 16	■	0317 10:43:42	1268819022921	Time: 1268819022921
row 17	■	0317 10:43:42	1268819022959	Time: 1268819022959
row 18	■	0317 10:43:42	1268819022992	Time: 1268819022992

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

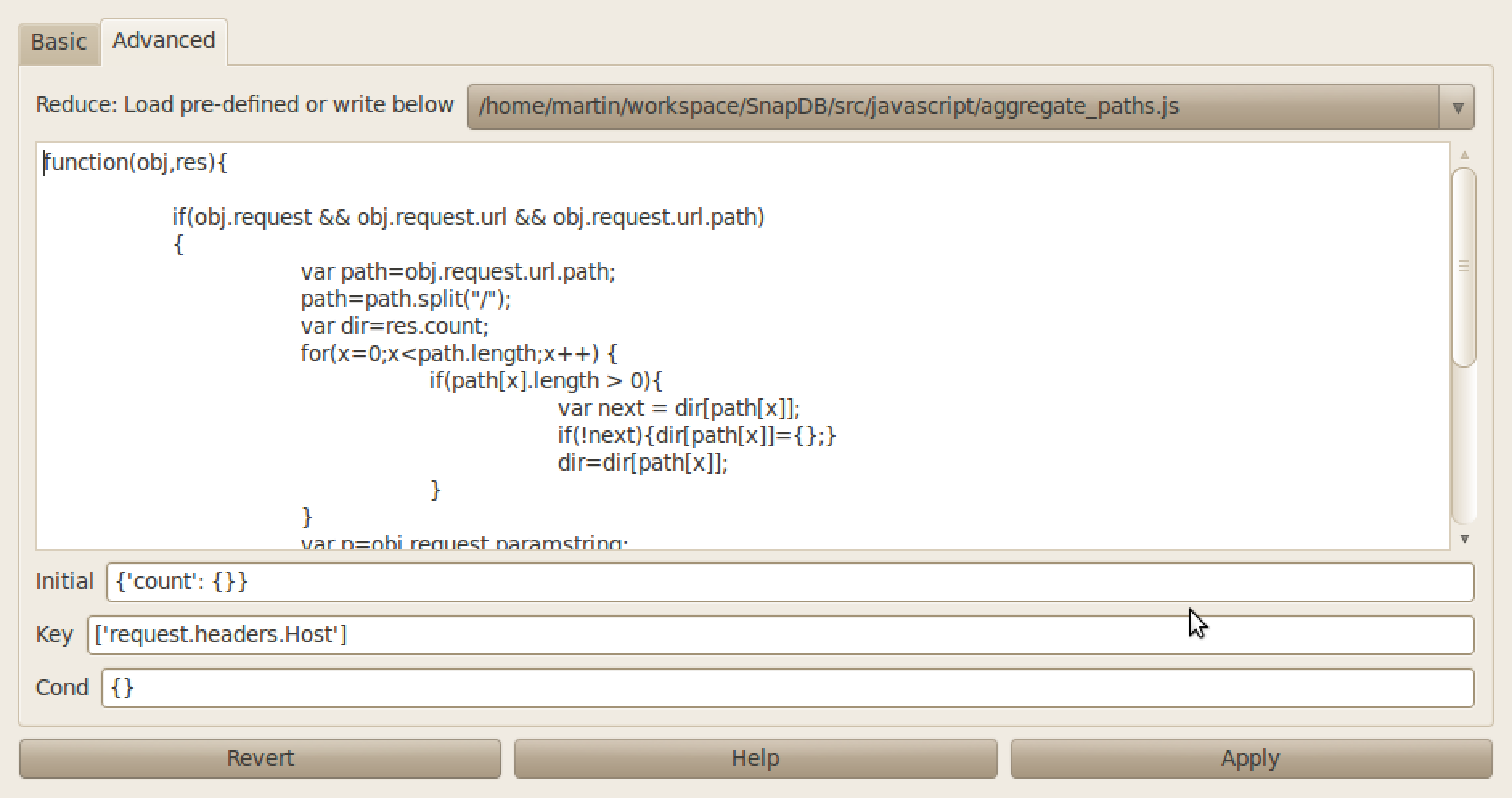


```
–  www.sec-t.org                          (4)
   –  assets                              (1)
      –  templates                        (1)
         –  2009                          (1)
            +  site.css                   (1)
   +  2010                                (1)
   +  About.html                          (1)
   +  2009.html                           (1)
+  dn.se                                  (2)
+  www.dn.se                              (248)
+  aftonbladet.se                         (14)
+  wwwc.aftonbladet.se                    (2)
+  iserver2.solutions.six.se              (2)
+  sifomedia.citypaketet.se               (4)
+  oas.dn.se                              (4)
+  sifomedia.dn.se                        (5)
+  web2.easyresearch.se                   (1)
+  sifomedia.aftonbladet.se               (3)
+  wwwapp.aftonbladet.se                  (2)
+  vader.hitta.se                         (1)
+  gfx.aftonbladet-cdn.se                 (2)
+  www.aftonbladet.se                     (22)
+  adsby.webtraffic.se                    (3)
+  aftonbladet.dallas-are.se              (3)
```



Aggregat
Here you ca
can also sw
the Advanc

Currently
Aggregates



Reduce: Load pre-defined or write below /home/martin/workspace/SnapDB/src/javascript/aggregate_paths.js

```
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': {}}

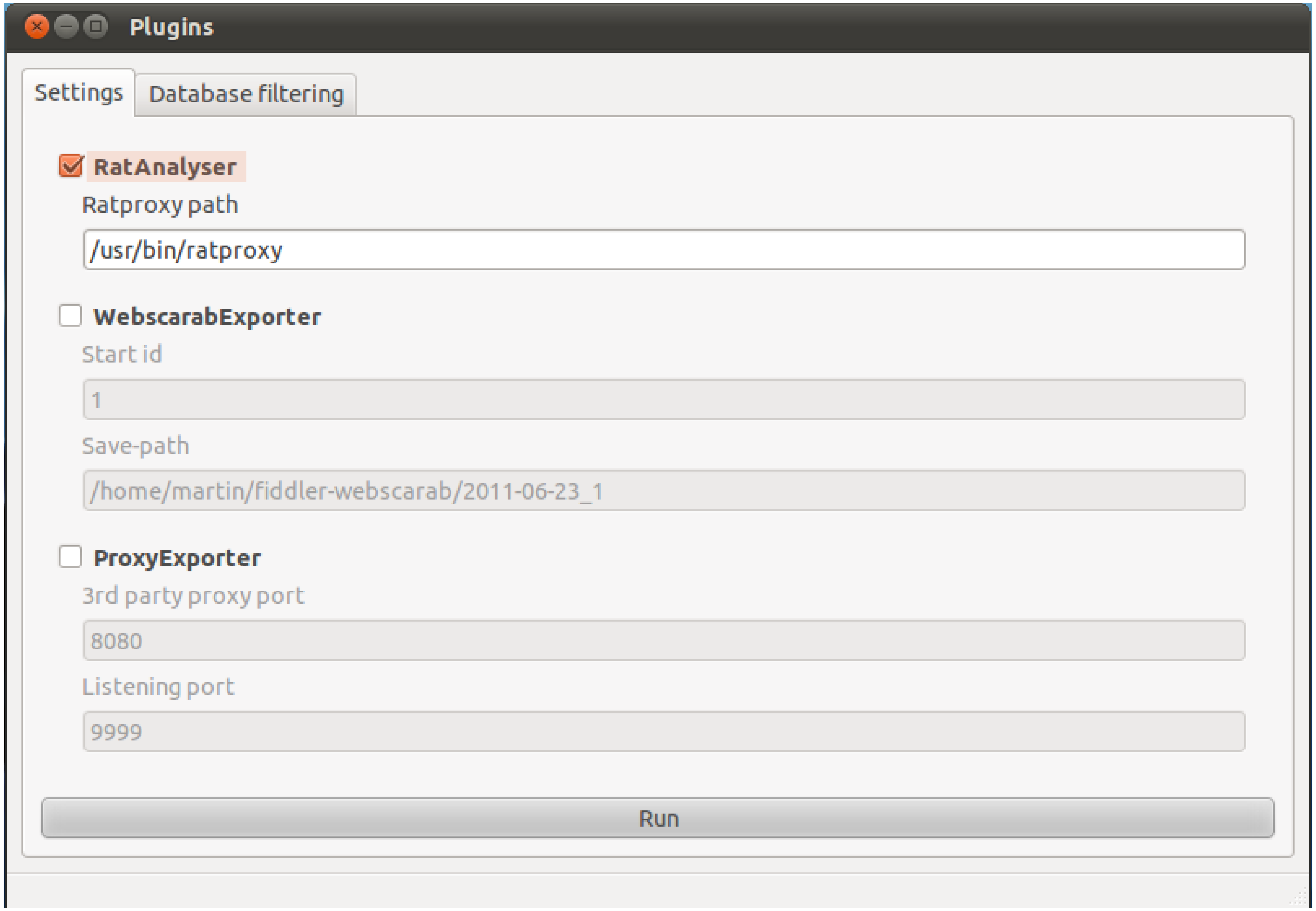
Key ['request.headers.Host']

Cond {}

Traffic analysis

- 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-webscarab/2011-06-23_1

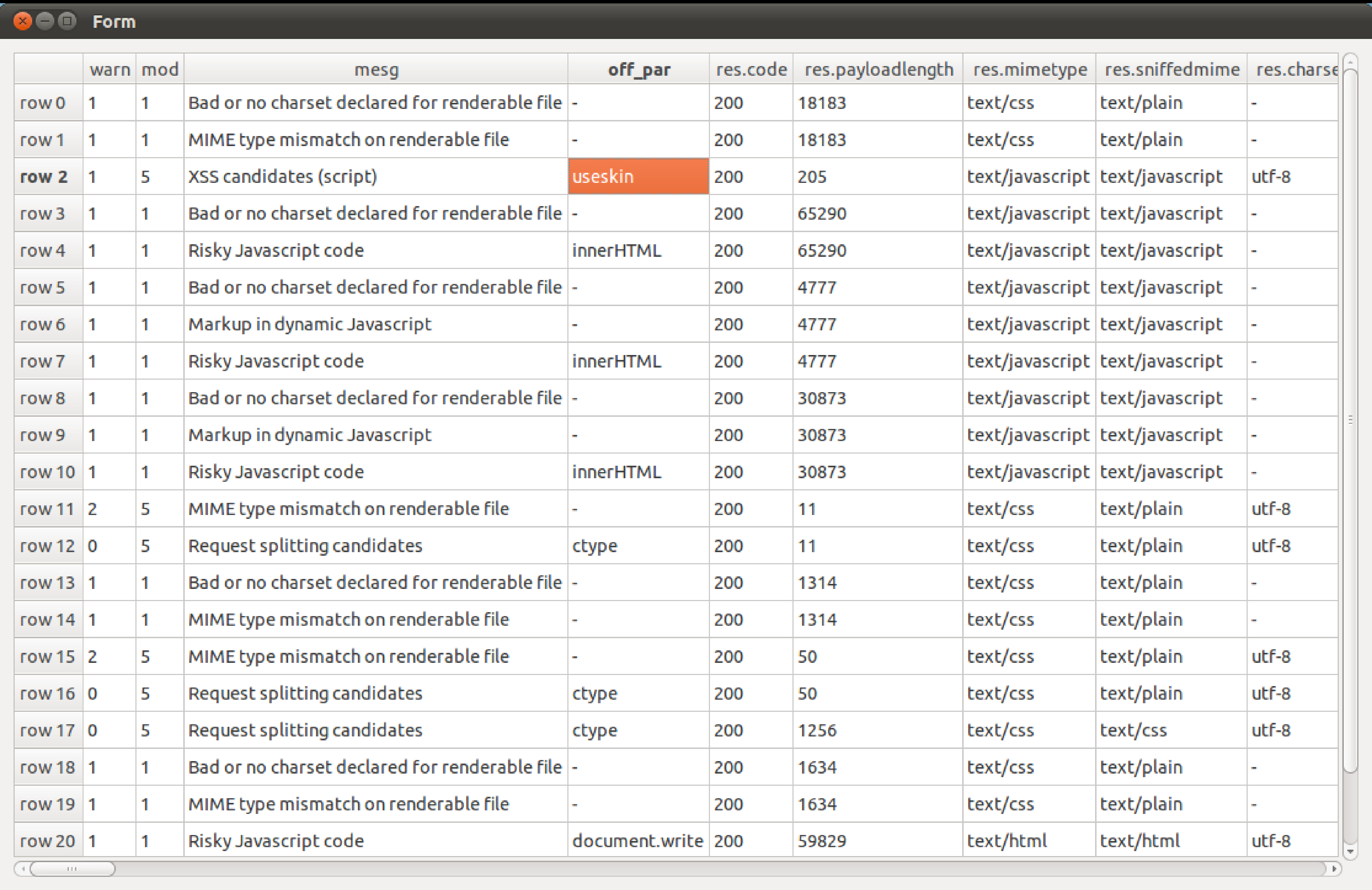
□ ProxyExporter

3rd party proxy port

8080

Listening port

9999



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

Hatkit Datafiddler

- 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

Hatkit Datafiddler

- 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
    - https://www.owasp.org/index.php/OWASP_Hatkit_Proxy_Project
    - https://www.owasp.org/index.php/OWASP_Hatkit_Datafiddler_Project

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

Hatkit Datafiddler

- 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!

Who should care?

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.

Contact

- 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?