Baymax Patch Tools Help Documentation

Full guide to hijacking patches

Ver 1.5

January 2023

Catalog

I. Overview of tools

(I) Software Introduction

Baymax Patch Tools (Baymax for short) implements dynamic patching of the target process by loading the functional module PYG.DLL/PYG64.DLL with the hijacked DLL released by the target process. baymax not only supports dynamic modification of the instructions and data of the target module, but also takes over the exception handling of process r3, simulating the exception handling of the debugger It supports setting breakpoints on the target module address and modifying its corresponding registers, flag registers, and memory data pointed to by registers after interruption, so as to achieve cracking and passwords on the target file without destroying it.

The tool has a shell protection, antivirus software may misreport the tool and patch files! Due to the use of the shell SDK, all components of the tool (including the generated patches) do **not contain networking capabilities**! The generated patches will not modify any files on the system when running (except for overwriting patch files). The generated patches will not modify any files on the system

when running (except for overwriting patch files). The tool itself has a verification mechanism and will load only after the module is successfully verified at startup, but for security reasons, please be sure to download and use it from the official site.

(II) Scope of application

Baymax is suitable for Win x86/x64 platform. Has been adapted to WinXP, Win7, Win8, Win8.1, Win10, Win11 and other systems, theoretically patching all processes that can load hijacked modules, supporting some .

(C) How to test

Baymax is intended for quick validation of patching solutions and is limited to technical exchanges only.

Baymax's function module is PYG.DLL/PYG64.DLL. The function module has a simple file name checksum built in and will not perform patching operations after modification. So, you can check whether the file exists in the process directory or check whether the process is loaded with the module as a basis for detection.

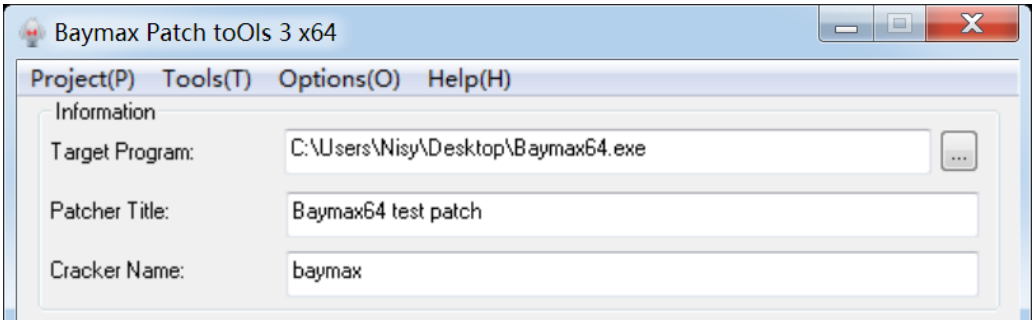
(iv) Disclaimer

This tool is for technical exchange only, please do not use it for commercial or illegal purposes! All effects caused by the patch files you create are not the responsibility of the author! When you click on the OK disclaimer when using the tool, you are agreeing to the terms and conditions.

II. Creating a patch project

(i) Setting the target process

The patch project first selects a target file. The target program set in the interface will be used as a reference for the generated patch program to intelligently release the hijacked file. If the patcher does not find the file, a pop-up box will prompt the user to select the target object.

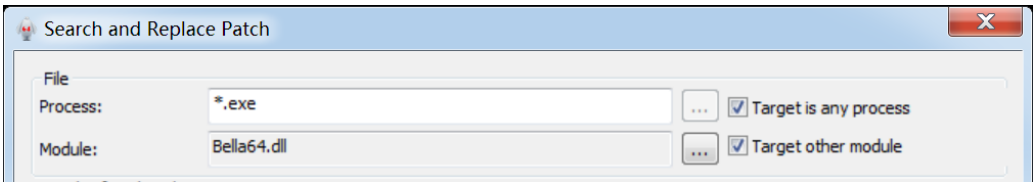


(ii) Adding patch items and setting up patch processes and modules

Add a search and replace or abnormal interrupt patch item.

The target process set in the project is the default process and patch module for each patch item, which can be adjusted by yourself.

Baymax supports patching projects containing multiple patch items, each of which can be set to patch a different process, or to patch all processes (check the target as any process), and supports patching different DLLs or other loaded modules of a process (set the target as other modules, and you can set one module loaded by that process in each patch item).

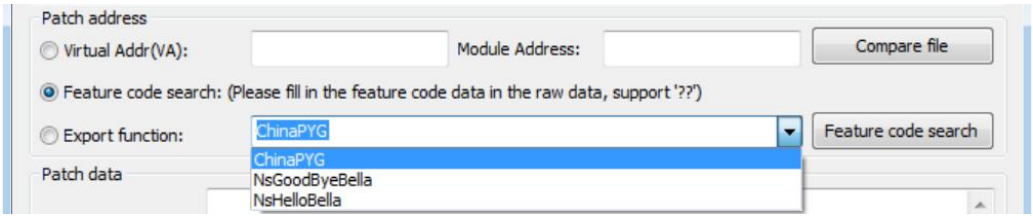


When you check the target for other modules, you don't need to care when the module is loaded, whether it will be unloaded, or loaded again, if the patch module is not an exe process, Baymax will monitor the module loading and try to patch it by default. Of course you can also handle the timing of the module patching yourself by selecting "Customize Patch DLL Timing" on the menu to turn off the smart monitoring feature.

(iii) Set the patch address

1. Offset patch

The "Search and Replace Patch", referred to as the offset patch, supports the following three options for setting the patch address.



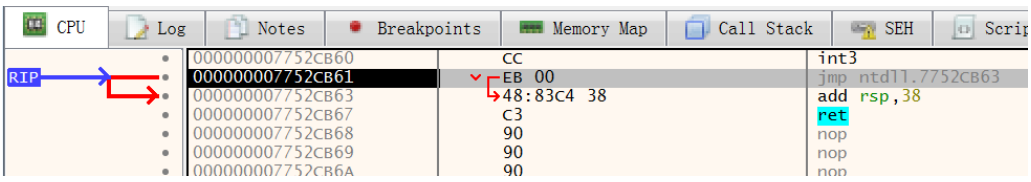
(1) Enter the patch virtual address (memory address) and the module base address

(2) Locating patch addresses by feature code

(3) Locate the patch address by exporting the function

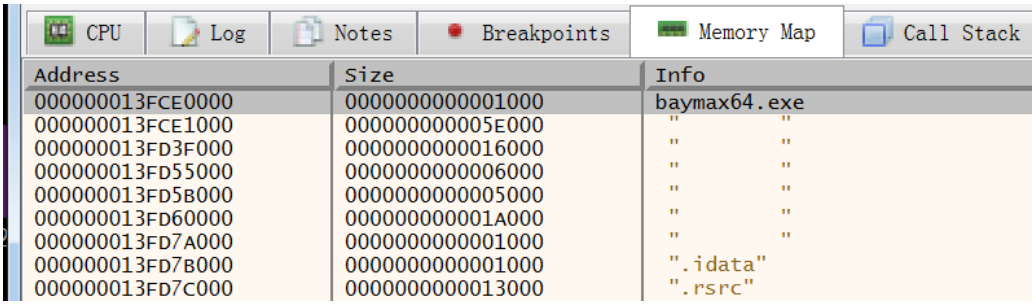
Baymax supports dynamic base addressing (ASLR), which automatically calculates RVA for patching from the patch address and module address entered by the user (without user concern).

The "Virtual Address (VA)" in Option 1 is the address we see in the debugger, so we can enter it directly.



The "module base address" needs to be checked by the

debugger to see the process memory, the module load address is the module base address.

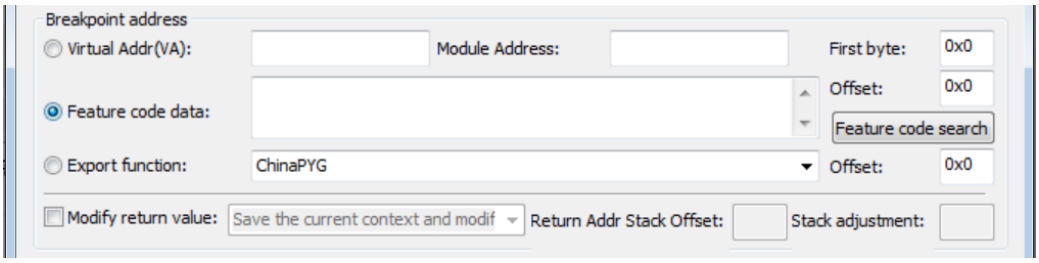


Option 2 locates the patch address by the feature code, which supports ? wildcard character. When using this option, it is recommended to use the "Feature Search" widget to check if the results match your expectations.

Option 3 If the module has an export table, you can set the export function to the patch address.

2. Interrupt patch

The "Abnormal Interrupt Patch", referred to as the interrupt patch, also supports three options for setting the patch address.



(1) Enter the patch virtual address and the module base address

(2) Locating patch addresses by feature code

(3) Locate the patch address by exporting the function

The difference between Option 1 and offset patch is that if the target module has a shell or is to be decoded, setting the INT3 breakpoint also requires entering the first byte of the instruction corresponding to the patch address, and the patch will first determine whether the address has been decoded by judging whether the memory byte is equal to the original value entered by the user. If the module has no shell or uses hardware breakpoint, the default value of 0 can be kept.

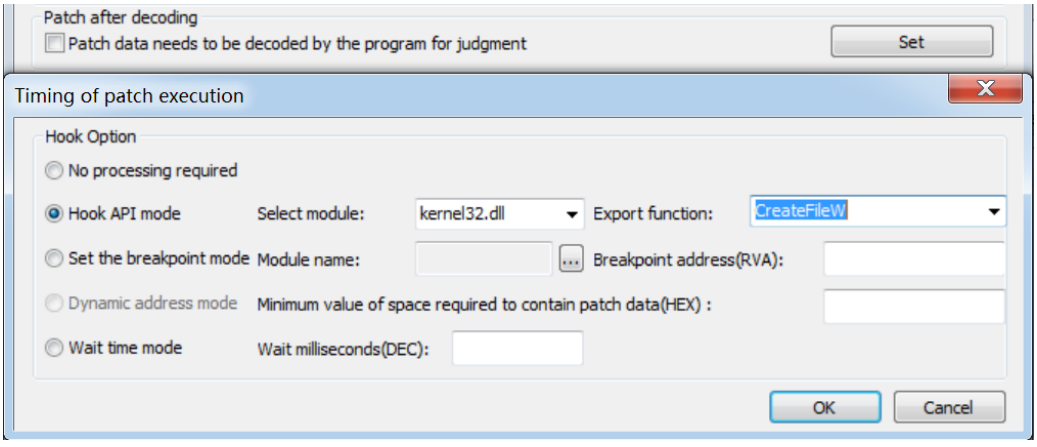
Option 2 can be used to obtain the patch address by using the feature code, and set a break after calculating the virtual address by searching the results + offset value (the offset value can be a negative number). Note that Baymax will set a break in all the results of the search + offset value, if the search result is more than 4, using smart break or hardware break may cause a miss. After extracting the feature codes, be sure to use the feature code search tool to check if the results are as expected, as setting a break at a non-expected address may cause the program to crash.

Option 3 certain shells will detect whether the API first byte is modified, so it provides the function of interrupt

at function + offset (offset value can be negative), here it should be noted that some functions may be different in different platform code, if the offset value is set and interrupt is set at non-instruction first address, there will be unknown result.

(iv) Patch and shell program

Baymax supports patch and shell programs. If the target module has a shell or the patch address is to be decoded, the cracked module cannot execute the patch operation immediately after loading and needs to wait for the code segment to be decoded. Users can use the following options to set the decoding timing of the target process and wait for the decoding before executing the patch. Check the box of "Decode the patch data to be judged by the program after decoding" and then set it in the pop-up box below.



1. Use HOOK mode (recommended)

Choosing the right API to decode by HOOK to crack the shelled program is an art, which is an accumulation of experience and requires more practice for novices to master. Here we can share a HOOK method: after the shell decodes the data, the APIs that have been executed during this period before the patch address can be used as an alternative (the fewer times the APIs are executed, the better).

The program patch shelling is done by HOOKing the target function, and the process tries to execute the patching scheme once each time the function is called until a certain call has been decoded and the patch is executed. So the fewer the number of calls to the selected function, the better, and the fewer the calls, the less the performance loss to the process.

How to choose the right API? The following options are available.

1. The APIs commonly used in shells (heap space application release, memory attribute modification), APIs called at module entry points, APIs in module import tables, APIs related to behavioral operations (such as window creation and destruction), APIs for debugging information output, etc. You will always find the right one after a few

more tries.

2. Pinpoint the API called near the OEP after the program is decoded by debugging, or backtrack up from the patch address and try the API called by the program.

If the selected API causes the program to fail to start or is not patched after starting, please test it with a different API. After finding an available API, we then pick an API with as few calls as possible, which will reduce the performance loss of process startup. How to determine the number of calls? You can determine the number of interrupts set in the debugger, or you can output the log statistics of the HOOK function execution through the Baymax debug patch.

With this scheme, the patch entries will not be patched when the PYG crack module is loaded, but the process will determine once whether to decode and try to patch each time it executes to the selected API, until all the corresponding patch entries are successfully set or patched, and will not continue to patch when it executes to the selected API again.

If you need to add a system DLL to the "Select Module" list, you can add the system module yourself in the Baymax ini configuration item [HOOKDLL].

2. Set the interrupt mode

If the target process does not detect a hardware breakpoint, e.g. uncased or compressed shell, we can set the interrupt mode by selecting the module and entering the offset address RVA (memory address - module base address).

With this scheme, the patch entry is not patched when the PYG module is loaded, but rather the patch operation is started when the process execution triggers a hardware breakpoint at that address.

3. Set the time delay mode

There must be a time interval from the start of the program to run to the patch address. Sometimes we do not want to care about the decoding process of the program shell, we can set a waiting time before patching. This program is a lazy method, which may cause unknown results for different system environments and is not universal.

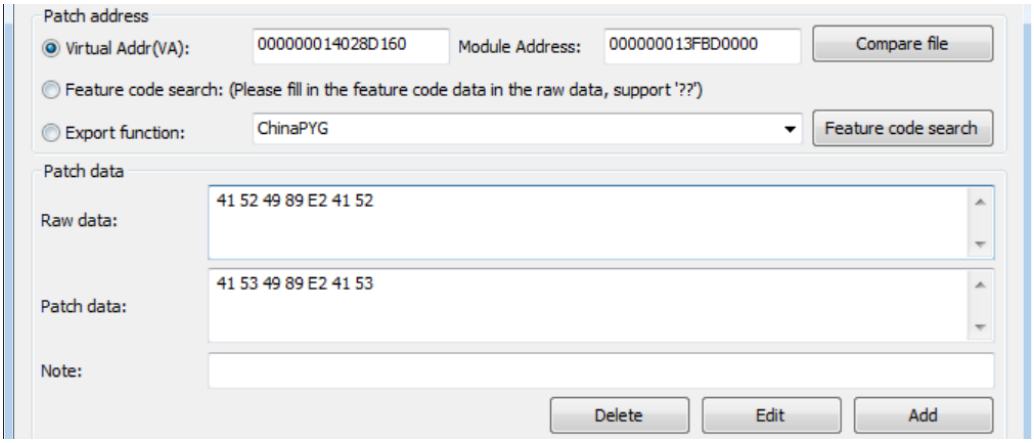
With this scheme, the patch entry will not be patched when the PYG module is loaded; the crack module will create a wait thread when it is loaded, and the new thread will perform the patch operation only when the user-set wait time is satisfied.

Note: The number of milliseconds entered is decimal (the only place in the software setup interface where decimal is

entered, 1 second = 1000 milliseconds).

III. Search for replacement patches

(i) Offset patch



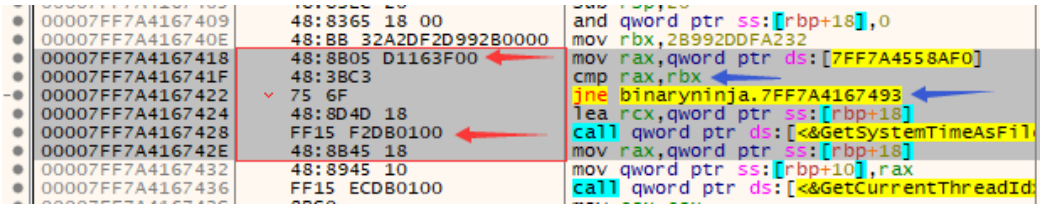
Add an offset patch entry.

First enter the patch address: you can choose to enter the memory address and module base address in option 1, or choose the export function in option 3.

Then enter the original data and patch data corresponding to the patch address (wildcard characters are not supported for offset patches), and click the "Add" button to complete the operation.

If the patch data is large, you can use the "Compare Files" function to compare the original file and the modified file to add the patch data at once.

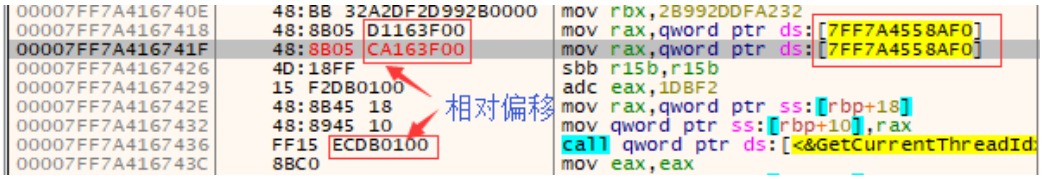
(ii) Feature code replacement patch



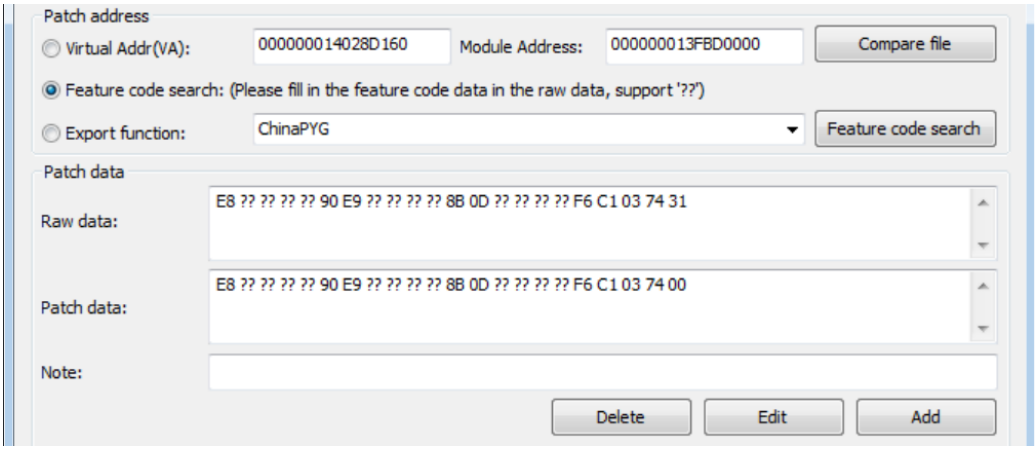
Add a feature code search patch entry for.

In order for patches to support subsequent versions, we often extract feature codes from patch addresses to make a pass-through patch. We use the "feature search" (or baymax plug-in for x64dbg) tool to retrieve our extracted data to make sure the results are as expected.

Feature code extraction requires avoiding the relative offset address in the instruction, as the relative offset address may change in subsequent versions. As in the instructions in the box above, there are two assembly codes that contain relative offset addresses (red arrow standard).

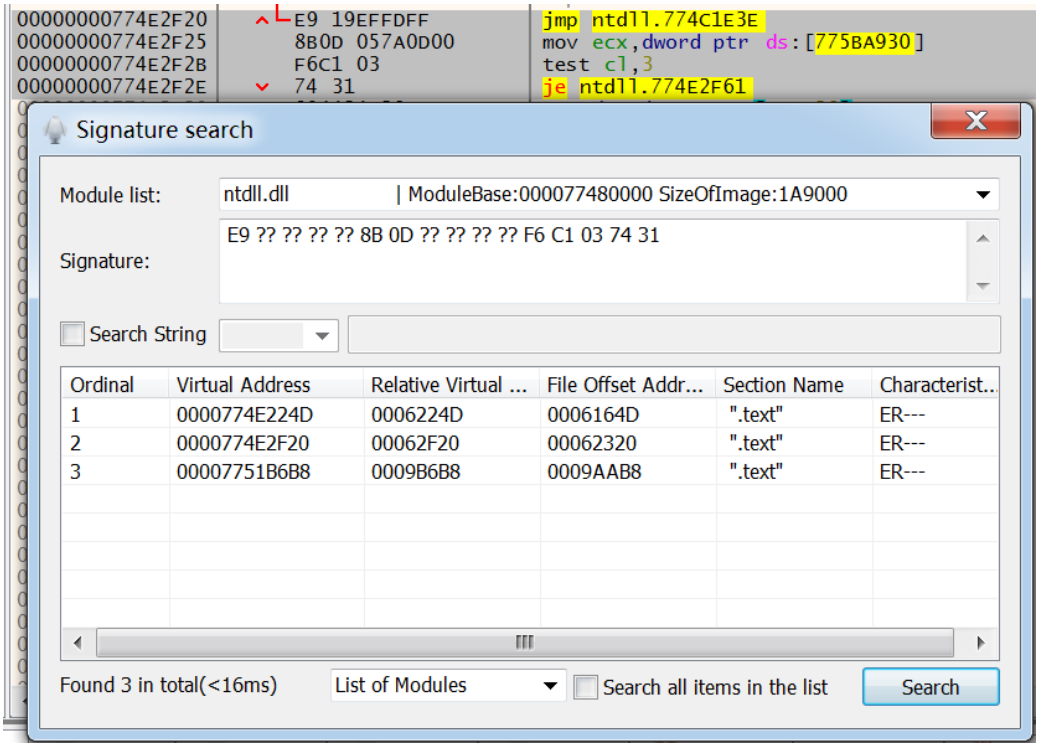


As shown above, since the relative offsets in the directives are highly likely to change in subsequent versions, we replace them with the ? wildcard for replacement. You can compare the extracted feature code data in the figure below.



Select option 2 feature code search, enter the original data and patch data, and click the "Add" button to complete the operation. The input field supports the inclusion of carriage return spaces in the feature code data, but the valid data must be of the same length.

It is recommended to use the baymax plug-in for x64dbg to obtain the feature code data.



IV. Exception handling patch

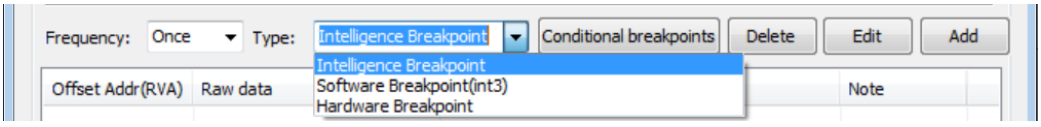
(i) Set interrupt address

We use debugger to analyze the program, set breakpoints on the module first, and then debug and analyze after running interrupts. The same idea is used in the design of abnormal interrupt patching, where the interrupt address must be set first and the patching operation can be performed only after the interrupt.

Each interrupt address can be set with a frequency of interrupt once or interrupt every time. Each interrupt address supports setting multiple patch entries, and each patch entry supports setting different conditional breakpoints.

(ii) Type of interruption

There are currently three types of interrupts: smart breakpoints, INT3 breakpoints, and hardware breakpoints.



1. Hardware breakpoints: Hardware breakpoints have thread dependency, each thread can set 4 hardware breakpoints at the same time, the default will be set on the thread when

the crack module is loaded (usually the main thread), if the HOOK API is used, it will be set to the corresponding thread where the API is executed. Currently there is no function to set hardware breakpoints for all threads.

Setting a hardware breakpoint only modifies the DrX register in the thread context, it does not modify the process memory bytes and does not trigger a memory checksum for the process.

2. Smart breakpoints: Since each thread can only set up to 4 hardware breakpoints at the same time, when the user sets more than 4 hardware breakpoints at the same time, the setting will fail. At this time, users can choose smart breakpoints, which are essentially equal to hardware breakpoints. Each time a hardware breakpoint is triggered, if there is a free DrX at the end of execution, it will detect whether there are still unset breakpoints in the queue and set them in order, and the order in the queue is the same as the order added by the user in the interface.

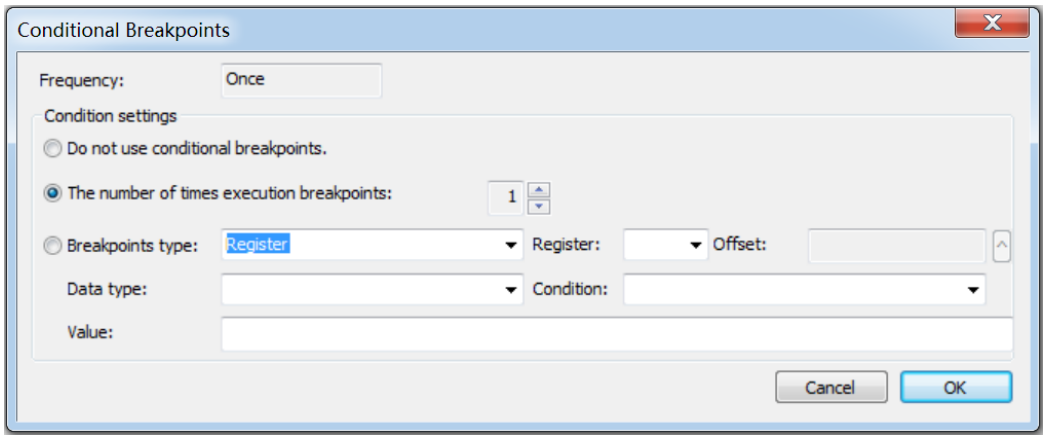
3. INT3 breakpoint: INT3 is different from the hardware breakpoint, no thread correlation, after setting all threads of the process execution to the address will trigger an interrupt. If the hardware setting is invalid, it may be

because the thread is not set on the breakpoint, can be modified to INT3 breakpoint for testing.

The INT3 breakpoint mode is not multi-thread compatible, so if multiple threads execute to the patch address at the same time, unpredictable results may occur.

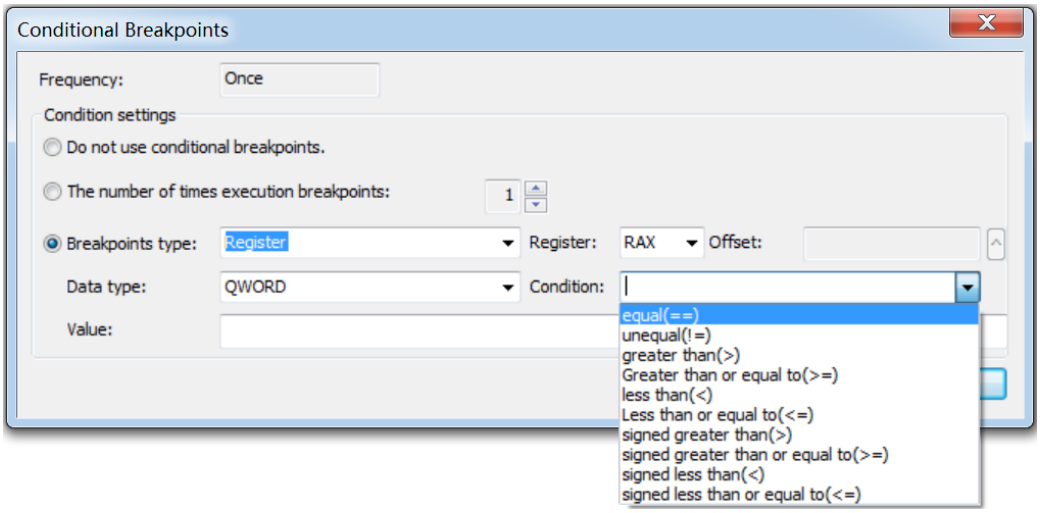
(iii) Conditional breakpoints

1. Determine the first few interruptions



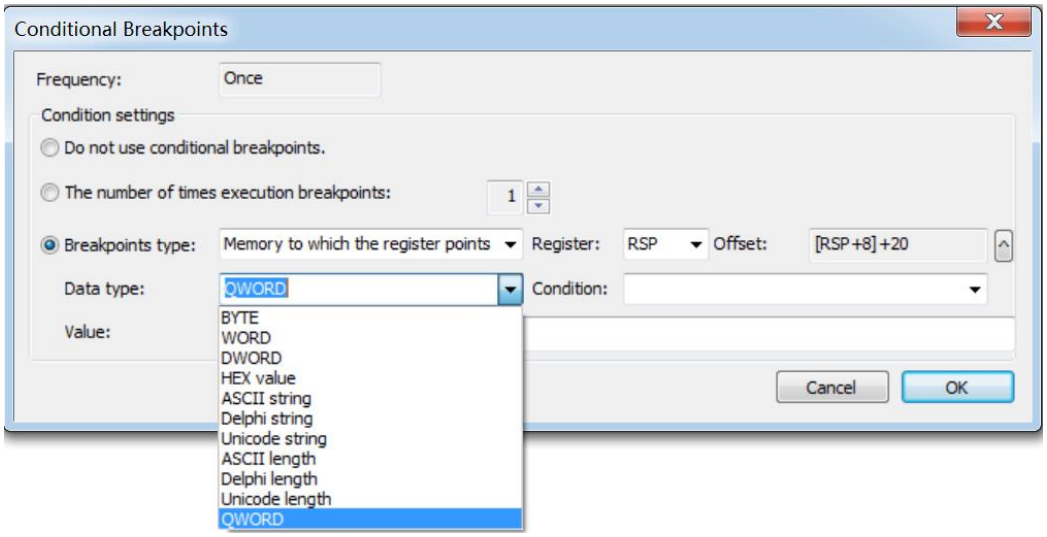
Conditional breakpoints can set the patch address to execute the patch scheme after the first few interrupts.

2. Determine the register value



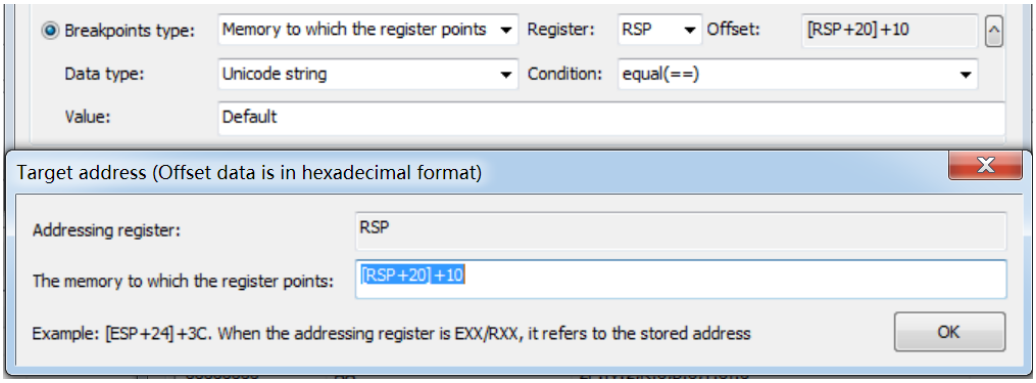
Conditional breakpoints can compare the values in the post-interrupt registers to use as a basis for whether or not to execute a patch.

3. Determine the memory value pointed to by the register



Conditional breakpoints can be used as a basis for whether or not to execute a patch by getting the memory data pointed to by the register.

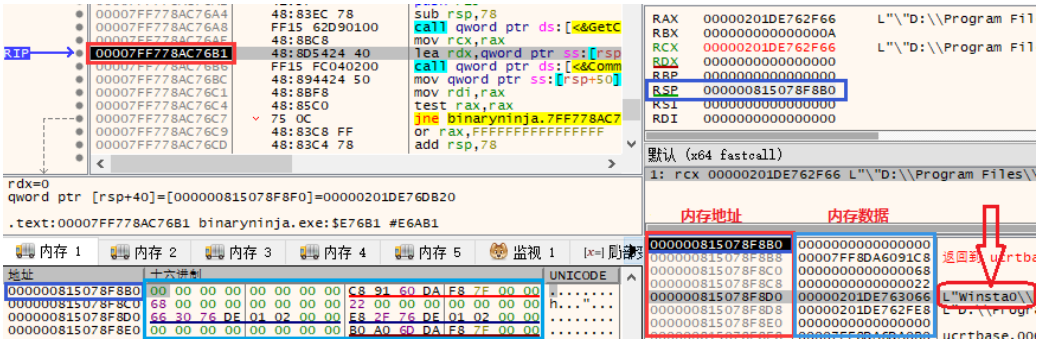
4. Locating memory addresses by register offset



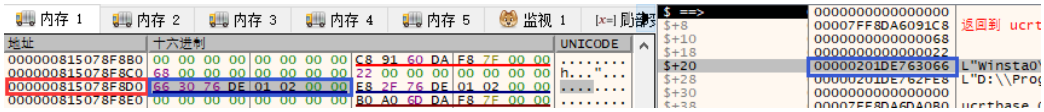
The conditional breakpoints and subsequent patch settings require knowledge of how to set the "memory pointed to by registers", so we will explain them here.

The register+offset is used to locate the memory address of the data. The "Patch Memory Address" field must point to the memory address of the data to be patched, not the memory data. If the memory data cannot be located by the register+offset corresponding to the interrupt address, the value cannot be fetched or assigned using this scheme.

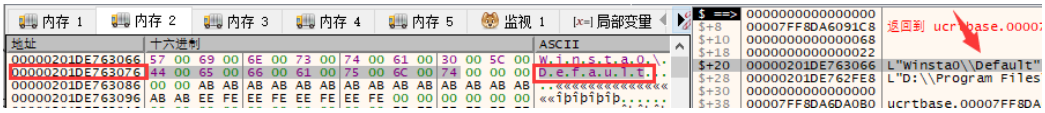
At the interrupt address, we first select a register with an offset value of 0x0 displayed by the interface by default (all data entered here are in hexadecimal). As shown above, the default data in the input field is: RAX+0, which means that the value of register RAX+ offset 0 is taken as the memory address.



In the above figure, RIP: 00007FF778AC76B1, RSP: 00000000815078F8B0, how can we locate the address of the string L "winsta0" in the figure through the register?



We observe that the RSP offset 0x20 memory address is 000000815078F8D0 (RSP+0x20), the address points to the memory data QWORD value of 00000201DE763066, which is the address of the string. So we first set the RSP offset to 20, that is, RSP + 20, and then take its value, that is, enter [RSP + 20].

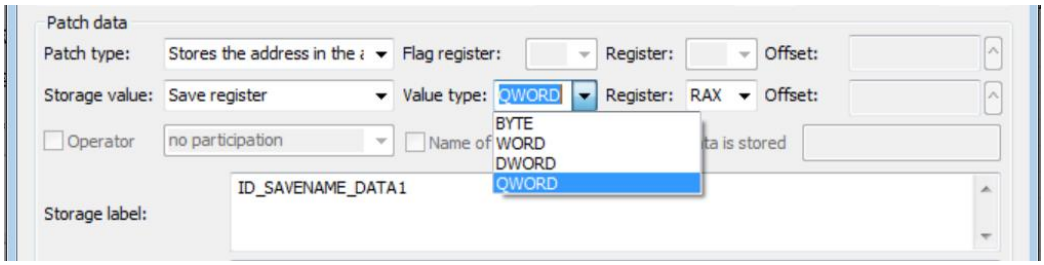


So how to locate the address of the string L "Default"? First locate the address of the string and then +0x10 offset, i.e. enter [RSP+20]+10.

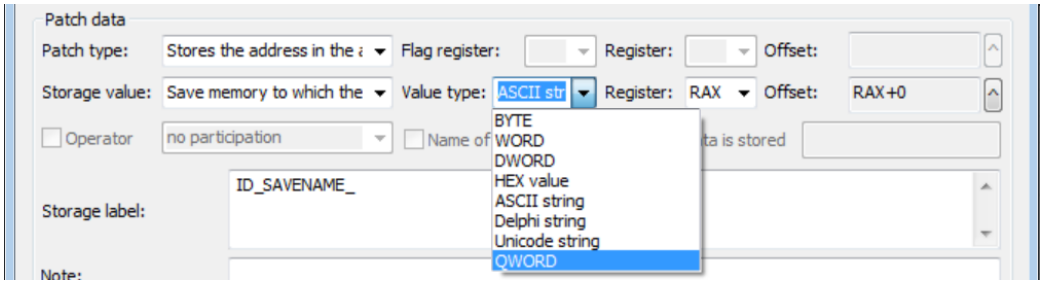
Baymax supports multiple levels of offset addressing, so that addressing the internal members of a class object by its address becomes more flexible and convenient.

(iv) Storage data

When an interrupt is triggered, the register value corresponding to the interrupt address or the memory data pointed to by the register can be stored.

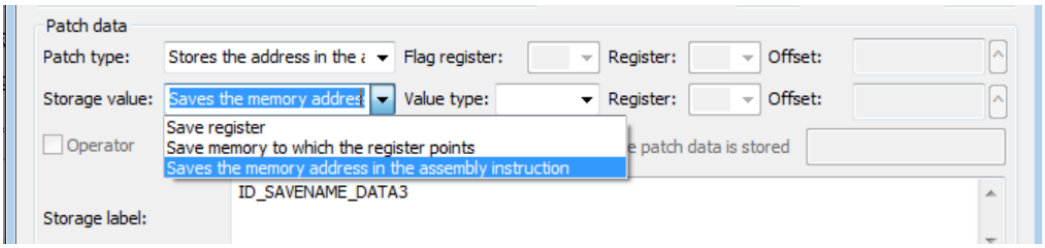


Select the type of stored value and the corresponding register, and enter a unique name (global variable name) for the stored data in the "Storage Label" field, so that we can refer to it when we use it later. Note that the data type set at the time of storage must be consistent with the type of subsequent references.



It is also possible to store more types of data by storing the data pointed to the memory by register + offset. When the value type is "HEX value", the data in the "Storage label" column is "ID_SAVENAME_,0", and ID_SAVENAME_ is the name of the stored ID_SAVENAME_ is the name of the stored

data, and the length of the HEX data is after the comma.



Storing the current data also supports extracting the value in the form [XXXX] from the assembly instruction at the interrupt address, i.e., the address of the global variable in the stored instruction.

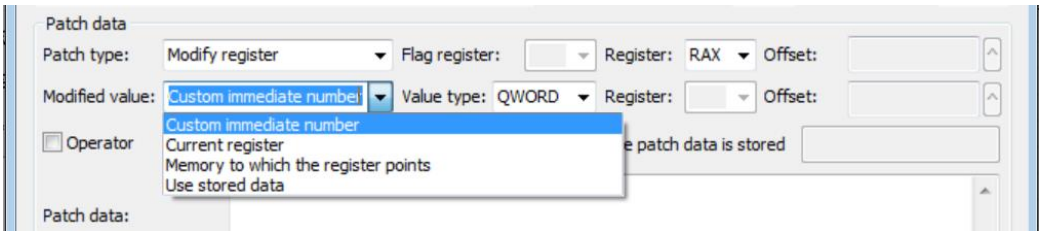
7738E9FB | 890D 74E73F77 | mov dword ptr ds:[773FE774],ecx

If the interrupt address is 7738E9FB, select "Save memory address in instruction" to store the value, and set the storage mark, the DWORD value 773FE774 will be stored after the interrupt.

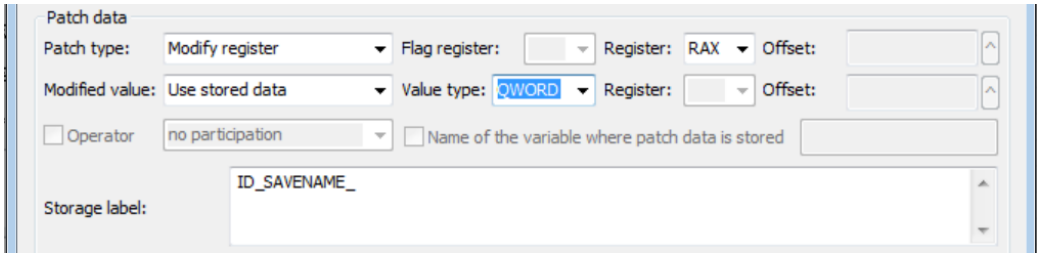
After saving the address, we can use the memory marker in the patch type "Modify memory pointed to" and modify the memory data pointed to by the memory marker to modify the value of the global variable in the instruction.

(V) Modify register

When an interrupt is triggered, the register value corresponding to the interrupt address can be modified.



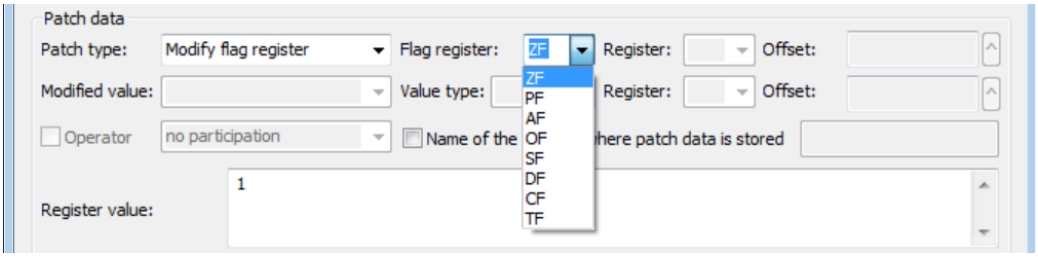
Patch data can be self-defined data, just enter it in the patch data field, and note that the value must not exceed the selected value type; you can also choose other registers, for example, the registers point to the true and false codes respectively when interrupting, you can replace the register value that stores the false code with the register value that stores the true code; you can also replace the data with the register pointing to the memory.



You can also use the data that we have stored. You must ensure that the value type is the same when using stored data. Enter the name of the data we have stored in the Storage Label field.

(F) Modify the flag register

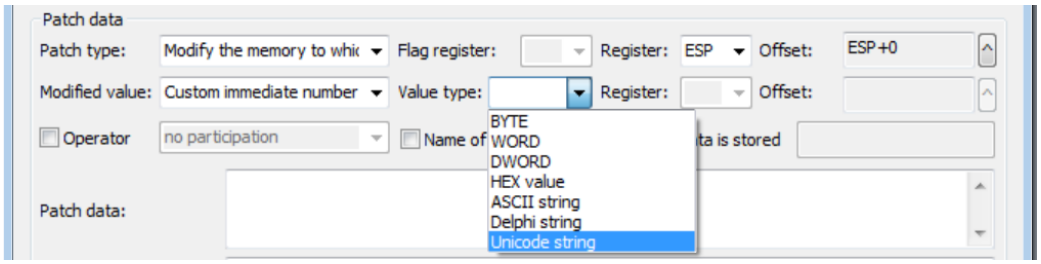
When an interrupt is triggered, the flag register corresponding to the interrupt address can be modified.



To modify the instruction jump, you can change the execution flow by modifying the flag register. You can add multiple patches to the same interrupt address to modify multiple flag registers, and fill in the modified value 0 or 1 in "Register Value".

(vii) Modify the memory pointed by the register

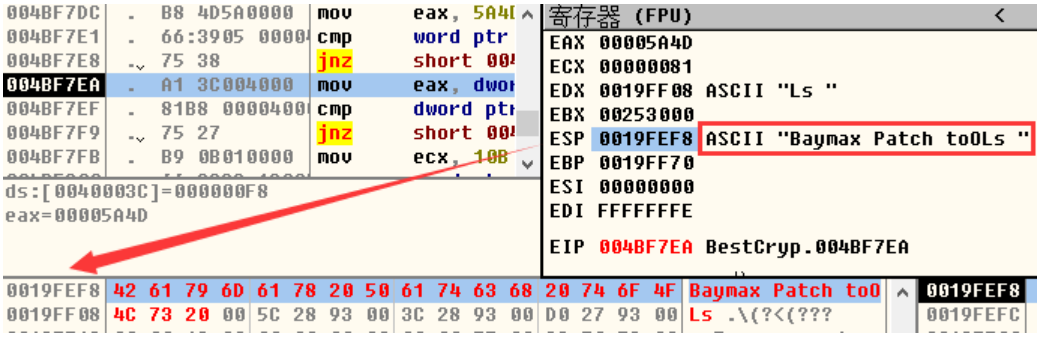
When an interrupt is triggered, the memory value pointed to by the interrupt address register can be modified.



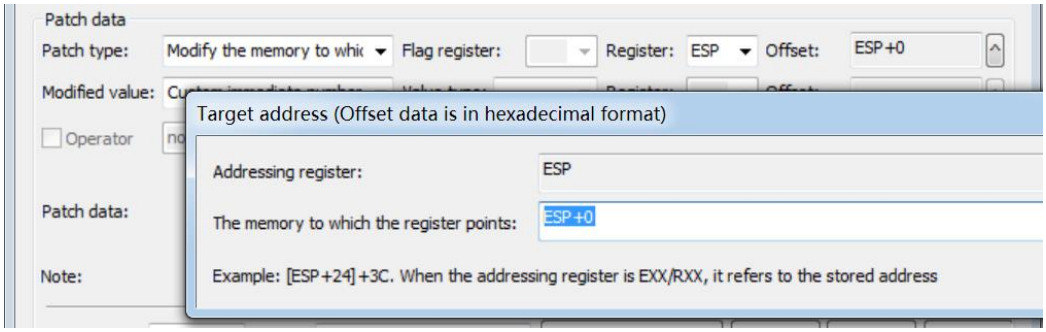
Modified values support custom immediate numbers (e.g. replacing the public key of RSA or decoded data), support using the current register value, support using the memory pointed by the register (e.g. two registers pointing to memory data for comparing values can be replaced before comparing), and can also use stored data.

The method of locating memory through registers is

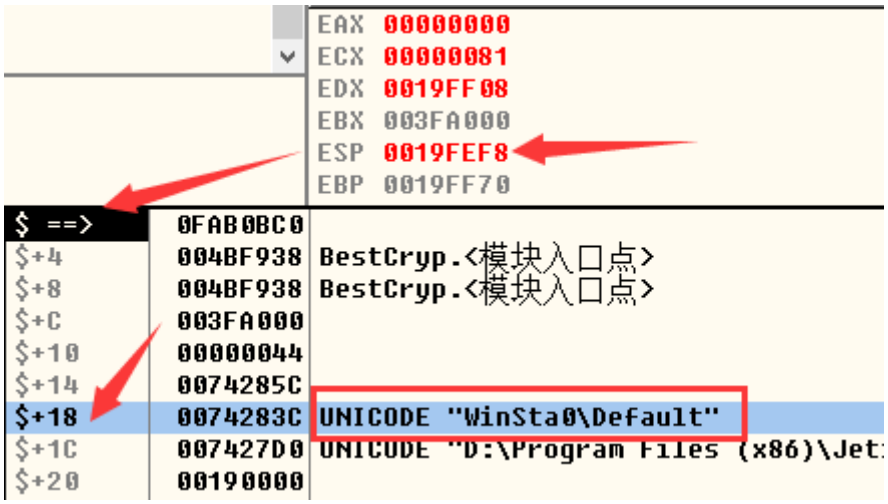
explained in detail in the "Conditional Breakpoints" section above, and the principle is that the input data must be the address of the memory data. For example, if we see a string in a register, its address is equal to the register value.



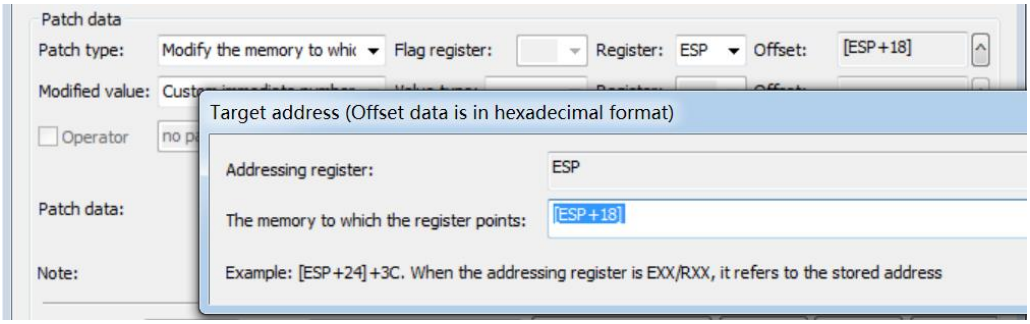
Patch memory address input: ESP or ESP+0 are available.



And the address of the string seen in the stack will need to be obtained by first locating the string line by ESP/RSP plus an offset, and then fetching the data at that address.



As shown above, ESP+18=0019FF10, the address points to the memory data 0074283C, 0074283C is the address of the string "WinSta0...".



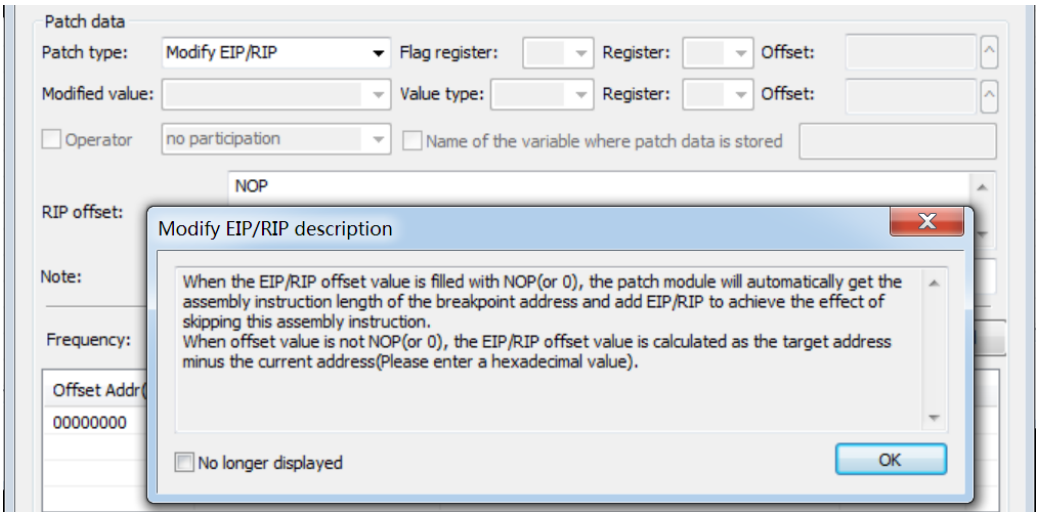
At this point, the patch memory address should be entered as:[ESP+18].

The input format supports multi-layer addressing in the format of [[EXX+X]+Y]+Z, where EXX is a register, XYZ is an offset value, and [] represents the data pointed to by the memory, which is taken as DWORD for 32-bit processes and QWORD for 64-bit processes.

Small summary: Take the address of the register pointing to the string: EXX+0/RXX+0. Take the address of the string in the stack: [ESP+N]/[RSP+N].

(viii) Modification of EIP/RIP

The EIP/RIP value can be modified when an interrupt is triggered.

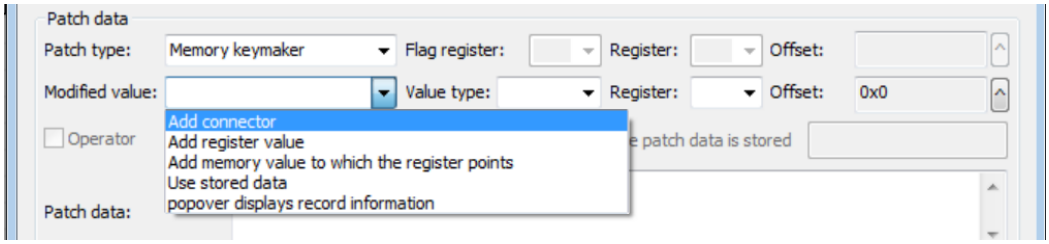


The default offset value is NOP, in fact, it is not to modify the instruction to NOP, but the patch module will automatically get the length of the assembly instruction of the interrupt address and add EIP/RIP to achieve the effect of skipping the instruction. If the offset value is not NOP (or 0), the EIP/RIP offset value is calculated by subtracting the current patch address from the target address (please enter the hexadecimal value), and supports positive and negative numbers.

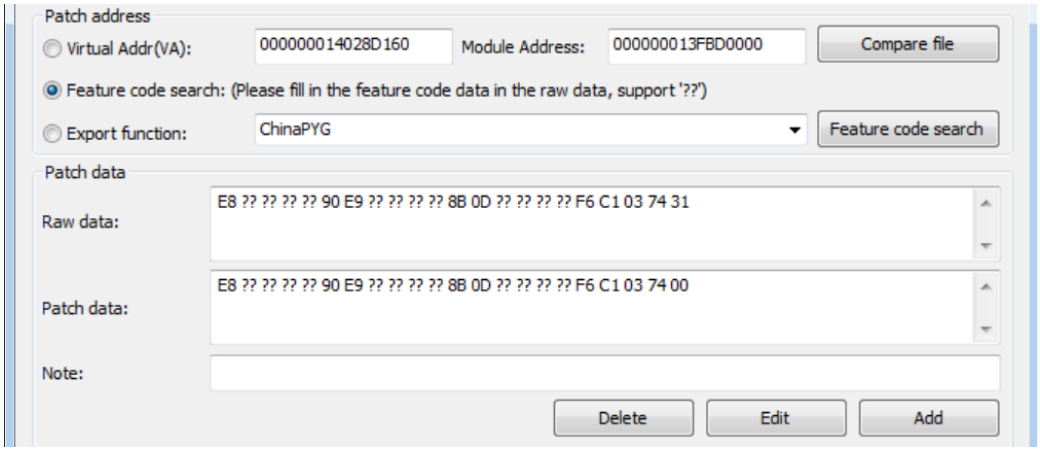
(ix) Making a memory registrar

Baymax provides a pop-up box to display extracted information, which allows you to capture a lot of information first and then display it all at once. This function can also be used as a memory registrar.

The implementation is in two steps, the first step adds data sequentially, supports reading information from registers or memory, supports adding connection characters, and can also add stored data.

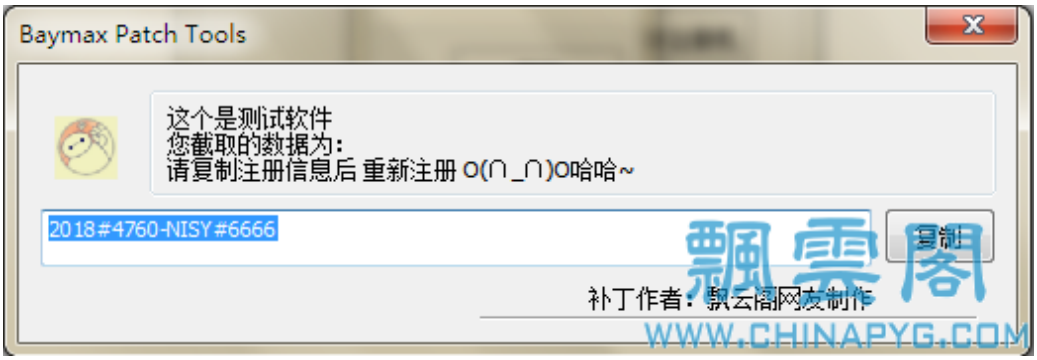


The second step is to display the intercept information. After adding all the information to be displayed, you can set an interrupt at any address and then add a "pop-up display log information", then when the interrupt is triggered at that address, a new thread will be created to stitch all the previously added data in order and display the pop-up. In addition, when the execution reaches the popup address, the log file will record the information of all registers at that address.



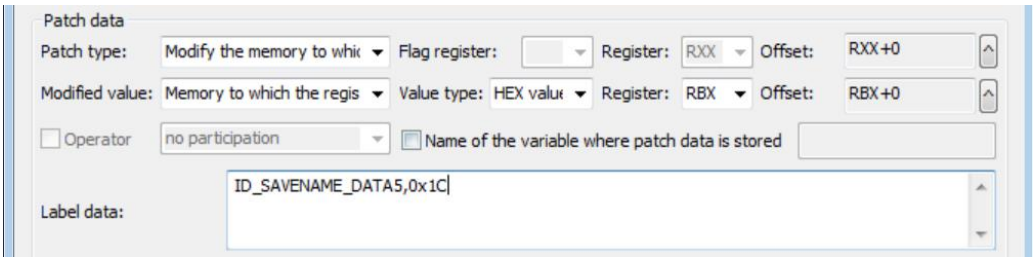
The patch description field can be customized with pop-

up box information, and the display interface is as follows.



(x) Modify the memory pointed to by the memory address

If the stored data is an address, we can set an interrupt at any address to modify the memory pointed to by that stored address.

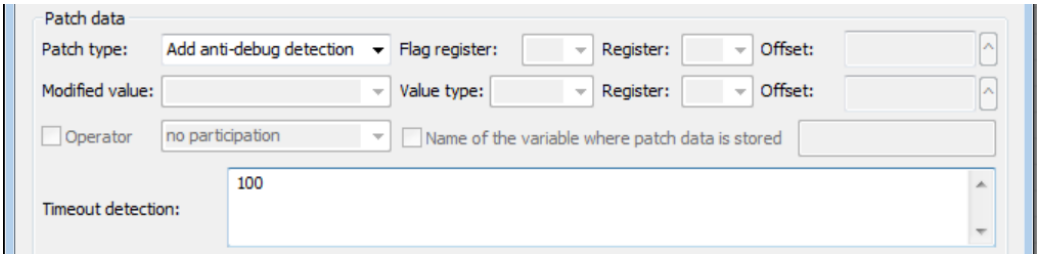


The modified value can be a custom immediate number, the register of the interrupt address or the memory data pointed to by the register. The input field format is "ID_SAVENAME_,0", ID_SAVENAME_ is the name of the DWORD (32-bit process) or QWORD (64-bit process) we saved before, after the comma we enter the custom data, where the modified value is " If you choose "HEX value", please enter the length of HEX data after the comma when you modify the value to "Memory

pointed to by register".

(xi) add anti-debugging detection

Baymax supports setting anti-debug detection for interrupt addresses. When the debugger dynamically analyzes, if the dwell time at the breakpoint exceeds the user preset value, it is assumed that the program may be being debugged and analyzed and the program will trigger a crash.



User preset value in milliseconds, input field is **decimal** value, default 100, minimum 20 (recommended not less than 50 milliseconds), maximum 2000 (2 seconds).

(xii) modify the return value of the function

Sometimes we need to modify the parameters or return value of a function. Baymax does not provide a function HOOK function, but it is simulated by exception interrupts to achieve this function.

The "Modify Return Value" function is available for all patch entries in the interrupt patch item. After the patch

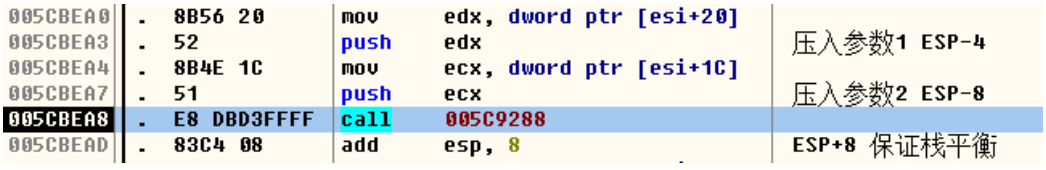
address is interrupted, the function return address is read and the interrupt is set, and after the function returns, the interrupt is triggered again and the patch operation is executed.

The function return process can be broken down into two steps: first Set New EIP/RIP, and then ESP/RSP + N to restore stack balance. The same is true for simulated function returns, so the following three modes of modifying return values also require attention to these two sets of information: getting the function return address from the stack and the stack balance adjustment value.

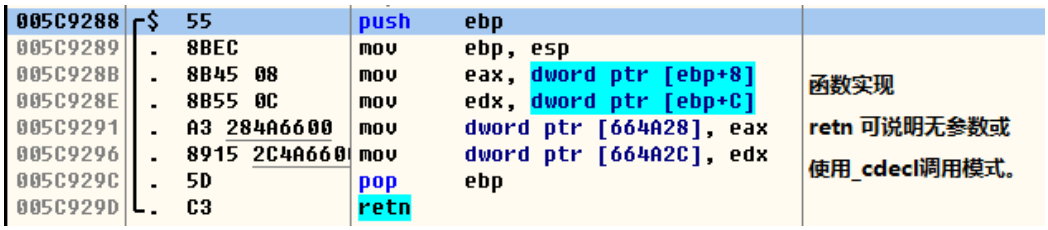
1. Function calling convention

We need to know about stack balancing, and here is a brief introduction to common function calling conventions.

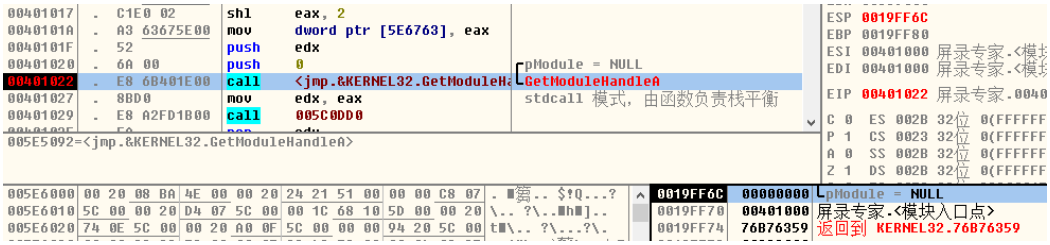
1.__cdecl is the default function call protocol of C/C++. All arguments are put on the stack in order from right to left, the function return is generally retn, the stack balance is balanced by the caller, the caller will execute add esp, n to balance the stack after the function return, n is the memory size occupied by the argument stack.



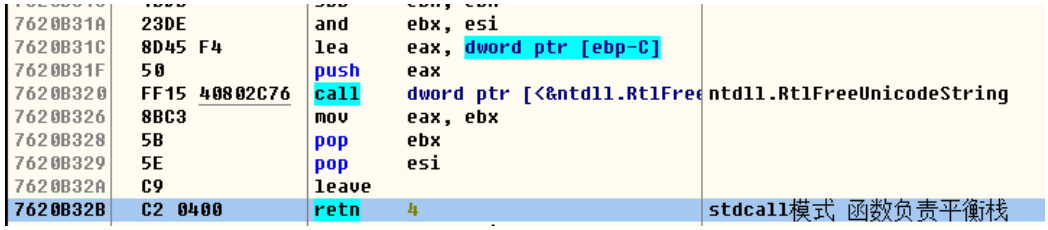
The function implementation is shown in the following figure, retn stands for function without parameters or using cdecl call protocol.



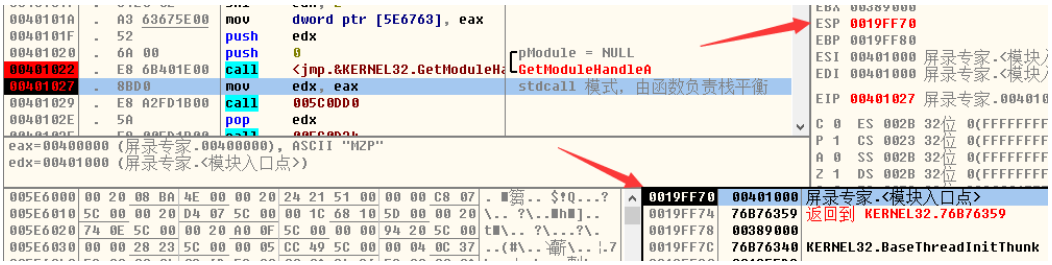
2. __stdcall is the default function call protocol of Windows API. All the arguments are put on the stack from right to left, and the stack balance is achieved by the function itself, and the function returns generally retn n. The caller does not need to pay attention to the stack balance.



When the system function GetModuleHandleA is called, the parameters are pressed into the stack and the top of the stack is 0019FF6C when the CALL instruction is executed.



The system function is responsible for stack balancing, this function executes retn 4 and returns.

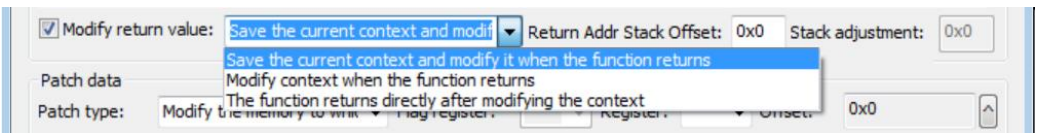


After the function returns, the top of the stack address is 0019FF70. because the system function is responsible for balancing the stack, so if we return directly to an address in the function, the top of the stack also needs to be consistent with the original function return, the stack balance adjustment needs to be set by yourself.

3. __fastcall is the function call protocol for x64 processes, and parameters are passed sequentially from right to left. The parameters are passed through the RCX, RDX, R8, and R9 registers. If there are more than four parameters, the stack space for the first four is also reserved on the stack, and subsequent parameters are written to the corresponding stack offsets.

2. Three kinds of modified function return value mode

Once we understand this knowledge, let's move on to look at three options for setting up the modified return value.

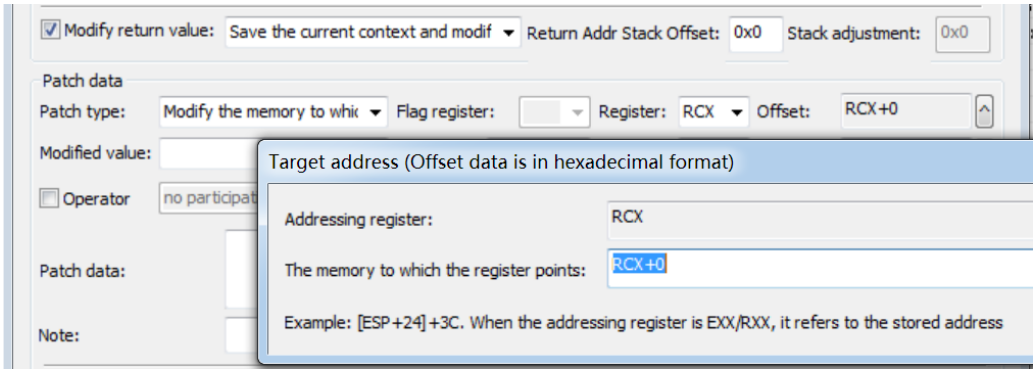


1. Save the current context and modify it on return.

Some functions use registers to pass parameters, and we need to modify the data received by the parameter when the function returns, but there is no way to find the value of the parameter passed in at the function return. In this case, we need to save the register value first, and then modify the memory data pointed to by the parameter when the function returns.

This requirement can also be achieved by "storing data" and "modifying the memory pointed to by the storage address", but it is not convenient enough.

With this solution, the user can set an interrupt at any address within the function where the parameters are available, and at the same time set the "return address stack offset" when the patch address is interrupted, i.e. the offset value of the function return address in the stack relative to the ESP/RSP after the interrupt. Since the incoming parameter must be a memory address, the patch type in the patch data field can only be selected as "Modify memory pointed to by register". The register + offset value set at interrupt will be saved, and the memory data pointed to by the saved address will be modified after the function returns.

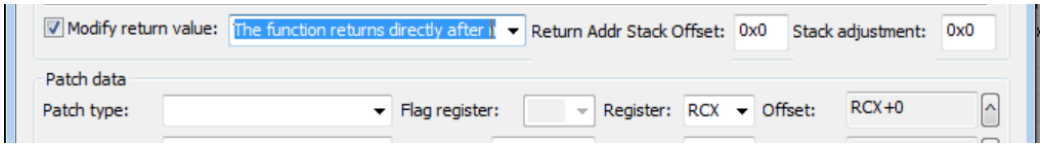


2. Modify the context when the function returns

When this scheme is used, it is also necessary to set the "return address stack offset". The patch data we set is not executed when the patch is interrupted, but after the interruption, we get the function return address from the stack and set the breakpoint, and only after the interruption at the function return does the function start to read the register context and execute the patch data.

3. Modify the context after the function returns directly

Some functions we do not need their execution, but only need to ensure that the function return value, then you can choose this option, in addition to the need to set the "return address stack offset", but also need to amend the "stack adjustment" value, to ensure that the function directly after the return of the stack balance. The operation can be set at any instruction within the function, stack adjustment, please refer to the normal return of the function when the top of the stack address.



3. better understanding of arbitrary instructions are located in the function body

Any instruction, which is in a function, (basically) has a function return. It is not only API functions that can set returns; patch addresses all support modifying function return values.

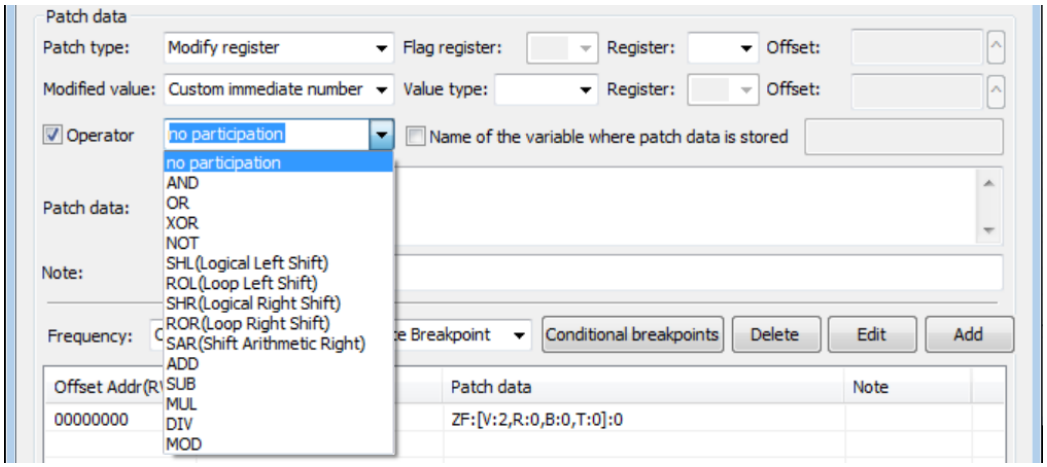
4. Any stack address returned before patching

Modify the return value of the function, not only after the return of this function to modify, just before the use of the modification can be, so the stack of some function return address can be used, the patch tool does not care about when the function returns, only the return address obtained from the stack.

(xiii) Perform operations on data

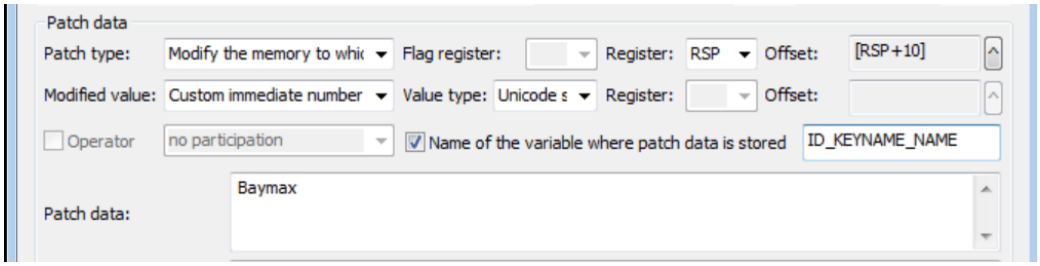
In some cases, we need to perform arithmetic operations on the data. When the modified value is "Custom Immediate", you can check the "Data Arithmetic" option, and the content entered in the patch data field will be the second value involved in the operation. If you select "Inverse operation

(NOT)", the input data will be ignored.



(xiv) Read patch information from INI files

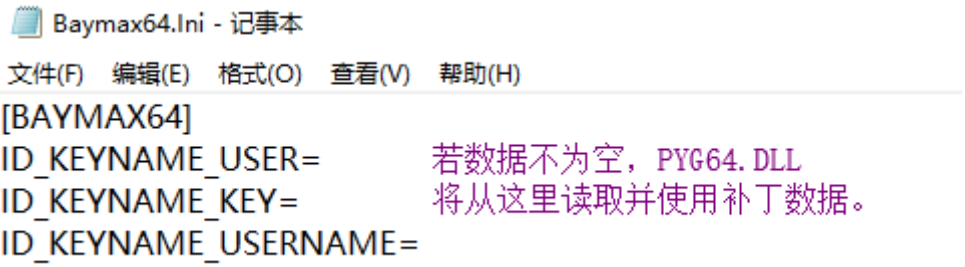
When "Patch data read from text" is checked, the Crack module can read user-defined data through the configuration file to complete the patching operation.



If the patch **entry** has the "Patch data read from text marker" checked, the patcher will release the Baymax.ini or Baymax64.ini file (see below for format) in addition to the hijacking and cracking module in the target folder, and **when the data corresponding to the marker entry is not empty, the** patcher **module will read the data corresponding to the marker When the data corresponding to the marker item is not empty,**

the patch module will read the data corresponding to the marker and replace the original patch data for patching. For example, if this option is checked in the patch item of the software about information, the user can customize the content to be displayed.

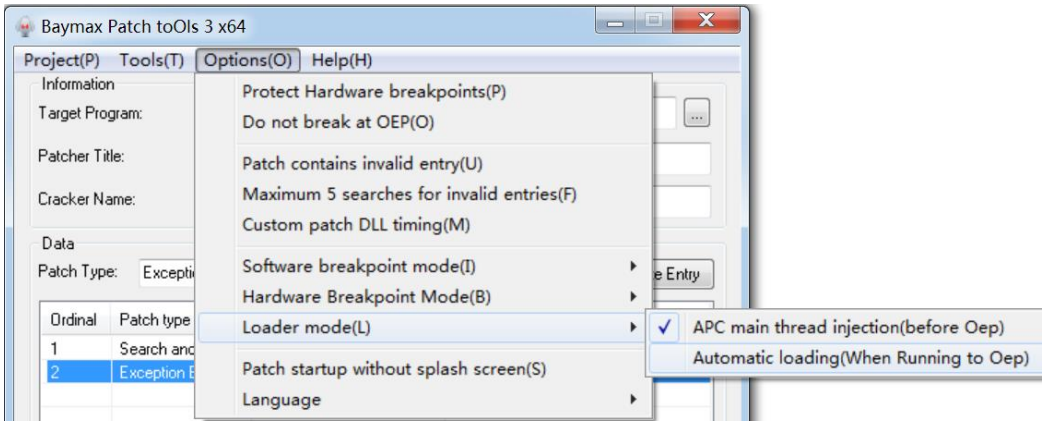
Note: The patch data filled in by the user must be the same as the value type in the patch entry. For example, if the value type of the patch entry is HEX value, the data entered by the user will be parsed and patched according to the HEX data.



V. Generating Patches and Saving Patch Projects

(A) Menu setting item description

A number of additional functions are available in the Baymax menu settings item.



1. Protect hardware breakpoints: Some shells will erase hardware breakpoints. If you analyze the log file and find that all DrX registers are 0, then the hardware breakpoints may be set to be cleared, so you can check this option at this time. At present, this function is still to be improved in x64.
2. OEP is not set to interrupt: Baymax's patch will set 0xCC at OEP by default, some shells such as Obsidium will detect code segment integrity, you need to check this option not to set it at this time.
3. Patch contains invalid entries: To improve startup performance, Baymax is internally optimized to consider the decoding as incomplete and not execute subsequent patch entries when invalid entries are encountered. If the patch is compatible with old and new versions and contains invalid entries, this option should be checked.
4. Invalid entries can be searched up to 5 times: If the

invalid entry is included and HOOK is set, the patch will be tried every time the API is executed. After checking this option, if other patches have been executed, the invalid entry will be marked as ignored after 5 executions and the entry will not be executed again.

5. Custom patch DLL timing: If the patch address is located in the DLL module, Crack Module will enable the function of monitoring module loading by default, and determine whether it is the target module and execute the patch function when the module is loaded. Some large programs are loaded with many DLL modules when they are started, and monitoring module loading will cause some performance loss and affect the startup speed, so users can check this option to cancel the monitoring of module loading and control the timing of DLL patching by themselves to improve the software startup speed.
6. Software breakpoint interrupt mode.
   1) Simulates single-step mode, the interrupt mode used by INT3 in versions prior to x86 v2.9.5 and x64 v2.5, by parsing and setting NEXT_IP to simulate single-step execution, where a patch entry records

and processes a patch data.

2) Simulates single-step mode II, based on mode I. The function return address will create a new node for processing.

3) Mimic HOOK mode, support multi-threading. A new INT3 interrupt mode, similar to HOOK mode, when triggering a CC interrupt, through the exception mechanism, jump to the new space to execute the assembly instruction at the original address to implement the multi-threading mechanism, this program is recommended.

7. Hardware breakpoint interrupt mode.

1) The current thread (the thread where the patch module is loaded, usually the main thread) is set by default, and the non-main thread can be HOOKed by the API called before the execution of the patch address of this thread to achieve setting hardware breakpoints for this thread, each thread has its own hardware breakpoint handling mechanism, which does not affect each other, and this solution is recommended.

2) Set all threads by default (set hardware

breakpoints for all threads via HOOK), and set recurring hardware breakpoints for newly created threads via the Hook API. Note: If the new thread will not execute the 4 break addresses currently set, it will not have the opportunity to set and execute subsequent breakpoints of the hardware list.

8. Loader loading mode.
   1) APC main thread injection (Before OEP), which injects the patch module into the process via APC before it executes to OEP.
   2) Autoload (When Running to OEP), when executing to OEP, loads the patch module directly via ShellCode.
9. No pop-up box for patch startup: Baymax displays the LOGO pop-up box by default when the patch is started to facilitate users to perceive that the patch has been loaded properly and working.
10. Switching languages

    Users can switch the desired language, if you want to support other languages, you can send me the language file (email or github).

(ii) Create hijacking, injection patch

Baymax can create release and debug patches. The "Create Patch" button on the main screen generates a release version of the hijacked patch, while the menu item allows you to create debug and injection patches.

Patch hijacking releases the hijacking module and cracking module to the process folder, and loads the cracking module through the hijacking module to execute the patch. Injecting the patch will release the crack module to the process folder at startup, and let the process load and execute the patch through the APC mechanism.

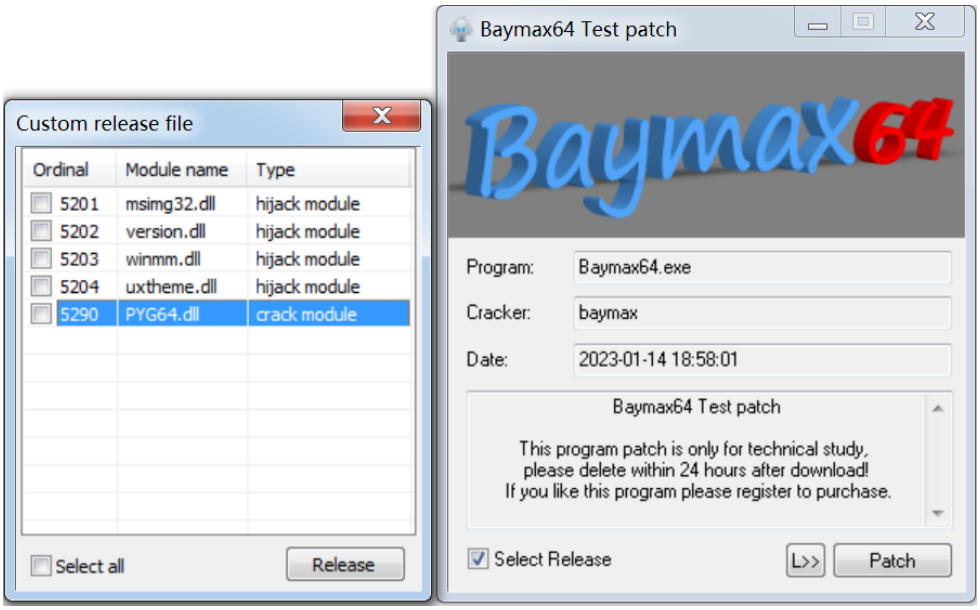
The patches created by Baymax will save the existing language resources into the patch, and when the patch is run, the corresponding language will be selected according to the running system (if the corresponding language is missing, English will be displayed), or the user can select the language in the patch menu (the "L>>" button in the interface).

(iii) Create debug version patch

The difference between the debug patch and the official patch is that the debug patch user data is not encrypted and

a .log file is generated to record the execution of all patch entries.

(iv) Hijacking module and cracking module



Baymax has two types of patch data: hijacking modules and cracking modules. Due to different platforms, or different system environments, the hijacking modules that can be loaded by different processes are unknown. So Baymax's patch release tool uses intelligent identification to release hijacked modules by default. If the identification timeout does not find a hijacked module that can be loaded, the patch tool will ask whether to enable injection mode to start the process and run the patch solution.

There are two cases of not finding the hijacking module that can be loaded: it may be that the hijacking module

within the patch cannot be loaded automatically, at this time you can create other hijacking modules that can be loaded, or add the export function of the cracking module in other loading modules for loading, or generate injection patches; it may also be that the software is not loaded directly when it starts, and some large programs may load the hijacking module during the startup process. At this time, it is recommended to manually release all hijacked modules and run the program through tools such as process explorer to see if the process is loaded with hijacked modules, or by determining whether the program starts with a pop-up box showing Baymax's unique LOGO to determine whether the cracked modules are loaded properly.

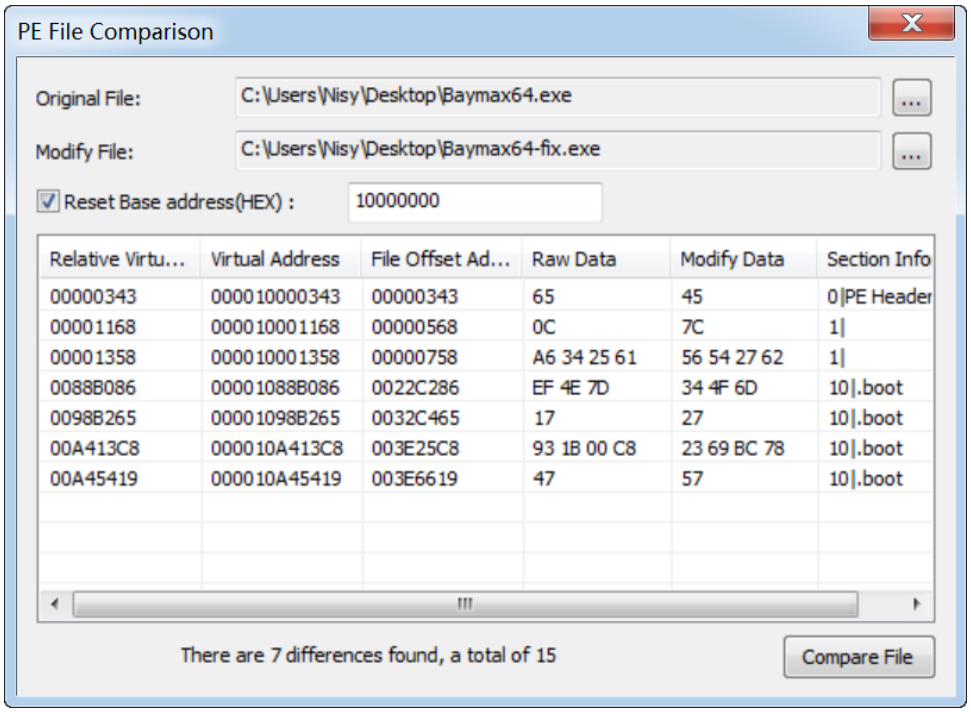
(v) Save and open the patch project

Baymax can save the patch project as *.bpt project file and support opening bpt file to create patch. If you do not understand the bpt data format, please do not edit and modify it manually. In addition, the bpt file contains version information, so if a new version of Baymax is included, it may fail to create a patch in a lower version of Baymax. Newer versions support backward compatibility and the latest

version is recommended.

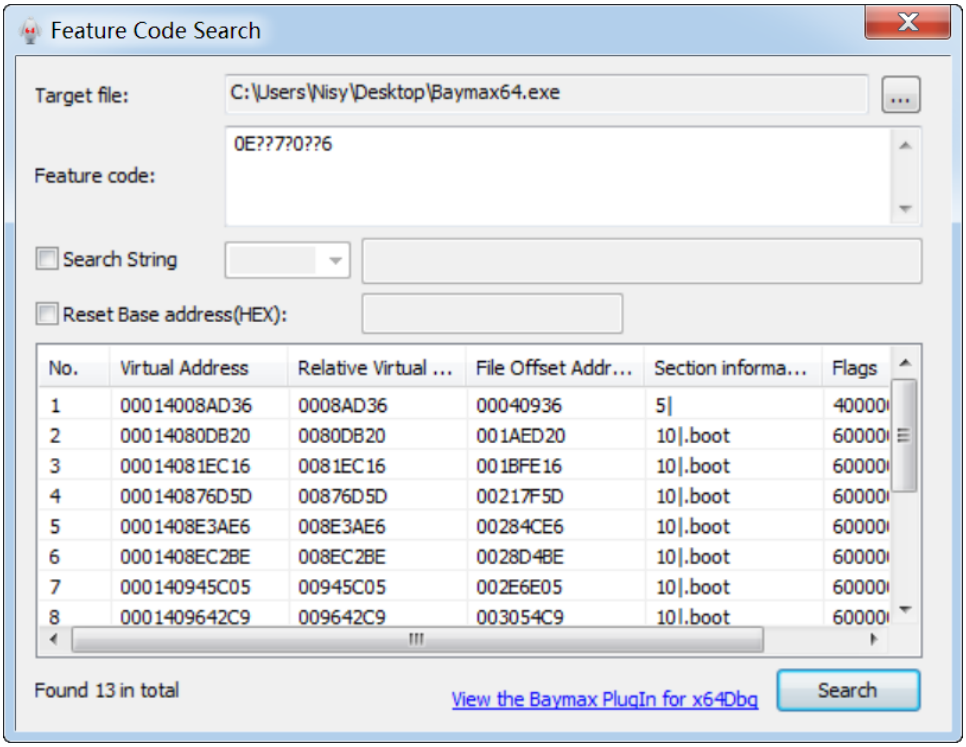
VI. Introduction of plug-ins

(i) Binary comparison of PE files



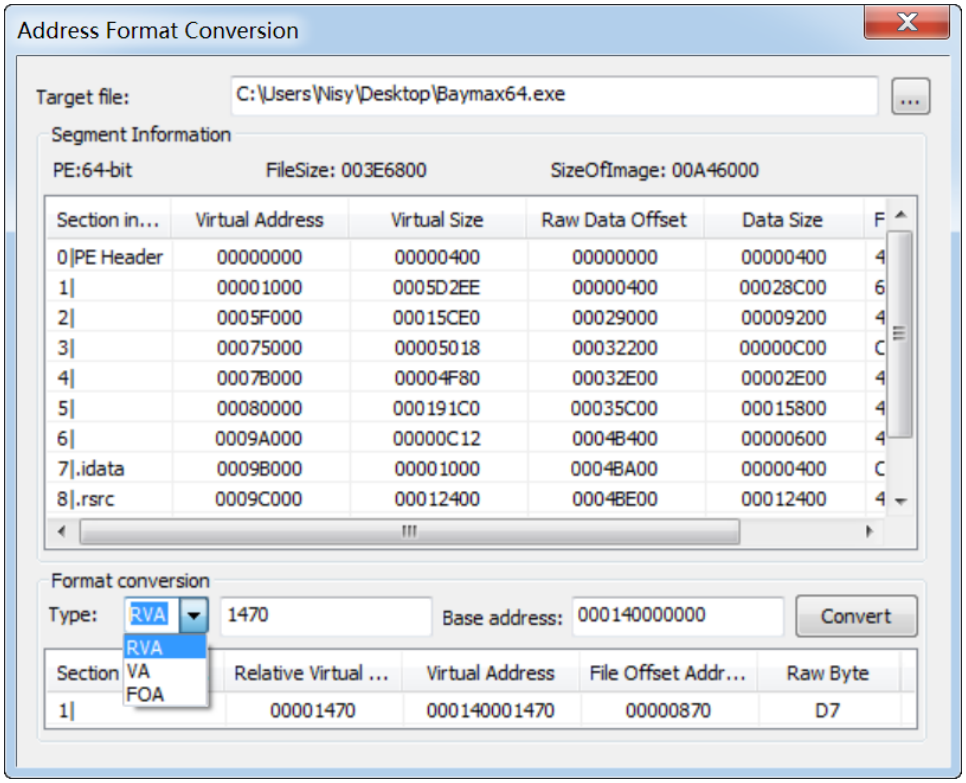
Support PE file binary comparison, the files involved in the comparison must be the same size. If this function is called up by the "Compare Files" button in the offset patch interface, you can click "Add Selected" to add the checked patch data at once.

(ii) feature code search tool



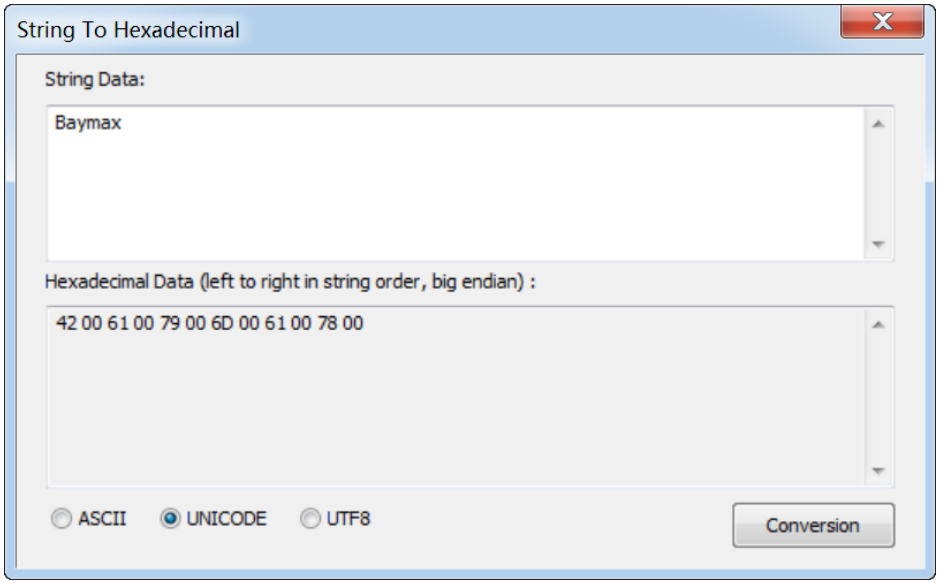
Feature code search tool. Currently, it only supports searching PE files, supports resetting the base address, and lists the VA, RVA, and FOA corresponding to the feature code.

(iii) Address format conversion tools



Parsing PE zone information and supporting conversion between RVA, VA and FOA formats.

(iv) string to hexadecimal tool



We sometimes need to convert strings to HEX values when filling in patch data, so this small tool is provided to facilitate the format conversion, and HEX data is output by memory low address to high address.

VII. Cases and Usage Tips

(i) Set hardware breakpoints on other threads through HOOK

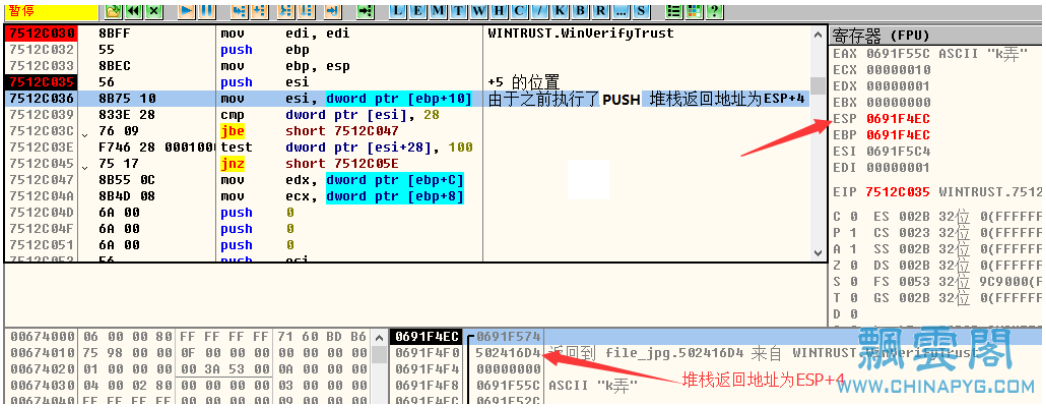
Hardware breakpoints are thread-dependent, and the crack module executes the "breakpoint setting function" to set a hard break only for the current thread, usually the main thread. Baymax currently does not provide the option to set hard break for all threads, the workaround is to HOOK API, every time the execution of the HOOKed API function will call the "breakpoint setting function" again and set the hard break for the current thread. Hard break.

(ii) through the function HOOK to achieve a fixed parameter or return value

The function "modify function return value" provided by Baymax has some limitations, if we use INT3 scheme, there will be problems when multiple threads call at the same time; if we use hardware breakpoints, we cannot set breakpoints

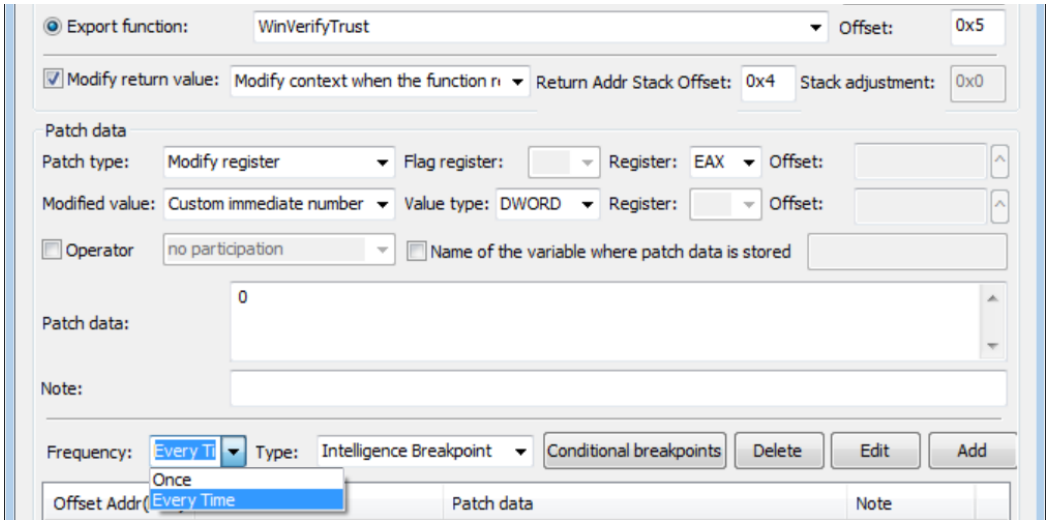
for all threads by default. Can we rely on existing functions to fix the return value of a function when it is called frequently by multiple threads? The answer is yes, let's take the exported function WinVerifyTrust in wintrust.dll as an example to explain.

First HOOK the function so that all threads calling the function will call the "breakpoint setting function" to set a hard break. Then set a hardware breakpoint at the +5 address of the function.



The reason for setting offset +5 is that Baymax's HOOK library will modify the function header to JMP XXXXXXXX instruction, and jump back to the next instruction of the overwritten data of the original function after executing the internal process, the function is overwritten by exactly 5 bytes, if the overwritten instruction is more than 5 bytes, choose the offset of the next instruction of the overwritten data. Check "Modify Return Value", the return address of the

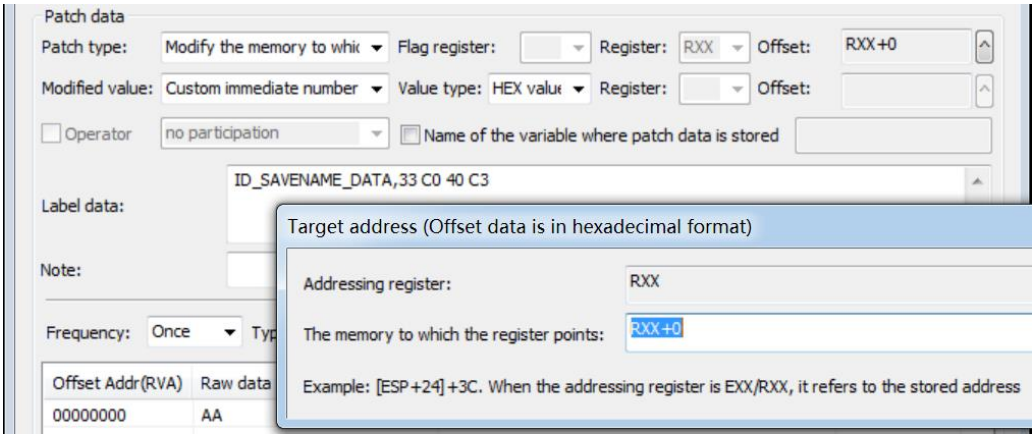
function at offset +5 is located at [ESP+4], so we set "Return Address Stack Offset" to 0x4.



The settings are as shown above, which indirectly achieves the function of fixing the return value of the function by setting a hard break on the calling thread through the completion of the HOOK function.

(iii) Patch heap space by storing data

Some programs map key code to memory (heap space) for execution. We can save the address after the program requests memory space, wait for the process to finish filling the heap space with data, and set a breakpoint at any address before the execution of the heap space code, using the memory address + offset to locate the key address.



When we select "Modify memory pointed to by memory address", we find that the register column is EXX/RXX and we can also set the offset.

Bayamx supports secondary addressing of our saved address, supporting addressing operations like [EXX+N]+N1, so that we can easily modify the values of its members after we save the class object. For example, if we save the address 10000000, and we need to modify the address 10000008, we set the offset to EXX+8 (x64 bit RXX+8).

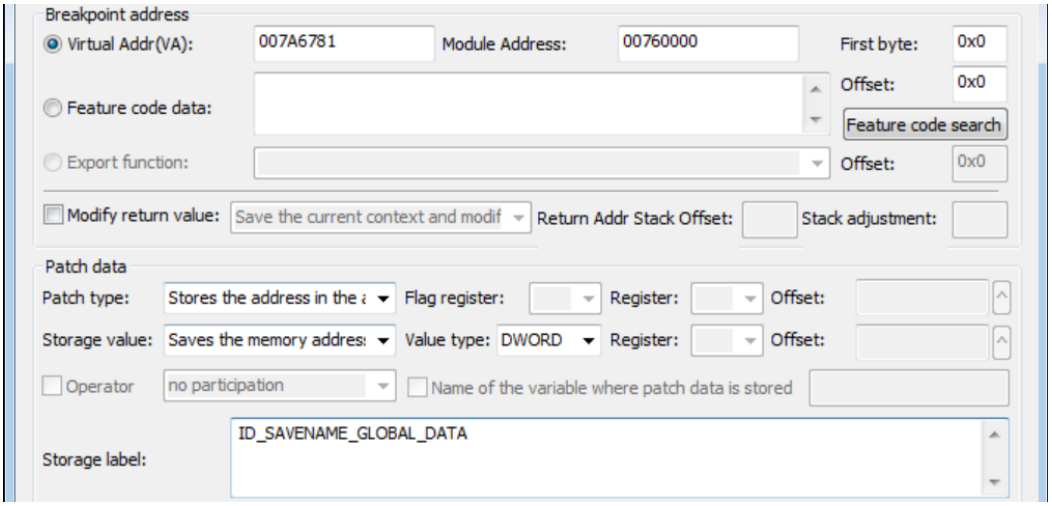
(iv) Modify the value of global variables in instructions

Many programs store the registration status in a global variable in the data segment and determine its status by means of the cmp instruction.

```
007A677E    880C10          mov byte ptr ds:[eax+edx],cl
007A6781    833D D8E88100 00  cmp dword ptr ds:[81E8D8],0
007A6788    75 56           jne xshell.7A67E0
007A678A    B9 90000000     mov ecx,90
```

Baymax provides a mechanism to store and then modify the

value of 0081E8D8 to 1.



Step 1: Store first.

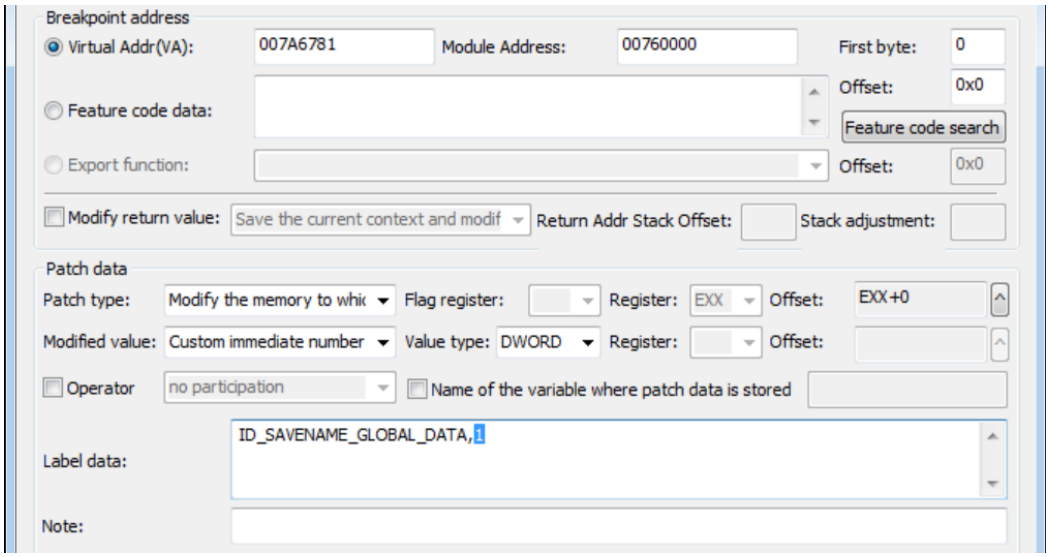
Set the interrupt address, and then set the following in the patch data.

Select "Store current data" for the patch type.

Store the value as "memory address in the save instruction".

The value type is "DWORD" (usually QWORD in x64).

The storage label is filled with a unique label named after the address.



Step 2, then modify it. It is still necessary to set the interrupt address (which can be the same as the storage address), which is set in the patch data as follows.

Select "Modify memory pointed to by memory" for the patch type;

The patch data can be selected as "Custom";

Value type "DWORD"; the format of the marker data is "marker name, patch value", here we should fill in the previously entered marker name and modify the value of "ID_SAVENAME_ GLOBAL_DATA,1".

By doing the above two steps, we achieve the modification of this address (global variable).

(V) Fixed hard disk information through HOOK

The machine code of some programs is generated by the

hard disk serial number operation, can we fix the hard disk serial number by Baymax without modifying the program?

The DeviceIoControl function can be called to obtain hardware information (see below for function declaration). When the parameter dwIoControlCode = 0x0007C088 (SMART_RCV_DRIVE_DATA), the data received by lpOutBuffer after the function returns contains the hard disk serial number information.

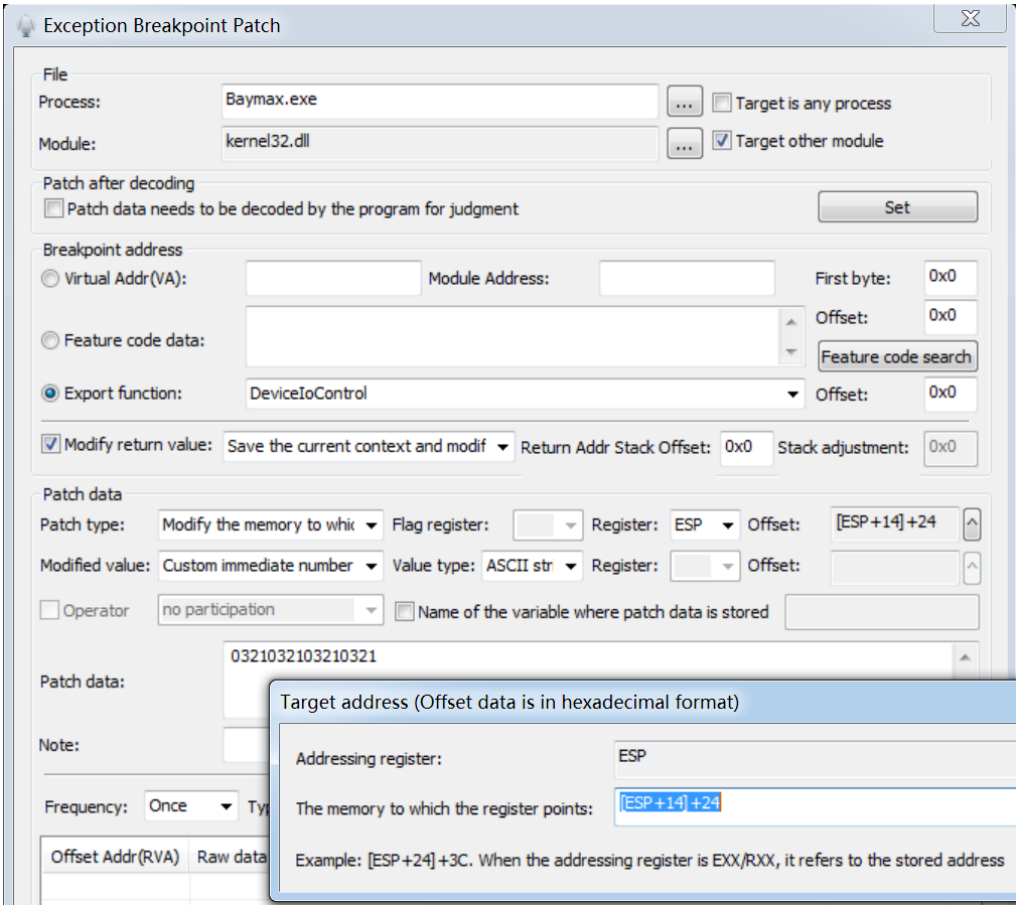
C++ 复制

```
BOOL DeviceIoControl(
  HANDLE       hDevice,
  DWORD        dwIoControlCode,
  LPVOID       lpInBuffer,
  DWORD        nInBufferSize,
  LPVOID       lpOutBuffer,
  DWORD        nOutBufferSize,
  LPDWORD      lpBytesReturned,
  LPOVERLAPPED lpOverlapped
);
```

We create the interrupt patch item, check "Target other modules" at the patch module, select the system module "\Windows\system32\kernel32.dll", and select the patch address from the export function Select the "DeviceIoControl" function from the export function. Check "Modify return value" and select "Save current context and modify on return" mode, and the debugger will break the stack at the function address as shown in the figure below.

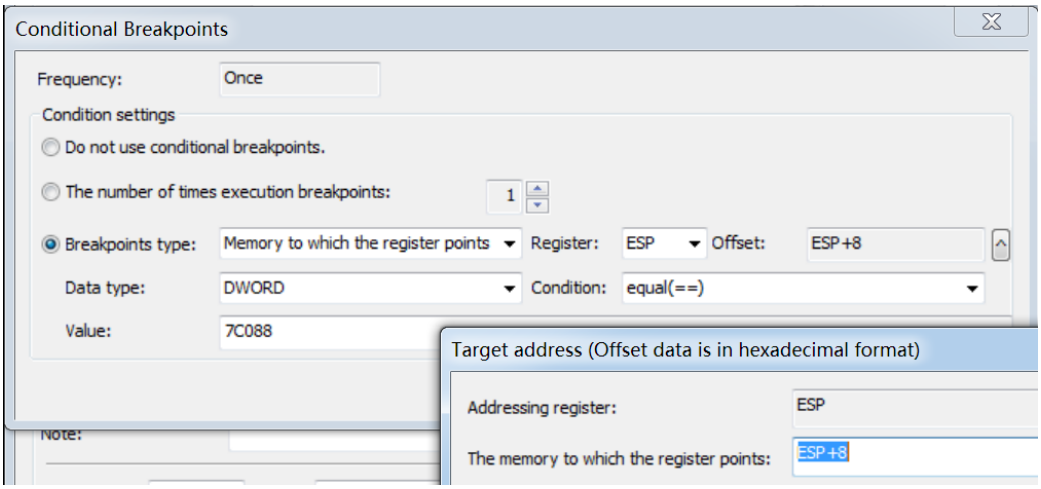
```
$ ==>    > 0092DD88  GetPcInf.0092DD88
$+4      > 000000E4
$+8      > 0007C088
$+C      > 0025E9D8
$+10     > 00000021
$+14     > 0025E7C0  // 參數 lpOutBuffer
$+18     > 00000210
$+1C     > 0025EC40
$+20     > 00000000
$+24     > 0025EE78
0025E7C0  00 02 00 00 00 00 01 00 00 A0 EC 00 00 00 00 00
0025E7D0  40 00 FF 3F 37 C8 10 00 00 00 00 00 3F 00 00 00
0025E7E0  00 00 00 00 30 33 32 31 30 33 32 31 30 33 32 31  ....032103210321  // 偏移0x24
0025E7F0  30 33 32 31 20 20 20 20 00 00 F0 4E 00 00 41 53  0321    ..
```

The parameter is stored in [ESP+14], and the sequence number is located at lpOutBuffer offset 0x24. We set the patch type to "modify register to memory", the register is set to ESP, and the patch memory address is [ESP+14]+24.



Check the "Conditional Breakpoint" button with the

condition [ESP+8]==7C088. note that the "Patch Memory Address" field is always the data address, i.e. ESP+8. set as follows.



After creating the patch and running it we found that the hard drive serial number obtained by the program has been changed to our custom ASCII string. We fixed the hard drive serial number without modifying any bytes of the program.

VIII. Communication Feedback

(i) How to analyze the reasons for patch failure

1. Please close the debugger first

The crack module has an anti-debugging mechanism, if the crack module is not loaded or the LOGO pop-up box is not seen at startup, please **close the debugger** and then test again. If you use the debugger to dynamically analyze the crack

module, it may cause the patch function to be abnormal.

2. Analyze the LOG of debugging patch

If the patch is not loaded as expected, please select Generate debug hijacking patch or debug injection patch from the menu, the program will create xxx_baymax.log file in the process folder after running (if the program is located in the system disk, you need administrator privileges to run it to create the .log file), 32-bit processes can also open the DbgView tool to receive debug information from the patch output to view it. We can use the output logs to analyze the patch execution and find the reason for failure. If you have no way to start, you can also add the feedback to the tool group, and you can attach the log file when giving feedback.

```
2019-07-25 22:34:39:931: [2128][BAYMAX64]: PYG.DLL ver: 3.0.1.1025 模块加载 ↓
2019-07-25 22:34:39:931: [2128][BAYMAX64]: Process Attach: C:\User[illegible]esktop\Debug\L[illegible]示\DiskT
2019-07-25 22:34:40:117: [2128][BAYMAX64]: Not Find Baymax IniFile ↓
2019-07-25 22:34:40:118: [2128][BAYMAX64]: Proc DiskTest.exe Module DiskTest.exe Name DiskTest.exe ↓
2019-07-25 22:34:40:118: [2128][BAYMAX64]: 补丁需要进行HOOK处理：DiskTest.exe ↓
2019-07-25 22:34:40:119: [2128][BAYMAX64]: 增加 HOOK 2 方案 user32.dll ==> CreateWindowExW ↓
2019-07-25 22:34:40:119: [2128][BAYMAX64]: 初始化完成 ... ↓
2019-07-25 22:34:40:119: [2128][BAYMAX64]: Add Hook: Type 2 user32.dll ==> CreateWindowExW ↓
2019-07-25 22:34:40:119: [2128][BAYMAX64]: End StartHook() ↓
2019-07-25 22:34:40:119: [2128][BAYMAX64]: 补丁设置初始化完成，若有HOOK或下断点操作，将会在下方进行打印输出。
2019-07-25 22:34:40:252: [2128][BAYMAX64]: Proc DiskTest.exe Module DiskTest.exe Name DiskTest.exe ↓
2019-07-25 22:34:40:252: [2128][BAYMAX64]: HOOK函数，执行补丁数据。 ↓
2019-07-25 22:34:40:253: [2128][BAYMAX64]: 设置断点补丁条目 ↓
2019-07-25 22:34:40:253: [2128][BAYMAX64]: 非 NS_TYPE_SETRVABREAK 类型 0 ↓
2019-07-25 22:34:40:253: [2128][BAYMAX64]: 断点补丁地址 0[illegible]补丁数据 RSP,[illegible]V:2,R:1,B:102,T:1,C:C
2019-07-25 22:34:40:253: [2128][BAYMAX64]: 设置断点 ↓
2019-07-25 22:34:40:254: [2128][BAYMAX64]: 解析异常断点数据成功 ThreadId: 2496 ↓
2019-07-25 22:34:40:254: [2128][BAYMAX64]: 设置INT3断点 ↓
```

3. Hijacking module loading time is too late

In some processes, due to late loading of the hijacked module, the patch address has already been executed when the cracked module is loaded, thus making the patch invalid. In this case, you need to set the debugger to "interrupt when module is loaded" and dynamically debug to check whether the crack module is loaded when the patch address is interrupted.

(ii) Join the official communication feedback group

Baymax Official Q Group: 112823588

(C) Tools Download

GitHub: https://github.com/sicaril/Baymax-Patch-toOls

(iv) Update log

GitHub: https://github.com/sicaril/Baymax-Patch-toOls