SaferPatches

Thomas Richter

COLLABORATORS			
	TITLE : SaferPatches		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Thomas Richter	July 8, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

SaferPatches

1.1 SaferPatches Guide

SaferPatches Guide

Guide Version 1.00

SaferPatches Version 2.10 ShowPatch Version 2.20

© THOR-Software

Thomas Richter

Rühmkorffstraße 10A

12209 Berlin

Germany

EMail: thor@math.tu-berlin.de

WWW: http://www.math.tu-berlin.de/~thor/thor/index.html

1.2 The THOR-Software Licence

The THOR-Software Licence (v2, 24th June 1998)

This License applies to the computer programs known as "SaferPatches", "ShowPatch" and the "SaferPatches.guide". The "Program", below, refers to such program. The "Archive" refers to the package of distribution, as prepared by the author of the Program, Thomas Richter. Each licensee is addressed as "you".

The Program and the data in the archive are freely distributable under the restrictions stated below, but are also Copyright (c) Thomas Richter.

Distribution of the Program, the Archive and the data in the Archive by a commercial organization without written permission from the author to any third party is prohibited if any payment is made in connection with such distribution, whether directly (as in payment for a copy of the Program) or indirectly (as in payment for some service related to the Program, or payment for some product or service that includes a copy of the Program "without charge"; these are only examples, and not an exhaustive enumeration of prohibited activities).

However, the following methods of distribution involving payment shall not in and of themselves be a violation of this restriction:

(i) Posting the Program on a public access information storage and retrieval service for which a fee is received for retrieving information (such as an on-line service), provided that the fee is not content-dependent (i.e., the fee would be the same for retrieving the same volume of information consisting of random data).

(ii) Distributing the Program on a CD-ROM, provided that

a) the Archive is reproduced entirely and verbatim on such CD-ROM, including especially this licence agreement;

b) the CD-ROM is made available to the public for a nominal fee only,

c) a copy of the CD is made available to the author for free except for shipment costs, and

d) provided further that all information on such CD-ROM is redistributable for non-commercial purposes without charge.

Redistribution of a modified version of the Archive, the Program or the contents of the Archive is prohibited in any way, by any organization, regardless whether commercial or non-commercial. Everything must be kept together, in original and unmodified form.

Limitations.

THE PROGRAM IS PROVIDED TO YOU "AS IS", WITHOUT WARRANTY. THERE IS NO WARRANTY FOR THE PROGRAM, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IF YOU DO NOT ACCEPT THIS LICENCE, YOU MUST DELETE THE PROGRAM, THE ARCHIVE AND ALL DATA OF THIS ARCHIVE FROM YOUR STORAGE SYSTEM. YOU ACCEPT THIS LICENCE BY USING OR REDISTRIBUTING THE PROGRAM.

Thomas Richter

1.3 About SaferPatches

The purpose of SaferPatches is simple:

Make the SetFunction (= patch function) of the OS safer, allow installation and removal of patches in any order. Record the patches made and the program that made the patch. Generate Alerts ("Guru meditations") for offending patches that break the Os rules.

Especially the last part makes SaferPatches a debugging tool, intentionally. Patches that do not like SaferPatches will most likely break in a future Os release because they rely on side effects that haven't been documented. Furthermore, SaferPatches keeps a list of all installed patches, to give you an overview what's all in your system. It is adviced to run SaferPatches permanently.

1.4 Installing SaferPatches

Copy the "SaferPatches" program from the archive to the C: directory. Then, open the "S:Startup-Sequence" file with an ASCII editor of your choice. In worst case, you've to use the system editor "Ed". Open a shell, then enter:

Ed S:Startup_Sequence

Then locate the line where "SetPatch" is run. This line should be one of the first lines in the script. Move the cursor to the start of next line below, and press Return. This should insert a line between "SetPatch" and the next command. Move the cursor upwards one line, and enter there

SaferPatches INSTALL >NIL:

This will run SaferPatches the next time you boot the system.

The program "ShowPatch" should be copied to wherever you want to keep it. In case the "FD" "function description" files are somewhere installed on your HD, set the ToolType "FDDIR" to the directory where these files are kept. For example,

FDDIR=SYS:V40/FD

will tell "ShowPatch" to scan the directory "V40/FD" on your boot partition to obtain the Os function names. Otherwise, ShowPatch will not be able to present the names of the patched functions.

This guide can be kept wherever you want to store it.

1.5 Configuring SaferPatches and ShowPatch

SaferPatches takes the following arguments on the shell command line:

SaferPatches INSTALL REMOVE REMEMBER WARN

INSTALL: This is the default, it installs SaferPatches in the system.

REMOVE: Removes SaferPatches from the system if it is installed. You are, however, "recommanded" not to remove it once it has been started.

REMEMBER: Tells SaferPatches to keep the names of the tasks that installed the patches. This option is required by ShowPatch to be able to show these names - hence to give you a hint about the origin of the patch.

Furthermore, ShowPatch requires this information for the "Disable Group" and "Enable Group" operation.

WARN: Makes SaferPatches even more picky. It will generate a guru 0x10000026 if a removed system vector is still called.

ShowPatch is configured by the "ToolTypes" of the workbench icon, or by giving arguments on the command line. The following ToolTypes resp. Arguments are known:

ShowPatch FDDIR,GUI/S, LEFTEDGE=LE/N,TOPEDGE=TE/N,WIDTH/N,HEIGHT/N, SCREEN

FDDIR: Specifies the directory where the "FD" files are kept. These files define the names of the Os functions and are parsed by ShowPatch to be able to present names and not only offsets. Except for better readability, these files are not required.

GUI/S: This is a shell-only argument, there's no ToolType like this. In case this switch is given on the command line, ShowPatch will present its graphical user interface. Without this switch, the patch list is only printed to the shell window, thru the standard output. This is the default if ShowPatch is run from the workbench.

The next arguments are only scanned if ShowPatch is run from the workbench, or with the GUI argument:

LEFTEDGE/N: The X coordinate of the user interface window.

TOPEDGE/N: The Y coordinate of the window. If this is "-1", the window will be aligned to the title bar of the screen it is opened upon.

WIDTH/N: The width of the window, in pixels.

HEIGHT/N: The height of the window, again in pixels.

SCREEN: The name of a public screen where ShowPatch should open its GUI. If this argument is not given, ShowPatch will open its window on the default public screen, which is again by default the workbench.

1.6 The ShowPatch utility

ShowPatch is a tiny workbench/shell tool that presents a list of all patches currently installed in your system. To run it, either type it as command in the shell, or double-click its workbench icon. More about its parameters and the ToolTypes is in the configuration chapter.

The main part of the output window is made up by the patch list requester. Just the same list is printed on the shell if you invoke this program without the "GUI" argument . Here's the meaning of the fields in this list, from left to right:

o) The library or device name that has been patched. The suffix ".library", ".device" or ".class" has been stripped to leave more room for other entries.

o) The function offset of the patched function, in hex notation. These offsets are always negative because the function entries are at lower addresses than the library or device base.

o) The state of the patch. Each patch can be in one of three different states:

Active: This means that the patch is currently installed in the system and running.

Removed: The program that installed the patch removed it already. SaferPatches keeps only a small "wedge" that survives the patch. The patch routine itself is no longer called, but the wedge remains installed to make sure that code doesn't run into the desert. SaferPatches will remove these wedges as soon as the library gets flushed from memory, i.e. by "avail flush". Removed patches cannot be re-activated by means of ShowPatch.

Disabled: The patch has been temporarely disabled by the "ShowPatch" utility and may be re-enabled any time. Programs won't call patches in this state, much like "Removed", but unlike them, can be re-activated.

o) The function entry point of the patch, given as hex address.

o) The function name, if it is available. ShowPatch will scan the directory specified by the FDDIR ToolType or shell argument to find this out. This slot is left blank if no matching ".fd" file is found.

o) The name of the task or program that installed the patch. This is only available of SaferPatches was run with the REMEMBER keyword, it won't keep this information otherwise. The name presented here is the name of the task, or the process, or if the installed program was run from CLI, the name of the CLI command.

Special care has to be taken for the tasks "ramlib" or "« IPrefs »". Both indicate patches that have been installed as part of the system, or as part of a library, on startup. It is unfortunately not possible to re-construct which library or which system component was responsible for these patches; you should leave these alone, unless you know what you're doing.

In the ShowPatch GUI, you may pick entries from this list, entries on which the four gadgets below will act. These entries appear then "highlighted".

Disable Group This gadget is only available if "SaferPatches" has been run with the REMEMBER option, hence if patch names are available. This gadget will remove complete patch groups, by having given one or more patches from that group. To be specific, if the program "TrueMultiAssigns" installed four patches, and you selected one of them, "Disable Group" will disable all four patches of this program at once.

A "patch group" is a set of patches which have been installed by the same program, identified by the name of this program.

Remember however that disabling patches is always a dangerous operation. The only program that possibly may be able to remove a patch most safely is the program that installed the patch, and not "ShowPatch". This is because it is not clear which system resources are required and have been altered by the patch. "Disable Group" is, if it is available, the safer alternative, it is recommended to use this option instead of "Disable Item" if you have to remove patches manually at all.

IMPORTANT: Patches installed by "ramlib" or "« IPrefs »" should never be removed.

Enable Group: Re-enables patch groups by picking at least one item of the group.

Disable Item: Disables only the patches picked from the patch list, not more, not less.

BE WARNED! This is a dangerous operation! Removing patches without knowing what these patches are good for, who installed them, for what purpose and which side-effects are caused by these patches is a very delicate operation and may easely lead to crashes. Avoid this operation whenever possible, unless you're really, really sure what you're doing.

Enable Item: Re-enable the selected patches from the patch list.

The two gadgets below are relatively easy to explain:

Update re-scans the patch list and updates the output.

Quit quits the program.

1.7 Compatibility notes

SaferPatches is intentionally incompatible to patches that somehow interpret the result code of SetFunction() - the Os patch installation procedure - to derive details of the patched procedure. Note that doing so won't work anyhow if another patch is already installed, or if an Os release unknown to the patch is used. These patches must be avoided, they will lead to "certain suprise factors" in your system anyhow.

One considerably (in)famous example is the "CEDPatch". Better buy an update of Ced instead of relying on these stunts.

Details about this behaivour are in the technical section.

Another example is "SetPatch", the Os bug fix. However, SetPatch requires this operation to fix known bugs of the Os, so this behaiviour should be tolerated. Therefore, "SaferPatches" must be run later than "SetPatch".

IMPORTANT: SaferPatches MUST be run after SetPatch.

VirusZ might report that libraries got patched from time to time. This is because SaferPatches modifies the Expunge vector of all libraries in order to free the memory occupied by the patches if the library gets closed. This warning message is harmless.

SaferPatches IS compatible to the debuging tool PatchWork.

1.8 Software failures generated by SaferPatches

Because "SaferPatches" is supposed to be a debug utility, it may create "guru meditations", i.e. software alerts. They are all recoverable, so they won't hurt you unnecessary, but in case you see them, you'd better remove the program that causes the trouble.

The following alerts may be thrown by SaferPatches:

0x01000020 Patch vector out of range.

A program tried to patch a vector in a library that isn't present.

0x01000021 Entry is not patchable.

A program tried to install a patch into a vector that can't be patched. Some vectors, like the GetCC() vector of the exec lib are directly inlined functions that can't be changed with SaferPatches. Same goes for the DOS library of releases 32 to 35.

0x01000022 No mem for patch vector.

SaferPatches failed to allocate memory for the patch.

0x01000023 Patched vector to ROM.

A program attempted to patch one ROM vector to a second vector which is actually in the ROM. Replacing one system function by a different one is considered to be illegal.

0x01000024 Found libentry without patches.

SaferPatches found a library in its patch list that isn't actually patched. This is an internal failure and shouldn't occur.

0x01000025 Unsecure patch.

A program tried to patch one vector that can't be patched in a safe way because it is called by the SetFunction procedure itself. One example is "SetFunction", or "SumLibrary".

0x01000026 Removed libentry called.

A libvector that has been removed was illegaly called again. A program restored an older patch, but was still calling the obsolete return vector. This can only happen with the WARN option enabled.

0x01000027 Function pointer is NULL.

Somebody tried to patch a function to NULL.

1.9 System friendly patches, and other internals

Here are a couple rules for patching the system as friendly as possible:

By SaferPatches, three operations are considered to be "legal" patching operations:

1) Install a patch

2) Remove a patch

3) Query whether a given patch is installed.

A legal way how to install a patch: (with or without SaferPatches)

-Call Forbid() or even Disable() in critical cases. Disable() is required for Os functions that are interrupt-callable. Signal() or PutMsg() are examples for these functions.

-Load the address of the new function in d0, the offset in a0, the library and a1, and ExecBase in a6.

-Call SetFunction()

-Store the return value of SetFunction as pointer to the old function. Keep this to call the Os function from within the patch, or to remove the patch later. DO NOT use this for anything else.

-Call Permit() or Enable().

The following methods of installing patches ARE CONSIDERED TO BE ILLEGAL:

-Forgetting to Permit()/Forbid() or to Disable()/Enable().

-Reading the vector directly from the library vectors instead of using the return value.

-Using the return code of SetFunction for anything else except jumping into the old code. Especially, DO NOT interpret the as routine and interpret it as direct pointer to the last installed patch. Do not try to "automagically disassemble" the patched function to draw conclusions about internals of the patched function. This kind of magic will break as soon as another patch is already installed.

-Patching one OS function to a different one.

-Patching functions that can't be patched, like inline functions r functions called by SaferPatches. This includes the functions cheClearU(), SetFunction() and SumLibrary().

-Using anything else but the return code of SetFunction for calling the old function.

-Writing the new function directly into the vector offset without using SetFunction()

A legal way to remove a patch: (with SaferPatches)

Even though the procedures presented below remove a patch vector safely from the Os (ONLY the vector, NOT the code) IT DOES NOT PREVENT, WITH SAFERPATCHES INSTALLED OR NOT, THAT SOME TASK IS STILL EXECUTING THE CODE OF YOUR PATCH. This is, you may not freely deallocate the memory your patch resides in, at least not immediately. Removing patches is always a problem, it is not recommended to release the memory allocated for the patch code at all UNLESS YOU CAN REALLY, ABSOLUTELY ENSURE THAT NO TASK IS EXECUTING YOUR CODE. There is, however, no general guide line how to do this. Sometimes, it is enough to wait long enough, much longer than the execution time of the patched function. However, this is no longer a good method in case the patched function may Wait(), for example for a user interaction.

The following algorithm "works" IF YOU KNOW the patch is installed and SaferPatches is installed. It DOES NOT work around general patch caveats:

-Load d0 with the return code of SetFunction() you got when installing the patch. Load a1 with the base address of the library. Load a6 with execbase.

-Call SetFunction().

This call will always succeed, and it will always remove your patch.

A legal and safe way to remove a patch: (with or without SaferPatches)

The following mechanism is recommended for general patch removal. Please note THAT THIS MIGHT FAIL and you won't be able to remove your patch at all. It WON'T fail with SaferPatches, though. It does, too, not work around the code memory caveats from above.

-Call Forbid() or Disable().

-Call SetFunction() with the same arguments AS IF you are installing he patch again, this is:

- Load the address of the new function in d0, the library base in a1 and ExecBase in a6 and offset in a0. Call SetFunction().

-Check the return code: If this is the entry point of your patch, then:

-Load d0 with the return code of SetFunction when installing the patch. Load a1 with the base address of the library. Load a6 with execbase.

-Call SetFunction().

This will remove your patch.

If this is not the address of your patch, then:

-Load d0 with THIS return code of SetFunction, i.e. what you got from the call a few lines on top. Load a1 with the base address of the library. Load a6 with execbase.

-Call SetFunction().

This will re-install the last patch.

YOUR PATCH CAN'T BE REMOVED SAFELY IN THIS CASE (SaferPatches is not running).

-Call Permit() or Enable().

Another protocol, which is also supported by SaferPatches and by Exec:

-Call Forbid() or Disable().

-Call SetFunction() with the arguments AS IF you are removing the patch.

-Load the address of the result code you got from SetFunction() when installing the patch in d0, the library base in a1 and ExecBase in a6 and offset in a0. Call SetFunction().

-Check the return code: If this is the entry point of your patch, then your patch was removed successfully.

If this is not the address of your patch, then:

-Load d0 with THIS return code of SetFunction, i.e. what you got from the call a few lines on top. Load a1 with the base address of the library. Load a6 with execbase.

-Call SetFunction().

This will re-install the last patch.

YOUR PATCH CAN'T BE REMOVED SAFELY IN THIS CASE (SaferPatches is not running).

-Call Permit() or Enable().

Which of the two protocols is used doesn't matter. SaferPatches understands both. The first has the advantage that it won't remove the patch, even temporarely, if it can't be removed.

The following methods of removing a patch ARE ILLEGAL:

-Use anything else but the return code of the installing SetFunction for removing the patch again.

-Removing the new function by writing the old return code directly into the vector offset.

-Executing the steps above not completely in Forbid() mode, i.e. call Permit() in between or calling any other Os function that might break the forbid.

(Side remark: SetMan has an ObtainSemaphore here, which might break the Forbid(). This is a design bug.)

A legal way how to verify whether your patch is still installed:

This algorithm can be used to check if your patch is still active, using SaferPatches. It will only tell you if the patch is "frontmost" if Safer- Patches is not installed.

-Call Forbid() or Disable()

-Call SetFunction with the same arguments AS IF you are installing the patch again, this is:

Load the address of the new function in d0,the library base in a1 and ExecBase in a6 and offset in a0. Call SetFunction.

-Check the return code. If it is the same pointer you passed in d0 to SetFunction, your patch is still active. It has been installed now, by the procedure above and must/can be removed if desired/ required with a second call to SaferPatches. It will NOT be installed twice by this algorithm, neither with nor without SaferPatches. (It will with SetMan. This is a SetMan problem.)

-Call Permit() or Enable().

This method is used by the RTPatch program and also considered to be legal.

The following verify methods are considered to be illegal:

-Failing to call Forbid() or Disable()

-Reading the vector entry from the vector base directly.

How to check if SaferPatches is installed:

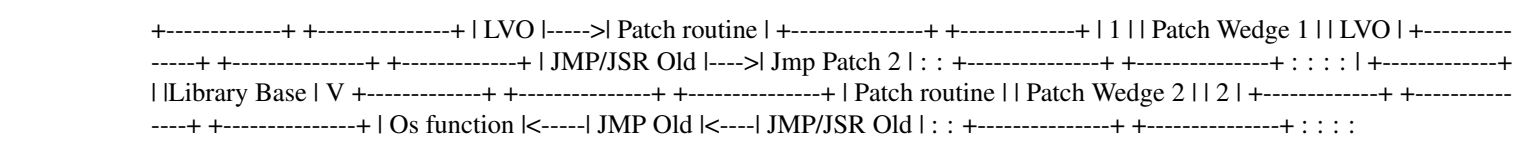
-Call FindPort("SetMan"). If the result is non-zero, SaferPatches, SetMan or PatchControl is installed. Do not use this port for anything else.

-Call FindPort("SaferPatches.rendezvous"). If the result is non- zero, it's specifically SaferPatches. There's usually no need to to check for this port. Don't use this port for anything else!

Some background material how SaferPatches works - do not depent on these internals!

SaferPatches builds a small "wedge procedure" for each patch installed. This wedge is installed between the patch and the old library function, this wedge is what is returned by SetFunction(). Hence, the procedures returned by SaferPatches are NEVER the Os functions themselves. The library vector will then be patched to the patch function itself.

Hence, a library vector with two patches installed looks like this:



Unlike some other "SaferPatches" like tools, "SaferPatches" keeps the wedge behind the patch.

In case patch number 2 gets removed, SaferPatches adjusts the wedge #1 to point to the Os function, hence patch #2 will call now the Os directly instead over patch #2, but the wedge #1 will remain in the system, to ensure that a task still running in patch #2 will find its way in the Os.

1.10 History of SaferPatches

New in 2.06:

Added a new GURU and fixed a bug of the Expunge replacement. Thanks to Magnus for reporting!

New in 2.07:

Added another GURU, added the WARN option to turn off the 0x01000026 guru. Patched the close function for completeness (forgot again about the delayed expunge function, sigh.). Updates the SHOWPATCH program. Added a SetMan port of completeness.

New in 2.07.1:

Updated the ReadMe a bit.

New in 2.08:

Hopefully fixed a minor compatibility problem with PoolMem and MuGuardian- Angel.

New in 2.09:

SaferPatches is now also removing the CloseLib() patch of a patched library on a library flush. This doesn't make any difference at all because the library is then going to be removed from memory anyways. Added a GUI to "ShowPatch" because this was often requested.

New in 2.10:

Updated ShowPatch *a lot*. It is now possible to enable or disable selected entries or groups of patches by one click. However, note THAT THIS OPERATION IS POTENTIALLY DANGEROUS!

Updated SaferPatches a tiny little bit to support the protocol by ShowPatch for enabling and disabling patches.

1.11 Index