

Baymax Patch Tools Help Documentation

Hijacked Patch Full Guide

Ver 1.7

October 2025

catalogs

I. Overview of tools Overview of tools	3
(i) Introduction to the software	3
(ii) Scope of application	4
(iii) How to test	4
(iv) Disclaimer	5
II. Creating a Patch Project	5
(i) Setting up the target process	5
(ii) Adding patches and setting up patch processes and modules	6
(iii) Setting the patch address	7
1. Offset Patch	7
2. Interrupt Patch	8
(iv) Patch-hardened programs	10
1. Use HOOK mode (recommended)	10
2. Setting the interrupt mode	13
3. Setting the delay mode	13
III. Search for replacement patches	14
(i) Offset patches	14
(ii) Feature code replacement patches	15
IV. Exception handling patches	17
(i) Setting the interrupt address	17
(ii) Types of interruptions	17
(iii) Conditional breakpoints	19
1. Determine how many interruptions	19
2. Judging register values	19
3. Determine the value of the memory pointed to by the registers	20
4. Locating memory addresses by register offsets	20
(iv) Storage of data	23
(v) Modifying registers	24
(vi) Modify the flag register	25
(vii) Modifying the memory pointed to by registers	26
(viii) Modification of EIP/RIP	29
(ix) Production of memory registers	30
(x) Modify the memory pointed to by the storage address	31
(xi) Add anti-debugging detection	32
(xii) Modifying the return value of a function	33
1. Function Call Conventions	33
2. Three modifications to the function return value mode	36
3. A better understanding of arbitrary instructions are located in the function body	38
4. Arbitrary stack address return before patching	38
(xiii) Arithmetic operations on data	39
(xiv) Reading patch information from INI files	39

V. Generating Patches and Saving Patch Projects	41
(i) Description of menu items	41
(ii) Creation of hijacking, injection patches	46
(iii) Creating debug patches.....	47
(iv) Hijacking and cracking modules	48
(v) Saving and opening patch projects.....	49
VI. Plug-in Introduction.....	50
(i) PE file binary comparison	50
(ii) Feature code search tools	51
(iii) Address format conversion tools	52
(iv) String to hexadecimal tools.....	52
(v) Hijacking code generators.....	53
(vi) Process Hijack DLL Detector	55
VII. Cases and tips for use	56
(i) Setting hardware breakpoints on other threads via HOOK implementation	56
(ii) Fixed parameters or return values through the function HOOK.....	57
(iii) Patch heap space by storing data	59
(iv) Modifying global variable values in instructions.....	60
(E) Fixing hard disk information by HOOK	62
VIII. Exchange of feedback	65
(i) How to analyze the cause of patch failure	65
1. Please close the debugger first.....	65
2. Analyze the LOG for debugging patches.....	65
3. Hijacking module loaded too late	66
(ii) Join the official communication and feedback group	67
(iii) Tool downloads.....	67

I. Overview of tools Overview of tools

(i) Introduction to the software

Baymax Patch Tools (Baymax for short) loads the function module PYG.DLL/PYG64.DLL through the hijacked DLL released by the target process, realizing dynamic patching of 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 the process r3, which simulates the exception handling of the debugger. It supports setting breakpoints on the address of the target module, and modifying its corresponding registers, flag registers, and memory data pointed by registers after the break, so as to realize cracking and killing the target file without destroying it.

The tool is protected by a shell, the antivirus may misdiagnose the tool and the patch file! 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 in the system when running (except for overwriting patch files). The tool itself has a validation mechanism, the startup validation

module will be loaded only after the success, but for security reasons, please be sure to download from the official use.

(ii) Scope of application

Baymax is available for Win x86/x64 platforms. It has been adapted to WinXP, Win7, Win8, Win8.1, Win10, Win11, etc. Theoretically, it patches all processes that can load hijacker modules, supports some .NET programs, and basically does not support scripting language programs.

(iii) How to test

Baymax is intended for rapid verification of patch solutions and is limited to technical communication only, please do not use it for copyright infringement or commercial purposes.

The function module of Baymax is PYG.DLL/PYG64.DLL, the function module has a simple built-in file name check, and will not perform a patch operation after modification. Therefore, you can check whether the file exists in the process directory or whether the process has loaded the module as the basis of detection.

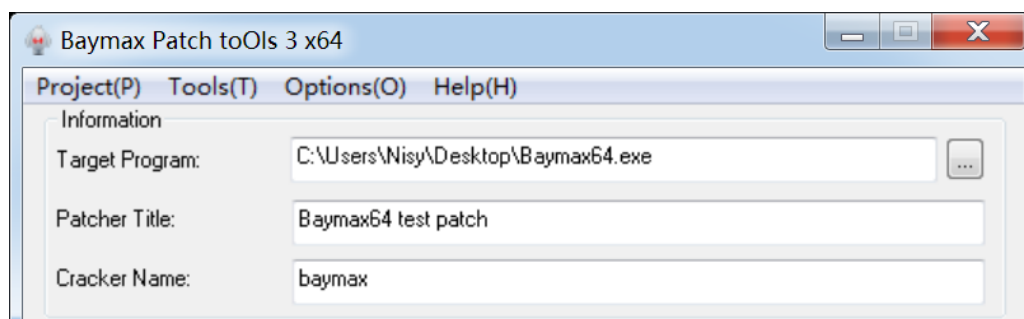
(iv) Disclaimer

This tool is for technical communication only, please do not use it for commercial or illegal purposes! This author has nothing to do with any effects caused by the patch files you create! By clicking OK on the disclaimer when you use the tool, you agree to the terms and conditions.

II. Creating a Patch Project

(i) Setting up the target process

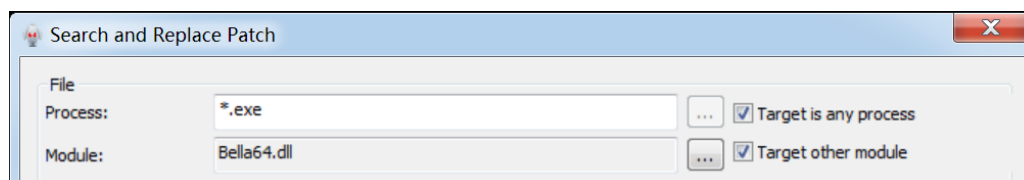
The patch program first selects a target file, and the target program set in the interface will be used as a reference for the generated patch program to intelligently release the hijacked file. If the patch program does not find the file, a pop-up box will prompt the user to select the target object.



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

Add a search and replace or exception 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 support patch project contains multiple patch items, each patch item can be set to patch a different process, you can also set the patch all processes (check the target for any process), support for the process of different DLL or other loaded modules (set the target for other modules, you can set the process in each patch item loaded by a module) for patching.



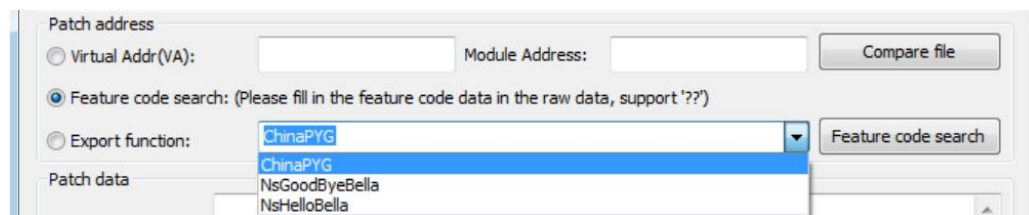
When you check the target for other modules, you don't need to care when the module is loaded, whether it will be uninstalled or reloaded, but if the patched module is not an exe process, Baymax will monitor the loading of the module and try to patch it by default. Of course, you can also handle the patch timing of the module by yourself, you can choose "Customize Patch DLL Timing" in the menu to

disable the smart monitoring function.

(iii) Setting the patch address

1. Offset Patch

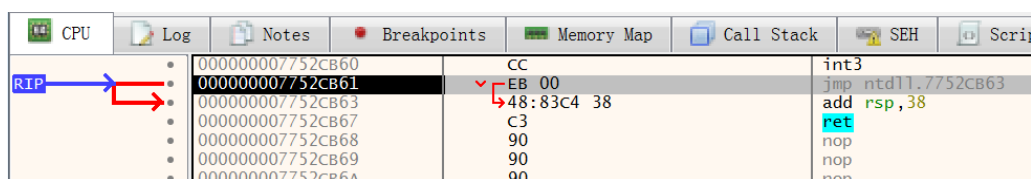
"Search and Replace Patch", referred to as Offset Patch, supports the following three patch address setting options:



- (1) Enter the patch virtual address (memory address) and the module base address
- (2) Locate patch address by feature code
- (3) Locating the patch address via the export function

Baymax supports Dynamic Base Addressing (ASLR), which automatically calculates the RVA for patching by using the patch address and module address entered by the user (no user concern).

In option 1, "Virtual Address (VA)" is the address we see in the debugger, and you can enter it directly.



The "module base address" needs to be viewed in the

process memory through the debugger, and the load address of the module is the module base address.

CPU Log Notes Breakpoints Memory Map Call Stack		
Address	Size	Info
000000013FCE0000	0000000000001000	baymax64.exe
000000013FCE1000	0000000000005E00	" "
000000013FD3F000	0000000000001600	" "
000000013FD55000	0000000000000600	" "
000000013FD5B000	0000000000000500	" "
000000013FD60000	0000000000001A00	" "
000000013FD7A000	0000000000001000	" "
000000013FD7B000	0000000000001000	".idata"
000000013FD7C000	0000000000001300	".rsrc"

Option 2 locates the patch address by a feature code, which supports ???? wildcards. When using this option, it is recommended to use the "Feature Code Search" widget to check if the results meet expectations.

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

2. Interrupt Patch

"Abnormal Interrupt Patch", referred to as Interrupt Patch, also supports three patch address setting schemes:

Breakpoint address

☐ Virtual Addr(VA): Module Address: First byte:

☒ Feature code data: Offset:

☐ Export function: Offset:

☒ Modify return value: Return Addr Stack Offset: Stack adjustment:

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

(2) Locate patch address by feature code

(3) Locating the patch address via the export function

The difference between Option 1 and offset patching is that if the target module has a shell or is to be decoded, setting the INT3 breakpoint also requires inputting the first byte of the instruction corresponding to the patched address, and the patch first determines whether the address has been decoded by judging whether the memory byte is equal to the original value input by the user. If the module is without shell or using hardware breakpoints, just keep the default value 0.

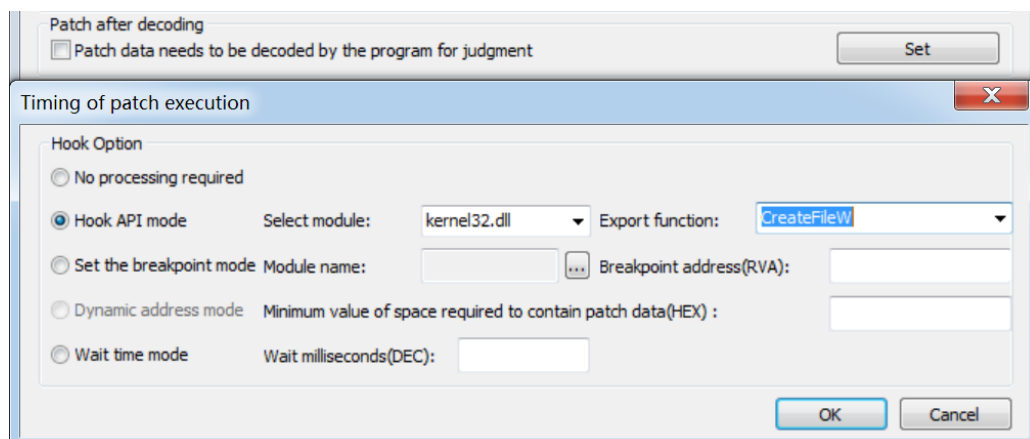
Option 2 can use the feature code's to get the patch address, and set the break after calculating the virtual address through the search result + offset value (the offset value can be negative). It should be noted that Baymax will set break in all the results of the search + offset value, if the search results are more than 4, the use of intelligent breakpoints or hardware breakpoints may result in omissions. After extracting the feature code, be sure to use the feature code search tool to check if the results are as expected, if an interrupt is set at a non-expected address, it may cause the program to crash.

Option 3 some shells will detect whether the API first byte is modified, so it provides the function to break 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 the interrupt is set at the non-instruction first address, there will be unknown results.

(iv) Patch-hardened programs

Baymax supports patch shelled programs. If the target module is shelled or the patch address is to be decoded, the cracking module can not execute the patch operation immediately after loading, and needs to wait for the code segment to be decoded. Users can use the following programs to set the decoding time for the target process and wait for the decoding before executing the patch. Check "Patch data need to be decoded by the program to judge" and set it in the popup box below.



1. Use HOOK mode (recommended)

Choosing the right API by HOOK to determine the decoding time, so as to crack the shell program is an art, this is the accumulation of experience, novices need to explore more practice to master. Here we can share a HOOK method: after the shell decodes the data and executes it to the patch address, the APIs that have been executed during this time can be used as alternatives (the fewer the number of times the APIs are executed, the better).

The program patch shelling program is through the HOOK target function, the process each time the function is called will try to execute a patch program, until a certain call has been decoded to complete and execute the patch. So the fewer the number of times the selected function is called, the better, the fewer the number of times it is called, the smaller the performance loss of the process.

How to choose the right API? The following options are available for reference:

1. Shell commonly used API (heap space application and release, memory attribute modification), the API called at the entrance of the module, the API in the module import table, behavioral operations related API (such as window creation and destruction), debugging information output and

other APIs, a few more tries will always find the right one.

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

If the chosen API causes the program to fail to start or not to patch after starting, please replace it with another API for testing. After finding an available API, we then pick an API with as few calls as possible, which reduces the performance loss of process startup. How to determine how many calls? You can determine the number of interrupts set in the debugger, but also through the Baymax debugging patch log statistics output HOOK function execution.

With this scheme, the patch entries will not be patched when the PYG Crack module is loaded, but the process will determine whether to decode and try to patch once every time it executes to the selected API until all the corresponding patch entries are successfully set or patched, and then it 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. Setting the interrupt mode

If the target process does not detect hardware breakpoints, such as unencapsulated or compressed shells, 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 executes to the address triggering a hardware breakpoint.

3. Setting the delay mode

Programs from startup to run to the patch address must be a time interval, sometimes we do not want to care about the decoding process of the program shell, you can set a waiting time before the patch. This program is a lazy method, may be different system environments will cause unknown results, not universal.

With this scheme, the patch entry will not be patched when the PYG module is loaded, the cracking module will create a waiting thread when it is loaded, and the new thread will perform the patch operation when the user-set waiting time is met.

Note: The number of milliseconds entered is in decimal (the only place in the software setup screen where decimal is entered, 1 second = 1000 milliseconds).

III. Search for replacement patches

(i) Offset patches

Patch address

☒ Virtual Addr(VA): 000000014028D160 Module Address: 000000013FBD0000 Compare file

☐ Feature code search: (Please fill in the feature code data in the raw data, support '?')

☐ Export function: ChinaPYG Feature code search

Patch data

Raw data: 41 52 49 89 E2 41 52

Patch data: 41 53 49 89 E2 41 53

Note:

Delete Edit Add

Adds an offset patch entry:

First enter the patch address: either enter the memory address and module base address in Option 1, or select the export function in Option 3.

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

If there is a lot of patch data, you can use the "Compare Files" function to compare the original file and the modified file to add the patch data at one time.

(ii) Feature code replacement patches

00007FF7A4167409	48:8365 18 00	and qword ptr ss:[rbp+18],0
00007FF7A416740E	48:BB 32A2DF2D992B0000	mov rbx,2B992DDFA232
00007FF7A4167418	48:8B05 D1163F00	mov rax,qword ptr ds:[7FF7A4558AF0]
00007FF7A416741F	48:38C3	cmp rax,rbx
00007FF7A4167422	75 6F	jne binaryninja.7FF7A4167493
00007FF7A4167424	48:8D4D 18	lea rcx,qword ptr ss:[rbp+18]
00007FF7A4167428	FF15 F2D80100	call qword ptr ds:[<&GetSystemTimeAsFileTime]
00007FF7A416742E	48:8845 18	mov rax,qword ptr ss:[rbp+18]
00007FF7A4167432	48:8945 10	mov qword ptr ss:[rbp+10],rax
00007FF7A4167436	FF15 ECD80100	call qword ptr ds:[<&GetCurrentThreadId]

Adds a feature code search patch entry:

In order for patches to support subsequent releases, we often extract feature codes from the patch address to make a pass-through patch. We use the "feature code 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 relative offset addresses in instructions, as the relative offset addresses may change in subsequent versions. For example, the instructions in the box above have two assembly codes that contain relative offset addresses (standardized by the red arrows).

00007FF7A416740E	48:BB 32A2DF2D992B0000	mov rbx,2B992DDFA232
00007FF7A4167418	48:8B05 D1163F00	mov rax,qword ptr ds:[7FF7A4558AF0]
00007FF7A416741F	48:8B05 CA163F00	mov rax,qword ptr ds:[7FF7A4558AF0]
00007FF7A4167426	4D:18FF	sbb r15b,r15b
00007FF7A4167429	15 F2D80100	adc eax,1D8F2
00007FF7A416742E	48:8845 18	mov rax,qword ptr ss:[rbp+18]
00007FF7A4167432	48:8945 10	mov qword ptr ss:[rbp+10],rax
00007FF7A4167436	FF15 ECD80100	call qword ptr ds:[<&GetCurrentThreadId]
00007FF7A416743C	8BC0	mov eax,eax

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

Patch address

☐ Virtual Addr(VA): 000000014028D160 Module Address: 000000013FBD0000 Compare file

☒ Feature code search: (Please fill in the feature code data in the raw data, support '?')

☐ Export function: ChinaPYG Feature code search

Patch data

Raw data: E8 ?? ?? ?? ?? 90 E9 ?? ?? ?? ?? 8B 0D ?? ?? ?? ?? F6 C1 03 74 31

Patch data: E8 ?? ?? ?? ?? 90 E9 ?? ?? ?? ?? 8B 0D ?? ?? ?? ?? F6 C1 03 74 00

Note:

Delete Edit Add

Select Option 2 Feature Code Search, enter the original data and patch data, and then 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 length of valid data must be the same.

It is recommended to use baymax plug-in for x64dbg to get the feature code data:

00000000774E2F20 ^ E9 19EFFF jmp ntdll.774C1E3E
 00000000774E2F25 L 8B0D 057A0D00 mov ecx,dword ptr ds:[775BA930]
 00000000774E2F2B F6C1 03 test cl,3
 00000000774E2F2E v 74 31 je ntdll.774E2F61

Signature search

Module list: ntdll.dll | ModuleBase:000077480000 SizeOfImage:1A9000

Signature: E9 ?? ?? ?? ?? 8B 0D ?? ?? ?? ?? F6 C1 03 74 31

☐ Search String

Ordinal	Virtual Address	Relative Virtual ...	File Offset Addr...	Section Name	Characterist..
1	0000774E224D	0006224D	0006164D	".text"	ER---
2	0000774E2F20	00062F20	00062320	".text"	ER---
3	00007751B6B8	0009B6B8	0009AAB8	".text"	ER---

Found 3 in total(<16ms) List of Modules ☐ Search all items in the list Search

IV. Exception handling patches

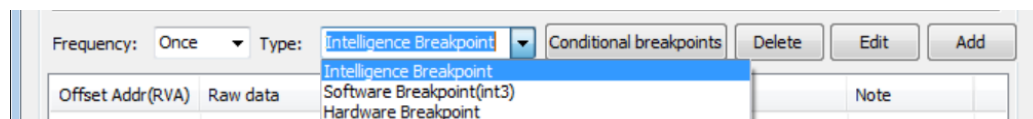
(i) Setting the interrupt address

We use the debugger to analyze the program, first set a breakpoint on the module, run the interrupt before debugging and analyzing, the design of the abnormal interrupt patch idea is also the same, the first to set the interrupt address, the interrupt before the patch operation.

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

(ii) Types of interruptions

There are currently 3 types of interrupts: intelligent breakpoints, INT3 breakpoints, and hardware breakpoints.



1. Hardware breakpoints: Hardware breakpoints have thread correlation, each thread can set 4 hardware breakpoints at the same time, by default, they will be set on the thread where the cracking module is loaded (usually the main thread), and if the HOOK API is used, they will be

set to the corresponding thread where the API is executed. Currently there is no function to set hardware breakpoints on 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 the process memory checksum.

2. Intelligent breakpoints: Since each thread can only set up to 4 hardware breakpoints at the same time, when the user sets up more than 4 hardware breakpoints at the same time, the setup will fail. At this time, the user can choose intelligent breakpoints, intelligent breakpoints are essentially equal to hardware breakpoints, each time a hardware breakpoint is triggered, at the end of the execution if there is a free DrX, it will detect whether there are still unset breakpoints in the queue and set them up in order, the order of the queue and the user to add the order of the interface is consistent with the order of the interface, if the settings are reasonable, it can be indirectly set to use more than one hardware breakpoints.

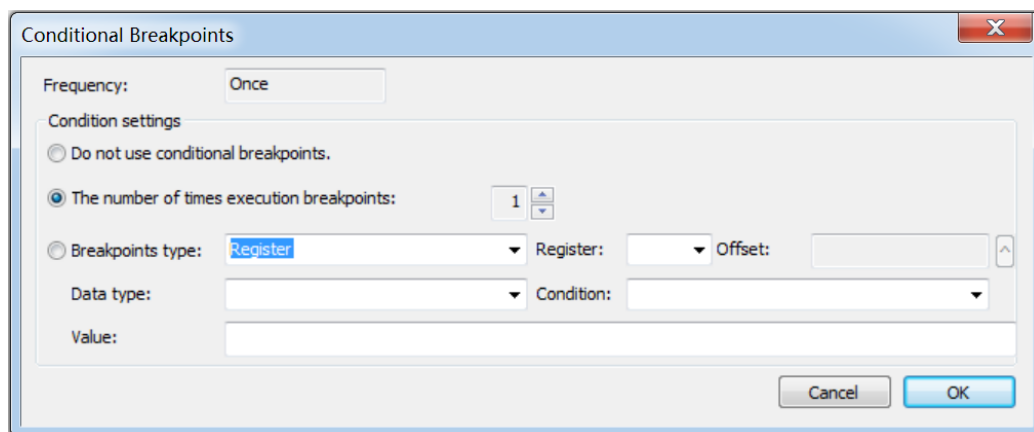
3. INT3 breakpoints: INT3 is different from the hardware breakpoints, without thread relevance, set up after the

process of all threads executed to the address will trigger an interrupt. If the set hardware settings are invalid, may be because the thread is not set on the breakpoint, can be modified to INT3 breakpoints for testing.

Using an INT3 breakpoint will modify the memory pointed to by the patch address to 0xCC, which will revert to the original instruction at the end of execution. INT3 breakpoint mode is not made multi-thread compatible, and unpredictable results may occur if multiple threads execute to the patch address at the same time.

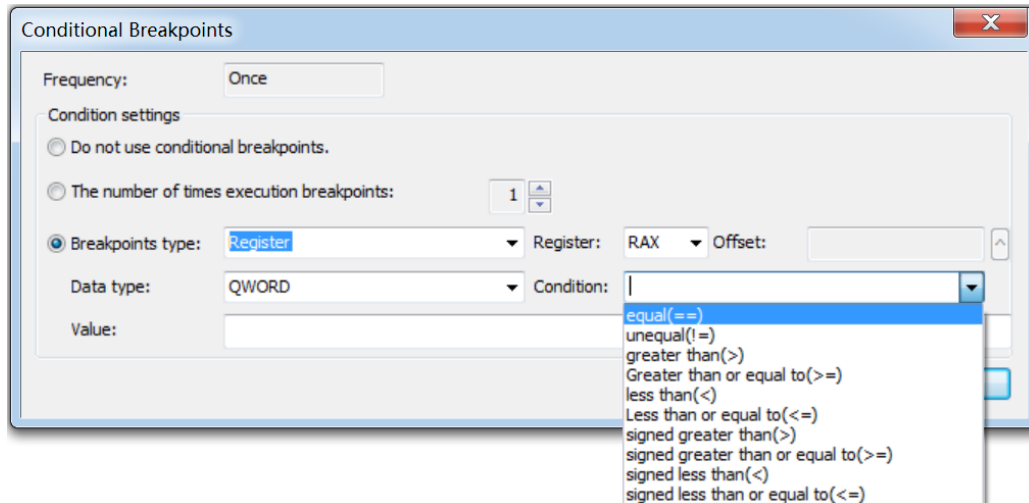
(iii) Conditional breakpoints

1. Determine how many interruptions



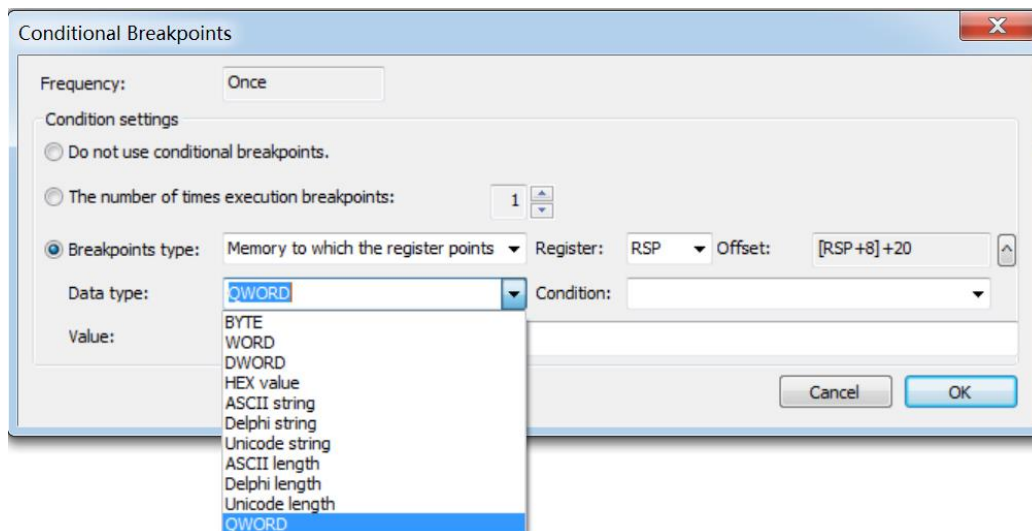
Conditional breakpoints can set the patch address to execute the patch program after the first few interrupts.

2. Judging register values



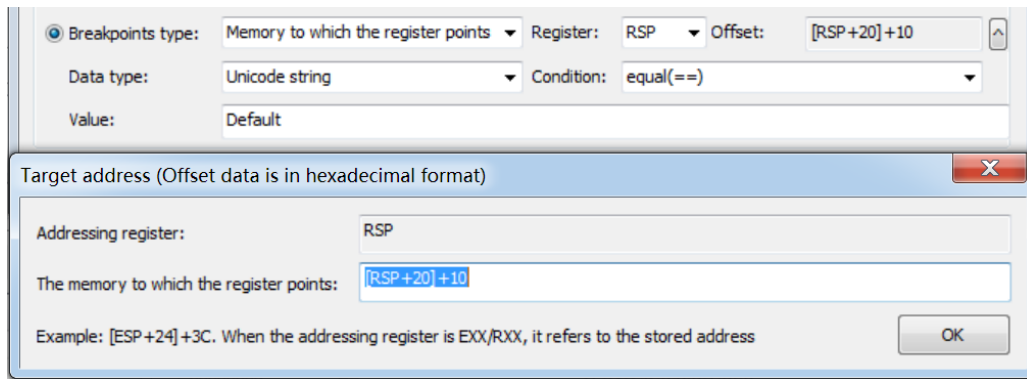
Conditional breakpoints allow you to compare the values of registers after an interrupt as a basis for whether or not to execute a patch.

3. Determine the value of the memory pointed to by the registers



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

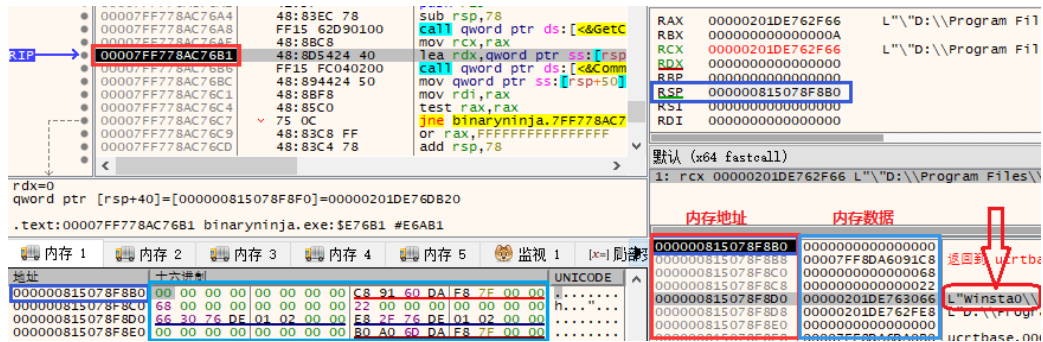
4. Locating memory addresses by register offsets



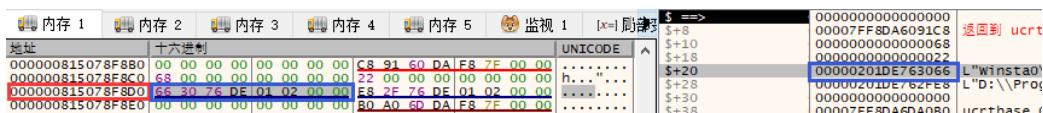
Conditional breakpoints and subsequent patch settings require knowledge of how to set the "memory pointed to by registers", which is explained here.

The register + offset is used to locate the memory address of the data, and the entry in 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 program.

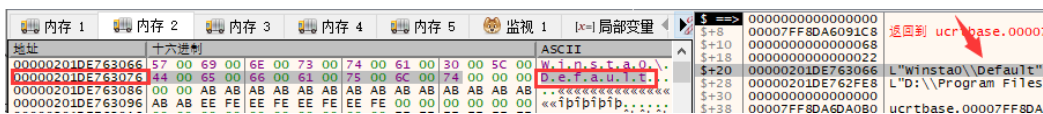
At the interrupt address, we first select a register, the interface default display offset value is 0x0 (all data entered here are in hexadecimal). As shown in the above figure, the default data in the input field is: RAX+0, which represents taking the value of register RAX+ offset to 0 as the memory address.



In the above figure, RIP: 00007FF778AC76B1, RSP: 00000815078F8B0, how can we locate the address of the string L "winsta0" in the figure by registers?



We observe that the memory address at offset 0x20 of RSP is 00000815078F8D0 (RSP+0x20), which points to the memory data QWORD value of 00000201DE763066, which is exactly the address of the string. So we first set the RSP offset to 20, that is, RSP + 20, and then take its value can be, that is, enter [RSP + 20].

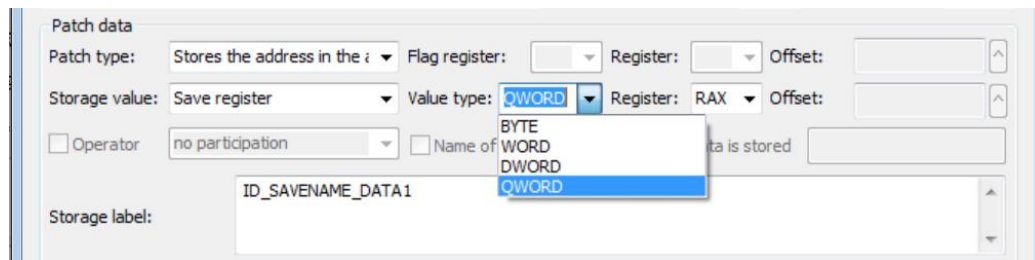


So how to locate the address of the string L "Default"? First locate the address of the string and then add an offset of 0x10, i.e. enter [RSP+20]+10.

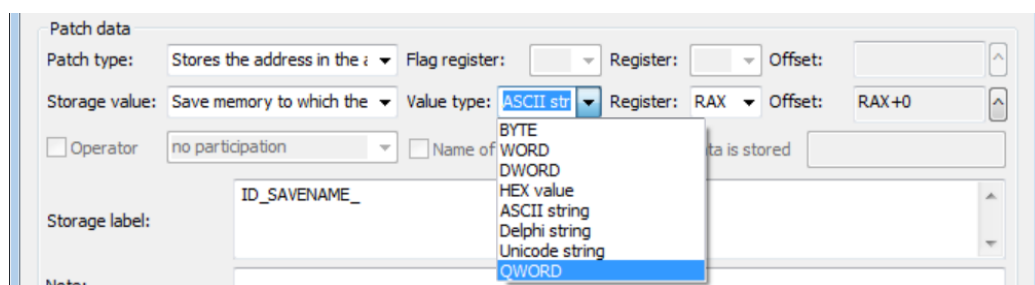
Baymax supports multi-level offset addressing, which makes addressing the internal members of a class object via its address much more flexible and convenient.

(iv) Storage of 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.

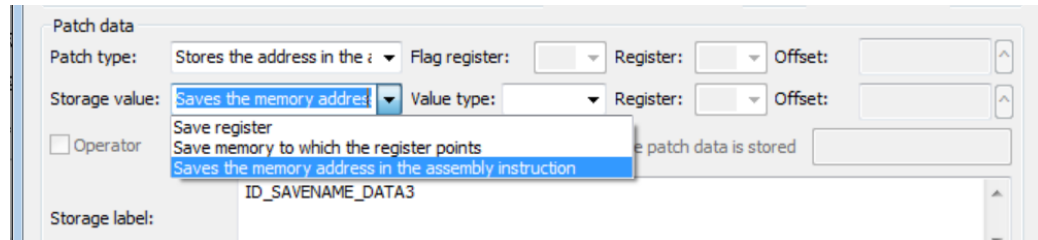


Select the type of stored value and the corresponding register, and enter the unique name (global variable name) we named for the stored data in the "Storage Label" field, so that we can refer to it when we use it later. Note that the type of data stored must be the same as the type of data referenced later.



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

after the comma is the length of the HEX data. 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 storage instruction.

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

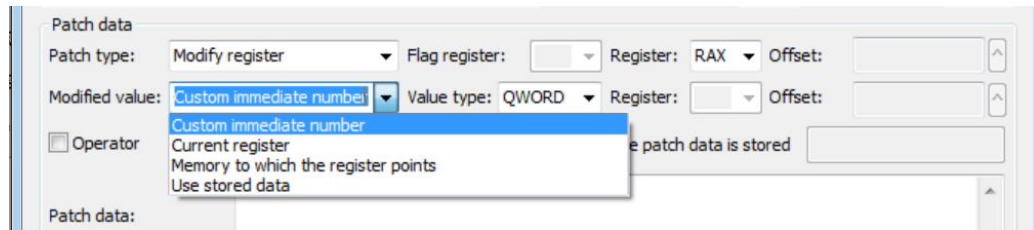
For example, if the interrupt address is 7738E9FB, select "Save memory address in instruction" for the storage value, set the storage marker, and store the DWORD value 773FE774 after the interrupt.

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

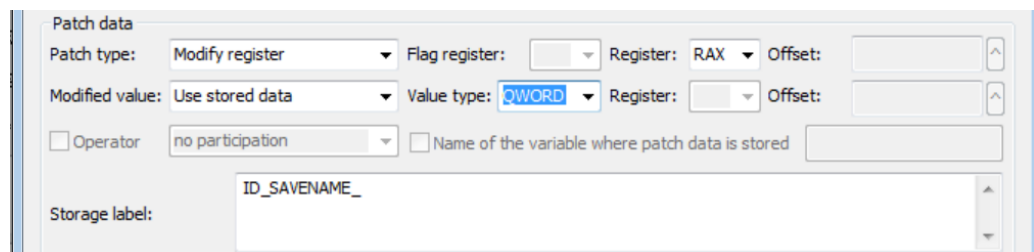
(v) Modifying registers

When an interrupt is triggered, the register value

corresponding to the interrupt address can be modified.



Patch data can be customized data, just enter it in the Patch Data field, note that the value must not exceed the selected value type; you can also select other registers, for example, the registers point to the true and false code respectively during interrupt, you can replace the value of the register storing the false code with the value of the register storing the true code; or you can replace it with the data in the register pointing to the memory.

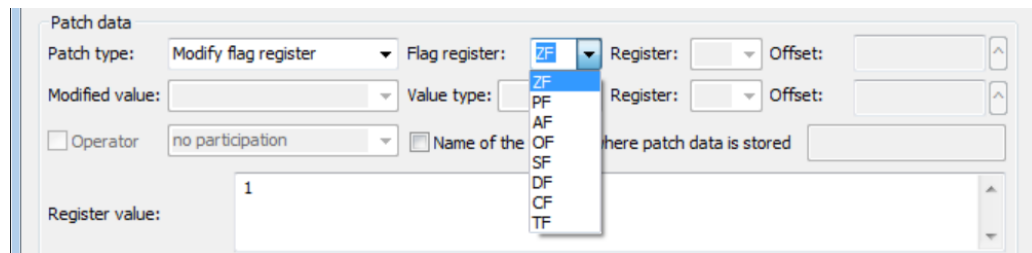


It is also possible to use data that we have stored before, using stored data must ensure that the value types are consistent. Enter the name of the data we have stored in the Storage label field.

(vi) Modify the flag register

When an interrupt is triggered, the flag register

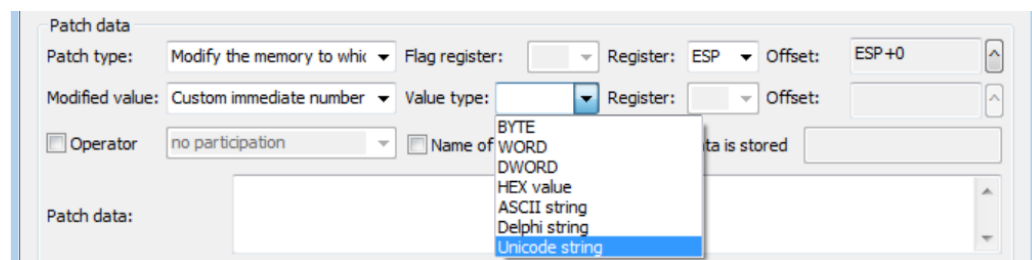
corresponding to the interrupt address can be modified.



Modify the instruction jump, you can modify the flag register to change the execution process, the same interrupt address can be added to more than one patch entry to realize the modification of more than one flag register, fill in the "register value" to modify the value of 0 or 1.

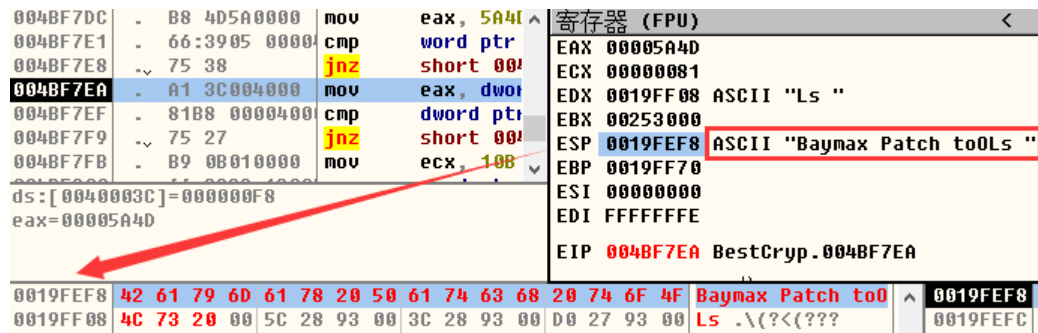
(vii) Modifying the memory pointed to by registers

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

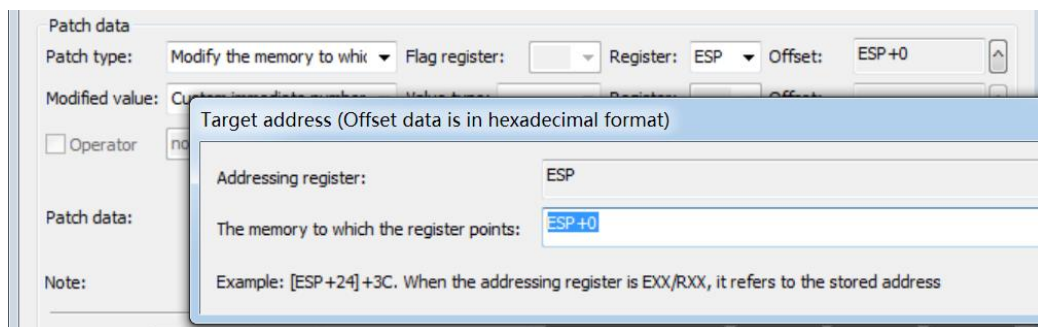


Modifying values supports customizing the number of immediately (e.g. replacing the public key of RSA or decoded data), using the current register value, using the memory pointed to by the register (e.g. two registers pointing to memory data as a comparative value, which can be replaced before comparing them), and also using stored data.

The method of locating memory via registers was explained in detail in the "Conditional Breakpoints" section above, based on the principle 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:



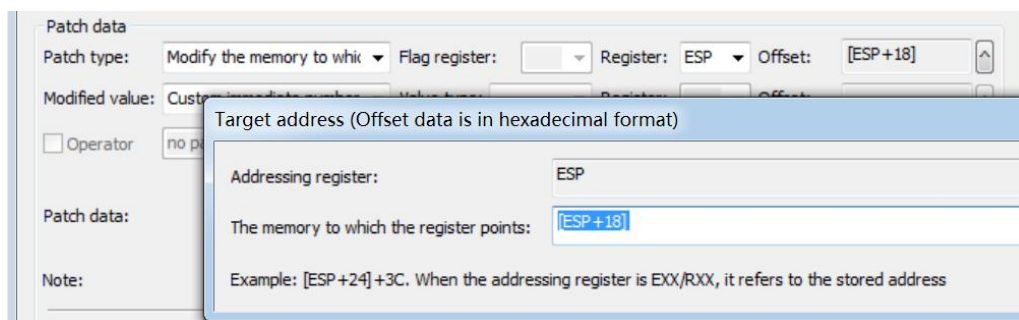
Patch memory address input: ESP or ESP+0 are available.



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

	EAX	00000000
	ECX	00000081
	EDX	0019FF08
	EBX	003FA000
	ESP	0019FEF8
	EBP	0019FF70
\$ ==>	0FAB0BC0	
\$+4	004BF938	BestCryp.<模块入口点>
\$+8	004BF938	BestCryp.<模块入口点>
\$+C	003FA000	
\$+10	00000044	
\$+14	0074285C	
\$+18	0074283C	UNICODE "WinSta0\Default"
\$+1C	007427D0	UNICODE "D:\Program Files (x86)\Jet:
\$+20	00190000	

In the above figure, ESP+18=0019FF10, the memory data pointed to by this address is 0074283C, which is the address of the string "WinSta0...".



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

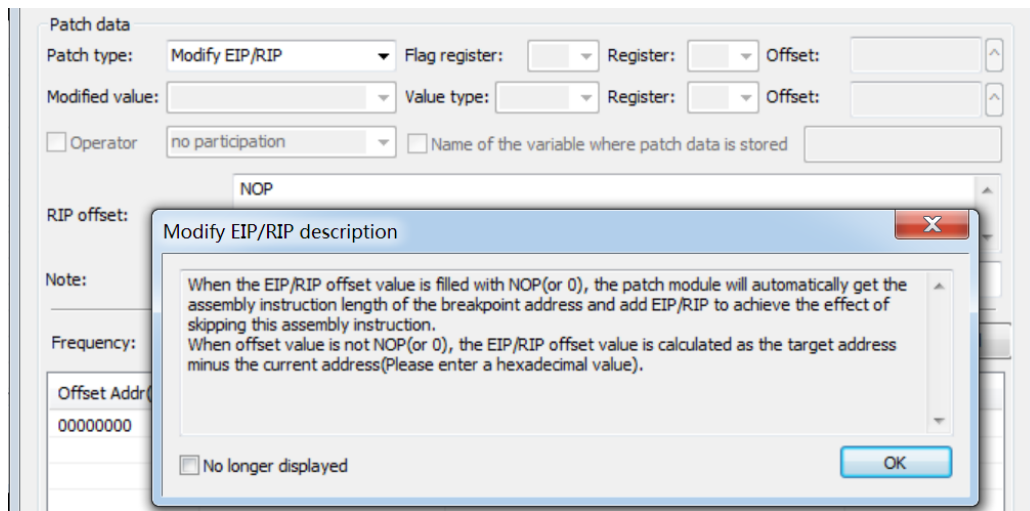
The input format supports multilevel addressing in the format: [[EXX+X]+Y]+Z, where EXX is a register, XYZ is an offset value, and [] stands for fetching the data pointed to by that memory, which takes the value of DWORD for 32-bit processes and QWORD for 64-bit processes.

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 values can be modified when an interrupt is triggered.

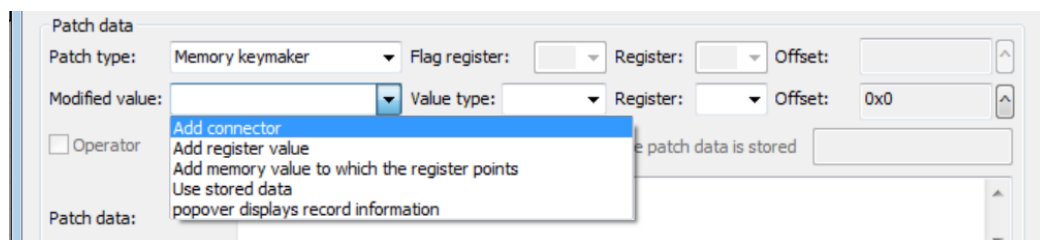


The default offset value is NOP, in fact, it is not to modify the instruction to NOP, but the patch module will automatically get the length of the assembly instruction at the interrupt address and increase the EIP/RIP to realize the effect of skipping the instruction. You can also customize the offset value. When 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 input the hexadecimal value), and it supports positive and negative numbers.

(ix) Production of memory registers

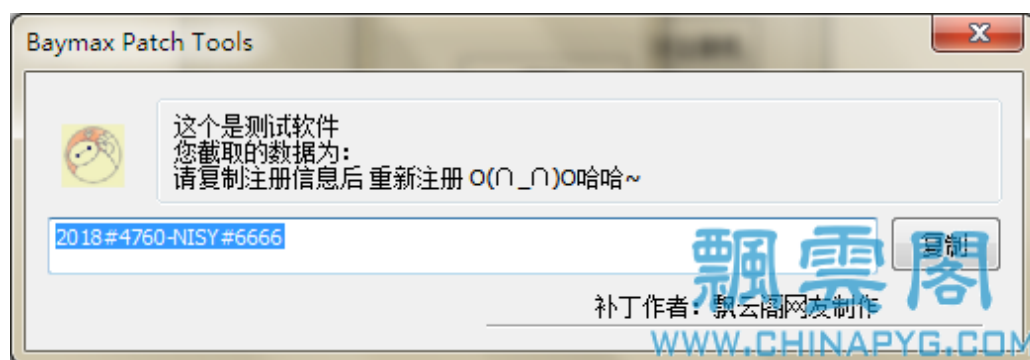
Baymax provides a popup box to display the extracted information, which can collect a lot of information first, and then display it all at once at the end. This feature can also be used as a memory register.

The implementation is a two-step process. The first step is to add data in sequential order, support reading information from registers or memory, support adding connecting characters, and also add stored data.



The second step is to display the intercepted information, after adding all the information to be displayed, you can set an interrupt at any address, and then add a "pop-up window to display the log information", which will trigger an interrupt at the address, and then create a new thread to splice all the data added before and display the pop-up window in order. In addition, when executing to the address of the popup box, the log file will record the information of all registers at that address.

The Patch Description field allows you to customize the popup box information and display the interface as shown below:



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

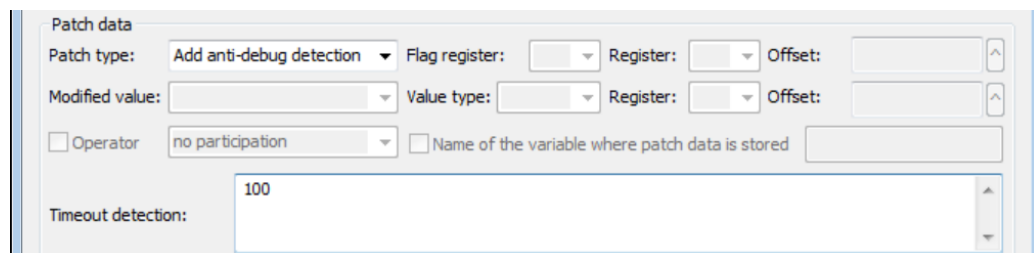
If the stored data is an address, we can set an interrupt at any address to modify the memory pointed to by that stored address.

The modified value can be a customized immediate number,

a register for the interrupt address or memory data pointed to by the register. The format of the input field is "ID_SAVENAME_,0", ID_SAVENAME_ is the name of the DWORD (32-bit process) or QWORD (64-bit process) that we have saved before, and after the comma, we can input the customized data, in which the modification value is "HEX value". If you select "HEX value" when modifying the value of "Memory pointed to by register", please input the length of the HEX data after the comma.

(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's preset value, it is assumed that the program may be being debugged and analyzed, and the program will trigger a crash.



The screenshot shows a 'Patch data' dialog box. The 'Patch type' dropdown is set to 'Add anti-debug detection'. Below it, there are fields for 'Flag register', 'Register', and 'Offset', all of which are empty. There are also fields for 'Modified value', 'Value type', 'Register', and 'Offset', which are also empty. A checkbox labeled 'Operator' is checked, and its value is 'no participation'. Another checkbox labeled 'Name of the variable where patch data is stored' is unchecked. At the bottom, there is a 'Timeout detection' field with a value of 100.

The user preset value is in milliseconds, and the input field is a **decimal** value with a default of 100, a minimum value of 20 (no less than 50 milliseconds is recommended),

and a maximum value of 2000 (2 seconds).

(xii) Modifying the return value of a function

Sometimes we need to modify the parameter or return value of a function. Baymax does not provide a function HOOK function, but simulates the implementation of this function by means of an exception interrupt.

The "Modify Return Value" function can be used for all the patch entries in the interrupt patch item. After the patch address is interrupted, the return address of the function 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, then ESP/RSP + N to restore stack balance. The same is true for simulating function return, so the following three modes of modifying the return value also need to pay attention to these two sets of information: getting the function return address from the stack and the stack balance adjustment value.

1. Function Call Conventions

We need to know a little bit about stack balancing, and

here is a brief introduction to common function call conventions:

1. `__cdecl` is the default function call protocol of C/C++. All parameters are placed on the stack in order from right to left, and the function return is usually `retn`. The stack balance is balanced by the caller, who will execute `add esp, n` to balance the stack after the function return, and `n` is the memory size occupied by the parameter pressure stack.

005CBEA0	. 8B56 20	mov	edx, dword ptr [esi+20]	
005CBEA3	. 52	push	edx	压入参数1 ESP-4
005CBEA4	. 8B4E 1C	mov	ecx, dword ptr [esi+1C]	
005CBEA7	. 51	push	ecx	压入参数2 ESP-8
005CBEA8	. E8 DBD3FFFF	call	005C9288	
005CBEAD	. 83C4 08	add	esp, 8	ESP+8 保证栈平衡

The function implementation is shown in the following figure. `retn` represents a function with no arguments or using the `cdecl` calling protocol.

005C9288	\$ 55	push	ebp	
005C9289	. 8BEC	mov	ebp, esp	
005C928B	. 8B45 08	mov	eax, dword ptr [ebp+8]	函数实现
005C928E	. 8B55 0C	mov	edx, dword ptr [ebp+C]	
005C9291	. A3 284A6600	mov	dword ptr [664A28], eax	retn 可说明无参数或
005C9296	. 8915 2C4A6600	mov	dword ptr [664A2C], edx	使用_cdecl调用模式。
005C929C	. 5D	pop	ebp	
005C929D	. C3	retn		

2. `__stdcall` is the default function call protocol of Windows API. All parameters are placed on the stack from right to left, and the stack is balanced by the function itself. The function returns to `retn n`, so the caller does not need to pay attention to the stack balance.

00401017	. C1E0 02	shl	eax, 2		ESP 0019FF6C	
0040101A	. A3 63675E00	mov	dword ptr [5E6763], eax		ESP 0019FF80	
0040101F	. 52	push	edx		EIP 00401000	屏录专家-<模块入口点>
00401020	. 6A 00	push	0		EDI 00401000	屏录专家-<模块入口点>
00401022	. E8 60401E00	call	<jmp.&KERNEL32.GetModuleHandleA>	[pModule = NULL GetModuleHandleA	EIP 00401022	屏录专家-00401022
00401027	. 8BD0	mov	edx, eax	stdcall 模式, 由函数负责栈平衡		
00401029	. E8 A2FD1800	call	005C00D0			
005E5092	. 5A	pop	edx			
005E5092	. <jmp.&KERNEL32.GetModuleHandleA>					
005E6000	00 20 08 BA 4E 00 08 20 24 21 51 00 00 00 C8 07	. 调... \$!Q...?		0019FF6C	00000000	pModule = NULL
005E6010	5C 00 08 20 D4 07 5C 00 0C 16 08 10 5D 20 5C 00	. \... ?...hmmj...		0019FF70	00401000	屏录专家-<模块入口点>
005E6020	74 BE 5C 00 20 D4 08 0F 5C 00 00 00 5A 20 5C 00	. tmm\... ?...?...		0019FF74	76B76359	返回到 KERNEL32.76B76359

7620B31A	23DE	and	ebx, esi	
7620B31C	8D45 F4	lea	eax, dword ptr [ebp-C]	
7620B31F	50	push	eax	
7620B320	FF15 40802C76	call	dword ptr [&ntdll.RtlFreeUnicodeString]	
7620B326	8BC3	mov	eax, ebx	
7620B328	5B	pop	ebx	
7620B329	5E	pop	esi	
7620B32A	C9	leave		
7620B32B	C2 0400	retn	4	stdcall模式 函数负责平衡栈

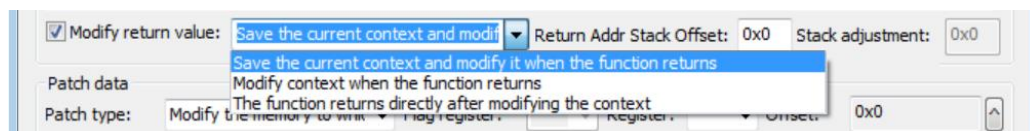
0040101A	A3 63675E00	mov	edx, ptr [5E6763], eax	EDX 00000000	ESP 0019FF70	ESP 0019FF70	
0040101F	52	push	edx	EBP 0019FF80	ESI 00401000	EDI 00401000	EIP 00401027
0040102E	6A 00	push	0				
00401022	E8 6B401E00	call	<jmp.&KERNEL32.GetModuleHandleA>				
00401027	8BD0	mov	edx, eax				
00401029	E8 A2FD1B00	call	005C00D0				
0040102E	5A	pop	edx				
0040102F	FA 005F4000	scasd	005C00D0				
eax=00400000 (屏录专家-00400000), ASCII "MZP"							
edx=00401000 (屏录专家-模块入口点)							
005E0000	00 20 08 BA	HE 00 00 20	24 21 51 00 00 00 C8 07	...	0019FF70	00401000	屏录专家-模块入口点
005E0010	5C 00 00 20	DA 07 5C 00	00 1C 68 10 50 00 00 20	...	0019FF74	76B76359	返回到 KERNEL32.76B76359
005E0020	74 0E 5C 00	00 20 0A 0F	5C 00 00 00 94 20 00 00	...	0019FF78	00B39000	
005E0030	00 00 28 23	5C 00 00 05	5C 49 5C 00 04 0C 37	...	0019FF7C	76B76340	KERNEL32.BaseThreadInitThunk

3. `fastcall` is a function call protocol for x64

processes, where arguments are passed in right-to-left order. Arguments are passed through the RCX, RDX, R8, R9 registers. If there are more than 4 arguments, the stack space for the first 4 arguments is also reserved on the stack, and the subsequent arguments are written to the corresponding stack offsets.

2. Three modifications to the function return value mode

With this knowledge, let's move on to look at three setup options for modifying the return value.

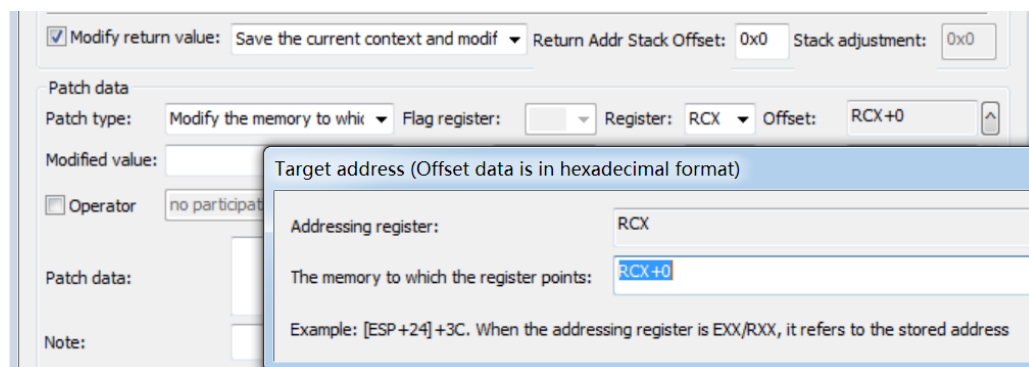


1. Saves the current context and modifies it on return.

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 passed parameter at the function return. In this case, you 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 realized by "storing data" and "modifying the memory pointed to by the storage address", but it is not convenient.

With this program, the user can set the interrupt at any address in the function where the parameter can be obtained, and set the "return address stack offset" when the patch address is interrupted, i.e., the offset value of the return address of the function in the stack after the interrupt relative to the ESP/RSP. In this case, the incoming parameter must be a memory address, so the patch type in the patch data field can only be selected as "Modify memory pointed to by register". The register + offset value set during interrupt is saved, and the memory data pointed to by the saved address is modified after the function returns.



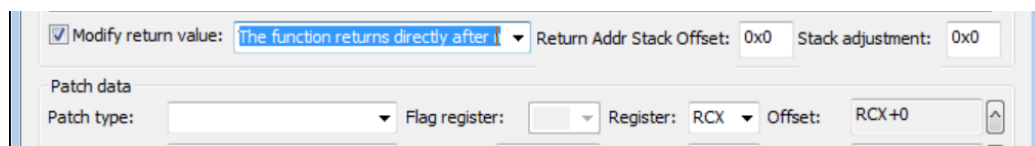
2. Modifying the context when a function returns

When using this scheme, you also need to set the "return address stack offset". Instead of executing the patch data when the patch is interrupted, we set it to get the function return address from the stack after the interrupt and set a breakpoint, and only after the interrupt at the return of the function do we start reading the register context and

executing the patch data.

3. Modifying the context after a function returns directly

Some functions we do not need to execute, just need to ensure that the function return value can be, then you can choose this program, 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. This operation can be set at any instruction within the function, stack adjustment, please refer to the top of the stack address when the function returns normally.



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

Any instruction that is in a function (basically) has a function return. API functions are not the only ones that can set returns; patch addresses support modifying function returns.

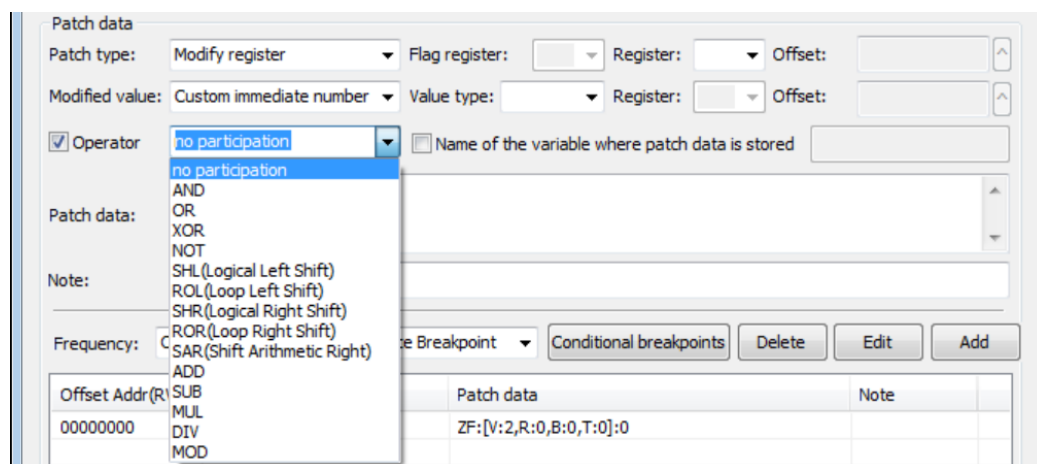
4. Arbitrary stack address return before patching

Modify the return value of the function, not only in this function to modify the return value, just before the use of

modification can be, so the stack of some function return address can be utilized, the patch tool does not care about the function when to return, only care about the return address obtained from the stack.

(xiii) Arithmetic operations on data

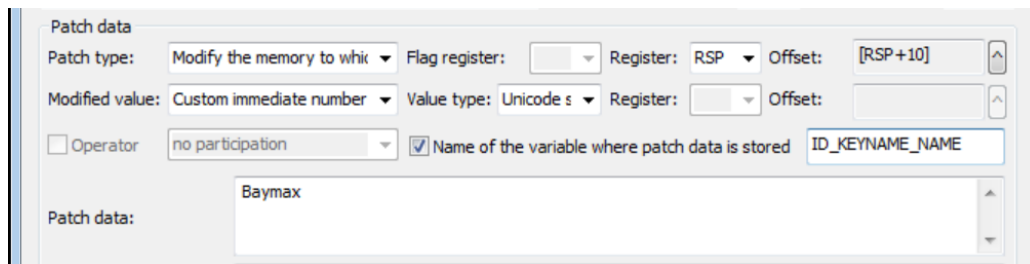
In some cases, we need to operate on the data, when the modified value is "Custom Immediate Number", you can check the "Data Operation" option, and the content entered in the Patch Data field will be the second value involved in the operation. If you select "Inverse operation (NOT)", the input data will be ignored.



(xiv) Reading patch information from INI files

When "Patch data from text" is checked, the cracking module can read user-defined data through the configuration

file to complete the patch operation.



Patch data

Patch type: Modify the memory to whik Flag register: Register: RSP Offset: [RSP+10]

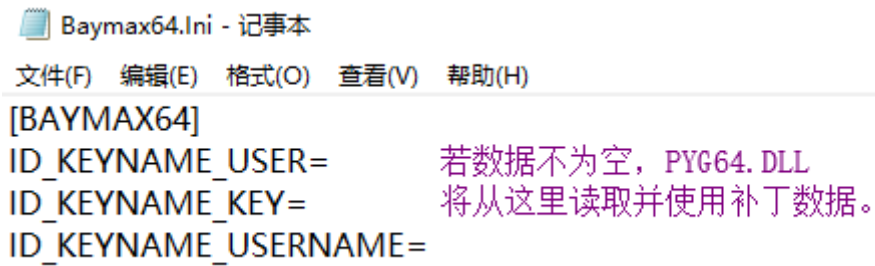
Modified value: Custom immediate number Value type: Unicode s Register: Offset:

☐ Operator: no participation ☒ Name of the variable where patch data is stored ID_KEYNAME_NAME

Patch data: Baymax

If the patch entry has "Patch data from text" checked, the patch program will not only release the hijacking and cracking module in the target folder, but also release the Baymax.ini or Baymax64.ini file (the format is shown in the figure below), and the patch module will read the data corresponding to the mark when the data corresponding to the mark item is not empty. corresponding data and replace the original patch data for patching. For example, if this option is checked in the patch item of software about information, users can customize the content to be displayed.

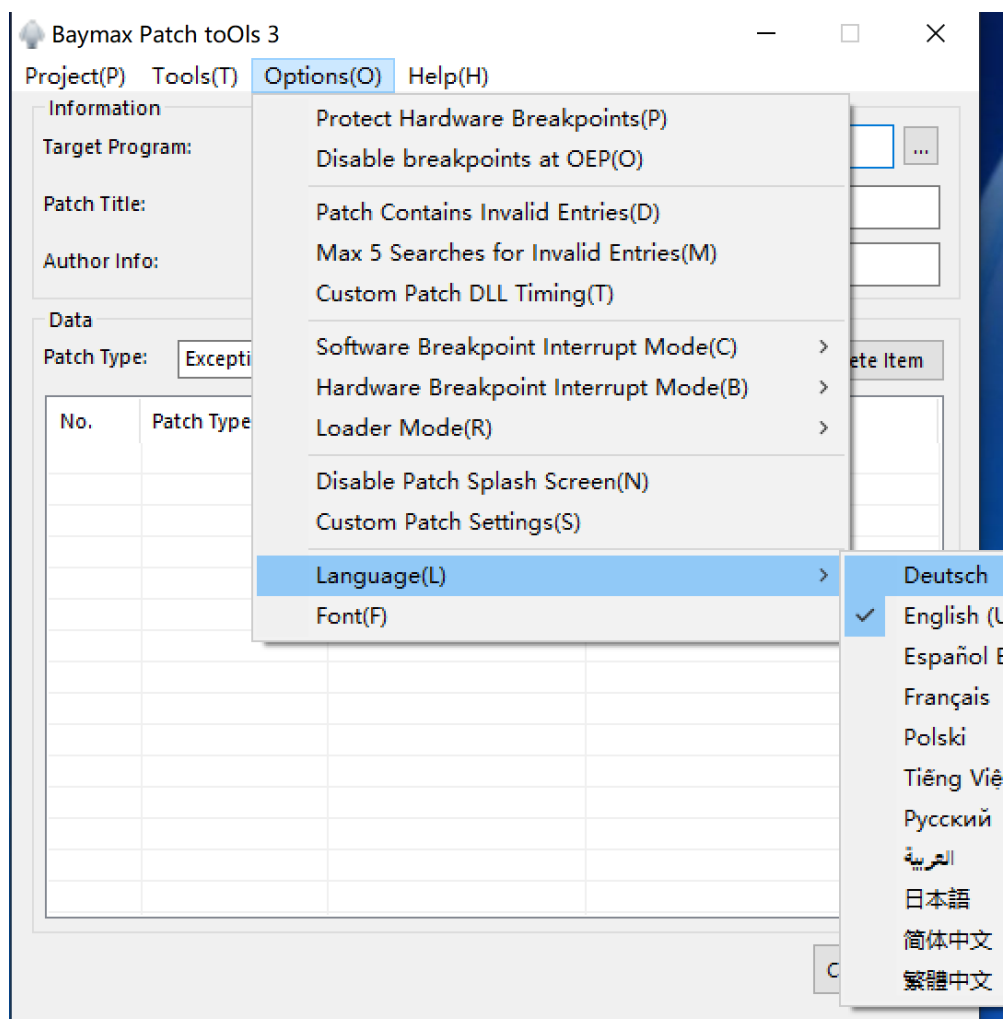
Note: The patch data filled in by the user must be consistent with the value type in the patch entry, for example, if the value type of the patch entry is HEX value, the data entered by the user will be parsed and patched according to the HEX data.



V. Generating Patches and Saving Patch Projects

(i) Description of menu items

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



1. Protect hardware breakpoints: some shells will erase hardware breakpoints, if you analyze the log file and find that all DrX registers are 0, the hardware breakpoints may be cleared, so you can check this option at this time. Currently.
2. OEP does not set interrupt: Baymax's patch will set 0xCC at OEP by default, some shells such as Obsidium will detect the integrity of the code segment, at this time you need to check the option not to set.
3. Patch with invalid entries: In order to improve startup performance, Baymax is internally optimized so that when an invalid entry is encountered, it is considered that the decoding has not been completed and no subsequent patch entries will be executed. If the patch is compatible with old and new versions and contains invalid entries, you need to check this option.
4. Invalid entries are searched for up to 5 times: If there are invalid entries and HOOK is set, the patch will be tried every time the API is executed. After this option is checked, if other patches have been executed, the invalid entries will be marked as ignored after 5 times, and the entries will not be executed again.

5. Customize Patch DLL Timing: If the patch address is located in a DLL module, the cracking module will turn on the function of monitoring module loading by default, and when the module is loaded, it will determine whether it is the target module and execute the patch function. Some large programs load many DLL modules when they start up. Since monitoring module loading will cause certain performance loss and affect the startup speed, users can check this option to cancel the monitoring of module loading and control the timing of DLL patching by themselves to improve the startup speed of the software.
6. Software breakpoint interrupt mode:
 - 1) Emulates single-step mode, the interrupt mode used by INT3 prior to x86 v2.9.5 and x64 v2.5, by parsing and setting NEXT_IP to emulate single-step execution, where a patch entry records and processes a single piece of 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 CC

interrupt is triggered, it jumps to the new space to execute the assembly instruction at the original address through the exception mechanism to realize the multi-threading mechanism, and it is recommended to use this program.

7. Hardware breakpoint interrupt mode:

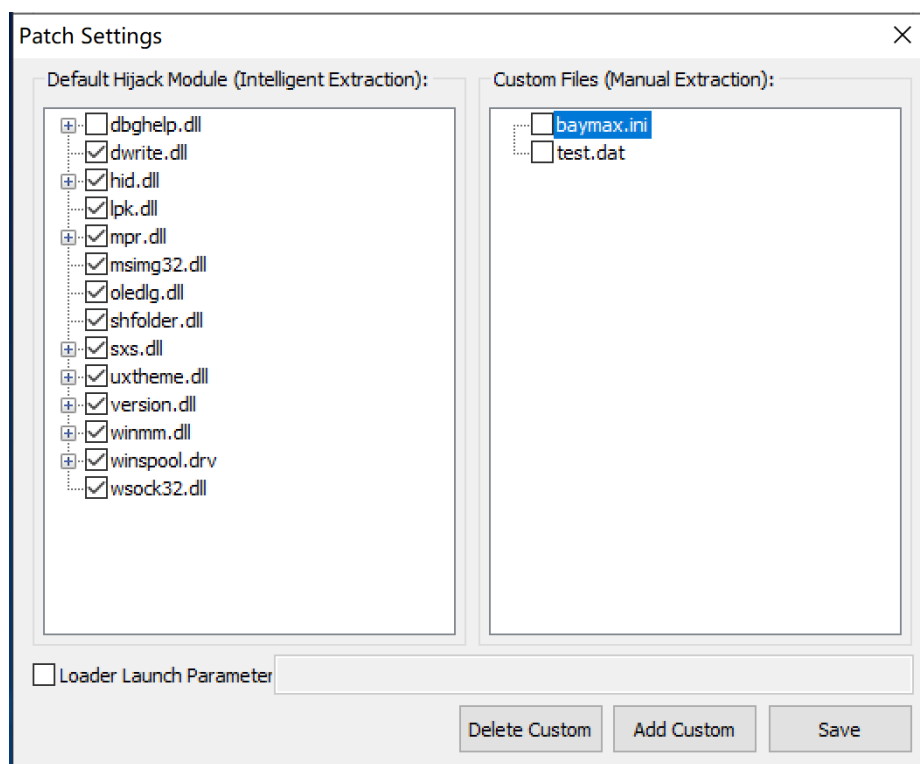
- 1) Default setting of the current thread (the thread in which the patch module is loaded, usually the main thread), the non-main thread can be HOOK through the API called before the execution of the patch address of the thread to realize the thread to set the hardware breakpoints, the threads have their own hardware breakpoint processing mechanism, do not affect each other, it is recommended that you use this program.
- 2) Set all threads by default (set hardware breakpoints on all threads via HOOK) and set cyclic hardware breakpoints on newly created threads via the Hook API. Note: If the new thread will not execute the 4 interrupt 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), injects the patch module into the process via APC before it executes to OEP.
- 2) Self-loading (When Running to OEP), when executing to OEP, load the patch module directly through ShellCode.

9. Patch startup without popup box: Baymax patch startup default display LOGO popup box, in order to facilitate the user to perceive the patch has been loaded properly work, if you do not need to display, you can check this option.

10. Custom Patch Settings



Here you can configure the built-in hijacking DLLs used by the hijacking patch. Some hijacking DLLs include builds for multiple platforms; if selected, they will be bundled together. When deploying the hijacking DLLs, the tool will locate and release the DLL that best matches the current system environment.

Some programs require startup parameters. When generating a loader patch, you can set the loader's startup parameters here; the loader will pass the specified parameters when launching the target program.

11. Switch Language

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

12. Font Settings

Users can switch the desired font and font size for a better experience with this tool.

(ii) Creation of hijacking, injection patches

Baymax can create release and debug patches. The "Create Patch" button in the main interface generates a release hijacking patch, and the menu items allow you to create

debugging and injection patches.

Hijacking patch releases the hijacking module and cracking module to the process folder, and executes the patch by loading the cracking module through the hijacking module. Injecting a patch releases the cracking module to the process folder on startup, and allows the process to realize loading and executing the patch through the APC mechanism.

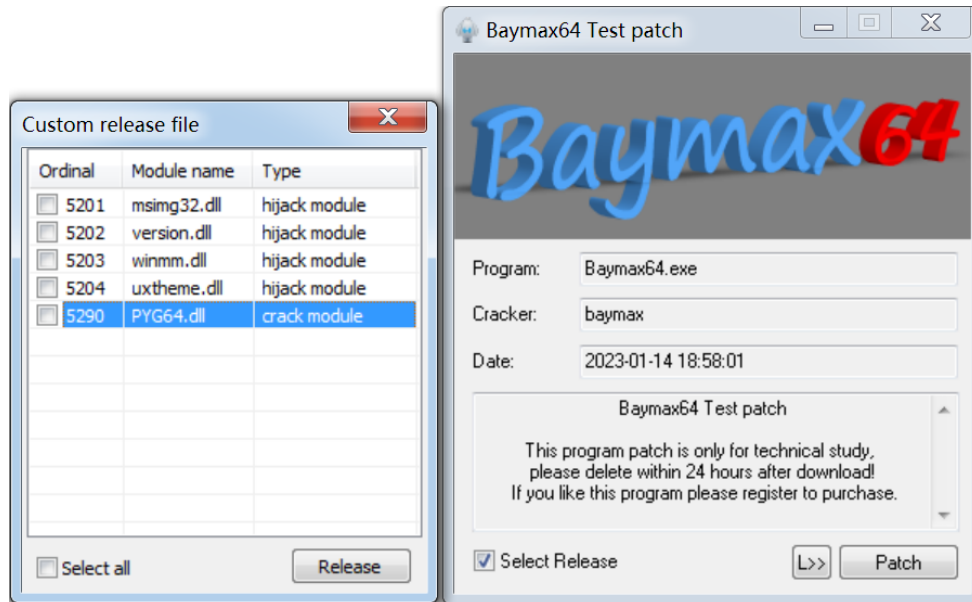
The patch created by Baymax will save the existing language resources into the patch, and when the patch is run, the corresponding language will be selected according to the running system (if the corresponding language is missing, English will be displayed), or the user can select the language in the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

(iii) Creating debug patches

In order to understand the implementation of the patch and troubleshooting, Baymax supports the generation of a debug version of the patch. The difference between the debug version of the patch and the official version of the patch is that the user data of the debug version of the patch is not encrypted, and a .log file is generated to record the

implementation of all the entries of the patch.

(iv) Hijacking and cracking modules



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

Not find loadable hijacking module there are two cases: may be the hijacking module within the patch can not be automatically loaded, at this time you can make other

loadable hijacking module, or in other load module to increase the export function of the crack module to load, or to generate the injection of patches; may also be the software is not directly loaded at startup, some large-scale programs in the startup process may load the hijacking module, we recommend that you manually release all the hijacking module, run the program by process explorer and other tools to see if the process is loaded, or by determining whether the program starts to show Baymax specific LOGO module. At this time, it is recommended to manually release all the hijacked modules, run the program through process explorer and other tools to see if the process is loaded with the hijacked module, or by determining whether the program startup is to display Baymax's unique logo pop-up box to determine whether the crack module is normally loaded.

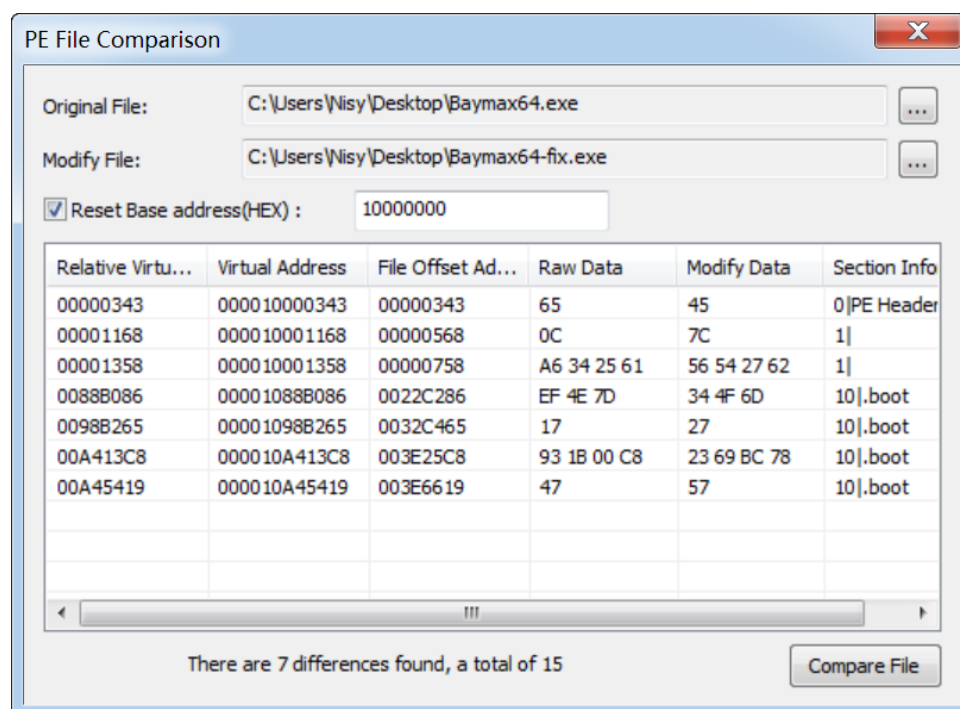
(v) Saving and opening patch projects

Baymax can save patch project as *.bpt project file, support open bpt file to create patch. If you do not understand the bpt data format, please do not edit it manually. In addition, the bpt file contains version information, so when a new version of a feature 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.

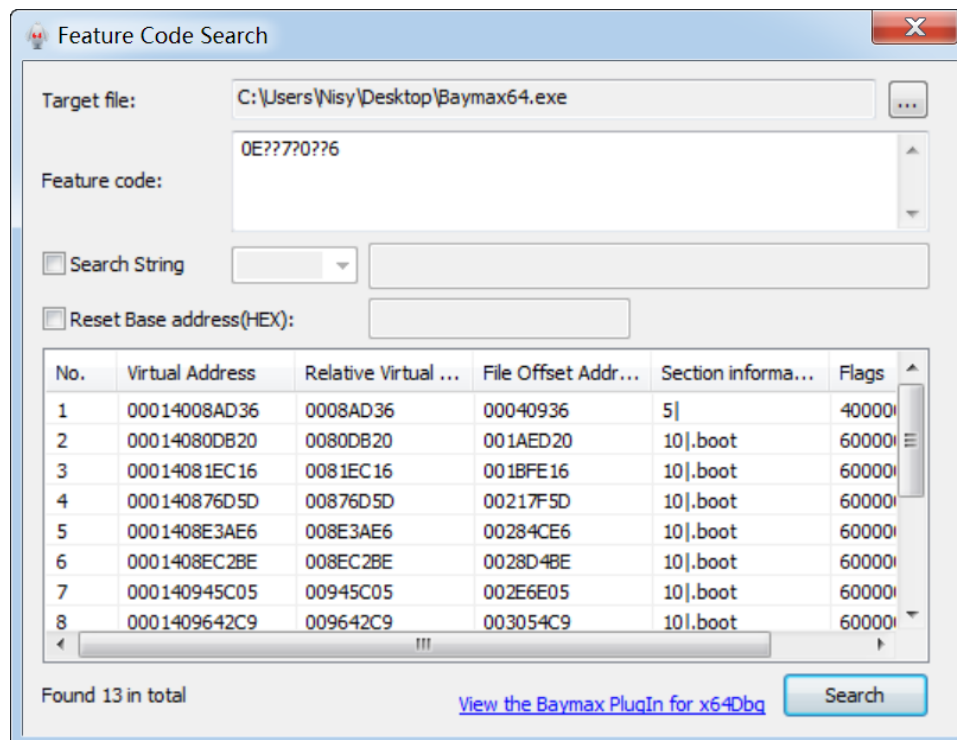
VI. Plug-in Introduction

(i) PE file binary comparison



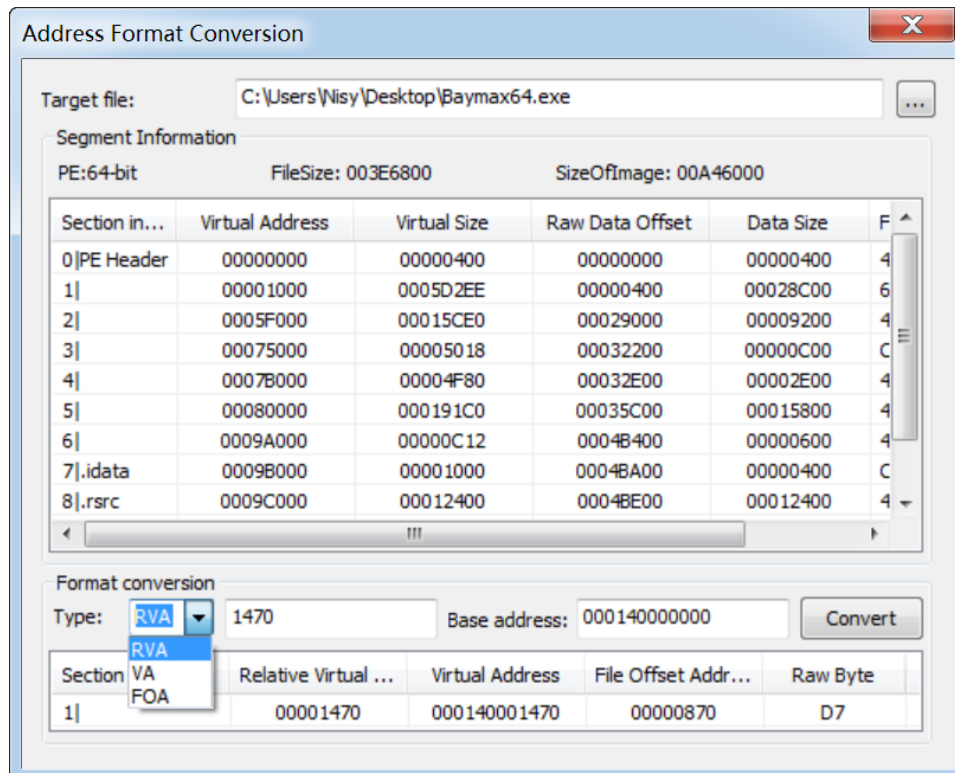
Support PE file binary comparison, the files involved in the comparison should be of the same size. If this function is called out by the "Compare Files" button in the offset patch interface, you can click "Add Checked" to add the checked patch data at once.

(ii) Feature code search tools



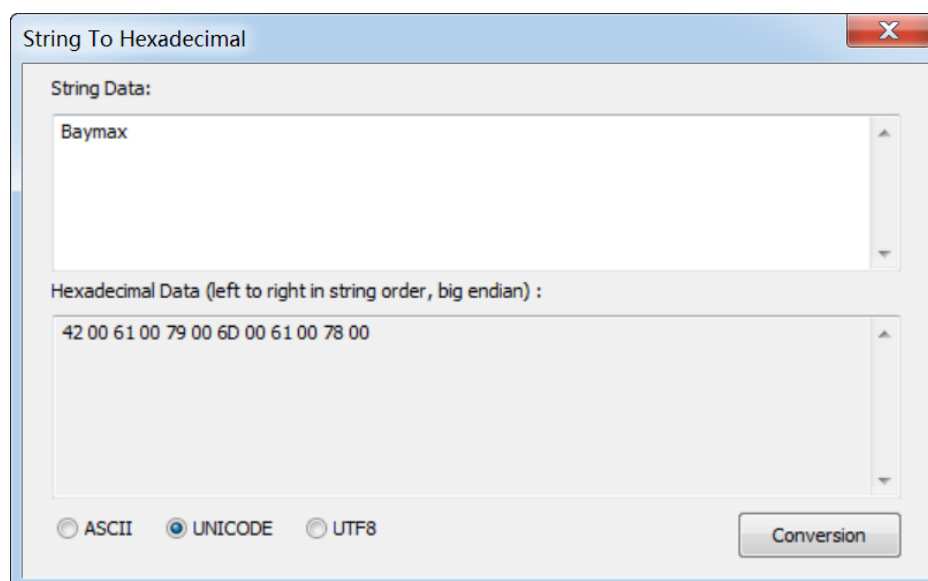
Feature code search tool. Currently, it only supports searching PE files, resetting the base address, and listing the VA, RVA, and FOA corresponding to the feature code.

(iii) Address format conversion tools



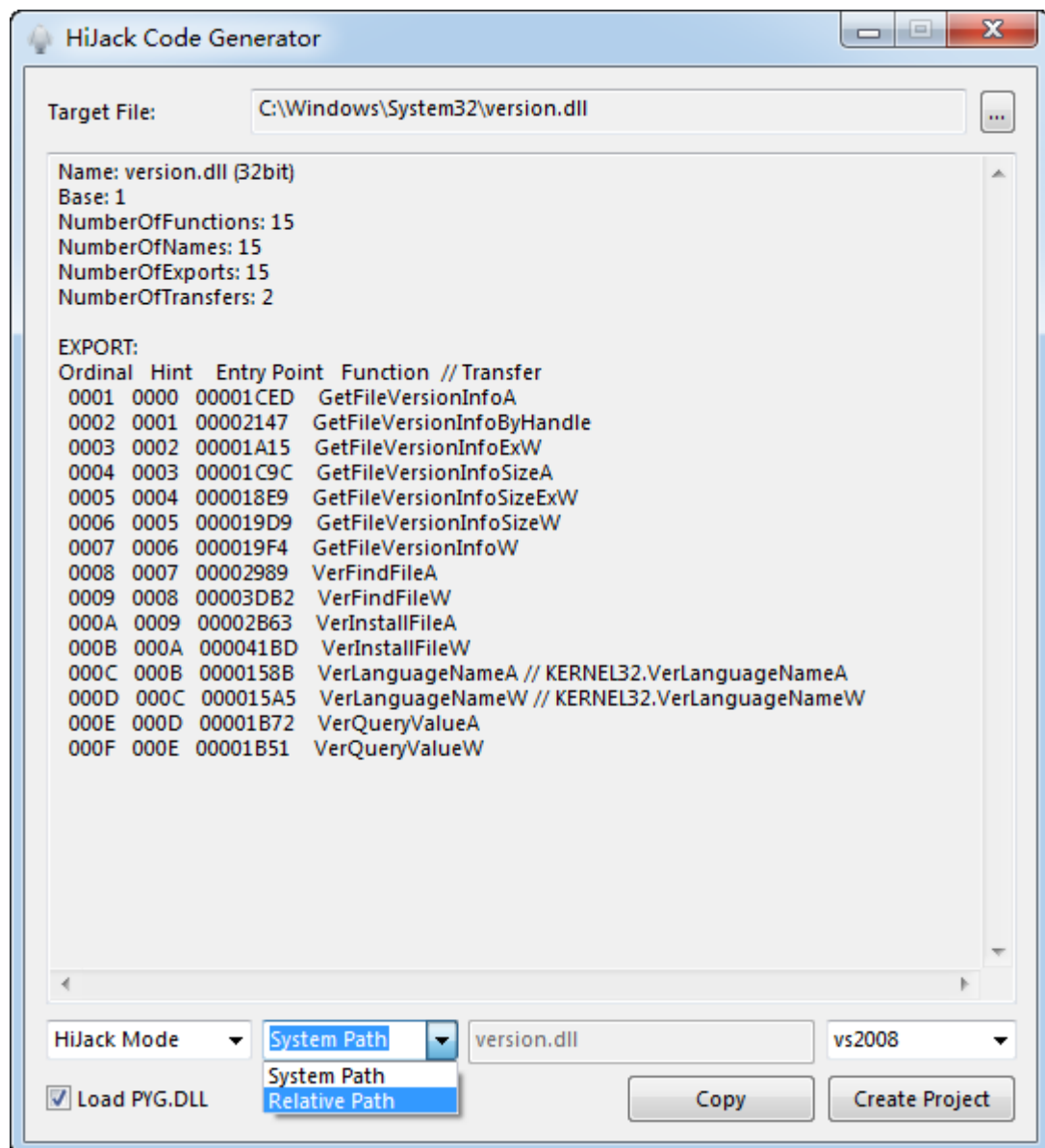
Parses PE segment information and supports conversion between RVA, VA, and FOA formats.

(iv) String to hexadecimal tools



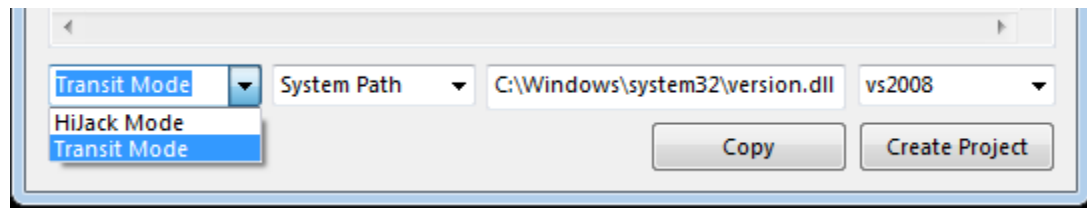
We fill in the patch data, sometimes need to convert the string to HEX values, so provide this small tool to facilitate the format conversion, HEX data according to the memory low address to high address output.

(v) Hijacking code generators



Drag and drop modules (dlls) that can be used for hijacking into the window to generate a hijacking project corresponding

to the VS version.



The tool supports hijacking mode and transit mode

Tools support some C++ export functions (namespaces, classes, virtual tables, static members, etc.)

Tools can parse and generate exported data (not functions)

.....

1. vs2008 x64 compilation instructions:

When compiling x64 modules, vs2008 needs to set the assembly compiler. Please change the asm compiler in \Microsoft Visual Studio 9.0\VC\VCProjectDefaults\masm.rules to ml64.exe and save it as masm64.rules file:

Original data:

```
CommandLine="ml.exe /c [AllOptions] [AdditionalOptions]  
/Ta[inputs]"
```

Modified to:

```
CommandLine="ml64.exe /c [AllOptions] [AdditionalOptions]  
/Ta[inputs]"
```

2. vs2008 or above

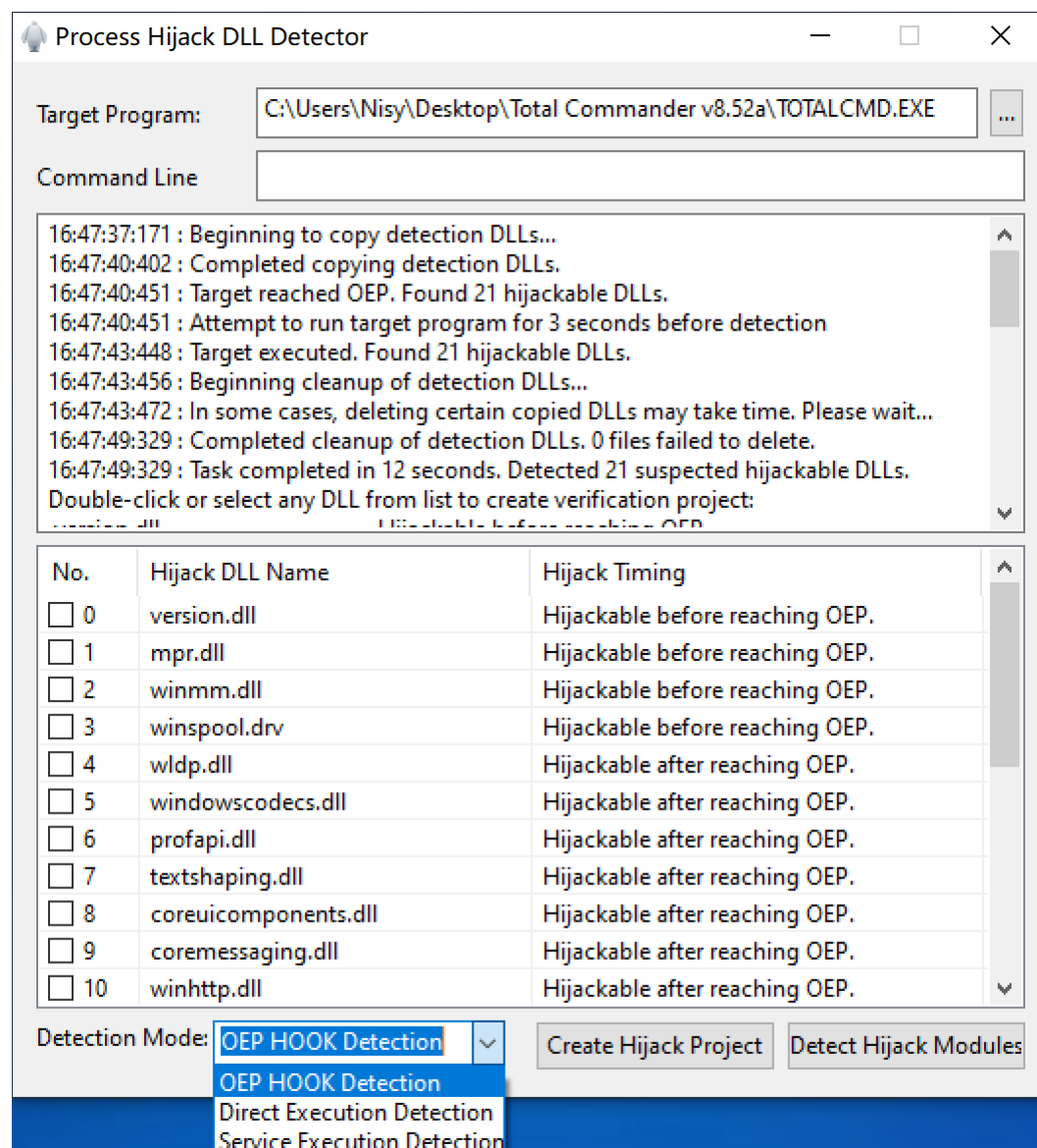
The project uses the default SDK version, if you can't

compile it, set it to the installed SDK version.

XP is unchecked by default for "Tools Platform Set", please set it by yourself if necessary.

vs2019 support XP setting: change "C/C++ - Language - Conformal mode" to No to pass compilation.

(vi) Process Hijack DLL Detector



The Process Hijack DLL Detector can discover system DLLs that

are hijackable by running the target program during startup.

It supports three modes:

- OEP Hook Scan: Accurately determines which system DLLs can be loaded by checking at the process' s Original Entry Point (OEP).
- Direct Run Scan: Detects which DLLs are loaded but cannot determine whether they load before or after the OEP.
- Service Run Scan: Designed for service processes; likewise cannot determine whether the target DLL loads before or after the OEP.

VII. Cases and tips for use

(i) Setting hardware breakpoints on other threads via HOOK implementation

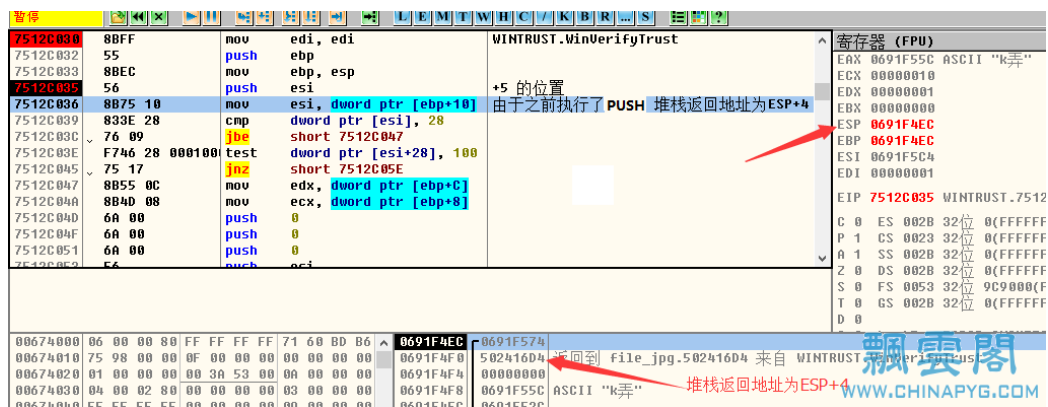
Hardware breakpoints have thread relevance, the cracking module executes the "breakpoint setting function" to set hard break only for the current thread, usually for the main thread, when other threads reach the patch address, the patch operation will not be executed because the hardware is not set. When other threads execute to the patch address, they will not execute the patch operation because the hardware is

not set. Baymax currently does not provide the option to set hard breaks for all threads, the workaround is to HOOK API, every time you execute to the API function that is HOOKed, you will call the "Breakpoint Setting Function" again and set hard breaks for the current thread. hard break for the current thread.

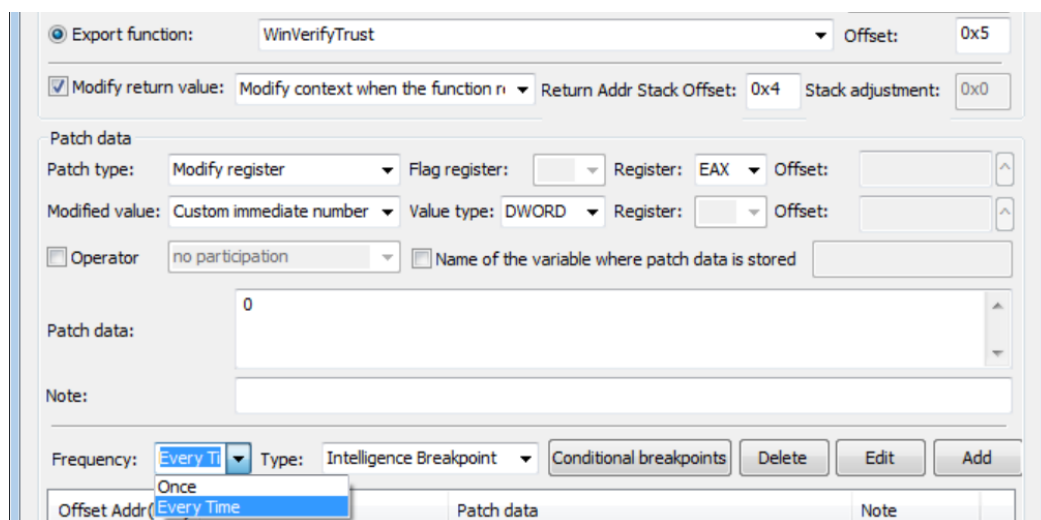
(ii) Fixed parameters or return values through the function HOOK

The "Modify Function Return Value" function provided by Baymax has some limitations, if we use INT3 program, there will be problems when multiple threads call at the same time; if we use hardware breakpoints, we can't set breakpoints for all threads by default. Can we rely on the existing function to fix the return value of a function when it is frequently called by multiple threads? The answer is yes, we export the 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 address +5 of the function.



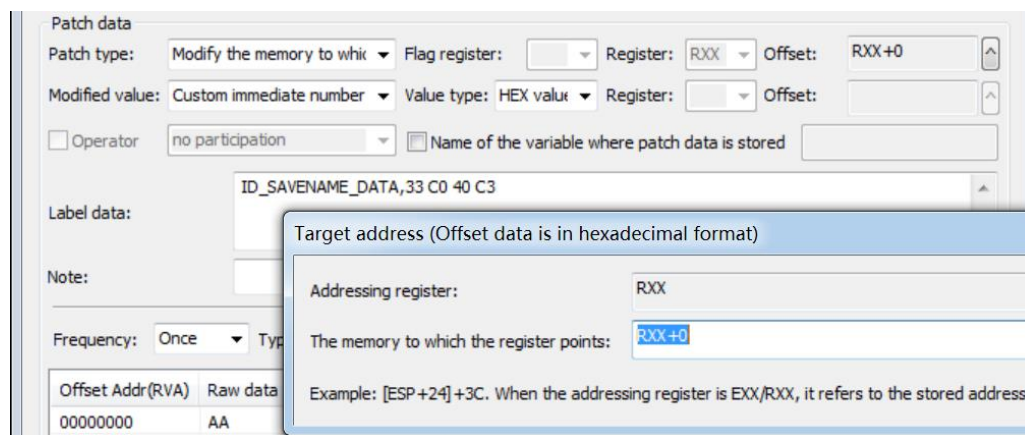
The reason for setting offset +5 is that Baymax HOOK library will modify the function header to JMP XXXXXXXXXXXX instruction, and after the execution of the internal process and then jump back to the original function to be covered by the data of the next instruction, the function is covered by the instruction is exactly 5 bytes, if the instruction is covered by more than 5 bytes, choose to cover the data of the next instruction offset value can be. Check the box "Modify Return Value", the return address of the function at offset +5 is located in [ESP+4], so we set the "Return Address Stack Offset" to 0x4.



Set up as above, so that indirectly realize the HOOK function to complete the call thread to set up a hard break, so as to achieve the function of fixed function return value.

(iii) Patch heap space by storing data

Some programs will map the key code into memory (heap space) to execute, we can save the address after the program applies for memory space, wait for the process to finish filling the heap space with data, and set a breakpoint in the heap space at any address before the code is executed, and use the storage address + offset to locate the key address.



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

Bayamx supports secondary addressing of our saved addresses, supporting addressing operations such as $[EXX+N]+N1$, so that we can easily modify the values of the

members of a class object after we save it. For example, if we save an address of 10000000, and we need to modify the address of 10000008, we can set the offset to EXX+8 (x64 bit RXX+8).

(iv) Modifying global variable values in instructions

Many programs store the registration status in a global variable in the data segment and determine its status with the cmp instruction:

007A677E	880C10	mov byte ptr ds:[eax+edx],cl
007A6781	833D 08E88100 00	cmp dword ptr ds:[81E8D8],0
007A6788	75 56	jne xshell.7A67E0
007A678A	B9 90000000	mov ecx,90
007A678E	66C9 00	imul ecx,ecx,0

If "007A6781 | cmp dword ptr ds:[0081E8D8],0" instruction, we need to modify the value of 0081E8D8 to 1, how to do it, Baymax provides the mechanism, the method is to store first, then modify.

Breakpoint address

☒ Virtual Addr(VA): 007A6781 Module Address: 00760000 First byte: 0x0

☐ Feature code data: Offset: 0x0

☐ Export function: Offset: 0x0

☐ Modify return value: Save the current context and modif Return Addr Stack Offset: Stack adjustment:

Patch data

Patch type: Stores the address in the i Flag register: Register: Offset:

Storage value: Saves the memory address: Value type: DWORD Register: Offset:

☐ Operator no participation ☐ Name of the variable where patch data is stored

Storage label: ID_SAVENAME_GLOBAL_DATA

Step 1: Store first.

Set the interrupt address and then set it in the patch

data as follows:

Select "Store Current Data" for Patch Type;

Stored value is "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.

The screenshot shows a debugger's configuration window with the following sections:

- Breakpoint address:** ☒ Virtual Addr(VA): 007A6781, Module Address: 00760000, First byte: 0, Offset: 0x0. There are also options for Feature code data and Export function, both with an Offset of 0x0.
- ☐ Modify return value: Save the current context and modify, Return Addr Stack Offset: , Stack adjustment: .
- Patch data:**
 - Patch type: Modify the memory to which, Flag register: , Register: EAX, Offset: EAX+0.
 - Modified value: Custom immediate number, Value type: DWORD, Register: , Offset: .
 - ☐ Operator: no participation, ☐ Name of the variable where patch data is stored: .
- Label data:** ID_SAVENAME_GLOBAL_DATA,1
- Note:**

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

Select the patch type "Modify memory pointed to by the number of banks".

Optional "customization" of patch data.

Value type "DWORD"; mark data format is "mark name, patch value", here we should fill in the mark name entered before and modify the value "ID_SAVENAME_GLOBAL_DATA,1".

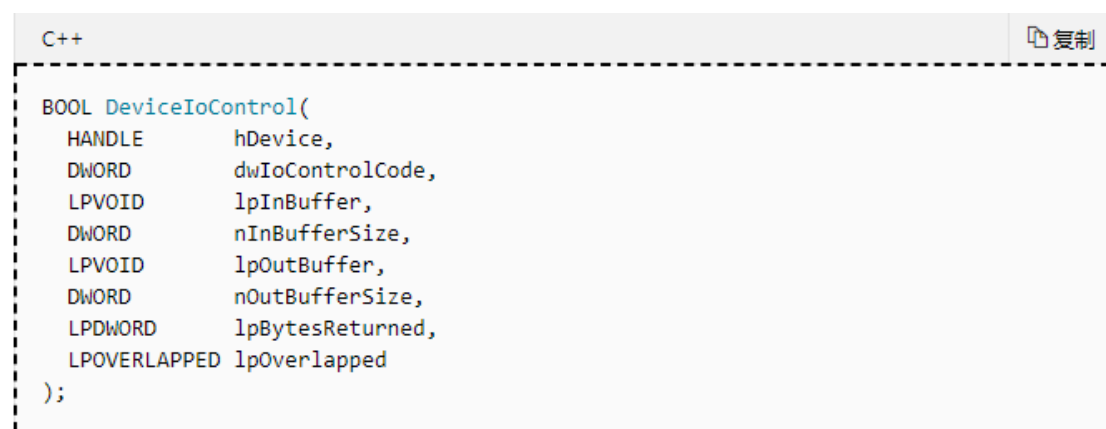
GLOBAL_DATA,1".

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

(E) Fixing hard disk information by 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 the lpOutBuffer after the function returns contains hard disk serial number information.

A screenshot of a C++ code editor window. The title bar shows "C++" and a "复制" (Copy) button. The code is enclosed in a dashed rectangular box. The function signature is as follows:

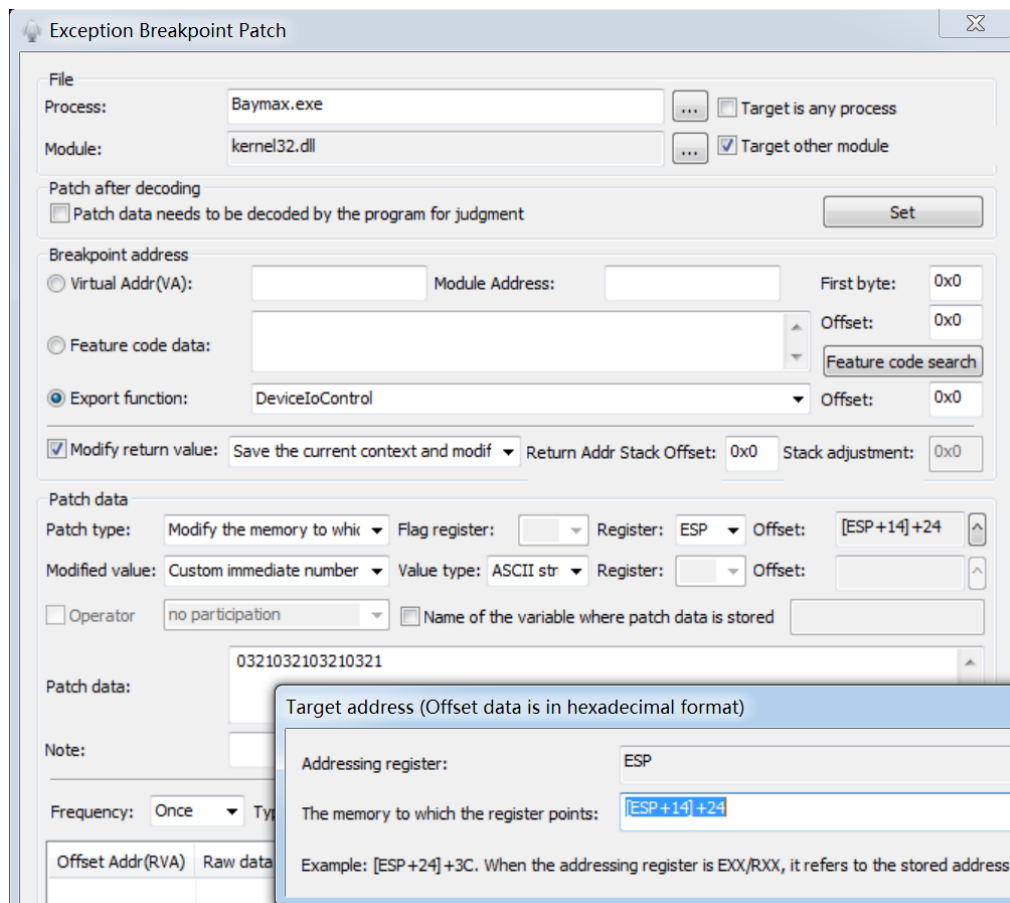
```
BOOL DeviceIoControl(  
    HANDLE      hDevice,  
    DWORD       dwIoControlCode,  
    LPVOID      lpInBuffer,  
    DWORD       nInBufferSize,  
    LPVOID      lpOutBuffer,  
    DWORD       nOutBufferSize,  
    LPDWORD     lpBytesReturned,  
    LPOVERLAPPED lpOverlapped  
);
```

Let's create an interrupt patch item, check "Target other modules" for the patch module, select the system module "\Windows\system32\kernel32.dll", and select the patch

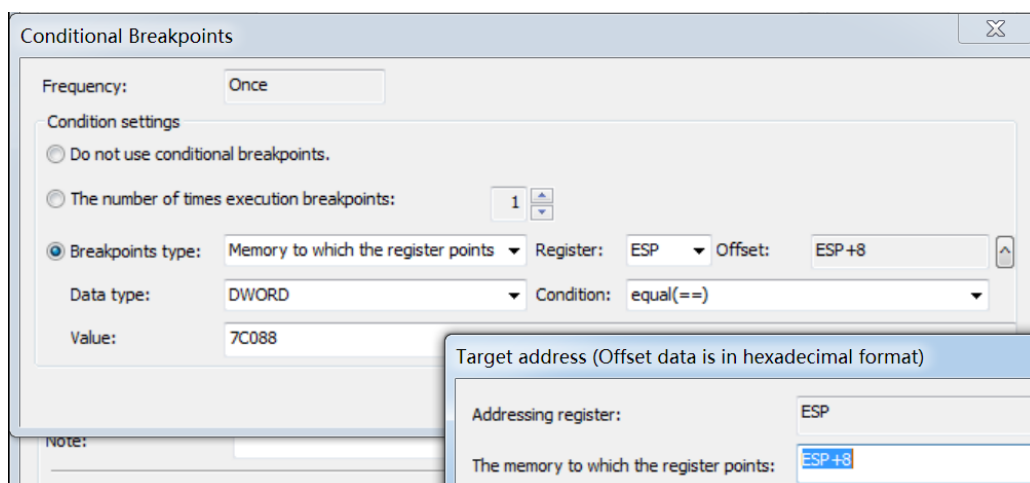
address from the exported functions. "DeviceIoControl" function. Check the box "Modify return value" and select "Save current context and modify on return" mode, the debugger in the function address after the interruption of the stack is shown in the figure below:

```
$ ==> > 0092DD88 GetPcInf.0092DD88
$+4 > 000000E4
$+8 > 0007C088
$+C > 0025E9D8
$+10 > 00000021
$+14 > 0025E7C0 // 参数 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 parameters are stored in [ESP+14], and the sequence number is located at offset 0x24 from lpOutBuffer. 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, and the condition is $[ESP+8]==7C088$. Note that the "Patch Memory Address" field is always the data address, i.e., $ESP+8$. The settings are as follows:



After creating the patch and running it we found that

the hard disk serial number obtained by the program had been changed to our custom ASCII string. We fixed the hard disk serial number without modifying any bytes of the program.

VIII. Exchange of feedback

(i) How to analyze the cause of patch failure

1. Please close the debugger first

The crack module has an anti-debugging mechanism, if the crack module is not loaded or you don't see the LOGO popup box when you start it, please **close the debugger** and test it again. If you use the debugger to analyze the crack module dynamically, it may lead to abnormal patch function.

2. Analyze the LOG for debugging patches

If the patch does not achieve the expected results after loading, please select Generate Debug Hijack Patch or Debug Inject Patch from the menu, the program will create a xxx_baymax.log file under the process folder (if the program is located on the system disk, you need administrator privileges to run it before you can create a .log file), 32-bit processes can also open the DbgView utility to receive debugging information from the output of

the patch. to view. We can analyze the execution of the patch through the output log to find the cause of failure. If there is no way to start, you can also add to the tool group feedback exchange, feedback can be attached to the log file, log contains patch entries and patch address, you can modify the key data and then provide, does not affect the analysis of the problem.

```
1 2019-07-25 22:34:39:931: [2128][BAYMAX64]: PYG.DLL ver: 3.0.1.1025 模块加载 ↓
2 2019-07-25 22:34:39:931: [2128][BAYMAX64]: Process Attach: C:\User: [REDACTED]\Desktop\Debug\ [REDACTED] 示DiskT
3 2019-07-25 22:34:40:117: [2128][BAYMAX64]: Not Find Baymax IniFile ↓
4 2019-07-25 22:34:40:118: [2128][BAYMAX64]: Proc DiskTest.exe Module DiskTest.exe Name DiskTest.exe ↓
5 2019-07-25 22:34:40:118: [2128][BAYMAX64]: 补丁需要进行HOOK处理: DiskTest.exe ↓
6 2019-07-25 22:34:40:119: [2128][BAYMAX64]: 增加 HOOK 2 方案 user32.dll ==> CreateWindowExW ↓
7 2019-07-25 22:34:40:119: [2128][BAYMAX64]: 初始化完成 ... ↓
8 2019-07-25 22:34:40:119: [2128][BAYMAX64]: Add Hook: Type 2 user32.dll ==> CreateWindowExW ↓
9 2019-07-25 22:34:40:119: [2128][BAYMAX64]: End StartHook() ↓
10 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。
11 2019-07-25 22:34:40:252: [2128][BAYMAX64]: Proc DiskTest.exe Module DiskTest.exe Name DiskTest.exe ↓
12 2019-07-25 22:34:40:252: [2128][BAYMAX64]: HOOK函数, 执行补丁数据。 ↓
13 2019-07-25 22:34:40:253: [2128][BAYMAX64]: 设置断点补丁条目 ↓
14 2019-07-25 22:34:40:253: [2128][BAYMAX64]: 非 NS_TYPE_SETTRVABREAK 类型 0 ↓
15 2019-07-25 22:34:40:253: [2128][BAYMAX64]: 断点补丁地址 C [REDACTED] 补丁数据 RSP, [REDACTED] V:2,R:1,B:102,T:1,C:C
16 2019-07-25 22:34:40:253: [2128][BAYMAX64]: 设置断点 ↓
17 2019-07-25 22:34:40:254: [2128][BAYMAX64]: 解析异常断点数据成功 ThreadId: 2496 ↓
18 2019-07-25 22:34:40:254: [2128][BAYMAX64]: 设置INT3断点 ↓
```

3. Hijacking module loaded too late

In some processes, the hijacker module is loaded too late, so the patch address has already been executed when the cracking module is loaded, which makes 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 cracking module is loaded when the patch address is interrupted.

(ii) Join the official communication and feedback group

Baymax official Q group: 112823588

(iii) Tool downloads

<https://sourceforge.net/projects/baymax-patch-tools/files/>

<https://forum.exetools.com/showthread.php?t=20426>