Old Dogs and New Tricks: Do You Know Where Your Handles Are?

Jeffrey Walton (jeffrey.walton, softwareintegrity.com)
Brooke Stephens (stephens, umbc.edu)
April 18, 2010

Abstract

In 2009 we examined the effects of manipulating synchronization objects in security software suites frequently found on personal computers running Windows XP and Vista. The synchronization objects were mutexes and events, and the security software included products from AVG, Avast, Avira, BitDefender, BullGuard, CheckPoint, Eset, F-Prot, F-Secure, Kaspersky, McAfee, Microsoft (Security Essentials), Norman, Norton, Panda, PC Tools, Quick Heal, Symantec, and Trend Micro.

The examinations revealed that nearly all suites suffered non-trivial faults originating from both standard and administrator accounts. The faults ranged from simple denial of service affecting the UI console and definition update service to scanner crashes and surreptitious suite shutdown.

1. Introduction

This paper offers incremental research in the area of untrusted program input via synchronization handle manipulations. Exploiting process weaknesses through operating system provided signaling mechanisms was introduced by Michal Zalewski in *Delivering Signals for Fun and Profit*. In the paper, Zalewski focused primarily on deficient code in Unix signal handlers, and not the source of the signal itself. Our work focuses on synchronization objects in the Windows environment.

Named object squatting (the ‘old dog’) is a classical DoS attack and most often associated with lock files and mutexes on Windows platforms. Research indicates one of the earliest vulnerabilities was reported by Arne Vidstrom and corrected by Microsoft in MS01-003, *Weak Permissions on Winsock Mutex Can Allow Service Failure* (*cf* CVE-2001-0006).

While Microsoft does not use Unix-like signals *per se*, the Windows operating system does provide equivalent functionality using primitives such as events and timers. Our ‘new tricks’ include extending squatting attacks onto unnamed mutexes, and attacking both named and unnamed events. CVE’s attributed to the ‘new tricks’ include CVE-2010-0106.

2. Background

Each program executing in the Windows operating system includes a handle table. The handles in the table represent resources used by the program – for example files and registry keys. Efficiency, complexity, and security concerns often require multiple components to coordinate their actions to achieve a single task. The coordination usually takes the form of IPC via synchronization objects such as events and mutexes, which are also represented as handles in the handle table.

Cooperation typically implies at least two components: a user space process which interacts with the user, and a second component which performs the requested actions. In the case of security software, the second component is usually the scanner and is a privileged service or kernel component. Two to three cooperating components proved to be reasonable, but it was not uncommon for a security suite to have 8 to 12 processes executing.

An unprivileged malicious process executing in a logged-on user's context can usually obtain a copy of another process' handles from the same context due to deficient ACLs. If the obtained handles are shared between unprivileged and privileged components, the attacker will obtain a number of handles to objects used by a privileged component. Once handles to key objects have been obtained, an adversary can often manipulate the state of a software suite in undesirable ways. State manipulations can include denying the UI console, denying the update process, terminating an update, denying the scanner, terminating an active scan, terminating the firewall, and shutting down the suite.

Though many assert the attack results are not indicative of vulnerability per se when a user is running as an administrator, we claim the affected software does suffer vulnerability. We base our claim on the fact that (1) users do not expect that their software – especially a high integrity suite – can be surreptitiously manipulated, (2) Howard and LeBlanc warned of the need for well behaved clients when using synchronization objects between clients and privileged services in *Writing Secure Code for Windows Vista*, (3) Windows Vista and above mitigate the vulnerability in support of Digital Rights Management (DRM) via Protected Processes, and (4) once informed, a number of vendors remediated the issues in their software.

3. Handle Duplication

An unprivileged user-space process can obtain a process local handle to both named and unnamed objects of a target process using `NtQuerySystemInformation` (with an information class of `SystemHandleInformation`) and `NtDuplicateObject` or `DuplicateHandle`. Successful duplication generally depends on three factors: the DACL on the target process, the DACL on the synchronization object from the target process, and the user's account.

An adversary enjoys a significant advantage when working directly from the operating system maintained handle table: programs are generally deterministic, so an object handle is usually located at the same position in the handle table across invocations and reboots. This behavior neutralizes some hardening techniques, such as using unnamed objects, private namespaces, and changing the name of a mutex or event across reboots. We take advantage of deterministic creation in our attack on Security Essentials demonstrated in Appendix A.

4. Attack Vectors

We used two attack vectors against the software suites – the first was directed at mutexes, the second targeted events. Both attacks used named and unnamed objects.

Depending on the architecture and design of the suite and the privileges associated with the account, handles which have been duplicated by the adversary may (or may not) bridge the privilege boundary. For ex-

ample, an attacker might duplicate handles from a user-space process which only affects the donor process or other related user-space processes.

By far, one the more interesting situations we encountered is depicted in Figure 1. In the figure below, a malicious user-space process successfully obtained handles to a privileged component through its unprivileged partner.

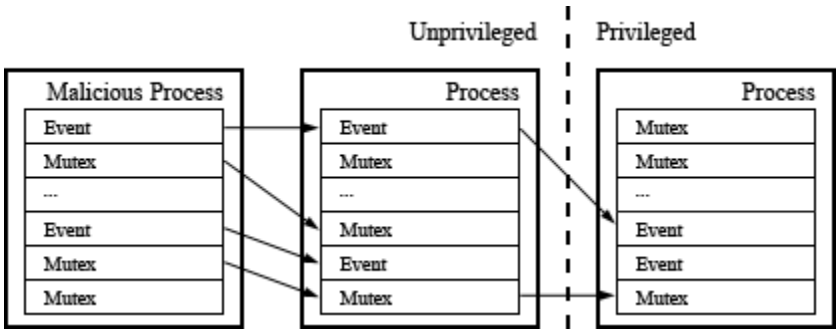

Figure 1: Handles obtained via unprivileged process

When attacking mutexes, we were successful in disrupting dialogs, denying access to resources such as shared memory, and misleading components into erroneously deducing an operation was in progress (since the component could not gain ownership of the mutex in question). Attacks on mutexes generally affected user-space components, though we note kernel components were not immune.

Attacks on events usually allowed us to prematurely terminate operations in progress, close individual components, and shut down the entire suite in a handful of instances. We also observed component crashes and run-away processes (100% CPU) when manipulating events. Both user-space and kernel-space components appeared to suffer equally.

5. Disk Defragmentation Example

To illustrate how the attacks might occur, suppose a set of programs offer a disk defragmentation service. The user selects 'defragment disk' from the non-privileged (user-space) component, which initiates a privileged service to perform the work on the physical disk. The user process then enters an efficient wait state so that it does not receive CPU scheduling while the privileged component performs the actual work. When the privileged service finishes its task, it signals the unprivileged user process and then exits.

In this example, there are at least two logical channels so that the processes can communicate and synchronize. The first channel allows the unprivileged component to send commands to the privileged task. This channel will probably host a ‘Start Task’ message to initiate a defragmentation, and the ‘Stop Task’ message to prematurely end a defragmentation. The second channel allows the privileged component to signal that defragmentation is complete – the ‘Task Complete’ message.

In addition to communications and synchronizing actions between components, the programs will also most likely have guards in place to ensure that only one instance of their respective processes are in memory and executing.

On NT based systems, the signaling mechanisms ‘Start Task,’ ‘Stop Task,’ and ‘Task Complete’ are typically implemented as events; while the guards used to ensure a single instance are generally mutexes.

An adversary who wishes to subvert the defragmentation process has a number of vectors. First, if the either of the processes use a named mutex as a singleton, the attacker may block the component from starting by claiming the component’s mutex. Second, if the defragmentation processes launch successfully, the attacker can attempt to stop the defragmentation by setting (or pulsing) an event associated with the ‘Stop Task’ signal.

A less obvious vector for the adversary involves sending the ‘Start Task’ signal to the privileged process before it is ready to begin processing. In this case, the component may crash due to incomplete initialization or missing/bad data since the signal did not originate from the expected user program. The adversary could also send the user-space component a ‘Task Complete’ signal in hopes that its premature exit will cause the privileged component to exit accordingly.

The final example vector includes creating named objects *a priori*, but with either (1) the wrong object type, or (2) more restrictive permissions than the target program expects. In the earlier case, a handful of suites incorrectly handled the `NULL` handle with an error code of `ERROR_INVALID_HANDLE`. In the latter case, the target’s call to `Open` or `Create` will succeed, but the target will not be able to manipulate the object as expected.

6. Malicious Software

For academic purposes, we created malicious software which would perform the actions outlined in previous sections on software provided by Adobe, Sun, and Microsoft. The malware was loaded through `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`.

Attacks against Adobe and Sun trivially set a named event which caused their update components to exit. The attack against Security Essentials was more interesting – it demonstrated a critical component shutdown using unnamed handles and events that is unlikely to be detected by the user. The attack is described in Appendix A.

We chose Security Essentials for three reasons. First, Security Essentials has an aggressive security posture: the suite minimizes IPC by using only two components. Second, Microsoft appears to have minimized synchronization objects which could be vulnerable to attack.

Finally, Microsoft has patched Security Essentials, so the described attack is inert and no longer a threat to users (we apologize to the 0-day crowd). For those who are inclined, more interesting (and vulnerable) targets exist from other vendors.

7. Mitigation

Vendors affected by synchronization object manipulations are encouraged to apply a DACL with appropriate restrictions to process and thread objects. To mitigate attacks on a suite which originate from a standard user account, a suite should perform the following:

- Add a deny ACE to the process DACL which includes `PROCESS_DUP_HANDLE`
- Use unnamed synchronization objects and inheritable object handles

It is usually not possible to remediate the attacks using a deny ACE under a privileged account (such as Administrator) since the account can enlist either (1) the `WRITE_DAC` right to assign an accommodating ACL or (2) the `SE_DEBUG_NAME` or `SE_TCB_NAME` privileges to bypass customary access checks.

In the past, antivirus software might directly patch kernel components, such as the System Service Dispatch Table (SSDT), to extend Object Manager functionality. Since kernel patching is not supported by Microsoft and operating system integrity is enforced on the latest versions of the operating system via Microsoft's Kernel Patch Protection (Patch Guard), companies are encouraged to use the Kernel Data and Filtering Support API.

If using inheritable object handles, and the handles are not used by a child process, the parent should mark the handles as non-inheritable by calling `SetHandleInformation` after removing `HANDLE_FLAG_INHERIT`.

Suites should also reduce both process counts and thread counts. Suites with an abundance of processes and threads suffered attacks with greater frequency and effect than suites with fewer processes and threads. By reducing process and thread counts, fewer objects, such as those used in our attacks, are required by the suites. Suite components should also favor critical sections and condition variables over mutexes and events when possible.

To neutralize the deterministic creation of objects (and the subsequent recurring placement in the handle table), a program has two choices: first, randomize the creation of required objects; and second, create spurious objects on occasion. The spurious objects could also include canaries in an effort to determine if the environment has become hostile, and if so, take appropriate actions.

Finally, we hope Microsoft or security vendors will offer a configuration utility, similar to the Enhanced Mitigation Experience Toolkit (EMET), which would allow users to mitigate the behavior while reducing the need for software vendor support.

8. Acknowledgements

We would like to thank Gary McGraw, Jeffrey Richter and Kevin Wall for helpful comments.

Appendix A

In Figure 2 below, we demonstrate that an attacker can cause Security Essentials' privileged component (MsMpEng.exe) to exit by setting the 13th unnamed event. However, the action causes Security Essentials' desktop component to display warnings, which is undesirable for the malware. Though not pictured, the Security Essentials' task bar icon turns red during this time.

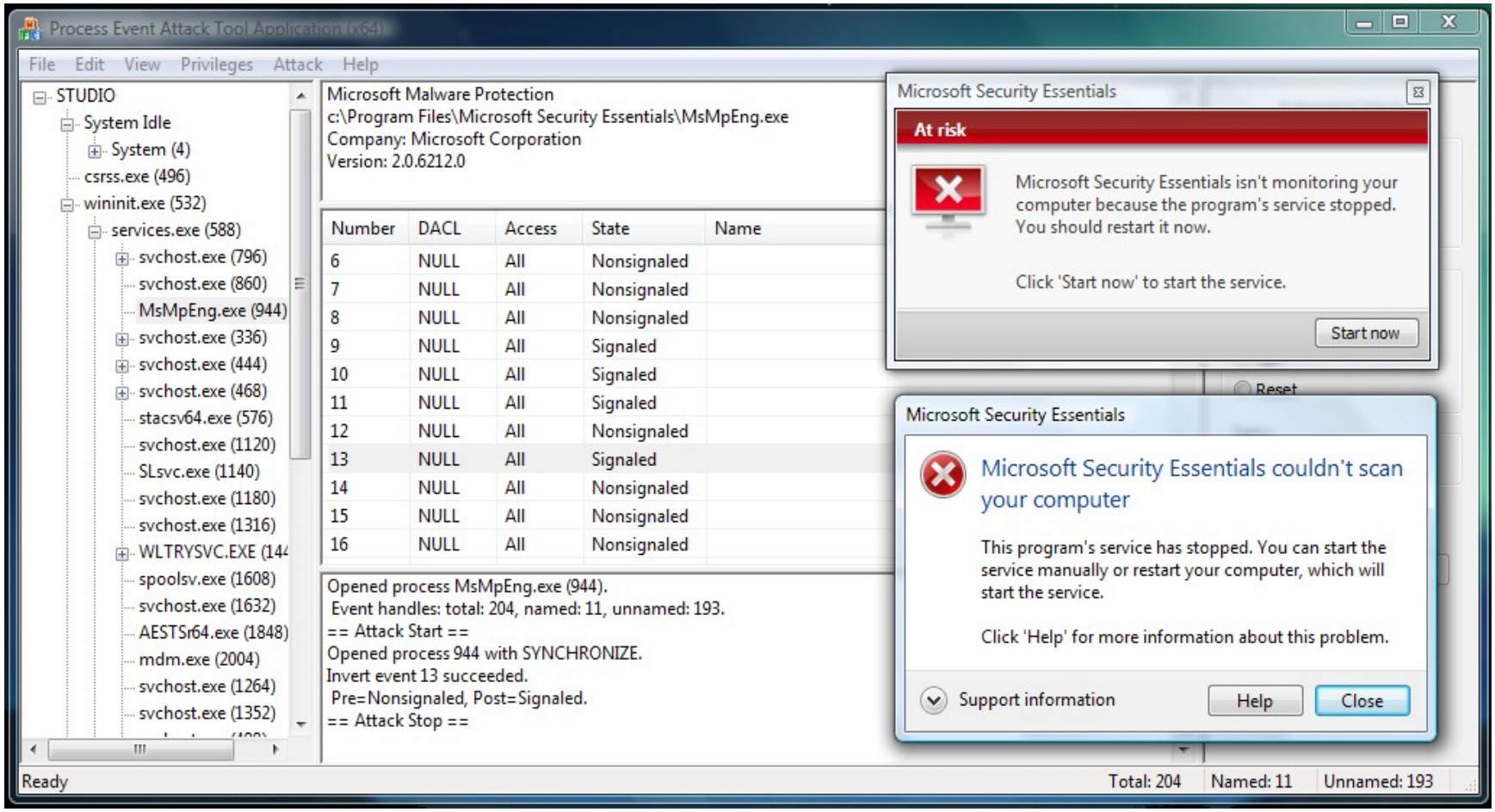

Figure 2: MsMpEng.exe shutdown

To inhibit Security Essentials' user component (MsSecEs.exe) from informing the user of the privileged component's exit, we simply squat unnamed mutex 7 from MsSecEs.exe. In Figure 3 below, the program on the right first squats the unnamed mutex (to block user warnings), and then the program on the left sets the unnamed event (to shut down the engine). The result of the attack is a surreptitious shutdown of the critical security component. Note that Security Essentials' task bar icon remains green – indicating normal operation to the user.

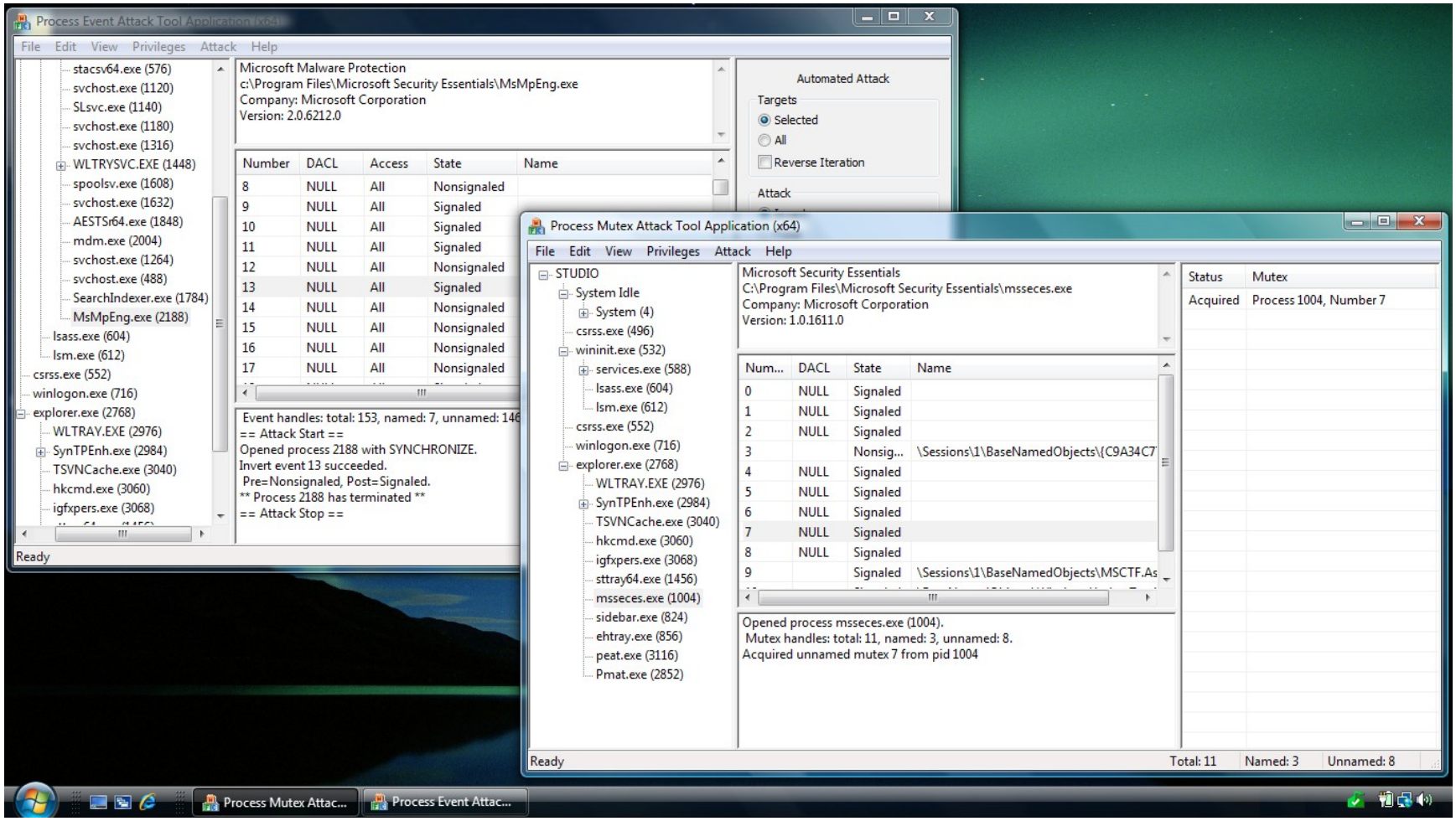

Figure 3: Surreptitious MsMpEng.exe shutdown