Shatter-proofing Windows

Tyler Close, Alan H. Karp, Marc Stiegler
Hewlett-Packard Laboratories
Palo Alto, California

Abstract

The Shatter attack uses the Windows API to subvert processes running with greater privilege than the attack code. The author of the Shatter code has made strong claims about the difficulty of fixing the underlying problem, while Microsoft has, with one exception, claimed that the attack isn't a problem at all. This paper presents a means of defeating this entire family of attacks with minimal breaking of applications and effect on the look and feel of the user interface.

1. Introduction

Shatter [1], so called because it "breaks" Windows, uses the Windows API to send messages to windows associated with processes that have greater authority than the process running the attacker's code. In the example exploits, the target is a system service that has a window on the interactive desktop. The attack uses Windows messages to remove the length restrictions on an input field in the target window and insert code into the service's address space. The final step sends a WM_TIMER message that induces the service to branch to the exploit.

Microsoft initially denied that the attack exploits an architectural flaw in Windows [2], citing three points:

- Privileged services should not have windows on the interactive desktop.
- Shatter requires that the attacker be able to log onto the system.
- No privileges are gained on the domain.

That note also states that "all services within the interactive desktop are peers", which implies that processes with different privilege levels should not be placed on the same desktop. Microsoft also notes that it has long recommended that processes with system privileges not have windows on the visible desktop. A response [4] points out that several services, some supplied by Microsoft, violate this rule.

Microsoft's second and third points appear disingenuous. Logging onto the system is not the most common way for attackers to run code. Viruses, worms, and ActiveX controls on web pages are far easier methods than finding passwords. Also, not exposing the domain to attack will be small comfort to users who have had all their local files corrupted.

Microsoft later released a security bulletin [3] that fixed the WM_TIMER flaw. This message informs the application that a kernel timer event has occurred and tells the application what address to jump to. The flaw was that the application did not check this address. Hence, once the exploit code was installed in the target application's address space, a WM_TIMER message would cause that code to be executed. The fix added a check to see if the address specified in the WM_TIMER message was registered as a call-back before taking the branch. The author of Shatter agreed that this fix largely blocked the attack [4], but claimed that this patch didn't totally solve the problem. A later paper [5] demonstrated that other Windows messages, such as EM_SETWORDBREAKPROC, can also be used in Shatter-like attacks.

The paper reporting Shatter makes several strong claims about the difficulty of making changes to prevent the attack from succeeding. The suggested solutions all break applications or change the behavior of the system. "Basically, there is no simple solution." summarizes the author's opinion.

We have found a feature in the Windows API that defeats Shatter while having minimal impact on application behavior or the user's interaction with the system. First, this paper describes how Windows is structured in Section 2. Next, Section 3 describes several rejected approaches to defeating Shatter. Our proposed solution and the tests showing that it works are presented in Section 4.

2. Windows Structure

Most people are aware that every window appears on a desktop. Fewer people know that there may be multiple desktops. For example, the login window appears on a dedicated desktop. Even fewer know that every desktop is assigned to a window station [8]. Understanding Microsoft's response to Shatter [8] and our approach to defeating it requires knowledge of the interaction among these structures.

The user interface component of Windows consists of a number of parts. One is the *windows station* [7], which is a securable object containing a clipboard, one or more desktop objects, and some other state accessible to objects in the window station. Each logon session is associated with a window station. A desktop is a securable object attached to a windows station that holds UI objects, such as windows, menus, and hooks. Note that windows are not securable objects in the Windows API.

Only one window station, called Winsta0, can interact with the user display, keyboard, and mouse, except on the Terminal Services version of the operating system where each session has such a window station. Only one desktop at a time can interact with the user, and that desktop must necessarily be associated with Winsta0. Every process is associated with a window station, and every thread is associated with a desktop. Threads can move between desktops, and processes can move between window stations, but windows are tied to the window station where they started.

3. Rejected Options

We considered a number of approaches to defeat Shatter. In addition to merely blocking Shatter, we felt that we also had to maintain the system's usability. After all, we'd get no security if nobody used our software. All the options described in this section failed that test.

1. Desktops

Microsoft states that "all services in the interactive desktop effectively have privileges commensurate with the most highly privileged service there" [2]. The implication is that processes with different privilege levels should run on different desktops. So, our first idea was to follow Microsoft's suggestion and run applications with different privileges on different desktops within Winsta0. However, the window station contains the name space of desktops. Although a thread can only enumerate windows on its desktop, it can switch itself to another desktop in its window station if it has the handle to one. It's even possible to guess a desktop name. For example, most systems have a desktop named DEFAULT. Once a thread has a handle to a desktop, it can assign itself to it, circumventing any security benefits.

We tested this attack by creating an alternate desktop, imaginatively named "alternate", and opened three windows on it. We then wrote attack code that did an OpenDesktop, specifying "alternate" for the desktop name and getting back a handle to the desktop. The code then did a SetThreadDesktop, enumerated the windows on that desktop, and sent them SW_MINIMIZE messages. None of the operations used require any privileges. While this attack did no harm and involved no escalation of privilege, it shows that Microsoft's instructions about not putting privileged applications on the "interactive desktop" are incomplete at best.

2. Window Stations

If desktops aren't the answer, perhaps Microsoft was really referring to window stations, not desktops. Unfortunately, using windows stations as the unit of protection instead of the desktop doesn't work for interactive applications running at different privilege levels, as done by Polaris [13]. Since only Winsta0 has access to the display, and windows can't move between window stations, there is no way to interact with such applications running on other window stations.

3. Terminal Server

A given login session has only one window station with access to user interactions, and the standard versions of Windows have only one interactive window station. The Server versions don't have this restriction, though. Hence, we can run each application in its own login session with its own displayable window station.

There are two problems with this approach. The first is cost. A single license for Windows XP Professional sells for $300 on the Microsoft web site. One for Server 2003 carries a $1,000 price tag, with an additional charge for Client Access Licenses (CALs) beyond the first five. It's not clear from the Microsoft description of the CAL whether one is needed for each login session.

The second problem arises primarily in corporate environments. A company's IT staff may spend many hours validating their software environment for desktop machines. Today, that effort is spent on the desktop version of the operating system. Many applications have not been tested on the Server version.

4. Virtual Machines

Virtual machines, such as VMWare [8], provide all the isolation needed to block Shatter. All that's needed is to run every application in a separate virtual machine. Unfortunately, virtual machines are expensive, almost $200 for a copy of VMWare. They also take considerable resources; VMWare specifies a minimum of 128 MB RAM for each running instance. It's clear that a machine with a standard configuration won't be able to run very many instances.

5. Virtual OSes

Defeating Shatter doesn't require the full emulation of the hardware done by virtual machines. Virtualizing the operating system, as done by Xen [10] and Virtuozzo [11], should suffice. A virtual OS is light-weight, allowing a machine to run a large number of instances. Unfortunately, OSes in widespread distribution, such as Linux and Microsoft Windows XP, need to be modified in order to run under Xen or Virtuozzo. There is no Windows version of Xen, and Virtuozzo only supports the Terminal Server 2003 version of Windows.

6. Common problem

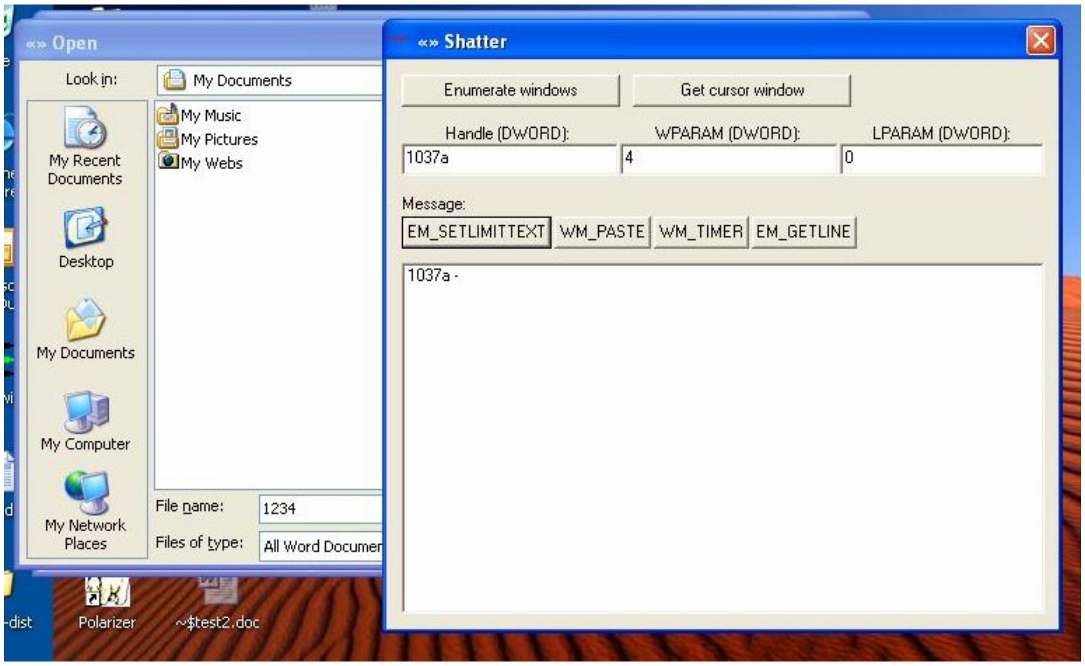
The rejected solutions just described all share a common characteristic; they put windows into separate environments. That means that producing the look and feel of using Windows would be hard. The problem is manageable for applications that run in full screen mode. In those cases, we only need to provide something that looks like the user's task bar to allow switching between environments. However, many people prefer to have overlapping windows, but windows that can overlap necessarily can be used to mount Shatter attacks against each other.

An alternative to overlapping the actual windows is to use screen scraping and keyboard stuffing. Say that file explorer is running on the default desktop and a System service is running on another. An interactive window opened by the service won't be visible to the user as long as the default desktop is active. However, we can write code that captures the bitmap of the service's interactive window and display those bits on the default desktop. Note that we have to monitor the window for changes in case it contains something like a progress bar. Windows messages sent to the window containing the bitmap won't reach the service. Keystrokes and mouse events that appear in it can be forwarded to a daemon running on the alternate desktop for forwarding to the actual interactive window. Implementing this scheme is a major effort with significant performance and usability risks.

4. Defeating Shatter

The process-handling part of the Windows API contains a feature that can be used to block Shatter. A Job object [12] is designed to allow control of a group of processes. Once a process has been assigned to a job object, the association cannot be removed, nor can it be changed. Child processes are part of the same job unless a breakaway privilege is granted explicitly.

Various restrictions can be placed on processes running within a job. In particular, we can set the JOB_OBJECT_UILIMIT_HANDLES restriction (UILIMIT for short), which prevents a process in the job from using handles to windows associated with processes outside the job. Figure 1 shows Shatter running without the UILIMIT restriction successfully changing the length field in a dialog box to 4. What you don't get from this figure is the beep heard when trying to type a fifth character, which demonstrates the success of a key step in the attack.



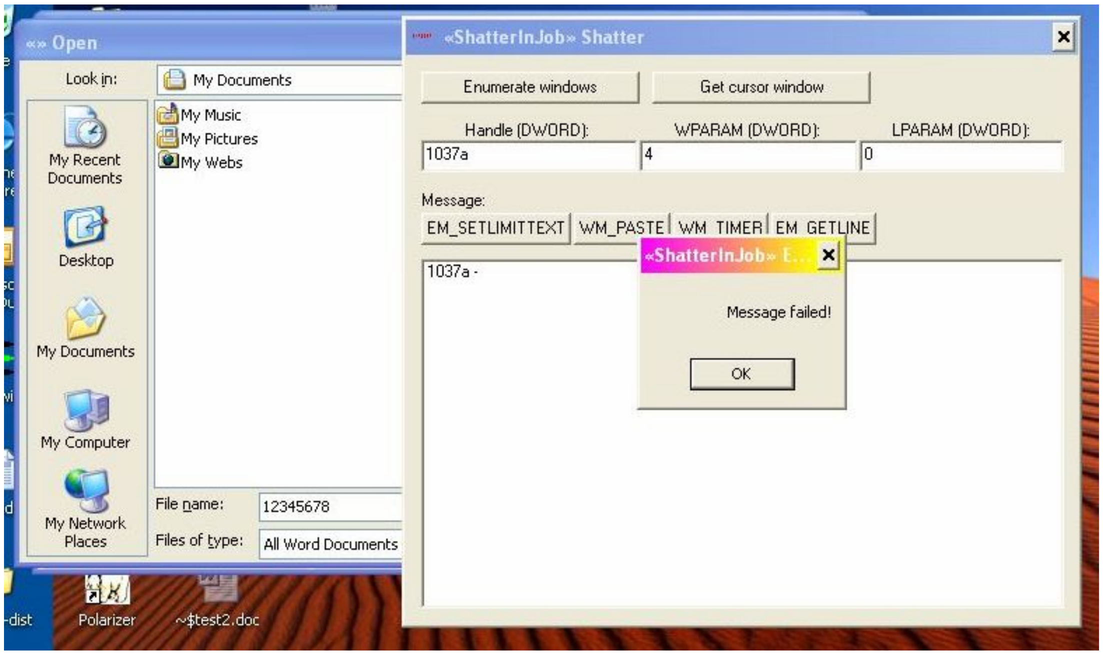
File name: 1234

Handle (DWORD):	WPARAM (DWORD):	LPARAM (DWORD):
1037a	4	0

```
1037a -
```

Figure 1. Shatter changed length field with no UI limit.

Figure 2 shows Shatter running in a job with the UI limit. First, you'll see that the attack succeeds in getting the window handle, which is the same as the one shown in Figure 1. However, this time the window message that changes the length of the input field to 4 fails, as shown by the error message and the typed text.



Look in: My Documents
File name: 12345678
Files of type: All Word Documents

Handle (DWORD): 1037a
WPARAM (DWORD): 4
LPARAM (DWORD): 0

```
1037a -
```

Message failed!

Figure 2. Shatter unable to change length field with UI limit.

We also tried passing a window handle into a job. A process in a job with UILIMIT was unable to use the handle. We even tried sending Windows messages using PostThreadMessage() instead of PostMessage(). These messages were silently dropped by the receiving thread as expected, based on the API and default implementation of the message queue.

We conclude that UILIMIT on job objects defeats all shatter-like attacks. That doesn't mean that there aren't flaws in Windows that can't be exploited using other messaging methods. However, using this restriction prevents the use of Window messaging, the defining characteristic of Shatter.

We have built a version of Polaris [13], a package that configures applications to run in restricted user accounts, to run processes in jobs with the UILIMIT. For the most part, there are no problems. We did find one problem. Although we haven't applied the available clipboard restrictions, processes running with UILIMIT are unable to read text from the clipboard. Since they can read bitmaps, we believe this problem is caused by a bug in the Windows implementation, and we are developing a work-around.

Unfortunately, there is a more serious bug in the Windows XP implementation. If you do a PostMessage() from within a job, specifying HWND_BROADCAST as the target window handle, the Windows message is delivered to all top level windows, both inside the job and outside the job. A test program assigned to a job with UILIMIT that sends WM_CLOSE to HWND_BROADCAST results in all open windows closing. While this denial of service attack is just an annoyance, it means that windows messages are escaping the confines of the job and could be used in Shatter attacks.

This behavior is in direct contradiction to that specified for UILIMIT [9]. The Remarks section for this restriction says:

> "If you specify the JOB_OBJECT_UILIMIT_HANDLES flag, when a process associated with the job broadcasts messages, they are only sent to top-level windows owned by processes associated with the same job."

We reported this behavior to Microsoft. Their response states

> "I have forwarded this information to the product group for further research as a bug. It appears after researching this, that this is not a security vulnerability. If this is not the case and I have overlooked the security implications, please send me details on how an attacker might able to exploit this vulnerability and what the results of an (*sic*) successful exploit might be."

It seems surprising that circumventing a restriction isn't considered a security vulnerability, but this position is consistent with Microsoft's original response to Shatter [2].

5. Conclusions

The Shatter attack is based on the ability of a process to send a Windows message to windows associated with processes running at a higher privilege level. While the WM_TIMER flaw exploited by the original attack has been closed, users are at risk that other such flaws might be discovered. Microsoft's response that the desktop is the unit of protection is at best incomplete. There appear to be ways to break that model.

We have shown that it is possible to defeat Shatter by assigning processes to jobs with UILIMIT that correspond to their privilege levels. Since UILIMIT restricts the use of window handles by those in the job, attacks like Shatter are blocked. Any attack based on the use of Windows messages would be evidence of a bug in the implementation that Microsoft would be compelled to fix.

Programs running in jobs with UILIMIT appear to behave normally, with two exceptions. Drag/drop only works between windows in the same job with UILIMIT. However, processes running with different privileges will most likely run in different logon sessions, and drag/drop doesn't work across sessions. The second difference is clearly due to a bug in the behavior of the clipboard. These jobs cannot paste text, although there is no problem pasting bitmaps. We believe Microsoft will eventually fix this bug. In any case, applications appear to run normally under UILIMIT, contrary to the opinion of the author of Shatter.

References

1. Foon, "Exploiting design flaws in the Win32 API for privilege escalation", http://security.tombom.co.uk/shatter.html

2. Microsoft, “Information About Reported Architecural Flaw in Windows”, http://www.microsoft.com/technet/archive/security/news/htshat.mspx, September 2002
3. Microsoft, “Microsoft Security Bulleting MS02-071: Flaw in Windows WM_TIMER Message Handling Could Enable Privilege Elevation (328310), http://www.microsoft.com/technet/security/bulletin/ms02-071.mspx, December 2002, updated April 2003
4. Foon, “Shatter attacks – more techniques, more detail, more juicy goodness”, http://security.tombon.co.uk/moreshatter.html
5. Lavery, Oliver, “Win32 Message Vulnerabilities Redux: Shatter Attacks Remain a Threat”, iDefense Inc., Reston, VA, http://www.netsys.com/library/papers/Shatter_Redux.pdf, July 2003
6. Brown, Keith, *Programming Windows Security*, Addison-Wesley, Boston, 2000
7. Microsoft, MSDN Library, http://msdn.microsoft.com/library/en-us/dllproc/base/window_stations_and_desktops.asp
8. VMWare, http://www.vmware.com/
9. Microsoft, MSDN Library, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/jobobject_basic_ui_restrictions_str.asp
10. Xen, http://www.cl.cam.ac.uk/Research/SRG/netos/xen/
11. SWSoft, Virtuozzo, http://www.sw-soft.com/virtuozzo
12. Microsoft, http://msdn.microsoft.com/library/en-us/dllproc/base/job_objects.asp
13. Stiegler, M., Karp, A. H., Yee, K.-P., Close, T., and Miller, M, “Polaris: Toward Virus Safe Computing for Windows XP”, HP Labs Tech Report HPL-2004-221, http://www.hpl.hp.com/techreports/2004/HPL-2004-221R1, 2004