VulnAlert Roadmap

Statement of Purpose:
To create a website that allows security aware users to be alerted to messages on selected email lists that match their specified criteria.

0.1: Basic functionality

- Database backend

- Perl Backend
  1. mail retrieval and parsing into db message format
  2. db storage
  3. db search&retrieval of messages matching criteria parameters
  4. daily rss creation (per user / from criteria)
  5. daily email alerts (per user / from criteria)

- Django/Python Frontend
  1. Basic Layout/Interface framework (using django's css as base)
  2. Basic Functionality
     1. signup
     2. login
     3. about
     4. preferences
        1. sources to search
        2. criteria with qualifiers
        3. alert preference (email/rss)
     5. View rss feeds (ie. nice interface thru the web) ?

0.2:
- flesh out site design
  - create base design layout
  - redo/tweak layout for all pages
  - add content
    - index
    - about
    - ...
  - expand message fields (date/time, from, ...)
    - add to criteria types

0.3: Improve frontend
- AJAXify
  - autosuggest for certain fields?
  - afaxify all forms
- Latest Advisories section
- Statistics section
- Search across all messages in the db
- User Specific
  - create new (mailing list) source
  - Add “Manual Run” on user page to get new alerts now
  - Add alert run frequency

        - i.e. daily/hourly/etc
    - update preferences so that user can tie particular criteria sets to a sources/delivery formats

0.4:

- Add criteria chains
    - i.e. allow users to create sets of criteria
    - keep default sources/alert method, but allow custom for each criteria chain
    - (subject *contains* "php" *and* message *startswith* ajax) and (subject *does not contain* phpnuke)
    - think best way to do this will be with a list of possible inputs
        - (,), and, or, not, contains, does not contain, is, is not, startswith, etc...
        - and use ajax to drag and drop and rearrage these inputs into a *chain* of criteria
        - trickiest part prob be validation and interpretation into sql
- allow users to add additional mailing list sources
- allow users to set up multiple RSS feeds (for differing criteria)

Todo:

- Perl Backend
    - add web retrieval / parsing
- Django Frontend
    - Latest Advisories
    - Statistics (granulate)
    - Search

Todo (more complex):

- Django/Frontend
    - create new source (mailing list)
    - Create new criteria
        - form: <select> <qualifier> < choice(s) >
            - select = one of our database fields i.e. source.name of message.subject
            - qualifier = is|is not|contains|...
            - choices = will morph based on <select> <qualifier>
                - i.e. if select=source.name and qualifier=is *or* is not will be radio
                - i.e. if select=source.name and qualifier=contains will be textinput
    - Create new criteria chain
        - this should be a set of separated criterias
            - i.e. <criteria> and <criteria2> not <criteria3>
            - add parens grouping as well

Todo (most complex):

- Turn this into a general alerting framework
    - i.e. not just security stuff
    - major addition will be the ability to choose any website and GUI-wise allow the user to manipulate it in order to create an alerting criteria
        - craigslist example:
            - 1. user will input http://portland.craigslist.com as the website
            - 2. backend will

- a.) record their actions that correspond to the criteria for their alert
  - i.e. user clicks 'for sale' link, and enters a search term
- b.) user interface will create something akin to Firefox's Web Developer tools extension's Outline Block Level elements, and the user will click on one of the instances
  - i.e. user searches for 'xbox'
    - would click the block level element:
      - Apr- 1 2 xbox 360 games for sale or trade (east vancouver)<<*electronics*
- obviously this will not be easy, however a few strategies:
  - only look at the first page
  - assume the immediate higher level element of the Block element chosen in b) is the containing element of all the items to grab
  - backend will perform search and show user a sample which will allow them to correct any mistakes that were made
  - searches can be made generic to a large extent – i.e. once one *action* has been created to search craigslist's 'for sale' page it can be made generic to a large extent and each user can simply extend the generic part for more specific queries.

- Note: as far as the ethical dilemma of page scraping – I think this should be fine as I won't be storing any original content, merely linking back to the orginal site.

FIX:

criteria_type:
  name AND search_field should be unique

criteria_qualifier should be rename just qualifier