03-02-2022-24064301-Hauptpost-0584

Log File Output Log file output is displayed in the following format, and explained further below:

```
Time: 09/08-15:07:31.603880
event_id: 2
192.168.169.3 -> 192.168.169.5 (portscan) TCP Filtered Portscan
Priority Count: 0
Connection Count: 200
IP Count: 2
Scanner IP Range: 192.168.169.3:192.168.169.4
Port/Proto Count: 200
Port/Proto Range: 20:47557
```

If there are open ports on the target, one or more additional tagged packet(s) will be appended:

```
Time: 09/08-15:07:31.603881
event_ref: 2
192.168.169.3 -> 192.168.169.5 (portscan) Open Port
Open Port: 38458
```



1. **Event_id/Event_ref**

   These fields are used to link an alert with the corresponding `Open Port` tagged packet

2. **Priority Count**

   `Priority Count` keeps track of bad responses (resets, unreachables). The higher the priority count, the more bad responses have been received.

3. **Connection Count**

   `Connection Count` lists how many connections are active on the hosts (src or dst). This is accurate for connection-based protocols, and is more of an estimate for others. Whether or not a portscan was filtered is determined here. High connection count and low priority count would indicate filtered (no response received from target).

4. **IP Count**

   IP Count keeps track of the last IP to contact a host, and increments the count if the next IP is different. For one-to-one scans, this is a low number. For active hosts this number will be high regardless, and one-to-one scans may appear as a distributed scan.

5. **Scanned/Scanner IP Range**

   This field changes depending on the type of alert. Portsweep (one-to-many) scans display the scanned IP range; Portscans (one-to-one) display the scanner IP.

6. **Port Count**

   Port Count keeps track of the last port contacted and increments this number when that changes. We use this count (along with IP Count) to determine the difference between one-to-one portscans and one-to-one decoys.

Tuning sfPortscan

The most important aspect in detecting portscans is tuning the detection engine for your network(s). Here are some tuning tips:

1. **Use the watch_ip, ignore_scanners, and ignore_scanned options.**

   It's important to correctly set these options. The `watch_ip` option is easy to understand. The analyst should set this option to the list of CIDR blocks and IPs that they want to watch. If no `watch_ip` is defined, sfPortscan will watch all network traffic.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 51

The ignore_scanners and ignore_scanned options come into play in weeding out legitimate hosts that are very active on your network. Some of the most common examples are NAT IPs, DNS cache servers, syslog servers, and nfs servers. sfPortscan may not generate false positives for these types of hosts, but be aware when first tuning sfPortscan for these IPs. Depending on the type of alert that the host generates, the analyst will know which to ignore it as. If the host is generating portsweep events, then add it to the ignore_scanners option. If the host is generating portscan alerts (and is the host that is being scanned), add it to the ignore_scanned option.

2. **Filtered scan alerts are much more prone to false positives.**

   When determining false positives, the alert type is very important. Most of the false positives that sfPortscan may generate are of the filtered scan alert type. So be much more suspicious of filtered portscans. Many times this just indicates that a host was very active during the time period in question. If the host continually generates these types of alerts, add it to the ignore_scanners list or use a lower sensitivity level.

3. **Make use of the Priority Count, Connection Count, IP Count, Port Count, IP Range, and Port Range to determine false positives.**

   The portscan alert details are vital in determining the scope of a portscan and also the confidence of the portscan. In the future, we hope to automate much of this analysis in assigning a scope level and confidence level, but for now the user must manually do this. The easiest way to determine false positives is through simple ratio estimations. The following is a list of ratios to estimate and the associated values that indicate a legitimate scan and not a false positive.

   Connection Count / IP Count: This ratio indicates an estimated average of connections per IP. For portscans, this ratio should be high, the higher the better. For portsweeps, this ratio should be low.

   Port Count / IP Count: This ratio indicates an estimated average of ports connected to per IP. For portscans, this ratio should be high and indicates that the scanned host's ports were connected to by fewer IPs. For portsweeps, this ratio should be low, indicating that the scanning host connected to few ports but on many hosts.

   Connection Count / Port Count: This ratio indicates an estimated average of connections per port. For portscans, this ratio should be low. This indicates that each connection was to a different port. For portsweeps, this ratio should be high. This indicates that there were many connections to the same port.

   The reason that Priority Count is not included, is because the priority count is included in the connection count and the above comparisons take that into consideration. The Priority Count play an important role in tuning because the higher the priority count the more likely it is a real portscan or portsweep (unless the host is firewalled).

4. **If all else fails, lower the sensitivity level.**

   If none of these other tuning techniques work or the analyst doesn't have the time for tuning, lower the sensitivity level. You get the best protection the higher the sensitivity level, but it's also important that the portscan detection engine generate alerts that the analyst will find informative. The low sensitivity level only generates alerts based on error responses. These responses indicate a portscan and the alerts generated by the low sensitivity level are highly accurate and require the least tuning. The low sensitivity level does not catch filtered scans; since these are more prone to false positives.

2.2.4 RPC Decode

The rpc_decode preprocessor normalizes RPC multiple fragmented records into a single un-fragmented record. It does this by normalizing the packet into the packet buffer. If stream5 is enabled, it will only process client-side traffic. By default, it runs against traffic on ports 111 and 32771.

Format

```
preprocessor rpc_decode: \
    <ports> [ alert_fragments ] \
    [no_alert_multiple_requests] \
    [no_alert_large_fragments] \
    [no_alert_incomplete]
```


1644470956460125853 19

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 52

03-02-2022-24064301-Hauptpost-0585

Option	Description
alert_fragments	Alert on any fragmented RPC record.
no_alert_multiple_requests	Don't alert when there are multiple records in one packet.
no_alert_large_fragments	Don't alert when the sum of fragmented records exceeds one packet.
no_alert_incomplete	Don't alert when a single fragment record exceeds the size of one packet.

2.2.5 Performance Monitor

This preprocessor measures Snort's real-time and theoretical maximum performance. Whenever this preprocessor is turned on, it should have an output mode enabled, either "console" which prints statistics to the console window or "file" with a file name, where statistics get printed to the specified file name. By default, Snort's real-time statistics are processed. This includes:

- Time Stamp
- Drop Rate
- Mbits/Sec (wire) [duplicated below for easy comparison with other rates]
- Alerts/Sec
- K-Pkts/Sec (wire) [duplicated below for easy comparison with other rates]
- Avg Bytes/Pkt (wire) [duplicated below for easy comparison with other rates]
- Pat-Matched [percent of data received that Snort processes in pattern matching]
- Syns/Sec
- SynAcks/Sec
- New Sessions Cached/Sec
- Sessions Del fr Cache/Sec
- Current Cached Sessions
- Max Cached Sessions
- Stream Flushes/Sec
- Stream Session Cache Faults
- Stream Session Cache Timeouts
- New Frag Trackers/Sec
- Frag-Completes/Sec
- Frag-Inserts/Sec
- Frag-Deletes/Sec
- Frag-Auto Deletes/Sec [memory DoS protection]
- Frag-Flushes/Sec
- Frag-Current [number of current Frag Trackers]
- Frag-Max [max number of Frag Trackers at any time]
- Frag-Timeouts
- Frag-Faults

16444709564601258531 9

- Number of CPUs [*** Only if compiled with LINUX_SMP ***, the next three appear for each CPU]
- CPU usage (user)
- CPU usage (sys)
- CPU usage (Idle)
- Mbits/Sec (wire) [average mbits of total traffic]
- Mbits/Sec (ipfrag) [average mbits of IP fragmented traffic]
- Mbits/Sec (ipreass) [average mbits Snort injects after IP reassembly]
- Mbits/Sec (tcprebuilt) [average mbits Snort injects after TCP reassembly]
- Mbits/Sec (applayer) [average mbits seen by rules and protocol decoders]
- Avg Bytes/Pkt (wire)
- Avg Bytes/Pkt (ipfrag)
- Avg Bytes/Pkt (ipreass)
- Avg Bytes/Pkt (tcprebuilt)
- Avg Bytes/Pkt (applayer)
- K-Pkts/Sec (wire)
- K-Pkts/Sec (ipfrag)
- K-Pkts/Sec (ipreass)
- K-Pkts/Sec (tcprebuilt)
- K-Pkts/Sec (applayer)
- Total Packets Received
- Total Packets Dropped (not processed)
- Total Packets Blocked (inline)
- Percentage of Packets Dropped
- Total Filtered TCP Packets
- Total Filtered UDP Packets
- Midstream TCP Sessions/Sec
- Closed TCP Sessions/Sec
- Pruned TCP Sessions/Sec
- TimedOut TCP Sessions/Sec
- Dropped Async TCP Sessions/Sec
- TCP Sessions Initializing
- TCP Sessions Established
- TCP Sessions Closing
- Max TCP Sessions (interval)
- New Cached UDP Sessions/Sec



03-02-2022-24064301-HauptPost-0586

- Cached UDP Ssns Del/Sec
- Current Cached UDP Sessions
- Max Cached UDP Sessions
- Current Attribute Table Hosts (Target Based)
- Attribute Table Reloads (Target Based)
- Mbits/Sec (Snort)
- Mbits/Sec (sniffing)
- Mbits/Sec (combined)
- uSeconds/Pkt (Snort)
- uSeconds/Pkt (sniffing)
- uSeconds/Pkt (combined)
- KPkts/Sec (Snort)
- KPkts/Sec (sniffing)
- KPkts/Sec (combined)



The following options can be used with the performance monitor:

- `flow` - Prints out statistics about the type of traffic and protocol distributions that Snort is seeing. This option can produce large amounts of output.
- `events` - Turns on event reporting. This prints out statistics as to the number of rules that were evaluated and didn't match (*non-qualified events*) vs. the number of rules that were evaluated and matched (*qualified events*). A high *non-qualified event* to *qualified event* ratio can indicate there are many rules with either minimal content or no content that are being evaluated without success. The fast pattern matcher is used to select a set of rules for evaluation based on the longest `content` or a `content` modified with the `fast_pattern` rule option in a rule. Rules with short, generic contents are more likely to be selected for evaluation than those with longer, more unique contents. Rules without `content` are not filtered via the fast pattern matcher and are always evaluated, so if possible, adding a `content` rule option to those rules can decrease the number of times they need to be evaluated and improve performance.
- `max` - Turns on the theoretical maximum performance that Snort calculates given the processor speed and current performance. This is only valid for uniprocessor machines, since many operating systems don't keep accurate kernel statistics for multiple CPUs.
- `console` - Prints statistics at the console.
- `file` - Prints statistics in a comma-delimited format to the file that is specified. Not all statistics are output to this file. You may also use `snortfile` which will output into your defined Snort log directory. Both of these directives can be overridden on the command line with the `-Z` or `--perfmon-file` options. At startup, Snort will log a distinctive line to this file with a timestamp to all readers to easily identify gaps in the stats caused by Snort not running.
- `pktcnt` - Adjusts the number of packets to process before checking for the time sample. This boosts performance, since checking the time sample reduces Snort's performance. By default, this is 10000.
- `time` - Represents the number of seconds between intervals.
- `accumulate` or `reset` - Defines which type of drop statistics are kept by the operating system. By default, `reset` is used.
- `atexitonly` - Dump stats for entire life of Snort.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 55

- `max_file_size` - Defines the maximum size of the comma-delimited file. Before the file exceeds this size, it will be rolled into a new date stamped file of the format YYYY-MM-DD, followed by YYYY-MM-DD.x, where x will be incremented each time the comma delimited file is rolled over. The minimum is 4096 bytes and the maximum is 2147483648 bytes (2GB). The default is the same as the maximum.
- `flow-ip` - Collects IP traffic distribution statistics based on host pairs. For each pair of hosts for which IP traffic has been seen, the following statistics are collected for both directions (A to B and B to A):
  - TCP Packets
  - TCP Traffic in Bytes
  - TCP Sessions Established
  - TCP Sessions Closed
  - UDP Packets
  - UDP Traffic in Bytes
  - UDP Sessions Created
  - Other IP Packets
  - Other IP Traffic in Bytes

  These statistics are printed and reset at the end of each interval.
- `flow-ip-file` - Prints the flow IP statistics in a comma-delimited format to the file that is specified. All of the statistics mentioned above, as well as the IP addresses of the host pairs in human-readable format, are included.
- `flow-ip-memcap` - Sets the memory cap on the hash table used to store IP traffic statistics for host pairs. Once the cap has been reached, the table will start to prune the statistics for the least recently seen host pairs to free memory. This value is in bytes and the default value is 52428800 (50MB).



Examples

```
preprocessor perfmonitor: \
    time 30 events flow file stats.profile max console pktcnt 10000

preprocessor perfmonitor: \
    time 300 file /var/tmp/snortstat pktcnt 10000

preprocessor perfmonitor: \
    time 30 flow-ip flow-ip-file flow-ip-stats.csv pktcnt 1000
```

2.2.6 HTTP Inspect

HTTP Inspect is a generic HTTP decoder for user applications. Given a data buffer, HTTP Inspect will decode the buffer, find HTTP fields, and normalize the fields. HTTP Inspect works on both client requests and server responses.

The current version of HTTP Inspect only handles stateless processing. This means that HTTP Inspect looks for HTTP fields on a packet-by-packet basis, and will be fooled if packets are not reassembled. This works fine when there is another module handling the reassembly, but there are limitations in analyzing the protocol. Future versions will have a stateful processing mode which will hook into various reassembly modules.

HTTP Inspect has a very "rich" user configuration. Users can configure individual HTTP servers with a variety of options, which should allow the user to emulate any type of web server. Within HTTP Inspect, there are two areas of configuration: global and server.

Global Configuration

The global configuration deals with configuration options that determine the global functioning of HTTP Inspect. The following example gives the generic global configuration format:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 56

03-02-2022-24064301-Hauptpost-0587

Format

```
    preprocessor http_inspect: \
        global \
        iis_unicode_map <map_filename> \
        codemap <integer> \
        [detect_anomalous_servers] \
        [proxy_alert] \
[max_gzip_mem <num>] \
[compress_depth <num>] [decompress_depth <num>] \
disabled
```

You can only have a single global configuration, you'll get an error if you try otherwise.

Configuration

1. `iis_unicode_map <map_filename> [codemap <integer>]`

   This is the global `iis_unicode_map` file. The `iis_unicode_map` is a required configuration parameter. The map file can reside in the same directory as `snort.conf` or be specified via a fully-qualified path to the map file.

   The `iis_unicode_map` file is a Unicode codepoint map which tells HTTP Inspect which codepage to use when decoding Unicode characters. For US servers, the codemap is usually 1252.

   A Microsoft US Unicode codepoint map is provided in the Snort source `etc` directory by default. It is called `unicode.map` and should be used if no other codepoint map is available. A tool is supplied with Snort to generate custom Unicode `maps--ms_unicode_generator.c`, which is available at `http://www.snort.org/dl/contrib/`.

   ⚠NOTE

   Remember that this configuration is for the global IIS Unicode map, individual servers can reference their own IIS Unicode map.

2. `detect_anomalous_servers`

   This global configuration option enables generic HTTP server traffic inspection on non-HTTP configured ports, and alerts if HTTP traffic is seen. Don't turn this on if you don't have a default server configuration that encompasses all of the HTTP server ports that your users might access. In the future, we want to limit this to specific networks so it's more useful, but for right now, this inspects all network traffic. This option is turned off by default.

3. `proxy_alert`

   This enables global alerting on HTTP server proxy usage. By configuring HTTP Inspect servers and enabling `allow_proxy_use`, you will only receive proxy use alerts for web users that aren't using the configured proxies or are using a rogue proxy server.

   Please note that if users aren't required to configure web proxy use, then you may get a lot of proxy alerts. So, please only use this feature with traditional proxy environments. Blind firewall proxies don't count.

4. `compress_depth <integer>` This option specifies the maximum amount of packet payload to decompress. This value can be set from 1 to 65535. The default for this option is 1460.

   ⚠NOTE

   Please note, in case of multiple policies, the value specified in the default policy is used and this value overwrites the values specified in the other policies. In case of `unlimited_decompress` this should be set to its max value. This value should be specified in the default policy even when the HTTP inspect is turned off using the `disabled` keyword.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 57

16444709564601258531 9

5. decompress_depth <integer> This option specifies the maximum amount of decompressed data to obtain from the compressed packet payload. This value can be set from 1 to 65535. The default for this option is 2920.

> ⚠NOTE
>
> Please note, in case of multiple policies, the value specified in the default policy is used and this value overwrites the values specified in the other policies. In case of unlimited_decompress this should be set to its max value. This value should be specified in the default policy even when the HTTP inspect is turned off using the disabled keyword.

6. max_gzip_mem

   This option determines (in bytes) the maximum amount of memory the HTTP Inspect preprocessor will use for decompression. This value can be set from 3276 bytes to 100MB. This option along with compress_depth and decompress_depth determines the gzip sessions that will be decompressed at any given instant. The default value for this option is 838860.

> ⚠NOTE
>
> This value should be specified in the default policy even when the HTTP inspect is turned off using the disabled keyword. It is suggested to set this value such that the max gzip session calculated as follows is at least 1.
> max gzip session = max_gzip_mem /(decompress_depth + compress_depth)

7. disabled

   This optional keyword is allowed with any policy to avoid packet processing. This option disables the preprocessor. When the preprocessor is disabled only the "max_gzip_mem", "compress_depth" and "decompress_depth" options are applied when specified with the configuration. Other options are parsed but not used. Any valid configuration may have "disabled" added to it.

1644470956460125853 19

Example Global Configuration

```
preprocessor http_inspect: \
    global iis_unicode_map unicode.map 1252
```

Server Configuration

There are two types of server configurations: default and by IP address.

Default This configuration supplies the default server configuration for any server that is not individually configured. Most of your web servers will most likely end up using the default configuration.

Example Default Configuration

```
preprocessor http_inspect_server: \
    server default profile all ports { 80 }
```

Configuration by IP Address This format is very similar to "default", the only difference being that specific IPs can be configured.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 58

03-02-2022-24064301-Hauptpost-0588

Example IP Configuration

```
preprocessor http_inspect_server: \
    server 10.1.1.1 profile all ports { 80 }
```

Configuration by Multiple IP Addresses This format is very similar to "Configuration by IP Address", the only difference being that multiple IPs can be specified via a space separated list. There is a limit of 40 IP addresses or CIDR notations per `http_inspect_server` line.

Example Multiple IP Configuration

```
preprocessor http_inspect_server: \
    server { 10.1.1.1 10.2.2.0/24 } profile all ports { 80 }
```

Server Configuration Options

Important: Some configuration options have an argument of 'yes' or 'no'. This argument specifies whether the user wants the configuration option to generate an HTTP Inspect alert or not. The 'yes/no' argument does not specify whether the configuration option itself is on or off, only the alerting functionality. In other words, whether set to 'yes' or 'no', HTTP normalization will still occur, and rules based on HTTP traffic will still trigger.



1. `profile <all|apache|iis|iis5_0|iis4_0>`

   Users can configure HTTP Inspect by using pre-defined HTTP server profiles. Profiles allow the user to easily configure the preprocessor for a certain type of server, but are not required for proper operation.

   There are five profiles available: all, apache, iis, iis5_0, and iis4_0.

   1-A. `all`

   The `all` profile is meant to normalize the URI using most of the common tricks available. We alert on the more serious forms of evasions. This is a great profile for detecting all types of attacks, regardless of the HTTP server. `profile all` sets the configuration options described in Table 2.3.

   1-B. `apache`

   The `apache` profile is used for Apache web servers. This differs from the `iis` profile by only accepting UTF-8 standard Unicode encoding and not accepting backslashes as legitimate slashes, like IIS does. Apache also accepts tabs as whitespace. `profile apache` sets the configuration options described in Table 2.4.

   1-C. `iis`

   The `iis` profile mimics IIS servers. So that means we use IIS Unicode codemaps for each server, %u encoding, bare-byte encoding, double decoding, backslashes, etc. `profile iis` sets the configuration options described in Table 2.5.

   1-D. `iis4_0, iis5_0`

   In IIS 4.0 and IIS 5.0, there was a double decoding vulnerability. These two profiles are identical to `iis`, except they will alert by default if a URL has a double encoding. Double decode is not supported in IIS 5.1 and beyond, so it's disabled by default.

   1-E. `default, no profile`

   The default options used by HTTP Inspect do not use a profile and are described in Table 2.6.

   Profiles must be specified as the first server option and cannot be combined with any other options except:

   - `ports`
   - `iis_unicode_map`
   - `allow_proxy_use`
   - `server_flow_depth`
   - `client_flow_depth`

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 59

Table 2.3: Options for the "all" Profile

Option	Setting
server_flow_depth	300
client_flow_depth	300
post_depth	0
chunk encoding	alert on chunks larger than 500000 bytes
iis_unicode_map	codepoint map in the global configuration
ASCII decoding	on, alert off
multiple slash	on, alert off
directory normalization	on, alert off
apache whitespace	on, alert off
double decoding	on, alert on
%u decoding	on, alert on
bare byte decoding	on, alert on
iis unicode codepoints	on, alert on
iis backslash	on, alert off
iis delimiter	on, alert off
webroot	on, alert on
non_strict URL parsing	on
tab_uri_delimiter	is set
max_header_length	0, header length not checked
max_headers	0, number of headers not checked

164447095646012585319

- `post_depth`
- `no_alerts`
- `inspect_uri_only`
- `oversize_dir_length`
- `normalize_headers`
- `normalize_cookies`
- `normalize_utf`
- `max_header_length`
- `max_headers`
- `extended_response_inspection`
- `enable_cookie`
- `inspect_gzip`
- `unlimited_decompress`
- `enable_xff`
- `http_methods`

These options must be specified after the `profile` option.

Example

```
preprocessor http_inspect_server: \
    server 1.1.1.1 profile all ports { 80 3128 }
```

2. `ports {<port> [<port><...>]}`

   This is how the user configures which ports to decode on the HTTP server. However, HTTPS traffic is encrypted and cannot be decoded with HTTP Inspect. To ignore HTTPS traffic, use the SSL preprocessor.

3. `iis_unicode_map <map_filename> codemap <integer>`

   The IIS Unicode map is generated by the program ms_unicode_generator.c. This program is located on the Snort.org web site at `http://www.snort.org/dl/contrib/` directory. Executing this program generates a

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 60

03-02-2022-24064301-Hauptpost-0589

Table 2.4: Options for the apache Profile

Option	Setting
server_flow_depth	300
client_flow_depth	300
post_depth	0
chunk encoding	alert on chunks larger than 500000 bytes
ASCII decoding	on, alert off
multiple slash	on, alert off
directory normalization	on, alert off
webroot	on, alert on
apache whitespace	on, alert on
utf_8 encoding	on, alert off
non_strict url parsing	on
tab_uri_delimiter	is set
max_header_length	0, header length not checked
max_headers	0, number of headers not checked



Unicode map for the system that it was run on. So, to get the specific Unicode mappings for an IIS web server, you run this program on that server and use that Unicode map in this configuration.

When using this option, the user needs to specify the file that contains the IIS Unicode map and also specify the Unicode map to use. For US servers, this is usually 1252. But the ms_unicode_generator program tells you which codemap to use for you server; it's the ANSI code page. You can select the correct code page by looking at the available code pages that the ms_unicode_generator outputs.

4. `extended_response_inspection` This enables the extended HTTP response inspection. The default http response inspection does not inspect the various fields of a HTTP response. By turning this option the HTTP response will be thoroughly inspected. The different fields of a HTTP response such as status code, status message, headers, cookie (when enable_cookie is configured) and body are extracted and saved into buffers. Different rule options are provided to inspect these buffers.

⚠NOTE

When this option is turned on, if the HTTP response packet has a body then any content pattern matches (without http modifiers) will search the response body ((decompressed in case of gzip) and not the entire packet payload. To search for patterns in the header of the response, one should use the http modifiers with content such as `http_header`, `http_stat_code`, `http_stat_msg` and `http_cookie`.

5. `enable_cookie` This options turns on the cookie extraction from HTTP requests and HTTP response. By default the cookie inspection and extraction will be turned off. The `Cookie` header line is extracted and stored in HTTP Cookie buffer for HTTP requests and `Set-Cookie` is extracted and stored in HTTP Cookie buffer for HTTP responses. In both cases the header name is also stored along with the cookie.

6. `inspect_gzip` This option specifies the HTTP inspect module to uncompress the compressed data(gzip/deflate) in HTTP response. You should select the config option "extended_response_inspection" before configuring this option. Decompression is done across packets. So the decompression will end when either the 'compress_depth' or 'decompress_depth' is reached or when the compressed data ends. When the compressed data is spanned across multiple packets, the state of the last decompressed packet is used to decompressed the data of the next packet. But the decompressed data are individually inspected. (i.e. the decompressed data from different packets are not combined while inspecting). Also the amount of decompressed data that will be inspected depends on the 'server_flow_depth' configured.

⚠NOTE

To enable compression of HTTP server response, Snort should be configured with the –enable-zlib flag.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 61

Table 2.5: Options for the iis Profile

Option	Setting
server_flow_depth	300
client_flow_depth	300
post_depth	-1
chunk encoding	alert on chunks larger than 500000 bytes
iis_unicode_map	codepoint map in the global configuration
ASCII decoding	on, alert off
multiple slash	on, alert off
directory normalization	on, alert off
webroot	on, alert on
double decoding	on, alert on
%u decoding	on, alert on
bare byte decoding	on, alert on
iis unicode codepoints	on, alert on
iis backslash	on, alert off
iis delimiter	on, alert on
apache whitespace	on, alert on
non_strict URL parsing	on
max_header_length	0, header length not checked
max_headers	0, number of headers not checked



7. `unlimited_decompress` This option enables the user to decompress unlimited gzip data (across multiple packets).Decompression will stop when the compressed data ends or when a out of sequence packet is received. To ensure unlimited decompression, user should set the 'compress_depth' and 'decompress_depth' to its maximum values in the default policy. The decompression in a single packet is still limited by the 'compress_depth' and 'decompress_depth'.

8. `enable_xff`

   This option enables Snort to parse and log the original client IP present in the X-Forwarded-For or True-Client-IP HTTP request headers along with the generated events. The XFF/True-Client-IP Original client IP address is logged only with unified2 output and is not logged with console (-A cmg) output.

   ⚠NOTE

   The original client IP from XFF/True-Client-IP in unified2 logs can be viewed using the tool u2spewfoo. This tool is present in the tools/u2spewfoo directory of snort source tree.

9. `server_flow_depth <integer>`

   This specifies the amount of server response payload to inspect. When `extended_response_inspection` is turned on, it is applied to the HTTP response body (decompressed data when `inspect_gzip` is turned on) and not the HTTP headers. When `extended_response_inspection` is turned off the `server_flow_depth` is applied to the entire HTTP response (including headers). Unlike `client_flow_depth` this option is applied per TCP session. This option can be used to balance the needs of IDS performance and level of inspection of HTTP server response data. Snort rules are targeted at HTTP server response traffic and when used with a small flow_depth value may cause false negatives. Most of these rules target either the HTTP header, or the content that is likely to be in the first hundred or so bytes of non-header data. Headers are usually under 300 bytes long, but your mileage may vary. It is suggested to set the `server_flow_depth` to its maximum value.

   This value can be set from -1 to 65535. A value of -1 causes Snort to ignore all server side traffic for ports defined in `ports` when `extended_response_inspection` is turned off. When the `extended_response_inspection` is turned on, value of -1 causes Snort to ignore the HTTP response body data and not the HTTP headers. Inversely, a value of 0 causes Snort to inspect all HTTP server payloads defined in "ports" (note that this will likely slow down IDS performance). Values above 0 tell Snort the number of bytes to inspect of the server response

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 62

03-02-2022-24064301-HauPtPost-0590

Table 2.6: Default HTTP Inspect Options

Option	Setting
port	80
server_flow_depth	300
client_flow_depth	300
post_depth	-1
chunk encoding	alert on chunks larger than 500000 bytes
ASCII decoding	on, alert off
utf_8 encoding	on, alert off
multiple slash	on, alert off
directory normalization	on, alert off
webroot	on, alert on
iis backslash	on, alert off
apache whitespace	on, alert off
iis delimiter	on, alert off
non_strict URL parsing	on
max_header_length	0, header length not checked
max_headers	0, number of headers not checked



(excluding the HTTP headers when `extended_response_inspection` is turned on) in a given HTTP session. Only packets payloads starting with 'HTTP' will be considered as the first packet of a server response. If less than flow_depth bytes are in the payload of the HTTP response packets in a given session, the entire payload will be inspected. If more than flow_depth bytes are in the payload of the HTTP response packet in a session only flow_depth bytes of the payload will be inspected for that session. Rules that are meant to inspect data in the payload of the HTTP response packets in a session beyond 65535 bytes will be ineffective unless flow_depth is set to 0. The default value for `server_flow_depth` is 300. Note that the 65535 byte maximum flow_depth applies to stream reassembled packets as well. It is suggested to set the `server_flow_depth` to its maximum value.

⚠NOTE

`server_flow_depth` is the same as the old `flow_depth` option, which will be deprecated in a future release.

10. `client_flow_depth <integer>`

This specifies the amount of raw client request payload to inspect. This value can be set from -1 to 1460. Unlike `server_flow_depth` this value is applied to the first packet of the HTTP request. It is not a session based flow depth. It has a default value of 300. It primarily eliminates Snort from inspecting larger HTTP Cookies that appear at the end of many client request Headers.

A value of -1 causes Snort to ignore all client side traffic for ports defined in "ports." Inversely, a value of 0 causes Snort to inspect all HTTP client side traffic defined in "ports" (note that this will likely slow down IDS performance). Values above 0 tell Snort the number of bytes to inspect in the first packet of the client request. If less than flow_depth bytes are in the TCP payload (HTTP request) of the first packet, the entire payload will be inspected. If more than flow_depth bytes are in the payload of the first packet only flow_depth bytes of the payload will be inspected. Rules that are meant to inspect data in the payload of the first packet of a client request beyond 1460 bytes will be ineffective unless flow_depth is set to 0. Note that the 1460 byte maximum flow_depth applies to stream reassembled packets as well. It is suggested to set the `client_flow_depth` to its maximum value.

11. `post_depth <integer>`

This specifies the amount of data to inspect in a client post message. The value can be set from -1 to 65495. The default value is -1. A value of -1 causes Snort to ignore all the data in the post message. Inversely, a value of 0 causes Snort to inspect all the client post message. This increases the performance by inspecting only specified bytes in the post message.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 63

12. `ascii <yes|no>`

    The `ascii` decode option tells us whether to decode encoded ASCII chars, a.k.a %2f = /, %2e = ., etc. It is normal to see ASCII encoding usage in URLs, so it is recommended that you disable HTTP Inspect alerting for this option.

13. `extended_ascii_uri`

    This option enables the support for extended ASCII codes in the HTTP request URI. This option is turned off by default and is not supported with any of the profiles.

14. `utf_8 <yes|no>`

    The `utf-8` decode option tells HTTP Inspect to decode standard UTF-8 Unicode sequences that are in the URI. This abides by the Unicode standard and only uses % encoding. Apache uses this standard, so for any Apache servers, make sure you have this option turned on. As for alerting, you may be interested in knowing when you have a UTF-8 encoded URI, but this will be prone to false positives as legitimate web clients use this type of encoding. When `utf_8` is enabled, ASCII decoding is also enabled to enforce correct functioning.

15. `u_encode <yes|no>`

    This option emulates the IIS %u encoding scheme. How the %u encoding scheme works is as follows: the encoding scheme is started by a %u followed by 4 characters, like %uxxxx. The xxxx is a hex-encoded value that correlates to an IIS Unicode codepoint. This value can most definitely be ASCII. An ASCII character is encoded like %u002f = /, %u002e = ., etc. If no iis_unicode_map is specified before or after this option, the default codemap is used.

    You should alert on %u encodings, because we are not aware of any legitimate clients that use this encoding. So it is most likely someone trying to be covert.

16. `bare_byte <yes|no>`

    Bare byte encoding is an IIS trick that uses non-ASCII characters as valid values when decoding UTF-8 values. This is not in the HTTP standard, as all non-ASCII values have to be encoded with a %. Bare byte encoding allows the user to emulate an IIS server and interpret non-standard encodings correctly.

    The alert on this decoding should be enabled, because there are no legitimate clients that encode UTF-8 this way since it is non-standard.

17. `base36 <yes|no>`

    This is an option to decode base36 encoded chars. This option is based on info from:

    `http://www.yk.rim.or.jp/~shikap/patch/spp_http_decode.patch.`

    If %u encoding is enabled, this option will not work. You have to use the `base36` option with the `utf_8` option. Don't use the %u option, because base36 won't work. When `base36` is enabled, ASCII encoding is also enabled to enforce correct behavior.

18. `iis_unicode <yes|no>`

    The `iis_unicode` option turns on the Unicode codepoint mapping. If there is no iis_unicode_map option specified with the server config, `iis_unicode` uses the default codemap. The `iis_unicode` option handles the mapping of non-ASCII codepoints that the IIS server accepts and decodes normal UTF-8 requests.

    You should alert on the `iis_unicode option`, because it is seen mainly in attacks and evasion attempts. When `iis_unicode` is enabled, ASCII and UTF-8 decoding are also enabled to enforce correct decoding. To alert on UTF-8 decoding, you must enable also enable `utf_8 yes`.

19. `double_decode <yes|no>`

    The `double_decode` option is once again IIS-specific and emulates IIS functionality. How this works is that IIS does two passes through the request URI, doing decodes in each one. In the first pass, it seems that all types of iis encoding is done: utf-8 unicode, ASCII, bare byte, and %u. In the second pass, the following encodings are done: ASCII, bare byte, and %u. We leave out utf-8 because I think how this works is that the % encoded utf-8 is decoded to the Unicode byte in the first pass, and then UTF-8 is decoded in the second stage. Anyway, this is really complex and adds tons of different encodings for one character. When `double_decode` is enabled, so ASCII is also enabled to enforce correct decoding.

1644470956460125853 19

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 64

03-02-2022-24064301-Hauptpost-0591

20. `non_rfc_char {<byte> [<byte ...>]}`

    This option lets users receive an alert if certain non-RFC chars are used in a request URI. For instance, a user may not want to see null bytes in the request URI and we can alert on that. Please use this option with care, because you could configure it to say, alert on all '/' or something like that. It's flexible, so be careful.

21. `multi_slash <yes|no>`

    This option normalizes multiple slashes in a row, so something like: "foo////////bar" get normalized to "foo/bar." If you want an alert when multiple slashes are seen, then configure with a `yes`; otherwise, use `no`.

22. `iis_backslash <yes|no>`

    Normalizes backslashes to slashes. This is again an IIS emulation. So a request URI of "/foo\bar" gets normalized to "/foo/bar."

23. `directory <yes|no>`

    This option normalizes directory traversals and self-referential directories.

    The directory:

    ```
    /foo/fake\_dir/../bar
    ```

    gets normalized to:

    ```
    /foo/bar
    ```

    The directory:

    ```
    /foo/./bar
    ```

    gets normalized to:

    ```
    /foo/bar
    ```

    If you want to configure an alert, specify `yes`, otherwise, specify `no`. This alert may give false positives, since some web sites refer to files using directory traversals.

24. `apache_whitespace <yes|no>`

    This option deals with the non-RFC standard of using tab for a space delimiter. Apache uses this, so if the emulated web server is Apache, enable this option. Alerts on this option may be interesting, but may also be false positive prone.

25. `iis_delimiter <yes|no>`

    This started out being IIS-specific, but Apache takes this non-standard delimiter was well. Since this is common, we always take this as standard since the most popular web servers accept it. But you can still get an alert on this option.

26. `chunk_length <non-zero positive integer>`

    This option is an anomaly detector for abnormally large chunk sizes. This picks up the Apache chunk encoding exploits, and may also alert on HTTP tunneling that uses chunk encoding.

27. `no_pipeline_req`

    This option turns HTTP pipeline decoding off, and is a performance enhancement if needed. By default, pipeline requests are inspected for attacks, but when this option is enabled, pipeline requests are not decoded and analyzed per HTTP protocol field. It is only inspected with the generic pattern matching.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 65

16444709564601258531 9

28. non_strict

    This option turns on non-strict URI parsing for the broken way in which Apache servers will decode a URI. Only use this option on servers that will accept URIs like this: "get /index.html alsjdfk alsj lj aj la jsj s\n". The non_strict option assumes the URI is between the first and second space even if there is no valid HTTP identifier after the second space.

29. allow_proxy_use

    By specifying this keyword, the user is allowing proxy use on this server. This means that no alert will be generated if the proxy_alert global keyword has been used. If the proxy_alert keyword is not enabled, then this option does nothing. The allow_proxy_use keyword is just a way to suppress unauthorized proxy use for an authorized server.

30. no_alerts

    This option turns off all alerts that are generated by the HTTP Inspect preprocessor module. This has no effect on HTTP rules in the rule set. No argument is specified.

31. oversize_dir_length <non-zero positive integer>

    This option takes a non-zero positive integer as an argument. The argument specifies the max char directory length for URL directory. If a url directory is larger than this argument size, an alert is generated. A good argument value is 300 characters. This should limit the alerts to IDS evasion type attacks, like whisker -i 4.



32. inspect_uri_only

    This is a performance optimization. When enabled, only the URI portion of HTTP requests will be inspected for attacks. As this field usually contains 90-95% of the web attacks, you'll catch most of the attacks. So if you need extra performance, enable this optimization. It's important to note that if this option is used without any uricontent rules, then no inspection will take place. This is obvious since the URI is only inspected with uricontent rules, and if there are none available, then there is nothing to inspect.

    For example, if we have the following rule set:

    ```
    alert tcp any any -> any 80 ( msg:"content"; content: "foo"; )
    ```

    and the we inspect the following URI:

    ```
    get /foo.htm http/1.0\r\n\r\n
    ```

    No alert will be generated when inspect_uri_only is enabled. The inspect_uri_only configuration turns off all forms of detection except uricontent inspection.

33. max_header_length <positive integer up to 65535>

    This option takes an integer as an argument. The integer is the maximum length allowed for an HTTP client request header field. Requests that exceed this length will cause a "Long Header" alert. This alert is off by default. To enable, specify an integer argument to max_header_length of 1 to 65535. Specifying a value of 0 is treated as disabling the alert.

34. webroot <yes|no>

    This option generates an alert when a directory traversal traverses past the web server root directory. This generates much fewer false positives than the directory option, because it doesn't alert on directory traversals that stay within the web server directory structure. It only alerts when the directory traversals go past the web server root directory, which is associated with certain web attacks.

35. tab_uri_delimiter

    This option turns on the use of the tab character (0x09) as a delimiter for a URI. Apache accepts tab as a delimiter; IIS does not. For IIS, a tab in the URI should be treated as any other character. Whether this option is on or not, a tab is treated as whitespace if a space character (0x20) precedes it. No argument is specified.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 66

03-02-2022-24064301-HauptPost-0592

36. `normalize_headers`

    This option turns on normalization for HTTP Header Fields, not including Cookies (using the same configuration parameters as the URI normalization (ie, multi-slash, directory, etc.). It is useful for normalizing Referrer URIs that may appear in the HTTP Header.

37. `normalize_cookies`

    This option turns on normalization for HTTP Cookie Fields (using the same configuration parameters as the URI normalization (ie, multi-slash, directory, etc.). It is useful for normalizing data in HTTP Cookies that may be encoded.

38. `normalize_utf`

    This option turns on normalization of HTTP response bodies where the Content-Type header lists the character set as "utf-16le", "utf-16be", "utf-32le", or "utf-32be". HTTP Inspect will attempt to normalize these back into 8-bit encoding, generating an alert if the extra bytes are non-zero.

39. `max_headers <positive integer up to 1024>`

    This option takes an integer as an argument. The integer is the maximum number of HTTP client request header fields. Requests that contain more HTTP Headers than this value will cause a "Max Header" alert. The alert is off by default. To enable, specify an integer argument to max_headers of 1 to 1024. Specifying a value of 0 is treated as disabling the alert.

40. `http_methods` {*cmd*[*cmd*]} This specifies additional HTTP Request Methods outside of those checked by default within the preprocessor (GET and POST). The list should be enclosed within braces and delimited by spaces, tabs, line feed or carriage return. The config option, braces and methods also needs to be separated by braces.

```
http_methods { PUT CONNECT }
```



⚠NOTE

Please note the maximum length for a method name is 7

Examples

```
preprocessor http_inspect_server: \
    server 10.1.1.1 \
    ports { 80 3128 8080 } \
    server_flow_depth 0 \
    ascii no \
    double_decode yes \
    non_rfc_char { 0x00 } \
    chunk_length 500000 \
    non_strict \
    no_alerts

preprocessor http_inspect_server: \
    server default \
    ports  { 80 3128 }  \
    non_strict \
    non_rfc_char  { 0x00 }  \
    server_flow_depth 300  \
    apache_whitespace yes \
    directory no \
    iis_backslash no \
    u_encode yes \
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 67

```
    ascii no \
    chunk_length 500000 \
    bare_byte yes \
    double_decode yes \
    iis_unicode yes \
    iis_delimiter yes \
    multi_slash no

preprocessor http_inspect_server: \
    server default \
    profile all \
    ports { 80 8080 }
```

2.2.7 SMTP Preprocessor

The SMTP preprocessor is an SMTP decoder for user applications. Given a data buffer, SMTP will decode the buffer and find SMTP commands and responses. It will also mark the command, data header data body sections, and TLS data.

SMTP handles stateless and stateful processing. It saves state between individual packets. However maintaining correct state is dependent on the reassembly of the client side of the stream (ie, a loss of coherent stream data results in a loss of state).



Configuration

SMTP has the usual configuration items, such as `port` and `inspection_type`. Also, SMTP command lines can be normalized to remove extraneous spaces. TLS-encrypted traffic can be ignored, which improves performance. In addition, regular mail data can be ignored for an additional performance boost. Since so few (none in the current snort rule set) exploits are against mail data, this is relatively safe to do and can improve the performance of data inspection.

The configuration options are described below:

1. `ports { <port> [<port>] ... }`

   This specifies on what ports to check for SMTP data. Typically, this will include 25 and possibly 465, for encrypted SMTP.

2. `inspection_type <stateful | stateless>`

   Indicate whether to operate in stateful or stateless mode.

3. `normalize <all | none | cmds>`

   This turns on normalization. Normalization checks for more than one space character after a command. Space characters are defined as space (ASCII 0x20) or tab (ASCII 0x09).

   `all` checks all commands

   `none` turns off normalization for all commands.

   `cmds` just checks commands listed with the `normalize_cmds` parameter.

4. `ignore_data`

   Ignore data section of mail (except for mail headers) when processing rules.

5. `ignore_tls_data`

   Ignore TLS-encrypted data when processing rules.

6. `max_command_line_len <int>`

   Alert if an SMTP command line is longer than this value. Absence of this option or a "0" means never alert on command line length. RFC 2821 recommends 512 as a maximum command line length.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 68

03-02-2022-24064301-Hauptpost-0593

7. `max_header_line_len <int>`

   Alert if an SMTP DATA header line is longer than this value. Absence of this option or a "0" means never alert on data header line length. RFC 2821 recommends 1024 as a maximum data header line length.

8. `max_response_line_len <int>`

   Alert if an SMTP response line is longer than this value. Absence of this option or a "0" means never alert on response line length. RFC 2821 recommends 512 as a maximum response line length.

9. `alt_max_command_line_len <int> { <cmd> [<cmd>] }`

   Overrides `max_command_line_len` for specific commands.

10. `no_alerts`

    Turn off all alerts for this preprocessor.

11. `invalid_cmds { <Space-delimited list of commands> }`

    Alert if this command is sent from client side. Default is an empty list.

12. `valid_cmds { <Space-delimited list of commands> }`

    List of valid commands. We do not alert on commands in this list. Default is an empty list, but preprocessor has this list hard-coded:

    { ATRN AUTH BDAT DATA DEBUG EHLO EMAL ESAM ESND ESOM ETRN EVFY EXPN } { HELO HELP IDENT MAIL NOOP QUIT RCPT RSET SAML SOML SEND ONEX QUEU } { STARTTLS TICK TIME TURN TURNME VERB VRFY X-EXPS X-LINK2STATE } { XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA XTRN XUSR }

13. `alert_unknown_cmds`

    Alert if we don't recognize command. Default is off.

14. `normalize_cmds { <Space-delimited list of commands> }`

    Normalize this list of commands Default is { RCPT VRFY EXPN }.

15. `xlink2state { enable | disable [drop] }`

    Enable/disable xlink2state alert. Drop if alerted. Default is `enable`.

16. `print_cmds`

    List all commands understood by the preprocessor. This not normally printed out with the configuration because it can print so much data.

17. `disabled`

    Disables the SMTP preprocessor in a policy. This is useful when specifying the `max_mime_depth` and `max_mime_mem` in default policy without turning on the SMTP preprocessor.

18. `enable_mime_decoding`

    Enables Base64 decoding of Mime attachments/data. Multiple base64 encoded MIME attachments/data in one packet are pipelined. When stateful inspection is turned on the base64 encoded MIME attachments/data across multiple packets are decoded too. The decoding of base64 encoded attachments/data ends when either the `max_mime_depth` or maximum MIME sessions (calculated using `max_mime_depth` and `max_mime_mem`) is reached or when the encoded data ends. The decoded data is available for detection using the rule option `file_data:mime`. See 3.5.24 rule option for more details.

19. `max_mime_depth <int>`

    Specifies the maximum number of base64 encoded data to decode per SMTP session. The option take values ranging from 5 to 20480 bytes. The default value for this in snort in 1460 bytes.

16447095646012585319

20. max_mime_mem <int>

    This option determines (in bytes) the maximum amount of memory the SMTP preprocessor will use for decoding base64 encode MIME attachments/data. This value can be set from 3276 bytes to 100MB. This option along with max_mime_depth determines the base64 encoded MIME/SMTP sessions that will be decoded at any given instant. The default value for this option is 838860.

    Note: It is suggested to set this value such that the max mime session calculated as follows is atleast 1.

    max mime session = max_mime_mem /(max_mime_depth + max decoded bytes)

    max decoded bytes = (max_mime_depth/4)*3

    Also note that these values for max_mime_mem and max_mime_depth need to be same across all policy. Hence user needs to define it in the default policy with the new keyword disabled (used to disable SMTP preprocessor in a policy)

Example

```
    preprocessor SMTP: \
        ports { 25 } \
        inspection_type stateful \
        normalize cmds \
        normalize_cmds { EXPN VRFY RCPT } \
        ignore_data \
        ignore_tls_data \
        max_command_line_len  512 \
        max_header_line_len   1024 \
        max_response_line_len 512 \
        no_alerts \
        alt_max_command_line_len 300 { RCPT } \
        invalid_cmds { } \
        valid_cmds { } \
        xlink2state { disable } \
        print_cmds

    preprocessor SMTP: \
        max_mime_depth 100 \
        max_mime_mem 4000 \
disabled
```

Default

```
    preprocessor SMTP: \
        ports { 25 } \
        inspection_type stateful \
        normalize cmds \
        normalize_cmds { EXPN VRFY RCPT } \
        alt_max_command_line_len 260 { MAIL } \
        alt_max_command_line_len 300 { RCPT } \
        alt_max_command_line_len 500 { HELP HELO ETRN } \
        alt_max_command_line_len 255 { EXPN VRFY }
```

Note

RCPT TO: and MAIL FROM: are SMTP commands. For the preprocessor configuration, they are referred to as RCPT and MAIL, respectively. Within the code, the preprocessor actually maps RCPT and MAIL to the correct command name.

164447095646012585319

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 70

03-02-2022-24064301-Hauptpost-0594

2.2.8 FTP/Telnet Preprocessor

FTP/Telnet is an improvement to the Telnet decoder and provides stateful inspection capability for both FTP and Telnet data streams. FTP/Telnet will decode the stream, identifying FTP commands and responses and Telnet escape sequences and normalize the fields. FTP/Telnet works on both client requests and server responses.

FTP/Telnet has the capability to handle stateless processing, meaning it only looks for information on a packet-by-packet basis.

The default is to run FTP/Telnet in stateful inspection mode, meaning it looks for information and handles reassembled data correctly.

FTP/Telnet has a very "rich" user configuration, similar to that of HTTP Inspect (See 2.2.6). Users can configure individual FTP servers and clients with a variety of options, which should allow the user to emulate any type of FTP server or FTP Client. Within FTP/Telnet, there are four areas of configuration: Global, Telnet, FTP Client, and FTP Server.



⚠NOTE

Some configuration options have an argument of yes or no. This argument specifies whether the user wants the configuration option to generate a ftptelnet alert or not. The presence of the option indicates the option itself is on, while the yes/no argument applies to the alerting functionality associated with that option.

Global Configuration

The global configuration deals with configuration options that determine the global functioning of FTP/Telnet. The following example gives the generic global configuration format:

Format

```
preprocessor ftp_telnet: \
    global \
    inspection_type stateful \
    encrypted_traffic yes \
    check_encrypted
```

You can only have a single global configuration, you'll get an error if you try otherwise. The FTP/Telnet global configuration must appear before the other three areas of configuration.

Configuration

1. `inspection_type`

   This indicates whether to operate in stateful or stateless mode.

2. `encrypted_traffic <yes|no>`

   This option enables detection and alerting on encrypted Telnet and FTP command channels.

   ⚠NOTE

   When `inspection_type` is in stateless mode, checks for encrypted traffic will occur on every packet, whereas in stateful mode, a particular session will be noted as encrypted and not inspected any further.

3. `check_encrypted`

   Instructs the preprocessor to continue to check an encrypted session for a subsequent command to cease encryption.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 71

Example Global Configuration

```
preprocessor ftp_telnet: \
    global inspection_type stateful encrypted_traffic no
```

Telnet Configuration

The telnet configuration deals with configuration options that determine the functioning of the Telnet portion of the preprocessor. The following example gives the generic telnet configuration format:

Format

```
preprocessor ftp_telnet_protocol: \
    telnet \
    ports { 23 } \
    normalize \
    ayt_attack_thresh 6 \
    detect_anomalies
```



There should only be a single telnet configuration, and subsequent instances will override previously set values.

Configuration

1. `ports {<port> [<port>< ... >]}`

   This is how the user configures which ports to decode as telnet traffic. SSH tunnels cannot be decoded, so adding port 22 will only yield false positives. Typically port 23 will be included.

2. `normalize`

   This option tells the preprocessor to normalize the telnet traffic by eliminating the telnet escape sequences. It functions similarly to its predecessor, the telnet_decode preprocessor. Rules written with 'raw' content options will ignore the normalized buffer that is created when this option is in use.

3. `ayt_attack_thresh < number >`

   This option causes the preprocessor to alert when the number of consecutive telnet Are You There (AYT) commands reaches the number specified. It is only applicable when the mode is stateful.

4. `detect_anomalies`

   In order to support certain options, Telnet supports subnegotiation. Per the Telnet RFC, subnegotiation begins with SB (subnegotiation begin) and must end with an SE (subnegotiation end). However, certain implementations of Telnet servers will ignore the SB without a corresponding SE. This is anomalous behavior which could be an evasion case. Being that FTP uses the Telnet protocol on the control connection, it is also susceptible to this behavior. The `detect_anomalies` option enables alerting on Telnet SB without the corresponding SE.

Example Telnet Configuration

```
preprocessor ftp_telnet_protocol: \
    telnet ports { 23 } normalize ayt_attack_thresh 6
```

FTP Server Configuration

There are two types of FTP server configurations: default and by IP address.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 72

Default This configuration supplies the default server configuration for any FTP server that is not individually configured. Most of your FTP servers will most likely end up using the default configuration.

Example Default FTP Server Configuration

```
preprocessor ftp_telnet_protocol: \
    ftp server default ports { 21 }
```

Refer to 74 for the list of options set in default ftp server configuration.

Configuration by IP Address This format is very similar to "default", the only difference being that specific IPs can be configured.

Example IP specific FTP Server Configuration

```
preprocessor _telnet_protocol: \
    ftp server 10.1.1.1 ports { 21 } ftp_cmds { XPWD XCWD }
```

1644709564601258531 9

FTP Server Configuration Options

1. `ports {<port> [<port><... >]}`

   This is how the user configures which ports to decode as FTP command channel traffic. Typically port 21 will be included.

2. `print_cmds`

   During initialization, this option causes the preprocessor to print the configuration for each of the FTP commands for this server.

3. `ftp_cmds {cmd[cmd]}`

   The preprocessor is configured to alert when it sees an FTP command that is not allowed by the server.

   This option specifies a list of additional commands allowed by this server, outside of the default FTP command set as specified in RFC 959. This may be used to allow the use of the 'X' commands identified in RFC 775, as well as any additional commands as needed.

   For example:

   ```
   ftp_cmds { XPWD XCWD XCUP XMKD XRMD }
   ```

4. `def_max_param_len <number>`

   This specifies the default maximum allowed parameter length for an FTP command. It can be used as a basic buffer overflow detection.

5. `alt_max_param_len <number> {cmd[cmd]}`

   This specifies the maximum allowed parameter length for the specified FTP command(s). It can be used as a more specific buffer overflow detection. For example the USER command – usernames may be no longer than 16 bytes, so the appropriate configuration would be:

   ```
   alt_max_param_len 16 { USER }
   ```

6. `chk_str_fmt {cmd[cmd]}`

   This option causes a check for string format attacks in the specified commands.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 73

7. `cmd_validity cmd < fmt >`

   This option specifies the valid format for parameters of a given command.

   fmt must be enclosed in <>'s and may contain the following:

Value	Description
int	Parameter must be an integer
number	Parameter must be an integer between 1 and 255
char <chars>	Parameter must be a single character, one of <chars>
date <datefmt>	Parameter follows format specified, where: n Number C Character [] optional format enclosed | OR {} choice of options . + - literal
string	Parameter is a string (effectively unrestricted)
host_port	Parameter must be a host/port specified, per RFC 959
long_host_port	Parameter must be a long host port specified, per RFC 1639
extended_host_port	Parameter must be an extended host port specified, per RFC 2428
{}, |	One of choices enclosed within, separated by |
{}, []	One of the choices enclosed within {}, optional value enclosed within []

Examples of the cmd_validity option are shown below. These examples are the default checks, per RFC 959 and others performed by the preprocessor.

```
cmd_validity MODE <char SBC>
cmd_validity STRU <char FRP>
cmd_validity ALLO < int [ char R int ] >
cmd_validity TYPE < { char AE [ char NTC ] | char I | char L [ number ] } >
cmd_validity PORT < host_port >
```

A cmd_validity line can be used to override these defaults and/or add a check for other commands.

```
# This allows additional modes, including mode Z which allows for
# zip-style compression.
cmd_validity MODE < char ASBCZ >

# Allow for a date in the MDTM command.
cmd_validity MDTM < [ date nnnnnnnnnnnnnn[.n[n[n]]] ] string >
```

MDTM is an off case that is worth discussing. While not part of an established standard, certain FTP servers accept MDTM commands that set the modification time on a file. The most common among servers that do, accept a format using YYYYMMDDHHmmss[.uuu]. Some others accept a format using YYYYMMDDHHmmss[+—]TZ format. The example above is for the first case (time format as specified in http://www.ietf.org/internet-drafts/draft-ietf-ftpext-mlst-16.txt)

To check validity for a server that uses the TZ format, use the following:

```
cmd_validity MDTM < [ date nnnnnnnnnnnnnn[{+|-}n[n]] ] string >
```

8. `telnet_cmds <yes|no>`

   This option turns on detection and alerting when telnet escape sequences are seen on the FTP command channel. Injection of telnet escape sequences could be used as an evasion attempt on an FTP command channel.

1644470956460125853 19

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 74

03-02-2022-24064301-Hauptpost-0596

9. ignore_telnet_erase_cmds <yes|no>

   This option allows Snort to ignore telnet escape sequences for erase character (TNC EAC) and erase line (TNC EAL) when normalizing FTP command channel. Some FTP servers do not process those telnet escape sequences.

10. data_chan

    This option causes the rest of snort (rules, other preprocessors) to ignore FTP data channel connections. Using this option means that **NO INSPECTION** other than TCP state will be performed on FTP data transfers. It can be used to improve performance, especially with large file transfers from a trusted source. If your rule set includes virus-type rules, it is recommended that this option not be used.

    Use of the "data_chan" option is deprecated in favor of the "ignore_data_chan" option. "data_chan" will be removed in a future release.

11. ignore_data_chan <yes|no>

    This option causes the rest of Snort (rules, other preprocessors) to ignore FTP data channel connections. Setting this option to "yes" means that **NO INSPECTION** other than TCP state will be performed on FTP data transfers. It can be used to improve performance, especially with large file transfers from a trusted source. If your rule set includes virus-type rules, it is recommended that this option not be used.



FTP Server Base Configuration Options

The base FTP server configuration is as follows. Options specified in the configuration file will modify this set of options. FTP commands are added to the set of allowed commands. The other options will override those in the base configuration.

```
def_max_param_len 100
ftp_cmds { USER PASS ACCT CWD CDUP SMNT
   QUIT REIN TYPE STRU MODE RETR
   STOR STOU APPE ALLO REST RNFR
   RNTO ABOR DELE RMD MKD PWD LIST
            NLST SITE SYST STAT HELP NOOP }
ftp_cmds { AUTH ADAT PROT PBSZ CONF ENC }
ftp_cmds { PORT PASV LPRT LPSV EPRT EPSV }
ftp_cmds { FEAT OPTS }
ftp_cmds { MDTM REST SIZE MLST MLSD }
alt_max_param_len 0 { CDUP QUIT REIN PASV STOU ABOR PWD SYST NOOP }
cmd_validity MODE < char SBC >
cmd_validity STRU < char FRPO [ string ] >
cmd_validity ALLO < int [ char R int ] >
cmd_validity TYPE < { char AE [ char NTC ] | char I | char L [ number ] } >
cmd_validity PORT < host_port >
cmd_validity LPRT < long_host_port >
cmd_validity EPRT < extd_host_port >
cmd_validity EPSV < [ { '1' | '2' | 'ALL' } ] >
```

FTP Client Configuration

Similar to the FTP Server configuration, the FTP client configurations has two types: default, and by IP address.

Default This configuration supplies the default client configuration for any FTP client that is not individually configured. Most of your FTP clients will most likely end up using the default configuration.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 75

Example Default FTP Client Configuration

```
preprocessor ftp_telnet_protocol: \
    ftp client default bounce no max_resp_len 200
```

Configuration by IP Address This format is very similar to "default", the only difference being that specific IPs can be configured.

Example IP specific FTP Client Configuration

```
preprocessor ftp_telnet_protocol: \
    ftp client 10.1.1.1 bounce yes max_resp_len 500
```

FTP Client Configuration Options

1. `max_resp_len <number>`

   This specifies the maximum allowed response length to an FTP command accepted by the client. It can be used as a basic buffer overflow detection.

2. `bounce <yes|no>`

   This option turns on detection and alerting of FTP bounce attacks. An FTP bounce attack occurs when the FTP PORT command is issued and the specified host does not match the host of the client.

3. `bounce_to < CIDR,[port|portlow,porthi] >`

   When the bounce option is turned on, this allows the PORT command to use the IP address (in CIDR format) and port (or inclusive port range) without generating an alert. It can be used to deal with proxied FTP connections where the FTP data channel is different from the client.

   A few examples:

   - Allow bounces to 192.162.1.1 port 20020 – ie, the use of `PORT 192,168,1,1,78,52`.

     ```
     bounce_to { 192.168.1.1,20020 }
     ```

   - Allow bounces to 192.162.1.1 ports 20020 through 20040 – ie, the use of `PORT 192,168,1,1,78,xx`, where xx is 52 through 72 inclusive.

     ```
     bounce_to { 192.168.1.1,20020,20040 }
     ```

   - Allow bounces to 192.162.1.1 port 20020 and 192.168.1.2 port 20030.

     ```
     bounce_to { 192.168.1.1,20020 192.168.1.2,20030 }
     ```

   - Allows bounces to IPv6 address fe8::5 port 59340.

     ⚠NOTE

     IPv6 support must be enabled.

     ```
     bounce_to { fe8::5,59340 }
     ```

4. `telnet_cmds <yes|no>`

   This option turns on detection and alerting when telnet escape sequences are seen on the FTP command channel. Injection of telnet escape sequences could be used as an evasion attempt on an FTP command channel.

5. `ignore_telnet_erase_cmds <yes|no>`

   This option allows Snort to ignore telnet escape sequences for erase character (TNC EAC) and erase line (TNC EAL) when normalizing FTP command channel. Some FTP clients do not process those telnet escape sequences.

1644470956460125853 19

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 76

03-02-2022-24064301-HauptPost-0597

Examples/Default Configuration from snort.conf

```
preprocessor ftp_telnet: \
    global \
    encrypted_traffic yes \
    inspection_type stateful

preprocessor ftp_telnet_protocol:\
    telnet \
    normalize \
    ayt_attack_thresh 200

# This is consistent with the FTP rules as of 18 Sept 2004.
# Set CWD to allow parameter length of 200
# MODE has an additional mode of Z (compressed)
# Check for string formats in USER & PASS commands
# Check MDTM commands that set modification time on the file.

preprocessor ftp_telnet_protocol: \
    ftp server default \
    def_max_param_len 100 \
    alt_max_param_len 200 { CWD } \
    cmd_validity MODE < char ASBCZ > \
    cmd_validity MDTM < [ date nnnnnnnnnnnnnn[.n[n[n]]] ] string > \
    chk_str_fmt { USER PASS RNFR RNTO SITE MKD } \
    telnet_cmds yes \
    ignore_data_chan yes

preprocessor ftp_telnet_protocol: \
    ftp client default \
    max_resp_len 256 \
    bounce yes \
    telnet_cmds yes
```

1644709564601585319

2.2.9 SSH

The SSH preprocessor detects the following exploits: Challenge-Response Buffer Overflow, CRC 32, Secure CRT, and the Protocol Mismatch exploit.

Both Challenge-Response Overflow and CRC 32 attacks occur after the key exchange, and are therefore encrypted. Both attacks involve sending a large payload (20kb+) to the server immediately after the authentication challenge. To detect the attacks, the SSH preprocessor counts the number of bytes transmitted to the server. If those bytes exceed a predefined limit within a predefined number of packets, an alert is generated. Since the Challenge-Response Overflow only effects SSHv2 and CRC 32 only effects SSHv1, the SSH version string exchange is used to distinguish the attacks.

The Secure CRT and protocol mismatch exploits are observable before the key exchange.

Configuration

By default, all alerts are disabled and the preprocessor checks traffic on port 22.

The available configuration options are described below.

1. `server_ports {<port> [<port><...>]}`
   This option specifies which ports the SSH preprocessor should inspect traffic to.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 77

2. max_encrypted_packets < number >

   The number of encrypted packets that Snort will inspect before ignoring a given SSH session. The SSH vulnerabilities that Snort can detect all happen at the very beginning of an SSH session. Once max_encrypted_packets packets have been seen, Snort ignores the session to increase performance. The default is set to 25. This value can be set from 0 to 65535.

3. max_client_bytes < number >

   The number of unanswered bytes allowed to be transferred before alerting on Challenge-Response Overflow or CRC 32. This number must be hit before max_encrypted_packets packets are sent, or else Snort will ignore the traffic. The default is set to 19600. This value can be set from 0 to 65535.

4. max_server_version_len < number >

   The maximum number of bytes allowed in the SSH server version string before alerting on the Secure CRT server version string overflow. The default is set to 80. This value can be set from 0 to 255.

5. autodetect

   Attempt to automatically detect SSH.

6. enable_respoverflow

   Enables checking for the Challenge-Response Overflow exploit.

7. enable_ssh1crc32

   Enables checking for the CRC 32 exploit.

8. enable_srvoverflow

   Enables checking for the Secure CRT exploit.

9. enable_protomismatch

   Enables checking for the Protocol Mismatch exploit.

10. enable_badmsgdir

    Enable alerts for traffic flowing the wrong direction. For instance, if the presumed server generates client traffic, or if a client generates server traffic.

11. enable_paysize

    Enables alerts for invalid payload sizes.

12. enable_recognition

    Enable alerts for non-SSH traffic on SSH ports.

The SSH preprocessor should work by default. After max_encrypted_packets is reached, the preprocessor will stop processing traffic for a given session. If Challenge-Response Overflow or CRC 32 false positive, try increasing the number of required client bytes with max_client_bytes.

Example Configuration from snort.conf

Looks for attacks on SSH server port 22. Alerts at 19600 unacknowledged bytes within 20 encrypted packets for the Challenge-Response Overflow/CRC32 exploits.

```
preprocessor ssh: \
    server_ports { 22 } \
    max_client_bytes 19600 \
    max_encrypted_packets 20 \
    enable_respoverflow \
    enable_ssh1crc32
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 78

16444709564601258531 9

03-02-2022-24064301-Hauptpost-0598

2.2.10 DNS

The DNS preprocessor decodes DNS Responses and can detect the following exploits: DNS Client RData Overflow, Obsolete Record Types, and Experimental Record Types.

DNS looks at DNS Response traffic over UDP and TCP and it requires Stream preprocessor to be enabled for TCP decoding.

Configuration

By default, all alerts are disabled and the preprocessor checks traffic on port 53.

The available configuration options are described below.

1. `ports {<port> [<port><... >]}`
   This option specifies the source ports that the DNS preprocessor should inspect traffic.
2. `enable_obsolete_types`
   Alert on Obsolete (per RFC 1035) Record Types
3. `enable_experimental_types`
   Alert on Experimental (per RFC 1035) Record Types
4. `enable_rdata_overflow`
   Check for DNS Client RData TXT Overflow

The DNS preprocessor does nothing if none of the 3 vulnerabilities it checks for are enabled. It will not operate on TCP sessions picked up midstream, and it will cease operation on a session if it loses state because of missing data (dropped packets).



Examples/Default Configuration from snort.conf

Looks for traffic on DNS server port 53. Check for the DNS Client RData overflow vulnerability. Do not alert on obsolete or experimental RData record types.

```
preprocessor dns: \
    ports { 53 } \
    enable_rdata_overflow
```

2.2.11 SSL/TLS

Encrypted traffic should be ignored by Snort for both performance reasons and to reduce false positives. The SSL Dynamic Preprocessor (SSLPP) decodes SSL and TLS traffic and optionally determines if and when Snort should stop inspection of it.

Typically, SSL is used over port 443 as HTTPS. By enabling the SSLPP to inspect port 443 and enabling the noinspect_encrypted option, only the SSL handshake of each connection will be inspected. Once the traffic is determined to be encrypted, no further inspection of the data on the connection is made.

By default, SSLPP looks for a handshake followed by encrypted traffic traveling to both sides. If one side responds with an indication that something has failed, such as the handshake, the session is not marked as encrypted. Verifying that faultless encrypted traffic is sent from both endpoints ensures two things: the last client-side handshake packet was not crafted to evade Snort, and that the traffic is legitimately encrypted.

In some cases, especially when packets may be missed, the only observed response from one endpoint will be TCP ACKs. Therefore, if a user knows that server-side encrypted data can be trusted to mark the session as encrypted, the user should use the 'trustservers' option, documented below.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 79

Configuration

1. `ports {<port> [<port><...>]}`

   This option specifies which ports SSLPP will inspect traffic on.

   By default, SSLPP watches the following ports:

   - 443 HTTPS
   - 465 SMTPS
   - 563 NNTPS
   - 636 LDAPS
   - 989 FTPS
   - 992 TelnetS
   - 993 IMAPS
   - 994 IRCS
   - 995 POPS

2. `noinspect_encrypted`

   Disable inspection on traffic that is encrypted. Default is off.

3. `trustservers`

   Disables the requirement that application (encrypted) data must be observed on both sides of the session before a session is marked encrypted. Use this option for slightly better performance if you trust that your servers are not compromised. This requires the `noinspect_encrypted` option to be useful. Default is off.

16444709564601258531 9

Examples/Default Configuration from snort.conf

Enables the SSL preprocessor and tells it to disable inspection on encrypted traffic.

```
preprocessor ssl: noinspect_encrypted
```

Rule Options

The following rule options are supported by enabling the `ssl` preprocessor:

```
ssl_version
ssl_state
```

`ssl_version`

The `ssl_version` rule option tracks the version negotiated between the endpoints of the SSL encryption. The list of version identifiers are below, and more than one identifier can be specified, via a comma separated list. Lists of identifiers are OR'ed together.

The option will match if any one of the OR'ed versions are used in the SSL connection. To check for two or more SSL versions in use simultaneously, multiple `ssl_version` rule options should be used.

Syntax

```
ssl_version: <version-list>

version-list = version | version , version-list
version      = ["!"] "sslv2" | "sslv3" | "tls1.0" | "tls1.1" | "tls1.2"
```

Examples

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 80

03-02-2022-24064301-HauptPost-0599

```
ssl_version:sslv3;
ssl_version:tls1.0,tls1.1,tls1.2;
ssl_version:!sslv2;
```

ssl_state

The ssl_state rule option tracks the state of the SSL encryption during the process of hello and key exchange. The list of states are below. More than one state can be specified, via a comma separated list, and are OR'ed together.

The option will match if the connection is currently in any one of the OR'ed states. To ensure the connection has reached each of a set of states, multiple rules using the ssl_state rule option should be used.

Syntax

```
ssl_state: <state-list>

state-list = state | state , state-list
state      = ["!"] "client_hello" | "server_hello" | "client_keyx" | "server_keyx" | "unknown"
```

Examples

```
ssl_state:client_hello;
ssl_state:client_keyx,server_keyx;
ssl_state:!server_hello;
```

1644709564601258531 9

2.2.12 ARP Spoof Preprocessor

The ARP spoof preprocessor decodes ARP packets and detects ARP attacks, unicast ARP requests, and inconsistent Ethernet to IP mapping.

When no arguments are specified to arpspoof, the preprocessor inspects Ethernet addresses and the addresses in the ARP packets. When inconsistency occurs, an alert with GID 112 and SID 2 or 3 is generated.

When "-unicast" is specified as the argument of arpspoof, the preprocessor checks for unicast ARP requests. An alert with GID 112 and SID 1 will be generated if a unicast ARP request is detected.

Specify a pair of IP and hardware address as the argument to arpspoof_detect_host. The host with the IP address should be on the same layer 2 segment as Snort is. Specify one host IP MAC combo per line. The preprocessor will use this list when detecting ARP cache overwrite attacks. Alert SID 4 is used in this case.

Format

```
preprocessor arpspoof[: -unicast]
preprocessor arpspoof_detect_host: ip mac
```

Option	Description
ip	IP address.
mac	The Ethernet address corresponding to the preceding IP.

Example Configuration

The first example configuration does neither unicast detection nor ARP mapping monitoring. The preprocessor merely looks for Ethernet address inconsistencies.

```
preprocessor arpspoof
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 81

The next example configuration does not do unicast detection but monitors ARP mapping for hosts 192.168.40.1 and 192.168.40.2.

```
preprocessor arpspoof
preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00
preprocessor arpspoof_detect_host: 192.168.40.2 f0:0f:00:f0:0f:01
```

The third example configuration has unicast detection enabled.

```
preprocessor arpspoof: -unicast
preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00
preprocessor arpspoof_detect_host: 192.168.40.2 f0:0f:00:f0:0f:01
```

2.2.13 DCE/RPC 2 Preprocessor

The main purpose of the preprocessor is to perform SMB desegmentation and DCE/RPC defragmentation to avoid rule evasion using these techniques. SMB desegmentation is performed for the following commands that can be used to transport DCE/RPC requests and responses: `Write`, `Write Block Raw`, `Write and Close`, `Write AndX`, `Transaction`, `Transaction Secondary`, `Read`, `Read Block Raw` and `Read AndX`. The following transports are supported for DCE/RPC: SMB, TCP, UDP and RPC over HTTP v.1 proxy and server. New rule options have been implemented to improve performance, reduce false positives and reduce the count and complexity of DCE/RPC based rules.



Dependency Requirements

For proper functioning of the preprocessor:

- Stream session tracking must be enabled, i.e. `stream5`. The preprocessor requires a session tracker to keep its data.
- Stream reassembly must be performed for TCP sessions. If it is decided that a session is SMB or DCE/RPC, either through configured ports, servers or autodetecting, the `dcerpc2` preprocessor will enable stream reassembly for that session if necessary.
- IP defragmentation should be enabled, i.e. the `frag3` preprocessor should be enabled and configured.

Target Based

There are enough important differences between Windows and Samba versions that a target based approach has been implemented. Some important differences:

Named pipe instance tracking

A combination of valid login handle or UID, share handle or TID and file/named pipe handle or FID must be used to write data to a named pipe. The binding between these is dependent on OS/software version.

Samba 3.0.22 and earlier

Any valid UID and TID, along with a valid FID can be used to make a request, however, if the TID used in creating the FID is deleted (via a tree disconnect), the FID that was created using this TID becomes invalid, i.e. no more requests can be written to that named pipe instance.

Samba greater than 3.0.22

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 82

Any valid TID, along with a valid FID can be used to make a request. However, only the UID used in opening the named pipe can be used to make a request using the FID handle to the named pipe instance. If the TID used to create the FID is deleted (via a tree disconnect), the FID that was created using this TID becomes invalid, i.e. no more requests can be written to that named pipe instance. If the UID used to create the named pipe instance is deleted (via a `Logoff AndX`), since it is necessary in making a request to the named pipe, the FID becomes invalid.

Windows 2003

Windows XP

Windows Vista

These Windows versions require strict binding between the UID, TID and FID used to make a request to a named pipe instance. Both the UID and TID used to open the named pipe instance must be used when writing data to the same named pipe instance. Therefore, deleting either the UID or TID invalidates the FID.

Windows 2000

Windows 2000 is interesting in that the first request to a named pipe must use the same binding as that of the other Windows versions. However, requests after that follow the same binding as Samba 3.0.22 and earlier, i.e. no binding. It also follows Samba greater than 3.0.22 in that deleting the UID or TID used to create the named pipe instance also invalidates it.



Accepted SMB commands

Samba in particular does not recognize certain commands under an `IPC$` tree.

Samba (all versions)

Under an `IPC$` tree, does not accept:

```
Open
Write And Close
Read
Read Block Raw
Write Block Raw
```

Windows (all versions)

Accepts all of the above commands under an `IPC$` tree.

AndX command chaining

Windows is very strict in what command combinations it allows to be chained. Samba, on the other hand, is very lax and allows some nonsensical combinations, e.g. multiple logins and tree connects (only one place to return handles for these), login/logoff and tree connect/tree disconnect. Ultimately, we don't want to keep track of data that the server won't accept. An evasion possibility would be accepting a fragment in a request that the server won't accept that gets sandwiched between an exploit.

Transaction tracking

The differences between a `Transaction` request and using one of the `Write*` commands to write data to a named pipe are that (1) a `Transaction` performs the operations of a write and a read from the named pipe, whereas in using the `Write*` commands, the client has to explicitly send one of the `Read*` requests to tell the server to send the response and (2) a `Transaction` request is not written to the named pipe until all of the data is received (via potential `Transaction Secondary` requests) whereas with the `Write*` commands, data is written to the named pipe as it is received by the server. Multiple Transaction requests can be made simultaneously to the same named pipe. These requests can also be segmented with `Transaction Secondary` commands. What distinguishes them (when the same named pipe is being written to, i.e. having the same FID) are fields in the SMB header representing a process id (PID) and multiplex id (MID). The PID represents the process this request is a part of. An MID represents different sub-processes within a process (or under a PID). Segments for each "thread" are stored separately and written to the named pipe when all segments are received. It is necessary to track this so as not to munge these requests together (which would be a potential evasion opportunity).

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 83

Windows (all versions)

Uses a combination of PID and MID to define a "thread".

Samba (all versions)

Uses just the MID to define a "thread".

Multiple Bind Requests

A `Bind` request is the first request that must be made in a connection-oriented DCE/RPC session in order to specify the interface/interfaces that one wants to communicate with.

Windows (all versions)

For all of the Windows versions, only one `Bind` can ever be made on a session whether or not it succeeds or fails. Any binding after that must use the `Alter Context` request. If another `Bind` is made, all previous interface bindings are invalidated.

Samba 3.0.20 and earlier

Any amount of `Bind` requests can be made.

Samba later than 3.0.20

Another `Bind` request can be made if the first failed and no interfaces were successfully bound to. If a `Bind` after a successful `Bind` is made, all previous interface bindings are invalidated.



DCE/RPC Fragmented requests - Context ID

Each fragment in a fragmented request carries the context id of the bound interface it wants to make the request to.

Windows (all versions)

The context id that is ultimately used for the request is contained in the first fragment. The context id field in any other fragment can contain any value.

Samba (all versions)

The context id that is ultimately used for the request is contained in the last fragment. The context id field in any other fragment can contain any value.

DCE/RPC Fragmented requests - Operation number

Each fragment in a fragmented request carries an operation number (opnum) which is more or less a handle to a function offered by the interface.

Samba (all versions)

Windows 2000

Windows 2003

Windows XP

The opnum that is ultimately used for the request is contained in the last fragment. The opnum field in any other fragment can contain any value.

Windows Vista

The opnum that is ultimately used for the request is contained in the first fragment. The opnum field in any other fragment can contain any value.

DCE/RPC Stub data byte order

The byte order of the stub data is determined differently for Windows and Samba.

Windows (all versions)

The byte order of the stub data is that which was used in the `Bind` request.

Samba (all versions)

The byte order of the stub data is that which is used in the request carrying the stub data.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 84

Configuration

The dcerpc2 preprocessor has a global configuration and one or more server configurations. The global preprocessor configuration name is dcerpc2 and the server preprocessor configuration name is dcerpc2_server.

Global Configuration

```
preprocessor dcerpc2
```

The global dcerpc2 configuration is required. Only one global dcerpc2 configuration can be specified.

Option syntax

Option	Argument	Required	Default
memcap	<memcap>	NO	memcap 102400
disable_defrag	NONE	NO	OFF
max_frag_len	<max-frag-len>	NO	OFF
events	<events>	NO	OFF
reassemble_threshold	<re-thresh>	NO	OFF
disabled	NONE	NO	OFF

```
memcap          = 1024-4194303 (kilobytes)
max-frag-len    = 1514-65535
events          = pseudo-event | event | '[' event-list ']'
pseudo-event    = "none" | "all"
event-list      = event | event ',' event-list
event           = "memcap" | "smb" | "co" | "cl"
re-thresh       = 0-65535
```

Option explanations

memcap

Specifies the maximum amount of run-time memory that can be allocated. Run-time memory includes any memory allocated after configuration. Default is 100 MB.

disabled

Disables the preprocessor. By default this value is turned off. When the preprocessor is disabled only the memcap option is applied when specified with the configuration.

disable_defrag

Tells the preprocessor not to do DCE/RPC defragmentation. Default is to do defragmentation.

max_frag_len

Specifies the maximum fragment size that will be added to the defragmention module. If a fragment is greater than this size, it is truncated before being added to the defragmentation module. Default is set to -1. The allowed range for this option is 1514 - 65535.

events

Specifies the classes of events to enable. (See Events section for an enumeration and explanation of events.)

memcap

Only one event. If the memcap is reached or exceeded, alert.

smb

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 85

16447095646012585319

Alert on events related to SMB processing.

co

Stands for connection-oriented DCE/RPC. Alert on events related to connection-oriented DCE/RPC processing.

cl

Stands for connectionless DCE/RPC. Alert on events related to connectionless DCE/RPC processing.

reassemble_threshold

Specifies a minimum number of bytes in the DCE/RPC desegmentation and defragmentation buffers before creating a reassembly packet to send to the detection engine. This option is useful in inline mode so as to potentially catch an exploit early before full defragmentation is done. A value of 0 supplied as an argument to this option will, in effect, disable this option. Default is disabled.

Option examples

```
memcap 30000
max_frag_len 16840
events none
events all
events smb
events co
events [co]
events [smb, co]
events [memcap, smb, co, cl]
reassemble_threshold 500
```

Configuration examples

```
preprocessor dcerpc2
preprocessor dcerpc2: memcap 500000
preprocessor dcerpc2: max_frag_len 16840, memcap 300000, events smb
preprocessor dcerpc2: memcap 50000, events [memcap, smb, co, cl], max_frag_len 14440
preprocessor dcerpc2: disable_defrag, events [memcap, smb]
preprocessor dcerpc2: reassemble_threshold 500
```

Default global configuration

```
preprocessor dcerpc2: memcap 102400
```

Server Configuration

```
preprocessor dcerpc2_server
```

The dcerpc2_server configuration is optional. A dcerpc2_server configuration must start with default or net options. The default and net options are mutually exclusive. At most one default configuration can be specified. If no default configuration is specified, default values will be used for the default configuration. Zero or more net configurations can be specified. For any dcerpc2_server configuration, if non-required options are not specified, the defaults will be used. When processing DCE/RPC traffic, the default configuration is used if no net configurations match. If a net configuration matches, it will override the default configuration. A net configuration matches if the packet's server IP address matches an IP address or net specified in the net configuration. The net option supports IPv6 addresses. Note that port and ip variables defined in snort.conf CANNOT be used.

Option syntax

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 86

16444709564601258531 9

03-02-2022-24064301-Hauptpost-0602

Option	Argument	Required	Default
default	NONE	YES	NONE
net	<net>	YES	NONE
policy	<policy>	NO	policy WinXP
detect	<detect>	NO	detect [smb [139,445], tcp 135, udp 135, rpc-over-http-server 593]
autodetect	<detect>	NO	autodetect [tcp 1025:, udp 1025:, rpc-over-http-server 1025:]
no_autodetect_http_proxy_ports	NONE	NO	DISABLED (The preprocessor autodetects on all proxy ports by default)
smb_invalid_shares	<shares>	NO	NONE
smb_max_chain	<max-chain>	NO	smb_max_chain 3

```
net          = ip | '[' ip-list ']'
ip-list      = ip | ip ',' ip-list
ip           = ip-addr | ip-addr '/' prefix | ip4-addr '/' netmask
ip-addr      = ip4-addr | ip6-addr
ip4-addr     = a valid IPv4 address
ip6-addr     = a valid IPv6 address (can be compressed)
prefix       = a valid CIDR
netmask      = a valid netmask
policy       = "Win2000" | "Win2003" | "WinXP" | "WinVista" |
               "Samba" | "Samba-3.0.22" | "Samba-3.0.20"
detect       = "none" | detect-opt | '[' detect-list ']'
detect-list  = detect-opt | detect-opt ',' detect-list
detect-opt   = transport | transport port-item |
               transport '[' port-list ']'
transport    = "smb" | "tcp" | "udp" | "rpc-over-http-proxy" |
               "rpc-over-http-server"
port-list    = port-item | port-item ',' port-list
port-item    = port | port-range
port-range   = ':' port | port ':' | port ':' port
port         = 0-65535
shares       = share | '[' share-list ']'
share-list   = share | share ',' share-list
share        = word | '"' word '"' | '"' var-word '"'
word         = graphical ASCII characters except ',' '"' '}' '[' '$'
var-word     = graphical ASCII characters except ',' '"' '}' '['
max-chain    = 0-255
```

Because the Snort main parser treats '$' as the start of a variable and tries to expand it, shares with '$' must be enclosed quotes.

Option explanations

`default`

Specifies that this configuration is for the default server configuration.

`net`

Specifies that this configuration is an IP or net specific configuration. The configuration will only apply to the IP addresses and nets supplied as an argument.

`policy`

Specifies the target-based policy to use when processing. Default is "WinXP".

`detect`

Specifies the DCE/RPC transport and server ports that should be detected on for the transport. Defaults are ports 139 and 445 for SMB, 135 for TCP and UDP, 593 for RPC over HTTP server and 80 for RPC over HTTP proxy.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 87

16444709564601258531 9

`autodetect`

Specifies the DCE/RPC transport and server ports that the preprocessor should attempt to autodetect on for the transport. The autodetect ports are only queried if no detect transport/ports match the packet. The order in which the preprocessor will attempt to autodetect will be - TCP/UDP, RPC over HTTP server, RPC over HTTP proxy and lastly SMB. Note that most dynamic DCE/RPC ports are above 1024 and ride directly over TCP or UDP. It would be very uncommon to see SMB on anything other than ports 139 and 445. Defaults are 1025-65535 for TCP, UDP and RPC over HTTP server.

`no_autodetect_http_proxy_ports`

By default, the preprocessor will always attempt to autodetect for ports specified in the detect configuration for rpc-over-http-proxy. This is because the proxy is likely a web server and the preprocessor should not look at all web traffic. This option is useful if the RPC over HTTP proxy configured with the detect option is only used to proxy DCE/RPC traffic. Default is to autodetect on RPC over HTTP proxy detect ports.

`smb_invalid_shares`

Specifies SMB shares that the preprocessor should alert on if an attempt is made to connect to them via a `Tree Connect` or `Tree Connect AndX`. Default is empty.

`smb_max_chain`

Specifies the maximum amount of AndX command chaining that is allowed before an alert is generated. Default maximum is 3 chained commands. A value of 0 disables this option. This value can be set from 0 to 255.

Option examples

```
net 192.168.0.10
net 192.168.0.0/24
net [192.168.0.0/24]
net 192.168.0.0/255.255.255.0
net feab:45b3:ab92:8ac4:d322:007f:e5aa:7845
net feab:45b3:ab92:8ac4:d322:007f:e5aa:7845/128
net feab:45b3::/32
net [192.168.0.10, feab:45b3::/32]
net [192.168.0.0/24, feab:45b3:ab92:8ac4:d322:007f:e5aa:7845]
policy Win2000
policy Samba-3.0.22
detect none
detect smb
detect [smb]
detect smb 445
detect [smb 445]
detect smb [139,445]
detect [smb [139,445]]
detect [smb, tcp]
detect [smb 139, tcp [135,2103]]
detect [smb [139,445], tcp 135, udp 135, rpc-over-http-server [593,6002:6004]]
autodetect none
autodetect tcp
autodetect [tcp]
autodetect tcp 2025:
autodetect [tcp 2025:]
autodetect tcp [2025:3001,3003:]
autodetect [tcp [2025:3001,3003:]]
autodetect [tcp, udp]
autodetect [tcp 2025:, udp 2025:]
autodetect [tcp 2025:, udp, rpc-over-http-server [1025:6001,6005:]]
smb_invalid_shares private
smb_invalid_shares "private"
smb_invalid_shares "C$"
smb_invalid_shares [private, "C$"]
smb_invalid_shares ["private", "C$"]
smb_max_chain 1
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 88

16444709564601258531 9

03-02-2022-24064301-Hauptpost-0603

Configuration examples

```
preprocessor dcerpc2_server: \
    default

preprocessor dcerpc2_server: \
    default, policy Win2000

preprocessor dcerpc2_server: \
    default, policy Win2000, detect [smb, tcp], autodetect tcp 1025:, \
    smb_invalid_shares ["C$", "D$", "ADMIN$"]

preprocessor dcerpc2_server: net 10.4.10.0/24, policy Win2000

preprocessor dcerpc2_server: \
    net [10.4.10.0/24,feab:45b3::/126], policy WinVista, smb_max_chain 1

preprocessor dcerpc2_server: \
    net [10.4.10.0/24,feab:45b3::/126], policy WinVista, \
    detect [smb, tcp, rpc-over-http-proxy 8081],
    autodetect [tcp, rpc-over-http-proxy [1025:6001,6005:]], \
    smb_invalid_shares ["C$", "ADMIN$"], no_autodetect_http_proxy_ports

preprocessor dcerpc2_server: \
    net [10.4.11.56,10.4.11.57], policy Samba, detect smb, autodetect none
```

Default server configuration

```
preprocessor dcerpc2_server: default, policy WinXP, \
    detect [smb [139,445], tcp 135, udp 135, rpc-over-http-server 593], \
    autodetect [tcp 1025:, udp 1025:, rpc-over-http-server 1025:], smb_max_chain 3
```

Complete dcerpc2 default configuration

```
preprocessor dcerpc2: memcap 102400

preprocessor dcerpc2_server: \
    default, policy WinXP, \
    detect [smb [139,445], tcp 135, udp 135, rpc-over-http-server 593], \
    autodetect [tcp 1025:, udp 1025:, rpc-over-http-server 1025:], smb_max_chain 3
```

Events

The preprocessor uses GID 133 to register events.

Memcap events

SID	Description
1	If the memory cap is reached and the preprocessor is configured to alert.

SMB events

SID	Description
2	An invalid NetBIOS Session Service type was specified in the header. Valid types are: `Message`, `Request` (only from client), `Positive Response` (only from server), `Negative Response` (only from server), `Retarget Response` (only from server) and `Keep Alive`.
3	An SMB message type was specified in the header. Either a request was made by the server or a response was given by the client.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 89

16444709564601258531 9

4	The SMB id does not equal \xffSMB. Note that since the preprocessor does not yet support SMB2, id of \xfeSMB is turned away before an eventable point is reached.
5	The word count of the command header is invalid. SMB commands have pretty specific word counts and if the preprocessor sees a command with a word count that doesn't jive with that command, the preprocessor will alert.
6	Some commands require a minimum number of bytes after the command header. If a command requires this and the byte count is less than the minimum required byte count for that command, the preprocessor will alert.
7	Some commands, especially the commands from the SMB Core implementation require a data format field that specifies the kind of data that will be coming next. Some commands require a specific format for the data. The preprocessor will alert if the format is not that which is expected for that command.
8	Many SMB commands have a field containing an offset from the beginning of the SMB header to where the data the command is carrying starts. If this offset puts us before data that has already been processed or after the end of payload, the preprocessor will alert.
9	Some SMB commands, such as Transaction, have a field containing the total amount of data to be transmitted. If this field is zero, the preprocessor will alert.
10	The preprocessor will alert if the NetBIOS Session Service length field contains a value less than the size of an SMB header.
11	The preprocessor will alert if the remaining NetBIOS packet length is less than the size of the SMB command header to be decoded.
12	The preprocessor will alert if the remaining NetBIOS packet length is less than the size of the SMB command byte count specified in the command header.
13	The preprocessor will alert if the remaining NetBIOS packet length is less than the size of the SMB command data size specified in the command header.
14	The preprocessor will alert if the total data count specified in the SMB command header is less than the data size specified in the SMB command header. (Total data count must always be greater than or equal to current data size.)
15	The preprocessor will alert if the total amount of data sent in a transaction is greater than the total data count specified in the SMB command header.
16	The preprocessor will alert if the byte count specified in the SMB command header is less than the data size specified in the SMB command. (The byte count must always be greater than or equal to the data size.)
17	Some of the Core Protocol commands (from the initial SMB implementation) require that the byte count be some value greater than the data size exactly. The preprocessor will alert if the byte count minus a predetermined amount based on the SMB command is not equal to the data size.
18	For the Tree Connect command (and not the Tree Connect AndX command), the preprocessor has to queue the requests up and wait for a server response to determine whether or not an IPC share was successfully connected to (which is what the preprocessor is interested in). Unlike the Tree Connect AndX response, there is no indication in the Tree Connect response as to whether the share is IPC or not. There should be under normal circumstances no more than a few pending tree connects at a time and the preprocessor will alert if this number is excessive.
19	After a client is done writing data using the Write* commands, it issues a Read* command to the server to tell it to send a response to the data it has written. In this case the preprocessor is concerned with the server response. The Read* request contains the file id associated with a named pipe instance that the preprocessor will ultimately send the data to. The server response, however, does not contain this file id, so it need to be queued with the request and dequeued with the response. If multiple Read* requests are sent to the server, they are responded to in the order they were sent. There should be under normal circumstances no more than a few pending Read* requests at a time and the preprocessor will alert if this number is excessive.
20	The preprocessor will alert if the number of chained commands in a single request is greater than or equal to the configured amount (default is 3).

16444709564601258531 9

03-02-2022-24064301-Hauptpost-0604

21	With AndX command chaining it is possible to chain multiple `Session Setup AndX` commands within the same request. There is, however, only one place in the SMB header to return a login handle (or Uid). Windows does not allow this behavior, however Samba does. This is anomalous behavior and the preprocessor will alert if it happens.
22	With AndX command chaining it is possible to chain multiple `Tree Connect AndX` commands within the same request. There is, however, only one place in the SMB header to return a tree handle (or Tid). Windows does not allow this behavior, however Samba does. This is anomalous behavior and the preprocessor will alert if it happens.
23	When a `Session Setup AndX` request is sent to the server, the server responds (if the client successfully authenticates) which a user id or login handle. This is used by the client in subsequent requests to indicate that it has authenticated. A `Logoff AndX` request is sent by the client to indicate it wants to end the session and invalidate the login handle. With commands that are chained after a `Session Setup AndX` request, the login handle returned by the server is used for the subsequent chained commands. The combination of a `Session Setup AndX` command with a chained `Logoff AndX` command, essentially logins in and logs off in the same request and is anomalous behavior. The preprocessor will alert if it sees this.
24	A `Tree Connect AndX` command is used to connect to a share. The `Tree Disconnect` command is used to disconnect from that share. The combination of a `Tree Connect AndX` command with a chained `Tree Disconnect` command, essentially connects to a share and disconnects from the same share in the same request and is anomalous behavior. The preprocessor will alert if it sees this.
25	An `Open AndX` or `Nt Create AndX` command is used to open/create a file or named pipe. (The preprocessor is only interested in named pipes as this is where DCE/RPC requests are written to.) The `Close` command is used to close that file or named pipe. The combination of a `Open AndX` or `Nt Create AndX` command with a chained `Close` command, essentially opens and closes the named pipe in the same request and is anomalous behavior. The preprocessor will alert if it sees this.
26	The preprocessor will alert if it sees any of the invalid SMB shares configured. It looks for a `Tree Connect` or `Tree Connect AndX` to the share.

16447095646012585319

Connection-oriented DCE/RPC events

SID	Description
27	The preprocessor will alert if the connection-oriented DCE/RPC major version contained in the header is not equal to 5.
28	The preprocessor will alert if the connection-oriented DCE/RPC minor version contained in the header is not equal to 0.
29	The preprocessor will alert if the connection-oriented DCE/RPC PDU type contained in the header is not a valid PDU type.
30	The preprocessor will alert if the fragment length defined in the header is less than the size of the header.
31	The preprocessor will alert if the remaining fragment length is less than the remaining packet size.
32	The preprocessor will alert if in a `Bind` or `Alter Context` request, there are no context items specified.
33	The preprocessor will alert if in a `Bind` or `Alter Context` request, there are no transfer syntaxes to go with the requested interface.
34	The preprocessor will alert if a non-last fragment is less than the size of the negotiated maximum fragment length. Most evasion techniques try to fragment the data as much as possible and usually each fragment comes well below the negotiated transmit size.
35	The preprocessor will alert if a fragment is larger than the maximum negotiated fragment length.
36	The byte order of the request data is determined by the Bind in connection-oriented DCE/RPC for Windows. It is anomalous behavior to attempt to change the byte order mid-session.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 91

37	The call id for a set of fragments in a fragmented request should stay the same (it is incremented for each complete request). The preprocessor will alert if it changes in a fragment mid-request.
38	The operation number specifies which function the request is calling on the bound interface. If a request is fragmented, this number should stay the same for all fragments. The preprocessor will alert if the opnum changes in a fragment mid-request.
39	The context id is a handle to a interface that was bound to. If a request if fragmented, this number should stay the same for all fragments. The preprocessor will alert if the context id changes in a fragment mid-request.

Connectionless DCE/RPC events

SID	Description
40	The preprocessor will alert if the connectionless DCE/RPC major version is not equal to 4.
41	The preprocessor will alert if the connectionless DCE/RPC PDU type is not a valid PDU type.
42	The preprocessor will alert if the packet data length is less than the size of the connectionless header.
43	The preprocessor will alert if the sequence number uses in a request is the same or less than a previously used sequence number on the session. In testing, wrapping the sequence number space produces strange behavior from the server, so this should be considered anomalous behavior.



Rule Options

New rule options are supported by enabling the `dcerpc2` preprocessor:

```
dce_iface
dce_opnum
dce_stub_data
```

New modifiers to existing `byte_test` and `byte_jump` rule options:

```
byte_test:dce
byte_jump:dce
```

`dce_iface`

For DCE/RPC based rules it has been necessary to set flow-bits based on a client bind to a service to avoid false positives. It is necessary for a client to bind to a service before being able to make a call to it. When a client sends a bind request to the server, it can, however, specify one or more service interfaces to bind to. Each interface is represented by a UUID. Each interface UUID is paired with a unique index (or context id) that future requests can use to reference the service that the client is making a call to. The server will respond with the interface UUIDs it accepts as valid and will allow the client to make requests to those services. When a client makes a request, it will specify the context id so the server knows what service the client is making a request to. Instead of using flow-bits, a rule can simply ask the preprocessor, using this rule option, whether or not the client has bound to a specific interface UUID and whether or not this client request is making a request to it. This can eliminate false positives where more than one service is bound to successfully since the preprocessor can correlate the bind UUID to the context id used in the request. A DCE/RPC request can specify whether numbers are represented as big endian or little endian. The representation of the interface UUID is different depending on the endianness specified in the DCE/RPC previously requiring two rules - one for big endian and

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 92

one for little endian. The preprocessor eliminates the need for two rules by normalizing the UUID. An interface contains a version. Some versions of an interface may not be vulnerable to a certain exploit. Also, a DCE/RPC request can be broken up into 1 or more fragments. Flags (and a field in the connectionless header) are set in the DCE/RPC header to indicate whether the fragment is the first, a middle or the last fragment. Many checks for data in the DCE/RPC request are only relevant if the DCE/RPC request is a first fragment (or full request), since subsequent fragments will contain data deeper into the DCE/RPC request. A rule which is looking for data, say 5 bytes into the request (maybe it's a length field), will be looking at the wrong data on a fragment other than the first, since the beginning of subsequent fragments are already offset some length from the beginning of the request. This can be a source of false positives in fragmented DCE/RPC traffic. By default it is reasonable to only evaluate if the request is a first fragment (or full request). However, if the `any_frag` option is used to specify evaluating on all fragments.

Syntax

```
dce_iface:<uuid>[, <operator><version>][, any_frag];

uuid       = hexlong '-' hexshort '-' hexshort '-' 2hexbyte '-' 6hexbyte
hexlong    = 4hexbyte
hexshort   = 2hexbyte
hexbyte    = 2HEXDIGIT
operator   = '<' | '>' | '=' | '!'
version    = 0-65535
```

Examples

```
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188;
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188, <2;
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188, any_frag;
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188, =1, any_frag;
```

This option is used to specify an interface UUID. Optional arguments are an interface version and operator to specify that the version be less than ('<'), greater than ('>'), equal to ('=') or not equal to ('!') the version specified. Also, by default the rule will only be evaluated for a first fragment (or full request, i.e. not a fragment) since most rules are written to start at the beginning of a request. The `any_frag` argument says to evaluate for middle and last fragments as well. This option requires tracking client `Bind` and `Alter Context` requests as well as server `Bind Ack` and `Alter Context` responses for connection-oriented DCE/RPC in the preprocessor. For each `Bind` and `Alter Context` request, the client specifies a list of interface UUIDs along with a handle (or context id) for each interface UUID that will be used during the DCE/RPC session to reference the interface. The server response indicates which interfaces it will allow the client to make requests to - it either accepts or rejects the client's wish to bind to a certain interface. This tracking is required so that when a request is processed, the context id used in the request can be correlated with the interface UUID it is a handle for.

`hexlong` and `hexshort` will be specified and interpreted to be in big endian order (this is usually the default way an interface UUID will be seen and represented). As an example, the following Messenger interface UUID as taken off the wire from a little endian `Bind` request:

```
|f8 91 7b 5a 00 ff d0 11 a9 b2 00 c0 4f b6 e6 fc|
```

must be written as:

```
5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc
```

The same UUID taken off the wire from a big endian `Bind` request:

```
|5a 7b 91 f8 ff 00 11 d0 a9 b2 00 c0 4f b6 e6 fc|
```

must be written the same way:

```
5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 93

16447095646012585319