User Guide

# Security Information Exchange (SIE)

July 2024

DomainTools

# In this Document

This guide gives you a functional overview of the Security Information Exchange (SIE), provides install and access instructions, and links to detailed user guides, reference, and tutorials for components of the SIE ecosystem.

# Overview

SIE gives you real-time access to data from our global sensor network. That data includes over half a million passive DNS (pDNS) observations per second, as well as other key security data points. DomainTools processes this data into usable formats, stream it over real-time channels, and provides you with tools for your use case.

**Channels** are available through the following mechanisms, which are explained in more detail below:

- **SIE Batch** is a web interface and REST API with access to the last 12-18 hours of data from your subscribed feeds.
- **SIE Remote Access** creates a tunnel from SIE to the analyst's system, and supports a REST API (see AXAMD, below).
- **SIE Direct Connect** is a leased blade server with pre-installed SIE tools that provides direct access to the SIE network.

**Data formats** are detailed in the File and Data Formats section. Feeds are typically available in NMSG and JSON formats, depending on the access method (some JSON availability is through conversion with `nmsgtool`).

- NMSG is a streaming binary format, based on Google Protocol Buffers, that handles high volume, real-time traffic. SIE delivers most channels with NMSG. The included nmsgtool (detailed below) can convert NMSG to JSON.
- JSON, typically in JSONL/NDJSON format, is provided directly by certain feeds.
- Packet Capture (libpcap) is used for channels that preserve packet structure.

The following table provides the channel, name, description, bitrate, payload rate, and notes on each channel. All channels are available in NMSG and JSON formats unless otherwise noted.

# Channels Overview

| Number | Name | Description | Bitrate | Payloads | Notes |
|---|---|---|---|---|---|
| 24 | **Spam-Full** | Spam-Full shares full copies of spam emails sent to email spamtrap systems. The honeypots have been configured to collect and store all email messages for analysis and they use email addresses that have never been used to receive email, or are no longer in use and should not be receiving email. | 2.4kbps (104kbps) | 2/sec (14/sec) | |
| 25 | **Spam-Select** | Spam-Select shares key fields from spam emails sent to email honeypot/spamtrap systems. The spamtraps have been configured to collect and store all email messages for analysis and they use email addresses that have never been used to receive email, or are no longer in use and should not be receiving email. | 2kbps (105kbps) | 1/sec (3/sec) | |
| 27 | **Phishing URLs** | Includes URLs, the brand target, and other information related to phishing campaigns. | <1kbps (2kbps) | <1/sec (2/sec) | |
| 42 | **IDS and Firewall Log Data** | Information about network traffic that is blocked by Intrusion Detection Systems (IDS) and firewall devices. The data is anonymized and batched every five minutes. | 7mbps (35mbps) | 390/sec (2kbps) | Not available over AXAMD. |
| 80 | **Conficker Sinkhole** | HTTP connection information from Conficker-infected clients to ‘sinkholed’ command and control servers. | 385kbps (520kbps) | 210/sec (280/sec) | Not available over AXAMD. |
| 115 | **DDos Events** | Evidence of DDoS and DRDoS (Distributed Reflection Denial of Service) attacks based on analysis of data based on analysis of data from captured network packets destined from unused network space. | <1kbps (<1kbps) | <1/sec (1.5/sec) | |
| 207 | **DNSDB De-Duplicated Data** | Passive DNS observations after the deduplication processing phase and immediately prior to the verification phase. | 144mbps (170mbps) | | Not available over Remote Access or AXAMD; Batch |

| Number | Name | Description | Bitrate | Payloads | Notes |
|---|---|---|---|---|---|
| | | | | | files available as NMSG. |
| 208 | **DNSDB Verified Data** | Deduplicated passive DNS data with low value entries removed, and verified for balliwick-appropriate data (misleading resource record information is removed). | 88mbps (117mbps) | | Not available over Remote Access or AXAMD; Batch files available as NMSG. |
| 211 | **Newly Active Domains (NAD)** | Previously observed domains (via Channel 204) becoming active after 10 or more days of inactivity. | 62kbps (900kbps) | 53/sec (760/sec) | |
| 212 | **Newly Observed Domains (NOD)** | Domains not previously observed by SIE. It can also be accessed by: NOD RPZ \| NOD rbldnsd \| NOD FAQ | 3kbps (18kbps) | 2/sec (13/sec) | |
| 213 | **Newly Observed Hostnames (NOH)** | Fully Qualified Domain Names (FQDNs) not previously seen when compared to the DNSDB historical database. | Under review | TBA | |
| 214 | **DNS Changes** | Domains, hostnames, or record data that is unknown to DNSDB, either because the data is for a new domain or hostname or because the record data for a domain or hostname has changed. These changes may include new RR types, new or changed IP addresses, or a change in the authoritative name servers for a domain.<br><br>**More information**: The DNS Changes Channel | 4.9mbps (8mbps) | TBA | Not available over AXAMD. |
| 220 | **DNS Errors** | Non-deduplicated DNS responses that returned a non-zero Response Code (RCODE). This includes NXDomain, ServFail, Refused, FormErr, NotImp, and other RCODEs. | 570mbps (860mbps) | TBA | Only available over Direct Connect . |
| 221 | **NX Domains** | Derived from Channel 220 DNS Errors, but only includes NXDomain RCODEs.<br>More information: Introducing NXD | 52mbps(73mbps) | TBA | Not available through AXAMD. |

| Number | Name | Description | Bitrate | Payloads | Notes |
|---|---|---|---|---|---|
| 255 | **Heartbeat (Testing)** | For monitoring purposes. | 1kbps (1kbps) | TBA | |

# Getting Started

## Access

Access to SIE is provisioned by DomainTools Enterprise Support at enterprisesupport@domaintools.com. Batch and SIE Remote Access is provisioned with an API key. SIE Direct Connect requires the customer's public key and originating IP address(es) for SSH access.

## System Requirements

A server configured by DomainTools for SIE Direct Connect has the following operating system (OS) and hardware specifications.

| Component | Specification |
|---|---|
| Compute | One (1) x Intel Quad Core Xeon E3 3.40Ghz |
| Memory | 16GB RAM |
| Storage | Two (2) x 2TB 7200RPM drives (configured for RAID 1, 2TB available for customer use) |
| Network (Internet connection) | 100Mbps |
| Network (SIE connection) | 10gigabit |

These hardware specifications are adequate for acquiring, processing, compressing, and buffering data from high throughput SIE channels. Intensive processing and analysis on data from SIE channels with the highest data volumes may require additional memory (RAM).

## Installation

### Debian GNU/Linux 11 "Bullseye"

Debian GNU/Linux is supported through an apt package repository. Ubuntu and other Debian-based systems may be able to use this repository as well.

Enable the `bullseye-farsightsec` package repository by copying the GPG key in place and adding a sources.list entry:

```
$ wget -O debian-farsightsec.gpg
https://dl.farsightsecurity.com/debian/debian-farsightsec.gpg

$ echo deb [arch=amd64] http://dl.farsightsecurity.com/debian
bullseye-farsightsec main | sudo tee -a
/etc/apt/sources.list.d/debian-farsightsec.list

$ sudo cp debian-farsightsec.gpg /etc/apt/trusted.gpg.d/

$ sudo apt-get update
```

### Package Signing Key

Our open source Debian packages are hosted at https://dl.farsightsecurity.com/debian/. The GPG keyid is A511AE06 (Download). Its GPG key details are:

```
pub rsa4096/A511AE06 2020-06-18 [SC] [expires: 2025-05-20]
    E26E47876B777A4CB285338CE4C4F714A511AE06

uid [ultimate] Farsight Security, Inc. Debian Package Signing Key
```

The previous package signing key was 779812D5.

For "apt-get update" use, this key file can be copied to the `/etc/apt/sources.list.d/` directory or imported using `apt-key add`.

### Package Installation

Install the core nmsg packages and SIE plugins:

```
$ sudo apt-get install nmsgtool nmsg-msg-module-sie
```

Install packages for developing C application:

```
$ sudo apt-get install libnmsg-dev nmsg-msg-module-sie-dev libwdns-dev
```

Install packages for developing Python applications:

```
$ sudo apt-get install python-nmsg python-wdns
```

Install packages for SIE remote access (sratool, sratunnel):

```
$ sudo apt-get install axa-tools
```

## From Source on Rocky 9 (RHEL-compatible)

Rocky Linux is compatible with Red Hat Enterprise Linux (RHEL).

This installation was tested on Rocky Linux 9.2 with default install options for the Server with GUI. These instructions should work for all RHEL-compatible distributions which have access to the `EPEL` and `CRB` repositories or their equivalents. It results in these changes:

- A local user was created with `sudo` rights
- VM name was set to `rocky9.local`
- A password was set for `root` to enable `su`

All updates and patches were applied:

```
$ sudo dnf update
```

### Setup, Dependencies, & Configuration

The following dependencies were installed. Note that both the “Extra Packages for Enterprise Linux: (a.k.a. `EPEL`) and "Code Ready Builder" (a.k.a `crb`) repositories containing extra libraries and developer tools are required to compile required packages.

```
$ sudo dnf install epel-release
$ sudo /usr/bin/crb enable   # OR sudo dnf config-manager
--set-enabled crb
$ sudo dnf group install "Development Tools"
$ sudo dnf install protobuf-c-devel libedit-devel yajl-devel
json-c-devel
$ sudo dnf install libpcap-devel zeromq-devel libbsd-devel lmdb-devel
```

Note: you should not need the following, as the group install should cover it:

```
$ sudo dnf install autoconf automake libtool pkg-config git zlib
```

Ensure paths are set correctly:

```
$ su
# echo "/usr/local/lib" > /etc/ld.so.conf.d/local.conf
# exit
$ export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

Create a space to work in:

```
$ mkdir ~/fsi
```

You are now ready to install axa and its dependencies.

### Install `wdns`

Clone the repository and install:

```
$ cd ~/fsi/
$ git clone https://github.com/farsightsec/wdns.git
$ cd wdns
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

Validate the install was successful. You should now see at least the following files. Note that other files could also be in this directory.

```
$ ls /usr/local/lib
libwdns.a  libwdns.la  libwdns.so  libwdns.so.1  libwdns.so.1.3.1
pkgconfig
```

Ensure the libraries are available from `/usr/local/lib`:

```
$ su
# ldconfig -v | grep libwdns
     libwdns.so.1 -> libwdns.so.1.3.1
```

```
# exit
```

### Install `nmsg`

Clone the repository and install:

```
$ cd ~/fsi/
$ git clone https://github.com/farsightsec/nmsg.git
$ cd nmsg
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

Validate the install was successful. You should now see at least the following files. Note that other files could also be in these directories:

```
$ ls /usr/local/lib
libnmsg.a   libnmsg.so   libnmsg.so.8.1.1   libwdns.la   libwdns.so.1
nmsg  libnmsg.la   libnmsg.so.8  libwdns.a   libwdns.so
libwdns.so.1.3.1   pkgconfig
$ ls /usr/local/lib/nmsg
nmsg_flt1_sample.la  nmsg_flt1_sample.so  nmsg_msg9_base.la
nmsg_msg9_base.so
```

Ensure the libraries are available from `/usr/local/lib`:

```
$ su
# ldconfig -v | grep libnmsg
     libnmsg.so.8 -> libnmsg.so.8.1.1
# exit
```

### Install `sie-nmsg`

Clone the repository and install:

```
$ cd ~/fsi/
$ git clone https://github.com/farsightsec/sie-nmsg.git
$ cd sie-nmsg
```

```
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

Validate the install was successful. You should now see at least the following files. Note that other files could also be in these dirs.

```
$ ls /usr/local/lib
libnmsg.a   libnmsg.so   libnmsg.so.8.1.1  libwdns.la  libwdns.so.1
nmsg   libnmsg.la   libnmsg.so.8  libwdns.a   libwdns.so
libwdns.so.1.3.1   pkgconfig
$ ls /usr/local/lib/nmsg
nmsg_flt1_sample.la   nmsg_msg9_base.la  nmsg_msg9_sie.a
nmsg_msg9_sie.so   nmsg_flt1_sample.so  nmsg_msg9_base.so
nmsg_msg9_sie.la
```

### Install `axa` client

Clone the repository and install:

```
$ cd ~/fsi/
$ git clone https://github.com/farsightsec/axa.git
$ cd axa
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

Validate the install was successful. You should now see at least the following files. Note that other files could also be in this directory.

```
$ ls /usr/local/bin
axa_link_certs  axa_server_cert  nmsgtool  sratunnel axa_make_cert
sratool
```

Ensure the libraries are available from `/usr/local/lib`:

```
$ su
```

```
# ldconfig -v | grep libaxa
     libaxa.so.3 -> libaxa.so.3.0.0
# exit
```

Validate the tool can successfully execute. Note that running `sratool` like this only proves all libraries and files are installed correctly; it is still necessary to configure access to stream data successfully.

```
$ sratool -V
sratool built using AXA library 3.0.1, supporting AXA protocols v1 to
v2;
currently using v2
client HELLO: {"hostname":"rocky9.local","uname_sysname":"Linux",
"uname_release":"5.14.0-284.30.1.el9_2.x86_64",
"uname_version":"#1 SMP PREEMPT_DYNAMIC Sat Sep 16 09:55:41 UTC 2023",
"uname_machine":"x86_64","origin":"sratool","libaxa":"3.0.1","libnmsg"
:"1.1.2",
"libwdns":"0.12.0","libyajl":20100,"OpenSSL":"OpenSSL 3.0.7 1 Nov
2022", "AXA protocol":2}
```

## FreeBSD

### Binary Package Installation

Install the core nmsg packages and SIE plugins:

```
pkg install nmsg sie-nmsg
```

Optionally: Install packages for developing Python applications:

```
pkg install py-pynmsg py-pywdns
```

Optionally: Install packages for SIE remote access (sratool, sratunnel):

```
pkg install axa
```

### Installation from Source using FreeBSD Ports

If you do not require the Doxygen generated API documentation, you may wish to disable the DOXYGEN option when building these ports to avoid building Doxygen and its many dependencies.

```
(cd /usr/ports/net/nmsg && make install clean)

(cd /usr/ports/net/sie-nmsg && make install clean)

(cd /usr/ports/net/py-pynmsg && make install clean)

(cd /usr/ports/dns/py-pywdns && make install clean)

(cd /usr/ports/net/axa && make install clean)
```

### Uninstallation

```
pkg delete wdns nmsg sie-nmsg py-pynmsg py-pywdns axa
```

## MacOS 14 Sonoma

These instructions are for installing AXA Tools (to enable SIE Remote Access) on Mac computers running MacOS 14.x Sonoma. See additional notes at the end for older Intel-based Macs.

### Setup, Dependencies, and Configuration

The following tools and dependencies were installed. This configuration depends on the third-party brew package manager to install all required dependent libraries. Install `Xcode` command line tools. This shell command opens up a new UX window to confirm the install of the `XCode` tools. If you have the full version of `XCode` installed, you can skip this step.

```
$ xcode-select --install
```

Install Homebrew from brew.sh:

```
$ /bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Homebrew may require you to make modifications to your shell's profile. Make these changes before proceeding to ensure brew is set up and configured correctly.

Use Homebrew to install all required tools and libraries for AXA and its dependencies:

```
$ brew install wget openssl@1.1
$ brew install autoconf automake libtool pkgconfig
$ brew install protobuf protobuf-c libpcap yajl zmq lmdb json-c
```

Create a space to work in:

```
$ mkdir ~/fsi
```

### OpenSSL Configuration

OpenSSL requires some extra configuration to ensure we use the brew installed version instead of the version pre-installed by macOS. Alter your `~/.profile` or `~/.zprofile` file with the following or set these exports directly in the terminal:

```
export LDFLAGS="-L/opt/homebrew/opt/openssl@1.1/lib"
export CPPFLAGS="-I/opt/homebrew/opt/openssl@1.1/include"
export PKG_CONFIG_PATH="/opt/homebrew/opt/openssl@1.1/lib/pkgconfig"
```

After setting these OpenSSL config changes, you may need to source your profile changes or restart your shell.

You are now ready to install axa and its dependencies.

### Install `wdns`

Clone the repository and run the install:

```
$ cd ~/fsi
$ git clone https://github.com/farsightsec/wdns.git
$ cd wdns
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

Validate the install was successful. You should now see at least the following files. Note that other files could also be in this dir.

```
$ ls /usr/local/lib
libwdns.1.dylib  libwdns.a  libwdns.dylib  libwdns.la  pkgconfig
```

### Install `nmsg`

Clone the repository and run the install. Note the change to `configure`:

```
$ cd ~/fsi
$ git clone https://github.com/farsightsec/nmsg.git
$ cd nmsg
$ ./autogen.sh
$ ./configure --with-libpcap=/opt/homebrew/opt/libpcap
$ make
$ sudo make install
```

Validate the install was successful. You should now see at least the following files. Note that other files could also be in these directories:

```
$ ls /usr/local/lib
libnmsg.8.dylib  libnmsg.dylib    libwdns.1.dylib  libwdns.dylib
nmsg   libnmsg.a   libnmsg.la   libwdns.a   libwdns.la   pkgconfig
$ ls /usr/local/lib/nmsg
nmsg_flt1_sample.la  nmsg_flt1_sample.so  nmsg_msg9_base.la
nmsg_msg9_base.so
```

### Install `sie-nmsg`

Clone the repository and run the install.

```
$ cd ~/fsi
$ git clone https://github.com/farsightsec/sie-nmsg.git
$ cd sie-nmsg
$ ./autogen.sh
$ ./configure
$ make
```

```
$ sudo make install
```

Validate the install was successful. You should now see at least the following files. Note that other files could also be in these dirs.

```
$ ls /usr/local/lib
libnmsg.8.dylib  libnmsg.dylib    libwdns.1.dylib  libwdns.dylib
nmsg   libnmsg.a   libnmsg.la   libwdns.a   libwdns.la   pkgconfig
$ ls /usr/local/lib/nmsg
nmsg_flt1_sample.la   nmsg_msg9_base.la   nmsg_msg9_sie.a
nmsg_msg9_sie.so   nmsg_flt1_sample.so   nmsg_msg9_base.so
nmsg_msg9_sie.la
```

### Install `axa` client

Clone the repository and run the install. Note the change to configure:

```
$ cd ~/fsi
$ git clone https://github.com/farsightsec/axa.git
$ cd axa
$ ./autogen.sh
$ ./configure --with-libpcap=/opt/homebrew/opt/libpcap
$ make
$ sudo make install
```

Validate the install was successful. You should now see at least the following files. Note that other files could also be in this directory.

```
$ ls /usr/local/bin
axa_link_certs  axa_server_cert  nmsgtool  sratunnel  axa_make_cert
sratool
```

Validate the tool can successfully execute. Note that running `sratool` like this only proves all libraries & files are installed correctly, you will still need to set up access to stream data successfully.

```
$ sratool -V
sratool built using AXA library 3.0.2, supporting AXA protocols v1 to
v2; currently using v2
```

```
client HELLO:
{"hostname":"My-M1-Mac.local","uname_sysname":"Darwin","uname_release"
:"23.2.0",
"uname_version":"Darwin Kernel Version 23.2.0: Wed Nov 15 21:53:34 PST
2023;
root:xnu-10002.61.3~2/RELEASE_ARM64_T8103","uname_machine":"arm64","or
igin":"sratool",
"libaxa":"3.0.2","libnmsg":"1.1.2","libwdns":"0.12.0","libyajl":20100,
"OpenSSL":"OpenSSL 3.1.4 24 Oct 2023","AXA protocol":2}
```

### Intel-based Mac Differences

Apple Silicon (M1) Macs and older Intel-based Macs have some differences to be aware of since brew operates differently on the two platforms.

`brew` installs all installed packages to `/opt/homebrew` on Apple Silicon Macs, it installs into `/usr/local` for Intel-based Macs. All references to `/opt/homebrew` above will need to be changed.

Sudo is not required for `make install` on Intel Macs as brew changes `/usr/local` permissions. Sudo is required for Apple Silicon Macs.

Since `brew` operates inside `/usr/local` on Intel-based Macs and is more permissive with directory permissions, other users of the Mac may have access to these tools by default

Intel-based Mac `sratool` output for V3.x:

```
$ sratool -V  # on an Intel Mac
sratool built using AXA library 3.0.1, supporting AXA protocols v1 to
v2; currently using v2
client HELLO:
{"hostname":"My-Mac.local","uname_sysname":"Darwin","uname_release":"2
3.1.0",
"uname_version":"Darwin Kernel Version 23.1.0: Mon Oct  9 21:27:27 PDT
2023;
root:xnu-10002.41.9~6/RELEASE_X86_64","uname_machine":"x86_64","origin
":"sratool",
"libaxa":"3.0.1","libnmsg":"1.1.2","libwdns":"0.12.0","libyajl":20100,
"OpenSSL":"OpenSSL 1.1.1w  11 Sep 2023","AXA protocol":2}
```

# Access Methods

This section outlines SIE access methods and links to detailed documentation for each. Information on the file and data formats is available in the Data Formats section.

## SIE Batch Web Interface

**Connect**: https://batch.sie-remote.net/
**More information**: SIE Batch User Guide | What's SIE Batch

The Batch web interface is a frontend for the REST API that provides full access to the last 12-18 hours of batched SIE data. This batch method is designed for periodic updates that can (with short delay periods) approach real-time. For real-time feeds, consult Remote Access and Direct Connect options, below.

Log in with your API key, which you obtain from enterprisesupport@domaintools.com. The interface will present each subscribed channel, the channel's data format, and default download windows.

To use the web interface, select the channel and date range. The SIE web interface will confirm the channel, display its average traffic, and set the default date range.

To download current data, set the end date-time to your current time, minus ~10 seconds. SIE Batch does not de-duplicate overlapping downloads.

## SIE Batch REST API

**Connect**: https://batch.sie-remote.net/siebatchd/v1/

**More information**: SIE Batch API Reference | SIE Batch OpenAPI Specification at SwaggerHub | SIE Batch API: A libcurl Example in C | Batch REST API White Paper

The Batch REST API is the backend to the Batch Web Interface. Like the web interface, it provides full access to the last 12-18 hours of batched SIE data. This batch method is designed for periodic downloads, rather than as a source of real-time data. For real-time feeds, see Remote Access and Direct Connect options, below.

### Validating an API Key

Obtain your API key from enterprisesupport@domaintools.com. Use the key to issue a POST command. For example, with `curl`:

```
APIKEY=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
curl -d '{"apikey":"'$APIKEY'"}'
https://batch.sie-remote.net/siebatchd/v1/validate
```

In response, SIE will issue a response similar to:

```
HTTP/1.1 200 OK
{
  "profile": {
    "username": "siebatch-customer",
    "siebatch": {
      "ch212": {
        "description": "Newly Observed Domains"
      },
      "ch213": {
        "description": "Newly Observed Fully Qualified Domain Names"
      },
    }
  },
  "_status": "OK",
  "_message": ""
}
```

# SIE Remote Access (AXA Toolkit)

**Connect**: Depending on the channel format, you may need to perform further processing after implementing Remote Access. See the Data Formats section below.

**More information**: Remote Access Toolkit GitHub repository + manpages | AXA User Guide | AXA Manual | AXA Migration Overview

SIE Remote Access (SRA) provides real-time streams of all but the highest bandwidth SIE channels. It uses a suite of tools referred to collectively as the AXA (Advanced Exchange Access) toolkit, which implement the Remote Access transport layer (also known as the AXA Transport Layer). In order to reduce bandwidth, SRA provides subscribers with the ability to invoke a server-side filtering capability across a set of channels, selecting only that subset of records that match specific domain name / IP address search criteria.

Use Remote Access (AXA) tools `sratool` and `sratunnel` to select and define search or filtering criteria for your SIE channels, control rate limiting, and receive accounting messages.

## SIE Remote Access Tool (`sratool`)

`sratool` is used to test, debug, inspect, or stream SIE Remote Access connections. In its most common invocation, it connects to a Remote Access server, issues AXA protocol messages, and displays the responses. `sratool` can be automated, but is typically used for interactive user-supplied commands.

**More information**: Remote Access Toolkit GitHub repository + manpages | AXA User Guide | AXA Manual | AXA Migration Overview

## SIE Remote Access Tunnel (`sratunnel`)

`sratunnel` is a production command-line tool that streams SIE data to the local network. It automates Remote Access server connections and updates.

**More information**: Remote Access Toolkit GitHub repository + manpages | AXA User Guide | AXA Manual | AXA Migration Overview

## Core Features

### Watches

There are four kinds of watches you can set with Remote Access:

- **IP Watches**: used to express interest in SIE messages containing a specified IP address or CIDR block. AXA understands both IPv4 and IPv6 address types.
- **DNS Watches**: used to express interest in SIE messages containing a specified hostname, domain, or wildcard.

- **Channel Watches**: used to express interest in SIE messages from an entire channel. This is useful to enable “the firehose” for a given channel and ask SRA to send everything from the specified channel rather than matching IP address information or DNS names.
- **Error watches**: used to express interest in SIE messages that cannot be decoded by the server.

### Rate Limiting

Rate limiting is used to limit the rate of incoming AXA messages as emitted from the server to the client.

### Accounting

**More information**: Understanding Accounting

Accounting tracks traffic totals. Server-side, SIE maintains a series of per-client packet counters, and emits periodic accounting messages. The command is available from sratool and sratunnel via the -A command line option.

## SIE Remote Access REST API (AXAMD)

**More Information**: axamd Client GitHub repository | Using AXAMD with Newly Observed Domains

The AXA Middleware Daemon (AXAMD) provides a REST API interface for the Remote Access service. A Python module, `axamd_client`, is also available in the axamd GitHub repository.

## SIE Direct Connect

**More information**: Introduction to nmsgtool

Direct Connect provides the fastest connection to the SIE network, with a dedicated, co-located blade server. DomainTools provides you with a pre-configured server (with root access). Contact enterprisesupport@domaintools.com to discuss your options.

With Direct Connect, you use `nmsgtool` to access and process SIE data on your blade server before transferring it to your network.

# Data Formats

SIE data can be available in NMSG, JSON (and NDJSON), and packet capture (pcaplib) formats.

The NMSG format, introduced next, is unique to DomainTools and includes libraries and reference implementations for processing SIE data.

## NMSG (Network Message)

**More information**: NMSG GitHub repository (readme, manpages), | NMSG User Guide | Introduction to NMSG | NMSG C API| NMSG headers and encoding | NMSG C API Reference | NMSG Python SDK | NMSG loss tracking | Introduction to nmsgtool

NMSG is a file and transmission format that encapsulates typed, structured data into payloads which are packed into containers. The NMSG format relies on Google Protocol Buffers to encode the payload header. Each payload is associated with a specific message schema. nmsg can be extended at runtime with new message types, and can be converted to JSON with the included `nmsgtool`.

### NMSG Libraries and Tools

libnmsg is the reference implementation of this format and provides an extensible interface for creating and parsing messages in NMSG format. Individual NMSG payloads are distinguished by assigned vendor ID and message type values and `libnmsg` provides a modular interface for registering handlers for specific message types. `libnmsg` makes it easy to build new message types using a Protocol Buffers compiler.

#### C Library: libnmsg

NMSG is delivered to the application programmer as a C library called `libnmsg`. The library presents a rich API for the programmer to build NMSG-capable applications and configure, tune, and/or tweak its many options and features.

The reference implementation of `libnmsg` is the included `nmsgtool`, a thin wrapper around `libnmsg` that provides powerful NMSG functionality at the Unix command-line.

### Python Library: pynmsg

Also available is a Python extension module, pynmsg, that enables NMSG development using the Python programming language. See the pynmsg distribution and the pynmsg Python extension source.

## JSON and NDJSON/JSONL

To see which channels and methods are available in JSON or Newline Delimited JSON (also known as JSON Lines), review the Channels and Formats table (above).

`nmsgtool` can natively convert from NSMG to JSON; consult the NMSG section above.

## Packet Capture Format (PCAP)

A small number of channels arrive in packet capture format.