MBA VARIABLES

Each not void variable is computed by the following relation:
Variable = Function1(Input1) Operator Function2(Input2).
A not void variable is a variable that does not call any void functions.
A void function is a function that executes some code but does not return any result.
For example the function ‘SetField’ does assign some value to the input selected for the function, but does not assign any value to the variable in which ‘SetField’ is declared.
Although void variables can be called by other variables, it’s pretty useless to do so since void variables will return always an empty string.
Void variables are computed in this way:
- Function1(Input1)
- Function2(Input2)

i.e. void variables correspond up to two lines of code IF two void functions are declared.
You cannot put in a variable both void functions and not void functions, so in the end you can have two types of variables in MBA:
1) Void variables: variables that contain at least 1 void function. They execute some code but do not return any result.
2) No void variables: variables that do not contain any void function. They execute some code and are assigned to the value defined by the code execution. They can be reused by other variables to further process the result.
A variable can be computed either in the login page stage or in the OCR stage or in the intermediate action stage or in the post action stage or in the form redirect stage or in the additional redirect stage.
The value computed can be used to modify all MBA stages, except of course the login page stage.
In each stage the variables are computed until execution is completed (i.e. all the stage variables have been computed) or a parsing error has been issued.
A parsing error can be issued either by a parsing input (see inputs context help for details) or by the void procedure TriggerError (see functions context help for details).
If a parsing error is issued, then the flow of variable computations will be established by the variable for which the parsing error occurred.
If the variable is a conditional variable (i.e. ‘ignore’ option checked) then variables computation will continue.
If the variable is not a conditional variable (i.e. ‘ignore’ option not checked, default setting) then variables computation will stop and the bot will generate a 404 parsing error.
Moreover if the variable for which the parsing error occurred has the option ‘Stop’ checked, then all variables will be skipped until the resuming variable, i.e. a variables for which ‘Resume’ option is checked, is found.
Finally a variable for which option ‘Loop’ is checked will mark a jump index that points to the variable itself.
After a jump index has been stored, a jump event, that can be triggered by the void procedure ‘Jump’, will cause the variables execution to be restored from the jump index.
It is also possible to nest multiple loops.

VARIABLES STAGES

The stage tells Sentry when the variable has to be computed.
If the stage for the variable is set to login page, the variable is computed just after the login page URL has been called.
f the stage for the variable is set to OCR stage, the variable is computed just after the capctha image has been dwonloaded AND recognized.
If the stage for the variable is set to intermediate action, the variable is computed just after the intermediate action URL has been called.
If the stage for the variable is set to post action, the variable is computed just after the post action URL has been called.
If the stage for the variable is set to redirect URL page, the variable is computed just after the redirect URL has been called.
The only variables that would need the redirect URL stage, are the ones used to configure by variables the additional redirect URLs.
A variable is computed by using the status of the stage for which the variable has been defined.
For example, if the stage for a variable is set to login page, then the variable will use, if needed, the HTML Source of the login page and the value of Cookie set by the login page, if any.
Finally if the stage for the variable is set to Additional Redirect, the variable is computed for each additional redirect call.

VARIABLES FUNCTIONS

<- SetParameterIndex(Input) -> Void procedure ->
Set the index of the parameter that will be assigned by SetParameterValue.
Parameters indexes for multi-input functions always start from zero. SetParameterIndex and SetParameterValue have to be used always in pair and in this order.

<- SetParameterValue(Input) -> Void procedure ->
Set the value of the parameter pointed by the index set by SetParameterIndex.
Parameters indexes for multi-input functions always start from zero. SetParameterIndex and SetParameterValue have to be used always in pair and in this order.

<- SetField(Input, Value) -> Void procedure ->
Set the field pointed by Input to Value
Value has to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- TriggerError(Input) -> Void procedure ->
Triggers a parsing error if Input is True.
Input has to be a boolean string, i.e. ‘0’ (False) or ‘1’ (True).

<- Jump(Input) -> Void procedure ->
Triggers a jump to the last loop variable found if Input is True.
Input has to be a boolean string, i.e. ‘0’ (False) or ‘1’ (True).

<- Exit(Input) -> Void procedure ->
Aborts variables computation if Input is True, i.e. "1".

<- TriggerKeyMatch(Input) -> Void procedure ->
Triggers a key match by the following schema:
Input = 0 -> Success.
Input = 1 -> Ban, 2 -> Conditional Ban, 3 -> Blacklist Ban, 4 -> Good OCR Code Ban.
Input = 5 -> Failure, 6 -> Bad User Failure, 7 -> Good User Failure, 8 -> Expired Combo Failure.
Input = 9 -> Normal Retry, 10 -> Bad Retry, 11 -> Bad OCR Code Retry.

<- SetCombo() -> Void procedure ->
Set the BOT combo to the one set by variables with SetField.

<- SetVectorElement(Input, Index, Value) -> Void procedure ->
Set the element Index of Vector Input to Value. i.e. Input[Index] = Value. Index is zero based.
Index and Value have to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- ConfigureIAURL(Input, URLData, PostData, CookieData, HeaderData, HTTPMethod) -> Void procedure ->

Set URL parameters for the Intermediate Action URL pointed by Input. Only IA URLs whose index is greater than 1 can be configured in this way.
The Intermediate Action URL will be built with the parameters set by URLData, PostData, CookieData, HeaderData.
These parameters act exactly as the ones that can be assigned by the Post Element Menu.
HTTPMethod defines how the URL will be called: 0 -> Head, 1 -> GET, 2 -> POST, 3 -> MultiForm POST, 4 -> JSON POST.
All the parameters except INPUT have to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- AddRedirectURL(Input, URLData, PostData, CookieData, HeaderData, HTTPMethod) -> Void procedure ->

Adds to the already defined chain of additional redirect URLs an additional redirect URL of value Input.
The additional redirect URL will be built with the parameters set by URLData, PostData, CookieData, HeaderData.
These parameters act exactly as the additional redirect URL parameters that can be assigned by the Post Element Menu.
HTTPMethod defines how the URL will be called: 0 -> Head, 1 -> GET, 2 -> POST, 3 -> MultiForm POST, 4 -> JSON POST.
All the parameters except INPUT have to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- AddCapturedKeys(Input, KeyName) -> Void procedure ->

Add Input to the list of captured keys. KeyName is the name assigned to the captured keys and has to be defined with the functions pair SetParameterIndex and SetParameterValue.

<- HMac(Input, Key, HashSelect, TrimBytes) -> Hexadecimal string ->

Crypts Input with Key (must be hexadecimal) by HMAC algorithm. Inner hash function is selected by HashSelect. Output is trimmed by TrimBytes (>0 -> trim from left, <0 -> trim from right).
HashSelect=0->MD5, 1 ->SHA1, 2->SHA224, 3->SHA256, 4->SHA384, 5->SHA512, 6->SHA512_224, 7->SHA512_256.Key,
HashSelect and TrimBytes have to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- RSAPKCS15(Input, KeyMod, KeyExp) -> Hexadecimal string ->

Crypts Input with public Key KeyMod(modulus):KeyExp(exponent) (have to be hexadecimal) by RSA PKCS 1.5. KeyMod and KeyExp have to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- MD5Hash(Input) -> Hexadecimal string ->

Computes the MD5 hash of a string.

<- SHA(Input, HashSelect) -> Hexadecimal string ->
Computes the SHA hash of a string by using the SHA algorithm set by HashSelect.
HashSelect=1->SHA1, 2->SHA224, 3->SHA256, 4->SHA384, 5->SHA512, 6->SHA512_224, 7->SHA512_256.
HashSelect has to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- PadHex(Input, PadDirection) ->
Pads a hexadecimal string with a zero only if the number of chars is odd.
If PadDirection = 0, the zero will be added from the right, if it is 1 then the zero will be added from the right (i.e. the equivalent binary data will be shifted by four bits to the left -> the equivalent integer will be multiplied by 16).
PadDirection has to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- HexTo64(Input) -> Base64 string ->
Convert a hexadecimal string to a base64 string.

<- 64ToHex(Input) -> Hexadecimal string ->
Convert a base64 string to a hexadecimal string.

<- HexToString(Input) -> ANSI string ->
Convert a hex string to an ANSI string.

<- StringToHex(Input) -> Hexadecimal string ->
Convert an ANSI string to a hex string.

<- Encode64(Input) -> Base64 string ->
Convert an ANSI string to a Base64 string.

<- Decode64(Input) -> ANSI string ->
Convert a base64 string to an ANSI string.

<- StringLength(Input) -> integer string ->
Gives the number of chars in a string.

<- StringPos(Input,StrToSearch,StartPos) -> integer string ->
Gives the starting position of the string StrToSearch in the string Input searching from position StartPos.
Initial position is 1. The function returns zero if StrToSearch is not found in Input.
StrToSearch and StartPos have to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- StringExtract(Input,StartPos,Length) -> ANSI string ->
Extract a substring of Length chars from Input starting from StartPos.

StartPos and Length have to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- StringGen(Input,N) -> ANSI string ->
Joins Input by N times.
N has to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- StringReplace(Input,StringToFind,StringToReplace) -> ANSI string ->
Replaces all occurrences of StringToFind in Input with StringToReplace.
StringToFind and StringToReplace have to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- StringInsert(Input, NewString, StartPos, Width) -> ANSI string ->
Insert NewString in the Input string starting from position StartPos deleting Width (can be also zero) characters from the Input string starting from the same position.
NewString, StartPos and Width have to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- Lowercase(Input) -> ANSI string ->
Converts all chars in Input to their lowercase version.

<- Uppercase(Input) -> ANSI string ->
Converts all chars in Input to their uppercase version.

<- URLEncode(Input) -> ANSI string ->
Encodes Input by using % encoding scheme.

<- LeftShiftString(Input) -> ANSI string ->
Shits to the start of the resulting string all chars whose ASCII code is not multiple of 2.

<- RightShiftString(Input) -> ANSI string ->
Shits to the end of the resulting string all chars whose ASCII code is not multiple of 2.

<- Ceil(Input) -> Integer string ->
Converts a float number to the next higher integer, i.e. Ceil(‘2.5’) = 3.

<- Floor(Input) -> Integer string ->
Converts a float number to the next lower integer, i.e. Floor(‘2.5’) = 2.

<- SetVectorSize(Input) -> vector of ANSI strings ->
Generates a vector of Input elements. Vector elements are initialized to "Empty".'

<- GetVectorSize(Input) -> Integer string ->
Gets the number of elements of vector Inputs.'

<- GetVectorElement(Input, Index) -> ANSI string ->
Get the Index-th element of vector Input, i.e. Input[Index]. Indexes of vectors are zero based.
Index has to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- Compute(Input) -> Integer string ->
Executes the mathematical operations underlined in Input, i.e. Compute(‘(2*5)+2’) = 12.

<- Compare(Input, CompareTarget, Operator) -> Boolean string ->
Compares Input against CompareTarget and returns the result of the operation as a boolean string, i.e. ‘0’ (False) or ‘1’ (True), by executing the comparison outlined by Operator.
Operator = 0 -> ‘<‘, 1 -> ‘<=‘, 2 -> ‘==‘, 3 -> ‘>=‘, 4 -> ‘>‘, 5 -> ‘<>‘.
CompareTarget and Operator have to be assigned with the functions pair SetParameterIndex and SetParameterValue.

<- Negate(Input) -> Boolean string ->
Executes Not(Input). Input has to be a boolean string, i.e. ‘0’ (False) or ‘1’ (True).

VARIABLES INPUTS

- User: is the Username of the Combo being tested.
- Pass: is the Password of the Combo being tested.
- Email: is the Email of the Combo being tested.
- Captcha: is the OCR code assigned by the OCR stage to the captcha image.
- UnixTime: is the actual time expressed in Unix format, i.e. seconds elapsed from 1 Jan 1970.
- RandomNumber: is a random float number between 0 and 1 (always <1).
- EmptyString: the name says it all.
- IANumber: the index of the Intermediate Action currently active.
- ARUNumber: the index of the Additional Redirect currently active.
- IsFingerPrint: true (i.e. "1") if the active combo is a random generated one.'
- Constant: it is a constant. The value of the constant must be specified in the input setting text box.
- Cookie: it outputs the cookie value of the cookie with the name equal to the one set in input setting text box. For example if the actual Coookie is equal to "Lang=eng; SessionID=463487328" and you set this input to SessionID, you'll get "463487328".
- ParsingCode: it outputs the string extracted from the actual stage body answer with the parsing code you can build by clicking on the wizard button that will be activated when you select this input. The Parsing Code will be used against the HTML Source of the stage for which the variable has been defined.
- HParsingCode: It acts like ParsingCode, but instead, the parsing code is processed against the stage Headers.

Finally you'll be able to use as input any variable already defined.
The Inputs Cookie, ParsingCode and HParsingCode will trigger a parsing code error if the matched string is empty.

VARIABLES OPERATORS

- "+,-,/,*": these are the basic mathematical operators - Inputs have to be numbers or an empty string will be returned.
- &: this is the join string operator - If the Input strings are both string vectors, they have to be of the same size or an empty string will be returned.
- AND, OR, XOR, NAND, NOR, NXOR: these are the basic boolean operators. Inputs have to be boolean strings (‘0’ for False and ‘1’ for True) or an empty string will be returned.

VARIABLES ASSIGNEMENT

You tell Sentry how a stage configuration has to be modified based on the computed variables. You can modify the following parameters for OCR stage, Intermediate Action stage, Post Action stage, Form Redirect Stage and Additional Redirect Stages:

- POST Data (let's call POSTDATA the value of this parameter before it is changed by variables).
- Action URL (let's call URL the value of this parameter before it is changed by variables).
- Cookie (let's call COOKIE the value of this parameter before it is changed by variables).
- Header (let's call HEADER the value of this parameter before it is changed by variables).

For example, if you have assigned one of these parameters to a variable named "VAR", which is equal respectively to Bruteforcer=MBA&coder=astaris, MBA.html, session = MBA_OCR and UserAgent = MBA, you will get the following behaviour:

If the variable is assigned to POST Data, the final POST Data parameter will be computed as "POSTDATA&Bruteforcer=MBA&coder=astaris".

- If the variable is assigned to Action URL, the final Action URL parameter will be computed as "URLMBA.html".
- If the variable is assigned to Cookie, the final Cookie parameter will be computed as "COOKIE; session = MBA_OCR".
- If the variable is assigned to Header, the final Cookie parameter will be computed as "HEADER\nUserAgent = MBA" (with \n line break control char).

ADDITIONAL REDIRECT STAGES

You can configure MBA additional redirect URLs too.
An additional redirect URL is called after a success key match, either upon post or after the form redirect stage.
So you can have two behaviours:
- Post -> success key match -> call additional redirect URLs.
- Post -> redirect key match -> get form redirect URL -> success key match -> call additional redirect URLs.

You can define how many additional URLs you want and MBA will capture keys from each body received.
However if after the call to all defined additional redirect URLs no keys have been captured, the combo will be moved to the "To Check" tab.
Otherwise the combo will be marked as a Hit and the keys captured saved.
Moreover you can link each additional redirect URL to MBA variables.
In particular you can assign the URL and the POST data parameters.
If you assign the POST data parameter to a variable, the additional redirect URL will be called with POST method.
By linking the additional redirect URLs to MBA variables, you'll get a conditional redirect URL.
This means that the additional redirect URL will be called only if the variables assigned to it have been successfully computed.
This is a feature that you must use when you don't know how many additional redirect URLs have to be called in order to get the full account details of the combo being tested.