Nuix Adaptive Security

Nuix Adaptive Security 2.16.0

Rule Language Reference Guide

December 2022

nuix

Version 2. March 2023

Content

Introduction

Welcome to the Nuix Adaptive Security Rule Language Reference Guide. The purpose of this guide is to allow users to become familiar with the Filter Engine Rules used in the Nuix Adaptive Security application.

Nuix Adaptive Security

Visibility into the security of your environment is crucial to your organization's success. Nuix Adaptive Security can help you answer questions about your organization, such as:

- Is my organization compromised?
- Has someone taken critical data out of my organization?
- How was someone able to access our environment?
- Is something about to happen?

When you don't have visibility, it leaves your organization in a precarious position, at a decision-making disadvantage, and open to greater risk.

Nuix Adaptive Security delivers a proactive approach that provides the kind of *visibility, adaptability,* and *control* that is missing with traditional endpoint products. By leveraging endpoint analytics, Nuix Adaptive Security reduces the time it takes to detect an impending or ongoing attack, accelerates recovery time, easily adapts to changing environments, regulations, and attack vectors, and ultimately, stops incidents in their tracks.

Nuix Adaptive Security has perfected the art of continuous monitoring and response to isolate the important (and often small) signals from the noise and identify when behaviors exhibit uncharacteristic patterns. Nuix Adaptive Security relies on two fundamental and unique elements to drive the *protect-detect-response-remediate* process:

- The Digital Behavior Recorder (TM) continuously monitors and records key digital behaviors.
- The patent-pending logic engine provides customizable logic on the endpoint, enabling it to recognize and act on threats in real-time.

About this guide

This guide explains how to protect a computer network by creating custom rules to effectively respond to events generated by the endpoint agents. Specifically, this guide discusses the Event Filter Language used within the Nuix Adaptive Security application.

Intended audience

This guide is for all users of Nuix Adaptive Security software.

Additional information

Refer to the following documentation for additional information:

- *Nuix Adaptive Security Installation Guide, Version 2.16.0*
- *Nuix Adaptive Security Quick Start Guide, Version 2.16.0*
- *Nuix Adaptive Security User Guide, Version 2.16.0*
- *Nuix Adaptive Security Administration Guide, Version 2.16.0*
- *Nuix Adaptive Security Release Notes, Version 2.16.0*

Filter engine rule language

The Nuix Adaptive Security endpoint agent supports a rule language called Event Filter Language (EFL).

Using this language, write rules to perform various actions in response to events generated by the endpoint agent. The EFL provides access to the following actions:

- Suppress events from writing to the digital behavior recorder (DBR).

- Forward events to the server immediately.
- Send alert messages to the server regarding the event.
- Block new process creation.
- Isolate the endpoint from the network.
- Trigger a collection of screenshots on the endpoint.
- Set user-defined variables to the track state.

Real time execution of rules occurs on the endpoint in response to each event generated by the endpoint agent. Each rule specifies values to match against attributes contained in the event data. In addition, the endpoint agent tracks certain state data for each process on the system, which you can reference in rules or use for the setting of user-defined variables at rule run time. This facilitates state tracking across multiple events over time to build more complex rule scenarios.

Rule syntax

Every rule set consists of at least one rule group.

Rule groups act as a namespace to partition variable declarations and also allow you to logically organize rules that can contain one or more rules. The simplest rule group and rule look like this.

```
rulegroup <name>
{
    <action> when <expression>;
    ...
};
```

Each rule specifies an action (alert, block, suppress, forward, isolate, set) followed by an expression. A simple rule example is a rule that blocks execution based on the file name.

```
rulegroup BlockCalc
{
    block when strstr(process.path, "calc.exe", false);
};
```

The stristr function performs a case-insensitive search for the string "calc.exe" inside the attribute value process.path. Rules are made more complex by linking multiple attribute comparisons with the logical conjunctions AND and OR. All language keywords are case insensitive.

The following rule sends alerts to the server in response to remote logins to the endpoint using an account name containing "admin" or "root".

```
rulegroup AdminLogin
{
    alert when session.event == SESSION_LOGON and
               session.type == SESSION_TYPE_REMOTE and
               strcmp(session.username,"admin",false);
};
```

Rules can be annotated with metadata. For example, everything to the right of a number sign (“#”) is treated as a comment and ignored by the compiler.

```
#
# Rules relating to admin logins
#
rulegroup AdminLogin
{

    # This rule generates an alert when a remote login
    # occurs using the username "admin".
    alert when session.event == SESSION_LOGON and
               session.type == SESSION_TYPE_REMOTE and
               strcmp(session.username,"admin",false);
};
```


Rule annotation helps to document rule behavior and is used by the Nuix Adaptive Security application when displaying data for rule matches. For more information about rule annotation, see Rule Annotation.

Rule actions

The following table describes the supported rule actions.

Action name	Behavior
alert	Send an alert message to the Nuix Adaptive Security server with this event data.
block	Block process creation. This form of the block action can only be used with process events.
block (uint32 blockMode)	Block the use of a removable media device on the endpoint. This form of the block action can only be used with media events. The blockMode parameter accepts the following constants: BLOCK_MOUNT. BLOCK_MOUNT automatically ejects a removable media device as soon as it is connected to the eject.
capturerss(pid processId, uint32 backseconds, uint32 forwardseconds) capturerss(uint32 sessionId, uint32 backseconds, uint32 forwardseconds)	Send rolling screenshots to the server for the logon session identified by sessionId or for the logon session containing the process identified by processId. Returns backseconds worth of screenshots from the rolling screenshot log prior to the rule match and forwardseconds worth of screenshots after the rule match.
CollectFile(string filepath)	Initiate a collect file task for the file specified by filepath. The file will be uploaded to the Adaptive Server. For more information see, **File content inspection and collection** in the *Nuix Adaptive Security User Guide*.
forward	Forward this event to the Nuix Adaptive Security server.
Inspectfile(string filepath, string regexes[], string label) Inspectfile(string filepath, string regexes[], string label, uint32 `contextLen, uint32 contentLen`)	Initiate a file inspection task for the file specified by filepath. The regexes parameter contains a list of regular expressions to search for. The label parameter is the text string that will appear in the resulting InspectFileMatch event to relate the match event back to a particular InspectFile rule. By default, the 25 characters before and after a regular expression match will be returned in the InspectFileMatch event. The contextLen parameter can be used to change that value. By default, the first 1000 characters of the file are returned in the InspectFileMatch event. That value can be changed using the contentLen parameter. For more information see, **File content inspection and collection** in the *Nuix Adaptive Security User Guide*.
isolate	Enable network isolation on the endpoint.
killprocess	Kill the process associated with this event.
memscan (uint32 scanMask, pid processId)	Initiate a memory scan task on the endpoint. The scanMask parameter accepts any combination of the following constants: MEMSCAN_INJECTED_MODULES, MEMSCAN_IMAGE_PATCHES. MEMSCAN_INJECTED_MODULES scans only for injected modules, while MEMSCAN_IMAGE_PATCHES scans only for code patches. These can be combined with a bitwise OR to perform both types of scans.

Action name	Behavior
	The processId parameter can be a specific process ID to scan. It can also specify the constant ALL_PROCESSES, which performs the scan against every active process on the system. Discovery of injected modules during the scan results in the generation of a Memory Scan Injection Event. Discovery of patches during the scan results in the generation of a Memory Scan Patch Event.
screenshot (pid processId , uint32 interval, uint32 timespan) screenshot (uint32 sessionId, uint32 interval, uint32 timespan)	Initiate a screenshot task on the endpoint. A screenshot will be taken of the desktop for the logon session identified by sessionId or for the session containing the process with the process identifier (processId) every (interval) seconds for the next (timespan) seconds.
screenshot (pid processId, uint32 timespan) screenshot (uint32 sessionId, uint32 timespan)	Initiate a screenshot task on the endpoint. A screenshot will be taken of the desktop for the logon session identified by sessionId or for the session containing the process with the process identifier (processId) every 250ms for the next (timespan) seconds.
set	Set the contents of a user-defined variable in the state engine.
startrss(pid processId, uint32 seconds) startrss(uint32 sessionId, uint32 seconds)	Start rolling screenshot collection on the endpoint for the logon session identified by sessionId or for the logon session containing the process identified by processId. The seconds parameter indicates the number of seconds worth of screenshots (collected at four per second) that will be maintained in the circular screenshot log. This is the maximum number of seconds worth of back screenshots that will be available for retrieval with the capturers action.
stoprss(pid processId) stoprss(uint32 sessionId)	Stop rolling screenshot collection on the endpoint for the logon session identified by sessionId or for the logon session containing the process identified by processId.
suppress	Suppress writing of this event to the endpoint DBR.

Operators

The following table describes the operators supported by the language.

Operator	Behavior
==, !=	Perform equal/not equal comparison for the following types: • uint32 • uint64 • double • sha256hash • md5hash • ipaddr • bool • string For string data types, the comparison is performed in a case-insensitive manner.

Operator	Behavior
	For ipaddr data types, the comparison can be performed against two host addresses, two network addresses, or a host address and a network address. In the last case, a match occurs if the host address falls within the range of the network address.
contains	Contains operator. Only valid for attributes that use set types. Performs a complete string match against items in the list.
&, |, ~, ^	Bitwise AND, OR, NOT, and XOR. Only valid for integer types (uint32, uint64).
&&, ||, !	Logical AND, OR, and NOT. Valid for numeric and Boolean types (uint32, uint64, bool).
<, >	Less than/greater than comparison for attributes that use numeric types (uint32, uint64).
=, +=, -=	Assignment, assignment by sum, assignment by difference (uint32, uint64).
context[pid, symbolname]	Returns the value of a state engine variable for the process context given in pid.

The following table describes the order of precedence and associativity of the operators.

Precedence	Operator	Description	Associativity
1	context[pid, symbolname]	Allows referencing of state engine values from process contexts other than the one associated with the event being processed.	Left to right
2	()	Function call	Left to right
3	! ~	Logical NOT and bitwise NOT	Right to left
4	<	Relational less than	Left to right
	>	Relational greater than	Left to right
5	== !=	Relational equals and not equals	Left to right
6	&	Bitwise AND	Left to right
7	^	Bitwise XOR	Left to right
8	|	Bitwise OR	Left to right
9	&&	Logical AND	Left to right
	||	Logical OR	Left to right
10	=	Simple assignment	Right to left
	+= -=	Assignment by sum and difference	Right to left

Built-in functions

The following table describes the precedence and associativity of the operators.

Operator	Behavior
bool strstr(string str1, string str2, bool bCaseSensitive) bool strstr(string str1, string[] str2, bool bCaseSensitive)	Performs a search for occurrences of str2 in str1. Returns **TRUE** if found, **FALSE** otherwise. If bCaseSensitive is **TRUE**, the comparison is case-sensitive. If a set is supplied for the second parameter, the set is iterated and each item is used in the strstr operation until the first match occurs or until the set has been fully iterated.
bool strstr(string str1, string str2)	Performs a case-sensitive search for occurrences of str2 in str1. Returns **TRUE** if found, **FALSE** otherwise. This is a legacy function superseded by strstr above.
bool stristr(string str1, string str2)	Performs a case-insensitive search for occurrences of str2 in str1. Returns **TRUE** if found, **FALSE** otherwise. This is a legacy function superseded by strstr above.
bool findword(string str1, string str2, bool bCaseSensitive) bool findword(string str1, string[] str2, bool bCaseSensitive)	Performs a search for the complete word str2 in str1. Returns **TRUE** only if str2 is found as a bounded word in str1 and not as a substring of some other word in str1. If a set is supplied for the second parameter, the set is iterated and each item is used in the findword operation until the first match occurs or until the set has been fully iterated. For more information on the findword function, see Matching on Word Boundaries with .
bool startswith(string str1, string prefix, bool bCaseSensitive) bool startswith(string str1, string[] prefix, bool bCaseSensitive)	Returns **TRUE** if the str1 starts with the prefix. If bCaseSensitive is **TRUE**, the comparison is case-sensitive. If a set is supplied for the second parameter, the set is iterated and each item is used in the startswith operation until the first match occurs or until the set has been fully iterated.
bool endswith(string str1, string suffix, bool bCaseSensitive) bool endswith(string str1, string[] suffix, bool bCaseSensitive)	Returns **TRUE** if the string ends with the suffix. If bCaseSensitive is **TRUE**, the comparison is case sensitive. If a set is supplied for the second parameter, the set is iterated and each item is used in the endswith operation until the first match occurs or until the set has been fully iterated.
bool strcmp(string str1, string str2, bool bCaseSensitive)	Performs a comparison of str1 and str2. Returns **TRUE** if the strings match, **FALSE** otherwise. If bCaseSensitive is **TRUE**, the comparison is case-sensitive.

Operator	Behavior
bool strcmp(string str1, string str2)	Performs a case-sensitive comparison of str1 and str2. Returns **TRUE** if the strings match, **FALSE** otherwise. This is a legacy function superseded by strcmp above.
bool stricmp(string str1, string str2)	Performs a case-insensitive comparison of str1 and str2. Returns **TRUE** if the strings match, **FALSE** otherwise. This is a legacy function superseded by strcmp above.
uint32 strlen(string str)	Returns the character count for the supplied string parameter.
pid getppid(uint32 pid)	Returns the parent process ID for the process specified by the pid parameter.
setinsert(uint32 s[], uint32 n) setinsert(uint64 s[], uint64 n) setinsert(double s[], double n) setinsert(bool s[], bool n) setinsert(ipaddr s[], ipaddr address) setinsert(md5digest s[], md5digest digest) setinsert(sha256digest s[], sha256digest digest)	Inserts an item into a set of the same type. For more information about set variables, see Using Set Variables.
setremove(uint32 s[], uint32 n) setremove(uint64 s[], uint64 n) setremove(double s[], double n) setremove(bool s[], bool b) setremove(ipaddr s[], ipaddr address) setremove(md5digest s[], md5digest digest) setremove(sha256digest s[], sha256digest digest)	Removes an item from a set of the same type. For more information about set variables, see Using Set Variables.
bool setstrstr(string haystack, string needles [], bool bCaseSensitive)	Iterates the needles string set and performs a search for occurrences of any element of the set in the string haystack. The function returns **TRUE** upon the first matching occurrence and returns **FALSE** otherwise. If bCaseSensitive is **TRUE**, the comparison is case-sensitive. This function is superseded by the strstr function which now can operate on string sets.
bool setipcmp(ipaddr ip , ipaddr ipAddrSet[])	Iterates ipAddrSet and compares each item in the set against ip.
string getregistrysubkey(string regEventPath)	Normalizes a registry path returned in event data and returns the result. Strips the hive name from the beginning of a kernel registry path if present. Strips User SIDs from user hive paths. Examples: \REGISTRY\MACHINE\xyz → \xyz \REGISTRY\USER\HKEY_USERS\S-1-5-21-2177698632-2771012605-3807011837-1001\xyz → \xyz

Operator	Behavior
string upper(string s)	Apply uppercase to a string in place.
string lower(string s)	Apply lowercase to a string in place.
string getbasename(string filePath)	Returns the base name of the supplied file path, for example, the portion after the last backslash. The whole string is returned if there is no backslash in the base name of the supplied file path. Examples: c:\windows\system32\etc\drivers\hosts → hosts hosts → hosts
bool iswhitelisted(md5digest digest)	Determines whether the supplied hash is on one of the endpoint agent's whitelists. This function returns **TRUE** when the supplied hash: • Is on a whitelist and whitelisting is enabled on the endpoint agent. • Is on a whitelist and whitelisting is disabled on the endpoint agent. • Is not on a whitelist and whitelisting is disabled on the endpoint agent. This function returns **FALSE** when the supplied hash: • Is not on a whitelist and whitelisting is enabled.
threshold threshold_create (uint32 eventLimit , uint32 timePeriodMs)	Initializes a threshold variable. eventLimit is the number of events that must be exceeded within timePeriodMs (milliseconds). For more information about threshold variables, see Using Threshold Variables.
bool threshold_increment (threshold t, uint64 timestamp, uint32 increment) bool threshold_increment (threshold t, uint64 timestamp, uint64 increment)	Increments the event count for a threshold variable. Returns **TRUE** if the threshold has been exceeded, **FALSE** otherwise. For more information about threshold variables, see Using Threshold Variables.
bool timematch (uint64 timestamp, uint32 day_mask, time_range range)	Performs time range matching. For more information about time range matching, see Using Time Range Matching.
uint64 getcurrenteventtimestamp()	Returns the timestamp of the event being processed in the rule.
bool dnsmatch (ipaddr ipAddress, string dnsName) bool dnsmatch (ipaddr ipAddress, string dnsName[])	Checks for the mapping of an IP address to a DNS name in the namespace cache maintained by the endpoint agent. For usage examples, see Namespace Event Caching.
uint64 timediff(uint64 timestamp1, uint64 timestamp2)	Calculate the absolute difference between two event timestamps in seconds.

Operator	Behavior
uint64 runthreadheuristics(uint32 threadId)	Examines a running thread on the system and returns a bit mask of interesting features. Currently the only bit supported in the bitmask is TH_PRIVATE_MEMORY, which indicates that the start address of the thread resides in a private memory allocation (i.e. a region of memory not backed by a file on disk non-file back memory. This can be indicative of the use of malicious code injection techniques.
uint64 getrulematchinterval()	Returns the number of seconds since the currently executing rule last matched. The value returned is calculated by subtracting the last time the rule matched from the current event timestamp. The last rule match time starts with a value of zero each time the agent is started. The last rule match time is reset to zero for all rules whenever any rule updates are pushed to the agent. This can be used to limit how frequently a rule will match. See Limiting the Frequency of Rule Match for more information.
uint64 getrulematchcount()	Returns the number of times the currently executing rule has matched. The rule match count for each rule starts at zero when the agent is started and is incremented by 1 each time the rule matched. The count is incremented after the match logic is matched. The match count is reset to zero for all rules whenever the agent restarts or whenever rules any rule changes to any rules are made. This can be used to limit how frequently a rule will match. See Limiting the Frequency of Rule Match for more information.

Rule evaluation order

The endpoint agent evaluates filter rules in response to each event generated by the endpoint agent.

The following list is the order of rule evaluation.

1. Any rule that specifies the action of *set* runs.
2. For process creation events, block rules only execute in response to process creation events and evaluate until the first match. If an event occurs, that event prevents process creation, and an alert is sent to the server indicating that the process is blocked from starting. The alert includes information about the process and the rule responsible for blocking the process.

The following table describes the evaluation that occurs for all event types.

Action type	Evaluation
suppress	Evaluates until the first match. In response to matching a suppression rule, the endpoint declines to write the matching event data to the endpoint DVR.

Action type	Evaluation
alert	Evaluates until every alert rule has evaluated. As a result, multiple alert rules fire for a single event. For each matching alert rule, the endpoint sends an alert message to the server with a copy of the event triggering the match and a copy of the matching rule.
forward	Evaluates until the first match. In response to a matching forward rule, the endpoint forwards a copy of that event to the server.
isolate	Evaluates until the first match. In response to a matching isolate rule, the network isolation firewall rules configured on the endpoint are enabled.
killprocess	Evaluates until the first match. In response to a matching kill process rule, the associated process terminates. The endpoint never terminates the endpoint agent process in response to a matching rule.
screenshot	Evaluates until every screenshot rule has been evaluated.
memscan	Evaluates until every memscan rule has been evaluated.

Declare and set variables

The filter language supports the declaration and assignment of custom variables.

Data types

The following table describes the built-in scalar data types that are supported by the language.

Type name	Description	Value representation	Coercion rules
uint32	unsigned 32-bit value	3	uint32 → uint64 uint32 → double uint32 → bool
uint64	unsigned 64-bit value	5	uint64 → bool
double	double value	4.0	double → bool
bool	Boolean value	TRUE FALSE	None
string	A string value	"a string"	None

ipaddr	IP version 4 or 6 address	IP address constants (IPv4 and IPv6) are represented in code using a quoted string with the ip() casting operator.
		`ip("192.168.1.0/24")`
		`ip("192.168.1.3")`
		`ip("fe80::ed12:dd3a:f496:0000/16")`
		`ip("fe80::ed12:dd3a:f496:b829")`
			None

sha256digest	A SHA256 hash digest	SHA256 constants are represented in code using a quoted string with the sha256 casting operator. `sha256("e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855")`	None

Type name	Description	Value representation	Coercion rules
md5digest	An MD5 hash digest	MD5 constants are represented in code using a quoted string with the MD5 casting operator. `md5("d41d8cd98f00b204e9800998ecf8427e")`	None
pid	A process identifier	This type cannot be assigned to.	pid → uint32

The following table describes various complex data types that are supported by the language.

Type name	Description	Value representation
threshold	Threshold variables provide a mechanism for detecting the occurrence of a fixed number of events within a specified time period. For more information about how to use threshold variables, see Using Threshold Variables.	None
set. Identified by "[]" following the variable name of scalar variable definition.	Sets are collections of scalar data types. Scalar values are inserted and removed from a set. Sets are also tested for the inclusion of a scalar value. For more information about how to use set variables, see Using Threshold Variables.	None
time_range	Represents a range of time with a starting hour and minute and ending hour and minute. For more information about using time_range variables, see Using Time Range Matching.	None

Variable initialization

Variables are declared within rule groups and initialized when declared. The right side of a variable initialization statement references only constants or other user-defined variables, and event attributes.

Example:

```
rulegroup Privileged_Logons
{
    # declare a set used to hold constant data
    global string priv_accounts[3] = { "admin", "root", "oper"};

    #
    # Alert on remote logins into privileged accounts
    #
    alert when session.event == SESSION_LOGON and
               session.type == SESSION_TYPE_REMOTE and
               priv_accounts contains session.username;

};

rulegroup Process_Injection
{
    uint32 mask = (PROCESS_VM_OPERATION | PROCESS_VM_WRITE);

    #
    # Some unsigned process is injecting into a non-child process
    #
    alert when (object.targetobjecttype == OBJECT_TYPE_PROCESS ) and
                  (
              ( object.accessmask & mask) ) == mask) and
              ( object.sourcepid != getppid ( object.targetpid ) )and
              ( ( curproc.exe.sig == SIG_STATUS_INVALID ) or ( curproc.exe.sig ==
SIG_STATUS_NO_SIG )
                  );
};
```


The following table describes the default initialization values for various variable types.

Type name	Value	Description
uint32	0	
uint64	0	
double	0.0	
bool	false	
string	""	empty string
ipaddr	ip(0.0.0.0)	IPv4 host address
sha256digest	sha256("e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855")	SHA256 hash of empty string
md5digest	md5("d41d8cd98f00b204e9800998ecf8427e")	MD5 hash of empty string

Some variable types, including time_range and threshold, must be initialized when declared. There is no default initialization for variables of those types, and it is a compile error not to initialize the variables at declare time.

Variable scope

A variable's scope is specified when the variable is declared.

The following two variable types are supported:

- Per-process
- Global

For per-process variables, a separate copy of the variable value is maintained for each process in the state engine. Values written to that variable by process X see only that process. The use of the *local* specifier marks a variable as local. For global variables, a single copy of the variable value is maintained for all processes in the state engine. An update to a global variable by process X is seen by all other processes. The use of the *global* specifier marks a variable as global. Local is the default if no scope is specified.

Examples:

```
# Global string set variable to keep track of the most recent 128 name queries
performed on the system.
global string gNameQuerySet[128] = {};

# Local bool variable to track whether the process has attempted injection into
other processes.
local bool injectionAttempted = FALSE;

# Global variable to track last 8 removable media device inserted.
global string gRemovableMediaSerialIds[8] = {};
```

An important difference between local and global variables pertains to their accessibility during rule execution. Events like process events or file events are associated with a specific process, for example, the process that is starting or exiting, or the process performing the file operation. These events provide a process context. However, certain event types are system activities that are not associated with a specific process. These events include logon events, print events, and media events, and the events provide no process context. Because local variables are per-process, local variables can only be accessed within a rule that is processing an event that supplies a process context. When processing event types that do not include a process context, the rule will not be able to read or write local variables. Global variables are accessed when processing both events that provide a process context and those that do not. The compiler produces a warning (for more information about this topic, see Compilation Warnings) when compiling a rule that accesses a local variable but does not include a reference to an event type with a process context.

Variable storage class

The storage class of a variable is specified when the variable is declared.

Temporary

Variables declared as temporary are reinitialized each time a new event is processed by the filter engine. Temporary variables are useful for signaling a condition during the processing of an event and are indicated by using the *temp* keyword in the declaration.

```
temp local bInjectionEvent = FALSE;
```

Persistent

Variables declared as persistent retain their values when a new rule set is loaded assuming a variable of the same name and same type exists in the new ruleset. If a variable is used to track some interesting process state over time, you may wish to retain the value of that variable even when a new rule set is loaded onto the endpoint agent. An example might be tracking the list of files that a process has written to. Persistent variables are indicated by using the *persist* keyword in the declaration.

```
persist local fileWriteList[128] = {};
```

Non-persistent

This is the default behavior if no lifetime is explicitly specified when the variable is declared.

The use of the *nonpersist* specifier indicates that the variable is not persistent. The variable's value does not persist across rule set reloads. If *nonpersist* is specified or no specifier is provided with the variable declaration, then the variable is treated as *nonpersist*.

Variable access

A rule set consists of one or more rule groups. Rule groups allow rules to be grouped logically and also serve as namespaces for variable declaration. By default, when a variable is declared in a rule group, the variable's name is referenced in other rule groups, allowing rules in other rule groups to read or write the variable. The use of the *private* specifier when declaring a variable indicates that other rule groups are not able to see that variable declaration. The use of the *public* specifier indicates that a variable is visible and readable or writable by rules in other rule groups. If neither specifier is provided, the variable is treated as public.

Set variables at rule run time

Variables are set at run time using the set action. The following example keeps a count of the total number of files written by a process and keeps a set of the last 128 file names written by a process.

Example:

```
rulegroup filewrites{
    local fileWriteList[128] = {};
    local fileWriteCount = 0;
    set { SetInsert(fileWriteList, file.path); fileWriteCount+=1;}
        when (file.event == FILE_WRITE);
}
```

Process contexts

There are situations where it is convenient to set or query the value of a local variable associated with a process context other than the one indicated by the event being processed. The context operator is used to accomplish this and can be used anywhere in the language where an identifier is used. An example of using the context operator follows.

The first parameter to the context operator is an identifier name of type uint32 indicating a process id. The second parameter is the identifier name that is to be accessed. This example uses the context operator to propagate a taint flag between parent and child processes. When a process is started, the process reads the taint flag from the parent process using the context operator.

```
rulegroup fileReaders {
local bool tainted = false;
#
# apply a marker to file reader processes
#
set {tainted = TRUE;}
when (process.state == PROCESS_STARTED &&
     (
       strstr(process.path,"winword.exe",false)||
       strstr(process.path,"excel.exe",false) ||
       strstr(process.path,"powerpnt.exe",false) ||
       strstr(process.path,"AcroRd32.exe",false)
      )
     );

#
# propagate the tainted flag from parent to child processes
#
set {tainted = TRUE;}
when (process.state == PROCESS_STARTED && context[process.ppid, tainted] == TRUE);
};
```

The following example shows a context operator that is used on the left side of an assignment operation, watching for possible process injection and sets a value in both the process performing the injection and the process that was injected into.

```
rulegroup injectionTracking{

temp bool bNewInjection = false;
local bool bInjector = false;
local bool bInjectee = false;
global uint32 injectionAcccessMask = (PROCESS_VM_OPERATION | PROCESS_VM_WRITE);

set { bInjector = true;
      context[object.targetpid, bInjectee] = true;
    }
when ( object.targetobjecttype == OBJECT_TYPE_PROCESS ) and
     ( ( object.accessmask & injectionAcccessMask ) == injectionAcccessMask ) and
     ( object.sourcepid != getppid ( object.targetpid ) ) and
     ( object.sourcepid != object.targetpid) and
     ( (curproc.exe.sig == SIG_STATUS_INVALID ) or
     ( curproc.exe.sig == SIG_STATUS_NO_SIG ) );

# alert if an injection just occurred
alert when (bNewInjection == TRUE);
};
```

Rule annotation

Rule annotation helps to document rule behavior and is used by the Nuix Adaptive Security application when displaying data for rule matches.

Comments

Any line starting with the number sign (#), as shown in the following example, is treated as a comment by the rule compiler.

```
    #
    # Task modified from cmdline (process artifact)
    #
  alert when process.state == PROCESS_STARTED and
              strcmp(getbasename(process.path),"schtasks.exe",false) and
              strstr(process.cmdline,"/change",false);
```

Key-value pair data

Rule groups and individual rules can have lists of key-value pair data associated with them. This allows for a richer annotation of rules than comments alone can provide. The key-value pair data is returned to the server when a rule match occurs. These values are used by the Nuix Adaptive Security application for various purposes (display, sorting, and searching).

Key-value pair data specified at the rule group level is inherited by rules in that rule group. The syntax for specifying key-value pair data at the rule group level looks like the following:

```
    rulegroup example (<key> = "<value>", ...)
```

where key is an alphanumeric string that must begin with an alphabetic character and the value is any quoted-string value. The syntax for specifying key/value pair data at the rule level looks like the following:

```
    rule (<key>="<value>", ...) { <action> when <expression> ; }
```

The rule compiler places no restrictions on the length or contents of the values; however, several key names carry significance to the Nuix Adaptive Security application.

The key name platform is used to associate a rule to a particular platform. Currently the values “windows” and “macOS” are recognized. The use of the platform keyword allows rules targeted for different platforms to exist in the same rule file. When compiling rules, the combination of the platform keyword and the platform target chosen in the Nuix Adaptive Security application determines whether the rule will be compiled. If no platform key name is specified, the rule is assumed to be valid for all platforms and compilation will be attempted for any platform target.

The following key names carry significance when displaying alert data:

- Name
- Description
- Severity
- Category
- Author
- Type

The values associated with these key names are prominently displayed in the Alerts pane of the Nuix Adaptive Security application. View all other key names associated with a matched rule in the Nuix Adaptive Security application Alerts pane by clicking “Rule Metadata Other”. In addition, the use of the category keyword with a case-sensitive value of “Threat-Hunting” will cause the associated alert data to display under the Threat Hunting data source in the Guided Investigations section of the Investigate pane module in the Nuix Adaptive Security application. The use of the severity keyword with values (case sensitive) of “Critical”, “Medium”, or “Low” are used in the Dashboard module in the Nuix Adaptive Security application in the event summary display.

Rules inherit key-value pair data from their enclosing rule group. For example, when the alert rule matches, the key-value pairs returned will be author="Nuix", type=”Persistence”, name="Task modified", and description=” The command line…”.

```
rulegroup Task_Scheduling
    (author="Nuix",
     type="Persistence")
{
    rule
    (name="Task modified",
    description="""
        The command line "schtasks.exe /change" was executed. If successful
        this will result in the modification of an existing scheduled task
        on the endpoint. Task scheduling may be associated with persistence
        techniques used by malware and insiders.""")
    {
        alert when process.state == PROCESS_STARTED and
strcmp(getbasename(process.path),"schtasks.exe",false) and
                strstr(process.cmdline,"/change",false);
    }
}
```

A key name and value specified in the rule group can also be overridden within the rule itself. For example, when the alert rule matches, the value returned for author will be “Bob” and not “Nuix”, as shown in the following example:

```
rulegroup Task_Scheduling
    (author="Nuix",
     type="Persistence")
{
    rule
    (author="Bob",
     name="Task modified",
     description="""
```



```
        The command line "schtasks.exe /change" was executed. If successful
        this will result in the modification of an existing scheduled task
        on the endpoint. Task scheduling may be associated with persistence
        techniques used by malware and insiders.""")
    {
        alert when process.state == PROCESS_STARTED and
                strcmp(getbasename(process.path),"schtasks.exe",false) and
                strstr(process.cmdline,"/change",false);
    }
}
```

Each key name should be specified only once within a particular rule group key-value pair list or individual rule key-value pair list. If a key name is reused within a list, a warning is issued by the compiler and the first definition is used. The following rule uses the key name of author twice.

```
rule (author="Alice", author="Bob") {...}
```

The compiler generates the following warning message for this rule:

```
warning  0x00020023: "test.txt" line 3, second instance of metadata
keyword "AUTHOR" with value "Bob" ignored
```


Rule comments can be used in conjunction with key-value pairs as shown in the following example:

```
rulegroup Task_Scheduling
    (author="Nuix",
     type="Persistence")
{
    #
    # Task modified from cmdline (process artifact)
    #
    rule
    (name="Task modified",
    description="""
The command line "schtasks.exe /change" was executed. If successful this will
result in the modification of an existing scheduled task on the endpoint. Task
scheduling may be associated with persistence techniques used by malware and
insiders.""")
    {
        alert when process.state == PROCESS_STARTED and
                   strcmp(getbasename(process.path),"schtasks.exe",false) and
                   strstr(process.cmdline,"/change", false);
    }
}
```

Use time range matching

Time range matching makes it possible to write rules that specify days of the week and hour/minute ranges as part of the rule matching criteria. This can be useful, for example, when filtering out an activity that is considered acceptable during business hours, but which is highly suspicious outside of business hours. Time range matching is performed using the *timematch* function, as shown in the following example:

```
bool timematch(uint64 timestamp, uint32 day_mask, time_range range)
```

where *timestamp* is the 64-bit timestamp attribute from an event, *day_mask* is a bitmask indicating days of the week, and *range* indicates an hour/minute time. The *range* value is specified using the *timerange* operator, as shown in the following example:

```
timerange(string start_time,  string end_time)
```

where *start_time* and *end_time* are specified as strings in the format "HH: MM". The *start_time* must be smaller than the *end_time*. When matching against a time range, the *start_time* is treated as inclusive and the *end_time* is treated as exclusive. So,

```
timerange("07:00", "07:30")
```

specifies any time greater than or equal to 7:00:00 AM and less than 7:30 AM. Hours are specified in a 24-hour format, and valid hour values for *range* include 00 through 24. In order to match up through the end of the day the pseudo-hour "24:00" is allowed, so events that occur before midnight are matched.

Time range matching is relative to the day and hour/minute on the endpoint where the rules are executed and not to the server from where the rules are being pushed.

Example rules:

```
uint32 weekday = MON | TUES | WEDS | THURS | FRI;
uint32 weekend = SAT | SUN;
temp bool bRemovableMediaUsage = false;
temp bool bNonWorkHours = false;

set  { bNonWorkHours = TRUE; } when
  timematch(CurrentEventTimestamp(), weekday, timerange("17:30","24:00")) or
  timematch(CurrentEventTimestamp(),weekday, timerange("00:00", "07:00"));

set  { bRemovableMediaUsage = TRUE; } when event.type == media;

alert when bNonWorkHours and bRemovableMediaUsage ;
```


The first rule sets a temporary variable when the timestamp of the current event being processed falls on a weekend between 17:30 and 24:00 or between 00:00 and 07:30. The second rule sets a temporary variable to indicate when a removable media insertion or removal has occurred. The third rule alerts if both temporary variables are true.

Use threshold variables

Threshold variables make it possible to write rules to detect the occurrence of a fixed number of events within a specified time period. A threshold variable is initialized in the language using the *threshold_create* function, as shown in the following example:

```
threshold threshold_create( uint64 eventLimit, uint32 timePeriodMs )
```

where *eventLimit* is the number of events that must be exceeded within *timePeriodMs* milliseconds. For example, the following rule:

```
threshold t = threshold_create(10, 10000);
```

defines a threshold with an event limit of 10 and a time period of 10000 ms (10 seconds). If more than 10 events occur in a 10-second period, the threshold is considered to have been exceeded. To increment the event count associated with a threshold variable and check whether the threshold has been exceeded, the *threshold_increment* function is used.

```
Bool threshold_increment( threshold t, uint64 increment, uint64 timestamp );
```

The threshold increment function is used only within a set clause. The *timestamp* parameter is the time stamp associated with the event being recorded. The increment parameter indicates how much to increment the threshold counter. This value must be greater than 0.

The following rules increment the threshold variable each time a file write with high entropy occurs:

```
Temp bool b = false;
Set { b = threshold_increment(t, 1, file.timestamp); } when
file.event == FILE_WRITE and file.entropy > 7.8;
```

If the increment causes the threshold to be exceeded, the function returns **TRUE**. Otherwise, it returns **FALSE**. In this example, the return populates a temporary variable. The temporary variable is then used in subsequent rules to act, as shown in the following example:

```
alert when b and...;
isolate when b and ...;
```

Use set variables

Set variables are collections of scalar data types. Scalar data values can be inserted and removed from a set. Sets can be tested for the inclusion of a scalar value. Items within a set are keyed, based on their value. This allows constant time data lookup even for large sets. To define a set, square brackets must follow the variable name of a scalar data type.

Example:

```
global string removableMediaDevices[16] = {};
```

Sets have a maximum element count, which is specified explicitly by including a numeric value in the brackets or specified implicitly based on the number of initialization values assigned to the variables, as shown in the following example:

```
string adminAccounts[] = {"ADMIN", "ROOT", "OPER"};
md5digest honeyFiles[] = {
md5(“12123098fdbfe0987451df4312314324”),  md5(“19847edf7ba34398f8b7dda738475839”)
};
```

To insert into a set, the *setinsert* function is used, as shown in the following example:

```
set { setinsert(removableMediaDevices, upper(media.devicePath)); }
when  media.event == DEVICE_INSERTED;
```


Insertion operations that cause the set to exceed its maximum element count result in the least recently accessed item in the set being purged and replaced with the new item.

To remove an item from a set, the *setremove* function is used, as shown in the following example:

```
set { setremove(removableMediaDevices, upper(media.devicePath)); }
when  media.event == DEVICE_REMOVED;
```

Check for the existence of a value in a set, as shown in the following example:

```
alert when file.event == FILE_WRITE and
           removableMediaDevices contains upper(file.devicePath) and
           honeyFiles contains file.md5;

alert when session.event == SESSION_LOGON and
           session.type == SESSION_TYPE_REMOTE and
           session.ipaddr.ipv4 = ("192.168.5.0./24") and
           adminAccounts contains upper(session.username);
```


Namespace event caching

The Adaptive Security endpoint agent maintains a DNS lookup cache that records the result of every successful DNS name resolution attempt made on the endpoint. This cache can be referenced in rules by using the dnsmatch function, which checks for the existence of a mapping between a particular IP address and a domain name in the DNS lookup cache. If the mapping is present, the function returns TRUE.

The following alert rule matches when a process attempts to establish a connection to an IP address that maps to the name "www.nuix.com".

```
alert when network.action == NETFLOW_ESTABLISHED and
           dnsmatch(network.remote.ipaddr, "www.nuix.com");
```

Fully qualified DNS names can be used with the dnsmatch function, as in the previous example. No trailing dot is necessary when specifying a fully qualified DNS name. Subdomains can also be used with the dnsmatch function. Subdomains must include a dot to the left of the name. For example, the following rule would match a connection to any host within the ".nuix.com" domain.

```
alert when network.action == NETFLOW_ESTABLISHED and
         dnsmatch(network.remote.ipaddr, ".nuix.com");
```

The following rule would match any connection attempt to any IP address that maps to a name under the ".com" domain.

```
alert when network.action == NETFLOW_ESTABLISHED and
         dnsmatch(network.remote.ipaddr, ".com");
```

The dnsmatch function can also be used to match against a list of DNS names as shown in the following example.

```
string dnsnames[]=
{
  ".dropbox.com",
  ".ru",
  "perugemstones.com"
};

alert when network.action == NETFLOW_ESTABLISHED and
           dnsmatch(network.remote.ipaddr, dnsnames);
```

Match substrings against string sets

A common scenario is the need to match a string attribute field in an event against a defined set of keywords. An example is a set of keywords to search for in keystroke data. Instead of creating *N* separate rules to match each keyword individually, define all the keywords in a string set and write a single rule using the setstrstr function.

Example:

```
strings keywords[]={
    "embezzle",
    "western union",
    "money",
    "bank account",
};
alert when setstrstr(keystroke.keydata, keywords, false);
```

setstrstr iterates the keywords set and performs a substring matching operation using each word in the keywords set as the substring to look for in the keystroke.keyData attribute. After the first match, setstrstr stops and returns **TRUE**. If there is no match, setststr iterates through the entire set and eventually returns **FALSE**. Whenever using the setstrstr function, it is important to make sure the substrings in your list are actually "substrings". Keywords are matched as substrings in other strings. Making individual strings in the list exceedingly long or precise reduces the effectiveness of the matching.

Another example of using setstrstr is searching the titles of printed documents in print events. The print.documentname field contains the text of the title bar of the Windows application that initiated the printing. Depending on the application, the title bar displays data other than just the file name, URL, etc. that is currently in use by the application. To look for specific documents being printed, it would be best to do substring matching against the print.documentname attribute.

Example:

```
strings documentNames[] =
{
    "finances2018", "customer_db", "employee_salaries", "new_leads"
};
alert when setstrstr(lower(print.documenttitle), documentNames);
```

In addition to reducing the number of individual rules, you would have to write to match against an individual event type. Defining the data separately from rules means the data can be reused in rules for other event types. For example, define documentNames as the authoritative set of sensitive document names to track across all of your rules. That list is then referenced for rules for various event types as shown in the following example.

Example:

```
strings documentNames[] =
{
    "finances2018", "customer_db", "employee_salaries", "new_leads"
};
# look for printing of sensitive documents
alert when setstrstr(print.documenttitle, documentNames, false);
# look for mentions of sensitive document names in keystrokes
alert when setstrstr(keystrok.keydata, documentNames, false);
# look for write to removable media of sensitive document names
alert when setstrstr(file.path, documentNames, false) and file.event == FILE_WRITE
and nias::bRemovableMediaWrite == TRUE;
```

Match IP addresses against ipaddr sets

Another common scenario is the need to match an IP address in an event attribute against a set of IP hosts or network addresses. Define a set of type ipaddr that defines "home" or "trusted" IP networks and hosts.

Example:

```
ipaddr HomeAddrList[] =
{
   ip("192.168.1.0/24"),
   ip("192.168.2.0/24"),
   ip("192.168.100.1)
};
```

Write a simple rule to alert on connections established to machines outside of those hosts or networks.

Example:

```
alert when network.action == NETFLOW_ESTABLISHED and not
setipcmp(network.remote.ipaddr, HomeAddrList);
```

setipcmp iterates the HomeAddrList set and performs IP comparisons between each IP address in the set and network.remote.ipaddr. The comparisons performed match both IP host and IP network addresses. If network.remote.ipaddr was equal to 192.168.1.52, the comparison matches with 192.168.1.0/24 on HomeAddrList, because the host address 192.168.1.52 falls within that network address.

Match on Word Boundaries with FindWord

Keystroke and clipboard event data will typically contain sentences or paragraphs of text composed in whatever language is in use on the endpoint. When writing rules to match keywords against the data fields of these events use of the strstr function can be imprecise because it searches for a matching substring anywhere within the data without consideration of word boundaries. For example, consider the two sentences below.

3. “That's why he wants to quit.”
4. “That's quite a result.”

The rule

alert when strstr(keystroke.keydata, “quit”, false);

will match both sentences one and two because the sequence of characters “quit” appears in both sentences. The findword function can produce more precise matches than strstr when matching language text because the findword function matches along word boundaries. The rule

alert when findword(keystroke.keydata,”quit”, false);

Will only match sentence one above, because “quit” appears as a bounded word, in this case, preceded by a space and followed by a period. It will not match sentence two in which “quit” appears as a substring in another word. The word boundary analysis performed by the findword relies upon the Unicode Consortium’s open-source ICU (International Components for Unicode) library which implements robust, language-specific rules for identifying word boundaries.

Limit the Frequency of Rule Match

It is often useful to limit the number of times a rule can match within a given time period. One example is screenshotting rules. Consider the rule below, which triggers a screenshot task when text is entered into a Skype window. The rule will take a screenshot every 5 seconds for the next 30 seconds.

```
screenshot(keystroke.pid, 5, 30) when
        strstr(keystroke.windowTitle, “skype”, false);
```

The problem with this rule is as the user continues to enter text into the Skype window the rule will be triggered over and over, potentially triggering numerous screenshot tasks with each tasking screen shots for the next 30 seconds. The getrulematchinterval function can be used to limit how frequently a rule can match. This function returns the numbers of seconds between the last time the rule matched and the current time. A modified screenshot rule below ensures that the screenshot rule can only fire as often as every 30 seconds.


```
screenshot(keystroke.pid, 5, 30) when
        strstr(keystroke.windowTitle, “skype”, false) and
        getrulematchinterval() > 30;
```

The value returned by getrulematchinterval is calculated by subtracting the rule’s last match time from the current event timestamp. The rule’s last match time starts with a value of zero each time the agent is started. The last rule match time is reset to zero for all rules whenever any rule updates are pushed to the agent.

Rule matches can also be limited based on the number of times a rule has matched. The getrulematchcount returns the number of times the rule has matched since the last time agent was restarted or since any rule updates were pushed to the agent. This is useful for limiting rules from firing more than a certain number of times regardless of how long ago it last matched. The rule below will only fire twice.

```
alert when  (process.state == PROCESS_STARTED || process.state == PROCESS_EXISTING) and
        endswith(process.path, CommonDefs::gPrivateMessagingApps, false) and
        getrulematchcount() < 2;
```

The rule match count for each rule starts at zero when the agent is started and is incremented by 1 each time the rule matched. The count is incremented after the match logic is matched. For that reason, use of getrulematchcount with a *greater than* or *equal* operator won’t work. For example, a rule such as “alert when getrulematchcount() == 1” will never match. The match count is reset to zero for all rules whenever the agent restarts or whenever rules any rule changes to any rules are made.

Path normalization

Currently, file and registry paths contained in Windows event data and state data appear in the format used by the kernel in which namespace links and aliases appear as fully resolved. This can cause confusion when attempting to match paths in a rule. File paths in events generally appear in the following format:

```
\Device\HarddiskVolume1\Windows\System32\svchost.exe
```

Instead of:

```
c:\windows\system32\svchost.exe
```

File paths referencing data on a Server Message Block (SMB) share appear as the following:

```
\Device\Mup\[remote ip address]\[share name]\...
```

Registry paths in events generally appear in the following form:

```
\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services
\REGISTRY\MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Schedule\TaskCache\Tasks\{BF1AEDE2-3B07-4D8A-A03E-9C8B80AAD258}
\REGISTRY\USER\S-1-5-21-586946019-3361328298-3290581161-
1000\Software\Microsoft\Windows\CurrentVersion\Explorer\StartPage2
```

Instead of in the following form:

```
HKLM\SYSTEM\CurrentControlSet\Services
HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Schedule\TaskCache\Tasks\{BF1AEDE2-3B07-4D8A-A03E-9C8B80AAD258}
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\StartPage2
```

This means that the following rule never matches:

```
alert when strcmp(file.path,"c:\windows\system32\file.txt",false);
```

The easiest solution for this is to instead write the rule using the strstr operator, as shown in the following example:

```
alert when strstr(file.path,"windows\system32\file.txt",false);
```


Similar logic can be applied to registry key matching rules, as shown in the following example:

```
alert when
strstr(registry.key,"Software\Microsoft\Windows\CurrentVersion\Explorer\StartPage2"
,false);
```

In addition, a built-in function exists to normalize kernel registry paths. The getregistrysubkey() function is used to trim off portions of the hive path, as shown in the following example:

```
alert when strstr(getregistrysubkey(registry.key),
"Software\Microsoft\Windows\CurrentVersion\Explorer\StartPage2",false);
```

The following table describes the file path formats used for various event types.

Event type	Event attribute	Path example
process	process.path	\Device\HardDisk...
process	process.parentpath	\Device\HardDisk...
object	object.targetprocess	\Device\HardDisk...
image load	imageload.imagepath	c:\...
file	file.path	\Device\HardDisk...
curproc	curproc.filewritelist	\Device\HardDisk...
registry	registry.keyname	\\REGISTRY\\MACHINE\\ \\REGISTRY\\USER\\

When comparing paths contained in event data against paths contained in curproc state attributes, there is no issue because the paths are both in the same format.

Example:

```
alert when curproc.parent.filewritelist contains process.path;
```

Match against lists

The state engine maintains string lists for a number of process state items including the list of loaded modules and the list of imported functions. Understanding how this data is or is not currently normalized is important for writing rules that match properly.

curproc.importlist

The string provided in the rule should be in format "MODULENAME.EXT!ImportedFunctionName" where the module portion of the name is uppercase, and the imported function name is left in its original case. This is because imported function names in Windows are case sensitive.

Example:

```
curproc.importlist contains "KERNEL32.DLL!VirtualAlloc";
```

curproc.modulelist

The module names stored in the curproc.modulelist are all uppercase. The module names contained in the modulelist are in the format MODULENAME.EXT.

Example:

```
curproc.exe.modulelist contains "ADVAPI32.DLL";
```

Compilation warnings

The rule compiler generates compilation warnings in certain scenarios. While warnings do not prevent rule compilation, any rules that generate compiler warnings should be rewritten.

Common compiler warnings

Access per-process variables

If a rule attempts to access a per-process variable and the rule does not reference an event type that has process context associated with the rule, then a compiler warning is generated. For more information about process context and variables, see Variable Scope.

Example:

```
local bool bSomeCondition;
set { bSomeCondition = true;} when media.deviceserial ....
alert when bSomeCondition;
```

When compiled, the alert rule generates the following warning:

```
variables requiring process context were referenced but no event providing process
context was referenced
```

The local variable bSomeCondition is referenced in the alert rule, but no event is referenced that provides process context. In addition, bSomeCondition, once set, remains **TRUE** unless another set rule is specifically written to clear the value. That means that the alert rule continues to fire on every subsequent future event of any type that is processed. The rule is best rewritten by making bSomeCondition a temporary variable.

Example:

```
temp bool bSomeCondition = false;
set { bSomeCondition = true;} when file.event ....
alert when bSomeCondition;
```

Temporary variables are reset to their default values automatically each time a new event is processed and are referenced in any rule without requiring that rule to reference an event that provides process context.

Use identifier names greater than 64 characters in length

Identifiers (rule group names and variable names) must be 64 characters or less in length. If longer names are used, the names are automatically truncated to the 64-character limit, and a compiler warning is generated. A variable declaration like:

```
uint32 ZZ0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef;
```

results in a compiler warning like:

```
identifier "ZZ0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef"
exceeded maximum allowed length of 64 and was truncated
```


Event rule attributes

The following tables, grouped by event type, detail the event attributes available for use in rules.

Clipboard paste event (Windows only)

Clipboard paste events are generated in response to the pasting of data from the clipboard. The following table describes the events attribute, allowed values, and description for each clipboard paste event type.

Universal Windows Platform apps, also known as Windows Store apps and Metro-style apps, are designed to run in Windows AppContainers. Applications running in Windows AppContainers do not have direct access to the Windows clipboard. Their access to the Windows clipboard is brokered through another process. Due to this architecture, Adaptive Security may not be able to monitor clipboard activity performed by these applications.

Clipboard attribute name	Allowed values	Description
clipboard.pid	pid	The process id for the process performing the paste operation.
clipboard.processname	string	The path of the executable for the process performing the paste operation.
clipboard.windowtitle	string	The window title for the process performing the paste operation.
clipboard.userid	string	Security identifier (SID) string representing the user associated with the paste operation. Well known SIDs are represented using the Microsoft Security Definition Language. Other SIDs use the standard string representation of a SID (S-*R*-*I*-*S*-*S*…).
clipboard.producer.pid	pid	The process id for the process that originally placed the data onto the clipboard. In some cases this may be zero if the data was placed onto the clipboard before the Adaptive Security agent was running.
clipboard.producer.processname	string	The path of the executable for the process that originally placed the data onto the clipboard. In some cases this may be empty if the data was placed onto the clipboard before the Adaptive Security agent was running.
clipboard.producer.windowtitle	string	The window title for the process that originally placed the data onto the clipboard. In some cases this may be zero if the data was placed onto the clipboard before the Adaptive Security agent was running.

Clipboard attribute name	Allowed values	Description
clipboard.datatimestamp	uint64	The timestamp indicating when the data was originally placed onto the clipboard. This will be zero is the data was placed onto the clipboard before the Adaptive Security agent was running.
clipboard.format	CB_FORMAT_STRING CB_FORMAT_JPEG CB_FORMAT_FILE_LIST	The format of the data that is being pasted. CB_FORMAT_STRING indicates that text data is being pasted. CB_FORMAT_JPEG indicates that an image is being pasted. CB_FORMAT_FILE_LIST indicates that files are being pasted on the explorer desktop.
clipboard.data.string	string	If clipboard.format is CB_FORMAT_STRING, this field will contain the string data.
clipboard.data.filelist	string	If clipboard.format is CB_FORMAT_FILE_LIST, this field will contain a comma separated list of file paths.
clipboard.timestamp	uint64	The timestamp of the event.

Event type (Windows, macOS)

These attributes exist to allow an entire event class to be specified at once. These are appropriate for use in suppress rules to suppress the writing of an entire class of events to the DVR or for use in forwarding rules to forward an entire class of events to the server.

The following table describes the event attribute, allowed values, and description for each event type.

Event attribute	Allowed values	Description
eventtype.process	TRUE or FALSE	Process events
eventtype.file	TRUE or FALSE	File events
eventype.inspectfilematch	TRUE or False	Inspect file match events (Windows only)
eventtype.session	TRUE or FALSE	Session events
eventtype.media	TRUE or FALSE	Media events
eventtype.thread	TRUE or FALSE	Thread events
eventtype.imageload	TRUE or FALSE	Image load events
eventtype.object	TRUE or FALSE	Object events
eventtype.registry	TRUE or FALSE	Registry events
eventtype.print	TRUE or FALSE	Print events
eventtype.namespace	TRUE or FALSE	Namespace events
eventtype.network	TRUE or FALSE	Network events
eventtype.keystroke	TRUE or FALSE	Keystroke events
eventtype.defender	TRUE or FALSE	Microsoft Defender events
eventtype.clipboard	TRUE or FALSE	Clipboard events
eventtype.url	TRUE or FALSE	URL events

File event (Windows, macOS, Linux)

File events are generated when the following occurs:

- A file is closed if the file has had content written to it.
- A file is renamed.
- A file is deleted.
- A file is opened.
- A volume is mounted.

Note: The macOS agent does not generate events for files written through command line redirection in the shell. For example, the shell command “cat myfile > output.txt” will not generate a file write event for the file output.txt.

The following table describes the file attribute, allowed values, and description for each file event type.

File attribute name	Allowed values	Description
file.event	FILE_WRITE FILE_RENAME FILE_DELETE FILE_OPEN	Indicates the file event type. • **FILE_WRITE** is generated at the time a file is closed if the file was written to. • **FILE_DELETE** is generated when a file is deleted. • **FILE_RENAME** is generated in response to files being renamed. • **FILE_OPEN** is generated in response to a non-directory file, not ending in .exe or .dll, that is opened for reading or writing of data.
file.path	string	Indicates the full path of the file that was written for **FILE_WRITE** operations or the full path of the newly renamed file for **FILE_RENAME** operations.
file.size	uint64	Indicates the size of the file after the file was written to and closed for **FILE_WRITE** operations or the size of the file that was deleted for **FILE_DELETE** operations. This will be 0 for **FILE_RENAME** operations.
file.md5	md5digest	The MD5 hash digest for the file after the file was written and closed for **FILE_WRITE** operations or the hash digest of the file that was deleted for **FILE_DELETE** operations. This will be 0 for **FILE_RENAME and FILE_OPEN** operations.
file.originalpath	string	Indicates the full path of the original filename for **FILE_RENAME** operations.
file.creationdisposition	FILE_CREATE_NEW FILE_CREATE_ALWAYS	Indicates the disposition with which the file was opened. This Windows-only field is valid only for **FILE_WRITE** and **FILE_OPEN** operations. This is only available on Windows.

File attribute name	Allowed values	Description
	FILE_OPEN_EXISTING FILE_OPEN_ALWAYS	
file.shareaccess	FILE_SHARE_READ FILE_SHARE_WRITE FILE_SHARE_DELETE	A bitmask indicating the shared access with which the file was opened. This field is valid only for **FILE_WRITE** and **FILE_OPEN** operations. This is only available on Windows.
file.desiredaccess	FILE_READ_DATA FILE_LIST_DIRECTORY FILE_WRITE_DATA FILE_ADD_FILE FILE_APPEND_DATA FILE_ADD_SUBDIRECTORY FILE_CREATE_PIPE_INSTANCE FILE_READ_EA FILE_WRITE_EA FILE_EXECUTE FILE_TRAVERSE FILE_DELETE_CHILD FILE_READ_ATTRIBUTES FILE_WRITE_ATTRIBUTES FILE_ALL_ACCESS FILE_GENERIC_READ FILE_GENERIC_WRITE FILE_GENERIC_EXECUTE	A bitmask indicating the access with which the file was opened. This field is only valid for **FILE_WRITE** and **FILE_OPEN** operations. This is only available on Windows.
file.pid	pid	The process id associated with the file action.
file.entropy	double	The entropy of the file that was written for **FILE_WRITE** operations or deleted for **FILE_DELETE** operations.

File attribute name	Allowed values	Description
file.usersid	string	Security identifier (SID) string representing the user associated with the file operation. Well-known SIDs are represented using Microsoft Security Definition Language. Other SIDs use the standard string representation of a SID (S-*R*-*I*-*S*-*S*…). This is only available on Windows.
file.userid	uint64	The numeric user id of the user associated with the file operation. This is only available on macOS.
file.devicepath	string	The kernel device path associated with the file, for example. \device\harddiskvolume1, or similar. This is only available on Windows.
file.timestamp	uint64	The timestamp of the event.
file.labels	string	Contains a key/value pair list of sensitivity labels from a Microsoft Office document. This field is only populated for Microsoft Office documents for FILE_WRITE and FILE_RENAME operations.

Imageload event (Windows, macOS, Linux)

Imageload events are generated whenever an executable image is loaded on the endpoint. This includes the loading of executable images and dynamic link libraries that take place whenever a new process is started. It also includes the loading of dynamic link libraries, which can occur after a process has started through the use of the Win32 LoadLibrary/LoadLibraryEx APIs or the macOS/Linux dlopen API.

The following table describes the ImageLoad attribute, allowed values, and description for each ImageLoad event type

Imageload attribute name	Allowed values	Description
imageload.imagepath	string	Indicates the full path of the image that was loaded.
imageload.baseaddr	uint64	Indicates the base address where the image was loaded. This is only available on Windows.
imageload.imagesize	uint64	Indicates the image size that was loaded.
imageload.loadstate	IMAGE_LOADED IMAGE_EXISTING	• **IMAGE_LOADED** indicates that the image has been loaded. • **IMAGE_EXISTING** is used in image load attributes generated for images already loaded at the time the endpoint agent started.
imageload.md5	string	Indicates the MD5 digest of the image that was loaded. This is only available on Windows.
imageload.usersid	string	Security identifier string representing the user associated with the image load operation. Well-known SIDs are represented using the Microsoft Security Definition Language. Other SIDs use the standard string representation of a SID (S-*R*-*I*-*S*-*S*…). This is only available on Windows.
imageload.pid	pid	The process id for the process performing the image load.
image.timestamp	uint64	The timestamp of the event.

InspectFileMatch event (Windows only)

InspectFileMatch events are generated whenever one or more regular expressions matches occur as the result of an InspectFile rule action. Only one InspectFileMatch event is returned per InspectFile action regardless of how many matches occurred within that file.

The following table describes the InspectFileMatch attribute, allowed values, and description for each InspectFileMatch event type.

InspectFileMatch attribute name	Allowed values	Description
inspectfilematch.path	string	Indicates the full path of the file that was inspected.
inspectfilematch.label	string	The label string corresponding to the label string used in the InspectFile rule action that resulted in this match event.
inspectfilematch.pid	pid	The process ID for the process that triggered the inspect rule.
inspectfilematch.timestamp	uint64	The timestamp of the event.

Keystroke event (Windows only)

Keystroke events are generated in response to keystrokes entered on the endpoint machine, but not for each individual keystroke entered. Instead, keystrokes are collected into a buffer, and then keystroke events are generated at specific tear points. These tear points are created when certain events occur including the following:

The user presses the enter/return key.

- The user presses a control sequence that includes any of the following keys: ALT, CTRL, WIN, F1-F24, ESCAPE, PRINT SCREEN.
- The user shifts focus to another window and enters a keystroke in that window.
- The keystroke log buffer has reached its *maximum buffered keys* limit.
- The timeout keystroke buffer *periodic flush timer* has elapsed.

Note: The keystroke *periodic flush timer* and the keystroke *maximum buffered keys* limit can be configured on a per endpoint basis from the Nuix Adaptive Security console.

The following table describes the keystroke attribute, allowed values, and description for each keystroke event type.

Keystroke attribute name	Allowed values	Description
keystroke.pid	pid	The process id for the application receiving the keystrokes.
keystroke.windowtitle	string	The contents of the title bar for the application window receiving the keystrokes.
keystroke.usersid	string	Security identifier (SID) string representing the user associated with the process creation. Well-known SIDs are represented using the Microsoft Security Definition Language. Other SIDs use the standard string representation of a SID (S-R-I-S-S…).
keystroke.keydata	string	A string of captured keystroke data. This is a decoded representation of the virtual keys codes captured from the operating system. When representing non-alphanumeric keys and control key sequences a special notation is used. See **Appendix A: Keystroke Event Special Character Notation** for more details and examples.
keystroke.timestamp	uint64	The timestamp of the event.

Media event (Windows, macOS, Linux)

Media events are generated in response to the insertion or removal of removable media, which includes USB thumb drives as well as external hard drives connected by USB or Firewire.

The following table describes the removable media attribute, allowed values, and description for each media event type.

Removable media attribute name	Allowed values	Description
media.event	DEVICE_INSERTED DEVICE_REMOVED	Indicates whether a removable media device is being inserted or removed from the system.
media.rootpathname	string	The pathname where the removable media device is mounted. • On Windows, the path is displayed in the format [DriveLetter]:\. ("e:\", "f:\"). • On macOS, the path is displayed as a volume path such as “/Volumes/mydevice”.
media.bustype	BUS_FIREWIRE BUS_USB BUS_ESATA	The type of bus the removable media device was attached to.
media.filesystemname	string	The name of the file system used to format the volume, which is usually one of the following: "NTFS", "FAT32", for example, FAT”, “HFS”, or “APFS”. On Linux, exFat and NTFS file system may appear with the name “fuseblk”. See Filesystem in Userspace.
media.volumelabel	string	The volume label assigned to the removable media volume. A user can set the volume labels anytime during or after formatting a volume. The volume label may give insight as to what the storage volume contains.
media.volumeserial	string	The volume serial number is a 32-bit value assigned to a volume by Windows at the time a volume is formatted. It is one way of tracking media volumes, although the volume serial number can potentially be changed by a user with enough effort. It is displayed in the header of the output from the command prompt "dir" command.
media.volumefreebytes	uint64	Indicates the number of free bytes on the removable media storage volume.
media.volumetotalbytes	uint64	Indicates the total size in bytes of the removable media storage volume.
media.deviceserial	string	For USB thumb drives, the serial number is embedded in the hardware by the manufacturer. An example serial number taken from a SanDisk thumb drive is: "SNDKAA03743213606705".

Removable media attribute name	Allowed values	Description
media.devicepid	string	The device product id string value. The value is set by the device manufacturer. An example pid string from a SanDisk Cruzer Mini thumb drive is: "Cruzer Mini".
media.devicevid	string	The device vendor id string. The value is set by the device manufacturer. An example vendor string from a SanDisk Cruzer Mini thumb drive is: "SanDisk".
media.devicepath	string	The kernel device path associated with the mounted drive, for example, \device\harddiskvolume4, or similar.
media.timestamp	uint64	The timestamp of the event.

Memory scan injection event (Windows only)

Memory scan injection events are generated in response to instances of covertly injected portable executable (PE) modules discovered when executing the memscan rule action on the endpoint.

The following table describes the injected image attribute, allowed values, and description for each memory scan injection event type.

Injected image attribute name	Allowed values	Description
injectedImage.timestamp	uint64	The timestamp of the event.
injectedImage.pid	pid	The process id of the process in which the injected module was found.
injectedImage.imagepath	string	The full path of the process executable in which the injected module was found.
injectedImage.baseaddress	uint64	The base virtual address where the injected module is loaded in memory.
injectedImage.checksum	uint32	The checksum of the injected PE module as pulled from the Checksum field in the PE headers of the injected module in memory.
injectedImage.sizeofimage	uint32	The size of the injected PE module, as pulled from the SizeOfImage field in the PE headers of the injected module in memory.
injectedImage.owner	MEMSCAN_OWNER_UNKNOWN MEMSCAN_OWNER_NUIX	Indicates the source or owner of the injected module: • MEMSCAN_OWNER_UNKNOWN indicates that the source of this injected module is unknown and may be malicious. • MEMSCAN_OWNER_NUIX indicates that this is an injected module that belongs to Nuix Adaptive Security and is not a security threat.

Memory scan patch event (Windows only)

Memory scan patch events are generated in response to instances of code patches discovered when executing the memscan rule action on the endpoint.

The following table describes the patched image attribute, allowed values, and description for each memory scan patch event type.

Patched image attribute name	Allowed values	Description
patchedImage.timestamp	uint64	The timestamp of the event.
patchedImage.pid	pid	The process id of the process in which the patched code was found.
patchedImage.imagepath	string	The full path of the process executable in which the injected module was found.
patchedImage.modulepath	string	The full path of the process executable in which the injected module was found.
patchedImage.baseaddress	uint64	Indicates the virtual address in memory where the patch begins.
patchedImage.length	uint64	The length in bytes of the patch.
patchedImage.inlinehook	bool	Indicates that the patch represents an inline hook used to redirect the flow of execution code to another location in memory.
patchedImage.eathook	bool	Indicates that the patch is a patch of the PE export address table.
patchedImage.targetaddress	uint64	If patchedImage.inlinehook or patchedImage.eathook is **TRUE**, this field contains the target virtual address for the hook.
patchedImage.targetsymbol	string	Symbolic name representing the target address in the event of an Export Address Table (EAT) hook or inline hook. Depending on the amount of symbolic data that can be resolved. the content of the string will vary. An example of symbolic data for an inline hook pointing into a non-exported function within chrome.exe is shown below: \Device\HarddiskVolume3\Program Files (x86)\Google\Chrome\Application\chrome.exe!0x00000000000bd9f0
patchedImage.patchsymbol	string	Symbolic name representing the address where the patch begins. Depending on the amount of symbolic data that can be resolved, the content of the string will vary. An example of symbolic data for a patch that occurs in a function exported from ntdll.dll is shown below: \Device\HarddiskVolume3\Windows\System32\ntdll.dll!NtOpenSymbolicLinkObject+0x00000020
patchedImage.owner	MEMSCAN_OWNER_UNKNOWN	Indicates the source or owner of the patch.

Patched image attribute name	Allowed values	Description
	MEMSCAN_OWNER_NUIX	• MEMSCAN_OWNER_UNKNOWN indicates that the source of this patch is unknown and may be malicious. • MEMSCAN_OWNER_NUIX indicates that this is a patch installed by Nuix Adaptive Security and is not a security threat.

Microsoft Defender events (Windows only)

The endpoint agent interacts with Microsoft Defender to identify active threats on the endpoint. When the endpoint agent starts, the endpoint agent polls Microsoft Defender for active threats. After startup, the endpoint agent periodically polls (using a 30-second interval) Microsoft Defender to identify subsequent threats. For each threat reported by Microsoft Defender, the endpoint agent generates an event and sends the event to the filter engine for processing.

The following table describes the defender attribute, allowed values, and description.

Defender attribute name	Allowed values	Description
defender.timestamp	uint64	The timestamp of the event.
defender.threatid	uint64	GUID uniquely identifying a threat for Microsoft Defender. **Note**: Windows 7 and versions of Windows greater than Windows 8 use different domains for their GUIDs.
defender.threatname	string	Name of the threat.
defender.threatstatus	THREAT_STATUS_UNKNOWN THREAT_STATUS_DETECTED THREAT_STATUS_CLEANED THREAT_STATUS_QUARANTINED THREAT_STATUS_REMOVED THREAT_STATUS_ALLOWED THREAT_STATUS_BLOCKED THREAT_STATUS_CLEAN_FAILED THREAT_STATUS_QUARANTINE_FAILED THREAT_STATUS_REMOVE_FAILED THREAT_STATUS_ALLOW_FAILED THREAT_STATUS_ABANDONED THREAT_STATUS_BLOCK_FAILED THREAT_STATUS_INVALID	The status of the threat at the time the event was received by the filter engine. It is important to understand that Microsoft Defender often cleans/removes/quarantines active threats automatically. If a threat event indicates **DETECTED**, the threat event may be cleaned automatically in very short order. • **DETECTED**: Threat is discovered, but not remediated. • **CLEANED**: Threat has been eliminated, likely including multiple steps (killing a malicious process and deleting a file). • **QUARANTINED**: Threat has been moved and access has been restricted in a way that it is mitigated presently. • **REMOVED**: Threat has been deleted. Often just deleting a file, otherwise, the status would likely be CLEANED. • **ALLOWED**: The user was prompted with a warning and authorized the threat to proceed.

Defender attribute name	Allowed values	Description
		• **BLOCKED**: The threat was prevented, blocked execution, for example, through user action or automatically. • **THREAT_STATUS_*_FAILED**: An attempt was made to perform the action, but that attempt did not succeed.
defender.detectiontime	uint64	The timestamp of when Microsoft Defender detected the threat.
defender.remediationtime	uint64	The timestamp of when Microsoft Defender cleaned/quarantined/removed the threat.
defender.threattype	THREAT_TYPE_UNKNOWN THREAT_TYPE_KNOWN_BAD THREAT_TYPE_BEHAVIOR THREAT_TYPE_KNOWN_GOOD THREAT_TYPE_NIS	Microsoft Defender characterizes threat types as follows: • **KNOWN_BAD**: Microsoft Defender matched a threat signature. • **BEHAVIOR**: Microsoft Defender noticed a behavior. • **KNOWN_GOOD**: Microsoft Defender matched a signature, but the "threat" is not a problem (likely used for their testing). • **NIS**: Microsoft Defender observed a network attack (or attempt to attack).
defender.threatorigin	THREAT_ORIGIN_UNKNOWN THREAT_ORIGIN_LOCAL_MACHINE THREAT_ORIGIN_NETWORK_SHARE THREAT_ORIGIN_INTERNET THREAT_ORIGIN_OUTBOUND THREAT_ORIGIN_INBOUND	Where the threat came from. • **LOCAL_MACHINE, NETWORK_SHARE,** and **INTERNET** are derived from the NTFS Zone ADS on the threat originating file. • **OUTBOUND** and **INBOUND** are only applications for NIS-detected events and indicate the direction of network traffic associated with the threat.
defender.threatexecutionstatus	THREAT_EXECUTION_STATUS_UNKNOWN THREAT_EXECUTION_STATUS_BLOCKED THREAT_EXECUTION_STATUS_ALLOWED	Whether or not the threat has executed. • **UNKNOWN**: Microsoft Defender does not know if the threat ran.

Defender attribute name	Allowed values	Description
	THREAT_EXECUTION_STATUS_EXECUTING THREAT_EXECUTION_STATUS_NOT_EXECUTING	• **BLOCKED**: Microsoft Defender blocked the threat from running. • **ALLOWED:** The threat was allowed to run. • **EXECUTING**: The threat is currently executing. • **NOT EXECUTING**: The threat is not currently executing. For a threat that was detected as Microsoft Defender attempted to execute, you should see **BLOCKED** or **ALLOWED**. For a threat that was detected after Microsoft Defender was already running, you should see **EXECUTING**. You would perhaps then expect to see **NOT_EXECUTING** for a threat that was found on disk that was not presently executing or discovered when Microsoft Defender was attempting to execute, but that has never been observed in testing. In that case, **UNKNOWN** is observed.
defender.detectionsource	THREAT_DETECTION_SOURCE_UNKNOWN THREAT_DETECTION_SOURCE_USER_SCAN THREAT_DETECTION_SOURCE_SYSTEM_SCAN THREAT_DETECTION_SOURCE_REALTIME THREAT_DETECTION_SOURCE_IE_OUTLOOK THREAT_DETECTION_SOURCE_NETWORK_INSPECTION THREAT_DETECTION_SOURCE_IE_BHO THREAT_DETECTION_SOURCE_ELAM THREAT_DETECTION_SOURCE_LOCAL_ATTESTATION THREAT_DETECTION_SOURCE_REMOTE_ATTESTATION THREAT_DETECTION_SOURCE_AMSI	How did Microsoft Defender come to learn about this threat? • **UNKNOWN**: Have not seen this occur. • **USER_SCAN**: The user requested a scan and the threat was found. • **SYSTEM_SCAN**: A regular recurring scan found the threat. • **REALTIME**: Real-time protection mechanisms. • **IE_OUTLOOK**: Found while scanning browser downloads or email attachments. • **NETWORK_INSPECTION**: Found looking at network traffic. • **IE_BHO**: IExtensionValidation failed to validate a web control. • **ELAM**: Early Load Anti-Malware detected a threat. • **LOCAL_ATTESTATION**: System integrity did not validate based on a local attestation client.

Defender attribute name	Allowed values	Description
		• **REMOTE_ATTESTATION**: System integrity did not validate based on a remote attestation client. • **AMSI**: Anti-Malware Scan Interface (used by third-party applications to ask Microsoft Defender to scan resources for them).
defender.detectiontype	THREAT_DETECTION_TYPE_UNKNOWN THREAT_DETECTION_TYPE_CONCRETE THREAT_DETECTION_TYPE_HEURISTIC THREAT_DETECTION_TYPE_GENERIC THREAT_DETECTION_TYPE_SUSPICIOUS THREAT_DETECTION_TYPE_FASTPATH	A nuanced field considering the threattype field; this field is basically a clarifying or subcategory of that threattype. If a threat is **THREAT_TYPE_KNOWN_BAD** and **THREAT_DETECTION_TYPE_CONCRETE** that means that a "concrete" signature (byte pattern or some other unambiguous attribute) was matched to declare the threat a **KNOWN_BAD**. If the threat is KNOWN_BAD, but the detection type is **THREAT_DETECTION_TYPE_HEURISTIC**, then the threat matched a heuristic, non-concrete, signature. A **THREAT_DETECTION_TYPE_SUSPICIOUS** indicates lower confidence that the threat event is a serious risk. It is unknown when THREAT_DETECTION_TYPE_GENERIC or THREAT_DETECTION_TYPE_FASTPATH are used.

Namespace event (Windows only)

Namespace events are generated in response to Domain Name System queries generated on the endpoint.

Note: Keep the following in mind when capturing namespace Insights:

All Namespace events are displayed in the Insights list, including blocked namespace events.

Some namespace events, like ns lookup, cannot be captured or blocked.

To know when a namespace is blocked, add an alert. Use the following as an example:

```
`rulegroup NamespaceAlert
   (author="Nuix",
   type="Namespace")
{

   #
   # Alert for Blocked Namespace Event
   #
   rule(name="Blocked Namespace Event",
   description="A blocked namespace event was generated.")
   {
       alert when namespace.blocked == true;
   }

}`
```

The following table describes the namespace attribute, allowed values, and description for each namespace type.

Namespace attribute name	Allowed values	Description
namespace.processname	string	The full path of the program that generated the request, for example, "c:\program files\Internet Explorer\iexplore.exe" or similar.
namespace.query	string	The name to be resolved, for example, "go.microsoft.com", "wpad", www.nuix.com.
namespace.pid	pid	Process id associated with the namespace activity.
namespace.timestamp	uint64	The timestamp of the event.
namespace.results	ipaddr	This attribute contains a list of IP addresses returned as a result of a successful query.
namespace.blocked	TRUE FALSE	Indicates whether the namespace query was blocked or not.

Network event (Windows, macOS, Linux)

Network events are generated when a network connection is established or terminated. Events are also generated periodically during the lifetime of a connection to provide updated statistics on the amount of data transferred to date. Finally, events are generated for certain socket state events of interest such as entering the listening state.

The following table describes the network attribute, allowed values, and description for each network event type.

Network attribute name	Allowed values	Description
network.action	NETFLOW_ESTABLISHED NETFLOW_TERMINATED NETFLOW_HEARTBEAT NETFLOW_OTHER	Indicates the type of netflow event to match against. • **NETFLOW_ESTABLISHED**: A new TCP session, or the flow of UDP or ICMP traffic between source and destination addresses/ports that have not been seen in the past 60 seconds. • **NETFLOW_TERMINATED**: Termination of a TCP session, or a UDP or ICMP session over which traffic has not flowed in the past 60 seconds. • **NETFLOW_HEARTBEAT**: A heartbeat event, which is generated on a periodic basis (every 10 minutes) for established network connections. • **NETFLOW_OTHER**: Used to indicate socket state change events generated separately from connection establishment/termination events, including when a socket is placed into listening mode. The network.socketstate attribute indicates the state.
network.socketstate	SOCKET_ESTABLISHED SOCKET_CLOSED SOCKET_LISTEN SOCKET_BOUND	Indicates the state of the socket associated with the network event. • **SOCKET_ESTABLISHED**: Expected for network actions of NETFLOW_ESTABLISHED or NETFLOW_HEARTBEAT. • **SOCKET_CLOSED**: Expected for actions of NETFLOW_TERMINATED. When network.action is **NETFLOW_OTHER**, the **SOCKET_BOUND** and **SOCKET_LISTEN** states can indicate that the socket has been bound to a local IP address and port, for example, the socket bind API has been called on the socket or has been placed in a listening state, for example, the socket listen API has been called on the socket. • **SOCKET_LISTEN** can be useful in detecting processes that are listening for remote connections even if no connections have been established yet. **Note**: The macOS agent only reports the SOCKET_LISTEN state. No other socket state changes are reported from macOS agents.
network.direction	INBOUND OUTBOUND	Indicates the direction of the flow associated with the event.

Network attribute name	Allowed values	Description
		• **INBOUND**: Traffic initiated from a remote system to the system running the endpoint client. • **OUTBOUND**: Traffic initiated from the endpoint client system to a remote system.
network.ipversion	IPV4 IPV6	Indicates the version of the IP protocol associated with the event.
network.local.ipaddr	ipaddr	Local address or netmask associated with an inbound or outbound network connection.
network.remote.ipaddr	ipaddr	The remote address or netmask associated with an inbound or outbound network connection.
network.local.port	uint32	Local port associated with an inbound or outbound network connection.
network.remote.port	uint32	Remote port associated with an inbound or outbound network connection.
network.icmp.type	uint32	ICMP type value associated with an inbound or outbound ICMP packet.
network.icmp.code	uint32	ICMP code value associated with an inbound or outbound ICMP packet.
network.proto	PROTO_ICMP PROTO_TCP PROTO_UDP	The protocol associated with an inbound or outbound network connection.
network.pid	pid	Process id associated with network activity.
network.socketstate	SOCKET_UNKNOWN SOCKET_CLOSED SOCKET_LISTEN SOCKET_BOUND	
network.preexisting	bool	When **TRUE**, indicates that this network event corresponds to a network state that was pre-existing when the endpoint agent started. For example, the endpoint agent generates network events with an action value of **NETFLOW_ESTABLISHED** for each existing network connection found when the endpoint agent starts. This field is set to **TRUE** for each of those events in that scenario.

Network attribute name	Allowed values	Description
network.sent.bytes	uint64	Total bytes of data sent over a network connection. This count does not include network or transport layer headers for TCP and UDP traffic. For ICMP traffic, the count does not include the network layer header, but it does include the full ICMP message (ICMP header plus the data portion of the ICMP message).
network.sent.packets	uint64	Total packets sent over the network connection.
network.recv.bytes	uint64	Total bytes of data sent over a network connection. This count does not include network or transport layer headers for TCP and UDP traffic. For ICMP traffic, the count does not include the network layer header, but the count does include the full ICMP message (ICMP header plus the data portion of the ICMP message).
network.recv.packets	uint64	Total packets received over the network connection.
network.sent.bytes.differential	uint64	Bytes sent over the network connection since the last network event generated for this connection.
network.sent.packets.differential	uint64	Packets sent over the network connection since the last network event generated for this connection.
network.recv.bytes.differential	uint64	Bytes received over the network connection since the last network event generated for this connection.
network.recv.packets.differential	uint64	Packets received over the network connection since the last network event generated for this connection.
network.timestamp	uint64	The timestamp of the event.

Object event (Windows only)

Object events are generated when the Windows kernel object manager creates a new process handle. This occurs when a user process opens a handle to another process. It also happens when a user process creates a child process. A typical use of an object event in a rule is to identify a process attempting to open a handle to a non-child process with access privileges that would allow it to manipulate the memory space of the target process. An example would be a process opening a handle to lsass.exe with access privileges that allow reading of memory from lsass. This is a technique sometimes used to steal credentials. The following table describes the object attribute, allowed values, and description for each object type.

Object attribute name	Allowed values	Description
object.operation	OBJECT_CREATE	A new object was created.
object.callingprocess	string	The full path to the executable associated with the process attempting to obtain a handle to another process, for example, \Device\HarddiskVolume1\Windows\explorer.exe or similar.
object.targetobjecttype	OBJECT_TYPE_PROCESS	The type of object being operated on.
object.targetprocess	string	The full path to the executable associated with the process whose handle is being obtained, for example, \Device\HarddiskVolume1\Windows\explorer.exe or similar.
object.accessmask	PROCESS_TERMINATE PROCESS_CREATE_THREAD PROCESS_SET_SESSIONID PROCESS_VM_OPERATION PROCESS_VM_READ PROCESS_VM_WRITE PROCESS_DUP_HANDLE PROCESS_CREATE_PROCESS PROCESS_SET_QUOTA PROCESS_SET_INFORMATION PROCESS_QUERY_INFORMATION PROCESS_SUSPEND_RESUME PROCESS_QUERY_LIMITED_INFORMATION	The access mask requested when creating the new object. These values are used to form a bitmask representing the access mask and are not enums.

Object attribute name	Allowed values	Description
	PROCESS_SET_LIMITED_INFORMATION PROCESS_SYNCHRONIZE PROCESS_ALL_ACCESS	
object.targetpid	pid	The process id of the process whose object is the target of the operation.
object.sourcepid	pid	The process id of the process performing the operation.
object.userid	string	Security identifier (SID) string representing the user associated with the object operation. Well-known SIDs are represented using the Microsoft Security Definition Language. Other SIDS use the standard string representation of a SID (S-*R*-*I*-*S*-*S*…).
object.timestamp	uint64	The timestamp of the event.

Print event (Windows only)

Print events are generated in response to the completion of a print job and are generated only for print jobs sent through print queues defined on the system where the agent is running.

The following table describes the print event attribute, allowed values, and description for each print event type.

Print event attribute name	Allowed values	Description
print.event	JOB_PRINTED	Indicates the type of print event.
print.totalbytes	uint32	Total bytes in the print job.
print.totalpages	uint32	Total pages in the print job.
print.printername	string	The name of the printer to which the document is printed.
print.machinename	string	The name of the machine from which the print job is initiated.
print.username	string	The user name of the user who initiated the print job.
print.documentname	string	The name of the document printed. The name usually corresponds to the name in the title bar of the application from which the document was printed.
print.timestamp	uint64	The timestamp of the event.

Process event (Windows, macOS, Linux)

Process events are generated in response to processes starting and processes terminating. In addition, when the endpoint agent starts up, the agent enumerates all existing processes on the system and generates events.

The following table describes the process attribute, allowed values, and description for each process event type.

Process attribute name	Allowed values	Description
process.state	PROCESS_EXITED PROCESS_STARTING PROCESS_STARTED PROCESS_EXISTING	Indicates the process state. • **PROCESS_STARTING**: Process is starting, subject to block rules. • **PROCESS_STARTED**: Process has started. • **PROCESS_EXITED**: Process has exited. • **PROCESS_EXISTING**: Used in process events generated for processes that were in existence at the time the endpoint agent started.
process.path	string	Indicates the full path of the executable associated with this process event, for example, \device\harddiskvolume1\windows\system32\calc.exe, or similar.
process.devicepath	string	The kernel device path associated with the executable, for example, \device\harddiskvolume1, or similar. This is only available on Windows.
process.parentpath	string	Indicates the full path of the executable associated with the parent process.
process.md5	md5digest	Indicates the MD5 digest of the executable associated with this process.
process.pid	pid	Process id for the process being created.
process.ppid	uint32	Process id for the parent process.
process.cmdline	string	Command-line string associated with the process.
process.usersid	string	Security identifier (SID) string representing the user associated with the process creation. Well-known SIDs are represented using the Microsoft Security Definition Language. Other SIDs use the standard string representation of a SID (S-*R*-*I*-*S*-*S*…). This is only available on Windows.
process.elevation	PROCESS_ELEVATION_UNKNOWN PROCESS_NOT_ELEVATED PROCESS_ELEVATED	Indicates whether the process is elevated. Windows considers a process to be elevated when running at an integrity level above medium. UAC elevated processes run with an integrity level of high.

Process attribute name	Allowed values	Description
process.timestamp	uint64	The timestamp of the event.

Registry event (Windows Only)

Registry events are generated in response to creating new registry keys, renaming existing registry keys, and setting values under registry keys.

The following table describes the registry attribute, allowed values, and description for each registry event type.

Registry attribute name	Allowed values	Description
registry.operation	REG_SET_VALUE REG_RENAME_KEY REG_PRE_CREATE_KEY_EX	Indicates the registry operation.
registry.valuetype	REG_NONE REG_SZ REG_EXPAND_SZ REG_BINARY REG_DWORD REG_DWORD_LITTLE_ENDIAN REG_DWORD_BIG_ENDIAN REG_LINK REG_MULTI_SZ REG_RESOURCE_LIST REG_FULL_RESOURCE_DESCRIPTOR REG_RESOURCE_REQUIREMENTS_LIST REG_QWORD REG_QWORD_LITTLE_ENDIAN	Indicates the type of data being written in the operation. This is REG_NONE for operations other than REG_SET_VALUE.
registry.valuesize	uint32	The size in bytes of the value being set in a REG_SET_VALUE operation.
registry.value.dword	uint32	The uint32 value being set in a REG_SET_VALUE operation when the registry.valuetype is REG_DWORD. This field is empty for other registry operation types.

Registry attribute name	Allowed values	Description
registry.value.qword	uint64	The uint64 value being set in a **REG_SET_VALUE** operation when the registry.valuetype is **REG_QWORD**. This field is empty for other registry operation types.
registry.value.string	string	The string value is set in a **REG_SET_VALUE** operation when the registry.valuetype is REG_SZ. This field is empty for other registry operation types.
registry.keyname	string	The full path of the registry key associated with the registry action. In the case of a REG_RENAME_KEY operation, this field represents the original key name.
registry.renamedkeyname	string	If the registry operation is **REG_RENAME_KEY**, this field contains the new name of the key. This field is empty for other registry operation types.
registry.valuename	string	Contains the name of the value being set or deleted in a **REG_SET_VALUE** or **REG_DELETE_VALUE** operation. This field is empty for other registry operation types.
registry.usersid	string	Security identifier (SID) string representing the user associated with the registry operation. Well known SIDs are represented using the Microsoft Security Definition Language. Other SIDs use the standard string representation of a SID (S-*R*-*I*-*S*-*S*…).
registry.pid	uint32	The process id associated with the registry activity.
registry.timestamp	uint64	The timestamp of the event.

Session event (Windows, macOS, Linux)

Session events are generated in response to logins such as console logins and RDP logins on Windows, and terminal logins and SSH sessions on macOS.

The following table describes the session attribute, allowed values, and description for each session event type.

Session attribute name	Allowed values	Description

session.event	SESSION_LOGON
	SESSION_LOGOFF
	SESSION_LOCK
	SESSION_UNLOCK
	SESSION_CONNECT
	SESSION_DISCONNECT
		Indicates the type of session event. **SESSION_LOGON** and **SESSION_LOGOFF** correspond to logging on and logging off. Both the Mac and Windows agents generate these alerts.
		The remaining types correspond only to Windows machines.
		• **SESSION_LOCK** and **SESSION_UNLOCK** correspond to locking and unlocking a Windows workstation.
		• **SESSION_CONNECT** occurs before the login to a new Windows session or when reconnecting to a pre-existing session using the Windows "Switch User" feature or Windows Terminal Services.
		• **SESSION_DISCONNECT** occurs on Windows when:
		  • A user logs out.
		  • The Windows “Switch User” feature is used to switch to another session.
		  • A Terminal Services session is disconnected (rather than logging out).

session.type	SESSION_TYPE_CONSOLE SESSION_TYPE_REMOTE SESSION_TYPE_SERVICE SESSION_TYPE_TERMINAL	Indicates the type of session. • **SESSION_TYPE_CONSOLE** indicates a console logon (a logon to the desktop of a Windows or macOS machine). • **SESSION_TYPE_REMOTE** indicates a remote login such as RDP logins or SSH logins. • **SESSION_TYPE_SERVICE** is used for logons performed by Windows services. • **SESSION_TYPE_TERMINA**L is used for macOS and represents a new terminal login, typically associated with a new instance of a terminal window.
session.domain	string	The domain name associated with the session event. This is only available on Windows.
session.username	string	The user name associated with the session event.
session.locked	TRUE FALSE	Indicates the current locked state of the session.

Session attribute name	Allowed values	Description
session.preexisting	TRUE FALSE	When **TRUE**, indicates that this session event corresponds to a session that was pre-existing when the endpoint agent started. For example, the endpoint agent generates session events with an event value of **SESSION_LOGON** for each existing session found when the endpoint agent starts. This field is set to **TRUE** for each of those events in that scenario. Currently, only the Windows agent generates events for pre-existing sessions. The macOS agent does not.
session.ipaddr	ipaddr	When session.type is **SESSION_TYPE_REMOTE**, this attribute indicates the IP address of the remote client.
session.timestamp	uint64	The timestamp of the event.

Thread event (Windows only)

Thread events are generated each time a thread is started or exits.

Thread attribute name	Allowed values	Description
thread.state	THREAD_EXITED THREAD_STARTED	Indicates whether the thread event is for a start or exit.
thread.pid	pid	The process id for the process in which the thread is running.
thread.timestamp	uint64	The timestamp of the event.
thread.tid	utin32	The thread id.

URL event (Windows, macOS, and Linux)

URL events are generated whenever a user clicks a link or enters a URL into the browser's address bar when using the Edge, Chrome, Firefox and Vivaldi web browsers. URL events effectively mirror the entries recorded in the browser's history file. If a user types in a URL that does not resolve, it will not show up in the URL events.

The following table describes the events attribute, allowed values, and description for each URL event type.

Clipboard attribute name	Allowed values	Description
url.pid	pid	The process id for the browser used to visit the URL.
url.visit_time	uint64	The timestamp corresponding to the time the URL was visited.
url.url	string	The URL visited. For example, "http://www.nuix.com".
url.usersid	string	Security identifier (SID) string representing the user associated with the URL operation. Well known SIDs are represented using the Microsoft Security Definition Language. Other SIDs use the standard string representation of a SID (S-*R*-*I*-*S*-*S*…).
url.title	string	The title of the web page visited. Corresponds with the text that is typically shown in the browser's title bar or in the page's tab.
url.timestamp	uint64	The timestamp of the event.

Process state rule attributes

The endpoint agent automatically populates several entries in the process state database for each process started on the endpoint. These attributes mostly pertain to information about the executable.

The following table describes the process state attributes, allowed values, description, and platform that can be specified in a rule.

Process state attribute	Allowed values	Description	Platform
curproc.path curproc.parent.path	string	The path of the executable associated with the current or parent process.	Windows macOS
curproc.exe.md5 curproc.parent.exe.md5	md5digest	The MD5 digest of the executable associated with the current or parent process.	Windows macOS
curproc.exe.sha256 curproc.parent.exe.sha256	sha256digest	The SHA256 digest of the executable associated with the current or parent process.	Windows macOS
curproc.exe.sig curproc.parent.exe.sig	SIG_STATUS_UNKNOWN SIG_STATUS_NO_SIG SIG_STATUS_VALID SIG_STATUS_INVALID	The signature status of the executable associated with the current or parent process. • **SIG_STATUS_UNKNOWN**: Endpoint failed at attempting to verify the presence or validity of the signature on the executable. The status is unknown. • **SIG_STATUS_NO_SIG**: No signature was present. • **SIG_STATUS_VALID**: A signature is present and valid. • **SIG_STATUS_INVALID**: A signature is present but invalid.	Windows macOS
curproc.exe.importlist curproc.parent.exe.importlist	string	This attribute contains the list of functions imported by the executable in the format "MODULENAME!FunctionName", for example, "KERNEL32.CreateFileW". For more information about this attribute, see Matching Against Lists.	Windows
curproc.exe.modulelist curproc.parent.exe.modulelist	string	This attribute contains the list of modules from which the executable has imported functions in the format "MODULENAME", for example, “KERNEL32.DLL”. For more information about this attribute, see Matching Against Lists.	Windows
curproc.exe.entropy.pe	double	Indicates the entropy calculated over the entire Portable Executable (PE) file.	Windows

Process state attribute	Allowed values	Description	Platform
curproc.parent.exe.entropy.pe			
curproc.exe.entropy.highestsection curproc.parent.entropy.highestsection	double	Indicates the highest entropy value calculated when calculating entropy over each individual section in the PE.	Windows
curproc.exe.entropy.highestcodesection curproc.parent.exe.entropy.highestcodesection	double	Indicates the highest entropy value calculated when calculating entropy over each individual section marked as containing the code in the PE.	Windows
curproc.exe.peagehours curproc.parent.exe.peagehours	double	This field is populated once when the process starts running. This field indicates the age in hours of the PE calculated by subtracting the PE linker timestamp from the current system time. This field could be negative if the timestamp is in the future.	Windows
curproc.exe.peagedays curproc.parent.exe.peagedays	double	This field is calculated once when the process starts running. This field indicates the age in days of the PE calculated by subtracting the PE linker timestamp from the current system time. This field could be negative if the timestamp is in the future.	Windows
curproc.exe.overlaypresent curproc.parent.exe.overlaypresent	bool	There is overlay data present in the executable associated.	Windows
curproc.exe.sectionamesuspect curproc.parent.exe.sectionamesuspect	bool	One or more section names in the executable are null or contain non-ASCII characters.	Windows
curproc.exe.arch curproc.parent.exe.arch	PE_ARCH_32 PE_ARCH_64	The architecture of the executable.	Windows
curproc.exe.heuristic_packer curproc.parent.exe.heuristic_packer	double	Indicates the probability that the executable is packed based on heuristics.	Windows
curproc.exe.importdllcount curproc.parent.exe.importdllcount	uint64	The number of DLLs from which the PE imports functions.	Windows
curproc.exe.importfunctioncount curproc.parent.exe.importfunctioncount	uint64	The number of functions that the PE imports.	Windows

Process state attribute	Allowed values	Description	Platform
curproc.exe.pe_parse_err curproc.parent.exe.pe_parse_err	bool	Indicates that an error occurred when parsing the PE. This may indicate packing.	Windows
curproc.exe.timestampsuspect curproc.parent.exe.timestampsuspect	bool	This field indicates that the PE linker timestamp of the executable represents a time in the future or a time before 1980. The field is populated based on a comparison of the PE linker timestamp to the current system time at the time the executable started running.	Windows

Insider threat rules

InsiderThreat.efl contains rules centered on tracking suspicious insider activity. The rules are "parameterized" and configured using variables in CommonDefs.efl.

General configuration

Several configuration items are used across multiple categories of rules, including the workdays and work hours used to define business hours.

Example:

```
global uint32 workdays = MON | TUES | WEDS | THURS | FRI;
global time_range workhours = timerange("07:00","17:30");
```

Valid values for workdays are SUN, MON, TUES, WEDS, THURS, FRI, SAT. The hours specified in work hours are in a 24-hour format.

Removable media usage

The following topics are discussed in this section:

- Off Hours Removable Media Usage
- General Removable Media Usage
- Specific File Writes to Removable Media (Windows Only)

Off-hours removable media usage

To generate alerts for any insertion/removal/write to/execute from a removable media device during off-hours, set the following variable to **TRUE**.

```
global bool bAlertOffHoursUSBUsage = false;
```

General removable media usage

To generate alerts for removable media activity regardless of the time of day, set one or more of the following variables to true. On Windows endpoints, the following variables can be set to true to enable alerts for file writes to USB removable media devices or execution of programs from USB removable media devices.

```
global bool bAlertUSBWrite = false;
global bool bAlertUSBExecution = false;
```

On Windows and macOS endpoints, the following variable can be set to true to enable alerts when USB removable media is inserted or removed.

```
global bool bAlertUSBInsertionRemoval = false;
```

Specific file writes to removable media (Windows only)

The writing of specific files to a removable media device based on path or MD5 can be monitored.

File Write by Path

To generate alerts when files with a specific name or path are written to a removable media device, set the following variable to **TRUE**.

```
global bool bAlertUSBWriteOnName = false;
```


In addition, the following variable must be populated with the substrings to match the file path.

```
global string USBFilePathSubStringList[]=
 {
 "example_file1.txt", # file name
 "confidential"       # partial name
 };
```

The example matches any file writes to a removable media event in which the substrings "example_file1.txt" or "confidential" appear in the full pathname of the file being written. Matching is done in a case-insensitive fashion.

File Write by MD5 (Windows Only)

To generate alerts when files with specific MD5 hash values are written to a removable media device, set the following variable to **TRUE**.

```
global bool bAlertUSBWriteOnMD5 = false;
```

In addition, the following variable must be populated with the MD5 hash digests to alert on.

```
global md5digest USBMD5WriteList[]=
{
 md5("d41d8cd98f00b204e9800998ecf8427e"), # example hash
 md5("9d7237897b71230aa001c07ed7138810")
};
```

Printing (Windows only)

The following topics are discussed in this section:

- Alert on Window Document Title
- Alert on Off-hours Printing
- Alert on High-volume Printing

Alert on window document title

Set the following variable to **TRUE** to generate alerts when printing is performed in a window with a particular window title.

```
global bool bAlertPrintingOnWindowTitle = false;
```

Populate the following list with substrings to search for in the window titles of applications performing printing. The strings in this list are searched for as substrings in the window title of the application performing the printing. Often, the window title includes some portion of the file name or path being viewed in the application, or the URL or title of the website being visited. Matching is done in a case-insensitive fashion.

```
global string PrinterWindowTitleList[]=
 {
 "example1.doc", # example file
 "Salesforce"
 };
```

The example matches printing from an application where the window title contains the string "example1.doc" or the string "Salesforce".

Alert on off-hours printing

Set the following variable to **TRUE** to generate alerts in response to any printing activity.


```
global bool bAlertOffHoursPrinting = false;
```

Alert on high-volume printing

Set bAlertPrintHighDocumentCount to true to generate alerts in response to a certain number of documents printed within a time frame.

Example:

```
global bool bAlertPrintHighDocumentCount = false;
```

Adjust PrintHighDocumentCount and PrintHighDocumentTimeWindowMs to define the alert threshold. Alerts are generated when PrintHighDocumentCount documents are printed within PrintHighDocumentTimeWindowMs milliseconds.

Example:

```
global uint32 PrintHighDocumentCount = 10;
global uint32 PrintHighDocumentTimeWindowMs = 60000;
```

Network connections and traffic

The variable HomeAddrList is used to describe IP addresses and subnets considered part of the home or trusted network.

```
global ipaddr HomeAddrList[] =
{
 ip("192.168.1.0/24"),  # example subnet
 ip("192.168.1.1")      # example IP
};
```

Alert on remote connections

Set the following variable to **TRUE** to generate alerts for any network connection successfully established to hosts not identified in the HomeAddrList variable.

```
global bool bAlertRemoteConnections = false;
```

Alert on off-hours remote connections

Set the following variable to **TRUE** to generate alerts for any network connection successfully established during off-hours to hosts not identified in the HomeAddrList variable.

```
global bool bAlertOffHoursRemoteConnections = false;
```


Alert on watchlisted hosts

To define the watchlisted host and network addresses, update the following variable.

```
global ipaddr IpWatchList[] =
{
 ip("162.244.31.0/24"), # example subnet
 ip("162.244.31.128") # example IP
};
```

To generate alerts in response to successful connections made to IP addresses present on the IpWatchList, set the following variable to **TRUE**.

```
global bool bAlertNetworkWatchList = false;
```

Alert on data exfil

Set the following variable to true to generate alerts when a certain amount of data has been transferred over the network to a non-home address.

```
global bool bAlertExfil = false;
```

Set the following variable to define the threshold (in bytes) for data transfer, after which the alert will be generated.

```
global uint64 alertExfilLimit = 10485760;
```

DNS queries (Windows only)

The following topics are discussed in this section:

- Alerting on Network Traffic to Watchlisted Domain Names
- Alerting on Watchlisted Name Queries
- Initiating Screenshots on Watchlisted Name Queries

Alert on network traffic to watchlisted domain names

Alerts can be generated when network traffic is initiated to an IP that maps to a particular fully qualified domain name or subdomain name. To define watchlisted DNS names for alerting of network traffic, edit the following variable.

```
global string domainTrafficWatchList[] =
{
    ".dropbox.com",
    ".ru"
};
```

Fully qualified domain names can be specified as “www.google.com”. Subdomains can also be used and must include a “.” to the left of the name, as in “.google.com” or “.com”). To enable the generation of alerts whenever network traffic is initiated to a watchlisted DNS name, set the following variable to **TRUE**.

```
global bool bAlertDNSQueryWatchList = false;
```


Alert on watchlisted name queries

To define watchlisted DNS names for alerting, edit the following variable.

```
global string dnsQueryWatchList[] =
 {
 "prjcode.com",
 "la21jeju.or.kr",
 "reltime2012.ru",
 "perugemstones.com"
 };
```

To generate alerts whenever a watchlisted DNS name is queried, set the following variable to **TRUE**.

```
global bool bAlertDNSQueryWatchList = false;
```

Initiate screenshots on watchlisted name queries

To define watchlisted DNS names for screenshotting, edit the following variable.

```
global string dnsQueryScreenshotWatchList[] =
{
"icanhas.cheezburger.com" # examples
};
```

The following variable enables the capturing of screenshots when a screenshot watchlisted name is queried.

```
global bool bDNSQueryScreenshotWatchList = false;
```

The following variables define how many screenshots are captured and over what period of time.

```
global uint32 dnsQueryScreenshotCount = 5;
global uint32 dnsQueryScreenshotPeriodSeconds = 25;
```

The previous example captures five screenshots spaced five seconds apart from each other.

Keystroke activity (Windows only)

To generate alerts in response to keystroke activity based on a keystroke watch list, set the following variable to **TRUE**.

```
global bool bKeystrokeWatchList = false;
```

To define watchlisted words, edit the following variable.

```
global string keystrokeWatchList[] =
 {
 "this is an example",
 "explosives",
 "company secrets"
 };
```

Matching is done in a case-insensitive fashion.

Clipboard activity (Windows only)

Alert on text pastes by size

To generate alerts in response to pastes of text data exceeding a particular size threshold in characters, set the following variable to **TRUE**.

```
global bool bAlertClipboardTransferThreshold = true;
```

To set the threshold in characters, edit the following variable.

```
global uint64 ClipboardTransferThresholdChars = 512;
```

Alert on pastes to watchlisted applications

To generate alerts in response to pastes of data into particular watchlisted applications, set the following variable to **TRUE**.

```
global bool bAlertClipboardPasteWatchList = true;
```

To define the watchlisted applications, edit the following variable.

```
global string ClipboardPasteWatchList[] =
{
  "skype.exe",
  "teams.exe"
};
```

URL visitation (Windows only)

To generate alerts in response to visitation of URLs based on a URL watch list, set the following variable to **TRUE**.

```
global bool bAlertURLWatchList = false;
```

To define the watchlisted URLs, edit the following variable.

```
global string URLWatchList[] =
{
"http://www.google.com",
"www.google.com",
```



```
"google.com"
};
```


Appendix: Keystroke event special character notation

The keystroke.keydata field in the keystroke event represents non-alphanumeric characters and control sequences using a special notation. For example, a print screen key is represented by the string “[PRNTSCRN]”. A function key, such as F1, is represented as “[F1]”. The list below shows the notations used for various non-alphanumeric keys.

- [BS]
- [PAUSE]
- [ESC]
- [PGUP]
- [PGDN]
- [END]
- [HOME]
- [UP]
- [LEFT]
- [RIGHT]
- [RIGHT]
- [DOWN]
- [PRNTSCRN]
- [INSERT]
- [DEL]
- [WIN]
- [APPS]
- [NUMLOCK]
- [SCRLLCK]
- [F%u] (eg. "[F1]")

The list below shows the notations used for various control sequences.

- [CTRL+%c] (eg. "[CTRL+V]")
- [CTRL+%u] (eg. "[CTRL+1]")
- [CTRL+F%u] (eg. "[CTRL+F1]")
- [ALT+%c] (eg. "[ALT+X]")
- [ALT+%u] (eg. "[ALT+1]")
- [ALT+F%u] (eg. "[ALT+F1]"
- [WIN+%u] (eg. "[WIN+1]")

The rule samples below demonstrate how to match against special keys and control sequences. These rules match against various key sequences used to cut, paste, copy, and screenshot.

```
screenshot (keystroke.pid, 2, 10) when  stristr (keystroke.keydata, "[WIN+SHIFT+S]");
screenshot (keystroke.pid, 2, 10) when stristr (keystroke.keydata,"[CTRL+C]");
screenshot (keystroke.pid, 2, 10) when stristr (keystroke.keydata,"[CTRL+X]");
screenshot (keystroke.pid, 2, 10) when stristr (keystroke.keydata,"[PRNTSCRN]");
screenshot (keystroke.pid, 2, 10) when stristr (keystroke.keydata, "[ALT+PRNTSCRN]");
screenshot (keystroke.pid, 2, 10) when stristr (keystroke.keydata, "[CTRL+V]");
screenshot (keystroke.pid, 2, 10) when stristr (keystroke.keydata,"[SHIFT+INSERT]");
```

