Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Obtaining forensic value from the cbWndExtra structures as used by Windows Common Controls, specifically for the Editbox control

Adam Bridge

University of Portsmouth, School of Computing, Buckingham Building, Lion Terrace, Portsmouth, PO1 3HE, United Kingdom

A R T I C L E I N F O

Article history:
Received 3 August 2016
Received in revised form
15 January 2017
Accepted 22 February 2017
Available online 24 February 2017

Keywords:
Windows Common Controls
Digital forensics
Microsoft windows
Volatile memory
Memory forensics
cbwndextra
Editbox
wndclassex

A B S T R A C T

The Windows Common Controls is a library which facilitates the construction of GUI controls commonly used by Windows applications. Each control is an extension of the basic 'window' class. The difference in the extension results in one control over another; for example, an Edit control as opposed to a Button control. The basic window class is documented by Microsoft and the generic information about a Window can be extracted, but this is of very limited use. There is no documentation and very little research into how these extensions are laid out in memory. This paper demonstrates how the extension bytes for the Edit control can be parsed leading to identification of previously unobtainable data which reveal information about the state of the control at runtime. Most notably, the undo buffer, that is, text that was previously present in the control can be recovered – an aspect which traditional disk forensics would simply not provide. The paper explains why previous attempts to achieve similar goals have failed, and how the technique could be applied to any control from the Windows Common Controls library.

Introduction

Research into volatile memory has grown in popularity in recent years because it provides access to data that are simply inaccessible without it. In the case of malware analysis, the data may be a malicious binary which is memory resident such as Latentbot (Karim and Regalado, 2015), or in the case of Law Enforcement investigations the data may be values entered manually by a user into an interface such as an email address or a password for decryption. In any case of memory analysis, one of the biggest challenges facing memory analysts is understanding the data structures as they are laid out in memory. Although access to the source code in open-source software may provide valuable insight into the data structures, such insight is likely completely unavailable for closed-source projects.

This paper focuses on the Edit control which is provided as part of the Microsoft Windows Common Controls library. The Common Controls library is a core component of Windows and is closed-source. The Edit control is ubiquitous in Windows and in Windows applications; it is arguably the most valuable of the Common Controls from a forensics point of view as it likely contains free text entered by the user or text of which the user would have been aware. For example, it is the control into which URLs are entered in Internet Explorer, it is the control into which command lines are typed in the Run dialog, it is the control into which searches are conducted in Windows Media Player, and it is the entire body of a Notepad document.

The Notepad example is notable because many attempts have been made to reliably extract the contents of a Notepad document from a memory capture, but with limited success. This paper examines the in-memory data structure of the Edit control, how the data structure came to be understood, and how applying this understanding resulted in being able to extract pertinent information about the status of an Edit control – not only the text contained therein, but also the contents of the undo buffer, the position of the cursor, what text is selected, and whether the control is configured to be a password Edit control. The approach is generalised meaning that it does not apply to an Edit control within a particular application (for example, Notepad), but to any Edit control, in any application running on any recent version of Windows.

The recovery of the text from the undo buffer is particularly notable because it represents the kind of data that can only be recovered when analysing volatile memory – this kind of discarded data would not be saved to disk.

E-mail address: adam.bridge@port.ac.uk.

http://dx.doi.org/10.1016/j.diin.2017.02.007
1742-2876/© 2017 Elsevier Ltd. All rights reserved.

The findings and process which resulted from the early parts of this research were developed into a proof of concept plugin for the Volatility Framework which was a winner in the 2014 Volatility Plugin Contest. As the research progressed, new results were confirmed and the plugin was updated to reflect the new understanding (Bridge, 2016).

Section "Related work" will describe the related work which is concerned with the recovery of Notepad documents from memory captures and will explain why the approaches taken are unreliable and limited. Section "The Window class" will detail the Window class and explain how the extra information needed to fully understand the Edit control fits with the existing documentation provided by Microsoft. Section "Methodology" will describe the tools and methodology used to identify the data structure and how the Volatility Framework was leveraged to provide a simple way of extracting the data. Section "Results" will describe the results of the experiments. Section "Handling unknown class names" will explain how the Edit control structures were proven even when Volatility was not able to identify the Common Control class. Section "Conclusions" details the conclusions, and Section "Future work" describes possible future developments of the work, for example, a generalised approach for all Common Controls.

Related work

There appears to be no documented research concerning the analysis of the Windows Common Controls in memory generally, either in published literature or from less formal sources such as blogs or forums. Similarly, there appears to be no work concerning analysis of any one of the Common Controls, again, either in published literature or in blogs or forums. The likely explanation for this is that researchers in the memory forensics field tend to focus on an application or family of applications rather than an element common to the Windows core, whereas we are interested in developing generic techniques that work with any application.

Dolan-Gavitt (2010) released code which provided a definition of some of the properties of a small number of controls from the Common Controls library for Windows XP 32-bit, but with no supporting documentation as to how he managed to derive the properties.

The closest match to the research described in this paper is analysis of Microsoft Notepad. This is the closest match because the body of Notepad, that is, the area into which the user types is one Edit control. Recovery of the text from a Notepad document may demonstrate tools and techniques that could be generalised and applied to all Edit controls.

Following are examples of published work that show the three different approaches to the recovery of the contents of Notepad instances seen in literature and along with explanations as to why the findings are unreliable or of very limited use.

Firstly, is the attempt to locate the text by identifying the VAD node in which it can be found. Xiao et al. (2014) conducted research using Windows 7 Service Pack 1 and concluded that the VAD node in which the text resides can be reliably identified as it is the node which is the "biggest" and has the 'PAGE_READWRITE' flag set. "[B]iggest" was taken to mean representing the largest number of pages in memory.

The PAGE_READWRITE flag reflects the protection which is assigned to the pages represented by this particular VAD node. As its name implies, the protection flag PAGE_READWRITE identifies that these pages can be both read from and written to. It is correct that the pages of memory containing the text which the user can modify would have this particular protection. However, our experimental results showed Xiao, Xu et al.'s technique to be unreliable: we found multiple examples of Notepad's data in a VAD node other than that described in their work.

In fact, during the research for this paper, several different statistics concerning the VAD nodes were analysed to see if a rule could be devised which would identify the VAD node representing the page or pages containing the Notepad text purely by metadata about the VAD node. However, no such rule could be devised, meaning that analysis of the VADs would not be a reliable way of identifying the text from the control.

Secondly, is the attempt to locate the text by identifying a marker which is a known length before or after the text contained within a Notepad instance. Chen et al. (2012) conducted research using Windows XP Service Pack 3 and concluded that a 12-byte marker could be found immediately before the text, and that the number of characters making up the text could be read from virtual page 0xAA, after a known marker. During the research for this paper, memory samples were taken from Windows XP Service Pack 3, but the conclusions borne out by Chen et al. could not be repeated, that is, the rules they devised could not be reliably followed to discover the text from the Notepad instance. Our experiments found that the text of the control is stored in a heap entry and therefore the bytes immediately preceding the text are in fact just the heap entry header: the header contains data about the heap and therefore will change with the heap meaning it is not a reliable marker.

Finally, is the approach taken by Hale Ligh et al. (2014) which is the most reliable of the three approaches. Hale Ligh et al. discovered that when working with 32-bit versions of Windows XP and Windows 2003, the metadata of heaps could be analysed to drastically reduce the regions of memory in which the text would be found. Hale Ligh et al. identified that the heap containing the text had the 'HEAP_ENTRY_EXTRA_PRESENT' flag set in the _HEAP_ENTRY.Flags structure; therefore, by iterating through the heaps the particular heap containing the text could be located. In testing for this paper, this method did indeed always find the text entered into Notepad, but it also found some false positives.

In summary, there has been no documented research into the reliable recovery of text from the Common Controls generally, or any one specific control. Although Hale Ligh et al.'s approach reliably extracts the text for Windows XP and 2003 32-bit, when false positives are returned, there is no way of knowing which of the returned data is the correct one. Also, the method has not been tested to be generalised to Edit controls in applications other than Notepad.

The Window class

Each of the Windows Common Control classes inherits from the Window class. In modern versions of Windows, the Window class is defined by the WNDCLASSEX structure as can be seen in Fig. 1 (Microsoft, n.d.).

As can be seen, the definition includes generic properties such as the window style, the icon, the cursor and so forth. This structure is well known and various tools exist to parse these values both at runtime in a running system and from memory samples.

In order to enhance the Window class, extra information is stored in the "extra" fields: 'cbClsExtra' and 'cbWndExtra'. As their names imply, cbClsExtra is used to store the "count of bytes" that are "extra" for the class, and cbWndExtra is used to store the "count of bytes" that are "extra" for the Window, that is, a particular instance of the class. It is this extra data, the data stored in the cbClsExtra and cbWndExtra, which differentiates one implementation of a window from another, for example, a List control from an Edit control.

The purpose of this research is to determine whether the data in a particular instance of the Edit class, that is, the data referenced by