sn0int Documentation

kpcyrd

Sep 09, 2023

Contents

sn0int is a semi-automatic OSINT framework and package manager. It was built for IT security professionals and bug hunters to gather intelligence about a given target or about yourself. sn0int is enumerating attack surface by semi-automatically processing public information and mapping the results in a unified format for followup investigations.

Among other things, sn0int is currently able to:

- Harvest subdomains from certificate transparency logs
- Harvest subdomains from various passive dns logs
- Sift through subdomain results for publicly accessible websites
- Harvest emails from pgp keyservers
- Enrich ip addresses with ASN and geoip info
- Harvest subdomains from the wayback machine
- Gather information about phonenumbers
- Bruteforce interesting urls

sn0int is heavily inspired by recon-ng and maltego, but remains more flexible and is fully opensource. None of the investigations listed above are hardcoded in the source, instead those are provided by modules that are executed in a sandbox. You can easily extend sn0int by writing your own modules and share them with other users by publishing them to the sn0int registry. This allows you to ship updates for your modules on your own since you don't need to send a pull request.

Join us on IRC: irc.hackint.org:6697/#sn0int

CHAPTER 1

Getting Started

1.1 Installation

If available, please prefer the package shipped by operating system. If your operating system has a package but you're running on older version, please use the build from source instructions instead.

1.1.1 Archlinux

```
$ pacman -S sn0int
```

1.1.2 Mac OSX

```
$ brew install sn0int
```

1.1.3 Debian >= bookwork, Ubuntu >= 22.10, Kali

There are prebuilt packages signed by a debian maintainer:

```
$ sudo apt install curl sq
$ curl -sSf https://apt.vulns.sexy/kpcyrd.pgp | sq keyring filter -B --handle 64B13F7117D6E07D661BBCE0FE763A64F5E54FD6 | sudo tee /etc/apt/trusted.gpg.d/apt-vulns-sexy.gpg > /dev/null
$ echo deb http://apt.vulns.sexy stable main | sudo tee /etc/apt/sources.list.d/apt-vulns-sexy.list
$ apt update
$ apt install sn0int
```

1.1.4 Debian <= bullseye, Ubuntu <= 22.04

There are prebuilt packages signed by a debian maintainer:

```
$ sudo apt install curl sq
$ curl -sSf https://apt.vulns.sexy/kpcyrd.pgp | sq dearmor | sudo tee /etc/apt/
↪trusted.gpg.d/apt-vulns-sexy.gpg > /dev/null
$ echo deb http://apt.vulns.sexy stable main | sudo tee /etc/apt/sources.list.d/apt-
↪vulns-sexy.list
$ apt update
$ apt install sn0int
```

1.1.5 Fedora/CentOS/Redhat

Using rust+cargo from the repos might work for you, but we only officially support rust+cargo installed with rustup. Have a look at the docker image as an alternative.

```
$ dnf install @development-tools libsq3-devel libseccomp-devel libsodium-devel␣
↪publicsuffix-list
$ git clone https://github.com/kpcyrd/sn0int.git
$ cd sn0int
$ cargo install -f --path .
```

1.1.6 Docker

```
$ docker run --rm --init -it -v "$PWD/.cache:/cache" -v "$PWD/.data:/data" kpcyrd/
↪sn0int
```

1.1.7 Alpine

```
$ apk add sn0int
```

1.1.8 OpenBSD

```
$ pkg_add sn0int
```

1.1.9 Gentoo

```
$ layman -a pentoo
$ emerge --ask net-analyzer/sn0int
```

1.1.10 NixOS

```
$ nix-env -i sn0int
```

1.1.11 Windows

This is not recommended and only passively maintained. Please prefer linux in a virtual machine if needed.

Make sure rust is installed and setup.

```
$ git clone https://github.com/kpcyrd/sn0int.git
$ cd sn0int
$ cargo install -f --path .
```

1.2 Build from source

It's generally recommended to install a package if available. This section is about building the binary from git.

1.2.1 Install dependencies

You need a recent rust compiler. It's usually recommended to install a rust compiler with rustup, but if you're system ships the most recent compiler in a package that works too. Note that some systems aren't fully supported by rustup (like OpenBSD and alpine) and you need to install rust from a package in that case.

Archlinux

```
$ pacman -S geoip2-database libseccomp libsodium publicsuffix-list sqlite
```

Mac OSX

```
$ brew install libsodium
```

Debian/Ubuntu/Kali

```
$ apt install build-essential libsqlite3-dev libseccomp-dev libsodium-dev␣
↪publicsuffix pkg-config
```

Warning: On a debian based system make sure you've installed rust with rustup.

Alpine

```
$ apk add sqlite-dev libseccomp-dev libsodium-dev
```

Docker

```
$ DOCKER_BUILDKIT=1 docker build -t kpcyrd/sn0int .
```

OpenBSD

```
$ pkg_add sqlite3 geolite2-city geolite2-asn libsodium
```

Gentoo

```
emerge --ask sys-libs/libseccomp dev-db/sqlite dev-libs/libsodium
```

Windows

You don't need to install any dependencies on windows, but you need to use a different build command in the next section.

1.2.2 Building

After all dependencies have been installed, simply build the binary:

```
$ cargo build --release
```

After the build finished the binary is located at `target/release/sn0int`.

1.3 Running your first investigation

This page is going to guide you through the process of setting up your environment and running your first investigation.

1.3.1 Installing the default modules

By default, sn0int doesn't have any modules installed. If you start up sn0int it's going to download some files that it needs and then suggests to install a number of recommended modules:

```
$ sn0int

                   ___/           .
     ____ , __   .'  /\ ` , __   _/_
    (     |'  `. |  / | | |'  `.  |
    `--.  |    | |,'  | | |    |  |
   \___.' /    | /`---' / /    |  \__/

        osint | recon | security
      irc.hackint.org:6697/#sn0int

[+] Connecting to database
[+] Downloading public suffix list
[+] Downloading "GeoLite2-City.mmdb"
[+] Downloading "GeoLite2-ASN.mmdb"
[+] Loaded 0 modules
[*] No modules found, run pkg quickstart to install default modules
[sn0int][default] >
```

Typing `pkg quickstart` is going to get you a fair number of featured modules:

```
[sn0int][default] > pkg quickstart
[+] Installing kpcyrd/asn
[+] Installing kpcyrd/ctlogs
[+] Installing kpcyrd/dns-resolve
[+] Installing kpcyrd/geoip
[+] Installing kpcyrd/hackertarget-subdomains
[+] Installing kpcyrd/otx-subdomains
[+] Installing kpcyrd/passive-spider
[+] Installing kpcyrd/pgp-keyserver
[+] Installing kpcyrd/threatminer-ipaddr
[+] Installing kpcyrd/threatminer-subdomains
[+] Installing kpcyrd/url-scan
[+] Installing kpcyrd/waybackurls
[+] Loaded 12 modules
[sn0int][default] >
```

1.3.2 Adding something to scope

You probably want to separate your investigations so you should select a workspace where your results should go:

```
[sn0int][default] > workspace demo
[+] Connecting to database
[sn0int][demo] >
```

Next, we have to start somewhere and add the first entity to our scope:

```
[sn0int][demo] > add domain
Domain: example.com
[sn0int][demo] >
```

Note: There is a concept of a domain vs a subdomain. We are referring to a domain as everything that is a subdomain of a public suffix. For example, .com is a public suffix, which makes example.com a domain in sn0int terms. Every subdomain of that, like www.example.com, is referred to as a subdomain.

Note that example.com can be added as a subdomain as well since it can hold records. In that case, example.com is both the name of the dns zone, while also being an entity in that zone.

You can confirm this by running a select on the domains we now have:

```
[sn0int][demo] > select domains
#1, "example.com"
[sn0int][demo] >
```

Something we don't need right now, but is going to be useful later on is the ability to filter your entities:

```
[sn0int][demo] > select domains where id=1
#1, "example.com"
[sn0int][demo] >
[sn0int][demo] > select domains where value like %.com
#1, "example.com"
[sn0int][demo] >
[sn0int][demo] > select domains where ( value like e% and value like %m ) or false
#1, "example.com"
[sn0int][demo] >
```

Note: Almost all entities have a `value` column that holds the primary value of the entity.

1.3.3 Running a module

Now that we have something to get started with, we can run our first module. First lets list all modules we have:

```
[sn0int][demo] > pkg list
kpcyrd/asn (0.1.0)
    Run a asn lookup for an ip address
kpcyrd/ctlogs (0.1.0)
    Query certificate transparency logs to discover subdomains
kpcyrd/dns-resolve (0.1.0)
    Query subdomains to discovery ip addresses and verify the record is visible
kpcyrd/geoip (0.1.0)
    Run a geoip lookup for an ip address
kpcyrd/hackertarget-subdomains (0.1.0)
    Query hackertarget for subdomains of a domain
kpcyrd/otx-subdomains (0.1.0)
    Query alienvault otx passive dns for subdomains of a domain
kpcyrd/passive-spider (0.1.0)
    Scrape known http responses for urls
kpcyrd/pgp-keyserver (0.1.0)
    Query pgp keyserver for email addresses
kpcyrd/threatminer-ipaddr (0.1.0)
    Query ThreatMiner passive dns for subdomains of an ip address
kpcyrd/threatminer-subdomains (0.1.0)
    Query ThreatMiner passive dns for subdomains of a domain
kpcyrd/url-scan (0.1.0)
    Scan subdomains for websites
kpcyrd/waybackurls (0.1.0)
    Discover subdomains from wayback machine
[sn0int][demo] >
```

Let's start by querying certificate transparency logs:

```
[sn0int][demo] > use ctlogs
[sn0int][demo][kpcyrd/ctlogs] > run
[*] "example.com"                                     : Subdomain: "www.example.com"
[*] "example.com"                                     : Subdomain: "m.example.com"
[*] "example.com"                                     : Subdomain: "dev.example.com"
[*] "example.com"                                     : Subdomain: "products.example.
↪com"
[*] "example.com"                                     : Subdomain: "support.example.
↪com"
[+] Finished kpcyrd/ctlogs
[sn0int][demo][kpcyrd/ctlogs] >
```

Looks like we've discovered some subdomains here. It might be tempting to throw some of them in a browser but hold on, there's a more efficient way to approach this.

Hint: You can run the modules concurrently with `run -j3`.

1.3.4 Running followup modules on the results

A lot of time has been spent on the database part. While it sort of feels like a no-sql database we are actually enforcing a schema for a reason instead of just using generic dictionaries and calling it a day.

It's crucial that entities created by one module can be picked up by another module, like LEGOs. Let's continue with a module to query the dns records:

```
[sn0int][demo][kpcyrd/ctlogs] > use dns-resolve
[sn0int][demo][kpcyrd/dns-resolve] > run
[*] "www.example.com"                                 : Updating "www.example.com"␣
↪(resolvable => true)
[*] "www.example.com"                                 : IpAddr: 93.184.216.34
[*] "www.example.com"                                 : "www.example.com" -> 93.184.
↪216.34
[*] "m.example.com"                                   : Updating "m.example.com"␣
↪(resolvable => false)
[*] "dev.example.com"                                 : Updating "dev.example.com"␣
↪(resolvable => false)
[*] "products.example.com"                            : Updating "products.example.com
↪" (resolvable => false)
[*] "support.example.com"                             : Updating "support.example.com
↪" (resolvable => false)
[+] Finished kpcyrd/dns-resolve
[sn0int][demo][kpcyrd/dns-resolve] >
```

Two things happened here: We've discovered some IP addresses and added them to scope, and we also updated our subdomain entities with new information, since we now know which of them are resolvable and which aren't.

Let's run the next module, which is actually going to check for websites on them, but let's only target subdomains that we know are resolvable:

```
[sn0int][demo][kpcyrd/dns-resolve] > use url-scan
[sn0int][demo][kpcyrd/url-scan] > target
#1, "www.example.com"
    93.184.216.34
#2, "m.example.com"
#3, "dev.example.com"
#4, "products.example.com"
#5, "support.example.com"
[sn0int][demo][kpcyrd/url-scan] > target where resolvable
[+] 1 entities selected
[sn0int][demo][kpcyrd/url-scan] > target
#1, "www.example.com"
    93.184.216.34
[sn0int][demo][kpcyrd/url-scan] >
```

We can both preview and limit the targets that are going to be passed to the module with the target command. Once we are satisfied with our selection we can run this module:

```
[sn0int][demo][kpcyrd/url-scan] > run
[*] "www.example.com"                                 : Url: "http://www.example.com/
↪" (200)
[*] "www.example.com"                                 : Url: "https://www.example.com/
↪" (200)
[+] Finished kpcyrd/url-scan
[sn0int][demo][kpcyrd/url-scan] >
```

We've now probed both port 80 and port 443 for each subdomain and found two http responses this way. If you want a list of urls you may want to visit in your browser can now query them:

```
[sn0int][demo][kpcyrd/url-scan] > select urls
#1, "http://www.example.com/" (200)
#2, "https://www.example.com/" (200)
[sn0int][demo][kpcyrd/url-scan] >
```

1.3.5 Unscoping entities

Something you are going to run into is that modules are too greedy and add things to the scope we are not interested in. You can delete them using the delete command, but those are likely picked up by a module again.

What you can do instead is setting a flag on an entity that removes it from our scope. This is done using the noscope command:

```
[sn0int][demo] > use ctlogs
[sn0int][demo][kpcyrd/ctlogs] > target
#1, "example.com"
[sn0int][demo][kpcyrd/ctlogs] > add domain
Domain: google.com
[sn0int][demo][kpcyrd/ctlogs] > target
#1, "example.com"
#2, "google.com"
[sn0int][demo][kpcyrd/ctlogs] > noscope domains where value=google.com
[+] Updated 1 rows
[sn0int][demo][kpcyrd/ctlogs] > target
#1, "example.com"
[sn0int][demo][kpcyrd/ctlogs] >
```

Entities that are unscoped are automatically ignored by all modules.

You can reverse this using the scope command:

```
[sn0int][demo][kpcyrd/ctlogs] > target
#1, "example.com"
[sn0int][demo][kpcyrd/ctlogs] > scope domains where true
[+] Updated 2 rows
[sn0int][demo][kpcyrd/ctlogs] > target
#1, "example.com"
#2, "google.com"
[sn0int][demo][kpcyrd/ctlogs] >
```

Hint: All entities have this field, you can refer to it in queries using `unscoped=1`.

1.4 Autonoscope

Instead of manually unscoping everything you can also define so called autonoscope rules. Those are executed from most specific to least specific and the first match wins. If no rule matches, the default is in-scope:

```
[sn0int][demo] > # add the domain first
[sn0int][demo] > # this is necessary because we only want to partially unscope␣
↪example.com
```

(continues on next page)

(continued from previous page)

```
[sn0int][demo] > add domain example.com
[sn0int][demo] >
[sn0int][demo] > # automatically noscope all subdomains
[sn0int][demo] > autonoscope add domain example.com
[sn0int][demo] > # except subdomains of prod.example.com
[sn0int][demo] > autoscope add domain prod.example.com
[sn0int][demo] >
[sn0int][demo] > autonoscope list
  scope domain "prod.example.com"
noscope domain "example.com"
[sn0int][demo] >
[sn0int][demo] > # this is going to be out-of-scope
[sn0int][demo] > add subdomain www.example.com
[sn0int][demo] > # this is going to be in-scope
[sn0int][demo] > add subdomain db.prod.example.com
[sn0int][demo] >
[sn0int][demo] > select subdomains
#1, "www.example.com"
#2, "db.prod.example.com"
[sn0int][demo] > select subdomains where unscoped=0
#2, "db.prod.example.com"
[sn0int][demo] > select subdomains where unscoped=1
#1, "www.example.com"
[sn0int][demo] >
```

1.4.1 Domains

Autonoscope rules for domains are applied to the following structs:

- domains
- subdomains
- urls

Example rules:

```
autonoscope add domain example.com
autonoscope add domain staging.example.com
autonoscope add domain com
autonoscope add domain .
```

1.4.2 IPs

Autonoscope rules for IPs are applied to the following structs:

- ipaddrs
- netblocks
- ports

Example rules:

```
autonoscope add ip 0.0.0.0/0
autonoscope add ip ::/0
autonoscope add ip 192.168.0.0/16
autonoscope add ip 10.13.33.37/32
```

1.4.3 URLs

Autonoscope rules for urls are applied to the following structs:

- urls

Note that these rules are specific to a certain origin (like `https://example.com`) and are used to filter paths.

Example rules:

```
autonoscope add url https://example.com/
autonoscope add url https://example.com/admin/
autonoscope add url https://example.com/a/b/c/d
```

1.5 Writing your first module

Scripting is the core feature in sn0int. It's not strictly required, but if you want to write your own modules, this section is for you.

1.5.1 Creating a repository

It's highly recommended to use a VCS for development, so let's start by setting that up. We're going to assume you store your repos in `~/repos` but you're free to change that to something else:

```
$ git init ~/repos/sn0int-modules
$ cd ~/repos/sn0int-modules
```

Note: If you're using github you can also create a repo from the module repo template.

We need to add this folder to the sn0int config file so it's correctly detected when starting sn0int. Open the config file in your prefered editor. Note that the file does not exist by default and the path is different depending on your operating system. On linux you would open the config file with:

```
$ vim ~/.config/sn0int.toml
```

Add the following:

```
[namespaces]
your_github_name = "~/repos/sn0int-modules"
```

Every module we're adding to `~/repos/sn0int-modules` is now going to be picked up by sn0int.

Make sure you're still in the right folder and add your first module:

```
sn0int new first.lua
```

This is going to generate some boilerplate for you that every module needs to load successfully. Afterwards we can edit it like this:

```
-- Description: ohai wurld
-- Version: 0.1.0
-- Source: domains
-- License: GPL-3.0

function run(arg)
    -- TODO: do something here
end
```

`Description` (mandatory) This should be a short text that describes what your module is doing.

`Version` (mandatory) Every module requires a semver version. You can just set it to `0.1.0` during development, but you need to increase it every time you publish your module. If you don't care about that one, just keep increasing `0.X.0`.

`Source` (mandatory) This is going to specify what kind of entities we're interested in. If we specify `domains` our module is going to be called with all domains that are targeted.

- `domains`
- `subdomains`
- `ipaddrs`
- `urls`
- `emails`

`License` (mandatory) This is somewhat special. We require that every module is licensed under an open source license. Pick one of the following licenses.

- `MIT` - https://opensource.org/licenses/MIT
- `GPL-3.0` - https://opensource.org/licenses/gpl-license
- `LGPL-3.0` - https://opensource.org/licenses/lgpl-license
- `BSD-2-Clause` - https://opensource.org/licenses/BSD-2-Clause
- `BSD-3-Clause` - https://opensource.org/licenses/BSD-3-Clause
- `WTFPL` - https://spdx.org/licenses/WTFPL.html

`function run(arg)` (mandatory) This is where the actual magic of our module happens. Our function is going to be called in a loop for each entity that is targeted by the user.

Let's continue. For the sake of an hello world we're going to take some `domains`, check if a `www` subdomain exists and if it does, add it to the database.

```
-- Description: Scan for www. subdomains
-- Version: 0.1.0
-- Source: domains
-- License: GPL-3.0

function run(arg)
    subdomain = 'www.' .. arg['value']
    info(subdomain)
end
```

This is already enough to execute it. Make sure you've added a domain to scope with `add domain example.com`, save your file and run it like this:

```
sn0int run -f ./first.lua
```

We should see some output by our info function.

Note: `info` is useful for development but you usually want your module to run quietly, so before publishing either remove it or replace it with `debug`.

Next, we want to actually resolve that name, we're going to use the `dns` function for that. This function takes a name and a query type and returns a result. Note that this function might fail, in which case we want to abort our function. We do that by checking if the return value of `last_err()` is truth-y.

```lua
-- Description: Scan for www. subdomains
-- Version: 0.1.0
-- Source: domains
-- License: GPL-3.0

function run(arg)
    subdomain = 'www.' .. arg['value']

    records = dns(subdomain, {
        record='A'
    })
    if last_err() then return end

    info(records)
end
```

If you run your module again you're going to see some output, either `{"answers":[somedata], "error":null}` or `{"answers":[],"error":"NXDomain"}`. If the dns reply doesn't indicate an error this means the subdomain exists and we can add it to our database with `resolvable` being set to `true`.

```lua
-- Description: Scan for www. subdomains
-- Version: 0.1.0
-- Source: domains
-- License: GPL-3.0

function run(arg)
    subdomain = 'www.' .. arg['value']

    records = dns(subdomain, {
        record='A'
    })
    if last_err() then return end

    if records['error'] == nil then
        db_add('subdomain', {
            domain_id=arg['id'],
            value=subdomain,
            resolvable=true,
        })
    end
end
```

Hint: See the database section to understand how the database works in detail.

If we execute our finished module one more time it's going to log that it discovered a subdomain, if it doesn't, try adding more domains to scope. Note that this only happens the first time. Modules that don't discover anything or don't discover anything new exit silently.

There's still some room for improvement, for example, since we already resolved that record, we could also add the ip address to the scope and link it to the subdomain we added.

Hint: For debugging purposes you can increase the verbosity with `sn0int run -v` so database operations are logged even if nothing was changed, or with `sn0int run -vv` to enable `debug()` output.

1.5.2 Publish your module

The public registry uses github usernames to namespace the registry. This means you need to authenticate to the registry using your github username. This can be done using:

```
sn0int login
```

sn0int is going to open a new tab in your browser, if you are already signed into your github account you only need to confirm an authorization request. The application doesn't need any of your data, so it's only asking you to confirm your identity.

Afterwards publish your module with:

```
sn0int publish ./first.lua
```

Please also make sure you publish your repository to github so other people can submit pull requests. The recommended repository location is:

```
https://github.com/<your-username>/sn0int-modules
```

1.5.3 Publish your repo

It is highly recommended to publish your repository on github so people can file issues and pull requests for your module. If you've been following along with the github template you can simply commit your changes and push them.

Your repository would look like one of these:

- https://github.com/kpcyrd/sn0int-modules
- https://github.com/ysf/sn0int-modules
- https://github.com/cybiere/sn0int-modules

1.5.4 Reading data from stdin

Sometimes you need to read data that can't be easily accessed from within the sandbox, like output of other programms or file content. In that case you can write a module that reads from stdin:

```
-- Description: Read from stdin
-- Version: 0.1.0
-- License: GPL-3.0

function run()
```

(continues on next page)

(continued from previous page)

```
    while true do
        x = stdin_readline()
        if x == nil then
            break
        end
        info(x)
    end
end
```

Write it to a file and run it like this:

```
% echo hello | sn0int run --stdin -vvf stdin.lua
[*] anonymous/stdin                                   : "hello\n"
[+] Finished anonymous/stdin
%
```

This is going to read one line at a time and allows you to process it with regular expressions and add data to the database.

Note: If you get an error like `Failed to read stdin: "stdin is unavailable"` make sure the `--stdin` flag is set.

1.6 Database

There are a few things you need to understand how the database works to use it efficiently.

The database that is backing sn0int is sqlite, but the api that is exposed to the user and scripts is an nosql-ish object store. The query language that is exposed to the user is still very similar to sql, except that it lacks a column statement:

```
select subdomains where value like %.example.com
^      ^          ^           ^    ^
|      |          |           |    this value is going to be quoted automatically
|      |          |           |
|      |          |           this triggers automatic quoting
|      |          |
|      |          apply a filter, this translates to sql quite literally
|      |
|      the entity we want to select is a subdomain.
|      this affects the table and the deserializer
|
select entities
```

This is how almost all user facing functions work that operate on the database. The functions that are available for scripting are a bit more object based and described below.

1.6.1 db_add

This operation is somewhat straight forward. It adds an entity to the database:

```
domain_id = db_add('domain', {
    value='example.com',
})
```

If this entity conflicts with an entity that already exists, an upsert is triggered and an db_update is performed instead.

Note: This function may return `nil` if the entity already exists, but has been removed from scope with `noscope`. Everytime you use `db_add` you need to make sure that the ID that has been returned is not `nil`.

1.6.2 db_add_ttl

Add a temporary entity to the database. This is commonly used to insert temporary links that automatically expire over time. If the entity already exists and is also marked as temporary the new ttl is going to replace the old ttl. If the entity already exists but never expires we are not going to add a ttl.

```
-- this link is valid for 2min
domain_id = db_add_ttl('network-device', {
    network_id=1,
    device_id=13,
}, 120)
```

1.6.3 db_activity

Log an activity event. A basic event looks like this:

```
db_activity({
    topic='harness/activity-ping:dummy',
    time=sn0int_time(),
    content={
        a='b',
        foo={
            bar=1337,
        },
        msg='ohai',
    },
})
```

This function is explained in detail in the activity section.

1.6.4 db_update

Update some mutable fields of an entity:

```
db_update('ipaddr', arg, {
    asn=lookup['asn'],
    as_org=lookup['as_org'],
})
```

The first parameter is usually the same arg that your script was called with. Usually you can use db_add instead of db_update due to the upsert feature, but db_update is still slightly faster.

Note: Some fields are immutable and can not be updated.

1.6.5 db_select

This function is used to check if something is in scope. If the entity has been added to the database and has not been removed from scope, this function returns that entities id. This is somewhat similar to `db_add`, except that `db_select` never adds anything to the database.

```
domain_id = db_select('domain', 'example.com')
if domain_id ~= nil then
    -- do something
end
```

This function only accepts a string instead of a lua table. This string is used to filter on the `value` column.

1.7 Structs

This section describes all supported structs in depth. Please refer to this section if in doubt about the correct usage of fields to ensure interoperability between modules.

1.7.1 Domains

Represents a registerable domain as defined by the public suffix list. If in doubt check psl_domain_from_dns_name.

`value` The domain name, like `example.co.uk`.

1.7.2 Subdomains

A subdomain of a *domain*. The depth is arbitrary, so `foo.example.co.uk` and `foo.bar.example.co.uk` are both valid subdomains of `example.co.uk`.

`value` The subdomain, like `foo.bar.example.co.uk`.

`domain_id` The numeric id of a domain struct.

`resolvable` Whether the subdomain can be resolved to a A/AAAA record. nil if unknown.

1.7.3 IpAddrs

An ip address. Note that most of these fields are geoip related and an approximation instead of an actual location.

`value` The ip address.

`family` The address family of the ip address, either `4` or `6`.

`continent` The continent associated with this ip address.

`continent_code` The continent code of the `continent` field, eg `NA`.

`country` The country associated with this ip address.

`country_code` The country code of the `country` field, eg `US`.

`city` The city associated with this ip address.

`latitude` Latitude associated with this ip address.

`longitude` Longitude associated with this ip address.

asn The number of the autonomous system this ip belongs to.

as_org The organization of the autonomous system this ip belongs to.

description This field is sn0int internal if we have additional information about this ip address, for example technical identifiers from aws.

reverse_dns The reverse dns name setup for this ip address.

1.7.4 URLs

subdomain_id The numeric id of a subdomain struct.

value The url, including a schema, hostname and path.

status The http status code, like `200`.

body The raw response body. This can be any mime type.

online Whether or not the url gives a http response (even if it's an error).

title The parsed `<title>` of the page, if available.

redirect If the server replied with a redirect, this is the url it redirected to.

1.7.5 Emails

value The email address.

displayname The display name of a given email address: `this is the name <foo@example.com>`.

valid Whether that email address is valid or has been disabled.

1.7.6 Phonenumbers

value The phone number in E.164 format (+491234567)

name An alias we can assign to this phone number. This alias is sn0int internal.

valid Whether the number is assigned to a customer.

last_online The last time this number has been online.

country The country this number is associated with.

carrier The name of the carrier this numer is registered with.

line The type of the phone number, can be `landline`, `mobile` or `voip`.

is_ported Whether this number has been ported to a different carrier.

last_ported The last time this number has been ported.

caller_name The name of the owner of the phone number.

caller_type The type of caller, eg `business` or `consumer`.

1.7.7 Devices

`value` The devices mac address or another identifier if needed.

`name` An alias we can assign to this device. This alias is sn0int internal.

`hostname` The hostname configured on the device.

`vendor` The hardware vendor of the device. This is usually derived from the mac address.

`last_seen` The last time we've observed the device somewhere.

1.7.8 Networks

A wired or wireless network at a specific location that a device could be connected to.

`value` The network name. This can be an ssid or any other identifier but should be unique.

`latitude` Latitude of the networks location.

`longitude` Longitude of the networks location.

`description` A human readable description in case the value is a technical identifier.

1.7.9 Accounts

A users account or profile on a webservice, like github or instagram.

`service` The identifier of the service/website. It's recommended to use the websites domain for this as defined in *Domains*.

`username` The users unique identifier, like the login name. If the login name is not known or the system doesn't use login names, use the email address instead.

`displayname` The users display name. This name is often not unique and may contain the users real name.

`email` The email address associated with the account.

`url` The url of the public profile if available.

`last_seen` The last time this account has been active/online.

`birthday` The users birthday set on the account.

`phonenumber` The phonenumber associated with the account.

`profile_pic` The blob identifier of the users current profile picture.

1.7.10 Breaches

Either a breach of a specific website, a breach compilation or a breach notification service.

`value` The name of the breach, breach compilation or notification service.

1.7.11 Images

`value` The id that identifies the blob. This id is deterministic based on file content.

`filename` This field is used if we have a well known filename for the content.

mime The image mimetype, like image/png or image/jpeg.

width The width of the image.

height The height of the image.

created The date and time this image has been taken.

latitude Latitude this picture has been taken.

longitude Longitude this picture has been taken.

nudity A score that classifies nudity in this picture. The score goes from 0 to 2 and is commonly calculated with img_nudity. A score above 1 means nudity has been detected.

ahash The Mean (aHash) perceptual hash.

dhash The Gradient (dHash) perceptual hash.

phash The DCT (pHash) perceptual hash.

1.7.12 Ports

The status of a port on an ip address.

ip_addr_id The numeric id of an ipaddr struct.

ip_addr The actual ipaddr.

port The port number.

status The status of the port, either open or closed.

banner The service banner we discovered on this port.

service The service that is running on this port.

version The version of the service running on this port.

1.7.13 Netblocks

A netblock is a network address range that has been allocated to an individual, organization or company. Those are commonly found when running whois lookups on an ip address.

Consider the following example: Running a whois lookup on 140.82.118.4 (one of the addresses currently in use by github) returns that this address belongs to the netrange 140.82.112.0 - 140.82.127.255, so the netblock in this case is 140.82.112.0/20.

family This is either 4 or 6 and populated automatically.

value This is the network range in CIDR notation.

asn The number of the autonomous system this network belongs to.

as_org The organization of the autonomous system this network belongs to.

description This field isn't strictly defined and meant to be used as a human meaningful name if available.

1.7.14 CryptoAddrs

A cryptoaddr is any cryptocurrency address and not tied to a specific currency.

value The address string. This looks like `1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2`.

currency The identifier for a specific currency. This is usually the ticker symbols, like `xbt`, `zec` or `xmr`.

denominator Balance is tracked internally using 64 bit integers (signed, for technical reasons). Balance is supposed to be the lowest unit, so in case of bitcoin you'd write `100,000,000` satoshi instead of `1` bitcoin. Since this value is inconvinient to work with we're using the denominator to display values. In case of bitcoin you'd set it to `8`.

balance The current balance of the address, in the lowest possible unit. In case of bitcoin this would be satoshis.

received The total amount of currency received by this address.

first_seen The first time currency was sent to this address.

last_withdrawal The last time a transaction signed by this address was observed.

description A human readable note for this address.

1.7.15 Activity

Activity is different from all other structs, have a look at the Activity Section.

1.7.16 Relations

Relations are linking two structs together. The link may contain additional information.

subdomain_ipaddr

Links an ip address to a subdomain.

subdomain_id The numeric id of a subdomain struct.

ip_addr_id The numeric id of an ip addr struct.

network_device

Links a device to a network. This is commonly used with `db_add_ttl` so the link automatically expires. This is frequently used to monitor networks for known and unknown devices.

network_id The numeric id of a network struct.

device_id The numeric id of a device struct.

ipaddr The ip address assigned to the device.

last_seen The last time we've seen the device on that network.

breach_email

Links an email to a breach. If we know the password as well we can add it to the link. If we don't know the password we can leave it blank and fill it later. An email can be linked to a breach multiple times with different passwords. There is a special upserting logic in place to support this.

`breach_id` The numeric id of a breach struct.

`email_id` The numeric id of an email struct.

`password` The password for that email in the breach.

1.8 Activity

So far we've learned about regular structs, but activity is special.

Activity is an event tied to a specific time and topic and has a small amount of data piggybacked to it.

1.8.1 Anatomy of an event

`topic` This is some freestyle text used to group events to a specific topic. This must not conflict with other modules unless there's a very good reason.

The topic should look like `kpcyrd/example:something`, with `something` being a meaningful unique identifier for whatever is generating these events, like a mac address or an account name/id.

The rules around this might become stricter in the future.

`time` The most important part of the event: The time and date it happened.

`initial` This value can not be set but might be present in sn0int output. See *Querying events*.

`uniq` (optional) This is an optional feature to deduplicate events. Assuming you're importing posts by an account, you wouldn't want to store a new event for each post you already imported. If you set this field to the technical post id then sn0int would skip the event if it already has an event with the same `topic` and `uniq` combination to avoid inserting duplicates.

`latitude` (optional) Latitude - if you can tie the event to a specific location.

`longitude` (optional) Longitude - if you can tie the event to a specific location.

`radius` (optional) The location radius in meters. If the position you got has a precision of 100 meters set this value to `100`.

`content` Arbitrary data that you want to attach to the event. This doesn't need to be a string and can be an arbitrary object that is then stored as json string.

1.8.2 Logging events

An `activity` event can be logged with `db_activity`:

```
db_activity({
    topic='harness/activity-ping:dummy',
    time=sn0int_time(),
    content={
        a='b',
        foo={
```

(continues on next page)

(continued from previous page)

```
            bar=1337,
        },
        msg='ohai',
    },
})
```

Logging an event that has a location attached could look like this:

```
db_activity({
    topic='harness/activity-ping:dummy',
    time=sn0int_time(),
    latitude=40.726662,
    longitude=-74.036677,
    radius=50,
    content={
        a='b',
        foo={
            bar=1337,
        },
        msg='ohai',
    },
})
```

Making sure an event is not logged twice can be done with `uniq`:

```
-- create the first event
db_activity({
    topic='harness/activity-ping:dummy',
    time=sn0int_time(),
    uniq='12345',
    content='ohai',
})

-- this does nothing because we already have an event with this topic+uniq combination
db_activity({
    topic='harness/activity-ping:dummy',
    time=sn0int_time(),
    uniq='12345',
    content='ohai',
})

-- this creates a new event because uniq is different
db_activity({
    topic='harness/activity-ping:dummy',
    time=sn0int_time(),
    uniq='6789',
    content='ohai',
})

-- this also creates a new event because topic is different
db_activity({
    topic='harness/activity-ping:something-else',
    time=sn0int_time(),
    uniq='6789',
    content='ohai',
})
```

1.8.3 Querying events

There is a commandline interface that can be used to query all events we've logged. To get everything (sorted by time):

```
sn0int activity
```

To limit the output to a specific topic:

```
sn0int activity -t harness/activity-ping:dummy
```

To limit it to a specific time frame:

```
# everything since
sn0int activity --since 2020-01-13T04:20:00
# everything until
sn0int activity --until 2020-01-13T04:20:00
# both
sn0int activity --since yesterday --until today
```

When using `--since` you might also want to know the previous state and use it as an initial value. Consider this example:

```
2020-01-13 14:30:00 # user goes offline
2020-01-13 23:59:00 # user goes online
2020-01-14 09:30:00 # user goes idle
2020-01-14 14:20:00 # user goes offline
```

If we're running a query like `sn0int activity --since 2020-01-14T00:00:00` the program consuming the output wouldn't know that the user is initially online because we're only getting this data:

```
{"id":8,"topic":"foo/bar:asdf","time":"2020-01-14T09:30:00","content":{"state":"idle"}
↪}
{"id":9,"topic":"foo/bar:asdf","time":"2020-01-14T14:20:00","content":{"state":
↪"offline"}}
```

We can tweak this with `sn0int activity --initial --since 2020-01-14T00:00:00` to include one more event that we only use to populate the intial state:

```
{"id":7,"initial":true,"topic":"foo/bar:asdf","time":"2020-01-13T23:59:00","content":{
↪"state":"online"}}
{"id":8,"topic":"foo/bar:asdf","time":"2020-01-14T09:30:00","content":{"state":"idle"}
↪}
{"id":9,"topic":"foo/bar:asdf","time":"2020-01-14T14:20:00","content":{"state":
↪"offline"}}
```

1.8.4 Visualization

There is no visualization built in, there may be external frontends for this in the future. You're very welcome to write one!

1.9 Notifications

If you run sn0int unattended nobody might see the sn0int output. For cases like this you can configure notifications to send you a push notification in case something interesting happens. This is also especially useful if you have sn0int setup to run automatically.

1.9.1 Receiving notifications

Notifications are just regular sn0int modules. You can install them just like any other module or write your own. This section contains walkthroughs on how to setup common integrations.

Telegram

Install the telegram notification module from the registry:

```
sn0int pkg install kpcyrd/notify-telegram
```

Open your telegram app and open a chat with `@botfather`. Send `/newbot` and answer the questions. Copy `bot_token` and open this url in your browser:

Back on your app, open the t.me link to start a new chat with your bot, then send `/start`. Reload the page in your browser, you should see the new message you sent. Copy the `chat_id`.

Test your tokens are working correctly by sending yourself a notification:

```
sn0int notify exec kpcyrd/notify-telegram -o bot_token=1337:foobar -o chat_id=1337
↪'hello world'
```

You should receive `hello world` from your bot on Telegram.

Pushover

Install the pushover notification module from the registry:

```
sn0int pkg install kpcyrd/notify-pushover
```

Signup for pushover and configure the app on your device. Copy th user key visible on the pushover dashboard. Click "Create an Application/API Token". Set "sn0int" as name and set an icon if you want to. Copy the api token.

Test your tokens are working correctly by sending yourself a notification:

```
sn0int notify exec kpcyrd/notify-pushover -o user_key=asdf1337 -o api_token=asdf1337
↪'hello world'
```

You should receive `hello world` as a push notification.

Discord

Install the discord notification module from the registry:

```
sn0int pkg install kpcyrd/notify-discord
```

Decide which channel should receive notifications (or create a new one). Open the "Server Settings" of your discord server. Click on "Webhooks". Click "Create Webhook". Configure the Name and Channel. Copy the Webhook URL.

Test your tokens are working correctly by sending yourself a notification:

```
sn0int notify exec kpcyrd/notify-discord -o url=https://discord.com/api/webhooks/1337/
↪asdf 'hello world'
```

You should receive `hello world` in your discord channel.

Signal

Install the sn0int notification module from the registry:

```
sn0int pkg install kpcyrd/notify-signal
```

This module allows end-to-end encrypted notifications, but it's also difficult to setup. You need a second phone number and install both signal-cli and sn0int-signal.

After you've registered your second phone number with signal-cli, you can use sn0int-signal to expose a minimal api for notify-signal. For more detailed instructions and how to start the api at boot, see the sn0int-signal README.

Read the secret key generated at `/etc/sn0int-signal.key` and send a notification to the signal phone number:

```
sn0int notify exec kpcyrd/notify-signal -o to=+31337 -o secret=asdf 'hello world'
```

You should receive `hello world` from the number signed up with signal-cli.

Writing your own module

Make sure you've read the detailed instructions on how to get setup with module development.

Create a new sn0int module like this:

```
sn0int new ~/repos/sn0int-modules/notify-custom.lua
```

Edit the `-- Source:` so it takes notifications as input:

```
-- Description: TODO your description here
-- Version: 0.1.0
-- License: GPL-3.0
-- Source: notifications

function run(arg)
    -- TODO your code here
    -- https://sn0int.readthedocs.io/en/stable/reference.html

    debug(arg)
    info(arg['subject'])
    info(arg['body'])
end
```

Execute your script:

```
sn0int notify exec notify-custom 'hello world'
```

You most likely need to pass options to avoid hard-coding keys into your script. Options can be fetched like this:

```
-- Description: TODO your description here
-- Version: 0.1.0
-- License: GPL-3.0
-- Source: notifications

function run(arg)
```

(continues on next page)

(continued from previous page)

```
    -- TODO your code here
    -- https://sn0int.readthedocs.io/en/stable/reference.html

    local foo = getopt('foo')
    if not foo then return 'Missing -o foo= option' end

    info('foo: ' .. foo)
    info('subject: ' .. arg['subject'])
end
```

And passed like this:

```
sn0int notify exec notify-custom -o "foo=hello world" 'ohai'
```

1.9.2 Setting up notification rules

We now know how to trigger notifications manually, but we would rather trigger notifications if a module runs into something interesting.

You can setup subscriptions on specific topics and then have a notification script execute automatically.

Lookup the location of your sn0int config file:

```
sn0int paths
```

And open it in an editor of your choice:

```
vim /home/user/.config/sn0int.toml
```

A basic configuration could look like this:

```
# You can have multiple notification sections, this one is named
# `demo-telegram-integration`
# The label can be set to whatever you want, but you may need to add
# double-quotes to use some characters.
[notifications.demo-telegram-integration]
# If this option is present, the notification must originate from one of
# the following workspaces.
workspaces = ["default", "some-workspace"]
# If this option is present, the notification must match one of the
# filters. You can use `*` as a wildcard to match everything except `:`.
topics = ["activity:harness/activity-ping:*"]
# Mandatory: the module to execute.
script = "kpcyrd/notify-telegram"
# The options to pass to the module, if any.
# Can be accessed with `getopt`
options = [
    "bot_token=1337:foobar",
    "chat_id=1337",
]
```

All options except `script` are optional, but setting filters is highly recommended.

1.9.3 Testing notifications

To test if your configuration works correctly you can create an event manually:

```
sn0int -w some-workspace notify send activity:harness/activity-ping:dummy "hello world
↪"
```

If it matches any of your rules you should receive a push notifications.

Note: If you want to test just the routing without actually sending something, add `--dry-run`.

1.9.4 Running sn0int automatically

Support for this is going to improve in the future, but you can already set this up if you're ok with a slightly buggy experience.

Monitors

Some modules are long-running and either wait for an event from a server or have custom polling built in that's usually configurable with an `-o interval=` option. If your module has a non-trivial setup phase, an author may take this approach.

Enable the service to run on boot:

```
systemctl enable --now sn0int-your-new-service.service
```

Timers

If the module is only one-shot you can set it up to run with a timer:

Setup the timer like this:

```
systemctl enable --now sn0int-your-other-service.timer
```

1.10 Keyring

A common problem is that you need either an api key or a username/password combination. Instead of hardcoding it in the script you should request them from the keyring. In order to do this you need to request permissions to those credentials.

1.10.1 Managing the keyring

The keyring is a simple namespaced key-value store:

```
[sn0int][default] > keyring add aws:AKIAIOSFODNN7EXAMPLE
Secretkey: keep-this-secret
[sn0int][default] > keyring list
aws:AKIAIOSFODNN7EXAMPLE
```

(continues on next page)

(continued from previous page)

```
[sn0int][default] >
[sn0int][default] > keyring list aws
aws:AKIAIOSFODNN7EXAMPLE
[sn0int][default] > keyring list instagram
[sn0int][default] >
[sn0int][default] > keyring get aws:AKIAIOSFODNN7EXAMPLE
Namespace:     "aws"
Access Key:    "AKIAIOSFODNN7EXAMPLE"
Secret:        "keep-this-secret"
[sn0int][default] >
```

If the service uses a username-password combination, set the username as the access key and the password as the secret.

If the service uses only a secret key for the api, set the secret key as the access key and leave the secret blank.

A script doesn't automatically get access to requested keyring namespaces. Instead the user is asked to confirm those requests to limit abusive scripts.

1.10.2 Using access keys in scripts

We can request all keys of a certain namespace in our script metadata. This is going to prompt the user to grant the script access. This can be done for multiple namespaces in the same script:

```
-- Keyring-Access: aws
-- Keyring-Access: asdf
```

If the user granted us access to those keys we can read them with `keyring`:

```
creds = keyring('aws')
debug(creds[1]['access_key'])
debug(creds[1]['secret_key'])
```

This returns a list of all keys in that namespace. Any empty list is returned if the user doesn't have any keys in that namespace.

If you want to allow the user to select a specific script you can introduce an option that is set by the user and then filter `creds` until the `access_key` matches.

1.10.3 Using access keys as source argument

We can also use the access keys as source argument. This is useful if each account has access to different things and we want to read through all of them.

Since access key permissions are granted per namespace we need to specify which credentials we want to use.

```
-- Keyring-Access: aws
-- Source: keyring:aws
```

1.11 Configuration

This section documents the config file. By default this file does not exist and a default configuration is used instead.

Linux/BSD `~/.config/sn0int.toml`

OSX `~/Library/Preferences/sn0int.toml`

Windows `%APPDATA%/sn0int.toml`

1.11.1 [core]

`registry` Configure the registry you want to use. Defaults to `https://sn0int.com`.

`no-autoupdate` sn0int is going to check if your modules are outdated during startout once a week. Set this option to `true` to disable this.

1.11.2 [namespaces]

By default sn0int modules are assumed to be installed from the registry. You may want to keep a local directory with private modules, especially during development. You can configure a folder that contains modules that aren't managed by sn0int by adding a namespace section to the config file:

```
[namespaces]
foo = "/opt/sn0int/foo"
bar = "~/repos/a/b/c/sn0int-modules"
```

This is going to load modules from these two folders and register them in the `foo` and `bar` namespace.

Note that sn0int is also going to assume that symlinks in `~/.local/share/sn0int/modules` and folders containing a `.git` folder are externally managed.

1.11.3 [network]

To enable a proxy, add the following to your config file:

```
[network]
proxy = "127.0.0.1:9050"
```

This forces everything through tor (or any other socks5 proxy) and restricts all other functions that depend on the network. For example the `dns` function is fully disabled if a proxy is configured.

1.12 Sandbox

Scripts are generally considered to be untrusted and executed exclusively in a child process. It's important to note that there's a basic sandbox that's active on every operating system, and there's a second line of defense on supported operating systems.

The first line of defense is the restrictive stdlib. It's assumed that an attacker gains full control over the lua code and is able to call any function with arbitrary arguments. The stdlib only provides functions that are considered safe, so for example it's not possible to start a process or open a file.

The second line of defense is supposed to make sure the system isn't compromised even if the first layer is fully broken and an attacker gains full control over the child process.

Right now this is only supported on linux and openbsd.

1.12.1 Linux

On linux we use seccomp to filter all syscalls that we don't need. We also use chroot to disable filesystem access. It's recommended to install the sn0int binary with `cap_sys_chroot` to make sure unprivileged users can use chroot. The chroot location is hard coded and all capabilities are removed after the chroot is done or if no chroot is going to happen.

1.12.2 OpenBSD

On openbsd we're using `pledge` to restrict syscalls and `unveil` to restrict filesystem access.

1.12.3 IPC Protocol

The parent process and the child process communicate using an IPC protocol that is line-based json.

For a simple hello world the parent process is only going to send a single line to the child process. This line contains:

- The function argument
- The dns config
- Keys that the module has been given access to
- The module metadata and code
- Options, if any
- A socks5 proxy, if any
- The log level

```
{"arg":null,"dns_config":{"ns":["1.1.1.1:53","1.0.0.1:53"],"tcp":false,"timeout":{"nanos":0,"secs":3}},"keyring":[],"module":{"author":"anonymous","description":"basic selftest","keyring_access":[],"name":"selftest","script":{"code":"-- Description: basic selftest\n-- Version: 0.1.0\n-- License: GPL-3.0\n\nfunction run()\n    -- nothing to do here\nend\n"},"source":null,"version":"0.1.0"},"options":{},"proxy":null,"verbose":2}
```

Saving this line in a file called `start.json` and sending it to a sandbox process should result in the following output:

```
$ sn0int sandbox foobar < start.json
{"Exit":"Ok"}
$
```

This line tells us that the script terminated successfully.

There are some functions that cause a notification to the parent process. We are going to add a call to the `info()` function to our module:

```
{"arg":null,"dns_config":{"ns":["1.1.1.1:53","1.0.0.1:53"],"tcp":false,"timeout":{"nanos":0,"secs":3}},"keyring":[],"module":{"author":"anonymous","description":"basic selftest","keyring_access":[],"name":"selftest","script":{"code":"-- Description: basic selftest\n-- Version: 0.1.0\n-- License: GPL-3.0\n\nfunction run()\n    info('ohai')\nend\n"},"source":null,"version":"0.1.0"},"options":{},"proxy":null,"verbose":2}
```

This is going to print an additional event:

```
$ sn0int sandbox foobar < start2.json
{"Log":{"Info":"\"ohai\""}}
{"Exit":"Ok"}
$
```

There are some functions that block the child process until the parent process sent a reply. These functions are mostly database related functions, since the child doesn't have direct database access. To demonstrate this, we're going to write two lines to our file this time, one is the init line and the second one is the reply for the database event:

```
{"arg":null,"dns_config":{"ns":["1.1.1.1:53","1.0.0.1:53"],"tcp":false,"timeout":{"nanos":0,"secs":3}},"keyring":[],"module":{"author":"anonymous","description":"basic selftest","keyring_access":[],"name":"selftest","script":{"code":"-- Description: basic selftest\n-- Version: 0.1.0\n-- License: GPL-3.0\n\nfunction run()\n    x = db_add('domain', {value=\"example.com\"})\n    info(x)\nend\n"},"source":null,"version":"0.1.0"},"options":{},"proxy":null,"verbose":2}
{"Ok":1337}
```

Results in the following output:

```
$ target/release/sn0int sandbox foobar < start3.json
{"Database":{"Insert":{"Domain":{"value":"example.com"}}}}
{"Log":{"Info":"1337.0"}}
{"Exit":"Ok"}
$
```

The first line is a database event and indicates that the child wants to insert data. After printing this line the child tries to read a line from stdin, this is why we needed to write two lines to our json file this time. In the second line the child learns if the insert was successful and which id was assigned to that entity.

1.12.4 Limitations

There are some limitations that you should be aware:

- Network access is available and network namespaces aren't isolated. This means scripts have access to your local network, the internet and also your localhost loopback interface.
- If chroot is unavailable an attacker could connect to unix domain sockets.

1.12.5 Diagnosing a sandbox failure

You might experience a sandbox failure, especially on architectures that are less popular. This usually looks like this:

```
[sn0int][example][kpcyrd/ctlogs] > run
[-] Failed "example.com": Sandbox child has crashed
[+] Finished kpcyrd/ctlogs (1 errors)
```

A module that never finishes could also mean an IO thread inside the worker got killed by the sandbox.

You can try to diagnose this yourself with strace:

```
strace -f sn0int run -vv ctlogs 2>&1 | tee strace.log
```

Open `strace.log`, look out for syscalls that didn't return by searching for `= ?` and ignore calls to exit and similar. You are looking for something like this:

```
seccomp(SECCOMP_SET_MODE_FILTER, 0, {len=48, filter=0xdd59094e490}) = 0
write(1, "[+] activated!\n", 15[+] activated!
)         = 15
getresuid( <unfinished ...>)              = ?
+++ killed by SIGSYS (core dumped) +++
```

This would indicate a call to `getresuid` which was not allowed by the seccomp filter.

If you don't want to diagnose this yourself open a new bug report with as much information as possible, specifically which distro, which release and which architecture you're using.

1.13 Function reference

1.13.1 asn_lookup

Run an ASN lookup for a given ip address. The function returns `asn` and `as_org`. This function may fail.

```
lookup = asn_lookup('1.1.1.1')
if last_err() then return end
```

1.13.2 base64_decode

Decode a base64 string with the default alphabet+padding.

```
base64_decode("ww==")
```

1.13.3 base64_encode

Encode a binary array with base64 and the default alphabet+padding.

```
base64_encode("\x00\xff")
```

1.13.4 base64_custom_decode

Decode a base64 string with custom alphabet+padding.

```
-- base64
base64_custom_decode('b2hhaQ==',
↪'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/', '=')
-- base64 no padding
base64_custom_decode('b2hhaQ',
↪'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/', '')
-- base64 url safe
base64_custom_decode('b2hhaQ==',
↪'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_', '=')
```

1.13.5 base64_custom_encode

Encode a binary array with base64 and custom alphabet+padding.

```
-- base64
base64_custom_encode('ohai',
↪'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/', '=')
-- base64 no padding
base64_custom_encode('ohai',
↪'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/', '')
-- base64 url safe
base64_custom_encode('ohai',
↪'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_', '=')
```

1.13.6 base32_custom_decode

Decode a base32 string with custom alphabet+padding.

```
-- rfc-4648 base32
base32_custom_decode('N5UGC2I=', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ234567', '=')
-- z-base-32
base32_custom_decode('p7wgn4e', 'ybndrfg8ejkmcpqxot1uwisza345h769', '')
```

1.13.7 base32_custom_encode

Encode a binary array with base32 and custom alphabet+padding.

```
-- rfc-4648 base32
x = base32_custom_encode('ohai', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ234567', '=')
-- z-base-32
x = base32_custom_encode('ohai', 'ybndrfg8ejkmcpqxot1uwisza345h769', '')
```

1.13.8 clear_err

Clear the last recorded error from the internal state. See also *last_err*.

```
if last_err() then
    -- ignore this error
    clear_err()
end
```

1.13.9 create_blob

Push a byte array into persistent blob storage. This allows passing those bytes to functions operating on blob storage. Returns a blob identifier that is deterministic based on the blob content. Blobs are immutable.

```
blob = create_blob("some bytes")
debug(blob)
```

1.13.10 datetime

Return current time in UTC. This function is suitable to determine datetimes for `DATETIME` database fields.

```
now = datetime()
```

Note: This format is sn0int specific, to get the current time for scripting use *time_unix* instead.

Warning: This function is going to be deprecated at some point. Prefer *sn0int_time* for new scripts.

1.13.11 db_add

Add an entity to the database or update it if it already exists. This function may fail or return `nil`. See db_add for details.

```
domain_id = db_add('domain', {
    value='example.com',
})
```

1.13.12 db_add_ttl

Add a temporary entity to the database. This is commonly used to insert temporary links that automatically expire over time. If the entity already exists and is also marked as temporary the new ttl is going to replace the old ttl. If the entity already exists but never expires we are not going to add a ttl.

```
-- this link is valid for 2min
domain_id = db_add_ttl('network-device', {
    network_id=1,
    device_id=13,
}, 120)
```

1.13.13 db_activity

Log an activity event. A basic event looks like this:

```
db_activity({
    topic='harness/activity-ping:dummy',
    time=sn0int_time(),
    content={
        a='b',
        foo={
            bar=1337,
        },
        msg='ohai',
    },
})
```

This function is explained in detail in the activity section.

1.13.14 db_select

Checks if a target is in scope. If non-nil is returned, this entity is in scope. This function may fail. See db_select for details.

```
domain_id = db_select('domain', 'example.com')
if domain_id ~= nil then
    -- do something
end
```

1.13.15 db_update

Update an entity in the database. This function may fail. See db_update for details.

```
db_update('ipaddr', arg, {
    asn=lookup['asn'],
    as_org=lookup['as_org'],
})
```

1.13.16 dns

Resolve a dns record. If the dns query was successful and the dns reply is `NoError` then `x['error']` is `nil`. The records of the reply are in `x['answers']`. This function may fail.

This function accepts the following options:

`record` The `query_type`, can be any of `A`, `AAAA`, `MX`, `AXFR`, etc.

`nameserver` The server that should be used for the lookup. Defaults to your system resolver.

`tcp` If the lookup should use tcp, true/false.

`timeout` The time until the query times out in milliseconds.

```
records = dns('example.com', {
    record='A',
})
if last_err() then return end
if records['error'] ~= nil then return end
records = records['answers']
```

Note: DNS replies with an error code set are not causing a change to `last_err()`. You have to test for this explicitly.

Note: This function is unavailable if a socks5 proxy is configured.

1.13.17 error

Log an error to the terminal.

```
error('ohai')
```

1.13.18 geoip_lookup

Run a geoip lookup for a given ip address. The function returns:

- continent
- continent_code
- country
- country_code
- city
- latitude
- longitude

This function may fail.

```
lookup = geoip_lookup('1.1.1.1')
if last_err() then return end
```

1.13.19 hex

Hex encode a list of bytes.

```
hex("\x6F\x68\x61\x69\x0A\x00")
```

1.13.20 hmac_md5

Calculate an hmac with md5. Returns a binary array.

```
hmac_md5("secret", "my authenticated message")
```

1.13.21 hmac_sha1

Calculate an hmac with sha1. Returns a binary array.

```
hmac_sha1("secret", "my authenticated message")
```

1.13.22 hmac_sha2_256

Calculate an hmac with sha2_256. Returns a binary array.

```
hmac_sha2_256("secret", "my authenticated message")
```

1.13.23 hmac_sha2_512

Calculate an hmac with sha2_512. Returns a binary array.

```
hmac_sha2_512("secret", "my authenticated message")
```

1.13.24 hmac_sha3_256

Calculate an hmac with sha3_256. Returns a binary array.

```
hmac_sha3_256("secret", "my authenticated message")
```

1.13.25 hmac_sha3_512

Calculate an hmac with sha3_512. Returns a binary array.

```
hmac_sha3_512("secret", "my authenticated message")
```

1.13.26 html_select

Parses an html document and returns the first element that matches the css selector. The return value is a table with *text* being the inner text and *attrs* being a table of the elements attributes.

```
csrf = html_select(html, 'input[name="csrf"]')
token = csrf["attrs"]["value"]
```

1.13.27 html_select_list

Same as *html_select* but returns all matches instead of the first one.

```
html_select_list(html, 'input[name="csrf"]')
```

1.13.28 http_mksession

Create a session object. This is similar to `requests.Session` in python-requests and keeps track of cookies.

```
session = http_mksession()
```

1.13.29 http_request

Prepares an http request. The first argument is the session reference and cookies from that session are copied into the request. After the request has been sent, the cookies from the response are copied back into the session.

The next arguments are the `method`, the `url` and additional options. Please note that you still need to specify an empty table `{}` even if no options are set. The following options are available:

`query` A map of query parameters that should be set on the url.

`headers` A map of headers that should be set.

basic_auth Configure the basic auth header with `{"user", "password"}`.

user_agent Overwrite the default user agent with a string.

json The request body that should be json encoded.

form The request body that should be form encoded.

follow_redirects Automatically follow redirects, up to the specified number. If set to 1, only one redirect is going to be followed. Defaults to 0 so redirects aren't followed.

body The raw request body as string.

into_blob If true, the response body is stored in blob storage and a blob reference is returned as `blob` instead of the full body.

proxy Use a socks5 proxy in the format `127.0.0.1:9050`. This option only works if it doesn't conflict with the global proxy settings.

binary Set to `true` to get the http response as raw bytes.

This function may fail.

```
req = http_request(session, 'POST', 'https://httpbin.org/post', {
    json={
        user=user,
        password=password,
    }
})
resp = http_send(req)
if last_err() then return end
if resp['status'] ~= 200 then return 'http status error: ' .. resp['status'] end
```

1.13.30 http_send

Send the request that has been built with *http_request*. Returns a table with the following keys:

status The http status code

headers A table of headers

text The response body as string

binary The response body as bytes (if `binary=true`)

blob If `into_blob` was enabled for the request the body is downloaded into blob storage with a reference to the body in this field.

```
req = http_request(session, 'POST', 'https://httpbin.org/post', {
    json={
        user=user,
        password=password,
    }
})
resp = http_send(req)
if last_err() then return end
if resp['status'] ~= 200 then return 'http status error: ' .. resp['status'] end
```

1.13.31 http_fetch

This does an *http_send* and also automatically validate the status code.

Note: You almost always want this when setting the `into_blob` option since this function validates the status code *before* inserting the response body into blob storage.

```
-- short form
data = http_fetch(req)
if last_err() then return end

-- long form
resp = http_send(req)
if last_err() then return end
if resp['status'] ~= 200 then return 'http status error: ' .. resp['status'] end
```

1.13.32 http_fetch_json

Identical to *http_fetch* but also automatically parses the response body as json.

```
-- short form
data = http_fetch_json(req)
if last_err() then return end

-- long form
resp = http_send(req)
if last_err() then return end
if resp['status'] ~= 200 then return 'http status error: ' .. resp['status'] end
data = json_decode(resp['text'])
if last_err() then return end
```

1.13.33 img_load

Attempt to decode a blob as an image and return some basic metadata like the mime type, height and width.

```
img = img_load(blob)
if last_err() then return end
debug(img)
```

1.13.34 img_exif

Extract exif metadata from an image.

```
exif = img_exif(blob)
if last_err() then return end
debug(exif)
```

1.13.35 img_ahash

Calculate the Mean (aHash) perceptual hash.

```
hash = img_ahash(blob)
if last_err() then return end
debug(hash)
```

1.13.36 img_dhash

Calculate the Gradient (dHash) perceptual hash.

```
hash = img_dhash(blob)
if last_err() then return end
debug(hash)
```

1.13.37 img_phash

Calculate the DCT (pHash) perceptual hash.

```
hash = img_phash(blob)
if last_err() then return end
debug(hash)
```

1.13.38 img_nudity

Classify an image for nudity. The score goes from 0 to 2. A score above 1 means nudity has been detected.

```
nudity = img_nudity(blob)
if last_err() then return end
debug(nudity)
```

1.13.39 info

Log an info to the terminal.

```
info('ohai')
```

1.13.40 intval

Parse a number from a string.

> x = strval(‘1234’)

1.13.41 json_decode

Decode a lua value from a json string.

```
json_decode("{\"data\":{\"password\":\"fizz\",\"user\":\"bar\"},\"list\":[1,3,3,7]}")
```

1.13.42 json_decode_stream

Very similar to *json_decode*, but works with multiple json objects directly concatenated to each other or separated by newlines.

```
json_decode_stream("{\"data\":1}{\"data\":2}")
```

1.13.43 json_encode

Encode a datastructure into a string.

```
x = json_encode({
    some=1,
    fancy={
        data='structures',
    }
})
print(x)
```

1.13.44 key_trunc_pad

Truncate/pad a key to a given length.

```
-- if longer than 32 bytes: truncate to 32
-- if shorter than 32 bytes: pad with \x00
local key = key_trunc_pad(password, 32, 0)
```

1.13.45 keyring

Request all keys from a given namespace. See the keyring section for details.

```
creds = keyring('aws')
print(creds[1]['accesskey'])
print(creds[1]['secretkey'])
```

1.13.46 last_err

Returns infos about the last error we've observed, if any. Returns `nil` otherwise.

```
if last_err() then
    -- Something went wrong, abort
    return
end
```

1.13.47 md5

Hash a byte array with md5 and return the results as bytes.

```
hex(md5("\x00\xff"))
```

1.13.48 mqtt_connect

Connect to an mqtt broker.

```
local sock = mqtt_connect('mqtts://mqtt.example.com', {
    username='foo',
    password='secret',
})
if last_err() then return end
```

1.13.49 mqtt_subscribe

Subscribe to a topic. Right now only QoS 0 is supported.

```
mqtt_subscribe(sock, '#', 0)
if last_err() then return end
```

1.13.50 mqtt_recv

Receive an mqtt packet. This is not necessarily a publish packet and more packets might be added in the future, so you need to check the type specifically.

If a read timeout has been set with *mqtt_connect* this function returns `nil` in case of a read timeout.

```
local pkt = mqtt_recv(sock)
if last_err() then return end
if pkt == nil then
    -- read timeout, consider sending a ping or disconnect if the previous ping failed
elseif pkt['type'] == 'pong' then
    -- broker sent a pong
elseif pkt['type'] == 'publish' then
    local payload = utf8_decode(pkt['body'])
    if last_err() then return end
    info(payload)
end
```

1.13.51 mqtt_ping

Send a pingreq packet, causing the broker to send a pingresp. This is used to make sure the connection is still working correctly.

```
mqtt_ping(sock)
if last_err() then return end
```

1.13.52 pgp_pubkey

Same as *pgp_pubkey_armored*, but without the unarmor step.

1.13.53 pgp_pubkey_armored

Extract `uids`, `sigs` and the `fingerprint` out of an rfc 4880 pgp public key. This function may fail.

```
key = pgp_pubkey_armored([===[
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2

mQENBFu6q90BCADgD7Q9aH5683yt7hzPktDkAUNAZJHwYhUNeyGK43frPyDRWQmq
N+oXTfiYWLQN+d7KNBTnF9uwyBdaLM7SH44lLNYo8W09mVM2eK+wt19uf5HYNgAE
8la45QLo/ce9CQVe1a4oXNWq6l0FOY7M+wLe+G2wMwz8RXGgwd/qQp4/PB5YpUhx
nAnzClxvwymrL6BQXsRcKSMSD5bIzIv95n105CvW5Hql7JR9zgOR+gHqVOH8HBUc
ZxMumrTM6aKLgAhgM8Sn36gCFOfjlG1b1OFLZhUtgro/nnEOmAurRsCZy8M5h8QM
FpZChIH8kgHs90F/CCvGjMq3qvWcH8ZsPUizABEBAAG0NUhhbnMgQWNrZXIgKGV4
YW1wbGUgY29tbWVudCkgPGhhbnMuYWNrZXJAZXhhbXBsZS5jb20+iQFOBBMBCAA4
FiEEyzeO1eEwbB03hcqBM00IodGdlj8FAlu6q90CGwMFCwkIBwIGFQgJCgsCBBYC
AwECHgECF4AACgkQM00IodGdlj/AJQgAjmk+iP5b7Jt7+f+lU4Oprlf3f3DG/uh5
Ge6MjV7cvtxlhZJRD5hxGt9RwwnEp61TBSbrem288pM89ilQfTNe0wUr9OzwWzh/
8Ngl5iWnD2ah3Mpi5R1V/YMNf2cnwVjqNvfkRHdNc43pZOkC2GoiTUn0QY0UBpOW
ZMN3//ANi6ZtiK/L0IZQND/gKvOzu/4tfaJeBl26T3cVYj53p3G3jhlb92vVa8SR
uL3S3bzd1h5snDgU1uXHmNHGbhkEc4KUneQ0V9/bdZrg6OzFAfM1ghgfoId+YpQH
er9L26ISL3QF58wdEXfIdHYEmMlANjBMO2cUlQXgONuCgkMuY7GBmrkBDQRbuqvd
AQgA41jqCumCxYV0NdSYNnTSSDRyd69dOUYCAPT80iZ739s7KKJS9X9KVfGmDjfi
u2RcfR/KYj53HoyOm4Pm/+ONN8De4ktzXpIpJxGC+O8NBvd9vkboAS6qnCjK7KVE
r91ymxxVKp2dzZvVfpIjWVZR5i2EAvS5vw8UK4gL8ALH+S9leJFZrQWcgyoJOJzH
Rzr9pesX2HvdgcNG1O6QUArlsnsTnqpi/hu7tQa8tifBpWDeArOA23Y2DgeehdDF
lSU/8KD4J+AkFrWWlcTaMsvSChXQkCHEMRIcSOfXtdpX5KJSE7UBQdD1opm+mR79
VeHnuJAAVZZtUZmJA7pjdKykYQARAQABiQE2BBgBCAAgFiEEyzeO1eEwbB03hcqB
M00IodGdlj8FAlu6q90CGwwACgkQM00IodGdlj8bMAf+Lq3Qive4vcrCTT4IgvVj
arOACdcbtt5RhVBTimT19rDWNH+m+PfPjo3FSlBj5cm70KAXUS2LBFFxhakTZ/Mq
cQroWZpVbBxj4kipEVVJZFdUZQaDERJql0xYGOQrNMQ4JGqJ84BRrtOExjSqo41K
hAhNe+bwPGH9/Igiixc4tH07xa7TOy4MyJv/6gpbHy/lW1hqpCAgM5fT/im5/6QF
k0tED6vIuc54IWiOmwCnjZiQnJ8uCwEu+cuJ5Exwy9CNERLp5v0y4eG+0E+at9j/
macOg39qf09t53pTqe9dWv5NIi319TeBsKZ2lb0crrQjsbHqk0DAUwgQuoANqLku
vA==
=kRIv
-----END PGP PUBLIC KEY BLOCK-----
]===])

if last_err() then return end
print(key)
```

1.13.54 print

Write something directly to the terminal.

```
print({
    some=1,
    fancy={
        data='structures',
    }
})
```

Warning: This function writes directly to the terminal and can interfere with other terminal features. This function should be used during development only.

1.13.55 psl_domain_from_dns_name

Returns the parent domain according to the public suffix list. For `www.a.b.c.d.example.co.uk` this is going to be `example.co.uk`.

```
domain = psl_domain_from_dns_name('www.a.b.c.d.example.co.uk')
print(domain == 'example.co.uk')
```

1.13.56 ratelimit_throttle

Create a ratelimit that can only be passed x times every y milliseconds. This limit is global for a single `run` and also works with threads.

```
-- allow this to pass every 250ms
ratelimit_throttle('foo', 1, 250)
-- allow this to pass not more than 4 times per second
ratelimit_throttle('foo', 4, 1000)
```

This is useful if you need to coordinate your executions to stay below a certain request threshold.

1.13.57 regex_find

Apply a regex to some text. Returns `nil` if the regex didn't match and the capture groups if it did.

```
m = regex_find(".(.)", "abcdef")

if m == nil then
    print('No captures')
end

print(m[1] == 'ab')
print(m[2] == 'b')
```

1.13.58 regex_find_all

Same as *regex_find*, but returns all matches.

```
m = regex_find_all(".(.)", "abcdef")

print(m[1][1] == 'ab')
print(m[1][2] == 'b')
print(m[2][1] == 'cd')
print(m[2][2] == 'd')
print(m[3][1] == 'ef')
print(m[3][2] == 'f')
```

1.13.59 semver_match

Compare a version to a version requirement. This can be used with *sn0int_version* to test for certain features or behavior.

```
semver_match('=0.11.2', sn0int_version())
semver_match('>0.11.2', sn0int_version())
semver_match('<0.11.2', sn0int_version())
semver_match('~0.11.2', sn0int_version())
semver_match('^0.11.2', sn0int_version())
semver_match('0.11.2', sn0int_version()) -- synonym for ^0.11.2
semver_match('<=0.11.2', sn0int_version())
semver_match('>=0.11.2', sn0int_version())
semver_match('>=0.4.0, <=0.10.0', sn0int_version())
```

1.13.60 set_err

Manipulate the global error object. If you want to exit the main `run` function with an error you can simply return a string, but those are difficult to propagate through functions. `set_err` specifically assigns an error to the global error object that are also used by all other rust functions.

```
function foo()
    set_err("something failed")
end

foo()
if last_err() then return end
```

1.13.61 sha1

Hash a byte array with sha1 and return the results as bytes.

```
hex(sha1("\x00\xff"))
```

1.13.62 sha2_256

Hash a byte array with sha2_256 and return the results as bytes.

```
hex(sha2_256("\x00\xff"))
```

1.13.63 sha2_512

Hash a byte array with sha2_512 and return the results as bytes.

```
hex(sha2_512("\x00\xff"))
```

1.13.64 sha3_256

Hash a byte array with sha3_256 and return the results as bytes.

```
hex(sha3_256("\x00\xff"))
```

1.13.65 sha3_512

Hash a byte array with sha3_512 and return the results as bytes.

```
hex(sha3_512("\x00\xff"))
```

1.13.66 sleep

Pause the current program for the specified number of seconds. This is usually only used for debugging.

```
sleep(1)
```

1.13.67 sn0int_time

Return current time in UTC. This function is suitable to determine datetimes for `DATETIME` database fields.

```
now = sn0int_time()
```

Note: This format is sn0int specific, to get the current time for scripting use *time_unix* instead.

1.13.68 sn0int_time_from

Identical to *sn0int_time* but uses a unix timestamp in seconds instead of the current time. This function is compatible with *time_unix* and *strptime*.

```
time = sn0int_time_from(1567931337)
```

1.13.69 sn0int_version

Get the current sn0int version string. This can be used with *semver_match* to test for certain features or behavior.

```
info(sn0int_version())
```

1.13.70 sock_connect

Create a tcp connection.

The following options are available:

`tls` Set to true to enable tls (certificates are validated)

`sni_value` Instead of the host argument, use a custom string for the sni extension.

`disable_tls_verify` **Danger**: disable tls verification. This disables all security on the connection. Note that sn0int is still rather strict, you're going to run into issues if you need support for insecure ciphers.

`proxy` Use a socks5 proxy in the format `127.0.0.1:9050`. This option only works if it doesn't conflict with the global proxy settings.

`connect_timeout` Abort tcp connection attempts after `n` seconds.

read_timeout Abort read attempts after n seconds. This can be used to wake up connections periodically.

write_timeout Abort write attempts after n seconds.

```
sock = sock_connect("127.0.0.1", 1337, {
    tls=true,
})
```

1.13.71 sock_upgrade_tls

Take an existing tcp connection and start a tls handshake. The options are the same as *sock_connect* but the tls value is always assumed to be true.

The sni value needs to be set specifically, otherwise the sni extension is disabled.

Using this function specifically returns some extra information that is discarded when using *sock_connect* directly with tls=true.

```
sock = sock_connect("127.0.0.1", 1337, {})
if last_err() then return end

tls = sock_upgrade_tls(sock, {
    sni_value='example.com',
})
if last_err() then return end

info(tls)
```

1.13.72 sock_options

Update options of an existing connection:

read_timeout Abort read attempts after n seconds. This can be used to wake up connections periodically.

write_timeout Abort write attempts after n seconds.

```
sock_options(sock, {
    read_timeout=3,
})
```

1.13.73 sock_send

Send data to the socket.

```
sock_send(sock, "hello world")
```

1.13.74 sock_recv

Receive up to 4096 bytes from the socket.

```
x = sock_recv(sock)
```

1.13.75 sock_sendline

Send a string to the socket. A newline is automatically appended to the string.

```
sock_sendline(sock, line)
```

1.13.76 sock_recvline

Receive a line from the socket. The line includes the newline.

```
x = sock_recvline(sock)
```

1.13.77 sock_recvall

Receive all data from the socket until EOF.

```
x = sock_recvall(sock)
```

1.13.78 sock_recvline_contains

Receive lines from the server until a line contains the needle, then return this line.

```
x = sock_recvline_contains(sock, needle)
```

1.13.79 sock_recvline_regex

Receive lines from the server until a line matches the regex, then return this line.

```
x = sock_recvline_regex(sock, "^250 ")
```

1.13.80 sock_recvn

Receive exactly n bytes from the socket.

```
x = sock_recvn(sock, 4)
```

1.13.81 sock_recvuntil

Receive until the needle is found, then return all data including the needle.

```
x = sock_recvuntil(sock, needle)
```

1.13.82 sock_sendafter

Receive until the needle is found, then write data to the socket.

```
sock_sendafter(sock, needle, data)
```

1.13.83 sock_newline

Overwrite the default `\n` newline.

```
sock_newline(sock, "\r\n")
```

1.13.84 sodium_secretbox_open

Use authenticated symetric crypto to decrypt a given message.

Internally this is `crypto_secretbox_xsalsa20poly1305`.

The key **must** be 32 bytes, see *key_trunc_pad* if necessary.

The first 24 bytes of the encrypted message are expected to be the nonce.

```
plain = sodium_secretbox_open(encrypted, key)
if last_err() then return end

txt = utf8_decode(plain)
if last_err() then return end

info(txt)
```

1.13.85 status

Update the label of the progress indicator.

```
status('ohai')
```

1.13.86 stdin_readline

Read a line from stdin. The final newline is not removed.

```
stdin_readline()
```

Note: This only works with *sn0int run –stdin*.

1.13.87 stdin_read_to_end

Read stdin until EOF as a utf-8 string.

```
stdin_read_to_end()
```

Note: This only works with *sn0int run –stdin*.

1.13.88 str_find

Returns the byte index of the first character that matches the pattern. This is explicitly a literal match instead of a lua pattern.

If no match is found, returns `nil`.

```
x = str_find('asdf', 'sd')
print(x == 2)
```

1.13.89 str_replace

Replaces all matches of a pattern in a string. This is explicitly a literal match instead of a lua pattern.

If no match is found, an unmodified copy is returned.

```
x = str_replace('this is old', 'old', 'new')
print(x == 'this is new')
```

1.13.90 strftime

Format a timestamp generated with *time_unix* into a date, see strftime rules.

```
t = strftime('%d/%m/%Y %H:%M', 1558584994)
```

1.13.91 strptime

Parse a date into a unix timestamp, see strftime rules.

```
t = strptime('%d/%m/%Y %H:%M', '23/05/2019 04:16')
```

1.13.92 strval

Convert a number into a string.

> x = strval(1234)

1.13.93 time_unix

Get the current time as seconds since `January 1, 1970 0:00:00 UTC`, also known as UNIX timestamp. This timestamp can be formated using *strftime*.

```
now = time_unix()
```

1.13.94 url_decode

Parse a query string into a map. For raw percent decoding see *url_unescape*.

```
v = url_decode('a=b&c=d')
print(v['a'] == 'b')
print(v['c'] == 'd')
```

1.13.95 url_encode

Encode a map into a query string. For raw percent encoding see *url_escape*.

```
v = url_encode({
    a='b',
    c='d',
})
print(v == 'a=b&c=d')
```

1.13.96 url_escape

Apply url escaping to a string.

```
v = url_escape('foo bar?')
print(v == 'foo%20bar%3F')
```

1.13.97 url_join

Join a relative link to an absolute link. If both links are absolute we just return the first one:

```
x = url_join('https://example.com/x', '/foo')
print(x == 'https://example.com/foo')

x = url_join('https://example.com/x', 'https://github.com/')
print(x == 'https://github.com/')
```

1.13.98 url_parse

Parse a url into its components. The following components are returned:

- scheme
- host
- port
- path
- query
- fragment

- params

```
url = url_parse('https://example.com')
print(url['scheme'] == 'https')
print(url['host'] == 'example.com')
print(url['path'] == '/')
```

1.13.99 url_unescape

Remove url escaping of a string.

```
v = url_unescape('foo%20bar%3F')
print(v == 'foo bar?')
```

1.13.100 utf8_decode

Decodes a list of bytes/numbers into a string. This function might fail.

```
x = utf8_decode({65, 65, 65, 65})
if last_err() then return end
print(x == 'AAAA')
```

1.13.101 warn

Log a warning to the terminal.

```
warn('ohai')
```

1.13.102 warn_once

Log a warning to the terminal once. This can be used to print a warning to the user without printing the same warning for each struct we're processing during a `run` execution.

```
warn_once('ohai')
warn_once('ohai')
```

1.13.103 ws_connect

Create a websocket connection. The url format is `ws://example.com/asdf`, `wss://` is also supported.

The following options are available:

`headers` A map of additional headers that should be set for the request.

`proxy` Use a socks5 proxy in the format `127.0.0.1:9050`. This option only works if it doesn't conflict with the global proxy settings.

`connect_timeout` Abort tcp connection attempts after `n` seconds.

`read_timeout` Abort read attempts after `n` seconds. This can be used to wake up connections periodically.

`write_timeout` Abort write attempts after `n` seconds.

```
sock = ws_connect("wss://example.com/asdf", {})
```

1.13.104 ws_options

Update options of an existing connection:

`read_timeout` Abort read attempts after `n` seconds. This can be used to wake up connections periodically.

`write_timeout` Abort write attempts after `n` seconds.

```
ws_options(sock, {
    read_timeout=3,
})
```

1.13.105 ws_recv_text

Wait until the server sends a text frame. A binary frame is considered an error. Ping requests are answered automatically.

```
msg = ws_recv_text(sock)
```

1.13.106 ws_recv_binary

Wait until the server sends a binary frame. A text frame is considered an error. Ping requests are answered automatically.

```
msg = ws_recv_binary(sock)
```

1.13.107 ws_recv_json

Identical to *ws_send_text* but automatically runs *json_decode* on the response.

```
msg = ws_recv_json(sock)
```

1.13.108 ws_send_text

Send a text frame on the websocket connection.

```
ws_send_text(sock, "ohai!")
```

1.13.109 ws_send_binary

Send a binary frame on the websocket connection.

```
ws_send_binary(sock, "\x00\x01\x02")
```

1.13.110 ws_send_json

Encode the object as json string and send it as a text frame on the websocket connection.

```
ws_send_text(sock, {
    foo="ohai!",
    x={
        y={1,3,3,7},
    },
})
```

1.13.111 x509_parse_pem

Parse a pem encoded certificate. This function might fail.

```
x = x509_parse_pem([[-----BEGIN CERTIFICATE-----
MIID9DCCA3qgAwIBAgIQBWzetBRl/ycHFsBukRYuGTAKBggqhkjOPQQDAjBMMQsw
CQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMSYwJAYDVQQDEx1EaWdp
Q2VydCBFQ0MgU2VjdXJlIFNlcnZlciBDQTAeFw0xODAzMzAwMDAwMDBaFw0yMDAz
MjUxMjAwMDBaMGwxCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJDQTEWMBQGA1UEBxMN
U2FuIEZyYW5jaXNjbzEZMBcGA1UEChMQQ2xvdWRmbGFyZSwgSW5jLjEdMBsGA1UE
AwwUKi5jbG91ZGZsYXJlLWRucy5jb20wWTATBgcqhkjOPQIBBggqhkjOPQMBBwNC
AASyRQsxrFBjziHmfDQjGsXBU0WWl3oxh7vg6h2V9f8lBMp18PY/td9R6VvJPa20
AwVzIJI+dL6OSxviaIZEbmK7o4ICHDCCAhgwHwYDVR0jBBgwFoAUo53mH/naOU/A
buiRy5Wl2jHiCp8wHQYDVR0OBBYEFN+XTeVDs7BBp0LykM+Jf64SV4ThMGMGA1Ud
EQRcMFqCFCouY2xvdWRmbGFyZS1kbnMuY29thwQBAQEBhwQBAAABghJjbG91ZGZs
YXJlLWRucy5jb22HECYGRwBHAAAAAAAAAAAAAERGHECYGRwBHAAAAAAAAAAAAAEAEw
DgYDVR0PAQH/BAQDAgeAMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjBp
BgNVHR8EYjBgMC6gLKAqhihodHRwOi8vY3JsMy5kaWdpY2VydC5jb20vc3NjYS1l
Y2MtZzEuY3JsMC6gLKAqhihodHRwOi8vY3JsNC5kaWdpY2VydC5jb20vc3NjYS1l
Y2MtZzEuY3JsMEwGA1UdIARFMEMwNwYJYIZIAYb9bAEBMCowKAYIKwYBBQUHAgEW
HGh0dHBzOi8vd3d3LmRpZ2ljZXJ0LmNvbS9DUFMwCAYGZ4EMAQICMHsGCCsGAQUF
BwEBBG8wbTAkBggrBgEFBQcwAYYYaHR0cDovL29jc3AuZGlnaWNlcnQuY29tMEUG
CCsGAQUFBzAChjlodHRwOi8vY2FjZXJ0cy5kaWdpY2VydC5jb20vRGlnaUNlcnRF
Q0NTZWN1cmVTZXJ2ZXJDQS5jcnQwDAYDVR0TAQH/BAIwADAKBggqhkjOPQQDAgNo
ADBlAjEAjoyy2Ogh1i1/Kh9+psMc1OChlQIvQF6AkojZS8yliar6m8q5nqC3qe0h
HR0fExwLAjAueWRnHX4QJ9loqMhsPk3NB0Cs0mStsNDNG6/DpCYw7XmjoG3y1LS7
ZkZZmqNn2Q8=
-----END CERTIFICATE-----
]])
if last_err() then return end
print(x)
```

1.13.112 xml_decode

Decode a lua value from an xml document.

```
x = xml_decode('<body><foo fizz="buzz">bar</foo></body>')
if last_err() then return end

body = x['children'][1]
foo = body['children'][1]

print(foo['attrs']['fizz'])
print(foo['text'])
```

1.13.113 xml_named

Get a named child element from a parent element.

```
x = xml_decode('<body><foo fizz="buzz">bar</foo></body>')
if last_err() then return end

body = x['children'][1]
foo = xml_named(body, 'foo')
if foo ~= nil then
    print(foo)
end
```