
SCOT Documentation

Release 3.8.1

Todd Bruner, Nick Georgieff

Aug 15, 2022

Contents

1	License	3
2	Installing SCOT	5
3	Important Update	7
3.1	Minimum System Requirements	7
3.2	System Preparation	7
3.2.1	Ubuntu 14.04	7
3.2.2	Ubuntu 16.04 and CENT 7	7
3.3	install.sh Options	8
3.4	Using install.sh to upgrade	9
3.5	Configuration Files	9
4	POST Install Procedures	11
4.1	Migration	11
4.2	SSL Certs	11
4.3	Configuration Files	11
4.3.1	scot.cfg.pl	11
4.3.2	alert.cfg.pl	22
4.3.3	flair.cfg.pl	24
4.3.4	game.cfg.pl	26
4.3.5	stretch.cfg.pl	26
4.4	CRON Jobs	27
5	Migration	29
6	Save Your Old Database	31
7	SCOT Feeding	33
7.1	Email Ingest	33
7.1.1	HTML Email	33
7.2	REST interface	34
8	Overview	37
8.1	Philosophy	37
8.2	Why Use SCOT	37
8.3	Terminology	38

8.3.1	IRT	38
8.3.2	Alertgroups	38
8.3.3	Alerts	38
8.3.4	Events	38
8.3.5	Entry	39
8.3.6	Task	39
8.3.7	Entity	39
8.3.8	Flair	39
8.3.9	Intel	39
8.3.10	Guide	40
8.3.11	Signature	40
9	User Guide	41
9.1	Views	41
9.1.1	List View	41
9.1.2	Detail View	41
9.2	Alert	41
9.3	Alert Details	43
9.4	Events	46
9.4.1	Event Grid View	46
9.4.2	Event Detail View	46
9.5	Incident	49
9.6	Intel	49
9.7	Guide	49
9.8	Task	49
9.9	Signature	49
9.10	Tags	51
9.11	Flair	51
9.11.1	What the heck is Flair?	51
9.11.2	The Process	51
9.11.3	User Defined Entity	52
9.12	Entities	52
9.12.1	Entity Types	52
9.12.2	Building Additional Entity Types	53
9.13	Permissions	53
9.13.1	Default Groups and Owners	53
9.13.2	Admin Group	53
9.13.3	Note about Group Names	54
9.14	HotKeys	54
9.15	Posting a global notificaton:	54
10	Administration Guide	55
10.1	Backup	55
10.1.1	Manual Backup	55
10.2	Restore	56
10.2.1	Manual Restore	56
10.3	SSL Certs	57
10.4	GeoIP	57
10.5	Upgrading	57
10.6	CRON Entries	57
10.7	Daemons	58
10.8	Logging	58
10.9	Manual Password Reset for Local Auth	58

11	Developing For SCOT	59
11.1	SCOT Architecture	59
11.2	SCOT Directory Map	59
11.3	SCOT REST API	60
11.3.1	SCOT get API	60
11.3.2	SCOT post API	61
11.3.3	SCOT put API	62
11.3.4	SCOT delete API	62
11.4	SCOT Event Queue	62
11.5	SCOT Server	63
11.6	SCOT UI	63
12	Signatures	65
12.1	Signature	65
12.2	Signature Metadata	65
12.2.1	Signature Body Options	66
13	REVL Visualization Guide	69
13.1	Read-Eval-Viz-Loop	69
13.2	Getting Started	69
13.3	Interacting with REVL	69
13.4	Using REVL with SCOT data	70
13.5	Make a bar chart	71
13.6	Event Timing	71
13.7	Other interesting command examples	72
14	Docker for SCOT	75
14.1	Table of Contents	75
14.1.1	Overview	75
14.1.2	SCOT containers	75
14.1.3	Docker Installation	76
14.1.4	SCOT Installation	76
14.1.5	Managing the containers	76
14.1.6	Configuration	77
14.1.7	FAQ / Common Issues	78
14.1.8	TODO	78
15	Indicies and Tables	79
	Index	81

Thanks for using SCOT! Please consider joining the scot-users mailing list:

`scot-users@sandia.gov`

Send “subscribe” in the subject or message body.

You can also “follow” us on twitter at _____ .

Regardless, if you are using SCOT, please send us an e-mail at:

`scot-dev@sandia.gov`

and let us know. It helps us with our management continuing to support SCOT development.

Contents:

CHAPTER 1

License

Copyright [2016] Sandia Corporation.

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

CHAPTER 2

Installing SCOT

Important Update

As of SCOT 3.6, you can now install SCOT via docker. Please see the *scot-docker docs*

3.1 Minimum System Requirements

- Ubuntu 14.04 LTS, 16.04 LTS, or CentOS 7.
- 2 Quad Core CPU
- 16 GB RAM
- 1 TB Disk

Note: Requirements are for production use. It is quite possible to run SCOT in a small VM for testing or demonstration purposes. Your VM should have access to at least 4 GB of RAM in this case.

3.2 System Preparation

3.2.1 Ubuntu 14.04

Only limited testing on 14.04 install has been performed. 16.04 is recommended.

3.2.2 Ubuntu 16.04 and CENT 7

Install the OS. Make sure that git is installed.

Now you are ready to pull the SCOT source from GitHub:

```
$ git clone https://github.com/sandialabs/scot.git scot
```

cd into the SCOT directory:

```
$ cd /home/user/scot
```

Are you upgrading from SCOT 3.4? It is recommended to install on a clean system, however, if that is not possible you should do the following

- Backup you existing SCOT database:

```
$ mongodump scotng-prod
$ tar czvf scotng-backup.tgz ./dump
```

- delete SCOT init script and crontab entries:

```
# rm /etc/init.d/scot3
# crontab -e
```

go ahead and become root:

```
$ sudo -E bash
```

Make sure that the http_proxy and https_proxy variables are set if needed:

```
# echo $http_proxy
# export http_proxy=http://yourproxy.domain.com:80
# export https_proxy=https://yourproxy.domain.com:88
```

You are now ready to begin the install:

```
# ./install.sh 2>&1 | tee ../scot.install.log
```

Go get a cup of coffee. Initial install will download and install all the dependencies for SCOT. At the end of the install, you will be asked for a password for the admin account. Then the install script will output the status of the following processes:

- mongod
- activemq
- scot
- elasticsearch
- scfd
- scep

If any of the above are not running, you will need to debug why. Often, the following will help: (using scfd as an example)

```
# systemctl start scfd.service # systemctl status -l scfd.service
```

The messages in the stats call will be useful in determining what is causing the problem.

Once the problem has been fixed. It is safe to re-run the installer script to make sure all the initialization scripts have run correctly.

3.3 install.sh Options

SCOT's installer, install.sh, is designed to automate many of the tasks need to install and upgrade SCOT. The installer takes the following flags to modify its installation behavior:

```
Usage: $0 [-A mode] [-M path] [-dersu]
```

```
-A mode      where mode = (default) "Local", "Ldap", or "Remoteuser"
-M path      where to locate installer for scot private modules
-C           replace existing config files
-D           delete target install directory before beginning install
-d           restart scot daemons (scepdc and scfcd)
-e           reset the Elasticsearch DB
-r           delete existing SCOT Database (DATA LOSS POTENTIAL)
-s           Install SCOT only, skip prerequisites (upgrade SCOT)
-u           same as -s
```

The default install with no options will attempt to install all prerequisites or upgrade them if they are already installed. Once successfully installed, this should be rarely needed.

3.4 Using install.sh to upgrade

Sometimes you just want to refresh the SCOT software to get the latest fix or new feature. This is when you should use the -s or -u flag. If the fix or feature is in the flaring engine (scfcd) or the elasticsearch push module (scepdc) you will want to give the -d flag to restart those daemons.

3.5 Configuration Files

Configuration templates are in SCOT/install/src/scot. The files end in “cfg.pl”. You may edit them prior to install to suite your environment, or you may edit them post install in /opt/scot/etc. All changes to config files after installation will require a restart of the service for changes to take place.

Ideally, you should keep your config file modifications under software control outside of the SCOT repo directory. Here’s how to set that up.

```
# cd /home/scotadmin # ls -l drwxrwxr-x scotadmin scotadmin 4096 Jan 1 19:19 SCOT
# mkdir Scot-Internal-Modules # cd Scot-Internal-Modules # mkdir etc # cd etc # cp
../../SCOT/install/src/scot/scot.cfg.pl . # vi scot.cfg.pl # make changes # cd .. # cp
../SCOT/install/src/localinstall.sh ./install.sh # chmod +x ./install.sh
```

Place all your local configs in the Scot-Internal-Modules/etc/ directory. Modify the install.sh to suit your site. While you are it, place that directory under software control (git, etc.) and now you can make changes to your config confidently. When ever you run SCOT/install.sh the final step is a check for Scot-Internal-Modules/install.sh. If it exists, it will be executed.

POST Install Procedures

4.1 Migration

If you backed up data from your 3.4 SCOT instance and wish to restore it, you will need to follow the migration procedure Migration

4.2 SSL Certs

SCOT will generate a “snake-oil” self signed certificate upon install. It is highly recommended to replace these certificates with real certs as soon as possible.

4.3 Configuration Files

The following sections details the parameters in the varios configuration files available in SCOT. Use your favorite editor to adjust the values to your site. You can test your changes for syntax errors by using the following command:

```
$ perl -wc scot.cfg.pl
```

Correct any syntax errors reported before continuing. Typically you will need to resart SCOT for any changes to be recognized.

4.3.1 scot.cfg.pl

This config controls many aspects of the SCOT application server.

```
1 %environment = (  
2  
3     time_zone    => 'America/Denver',
```

(continues on next page)

(continued from previous page)

```
4  # scot version
5  version      => '3.5.1',
6
7  # set this to hostname of the scot server
8  servername   => '127.0.0.1',
9
10 # the mode can be prod or dev
11 mode         => 'prod',
12
13 # authentication can be "Remoteuser", "Local", or "Ldap"
14 auth_type    => 'Local',
15
16 # group mode can be "local" or "ldap"
17 group_mode   => 'local',
18
19 # default owner of new stuff
20 default_owner => 'scot-admin',
21
22 # default set of groups to apply to stuff
23 default_groups => {
24     read      => [ 'wg-scot-ir', 'wg-scot-researchers' ],
25     modify    => [ 'wg-scot-ir' ],
26 },
27
28 # the group that can perform admin functions
29 admin_group  => 'wg-scot-admin',
30
31 # filestore is where scot stores uploaded and extracted files
32 file_store_root => '/opt/scotfiles',
33
34 epoch_cols   => [ qw(when updated created occurred) ],
35
36 int_cols     => [ qw(views) ],
37
38 site_identifier => 'Sandia',
39
40 default_share_policy => 1,
41
42 share_after_time  => 10, # minutes
43
44 stomp_host      => "localhost",
45 stomp_port      => 61613,
46 topic           => "/topic/scot",
47
48 # location and site_identifier (future use)
49 location        => 'demosite',
50 site_identifier  => "demosite",
51 default_share_policy => "none",
52
53 # mojo defaults are values for the mojolicious startup
54 mojo_defaults   => {
55     # change this after install and restart scot
56     secrets => [qw(scot1sfun sc0t1sc001)],
57
58     # see mojolicious docs
59     default_expiration => 14400,
60 }
```

(continues on next page)

(continued from previous page)

```

61     # hypnotoad workers, 50-100 heavy use, 20 - 50 light
62     # hypnotoad_workers => 75,
63     hypnotoad => {
64         listen => [ 'http://localhost:3000?reuse=1' ],
65         workers => 20,
66         clients => 1,
67         proxy => 1,
68         pidfile => '/var/run/hypno.pid',
69         heartbeat_timeout => 40,
70     },
71
72 },
73
74 log_config => {
75     logger_name => 'SCOT',
76     layout => '%d %7p [%P] %15F{1}: %4L %m%n',
77     appender_name => 'scot_log',
78     logfile => '/var/log/scot/scot.log',
79     log_level => 'DEBUG',
80 },
81
82 cgi_ids_config => {
83     whitelist_file => '',
84     disable_filters => [],
85 },
86
87 # this file helps scot determine valid domain "entities"
88 # keep up to date, by creating a root cron job that does the following:
89 # @daily (cd /opt/scot/etc; export https_proxy=yourproxy.com; wget -q -N https://
90 ↪publicsuffix.org/list/public_suffix_list.dat)
91 mozilla_public_suffix_file => '/opt/scot/etc/public_suffix_list.dat',
92
93 # modules to instantiate at Env.pm startup. will be done in
94 # order of the array
95 modules => [
96     {
97         attr => 'mongo',
98         class => 'Scot::Util::MongoFactory',
99         config => {
100             db_name => 'scot-prod',
101             host => 'mongodb://localhost',
102             write_safety => 1,
103             find_master => 1,
104         },
105     },
106     {
107         attr => "es",
108         class => "Scot::Util::ElasticSearch",
109         config => {
110             nodes => [qw(localhost:9200)],
111         },
112     },
113     {
114         attr => 'esproxy',
115         class => 'Scot::Util::ESProxy',
116         config => {
117             nodes => [ qw(localhost:9200) ],

```

(continues on next page)

(continued from previous page)

```

117         max_workers => 1,
118         proto        => 'http',
119         servername    => 'localhost',
120         serverport    => 9200,
121         username      => ' ',
122         password      => ' ',
123     },
124 },
125 {
126     attr    => 'mongoquerymaker',
127     class   => 'Scot::Util::MongoQueryMaker',
128     config  => {
129     },
130 },
131 {
132     attr    => 'mq',
133     class   => 'Scot::Util::Messageq',
134     config  => {
135         destination => "scot",
136         stomp_host   => "localhost",
137         stomp_port   => 61613,
138     },
139 },
140 ## uncomment and configure if you wish to use LDAP
141 #{
142     # attr    => 'imap',
143     # class   => 'Scot::Util::Imap',
144     # config  => {
145     #     mailbox    => 'INBOX',           # mailbox, typically INBOX
146     #     hostname   => 'mail.domain.tld', # hostname of the imap server
147     #     port       => 993,               # port of the imap server
148     #     username   => 'scot-alerts',     # username of the
149     #                                     # account receiving alert email
150     #     password   => 'changemenow',     # password
151     #     ssl        => [
152     #         'SSL_verify_mode', 0        # ssl options
153     #     ],                               # see perldoc IO::SSL
154     #     uid        => 1,                 # uid IMAP config item
155     #     ignore_size_errors => 1,        # ignore_size_errors
156     #     },
157     # },
158     {
159         attr    => 'enrichments',
160         class   => 'Scot::Util::Enrichments',
161         config  => {
162             # mappings map the enrichments that are available
163             # for a entity type
164             mappings => {
165                 ipaddr    => [ qw(splunk geoip robtex_ip ) ],
166                 ipv6      => [ qw(splunk geoip robtex_ip ) ],
167                 email     => [ qw(splunk ) ],
168                 md5        => [ qw(splunk ) ],
169                 sha1       => [ qw(splunk ) ],
170                 sha256     => [ qw(splunk ) ],
171                 domain    => [ qw(splunk robtex_dns ) ],
172                 file       => [ qw(splunk ) ],
173                 ganalytics => [ qw(splunk ) ],

```

(continues on next page)

(continued from previous page)

```

174         snumber      => [ qw(splunk ) ],
175         message_id    => [ qw(splunk ) ],
176         cve           => [ qw(cve_lookup ) ],
177     },
178
179     # foreach enrichment listed above place any
180     # config info for it here
181     enrichers => {
182         geoip      => {
183             type    => 'native',
184             module  => 'Scot::Util::Geoip',
185         },
186         robtex_ip  => {
187             type    => 'external_link',
188             url     => 'https://www.robtex.com/ip/%s.html',
189             field   => 'value',
190             title   => 'Lookup on Robtex (external)',
191         },
192         robtex_dns => {
193             type    => 'external_link',
194             url     => 'https://www.robtex.com/dns/%s.html',
195             field   => 'value',
196             title   => 'Lookup on Robtex (external)',
197         },
198         splunk     => {
199             type    => 'internal_link',
200             url     => 'https://splunk.domain.tld/en-US/app/search/search?
→q=search%%20%s',
201             field   => 'value',
202             title   => 'Search on Splunk',
203         },
204         cve_lookup => {
205             type    => 'external_link',
206             url     => "https://cve.mitre.org/cgi-bin/cvename.cgi?name=%s
→",
207             field   => "value",
208             title   => "Lookup CVE description",
209         },
210     }, # end enrichment module enrichers
211 }, # end ennrichmenst config stanza
212 }, # end enrichments stanza
213 ###
214 ### uncomment and fill out this section if you want to use ldap authentication
215 ###
216 #
217 #     attr      => 'ldap',
218 #     class     => 'Scot::Util::Ldap',
219 #     config    => {
220 #         servername => 'ldap.domain.tld',
221 #         dn         => 'cn=cn_name,ou=local config,dc=tld',
222 #         password   => 'changemenow',
223 #         scheme     => 'ldap',
224 #         group_search => {
225 #             base    => 'ou=groups,ou=orgname1,dc=dcname1,dc=dcname2,
→dc=dcname3',
226 #             filter  => '(| (cn=wg-scot*))',
227 #             attrs   => [ 'cn' ],

```

(continues on next page)

(continued from previous page)

```

228 #             },
229 #             user_groups => {
230 #                 base      => 'ou=accounts,ou=ouname,dc=dcname1,dc=dcname1,
↳dc=dcname1',
231 #                 filter   => 'uid=%s',
232 #                 attrs    => ['memberOf'],
233 #             }
234 #             }, # end ldap config
235 #             }, # end ldap
236 ],
237 entity_regexes => [],
238 #
239 # form contain directions on how to build the custom incident form fields
240 # and signatures (and others later?)
241 #
242 forms      => {
243     signature => [
244         {
245             type      => "textarea",
246             key       => "description",
247             value     => '',
248             value_type => {
249                 type   => 'static',
250                 url    => undef,
251                 key    => 'description',
252             },
253             label     => "Description",
254             help      => "Enter a short description of the signature's purpose",
255         },
256         {
257             type      => "input",
258             key       => "type",
259             value     => '',
260             label     => "Type",
261             help      => "Enter the signature type, e.g. yara, snort, etc.",
262             value_type => {
263                 type   => 'static',
264                 url    => undef,
265                 key    => 'type',
266             },
267         },
268         {
269             type      => "dropdown",
270             key       => "prod_sigbody_id",
271             value     => [],
272             value_type => {
273                 type   => "dynamic",
274                 url    => '/scot/api/v2/signature/%s',
275                 key    => 'prod_sigbody_id',
276             },
277             label     => "Production Signature Body Version",
278             help      => "Select the version of the signature body you wish to be_
↳used in production",
279         },
280         {
281             type      => "dropdown",
282             key       => "qual_sigbody_id",

```

(continues on next page)

(continued from previous page)

```

283         value    => [],
284         value_type => {
285             type    => "dynamic",
286             url      => '/scot/api/v2/signature/%s',
287             key      => 'qual_sigbody_id',
288         },
289         label    => "Quality Signature Body Version",
290         help     => "Select the version of the signature body you wish to be_
↳ used in quality",
291     },
292     {
293         type    => "input_multi",
294         key      => 'signature_group',
295         value    => [],
296         value_type => {
297             type    => "static",
298             url      => undef,
299             key      => 'signature_group',
300         },
301         label    => "Signature Group",
302         help     => "Group signatures under common names",
303     },
304     {
305         type    => 'input',
306         key      => 'target.type',
307         value    => '',
308         value_type => {
309             type    => "static",
310             url      => undef,
311             key      => 'target.type',
312         },
313         label    => "Reference Type",
314         help     => "The SCOT datatype that originated this signature",
315     },
316     {
317         type    => 'input',
318         key      => 'target.id',
319         value    => '',
320         value_type => {
321             type    => "static",
322             url      => undef,
323             key      => 'target.id',
324         },
325         help     => 'The id of the SCOT datatype that originated this sig',
326         label    => "Reference ID",
327     },
328     {
329         type    => "multi_select",
330         key      => "action",
331         value    => [
332             { value => 'alert', selected => 0 },
333             { value => 'block', selected => 0 },
334         ],
335         value_type => {
336             type    => "static",
337             url      => undef,
338             key      => 'action',

```

(continues on next page)

(continued from previous page)

```

339         },
340         label    => "Action",
341         help     => "The automated action that should take place when this_
↪signature is triggered. Select multiple actions using ctrl/command key.",
342     },
343 ],
344 incident    => [
345     # substitute your text and values here to match your
346     # incident types
347     {
348         type     => "dropdown",
349         key       => 'type',
350         value     => [
351             { value => 'NONE', selected => 1 },
352             { value => 'FYI', selected => 0 },
353             { value => 'Type 1 : Root Compromise', selected => 0 },
354             { value => 'Type 1 : User Compromise', selected => 0 },
355             { value => 'Type 1 : Loss/Theft/Missing Desktop', selected => 0 },
356             { value => 'Type 1 : Loss/Theft/Missing Laptop', selected => 0 },
357             { value => 'Type 1 : Loss/Theft/Missing Media', selected => 0 },
358             { value => 'Type 1 : Loss/Theft/Missing Other', selected => 0 },
359             { value => 'Type 1 : Malicious Code Trojan', selected => 0 },
360             { value => 'Type 1 : Malicious Code Virus', selected => 0 },
361             { value => 'Type 1 : Malicious Code Worm', selected => 0 },
362             { value => 'Type 1 : Malicious Code Other', selected => 0 },
363             { value => 'Type 1 : Web Site Defacement', selected => 0 },
364             { value => 'Type 1 : Denial of Service', selected => 0 },
365             ↪selected => 0 },
366             { value => 'Type 1 : Unauthorized Use', selected => 0 },
367             { value => 'Type 1 : Information Compromise', selected => 0 },
368             { value => 'Type 2 : Attempted Intrusion', selected => 0 },
369             { value => 'Type 2 : Reconnaissance Activity', selected => 0 },
370         ],
371         value_type => {
372             type     => "static",
373             url       => undef,
374             key       => 'type',
375         },
376         label     => 'Incident Type',
377         help      => "Select best match for incident type",
378     },
379     # substitute your text and values to match your incident cats
380     {
381         type     => "dropdown",
382         key       => "category",
383         value     => [
384             { value => 'NONE', selected => 1 },
385             { value => 'IMI-1', selected => 0 },
386             { value => 'IMI-2', selected => 0 },
387             { value => 'IMI-3', selected => 0 },
388             { value => 'IMI-4', selected => 0 },
389         ],
390         value_type => {
391             type     => "static",
392             url       => undef,
393             key       => 'category',

```

(continues on next page)

(continued from previous page)

```

394     },
395     label    => 'Incident Category',
396     help     => "Select best match for incident category",
397   },
398   {
399     type      => "dropdown",
400     key       => "sensitivity",
401     value     => [
402       {value => 'NONE', selected => 1},
403       {value => 'OUO', selected => 0},
404       {value => 'PII', selected => 0},
405       {value => 'SUI', selected => 0},
406       {value => 'UCNI', selected => 0},
407       {value => 'Other', selected => 0},
408     ],
409     value_type => {
410       type      => "static",
411       url       => undef,
412       key       => 'sensitivity',
413     },
414     label     => 'Incident Sensitivity',
415     help      => "Select best match for incident sensitivity",
416   },
417   {
418     type      => "dropdown",
419     key       => "security_category",
420     value     => [
421       {value => 'NONE', selected => 1},
422       {value => 'Low', selected => 0},
423       {value => 'Moderate', selected => 0},
424       {value => 'High', selected => 0},
425     ],
426     value_type => {
427       type      => "static",
428       url       => undef,
429       key       => 'security_category',
430     },
431     label     => 'Incident Security Category',
432     help      => "Select best match for incident security category",
433   },
434   #date field for tracking when incident occurred
435   {
436     type      => "calendar",
437     key       => "occurred",
438     value     => "",
439     value_type => {
440       type      => "static",
441       url       => undef,
442       key       => 'occurred',
443     },
444     label     => "Date/Time Occurred",
445     help      => "Select Date/Time Incident Occurred",
446   },
447   {
448     type      => "calendar",
449     key       => "discovered",
450     value     => "",

```

(continues on next page)

(continued from previous page)

```

451         value_type => {
452             type     => "static",
453             url      => undef,
454             key      => 'discovered',
455         },
456         label     => "Date/Time Discovered",
457         help      => "Select Date/Time Incident was discovered",
458     },
459     {
460         type      => "calendar",
461         key       => "reported",
462         value     => "",
463         value_type => {
464             type     => "static",
465             url      => undef,
466             key      => 'reported',
467         },
468         label     => "Date/Time Reported",
469         help      => "Select Date/Time Incident was reported",
470     },
471     {
472         type      => "calendar",
473         key       => "closed",
474         value     => "",
475         value_type => {
476             type     => "static",
477             url      => undef,
478             key      => 'closed',
479         },
480         label     => "Date/Time Closed",
481         help      => "Select Date/Time Incident was closed",
482     },
483 ],
484 incident_v2 => [
485     {
486         type      => 'dropdown',
487         key       => 'type',
488         value     => [
489             # place your types here...
490             { value => "none",      selected => 1 },
491             { value => "intrusion", selected => 0 },
492             { value => "malware",   selected => 0 },
493         ],
494         value_type => {
495             type     => "static",
496             url      => undef,
497             key      => 'type',
498         },
499         label     => "Incident Type",
500         help      => <<'EOF',
501 <table>
502     <tr> <th>intrusion</th><td>An intrusion occurred</td> </tr>
503     <tr> <th>malware</th> <td>Malware detected</td> </tr>
504 </table>
505 EOF
506     },
507     {

```

(continues on next page)

(continued from previous page)

```

508         type    => "calendar",
509         key      => "discovered",
510         value    => "",
511         value_type => {
512             type    => "static",
513             url      => undef,
514             key      => 'discovered',
515         },
516         label    => "Date/Time Discovered",
517         help     => "Select Date/Time Incident was discovered",
518     },
519     {
520         type    => "dropdown",
521         key      => "severity",
522         value    => [
523             {value => 'NONE', selected => 1},
524             {value => 'Low', selected => 0},
525             {value => 'Moderate', selected => 0},
526             {value => 'High', selected => 0},
527         ],
528         value_type => {
529             type    => "static",
530             url      => undef,
531             key      => 'severity',
532         },
533         label    => 'Incident severity',
534         help     => "Select best match for incident severity",
535     },
536 ],
537 guide    => [
538     {
539         type    => "input_multi",
540         key      => "applies_to",
541         value    => '',
542         value_type => {
543             type    => "static",
544             url      => undef,
545             key      => 'applies_to',
546         },
547         label    => 'Guide applies to',
548         help     => 'Enter string matching subject that this guide applies to',
549     },
550 ],
551 },
552 dailybrief => {
553     mail    => {
554         from    => 'scot@yourdomain.com',
555         to      => 'tbruner@scotdemo.com',
556         host    => 'smtp.yourdomain.com',
557     },
558     url      => 'https://scot.yourdomain.com/'
559 },
560 incident_summary_template => <<EOF,
561 <table>
562     <tr><th>Description</th><td><i>place description of the incident here</i></td></
563     <tr>
564     <tr><th>Related Indicators</th><td><i>Place IOC's here</i></td></tr>

```

(continues on next page)

(continued from previous page)

```

564     <tr><th>Source Details</th><td><i>Place wource port, ip, protocol, etc. here</i></
    ↪td></tr>
565     <tr><th>Compromised System Details</th><td><i>Place details about compromised_
    ↪System here</i></td></tr>
566     <tr><th>Recovery/Mitigation Actions</th><td><i>Place recovery/mitigation details_
    ↪here</i></td></tr>
567     <tr><th>Physical Location of System</th><td><i>Place the city and State of system_
    ↪location</i></td></tr>
568     <tr><th>Detection Details</th><td><i>Place Source, methods, or tools used to_
    ↪identify incident</i></td></tr>
569 </table>
570 EOF
571 );

```

4.3.2 alert.cfg.pl

This config file controls how alerts are received from an IMAP server.

```

1  #####
2  ##### alert.cfg.pl
3  #####
4  ##### Used to configure the SCOT email alert input program
5  ##### bin/alert.pl which uses Scot::App::Mail
6  #####
7
8  %environment = (
9
10     ## See perl DateTime documentation for values matching your locale
11     time_zone => 'America/Denver',
12
13     ## Set up Scot Logging to your liking. See Log::Log4perl documentaton
14     ## for details on layout and log_level. By default, log_level of DEBUB
15     ## is very verbose, but is probably the level you want to be able to
16     ## figure out an error after it occurs.
17     log_config => {
18         logger_name    => 'SCOT',
19         layout          => '%d %7p [%P] %15F{1}: %4L %m%n',
20         appender_name  => 'scot_log',
21         logfile         => '/var/log/scot/scot.mail.log',
22         log_level       => 'DEBUG',
23     },
24
25     ## MODULES
26     ## Each hash in the following array, will result in an attribute
27     ## being created in the Scot/Env.pm module that points to the class
28     ## described. if you ever get a "cant find foo in Scot::Env" you might
29     ## be missing something here
30
31     modules => [
32         ## describe to SCOT how to talk to your imap server
33         {
34             attr    => 'imap',
35             class   => 'Scot::Util::Imap',
36             config  => {
37                 mailbox    => 'INBOX',          # mailbox, typically INBOX

```

(continues on next page)

(continued from previous page)

```

38         hostname => 'mail.domain.tld', # hostname of the imap server
39         port      => 993,              # port of the imap server
40         username  => 'scot-alerts',    # username of the
41                                         # account receiving alert email
42         password  => 'changemenow',    # password
43         ssl       => [
44             'SSL_verify_mode', 0      # ssl options
45         ],                             # see perldoc IO::SSL
46         uid       => 1,                # uid IMAP config item
47         ignore_size_errors => 1,      # ignore_size_errors
48     },
49 },
50 ## describe how for the Scot Perl client to find the SCOT server
51 {
52     attr    => 'scot',
53     class   => 'Scot::Util::ScotClient',
54     config  => {
55         servername => 'scotserver',
56         # username with sufficient scot perms to create alert(groups)
57         username   => 'scot-alerts',
58         # the password for that user
59         password   => 'changemenow',
60         # authentication type: RemoteUser, LDAP, Local
61         authtype   => 'Local',
62     },
63 },
64 ## mongodb connection information
65 {
66     attr    => 'mongo',
67     class   => 'Scot::Util::MongoFactory',
68     config  => {
69         db_name      => 'scot-prod',
70         host         => 'mongodb://localhost',
71         write_safety => 1,
72         find_master  => 1,
73     },
74 },
75 ## ActiveMQ connection info
76 {
77     attr    => 'mq',
78     class   => 'Scot::Util::Messageq',
79     config  => {
80         destination => "scot",
81         stomp_host  => "localhost",
82         stomp_port  => 61613,
83     },
84 },
85 ## Elasticsearch connection info
86 {
87     attr    => 'es',
88     class   => 'Scot::Util::ElasticSearch',
89     config  => {
90         nodes      => [ qw(localhost:9200) ],
91         max_workers => 1,
92     },
93 },
94 ],

```

(continues on next page)

(continued from previous page)

```

95  ## parser_dir is where to find the modules that can parse the emails
96  parser_dir => '/opt/scot/lib/Scot/Parser',
97  ## alert.pl can utilize rest or direct mongo connection to input data
98  get_method    => "mongo",      # other value is "rest"
99  ## leave_unseen = 1 means SCOT will leave emails marked "unread"
100  ## leave_unseen = 0 means SCOT marks emails read after processing
101  leave_unseen  => 1,
102  # interactive => [ yes | no ]
103  # pauses processing after each message and writes to console
104  interactive    => 'no',
105  verbose        => 1,
106  # max_processes => 0 to positive int
107  # number of child processes to fork to parse messages in parallel
108  # 0 = disable forking and do all messages sequentially
109  # recommendation is 5-10 in production, 0 for testing.
110  max_processes  => 0,
111  # fetch_mode    => [ unseen | time ]
112  # unseen looks for unseen messages via imap protocol
113  # time gets all message since a given time
114  # both modes check unique message_id and will not reprocess something
115  # already in SCOT database
116  fetch_mode     => 'unseen',
117  # since         => { unit => amount }
118  # hashref where key is the unit [ day, hour, minute ]
119  # amount is integer value
120  # used by time fetch_mode
121  since          => { hour => 2 },
122  # approved_alert_domains => [ 'domain1\.org', ... ]
123  # only domains listed in this array can send email to scot
124  # periods need to be escaped by \
125  approved_alert_domains => [ 'domain\.tld' ],
126  # approve_accounts => [ 'user@email.addr' ];
127  # account in this domain can also send to scot
128  approved_accounts  => [ 'user@server.domain.tld' ],
129
130  # future use:
131  location          => "scot_demo",
132  site_identifier    => "scot_demo",
133  default_share_policy => "none",
134 );

```

4.3.3 flair.cfg.pl

The Flair app automatically detects enties, see Entities. This config file look like:

```

1  %environment = (
2      max_workers => 4,
3      location    => 'sn1',
4      fetch_mode  => 'mongo',
5      mozilla_public_suffix_file => '/opt/scot/etc/public_suffix_list.dat',
6      log_config => {
7          logger_name => 'flair',
8          layout      => '%d %7p [%P] %15F{1}: %4L %m%n',
9          appender_name => 'regex_log',
10         logfile      => '/var/log/scot/flair.log',

```

(continues on next page)

(continued from previous page)

```

11     log_level    => 'DEBUG',
12 },
13 default_groups => {
14     read    => [ 'wg-scot-ir'],
15     modify => [ 'wg-scot-ir'],
16 },
17 default_owner  => 'scot-admin',
18 img_dir        => '/opt/scot/public/cached_images',
19 html_root      => '/cached_images',
20 modules => [
21     {
22         attr    => 'mongo',
23         class   => 'Scot::Util::MongoFactory',
24         config  => {
25             db_name      => 'scot-prod',
26             host         => 'mongodb://localhost',
27             write_safety => 1,
28             find_master  => 1,
29         },
30     },
31     {
32         attr    => 'mq',
33         class   => 'Scot::Util::Messageq',
34         config  => {
35             destination => 'scot',
36             stomp_host  => 'localhost',
37             stomp_port  => 61613,
38         },
39     },
40 ],
41 local_regexes => [
42     {
43         type    => 'snumber',
44         regex   => '\b([sS][0-9]{6,7})\b',
45         order   => 501,
46         options => { multiword => "no" },
47     },
48     {
49         type    => 'sandia_server',
50         regex   => '\bas[0-9]+sntl(lx|nt)\b',
51         order   => 500,
52         options => { multiword => "no" },
53     },
54 ],
55 lwp            => {
56     use_proxy    => 1,
57     timeout      => 10,
58     ssl_verify_mode => 1,
59     verify_hostname => 1,
60     ssl_ca_path  => '/etc/ssl/certs',
61     proxy_protocols => ['http', 'https'],
62     proxy_uri    => 'http://proxy.sandia.gov:80',
63     lwp_ua_string => "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1)
→ AppleWebKit /537.36 (KHTML, like Gecko) Chrome/41.0.2227.1 Safari/537.36",
64 },
65 );

```

4.3.4 game.cfg.pl

This controls aspects of the gamification system.

```
1 %environment = (  
2     log_config => {  
3         logger_name    => 'SCOT',  
4         layout         => '%d %7p [%P] %15F{1}: %4L %m%n',  
5         appender_name  => 'scot_log',  
6         logfile        => '/var/log/scot/scot.game.log',  
7         log_level      => 'DEBUG',  
8     },  
9     days_ago    => 30,  
10    modules => [  
11        {  
12            attr    => 'mongo',  
13            class   => 'Scot::Util::MongoFactory',  
14            config  => {  
15                db_name    => 'scot-prod',  
16                host       => 'mongodb://localhost',  
17                write_safety => 1,  
18                find_master => 1,  
19            },  
20        },  
21    ],  
22 );
```

4.3.5 stretch.cfg.pl

This controls aspects of the elastic search input system.

```
1 %environment = (  
2     time_zone    => 'America/Denver',  
3     max_workers  => 1,  
4     log_config  => {  
5         logger_name    => 'SCOT',  
6         layout         => '%d %7p [%P] %15F{1}: %4L %m%n',  
7         appender_name  => 'scot_log',  
8         logfile        => '/var/log/scot/scot.stretch.log',  
9         log_level      => 'DEBUG',  
10    },  
11    max_workers  => 2,  
12    stomp_host   => 'localhost',  
13    stomp_port   => 61613,  
14    topic        => '/topic/scot',  
15    default_owner => 'scot-admin',  
16    modules => [  
17        {  
18            attr    => 'es',  
19            class   => 'Scot::Util::ElasticSearch',  
20            config  => {  
21                nodes => [ qw(localhost:9200) ],  
22            },  
23        },  
24        {  
25            attr    => 'scot',
```

(continues on next page)

(continued from previous page)

```

26     class => 'Scot::Util::ScotClient',
27     config => {
28         servername => "localhost",
29         username   => "scot-alerts",
30         password   => "changemenow",
31         auth_type  => "basic",
32     },
33 },
34 {
35     attr   => 'mongo',
36     class  => 'Scot::Util::MongoFactory',
37     config => {
38         db_name      => 'scot-prod',
39         host          => 'mongodb://localhost',
40         write_safety => 1,
41         find_master  => 1,
42     },
43 },
44 ],
45 # future use:
46 location      => "scot_demo",
47 site_identifier => "scot_demo",
48 default_share_policy => "none",
49 );

```

4.4 CRON Jobs

The `/opt/scot/alert.pl` program that reads in alerts from the IMAP server needs a crontab entry. It is recommended to run this every 2 to 5 minutes. Here's the crontab entry:

```
* /5 * * * * /opt/scot/bin/alert.pl
```

Automating SCOT backups are a good idea as well:

```
0 3,12,20 * * * /opt/scot/bin/backup.pl    # backup scot at 3am 12 noon and 8pm
```

The `game.pl` job populates the analyst leaderboard:

```
30 4 * * * /opt/scot/bin/game.pl
```

The `metric.pl` job calculates response time metrics:

```
15 2 * * * /opt/scot/bin/metric.pl
```

Migration

Many parts of the database have changed from the 3.4 version of SCOT and it is necessary to migrate that data if you wish to continue to access that data in SCOT 3.5. We have developed a migration program to assist with this task.

We are assuming that your Mongo instance has sufficient space to keep the 3.4 database and the new 3.5 database on it during the migration. The 3.5 instance will be roughly the same size as the 3.4 instance.

Depending on the amount of data you need to migrate, this process could take a while. It is hard to estimate, but from my experience, the migration will process a million alerts in 24 hours.

Migration is designed to be parallelized. Not only can each collection be migrated concurrently, but you can also specify the number of processes to operate on each collection. For example, if you have 1 million alerts to process, you can specify 4 processes to work on alerts and each process will migrate 250,000 alerts. Unless you have very large databases, my recommendation is to allow a single process to work on each collection because this will make it easier to detect and correct any anomalies in the data migration.

The migration command:

```
$ cd /opt/scot/bin
$ ./migrate.pl alert 2
```

would begin migrating alerts from the 3.4 database using two processes.

Best practice in migration is to open a terminal for each collection, start tmux or screen, and then start the migration for a collection. Extensive logging is performed in `/var/log/scot/migration.alert.log`, where `alert` is the actual collection being migrated. Pro tip: `'grep -i error /var/log/scot/migration*'`

The list of collections to migrate:

```
# alertgroup # alert # event # entry # user # guide # handler # user # file
```

If you wish for totally hands off operation, do the following:

```
$ cd /opt/scot/bin
$ ./migrate.pl all
```

This will sequentially migrate the collections listed above. The migration will take a bit longer, though.

NOTE: Migration assumes that the database to be migrated is on the same database server as the new server. So in other words, if you are installing SCOT 3.5 on a new system, and want to migrate your database to that server, you will need to use the mongodump and mongorestore to move the old database to the new server first.

Example Migration:

```
$ ssh oldscot
oldscot:/home/scot> mongodump scotng-prod
...
oldscot:/home/scot> tar czvf ./scotng-prod.tgz ./dump
...
oldscot:/home/scot> scp scotng-prod.tgz scot@newscot:/home/scot
...
oldscot:/home/scot> exit
$ ssh newscot
newscot:/home/scot> tar xzvf ./scotng-prod.tgz
...
newscot:/home/scot> mongorestore --db scotng-prod ./dump/scotng-prod
...
newscot:/home/scot> cd /opt/scot/bin
newscot:/opt/scot/bin> ./migrate.pl all
```

CHAPTER 6

Save Your Old Database

The migration tool has been tested, but as with any process that operates on user data, things can happen. The only defense is to save a copy of the last 3.4 SCOT database backup.

or How to get alerts into SCOT.

SCOT is designed to receive data from detection systems in two ways.

7.1 Email Ingest

Many detection systems have the ability to generate email alerts. For these systems, you should configure those alerts to go to an email inbox that SCOT will have permission to access, e.g. scot-alerts@yourdomain.com. The `Scot alert.pl` program upon start will query that mailbox for messages. Configuration of the `alert.pl` program is handled in the `/opt/scot/etc/alert.cfg.pl` file.

Email ingest has many advantages, such as a flexible and resilient method of message delivery. To use this method, though, you must create a Parser for the type of Email message. SCOT comes with sample parsers for Fireeye, Microsoft Forefront, Sourcefire, and Splunk emails. These parsers, located in `/opt/scot/lib/Scot/Parser` should provide a template to create your own parsers for the email from your detection system.

The following section will show how the `Scot::Parser::Splunk (/opt/scot/lib/Scot/Parser/Splunk.pm)` module parses an HTML formatted email.

7.1.1 HTML Email

When creating a Parser module, you must first implement a “will_parse” function, that will return true if your parser can parse the e-mail message. Looking at `Splunk.pm`’s `will_parse` function, we see the following:

```
if ( $subject =~ /splunk alert/i ) {  
    return 1;  
}
```

This means that if the subject of the email contains the case insensitive phrase “splunk alert”, tell the alert ingester that this is the parsing module to use to parse this email.

Another way to test would be to check the address of the email sender like this:

```
my $from = $href->{from};
if ( $from =~ /splunk\@yourdomain.com/i ) {
    return 1;
}
```

Remember, the `will_parse` should return “false” (undef in Perl) if this parser can not parse the email.

The next function that must be implemented is the “`parse_message`” function. It is passed a hash reference that contains the email’s subject, `message_id`, plain text of email, and html version of email (if it exists). At this point you have to refer to sample parsers provided on ideas how to parse your message. If you get stuck, please feel free to ask for help on our [github page](#).

The result of the parsing should be a hash that looks like the following:

```
%json = (
  data    => [
    { column1 => dataval11, column2 => dataval12, ... },
    { column1 => dataval21, column2 => dataval22, ... },
    ...
  ],
  columns => [ column1, column2 ... ],
);
```

Note: the hash may contain other keys besides `data` and `columns` depending on that data you want to extract from the email.

7.2 REST interface

OK, you’ve looked at the parsers, and for whatever reason you decide that creating your own is not the way you wish to go. In that case, the REST API is the way for you to go. Essentially, you will need a username and password, or an apikey from SCOT. Then you will have to configure your detector to POST to SCOT via the API. Alternatively, you could write your own wrapper to do the REST calls.

Here’s a sample curl command to insert an alertgroup:

```
curl -H "Authorization: apikey $SCOT_KEY" -H "Content-Type: application/json" -X POST -d '{
  "source": [ "email_examinr" ],
  "subject": "External HREF in Email",
  "tag": [ "email href" ],
  "groups": {
    "read": [ "wg-scot-ir" ],
    "modify": [ "wg-scot-ir" ],
  },
  "columns": [ "MAIL_FROM", "MAIL_TO", "HREFS", "SUBJECT" ],
  "data": [
    {
      "MAIL_FROM": "amlegit@partner.net",
      "MAIL_TO": "br549@watermellon.com",
      "HREFS": "http://spmiller.org/news/please_read.html",
      "SUBJECT": "Groundbreaking research!"
    },
    {
      "MAIL_FROM": "scbrb@aa.edu",
      "MAIL_TO": "tbruner@watermellon.com",
      "HREFS": "https://www.aa.edu/athletics/schedule",
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
        "SUBJECT": "Schedule for next week"
    },
    {
        "MAIL_FROM": "bubba@bbn.com"
        "MAIL_TO": "fmilszx@watermellon.com",
        "HREFS": "https://youtu.be/JAUoeqvedMo",
        "SUBJECT": "Can not wait!"
    }
],
}' https://scot.yourdomain.com/scot/api/v2/alertgroup
```


Cyber security incident response can be a dynamic and unpredictable process. What starts out as a simple alert may lead a team down a rabbit hole full of surprises. The Sandia Cyber Omni Tracker, SCOT, is a cyber security incident response (IR) management system designed by cyber security incident responders to provide a new approach for managing security alerts, including coordinating team efforts, capturing team knowledge, and analyzing data for deeper patterns. SCOT integrates with existing security applications to provide a consistent, easy to use interface that enhances analyst effectiveness.

8.1 Philosophy

When Sandia's IR team was looking for a system to capture its work product, they already had a great deal of experience with existing products. RT and its IR extensions, TRAC and Remedy were products that had been tested, along with several large SEIMS. For various reasons, none of these tools were adopted by the team. For some, the complexity of the tool prevented an already overloaded team from utilizing it. For others, there was a mismatch between what would be required of the team in order to use it and the reality of how the current IR process worked. From these frustrations and to fill the IR team's need, SCOT was developed with these principles in mind:

- SCOT should require minimal training to use and understand
- SCOT should aim to always improve the effectiveness and efficiency of the IR analyst
- SCOT should reward the IRT for its use.

8.2 Why Use SCOT

- Designed to be easy to use, learn, and maintain.
- Real-time updating keeps team in sync and their efforts coordinated.
- Automated detection and correlation of common cyber security indicators such as IP addresses, domain names, file hashes and e-mail addresses.
- Alert centralization from a wide range of security systems.

- Extensible plugin infrastructure to allow additional automated processing.
- Full Text searchable knowledge base that allows the entire team to easily discover and learn from past cyber security events.
- Open Source. Hack it up to meet your needs. (Please share!)

Our current users state:

- SCOT just works and never slows me down.
- I'm putting more and more of my investigation notes into SCOT. It has paid off tremendously for me and helped me discover several non-obvious patterns.
- Aside from my e-mail client, it's the one application that is always on my screen.
- Give up SCOT? I'd leave incident response first!

8.3 Terminology

Having a common vocabulary is very important to improve understanding. So many terms are overloaded with meanings or have different connotations to different teams. The following sections define terms that SCOT uses and the interpretation of those terms in SCOT's context.

8.3.1 IRT

Incident Response Team, a group of analysts that look for and respond to security conditions.

8.3.2 Alertgroups

Many security applications have the ability to send data to other systems when certain conditions are met. These methods often vary widely. There is one commonality among most applications, however, and that is E-mail.

One of the ways SCOT can accept information from other systems is by setting up an IMAP inbox that these systems can send E-mail to. SCOT then periodically checks that inbox, and ingests any new email messages it finds. Creation of Alertgroups are possible via the REST API as well.

The E-mail messages may often contain several "rows" of data. For example, several IDS alerts could be bundled up into a single E-mail message. This grouping of alerts is called an "Alertgroup." In other words, an Alertgroup consists of one or more "Alerts" that were entered into SCOT at the same time.

8.3.3 Alerts

An individual "row" from an "Alertgroup." An alert can consist of any number of key-value pairs. An example alert could be the output of an IDS system that shows a specific rule had triggered and the relevant details of that triggering.

The "Alert" is the starting point for the SCOT processing workflow. Analysts triage the incoming alerts and close or promote those alerts into "Events."

8.3.4 Events

Typically, an experienced analyst should be able to determine if there is something interesting about an alert in a few minutes time. If further investigation is merited, the alert should be promoted to an "Event."

The Event is where the majority of the IRT's work will be recorded. Promotion to an "Event" is a signal for the IRT that there might be something that needs the team's attention.

Each analyst is capable of adding information into "Entries" and those notes are instantly available to the rest of the team (assuming proper authorization).

The event lifecycle includes research and collection of data about the event from the resources available to the team. A summary may be created to allow others coming late to the party to get up to speed. Mitigation activities can be documented as well. Some Events are so serious in nature or group of Events can be aggregated into an "Incident." Finally, Events can be closed as a signal that no further activity on that Event is expected. If that expectation proves false, the team can still add entries or even reopen the event.

8.3.5 Entry

An entry is a chunk of text or graphic data that is stored in SCOT. Entries are associated with Alertgroups/Alerts, Events, Incidents, Intel, and Entities. The data entered into a Entry is scanned for "Entities" and "Flair" is applied to entry to aid the analysts.

Entries are "owned" by the creator of the entry, but may be edited by anyone in the modify group. Any entry can be "promoted" to Summary status and will appear at the top of the detail page. Entries may also be designated as a "Task."

8.3.6 Task

An Entry that has been marked as a task. Tasks track "to^do" items and serve as reminders to do things as well as requests for help from your team. Tasks can not be assigned to others, as a user must "take" ownership of a task. This prevents people from claiming not to see tasks in their "queue" and promotes a proactive approach to team coordination.

8.3.7 Entity

Entities are a growing list of string fragments that SCOT can detect within entry data. Examples include IP addresses, E^mail addresses, Domain names, MD5 hashes, SHA1 hashes, SHA256 hashes, E^mail message id strings, and filenames with common extensions. Entities can be thought of as IOC's.

Once detected in Entry data, Entities are cross referenced with the entirety of SCOT's historical records. Various data enrichment activities can also take place based on the type of Entity. Finally, the source Entry data is rewritten to "Flair" or highlight these strings.

8.3.8 Flair

Flair is the highlighting and decoration of Entities. First Entities are wrapped in a span that highlights them in yellow. Next various icons are attached to the span that represent the number of times this entity has appeared in SCOT, flags for geoup data, if additional notes about the Entity are available, and others that you can implement.

8.3.9 Intel

Often, IRTs receive information about threats, reports from other entities, and other general information that can be thought of as Intel. Storing these items within SCOT allows SCOT to detect entities, flair them, and cross reference these entities in existing and future Alerts and Entries.

8.3.10 Guide

Guides are mini instruction manuals that help your analysts know how to respond to incoming Alertgroups. Guides are linked to the Alertgroup through the Subject value of the Alertgroup.

8.3.11 Signature

Do you wish there was a place to store all your Yara, Snort, and other detection signatures? We look no further. Here you can store, discuss, and revise your signatures and link them to your activities tracked in SCOT. You can use the REST API to serve these signatures to your detection systems.

SCOT is divided into several main sections. Alerts flow into SCOT and are processed in the Alert tab. Some subject of those Alerts will be promoted into Events, where the team records the results of their investigations. A subset of those Events will be important enough to be promoted into Incidents. Simultaneously, new Intel reports will be created and added to SCOT. The sections below describe how to use each section.

9.1 Views

9.1.1 List View

A grid that can be filtered that display many things (alertgroups, events, etc.)

9.1.2 Detail View

The section of the page that displays details about the thing selected in the the List view.

9.2 Alert

To access the Alert Grid, click on the “Alert” button in the Navigation Bar at the top. The grid will appear in which we see a list of alerts that have come in and need to be triaged. Each row represents a group of alerts that came in together and are possibly related. Let’s go over what each of the columns in the grid means.

Status The status can be either *open* (no action taken), *closed* (no action needed), or *promoted* (more analysis needed). All alertgroups come into SCOT as “open”.

Sources The “sources” column identifies the systems/organizations/processes that are responsible for creating the alert.

Subject The subject is a quick description of the alertgroup. If the alert came in through email, this is the email subject.

Tags Tags are seen as a catchall and are useful in subsequent searches for alerts with a specific set of tags.

Views You can also filter by the number of times a particular alert has been viewed to know if anyone else on your team has looked at it.



Each column contains a textbox to filter the grid results. Just enter in a filter and press 'Enter' on your keyboard to activate the filter. You can also click on a column above the filter textbox to sort by that column. We can see the default sort order is by 'created' which is indicated by the chevron next to the column name.

A summary of alert status within an alertgroup can be quickly determined by glancing at the status column. The legend below explains the shapes/colors used.

9.3 Alert Details

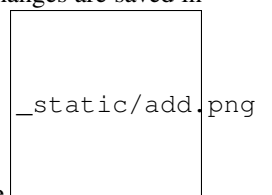
Let's look at the contents of an alert by clicking on one of the rows in the Alert Grid. Note:



_static/alert_details.png

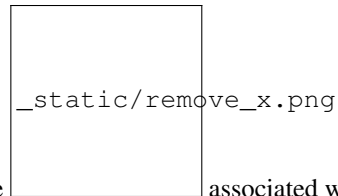
Inside the Details view, we see the header (black background, white text). The header allows us to edit basic metadata about the alert such as the subject, close/open it, add/remove tags and sources.

To change the subject, click in the black subject box and edit like you would any other textbox; changes are saved in



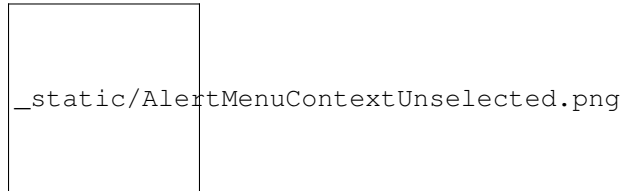
_static/add.png

real time. To change the status of an alert, click on the button titled “open”. To add a tag, click on the



button and start typing. To remove a tag, click the associated with it.

Now let's look at the context sensitive command bar located directly below the header.



Toggle Flair Toggle the display of Flair.

Reparse Flair Mark the Alertgroup for re-parsing by the Flair engine

Guide Display the Guide for this Alertgroup type.

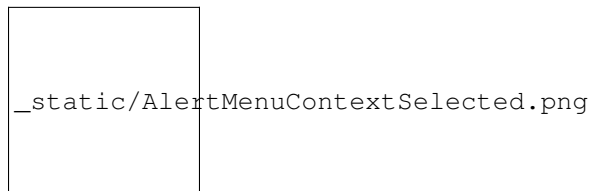
View Source View the raw unparsed versions of the Alertgroup

View Entities View the list of discovered Entities in this Alertgroup

Viewed by History See who viewed this alertgroup.

Alertgroup History See the history of actions taken on this Alertgroup

After clicking on one or more Alerts in the detail view, the alertgroup context menu changes to



Open Selected Change status of selected alerts to “open”

Closed Selected Change status of selected alerts to “closed”

Promote Selected Change status of the selected alerts to “promoted”

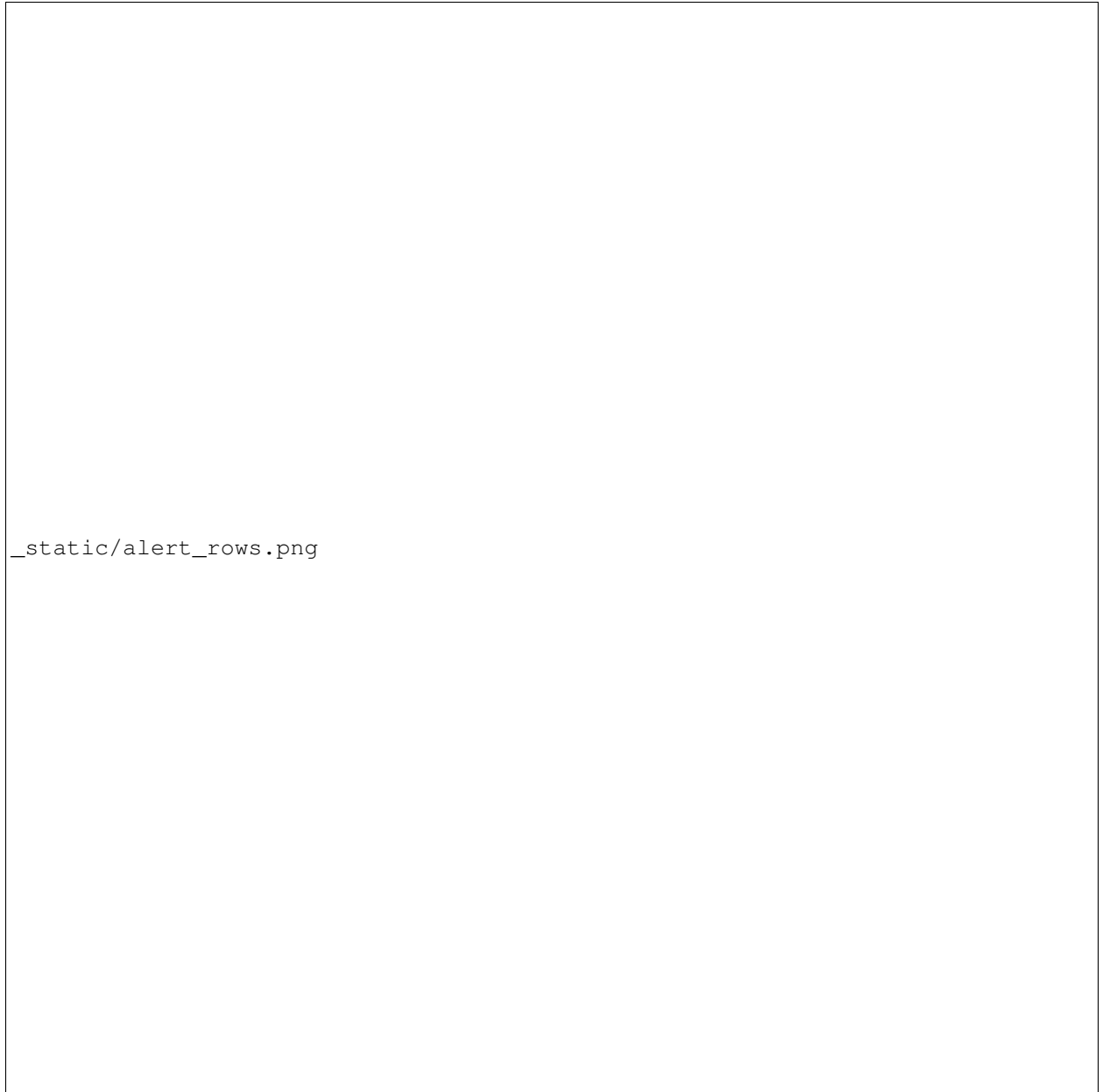
Add Selected to Existing Event Add the selected alerts to an existing event. You will need to know the event number.

Add Entry Add an Entry to the selected alerts

Upload File Upload a file and associate it with the selected alerts

Export to CSV Export the detail view into a CSV file and download it to the desktop.

Let's look at the actual alerts in this alertgroup now. Each row in the table represents an alert, which may or may not be related to the other alerts. You can select one or more rows by clicking on them and utilizing the Shift and Ctrl keys as you would when selecting files in Windows Explorer. Selected row(s) are highlighted in green.



_static/alert_rows.png

For each alert, we want to answer the following:

- Is this a false positive?
- Do we have enough information to continue?
- Should we investigate further, or is this known to be malicious?

If this is a **false positive**, we can go ahead and close the alert by first selecting it, then choosing the “Close Selected” button from the context sensitive menu above. The status for this alert will change to closed and this status change will appear instantly on the screen of all other analysts.

If there is not **enough information** to continue, but there is some information about this alert that could be helpful to another analyst, select the alert and click “Add Note”. In the new textarea that pops up, type your note (full HTML support) and click “Save”.

If we need to **investigate further**, select the row(s) in question and click ‘Promote Selected’. This will create a new

Event where you can document your findings and collaborate with other analysts on your team. This event is linked back to the original alert, so no data is lost.

9.4 Events

This is where the fun begins! Promotion to an Event, is a signal to the team that the promoting analyst thinks that there is something in this alert that merits the attention of the team. During this phase, the team is investigating the alert, dropping their results into Entries, creating a summary, asking each other for help via Tasks, and tagging the results.

9.4.1 Event Grid View

The Event grid allows you to view sets of Events and to filter those sets in various ways.

9.4.2 Event Detail View

Event Id Each Event has a unique integer id assigned to it.

Subject The team can give each event a subject. By default, it will be the same as the alertgroup.

Status The event can be “open”, “closed”, or “promoted.” Many events can remain “open.” Some people get hung up on an event being open for months, but it really only means that the team thinks that there may be more to come on this event in the future. “Closed” should be reserved for this is no longer actively being worked. Promoted gets assigned if the Event becomes an Incident. The status is easily changed using the pull down.

Owner Every Event has an owner.

Updated This the time of the last update of this event.

Promoted From Links back to the alerts that originated this Event.

Tags Add, Delete, or Edit the set of tags applied to this event.

Source The Source of this Event. Analysts can add to this.

Add Entry This is how the analyst creates an new entry box to enter information about the Event.

Upload file Upload a file and associate that file with this event.

Toggle Flair Turn Flair on or off

Viewed by History See who has been viewing the event

Event History See the changes that have happend to the event

Permissions View and change the groups that have read and write access to this event

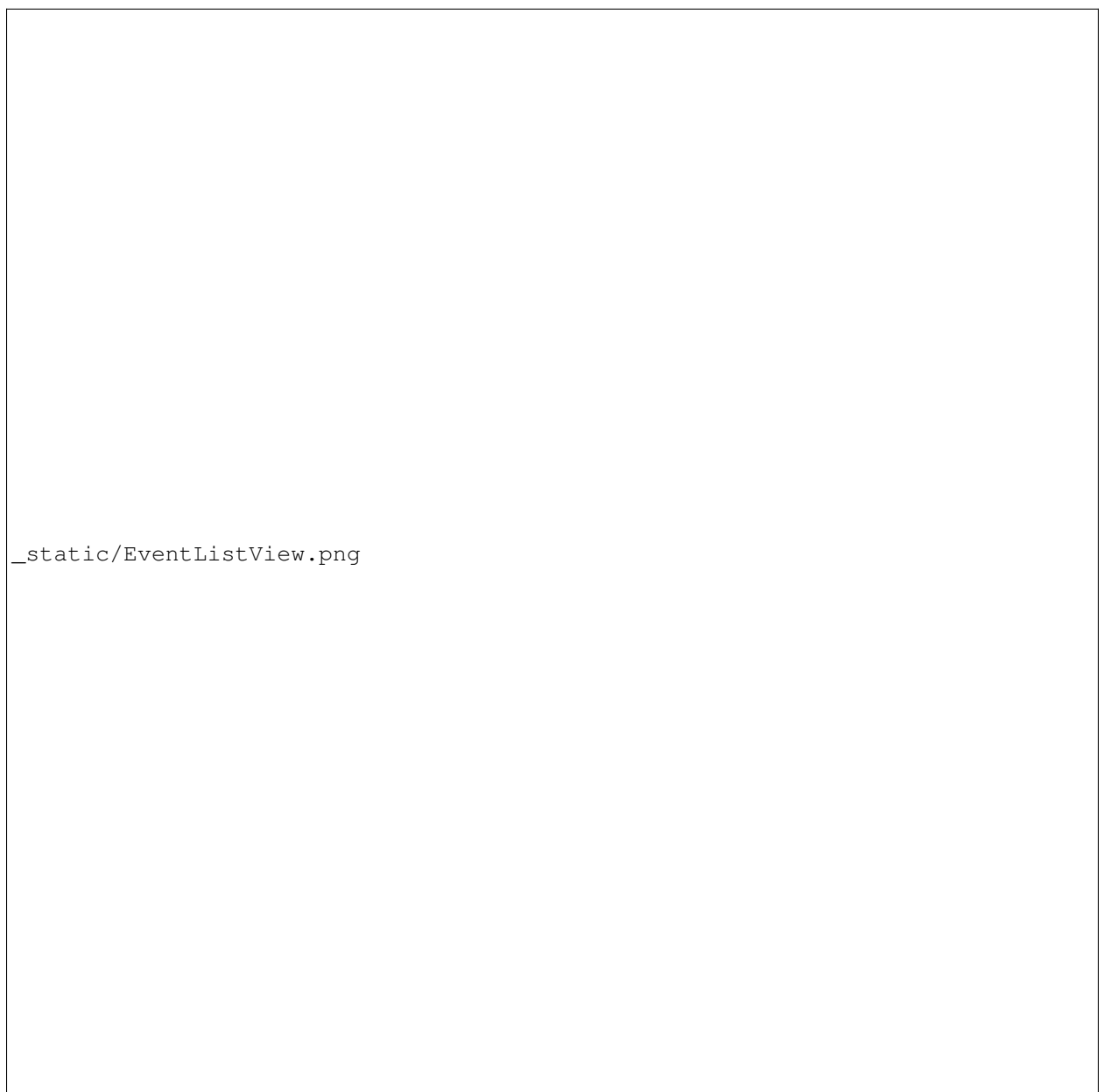
View Entities See all discovered Entities

Promote to Incident promote the event up the food chain

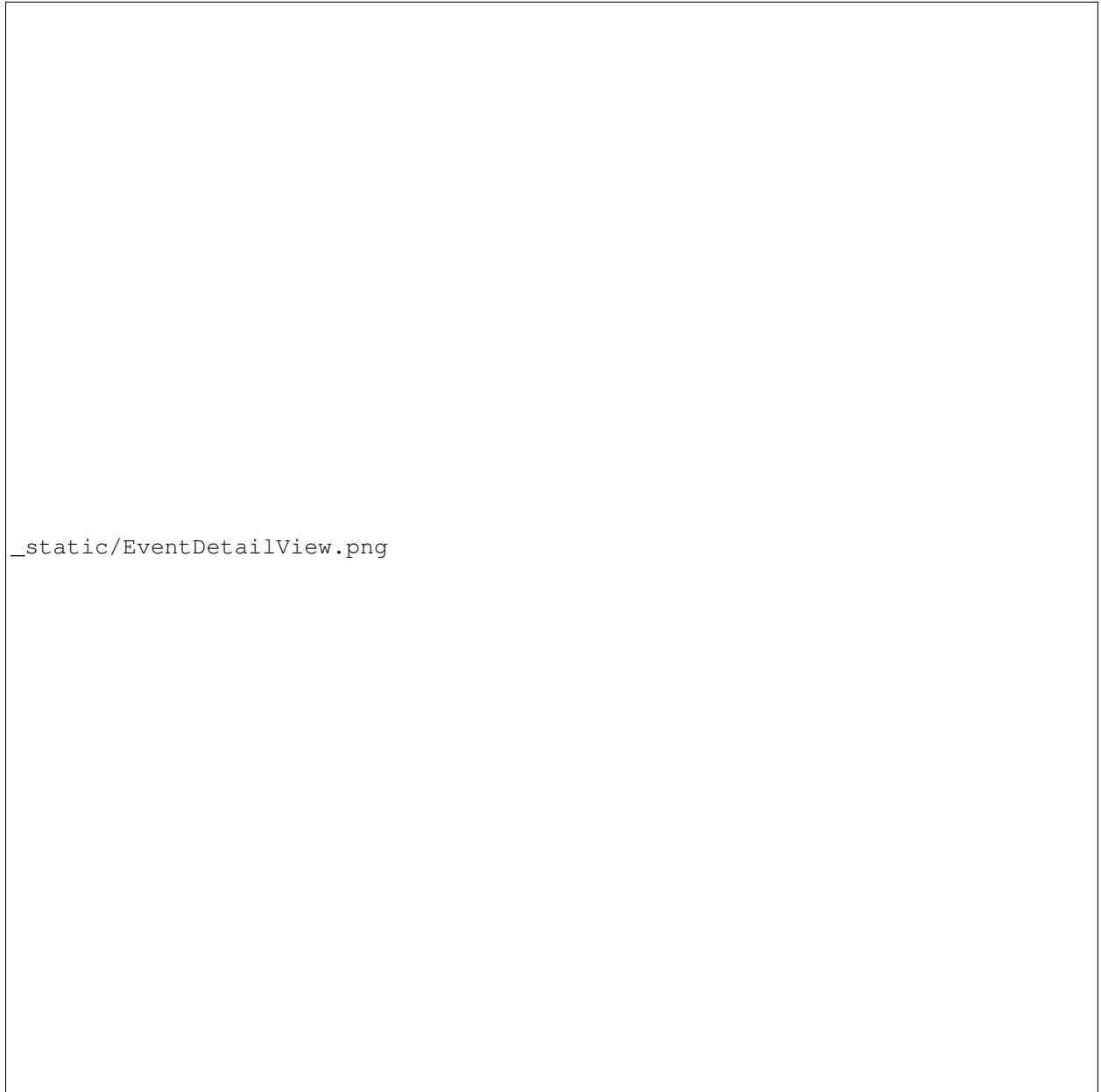
Delete Event Delete this event.

The first boxes after the command buttons are known as Entries. There are several types of entry box. The first entries to appear will be Summary entries, if they exists. Summary entries are highlighted in light yellow. In the example above, a summary entry has not yet been created.

Alert recap Entries usually appear next. These entries contain a copy of the alert data so the analysts does not have to switch back to alert view to see the details.



`_static/EventListView.png`



_static/EventDetailView.png

Other entries follow and contain data input by the analyst and, in the future, from automated processes. Entries with a red title bar are Tasks that have yet to be marked as completed. Green title bars denote completed tasks.

9.5 Incident

Incidents are groupings of Events. One way to use Incidents is to track the Events that rise in importance that they need to be reported to another organization or to higher management. Incidents track metadata such as type of incident, category, sensitivity, dates when the incident occurred, was discovered, was reported, and when closed. Also the Incident can be linked to external reporting ids.

9.6 Intel

The Intel collection is for tracking cyber security intel reports from public or private sources. Once input, the Flair engine will detect and cross correlate all Entities found with the Intel record. This can be very powerful and easy way to find threat actors withing you existing SCOT data as well as flagging it in new incoming alerts.

In the example above, we see that an analyst received a heads up from a friend at the XYZ corp. The analyst created the intel record, and SCOT flaired the screensaver name and the hash of that file. Now the analyst can immediately see that the kittens.scr has been seen in the SCOT data one other time and can click on the kittens.scr to see where and what was done about it.

9.7 Guide

Guides are instruction manuals, built by the team over time, on how to handle an Alert type. This can greatly speed the training of new analysts and provide a forum for the team to share their techniques for investigating an alert type.

9.8 Task

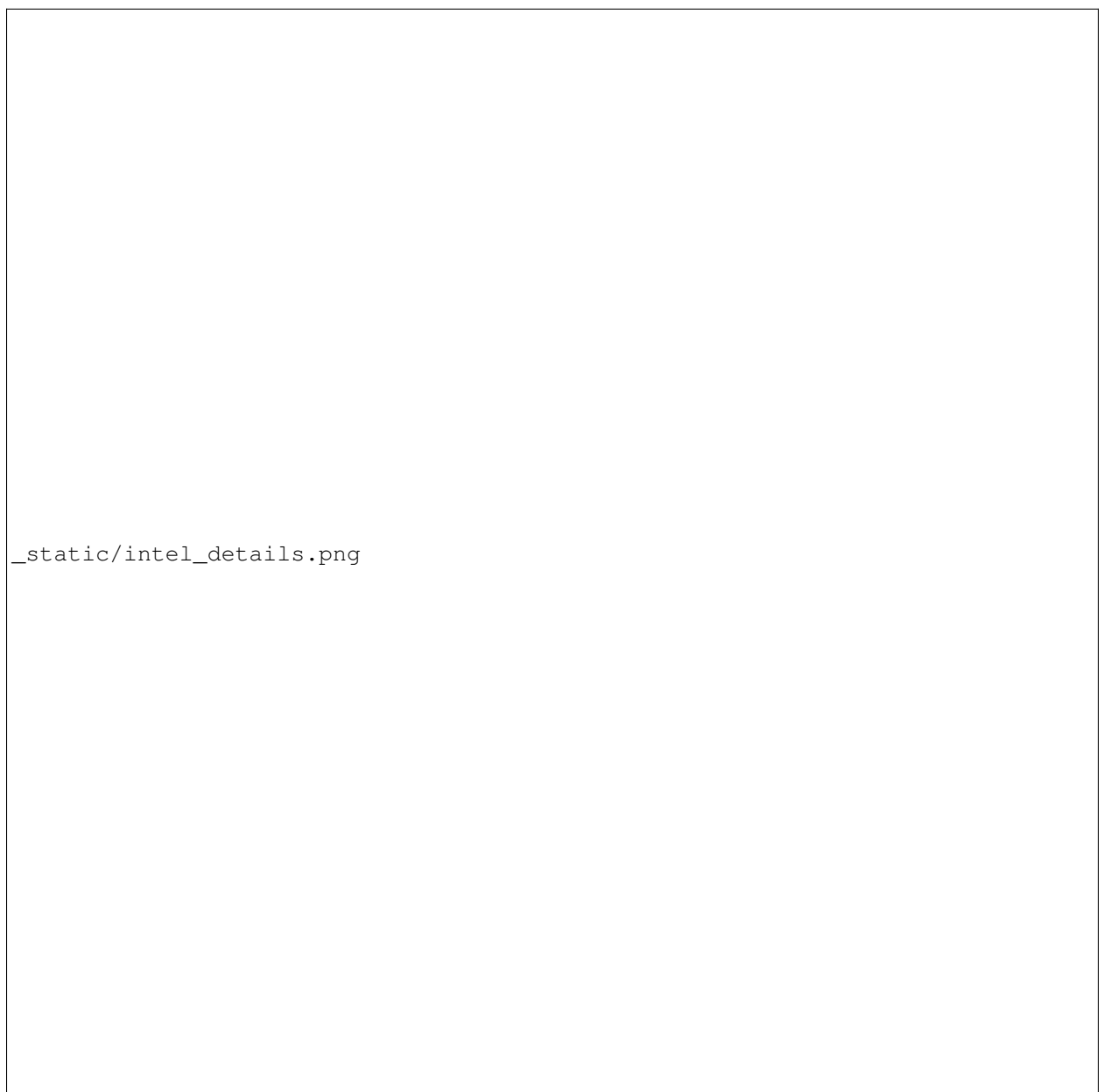
Tasks are used mostly in Events to note a unit of work that still needs to be performed for example, “Pull pcap for 8.8.8.8 from 8am - 2pm”. Some people use these as reminders for tasks they have to do later in an investigation, and some use them to request help from other members on their team.

This feature has proven very helpful when working on large events by coordinating what work still needs to be done, and those who are working on it. The user creates an entry and by clicking the dropdown selects “Make Task”. This task now shows up on the task list, and anyone from the team can take ownership of the task. This way an analyst that just came back from lunch, or just arrived at work can jump right in.

When a task is created, the creator owns it. The only way to transfer ownership of a task is for another team member to “take ownership.” This prevents tasks being pushed onto someone who may be on vacation. If you want to help, take the task. If someone whats it back, they can take ownership again.

9.9 Signature

Signatures are used to integrate the version control of signatures within 3rd party mitigation tools (firewalls, IDS, etc.) while being managed by SCOT as a central repository. Signature’s have the same list view as other “things” within SCOT, but they have a slightly modified detail view.



`_static/intel_details.png`

The detail view of signatures contain metadata editing, where you can modify a description, signature type (yara, ids, firewall, etc.), production and quality signature body versions, signature group that the signature belongs in, and signature options (written in JSON format). The final new item within the detail view is the Signature body editor. This editor should be used to add in the signature's that will be used in the 3rd party tool. The output of the body is converted to a string format, which can then be ingested by the other tool.

Below these new sections, the entries that are found in other "things" still exist.

See the signature page for more information.

9.10 Tags

Tags are way to annotate AlertGroups, Events, Intel, Incidents, and Entities within SCOT. Say an particular Alert was a false positive. Tagging that alert as "false_positive" will give the team to track all false positive alerts and their occurrence over time. This can be very helpful in debugging or improving detectors.

Tags are space delimited. In other words tags can not contain a space. You can apply many tags to a taggable object. With some creativity you can create grouping of tags by placing a seperator in the string like: "ids:false_positive" to track false positives in the ids system.

9.11 Flair

9.11.1 What the heck is Flair?

The inspiration for the term comes from the classic film "Office Space" (see https://youtu.be/_ChQK8j6so8). We wanted to add pieces of "flair" to indicators of compromise (Entities) to give instant information to analysts. Currently, flair items include a growing list including:

- number of times the Entity appears in SCOT
- country flag for Geo location of IP address
- the existence of notes about the Entity

9.11.2 The Process

Upon Alert or Entry input to SCOT, a message is emitted on the SCOT activity queue. The Flair process, which eagerly listens to this queue, retrieves the Alert or Entry via the REST API and begins processing the HTML.

If we are processing an Entry, we first scan for tags. IMG tags may be links to external websites, internal websites, or Base64 encoded data. Links to external sites may open you up to data leakage (think web bugs), internal sites may require annoying re-authentication, and storing Base64 images within Entries can cause slow downs in storing and indexing those images within SCOT. So let's cache those images locally on the SCOT server.

Assuming that Flairing is running on the SCOT server, external and internal images are pulled down to the server. Base64 images are saved as a file. The HTML of the Entry is modified to point to the new location of the cached file. If flairing is running on a separate system, it will upload the cache image file to SCOT via the REST API and issue an update to the Entry's HTML.

We don't usually encounter IMG tags in Alerts, so we skip scanning for IMG tags. (If you do, place a feature request for us to handle it!)

Next, the Flairing process parses the HTML in the Alert or Entry begins looking for flairable items. The following items are detectable:

- IP addresses
- E-mail addresses
- Domain Names
- File names with common extensions
- Hexidecimal Hash representations like MD5, SHA1, and SHA256
- Latitude/Longitude coordinates

These Entities are extracted via Regular Expressions. If you develop other interesting extractions, please submit a Pull request to have them included in `Scot::Util::EntityExtractor`.

Extracted Entities are stored within the SCOT database and Links are created to the Alert/Alertgroup or Entry and parent Alertgroup, Intel, Event, or Incident. The source HTML is also modified to wrap the Entity with a `` tag of class “entity.” Addition classes may be applied to the Entity based on the type of the Entity.

9.11.3 User Defined Entity

You can highlight text within an Entry and a new button will appear that will allow you to “create user defined entity.” You will be asked to describe the type of the entity. Once you click on create, SCOT will then search through all alerts and entries for that string and flair them. All future appearances of that string will also be flaired as an entity of the type you created.

For example, let’s pretend you just created an entry that says:

The fuzzy foobar group's fingerprints are all over this.

You want to link “fuzzy foobar” as a threat actor group and be able to find other references to this group within SCOT. Highlight the text “fuzzy foobar” click “create entity” and enter “threat-actor-group” as the type. (spaces are autocoverted to dashes ‘-’). Now SCOT will add “fuzzy foobar” to the list of flairable entities and well as “reflairing” instances in previous entries.

9.12 Entities

SCOT parses the alert data and entries for various entities. Entities are commonly refred to as IOC’s (indicators of compromise) but are only limited by the ability to parse and extract strings within the alert and entry data.

Once identified, SCOT, stores metadata about these entities that allows the SCOT UI to “flair” them with highlighting, badges, and other useful visual indicators that help analysts to rapidly identify the impact of the entity.

9.12.1 Entity Types

SCOT can automatically detect and extract the following Entities:

Domain Names SCOT extracts domain names in form of `host.sub.domain.tld`, where `tld` is 2 to 6 characters in length. Secondary validation against Mozilla’s TLD database (`effective_tld_names.dat`)

File Names Common filename extensions such as `exe`, `pdf`, and so on are detected by SCOT.

Hashes SCOT can extract MD5, SHA1, and SHA256 hashes from input data.

IP Addresses SCOT will extract IP addresses. IP version 4 address will have type `ipaddr`, and IP version 6 addresses will have type `ipv6`.

Email Addresses E-mail addresses, both email username and the domain, are extracted and watched.

Latitude/Longitude In the form of -120.093 +100.234

CVE SCOT will detect CVE names in the form of “CVE-YYYY-XXXX”

CIDR SCOT will detect CIDR blocks in the form of X.Y.Z.0/A, where A is 0 through 32. As an added benefit, SCOT will also link ip address entities that are in that CIDR block to this entity as well.

9.12.2 Building Additional Entity Types

The primary tool for entity extraction is the Perl module `Scot::Extractor::Regex`. Additional regular expression may be added to this module to extract additional entities.

Another way to add additional Regexes is to add them to your `scot.cfg.pl`. Add a key “entity_regexes” to the `cfg` file. This array of items will be added to the `Regex` module at server start time.

The format of the regex item is:

```
{
  type    => "name_of_the_entity",
  regex   => qr{ regex_here },
  order   => number,      # lower numbers get precedence over higher
}
```

9.13 Permissions

The SCOT security model is a group based access control. Top level “things” like alertgroups, events, incidents, guides and intel have a attribute called “groups.” The groups attribute an object of the form:

```
group: {
  read: [ 'group1', 'group2' ],
  modify: [ 'group1' ]
}
```

As you would imagine the read attribute lists the groups that are allowed to read this “thing.” Similarly, the modify field lists the groups that can modify the “thing.” When an Entry is created for this “thing,” unless expressly set, the permissions of the entry will default to this set of permissions.

Somewhat surprisingly, a subset of data about a thing, namely the details in the “list view” are viewable by everyone regardless of group membership. The primary reason for this is to allow teammates to see that an alert or an event exists. If that teammate is not in the proper group membership, SCOT will inform them, and the teammate can inquire with his team administrator about joining that group. We feel that the small risk of data “leakage” is outweighed by the benefit of the team being able to discover events that they may be able to contribute to.

9.13.1 Default Groups and Owners

Default groups are set in the `/opt/scot/etc/scot_env.cfg` file. The default owner is also set in this file.

9.13.2 Admin Group

The admin group name is also defined in the `/opt/scot/etc/scot_env.cfg` file. Members of this group have “root” powers and can change ownerships, and read and modify group settings.

9.13.3 Note about Group Names

If you are using LDAP to manage group membership, and your team members have large sets of groups they belong to, you can run into a limit in the number of characters returned from LDAP. This sometimes truncates the grouplist in such a way that the SCOT group may not be returned.

To help avoid this, SCOT filters the LDAP query looking for a common string in all SCOT groups. By default this is “wg-scot” but can be changed in the `/opt/scot/etc/ldap.cfg` file. The line:

```
filter => '(| (cn=wg-scot*))'
```

can be changed to whatever naming convention you decide upon.

9.14 HotKeys

The following hotkeys are supported:

```
f: Toggle full screen mode when a detail section is open
t: Toggle flair on/off
o: This will open all alerts within the alertgroup when in the alertgroup list view
   ↳ as your focus.
c: This will close all alerts within the alertgroup when in the alertgroup list view
   ↳ as your focus.
j: This will select one row down within the list view.
k: This will select one row up within the list view.
esc: This will close the entity pop-up window.
```

9.15 Posting a global notificaton:

A global notification can be posted to all users by navigating to:

```
https://<scot instance>/#/wall
```

Note that only raw text will be displayed.

10.1 Backup

SCOT supports on-demand and scheduled backups. The backup script is:

```
/opt/scot/bin/backup.pl
```

and will back up the SCOT's mongo database and the ElasticSearch collections. The backup is a gzipped tar file and will be stored in /opt/scotbackup. Moving these backups to another system is left as an exercise to the admin. By default, the last 7 days of backups are kept in /opt/scotbackup and files older than 7 days are removed.

10.1.1 Manual Backup

I get it, you don't trust some fancy script to back up. Here's what is going on behind the scenes.

1. Back up the mongo database with the “mongodump” command.

```
$ cd /directory/with/space $ mongodump --db scot-prod $ tar czvf /another/dir/scot-prod.tgz ./dump
```

2. Use unix tools to copy SCOT config in /opt/scot/etc

3. ElasticSearch backup is more involved:

##. if you have never backed up elastic, you will need to create a repo:

```
| curl -XPUT localhost:9200/_snapshot/scot_backup -d '{  
|   "scot_backup": {  
|     "type": "fs",  
|     "settings": {  
|       "compress": "true",  
|       "location": "/opt/esback"  
|     }  
|   }  
| }'
```

##. if you have already backup up once before, remove any conflicting snapshot (or use different snapshot name):

```
$ curl -XDELETE localhost:9200/_snapshot/scot_backup/snapshot_1
```

###. Create the Snapshot:

```
$ curl -XPUT localhost:9200/_snapshot/scot_backup/snapshot_1
```

##. Check on status:

```
$ curl -XGET localhost:9200/_snapshot/scot_backup/_all
```

##. When complete, use tar to back up /opt/esback:

```
$ tar czvf /home/scot/esback.tgz /opt/esback
```

##. store scot-prod.tgz and esback.tgz in a safe place.

10.2 Restore

Extract the timestamped SCOT backup tar file:

```
tar xzvf scotback.201701211832.tgz
```

This will create a directory “./dump/scot-prod”. Restore the MongoDB with:

```
mongorestore --dropdatabase --db scot-prod ./dump/scot-prod
```

10.2.1 Manual Restore

1. Restore Mongo:

##. remove existing scot-prod database:

```
$ mongo scot-prod < /opt/scot/etc/database/reset.js

##. extract scot-prod.tgz::

$ cd /home/scot
$ tar xzvf /tmp/scot-prod.tgz
$ cd dump
$ mongorestore --db=scot-prod .
```

1. Restore configs by copying backup of /opt/scot/etc/ directory

2. Restore ElasticSearch

##. Close ElasticSearch indexes that are active.:

```
$ curl -XPOST localhost:9200/scot/_close
```

##. Remove existing contents of /opt/esback:

```
$ rm -rf /opt/esback/*
```

##. extract esback.tgz:

```
$ cd /opt/esback
$ tar xzvf /tmp/esback.tgz
```

##. Make sure that /etc/elasticsearch/elasticsearch.yml has the following:

```
repo.path: [ '/opt/esback' ]
(restart es if you have to make a change to the yml file
```

##. Create the “scot_backup” repo if it doesn’t exist (see above)

##. curl -XPOST localhost:9200/_snapshot/scot_backup/snapshot_1/_restore

3. Finally, restart scot.:

```
# service scot restart
```

10.3 SSL Certs

The initial install of SCOT will use self-signed SSL Certs. Please update these certs as soon as possible.

10.4 GeoIP

SCOT use the MaxMind GEOIP2 libraries and databases for geo location. Please see the MaxMind website for details on how to update the database files.

10.5 Upgrading

Pull or Clone the latest from github (<https://github.com/sandialabs/scot>). CD into the downloaded directory, run:

```
./install.sh -s
```

You probably want to do this when your analysts are not very busy.

10.6 CRON Entries

If you are using /opt/scot/bin/alert.pl to import events you will need a crontab entry like:

```
*/5 * * * * /opt/scot/bin/alert.pl
```

To automate your backups:

```
0 3,12,20 * * * /opt/scot/bin/backup.pl
```

10.7 Daemons

A properly functioning SCOT has the following services running:

- ActiveMQ
- MongoDB
- Apache2
- Scot
- scfd (scot flaring daemon)
- scrfd (scot reflairing daemon)
- scepd (scot elastic push daemon)

Depending on the Linux version, these will have init style startup scripts or systemd style entries.

10.8 Logging

SCOT is a prolific logger. All logs are stored in `/var/log/scot`. It is highly recommended to set up logrotate to avoid filling you disk. Create a `/etc/logrotate.d/scot` like:

```
/var/log/scot.*.log {
    daily
    missingok
    rotate 5
    compress
    notifempty
    copytruncate
}
/var/log/error.*.log {
    daily
    missingok
    rotate 5
    compress
    notifempty
}
```

10.9 Manual Password Reset for Local Auth

Let's say you forgot the admin password, what to do?

1. Run `/opt/scot/bin/passwd.pl`

```
$ /opt/scot/bin/passwd.pl Enter New Admin Password : * Reenter Admin Password : * {X-
PBKDF2}HMACSHA2+512:AAAnEA:2/oQYlnzjibzWoCs2aPv:KAZIhhNUgPBw4M7ZOVU1/2yT/P07FRe2bhacBw6J6ru4jw
```

2. Enter mongodb shell and issue the following:

```
$ mongo scot-prod <enter> > db.user.update({username:"admin"},{$set:{hash:'{X-
PBKDF2}HMACSHA2+512:AAAnEA:2/oQYlnzjibzWoCs2aPv:KAZIhhNUgPBw4M7ZOVU1/2yT/P07FRe2bhacBw6J6ru4jw
```

3. Now you (admin) will be able to log in via Local Auth using the password you entered.

11.1 SCOT Architecture

overview of the puzzle pieces go here

11.2 SCOT Directory Map

bin/ scripts, executables that run stand alone (not part of the webservice) Ex: bots, import scripts, export scripts, etc.

docs/ text files that form documentation for SCOT

etc/ configuration files for Scot go here

etcsrc/ starting templates for your config files.

lib/ Perl library Hierarchy

 Scot.pm - the top level mojolicious application library containing route info

Scot/ the top level of the Scot:: modules

Bot/ modules for use by scot bots

Controller/ modules for handling routes defined in Scot.pm

Model/ modules describing the data model for Scot data types

Util/ authentication, database, and other general utility modules

public/ Static served files by mojolicious

css/ css files for scot, and frameworks

img/ images used by SCOT

fonts/ Any fonts used by the CSS go here

lib/ javascript 3rd party libraries

angular/ angular libraries go here

bootstrap/ bootstrap stuff

jquery/ jquery stuff

js/ javascript that we create for scot including react components

api/ api documentation

docs/ online documentation

script/ usually holds the mojolicious startup script

t/ tests, tests, tests of mojolicious back end

templates/ templates used for rendering data that was passed through mojolicious

11.3 SCOT REST API

[SCOT API Documentation.](#)

11.3.1 SCOT get API

1. Retrieve one “thing” when you know the id:

```
/scot/api/v2/event/123

output:

JSON object representing event 123
```

2. Retrieve list of “things”:

```
/scot/api/v2/event

output

{
  queryRecordCount: 50,
  totalRecordCount: 10060,
  records: [
    { event json object 1 },
    ...
  ]
}
```

3. Retrieve list of “things” based on time range:

```
/scot/api/v2/event?created=1472244135&created=1472244137
```

4. Retrieve list of “things” based on string match:

```
/scot/api/v2/event?subject=Symnatic
```

5. Retrieve list of “things” base on Numerical conditions

- a. matching a single number:

```
/scot/api/v2/event?views=2
```

```
output: events with two views
```

b. matching a set of numbers:

```
/scot/api/v2/event?entry_count=2&entry_count=4&entry_count=6
```

```
output: events with an entry_count of 2, 4, or 6
```

c. matching everything but a number:

```
/scot/api/v2/event?views!=1
```

```
output: events with views not equal to 1
```

d. matching everything but a set of numbers:

```
/scot/api/v2/event?views!=1&views!=2&views!=3
```

```
output: events with views not equal to 1,2, or 3.
```

```
(note: if ! appears in any element, all are treated as if they are !
```

e. matching an expression:

```
/scot/api/v2/event?views=4<x<8
/scot/api/v2/event?views=4<=x<8
/scot/api/v2/event?views=4<=x<=8
/scot/api/v2/event?views=4<x<=8
/scot/api/v2/event?views=9>x>=2
```

```
output: events with views (represented by x) matching the expression
syntax notes: the expression must be of the form some number of digit,
followed immediately by one of the following operands: < <= > >=, the
letter lower case x (which represents the column name) followed
immediately by the comparison operands, and finally followed
immediately by some number of digits.
```

1. Retrieve list of “things” based on Set Fields like “tag” or “source”:

```
/scot/api/v2/event?tag=email&tag=malware&tag=!false_positive
```

```
output: list of events with tags email and malware but not containing
the tag false_positive
```

11.3.2 SCOT post API

1. Create an Alertgroup containing several alertgroups:

curl -XPOST /scot/api/v2/alertgroup -d '{

```
  "message_id": "112233445566778899aabbccddeeff", "subject": "Detection of Bad Stuff",
  "data": [
```

```
    { "column1": "data11", "column2": "data12", "column3": "data13" }, { "column2":
    "data21", "column2": "data22", "column3": "data23" }, { "column3": "data31", "col-
    umn2": "data32", "column3": "data33" },
```

```
], "tag": [ 'tag1','tag2','tag3' ], "source": [ 'source1' ], columns: [ 'column1', 'column2', 'column3' ],  
  },  
}
```

2. Create an Event:

```
curl -XPOST /scot/api/v2/event -d '{  
  "subject": "Test Event", "source": [ "Orchestration" ], "tag": [ "tag1", "tag2" ], "status": "open",  
  "groups": [  
    read: [ "scot-group" ], modify: [ "scot-group" ],  
  ]  
}'
```

3. Create an Entry attached to a known event:

```
curl -XPOST /scot/api/v2/entry -d'{  
  "target_id": 123 "target_type": "event", "body": "any text/html here",  
}'
```

11.3.3 SCOT put API

1. Update and event status:

```
curl -XPUT /scot/api/v2/event/123 -d '{  
  "status": "closed"  
}'
```

11.3.4 SCOT delete API

1. Delete and entry:

```
curl -XDELETE /scot/api/v2/entry/12345
```

11.4 SCOT Event Queue

SCOT uses a message queue to publish events that have occurred. This allows your process to subscribe to be asynchronously updated and to take actions on these event. SCOT uses ActiveMQ, it gets the job done and just about every language under the sun has a way to interface with it.

The message format is:

```
{  
  guid:    "unique_guid_string",  
  action:  "action_string",  
  data:    {  
    type:   "type_of_data_structure",  
    id:     integer_id_of_data,  
    who:    username,  
  }  
}
```

unique_guid_string is a requirement of the STOMP protocol and is generated

action_string is a member of the following: * “created” = something was created * “updated” = something was updated * “deleted” = something was deleted * “viewed” = something was viewed * “message” = send a message to a subscriber

type

describes the data type that was operated on and is one of:

- alert
- alertgroup
- entry
- event
- incident
- intel

or in the case of a “message” it can be any string that your client is listening for.

id is an integer id for the “type” above. if sending a message, this could be the an epoch time.

data is a json structure that you are free to put stuff in.

11.5 SCOT Server

will discuss how to work in the Perl base server. Perldocs will be linked here as well.

11.6 SCOT UI

SCOT’S front end is primarily developed using React JS. See <https://facebook.github.io/react/> to read more about it.

pubdev/ Contains files necessary to modify the React-based front end

Note: Not all of the front end of SCOT is developed in React. Currently, the Incident Handler calendar and administration pages are written using jQuery and HTML, without React.

SCOT has been written using the JSX format. See <https://facebook.github.io/react/docs/introducing-jsx.html> to read more about it.

JSX Libraries Most libraries that the SCOT JSX components (found in /pubdev/jsdev/react_components/) rely on are found in /pubdev/node_modules and can be installed/updated using npm.

JSX Compiling Compiling the JSX files into a single javascript file is done by using Gulp. The file that specifies the compiling directories is /pubdev/gulpfile.js. The final file that is ultimately compiled and used is /public/scot-3.5.js

JSX Dev If you would like to contribute to, or modify the front end of SCOT, you can do so by creating/modifying files in /pubdev/ and then compile your changes using gulp.

Final HTML/JS The files ultimately used to display and control the front end are found in /public/

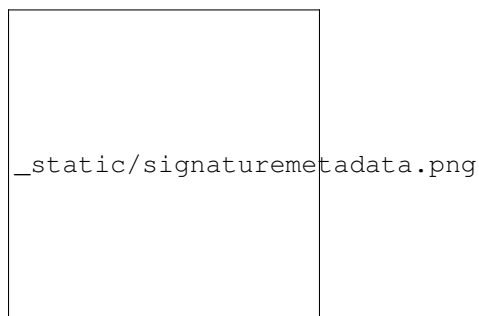
12.1 Signature

Signatures are used to integrate the version control of signatures within 3rd party mitigation tools (firewalls, IDS, etc.) while being managed by SCOT as a central repository. Signature's have the same list view as other "things" within SCOT, but they have a slightly modified detail view.

The detail view of signatures contain metadata editing, where you can modify a description, signature type (yara, ids, firewall, etc.), production and quality signature body versions, signature group that the signature belongs in, and signature options (written in JSON format). The final new item within the detail view is the Signature body editor. This editor should be used to add in the signature's that will be used in the 3rd party tool. The output of the body is converted to a string format, which can then be ingested by the other tool.

Below these new sections, the entries that are found in other "things" still exist.

12.2 Signature Metadata



Signatures contain their own unique metadata that can be used for version control, describing the signature, and grouping the signatures. The metadata contains the following options:

Description Describes the signature

Type Defines the type of signature being created (yara, firewall, ids, etc.)

Production Signature Body Version Declares the version of the signature body to be used in production

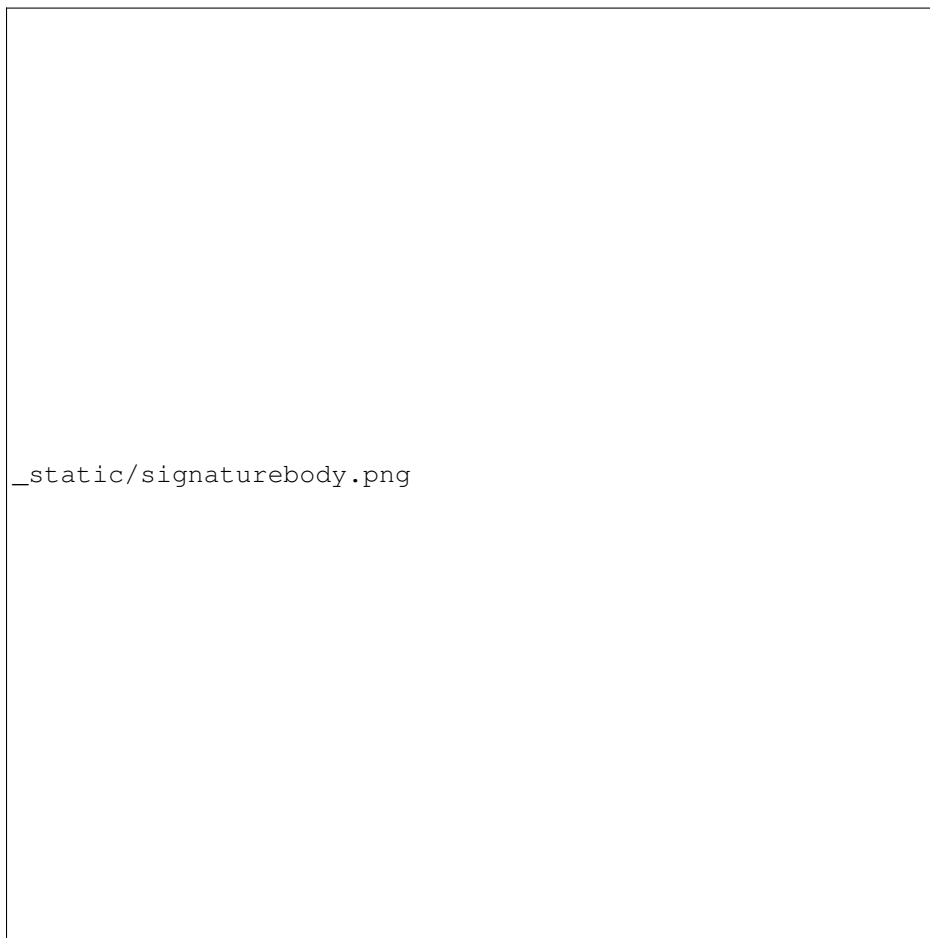
Quality Signature Body Version Declares the version of the signature body to be used in quality

Signature Group Declares a group name if signature will be grouped together.

Signature Options This is the first editor window you see that accepts JSON formatted data that can be used to pass on specific options for the signature being applied.

Signature Body This is the second and larger editor window you will see that will pass along the contents as a string to the SCOT server that can then be used by the tool ingesting the signature. The signature body editor contains a few options - Editor Theme, Language Handler, Keyboard Handler, Signature Body Version, and the following buttons - Create new version, Create new version using this base, updating displayed version.

12.2.1 Signature Body Options



Note: The Editor Theme, Language Handler, and Keyboard Handler all save their settings in a cookie file. If you change these settings, they will persist until you change them again or clear your browser's cookies.

Editor Theme You can select the theme you prefer to use for your code editor. There are a variety of color options depending on your color tastes

Language Handler You can select the language of the signature that you are writing, or one that closely resembles it

Keyboard Handler You can select none, vim, or emacs if you prefer a keyboard handler

Signature Body Version You can select the version you would like to view here.

Create new version This button will empty the editor and allow editing within the editor so you can create a new signature body. Any other signature body's created will remain attached to this Signature to be viewed/edited. Note that this WILL NOT make the new signature body automatically the "qual/prod" versions, as that must be done manually in the metadata section.

Create new version using this base This button will also create a new signature body version, but it will start the editor off with whatever contents are already in the editor based on the version selected.

Update displayed version This button will allow editing of the version selected. It will not create a new version of the signature body, but instead just update the version selected.

13.1 Read-Eval-Viz-Loop

REVL is a tool for quickly reorganizing awkward data formats so that you can inspect the data and use a variety of visualizations to find interesting relationships and properties that would be hard to spot otherwise. It works in a way similar to a powerful command line in that you get data on one end, run it through a series of transformations to pick out the bits you're interested in and stick them to other bits, finally ending up with just the interesting parts in a format that's easy to comprehend or ship off to a visualization tool (of which many are included). Internally, REVL uses a result monad to do the value handling, so you're actually working with a data structure instead of raw text. In this case, this makes it quite a bit more convenient to use than the standard command line.

13.2 Getting Started

When you open SCOT, click the `Visualization` link in the navbar. This will open REVL, which will look like a big blank screen with a little command prompt at the bottom. You will interact with the system by typing strings of commands at the prompt and observing the results either in the text output area (just above the prompt) or in the visualization area (the bulk of the page, which is blank white at this point).

13.3 Interacting with REVL

To Get some help click in the prompt and type `help` (and press Enter).

Just above the prompt, you will see a text output area. You can drag the top of this area to resize it, so drag it up now to see the REVL default help message. This message gives a little background and lists all currently loaded commands. If you can't remember the name of something, you should be able to jog your memory by looking it up here.

Now type `help map` at the prompt. This will display the command-specific help for the `map` command, which is something you will be using a *lot*.

REVL tries to be convenient - if it recognizes the first word you type in a command segment to be a command, it treats it as one. If not, it will evaluate whatever you type in the context of the shell, which includes variable definitions, locally defined helper functions, and the entire API behind the command system. The syntax is coffeescript, which you can find out more about at [\[\[http://coffeescript.org\]\]](http://coffeescript.org) [\[{}\]{ }CoffeeScript.org](http://coffeescript.org)].

Type:

```
[1..10]
```

at the prompt and hit Enter. You will see the result of evaluating that coffeescript value, which is

```
[1,2,3,4,5,6,7,8,9,10]
```

This particular trick (generating a list of integers) is surprisingly useful for seeding queries later on. Keep it in mind when you want to do something like look at all of the events that came in between two other events (you can sequence their id fields using this list, eg [1044..1102]).

Now hit the up arrow to repeat the last command, then add to the back of it until you get this (the thing right after the list is a single backslash character):

```
[1..10] \ (n)->n*n
```

After you hit enter, you'll see a list of the squares of the integers from the first list: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]. You just used the `map` command. You could also have explicitly written out the `map` name in front of the function definition, but this particular command is so common that it's implied after a backslash if no other command is specified.

Commands are chained together using the backslash (‘\’) character. Normally the pipe (‘|’) would have been used, but in this case it was just much simpler and more reliable to use the backslash because the pipe is an important character in user-defined coffeescript code, and it would have led to significant ambiguity in parsing the commands.

13.4 Using REVL with SCOT data

Now we can do something interesting. Let's get all the entities with ids from 10000 to 10100:

```
entity offset:10000,limit:101
```

This command will take a few seconds to complete, and when it does you'll see a list of entities in the text output. However, our plan was foiled - our first id is not 10000, it's something else. If we want to actually get entities with ids 10000 to 10100, we'll need to specify those ids. Let's do that:

```
[10000..10100] \ (n)->API.entity id:n
```

After you press Enter and wait, you'll find that you got a list of 100 somethings back, but they aren't entities. REVL uses asynchronous calls for the API to make things a little faster. This is hidden when you use the top level commands because the shell knows to wait when the result is a promise, but when you make calls directly to the API and embed them in another data structure, you have to be more explicit. Let's go ahead and tell it to wait on those results:

```
[10000..10100] \ (n)->API.entity id:n \ wait
```

The `wait` command will scan through the data it gets from the pipeline and replace all of the promises with the fulfillments of those promises as they come in. It also has an optional timeout which will cause the wait to stop if it has been more than that long since an update was received. The default timeout is 60 seconds, and you can change it by simply specifying a different number as an argument to the `wait` command. This argument is a full coffeescript value, so you can use variables and functions if you need to for some reason.

As you wait for the entities to come down, notice that there is a progress bar on top of the command line to let you know something is happening in the background, and the fraction of finished to total promises is displayed at the right end of the command line.

When it's all said and done, you should have a list of 101 entities in your text window.

13.5 Make a bar chart

Let's take those entities and see how they're distributed by type. To do that, we'll fetch the entities, then pick out the type field, group them by that field, and make a chart that has a bar for each type and shows the number of instances of that type. First, let's get the entities again and stash them so that we don't have to wait for them to download at each step:

```
[10000..10100] \ (n)->API.entity id:n \ wait \ store ents
```

The `store` command takes a variable name and stores the result of the preceding command in the scope under that name. Now you can access that list of entities using the name `ents` from anywhere in future commands. First, let's strip out all of the data we don't care about from them:

```
ents \ (e)->e.type
```

Now you should see a list of the type fields from each entity. Next we'll group them according to that field:

```
ents \ (e)->e.type \ group (x)->x
```

This command uses the `group` command, which takes a function and returns an object. The function should return a name for its input that specifies what group it belongs in. In this case, all we have are names, so we just tell it to return its input unchanged (that's what the `(x)->x` means - a coffeescript identity function).

The output of the `group` command was an object with a key for each group name, and the list of things in that group for the value. Now we're going to replace the lists with their lengths, which will give us a nice data structure to pass to the `barchart` visualization primitive:

```
ents \ (e)->e.type \ group (x)->x \ (ls)->ls.length
```

This uses the `map` command to iterate over the keys of the object returned by `group` and replace each value by its length. You should now have an object with a few keys, each with a number as its value. This is exactly the format we need for a bar chart, so let's see what we get:

```
ents \ (e)->e.type \ group (x)->x \ (ls)->ls.length \ barchart
```

You should now see a chart showing the relative frequencies of the different entity types in your set. If your text area is covering the chart, you can double click the top of it to auto-minimize. It will remember the last setting for the height, so if you double click it again it will go back to where it was.

13.6 Event Timing

Next we'll use a dot chart to look at the timing of a set of alerts coming in within an alert group. First, let's get the alerts:

```
alertgroup: id:1512214,sub:'alert'
```

After this comes in you should have a list of alerts. There's a lot of data we don't really care about there, so let's tell the server to only send what's important:

```
alertgroup: id:1512214,sub:'alert',columns:['id','when']
```

This filters the data coming in down to just the `id` and `when` columns, which suits our needs for this example. We can store that data for future reference:

```
alertgroup: id:1512214,sub:'alert',columns:['id','data'] \ store a151
```

We're going to make a dot chart with time on the horizontal axis and item number on the vertical (vertical axis is just here to separate things for visibility). We need to pull out the time value for each and pair it with its position in the list:

```
a151 \ (alert,pos)->[pos,alert.data._time]
```

The `map` function implicitly passes the index of the current element to the handler function (or the key if it's an object). We just use the object's list position to get a vertical coordinate for it. Unfortunately, this timestamp is in human-readable format, which makes it a pain to use. We can parse it using the `Strings` function though:

```
a151 \ (r)->r.data._time \
  pick Strings.pat.hms \
  (ls)->(map ls[1..], (s,i)->(60**(2-i))*(parseInt s))).reduce (a,b)->a+b
```

This takes the alerts and uses the `Strings` predefined `hms` (hours:minutes:seconds) pattern to parse just the clock time from the timestamp. The pattern returns the matched string along with its captured substrings, which in this case gives us the hour, minute, and second. The function mapped over it just converts this into a number of seconds since midnight. Coffeescript has a `**` operator for exponentiation, if you're trying to parse out how that function works. Now we have a list of timestamps, so let's convert it to a list of coordinate pairs that `dotchart` can use:

```
a151 \ (r)->r.data._time \
  pick Strings.pat.hms \
  (ls)->(map ls[1..], (s,i)->(60**(2-i))*(parseInt s))).reduce (a,b)->a+b \
  (n,i)->[n,i] \
  dotchart
```

Whoops, looks like the timing data is all over the map! We need to sort our timestamps in ascending order since they didn't come that way from the server:

```
a151 \ (r)->r.data._time \
  pick Strings.pat.hms \
  (ls)->(map ls[1..], (s,i)->(60**(2-i))*(parseInt s))).reduce (a,b)->a+b \
  sort \
  (n,i)->[n,i] \
  dotchart
```

`sort` does just what you'd think. You can optionally pass it a comparison function, which should return `-1`, `0`, or `1` depending on whether the first argument is less, equal, or greater than the second. Note that javascript has some very weird ideas about ordering, so if you want to get the expected sort order for normal data (numbers, strings, etc.) REVL provides a sort function in the `Utils` module called `Utils.smartcmp`. This basically says numbers go in numeric order and strings go in alphabetic order. In javascript by default, numbers go in alphabetic order (!). Running this command we can now see a nice progression of alerts that ended up in this alert group.

13.7 Other interesting command examples

Here are some other commands you might want to play with to get a feel for the system. All of the basic commands have documentation with examples, so if you need to look something up to see how it works start with the help system.

- Entity Frequencies over time

Query 1000 entries, pull the entities for each of them, group them by type, and create a barchart to show the relative frequency of each type of entity:

```
$ [10000...11000] \
  (n)->API.entry {id:n,sub:'entity'} \
  wait \
  (r)->Struct.tolist (Struct.map r,(v)->v.type) \
  flatten \
  group (ls)->ls[1] \
  (ls)->ls.length \
  barchart
```

- Examine event timing over long periods

Query 500 events, extract the creation timestamp, sort them in ascending order, rebase the time to show time delta in minutes from start of record, and create a dot chart to show the timing of clusters of events and highlight gaps in the record:

```
$ event limit:500 \
  (e)->e.created \
  sort \
  into (ls)->map ls,(n)->(n-ls[0])/60000.0 \
  (n,i)->[n,i] \
  dotchart
```

- Look at sequence of alerts in alertgroup:

```
$ alertgroup id:1512214,limit:100,sub:'alert' \
  (r)->r.data._time \
  pick Strings.pat.hms \
  (ls)->(map ls[1..],(s,i)->(60**(2-i))*(parseInt s)).reduce (a,b)->a+b \
  sort \
  (n,i)->[n,i] \
  dotchart
```

- Network connections between emails mentioned together in an alert for an alert group

Get the alerts for alertgroup 1512214, concatenate all of the strings in the data field of each, pick out all of the email addresses in the resulting strings, generate pairs from all emails that were in the same alert, and make a force-directed graph from the resulting structure.:

```
$ alertgroup id:1512214,limit:100,sub:'alert' \
  (r)->(squash (Struct.tolist r.data)).join ' ' \
  (s)->Strings.pick Strings.pat.email, s \
  (ls)->ls.map (m)->m[0] \
  (ls)->cmb ls,2 \
  flatten \
  forcegraph
```

- Association matrix of emails from one alertgroup

This is a very heavy computation, but it eventually finishes. Need to look into ways to optimize this to make it more convenient, but the filling out of the table really explodes the size of the data set.:

```
$ alertgroup id:1512214,limit:100,sub:'alert' \
  (r)->(squash (Struct.tolist r.data)).join ' '\
  (s)->Strings.pick Strings.pat.email, s \
```

(continues on next page)

(continued from previous page)

```

(ls)->ls.map (m)->m[0] \
(ls)->cmb ls,2 \
flatten \
nest (n)->n \
(row)->Struct.map row,(col)->col.$.length \
tabulate {} \
grid \
eachpoly (p)->if p.input == {} then p.color='#000' else p.color=Utils.heatColor p.
↪input,10 \
draw

```

- Draw a treemap from an Nspace:

```

$ [1..100] \
  foldl new Nspace (s,pt) -> s.insert pt,['x',Math.random()],['y',Math.
↪random()]]; s \
  into (s)->s.subdivide() \
  into (sp)->sp.leaves() \
  (l)->l.bounds \
  (bnd)-> zip bnd \
  (pts)->[[pts[0][0],pts[0][1]], [pts[0][0],pts[1][1]], [pts[1][0],pts[1][1]],
↪[pts[1][0],pts[0][1]]] \
  (pts)->(polygon pts).scale 200 \
  into (polys)->{polygons: polys} \
  draw

```

- Network showing relationship between events and entities

Query an event, find all the entities associated with it, then find all the events associated with those entities. Make links accordingly, then display as a force-directed graph. Mousing over the network nodes will display the entity name or event id number depending on what kind of node it is.:

```

$ event id:10982,sub:'entity' \
  (e,k)->[{id:e.id,name:k},10982] \
  tolist \
  (ls)->ls[1] \
  filter (ls)->ls[0].id not in [4802,97248,19,533065,97249] \
  (ls)-> [[ls[0].name,ls[1]], (API.entity sub:'event',id:ls[0].id).map (e)->([ev.
↪id,ls[0].name]) for ev in e] \
  wait \
  flatten \
  flatten \
  forcegraph

```

- Barchart of event count for each entity

Fetch the entities associated with an event, then fetch all of the events for each entity and make a barchart that shows how many events are associated to each entity.:

```

$ event id:10982,sub:'entity' \
  (ent)->(API.entity id:ent.id,sub:'event',columns:['id']).map (ls)->ls.length \
  wait \
  filter (n)->n>20 \
  barchart

```


Updated - 12/10/2019

14.1 Table of Contents

- Overview
- Docker-SCOT containers
- Managing the containers
- Configuration
- FAQ / Common Issues

14.1.1 Overview

SCOT's primary deployment technology is now via docker.

IMPORTANT

Backup your database via the backup.pl in the /opt/scot/bin/ directory before upgrading to the docker version of SCOT. If you are upgrading, you will also need to turn off all services that the older version of SCOT uses such as Apache, Activemq, Mongoddb, ElasticSearch and SCOT (i.e. `sudo service stop scot`). Also as far as upgrading, we have **not** tested upgrading from any version before 3.4. Upgrade from versions prior to 3.4 to 3.5 first before upgrading to Docker-SCOT.

14.1.2 SCOT containers

SCOT is comprised of the following services:

- **SCOT** - SCOT Application and associated API
- **MongoDB** - Storage for SCOT

- **ActiveMQ** - Message broker for services interested in SCOT data
- **Apache** - Proxy for traffic between some services
- **ElasticSearch** - Search engine
- **Flair Engine** - 'Entities' found within SCOT are highlighted with a count of the number of times SCOT has 'seen' them before
- **Game Engine** - Used for homepage statistics
- **Stretch** - Used for adding data to ElasticSearch
- **Mail** - Used as a resilient mechanism for importing data to SCOT (not enabled by default - See configuration section)
- **Reflair** Similar to flair

14.1.3 Docker Installation

To get started, refer to the Docker Community Edition documentation for installing the Docker engine on your respective OS: <https://docs.docker.com/engine/installation/>

Next, Docker-SCOT relies on docker-compose to build, run and manage services. Docker-compose does not ship with Docker engine, so you will need to refer to the following documentation for installation of Docker-Compose: <https://docs.docker.com/compose/install/>

14.1.4 SCOT Installation

Note These steps will most likely change slightly in 2019 as the SCOT team will be working on making the install script more robust, but easier to use so you can begin using (and even developing) with SCOT quickly.

There are two methods for getting started with SCOT. Run the SCOT/restart-build-deploy.sh script (will be prompted to enter sudo credentials) and follow the on screen prompts for either.

1. Quick mode (fast) - this mode will pull all necessary docker images from Dockerhub (preconfigured). As for 12/10/2019, this is the preferred method for SCOT installation. If you need to enable LDAP auth, configure TLS certificates, or other changes, these can be done in this mode but with volume mounts and possibly some manipulation of compose files / configs.
2. Custom Mode (slow) - This mode should only be chosen if you are wanting to rebuild docker images for further customization.

14.1.5 Managing the containers

The restart-build-deploy.sh script will handle stopping and then restarting containers automatically. However if you need more granular control run the following:

To stop Docker-SCOT:

```
sudo docker-compose stop
```

To start a specific service:

```
sudo docker-compose up --build name_of_service
```

To stop a specific service:

```
sudo docker-compose stop name_of_service
```

To restart a specific service and build in any particular changes you have made to source:

```
sudo docker-compose up -d --build name_of_service
```

14.1.6 Configuration

SCOT's implementation of docker relies on the docker-compose.yml or docker-compose-custom.yml file to define the execution of the services, the DockerFiles that define the dependencies for each container, and two directories (docker-scripts & docker-configs).

docker-compose.yml

The docker-compose.yml references the prebuilt images from Dockerhub.

docker-compose-custom.yml

The docker-compose-custom.yml file will build the SCOT docker images from source.

docker-scripts

The docker-scripts directory contains scripts for backing up the data contained in MongoDB container and will eventually house other scripts that are similar.

The following scripts are currently supported:

1. /opt/scot/bin/restore_remote_scotdb.pl
2. opt/scot/bin/backup.pl

To execute one of the above scripts, simply connect to the scot container via:

```
sudo docker exec -i -t -u scot scot /bin/bash
```

cd to /opt/scot/bin/

and run:

```
./scriptexample.pl
```

Restoring a database

For any questions about backing up and restoring databases, please contact the SCOT development team.

docker-configs

The docker-configs directory contains modified config files, perl modules, scripts, etc. that allow SCOT to function properly in a containerized environment. Most changes are references to localhost in the standard SCOT codebase where we modify those addresses to reference the ip addresses on the scot_docker subnet.

MongoDB Default password

MongoDB default password (also used for logging in to SCOT if local auth is enabled (by default)), is:

- Username: admin
- Password: admin

Note: If by chance you ever go to wipe your mongo database and would like to start fresh, you would need to delete the file /var/lib/mongodb/.mongodb_password_set.

Persisted Data

You can view which data is being persisted by viewing the docker-compose.yml script and referring to the various 'Volumes'. With regard to MongoDB (where SCOT records are persisted), the data from mongodb is persisted via nmaed volumes to /var/lib/docker/volumes/mongodb_data..

Mail

To begin using mail, you will need to uncomment the 'mail' service in the docker-compose.yml file and also add any of your organization's mail configurations into the docker-configs/mail/alert.cfg.pl file.

LDAP

By default, LDAP configuration is not enabled in docker-configs/scot/scot.cfg.pl. To enable, simply uncomment the LDAP configuration lines in docker-configs/scot/scot.cfg.pl and edit the necessary information to begin checking LDAP for group membership / auth.

Custom SSL

Docker-SCOT's Apache instance comes configured with a self-signed SSL cert baked into the container. However, if you wish to use your own certificates, do the following:

1. Remove the SSL cert creation lines from the Dockerfile-Apache file.
2. In docker-configs/apache/ directory, there is a scot-revproxy-Ubuntu.conf. Replace the following line:

```
ServerName apache
```

with:

```
Servername nameofyourhost
```

3. In the same file, replace the following lines:

```
SSLCertificateFile /etc/apache2/ssl/scot.crt
SSLCertificateKeyFile /etc/apache2/ssl/scot.key
```

with the path and name of the eventual location where you will map your certs to via a shared data volume. 4. Next, as mentioned above, you need to pump your certs from your host machine into the container via a data volume (you can also copy them into the container at build time via COPY directive). In order to map them in via a data volume, add a new data volume under the apache service in the docker-compose.yml file. Eg.:

```
volumes:
- "/path/to/your/cert:/path/to/file/location/you/defined/in/step/3
- "/path/to/your/key:/path/to/file/location/you/defined/in/step/3
```

5. Re-run the restart-build-deploy.sh script and you should be set!

14.1.7 FAQ / Common Issues

Common Issues

1. Apache frequently will throw an error on run time that the process is already running and will subsequently die. In the event this happens, simply re-run the script.

14.1.8 TODO

1. Complete backup and restore scripts in bash
2. Update docs - better examples

CHAPTER 15

Indicies and Tables

- `genindex`
- `modindex`
- `search`

C

CIDR, [53](#)

CVE, [53](#)

D

Domain Names, [52](#)

E

Email Addresses, [52](#)

F

File Names, [52](#)

H

Hashes, [52](#)

I

IP Addresses, [52](#)

L

Latitude/Longitude, [53](#)