SOPHOS

sav interface

User manual

Sophos Simple Scanning Protocol

Version 1.2

Contents

1 SSSP

1.1 Introduction

SSSP is intended to be a simple protocol to enable scanning of files and data over a network. It includes requests:-

- to set options and control the scanning, and reporting, process
- to query the status and capabilities of the server
- to scan by file name and data

1.2 General Message Format

All transactions, other than datastreams for scanning, are in plain text.

Client requests are mostly single line terminated by a single end of line sequence. A request that is multi-line (e.g. OPTIONS) is terminated by a double end of line sequence, i.e. a blank line.

Responses from the server are in two steps: a response to indicate that the request is acceptable, or not, followed by the results of the request. The request results are multi-line and so terminated by a blank line.

Datastreams, where used, are not included in a message but follow after and are treated as raw data.

The server will use “\r\n” as its end of line sequence but will ignore carriage returns (“\r”) in received messages so that end of lines may be either “\n” or “\r\n”.

The %XX escape convention will be used where needed. See the appendix on %XX Encoding for a description. Datastreams will be treated as binary data and escaping will not be used.

1.3 Features

The client may send a file, or directory, name for scanning, or the data itself.

SAVI and Engine configuration options can be set using the OPTIONS request. Once set, configuration options stay set for the remainder of the session with the client unless explicitly changed by another OPTIONS request.

SAVI and Engine configuration options may be reset to the default specified for the server.

Server, SAVI and Engine configuration can be queried by the client.

SAVI responses may be in one of 2 formats: a simple brief format or XML if more details in a well-defined format are required.

Allowance has been made for new versions of the protocol.

1.4 Character Set

The keywords and information as described here are in 7-bit ASCII. The file paths and explanatory text accompanying the result codes in the DONE statement are locale dependent. See the appendix on Character Encodings.

1.5 Version Negotiation

On connection the server will answer OK followed by the protocol and the protocol version:-

OK SSSP/1.0

The client should respond with a request that starts with the protocol and version that it will use:-

SSSP/1.0 {optional request}

The client should step down to the server version if necessary and report that version; similarly the server on receiving the client version should step down to the client version.

1.6 Client Requests

All client requests consist of a line with an optional protocol and version specification, followed by the command and any required parameters. The initial line may be followed by lines of additional and optional parameters which have the form of a keyword followed by a colon and the value of the parameter.

N.B. A multi-line response is terminated by a blank line.

QUERY {SERVER|SAVI|ENGINE}

QUERY enables the client to discover the configuration of the server.

The SERVER option requests the configuration of the server itself. In particular it will inform the client of the requests, or methods, it may use. SCANFILE, for example, may not be available for all clients.

The SAVI option requests the SAVI configuration options.

The ENGINE option requests the engine and virus data version.

If no option is specified it defaults to SERVER.

The response to a QUERY is multi-line.

OPTIONS

The client may set options for the duration of the session with the server, including SAVI/Engine options. Once set they will be carried forward for the subsequent requests until explicitly changed by another OPTIONS request.

The OPTIONS request is multi-line and so is terminated by a blank line.

The response to an OPTIONS request is not multi-line but is treated as such for consistency and convenience.

SCANFILE <filename>

The client sends the full path name of a file it wishes to be scanned. The file must be readable by the server using the path name.

The response to a SCANFILE is multi-line.

SCANDIR <filename>

The client sends the full path name of a file or directory it wishes to be scanned. If a directory is specified it will scan that directory only and will not scan sub-directories. The directory, or file, must be readable by the server using the path name.

The response to a SCANDIR is multi-line.

SCANDIRR <filename>

The client sends the full path name of a file or directory it wishes to be scanned. If a directory is specified it will scan that directory and will recurse into any sub-directories. The directory, or file, must be readable by the server using the path name.

The response to a SCANDIRR is multi-line.

SCANDATA <data size>

The client sends the data to be scanned. The data follows on and is <data size> bytes long. Exactly <data size> bytes of data must be sent or the request will fail.

The response to a SCANDATA is multi-line.

BYE

Enables the client to end the session with the server in a well-behaved manner. The server will also respond with 'BYE'.

1.7 Client Options

The OPTIONS request is followed by a list of one or more optional parameters. These all have the format of <option name>, followed by a colon and the value of the parameter. Spaces may be included after the colon and before the value. The list of options is terminated by a blank line.

event: <event type> {<engine action>}

Specifies the actions the server should take when certain events happen during the scan process.

<event type> is one of:-

- ‘onerrorfound’
- ‘onvirusfound’

<engine action> is one of:-

- ‘continuethis’ Continue scanning this object
- ‘continuenext’ Stop scanning this object and go on to the next one
- ‘stopthis’ Stop scanning this file.
- ‘terminate’ Stop scanning and finish the request immediately.

Not specifying an <engine action> will reset the action to its default.

savidefault:

Causes the server to reset SAVI to the default server configuration. This is the configuration as specified for the server by the administrator and not the default SAVI configuration.

savigrp: <config name> <config value>

Specifies the name and value of an engine group configuration option. Only status values (U32) are permitted, ie the value should be 0 or 1 only.

savists: <config name> <config value>

Specifies the name and value of a SAVI or engine configuration option. Only status values (U32) are permitted, ie the value should be 0 or 1 only.

output: <style>

Specifies the style in which the client would like to see the results of requests. <style> may be one of:-

- ‘brief’
- ‘xml’

The last presents more detailed information in XML format. The default form is ‘brief’.

report: {<report types>}

Specifies the events that the server should report to the client. <report types> is a space separated list of:-

- ‘file’. The client is notified as the server starts to scan a file, or directory.
- ‘classification’ The client is notified of the classification of the object as determined by the engine.
- ‘archive’. The client is notified as the server starts to scan an object contained within a file, such as an archive file.
- ‘virus’. The client is notified when a virus is found. This is in addition to the normal end of scan report.
- ‘error’. The client is notified whenever the server encounters an error condition. Normally errors are collected and only the worst error is reported.

If all these reports are required ‘all’ will suffice.

If no <report types> are specified there will be no reports.

1.8 Server Responses

The server has a limited number of responses, two of which are for the initial connection and when the client leaves. The remaining two are in reply to a request and indicate the acceptability of the request.

OK <protocol>

Initial response to a connection giving the server's protocol and protocol version.

ACC <request id>

The server has recognised and accepted the request. The results of the request will follow the ACC. <request id> is a unique identifier for this request. The results of the request will follow.

REJ <reject code>

The server has rejected the request. The <reject code> gives an indication of the reason for the rejection.

BYE

Sent in response to a client BYE request, after which the server will close the connection and end the session with the client.

1.9.1 Brief Request Results

Request results can be returned in one of two forms depending on the level of detail required: brief and XML. The XML form contains more details in a more precise format.

All responses are in plain text but see also the appendix on Character Encodings.

The server signals the end of the response with a blank line.

Note: The values of the <result code> items are the results codes from the scanner plus some from the daemon. See the scanner document, savi_sen.pdf, from the SAVI Developer Toolkit, and the daemon documentation.

1.9.1.1 Responses to SCAN requests

The following may be seen in each is terminate by an EOL sequence:-

EVENT FILE <location>

EVENT DIR <location>

Sent whenever the engine starts to scan a file or directory. Only seen if "report: file" has been set.

EVENT ARCHIVE <location>

Sent whenever the engine starts to scan an object in an archive type file. Only seen if "report: archive" has been set.

EVENT ERROR <result code> <location>

Sent whenever the engine detects an error. Only seen if "report: error" has been set.

EVENT VIRUS <virus name> <location>

Sent whenever the engine has detected a threat (the use of the word 'VIRUS' is conventional). Only seen if "report: virus" has been set.

TYPE <sophos type> <mime type>

Sent whenever the engine has classified a file. There may be more than one TYPE per file depending on its structure. Only seen if "report: classification" has been set.

FILE <location>

Sent if "report: classification" has been set.

VIRUS <virus name> <location>

Sent if a threat, such as a virus, was found. Threats are always reported

OK <result code> <text>

The SAVI function on the file or object completed successfully. <text> is a brief explanatory text for <result code>.

FAIL <result code> <location>

The SAVI function on the file or object failed to complete, for example the file that was to be scanned may not exist or the user may not have permission to read the file. <location> will be blank if the request was for SCANDATA.

DONE <worst result> <text>

The request has been completed. The <worst result> is the most serious result seen during the course of the operation. <text> is a brief explanatory text for <worst result>. The results are ordered from best to worst as OK, Failure, Virus detected

1.9.1.2 Responses to OPTIONS requests

The following may be seen in response to a OPTIONS request, each is terminate by an EOL sequence:-

OK <result code>

The SAVI function completed successfully

FAIL <result code>

The SAVI function failed to complete, for example the file that was requested to be scanned may not exist or the user may not have permission to read the file.

1.9.1.3 Responses to QUERY SERVER requests

The following may be seen in response to a QUERY SERVER request, each is terminated by an EOL sequence (order is not significant) :-

method: <method name>

<method name> is a method available to the client.

maxscandata: <bytes>

<bytes> is the maximum amount of data the server will accept for a SCANDATA request. Zero means it is unlimited.

maxmemorysize: <bytes>

<bytes> is the maximum amount of data the server will hold in memory before using a temporary file when implementing a SCANDATA request.

maxclassificationsize: <bytes>

<bytes> is the maximum amount of data from the start of a file that the client needs to send in a SCANDATA request when requesting a classification only.

version

The version of the server.

1.9.2 Full Request Results

The full request results return more details in XML format than the brief format.

The descriptions below show the tag name and attributes, if any.

1.9.2.1 Responses to SCAN requests

The following may be seen in response to a SCAN request, each is terminate by an EOL sequence:-

results

An envelope for the results of the request

directory name=<directory name>

directory is used as an envelope when a SCANDIR or SCANDIRR requests a scan of a directory. It is not used otherwise. The name may be either a full or relative path name.

file name=<file name>

file is used as an envelope for the results of a file scan The name may be either full or relative path names.

callback function=”OnFileFound”

An envelope for details from an onfilefound event.

See location.

callback function=”OnVirusFound”

An envelope for details from an onvirusfound event.

See sweepresult.

classification

An envelope for details from an onclassification event.

See type, mime.

sweepresult

An envelope for details of the results of a scan (sweep).

See virustype, location, name, disinfectable

done status=<status> result =<result code>

The overall result of the request. It is of significance when a request requires more than one call into SAVI, for example when scanning a directory.

<result code> is the worst result code returned by SAVI. See the SAV Interface documentation for details.

<status> is either “OK” or “FAIL” as determined by the SAVI result code.

NB FAIL indicates that SAVI could not complete an operation, it does not indicate the presence , or absence, of a threat.

See text. Also see result.

result status=<status> result =<result code>

The result code returned by SAVI. See the SAV Interface documentation for details.

<result code> is the actual SAVI result code.

<status> is either "OK" or "FAIL" as determined by the SAVI result code.

NB FAIL indicates that SAVI could not complete the request operation, it does not indicate the presence of a threat.

See text

text

Explanatory text, if available describing the <result code>

Context: result, done

archive

If an archive file is being scanned the location of an object within the archive. The files path is extended to show the location of the object within the archive.

Context: callback

location

The file or path of the object being scanned. If an archive file is being scanned the location the files path will be extended to show the location of objects within the archive.

Context: sweepresult, callback

type

The Sophos type, or classification, of an object. See the SAV Interface documentation for details.

Context: classification

virustype

An indication of the type of threat and how it was detected.

Context: sweepresult

name

The name of the threat.

Context: sweepresult

disinfectable

Indicates if a threat is disinfectable or not.

1.9.2.2 Responses to OPTIONS requests

The following may be seen in response to a OPTIONS request, each is terminated by an EOL sequence:-

done status=<status>

The result code returned by SAVI. See the SAV Interface documentation for details. <status> is either “OK” or “FAIL” as determined by the SAVI result code. For consistency, a result is returned irrespective of whether the OPTIONS required SAVI functionality or not.

1.9.2.3 Responses to QUERY SERVER requests

The following may be seen in response to a QUERY SERVER request, each is terminated by an EOL sequence:-

results

An envelope for the results of the request

methods

An envelope for the list of methods.

See method.

Context: results.

method

The name of a method (request) available to the client.

Context: method.

maxmemorysize

The maximum amount of data the server will put in memory before using a temporary file when implementing a SCANDATA request.

Context: results.

maxclassificationsize

The maximum amount of data from the start of a file that the client needs to send in a SCANDATA request when requesting a classification only.

Context: results.

version

The version of the server.

Context:results

2 Client Code Illustration

The following piece of pseudo-code is to illustrate how a client communicates with the server.

The function “Receive Line” receives characters up to the end of line sequence.

The function “Receive Message” receives lines up to the first blank line.

```
Connect to server
Receive Line
If protocol or version is incorrect
    Abort
Send protocol and version
Receive Line
If response is reject
    Abort
While not finished
    Send Request
    Receive Line
    If response is reject
        Abort
    Receive Message
    Analyse Message
Send BYE
Receive Line
Disconnect from server
```

If there is only one request to make of the server the request may be combined with the initial sending of the protocol and version:-

```
Connect to server
Receive Line
If protocol or version is incorrect
        Abort
Send protocol and version + request
Receive Line
If response is reject
        Abort
Receive Message
Analyse Message
Send BYE
Receive Line
Disconnect from server
```

3 Examples

3.1 Scanning a file

Simplest request is to scan a local file:-

SSSP/1.0 SCANFILE / var/scan//incoming/nasty.thing

To sweep the given data is:-

SSSP/1.0 SCANDATA 12345

[12345 bytes of data to be swept]

Example exchange

Client	Server
[Connect]	
	[Accept connection] OK SSSP/1.0
SSSP/1.0	
	ACC 8AB007/1
OPTIONS option: zipdecompression 1 option: html 1	
	ACC 8AB007/2 DONE OK 0000
SCANFILE / var/scan/xyz.zip	
	ACC 8AB007/3 [scans file] VIRUS EICAR / var/scan/xyz.zip/eicar.com OK 0203 Virus found during scan DONE OK 0203 Virus found during scan

```
SSSP/1.0 SCANFILE / var/scan/ abc.html
                                        ACC 8AB007/4
                                        [scans file]
                                        OK 0000
                                        DONE OK 0000
    Disconnect]
```

3.2 Scanning a directory, xml output

This is the same request as above except that XML form output has been requested.

```
SSSP/1.0 OPTIONS
output: xml
report: file archive virus classification

SCANDIRR /var/scan/ssspd/testfiles/
```

The server responds with an ACC to indicate that it has accepted the request and follows with the details in XML form.

```
ACC 5DB042/2
<results>
  <directory name=”/ var/scan/ssspd/testfiles/”>
    <file name=”pemid4.exe”>
      <callback function=”OnFileFound”>
        <location>pemid4.exe</location>
      </callback>
      <classification>
        <type>0x00000060</type>
      </classification>
    </file>
    <file name=”pemid3.exe”>
      <callback function=”OnFileFound”>
        <location>pemid3.exe</location>
      </callback>
      <classification>
        <type>0x00000060</type>
     </classification>
    </file>
```

```
...
    <directory name=”tstfiles”>
      <directory name=”PUA”>
        <directory name=”Idents”>
          <file name=”pua.dat”>
            <callback function=”OnFileFound”>
              <location>pua.dat</location>
            </callback>
            <classification>
              <type>0x00000080</type>
            </classification>
          </file>
...
    <file name=”EICAR.COM”>
      <callback function=”OnFileFound”>
        <location>EICAR.COM</location>
      </callback>
      <callback function=”OnVirusFound”>
        <sweepresult>
          <virustype>Identity</virustype>
   <location> var/scan/ssspd/testfiles/EICAR.COM</location>
          <name>EICAR-AV-Test</name>
          <disinfectable>No</disinfectable>
        </sweepresult>
        <return>0x00040226</return>
      </callback>
      <sweepresult>
        <virustype>Identity</virustype>
  <location> var/scan/ssspd/testfiles/EICAR.COM</location>
       <name>EICAR-AV-Test</name>
        <disinfectable>No</disinfectable>
      </sweepresult>
    </file>
    <file name=”EICAR.zip”>
      <callback function=”OnFileFound”>
        <location>EICAR.zip</location>
```

```
            <return>0x00040226</return>
         </callback>
         <classification>
            <type>0x00000030</type>
         </classification>
      </file>
      <file name=”rootowned”>
      </file>
      <file name=”libxslt-1.1.15-1.i386.rpm”>
         <callback function=”OnFileFound”>
            <location>libxslt-1.1.15-1.i386.rpm</location>
         </callback>
         <classification>
            <type>0x000000F8</type>
         </classification>
      </file>

...

      <file name=”jde-ug-content.xml”>
         <callback function=”OnFileFound”>
            <location>jde-ug-content.xml</location>
         </callback>
         <classification>
            <type>0x000000D9</type>
         </classification>
      </file>
   </directory>
   <result status=”FAIL”>0x00000210</result>
</results>
```

Appendix: Request Format Specification

In the following specification:-

italics are used to indicate a server response when needed to distinguish between server and client.

{} are used to indicate optional elements

[a|b|..] is used to indicate a choice of a, b or ...

a) Session entities:-

On accepting the connection the server responds with a "OK SSSP/1.0" message

<session> := <server greeting><first request><response>{<exchange>}

<server greeting> := "OK "<sep><protocol><eol><eol>

The first message from the client must include the version of the protocol that it will use. It must not be more advanced than the one the server specified.

<first request> := <protocol>{<client command>}<eol>

<exchange> := <client request><response>{<client request><response>}

On subsequent exchanges the client may drop the protocol specification.

b) Client entities:-

<client request> := {<protocol>} <client command><eol>

<client command> := <scan file command>| <scan dir command>|<scan dirr comand>|<scan data command>|<options command>|<query command>|<bye command>

Note the filename is terminated by the eol and so does not need escaping etc., just so long as it does not contain any carriage return or newline characters.

<scan file command> := “SCANFILE”<sep><filename><eol>

<scan dir command> := “SCANDIR”<sep><filename><eol>

<scan dirr command> := “SCANDIRR”<sep><filename><eol>

<scan data command> := “SCANDATA”<sep><size><eol>
<datastream>

<size> is the number of bytes in the data stream.

<options command> := “OPTIONS”<eol><options><eol>

The query command will result in a list of the commands available to the client.

<query command> :=
    <query server command> |
    <query savi command> |
    <query engine command>

<query server command> := “QUERY” {<sep>“SERVER”}<eol>

<query savi command> := “QUERY”<sep>”SAVI”<eol>

<query engine command> := “QUERY” <sep> “ENGINE”<eol>

<bye command> := “BYE”<eol>

<options> := [| <option><options>]

<option> := [<output option>|<savi option>|
<savi default option>|<time limit option>|<event option>|
<report option>]<eol>

SAVI options are limited to the status options, those specified as U32 in the SAVI Supplement. The default savi option resets SAVI to the configuration specified by the server’s SAVI configuration. SAVI options are implemented in order so that if used the default option should be first.

<output option> := “output”<optionsep>[“brief”|”xml”]

<savi option> := [“savists”|”savigrp”]<optionsep>
<savi option name><sep><savi option value>

<savi default option> := “savidefault”<optionsep>

The client can request notification of certain events during the scan process and specify the action the engine should take at that time. The event handling can be reset by specifying the event with no parameters.

<event option> := “event”<optionsep>{
<event type>{<sep><engine action>}}

<event type> := [“onvirusfound”|”onerrorfound”]

<engine action> := [“continuethis”|”continuenext”|
”stopthis”|”terminate”]

NB The onclassification report type will cause the StorageReportAll engine option to be set until the onclassification event is reset.

<report option> := “report”<optionsep>{<report type list>}

<report type list> := <report>{<sep><report type list>}

<report> := [“file”|”virus”|”error”|”classification”|”all”]

c) Server entities:

ACC and REJ indicate whether the server understood the message, or not. If acceptable to the server, the request goes on to SAVI and SAVI’s response follows.

<response> := [<accept><results>|<reject>]

<accept> := “ACC”<sep><request id><eol>

<reject> := “REJ”<sep><reject code><eol>

See the main document for the format of the various responses from the server. Note that even for the XML format the response is terminated with a blank line.

<results> := [<savi response>|<query server response>]<eol>

d) Misc entities:

<protocol> := "SSSP"."/".<version>

The minor part of the version is a number in its own write so that 1.10 is later than 1.1, 1.2 ... 1.9.

<version> := N."".N

<reject code> := N

<size> := N

<datastream> := <binary data>

<sep> := <spaces>

<optionsep> := ":"

<eol> := "\r\n"

<request id> := non-space string

<spaces> are one or more space characters.

Appendix: %XX Encoding

%XX encoding is used to encode inconvenient characters, for example carriage returns that occur in file names.

The XX are the hex equivalent of the character. A carriage return is represented as "%0C", a space as "%20", and so on. The % character itself may occur in a string and if it does it must also be encoded. There are two ways to do this, either represent it as "%%" or %XX encode it as "%25".

Decoding consists of scanning the string and replacing all occurrences of "%%" with "%" and "%XX" sequences with the character equivalent of the "XX".

SSSP has a few special characters. These are space, carriage return, new line and percent characters. Whenever these characters occur in places other than as required by the protocol they should be replaced by their %XX equivalent.

Appendix: Character Encodings

There are 3 elements to the characters used by SSSP:-

1. The requests, SAVI options, server information, threat names and so on
2. The text which accompanies the result code in the DONE statement.
3. Threat locations

For (1) the text is always 7-bit ASCII

For (2) the text is as taken from the language file, so whatever encoding has been used there, is the encoding of this text.

For (3) the situation is more complex. Nominally the encoding is the encoding in use by the file system, however it is entirely possible whether by accident or malicious intent to create files and directories with characters that are illegal in the file system's encoding scheme. The file system could even have a mix of encoding schemes in use. To compound this problem, archive files may use entirely different encoding schemes.

As a consequence of all this the server is very aggressive in using the %XX encoding mechanism when communicating locations leaving it to the client to handle the encoding scheme. It is unnecessary for the client to be this aggressive when sending a pathname to the server and the 'raw' characters will suffice provided the necessary percent encoding of the special characters (carriage return, new line, space, percent) are all %XX encoded.

Appendix: Classifying Files

This is an example of what can be done using SSSP. It demonstrates using the virus engine to classify a file.

Note: the engine actually classifies streams, the components of a file and does so for internal purposes. As a consequence the results of using this technique should be treated with some circumspect. The file types are the types as determined by the virus engine for the virus engine; this is not a general purpose file typing function. See the SAV Interface documentation for a list of file types recognised by the virus engine.

Normally the engine can determine the type of file well within 4K of the start of the file and it may be advantageous to a client to be able to determine the type of a large file before sending it for scanning. It was considered to add a method to SSSP that would allow a client to classify a file, however this can be better achieved using a SCANDATA method with suitable options set:-

```
OPTIONS
report: classification
savists: storagedetonly 1
```

```
SCANDATA 4096
```
followed by the first 4K of the file. 4096 is the number of bytes being sent to the server and is the maximum that should be needed for a classification, obviously if the file is smaller it is sent in toto with the size set accordingly. The StorageDetOnly option is needed to prevent the engine from scanning the data for malware.

The daemon will set the StorageReportAll option as otherwise the OnClassification callback is not called. The daemon will also unset StorageReportAll when the onclassification report is cleared:-

```
OPTIONS
report:
```

Appendix: Sophie protocol

(This protocol was determined from examining the code; it may be missing details or be incorrect.)

Sophie was developed by Vanja Hrustic as a free open source project.

The protocol for Sophie depends on whether a TCP/IP socket is being used or a unix socket.

Unix Socket:

Client	Server
[Connect]	
	[Accept connection]
</full/path/name>	
	[Scan file/dir]
	<scan result>
[Repeat if required]	
[Disconnect]	

TCP/IP Socket:

[Connect]	
	[Accept connection]
<name/size>	
	“OK”
<data>	
	[Scan data sent]
	<scan result>
[Repeat if required]	
[Disconnect]	

All text sent is terminated with an EOL character, a newline.

</full/path/name> may be a file or a directory path.

In <name>/<size> name can be any string but the size must be the size of the data being sent.

<data> is the raw file contents.

<scan result> is <code>{:<virusname>}. Code is 1 for virus in which case <virusname> is sent as well, 0 is clean, and -1 is for an unknown status.

If <full/path/name> specifies a directory only one <scan result> is returned, one or more detected viruses will cause a code of 1 to be returned.