Threat Intelligence

EPS Processing Zero-Days Exploited by Multiple Threat Actors

May 9, 2017

Mandiant

Written by: Genwei Jiang, Alex Lanstein, Alex Berry, Ben Read, Dhanesh Kizhakkinan, Greg Macmanus

In 2015, FireEye published details about two attacks exploiting vulnerabilities in Encapsulated PostScript (EPS) of Microsoft Office. One was a zero-day and one was patched weeks before the attack launched.

Recently, FireEye identified three new zero-day vulnerabilities in Microsoft Office products that are being exploited in the wild.

At the end of March 2017, we detected another malicious document leveraging an unknown vulnerability in EPS and a recently patched vulnerability in Windows Graphics

EPS, FireEye detected a second unknown vulnerability in EPS.

FireEye believes that two actors – Turla and an unknown financially motivated actor – were using the first EPS zero-day (CVE-2017-0261), and APT28 was using the second EPS zero-day (CVE-2017-0262) along with a new Escalation of Privilege (EOP) zero-day (CVE-2017-0263). Turla and APT28 are Russian cyber espionage groups that have used these zero-days against European diplomatic and military entities. The unidentified financial group targeted regional and global banks with offices in the Middle East. The following is a description of the EPS zero-days, associated malware, and the new EOP zero-day. Each EPS zero-day is accompanied by an EOP exploit, with the EOP being required to escape the sandbox that executes the FLTLDR.EXE instance used for EPS processing.

The malicious documents have been used to deliver three different payloads. CVE-2017-0261 was used to deliver SHIRIME (Turla) and NETWIRE (unknown financially motivated actor), and CVE-2017-0262 was used to deliver GAMEFISH (APT28). CVE-2017-0263 is used to escalate privileges during the delivery of the GAMEFISH payload.

FireEye email and network products detected the malicious documents.

FireEye has been coordinating with the Microsoft Security Response Center (MSRC) for the responsible disclosure defense-in-depth measure against EPS filter vulnerabilities.

CVE-2017-0261 – EPS *"restore"* Use-After-Free

Upon opening the Office document, the FLTLDR.EXE is utilized to render an embedded EPS image, which contains the exploit. The EPS file is a PostScript program, which leverages a Use-After-Free vulnerability in "*restore*" operand.

From the PostScript Manual: "Allocations in local VM and modifications to existing objects in local VM are subject to a feature called **save** and **restore**, named after the operators that invoke it. **save** and **restore** bracket a section of a PostScript language program whose local VM activity is to be encapsulated. **restore** deallocates new objects and undoes modifications to existing objects that were made since the matching **save**."

As the manual described, the *restore* operator will reclaim memory allocated since the *save* operator. This makes a perfect condition of Use-After-Free, when combined with *forall* operator. Figure 1 shows the pseudo code to exploit the save and restore operation.

Figure 1: Pseudo code for the exploit

1. forall_proc array is created with a single element of the restore proc
2. The EPS state is ***saved*** to eps_state
3. uaf_array is created after the save
4. The forall operator loops over the elements of the uaf_array calling forall_proc for each element
5. The first element of uaf_array is passed to a call of restore_proc, the procedure contained in forall_proc
6. restore_proc
    - ***restores*** the initial state freeing the uaf_array
    - The alloc_string procedure reclaims the freed uaf_array
    - The forall_proc is updated to call leak_proc
7. Subsequent calls by the forall operator call the leak_proc on each element of the reclaimed uaf_array which elements now contain the result of the alloc_string procedure

Figure 2 demonstrates a debug log of the uaf_array being used after being reclaimed.

Figure 2: uaf_array reclaimed debug log

By manipulating the operations after the *save* operator, the attacker is able to manipulate the memory layouts and as 0x7fffffff, base as 0.

Figure 3: Faked String Object

Leveraging the power of reading and writing arbitrary user memory, the EPS program continues by searching for gadgets to build the ROP chain, and creates a ***file*** object. Figure 4 demonstrates the faked file object in memory.

Figure 4: Fake File Object, with ROP

By calling ***closefile*** operand with the faked file object, the exploit pivots to the ROP and starts the shellcode. Figure 5 shows part of the disassembler of ***closefile*** operand handler.

Figure 5: Stack Pivot disassembler of closefile

Once execution has been achieved, the malware uses the ROP chain to change the execution protection of the memory region containing the shellcode. At this point, the shellcode is running within a sandbox that was executing

FireEye detected two different versions of the EPS program exploiting this vulnerability. The first, st07383.en17.docx, continues by utilizing 32 or 64 bit versions of CVE-2017-0001 to escalate privileges before executing a final JavaScript payload containing a malware implant known as SHIRIME. SHIRIME is one of multiple custom JavaScript implants used by Turla as a first stage payload to conduct initial profiling of a target system and implement command and control. Since early 2016, we have observed multiple iterations of SHIRIME used in the wild, having the most recent version (v1.0.1004) employed in this zero-day

The second document, Confirmation_letter.docx, continues by utilizing 32 or 64 bit versions of CVE-2016-7255 to escalate privilege before dropping a new variant of the NETWIRE malware family. Several versions of this document were seen with similar filenames.

The EPS programs contained within these documents contained different logic to perform the construction of the ROP chain as well as build the shellcode. The first took the additional step of using a simple algorithm, shown in Figure 6, to obfuscate sections of the shellcode.

Figure 6: Shellcode obfuscation algorithm

CVE-2017-0262 – Type Confusion in EPS

execution flow allowing an attacker to control values onto the operand stack. This vulnerability was found in a document named "Trump's_Attack_on_Syria_English.docx".

Before triggering the vulnerability, the EPS program sprays the memory with predefined data to occupy specific memory address and facilitate the exploitation. Figure 7 demonstrates the PostScript code snippet of spraying memory with a string.

Figure 7: PostScript code snippet of spray

After execution, the content of string occupies the memory at address 0x0d80d000, leading to the memory layout as shown in Figure 8. The exploit leverages this layout and the content to forge a procedure object and manipulate the code flow to store predefined value, in yellow, to the operator stack.

Figure 8: Memory layout of the sprayed data

After spraying the heap, the exploit goes on to call a code statement in the following format: *1 array 16#D80D020 forall*. It creates an Array object, sets the procedure as the hex number 0xD80D020, and calls the execution flow to store values of the attacker's choices to operand stack. Figure 9 shows the major code flow consuming the forged procedure.

Figure 9: Consuming the forged procedure

After execution of *forall*, the contents on the stack are under the attacker's control. This is s shown in Figure 10.

Figure 10: Stack after the forall execution

Since the operand stack has been manipulated, the subsequent operations of *exch* defines objects based on the data from the manipulated stack, as shown in Figure 11.

Figure 11: Subsequent code to retrieve data from stack

The A18 is a string type object, which has a length field of 0x7ffffff0, based from 0. Within memory, the layout as shown in Figure 12.

Figure 12: A18 String Object

object and puts it into the forged array A19. By performing these operations, it puts the newly created array object pointer into A19. The exploit can then directly read the value from the predictable address, 0xD80D020 + 0x38, and leak its vftable and infer module base address of EPSIMP32.flt. Figure 13 shows code snippets of leaking EPSIMP32 base address.

Figure 13: Code snippet of leaking module base

Figure 14 shows the operand stack of calling *put* operator and the forged Array A19 after finishing the *put* operation.

Figure 14: Array A19 after the put operation

By leveraging the RW primitive string and the leaked module base of EPSIMP32, the exploit continues by searching ROP gadgets, creating a fake file object, and pivoting to shellcode through the *bytesavailable* operator. Figure 15 shows the forged file type object and disassembling of pivoting to ROP and shellcode.

Figure 15: Pivots to ROP and Shellcode

sandbox running FLTLDR.EXE, and then drop and execute a GAMEFISH payload. Only a 32-bit version of CVE-2017-0263 is contained in the shellcode.

CVE-2017-0263 – win32k!xxxDestroyWindow Use-After-Free

The EOP Exploit setup starts by suspending all threads other than the current thread and saving the thread handles to a table, as shown in Figure 16.

Figure 16: Suspending Threads

The exploit then checks for OS version and uses that information to populate version specific fields such as token offset, syscall number, etc. An executable memory area is allocated and populated with kernel mode shellcode as wells as address information required by the shellcode. A new thread is created for triggering the vulnerability and further control of exploitation.

The exploit starts by creating three PopupMenus and appending menus to them, as shown in Figure 17. The exploit creates 0x100 windows with random classnames. The User32!HMValidateHandle trick is used to leak the

Figure 17: Popup menu creation

RegisterClassExW is then used to register a window class “Main_Window_Class” with a WndProc pointing to a function, which calls DestroyWindow on window table created by EventHookProc, explained later in the blog. This function also shows the first popup menu, which was created earlier.

Two extra windows are created with class name as “Main_Window_Class”. SetWindowLong is used to change WndProc of second window, wnd2, to a shellcode address. An application defined hook, WindowHookProc, and an event hook, EventHookProc, are installed by SetWindowsHookExW and SetWinEventHook respectively. PostMessage is used to post 0xABCD to first window, wnd1.

The EventHookProc waits for EVENT_SYSTEM_MENUPOPUPSTART and saves the window’s handle to a table. WindowHookProc looks for **SysShadow** classname and sets a new WndProc for the corresponding window. Inside this WndProc, NtUserMNDragLeave syscall is invoked and SendMessage is used to send 0x9f9f to wnd2, invoking the shellcode shown in Figure 18.

The Use-After-Free happens inside WM_NCDESTROY event in kernel and overwrites wnd2's tagWnd structure, which sets bServerSideWindowProc flag. With bServerSideWindowProc set, the user mode WndProc is considered as a kernel callback and will be invoked from kernel context – in this case wnd2's WndProc is the shellcode.

The shellcode checks whether the memory corruption has occurred by checking if the code segment is not the user mode code segment. It also checks whether the message sent is 0x9f9f. Once the validation is completed, shellcode finds the TOKEN address of current process and TOKEN of system process (pid 4). The shellcode then copies the system process' token to current process, which elevates current process privilege to SYSTEM.

Conclusion

EPS processing has become a ripe exploitation space for attackers.

FireEye has discovered and analyzed two of these recent EPS zero-days with examples seen before and after Microsoft disabled EPS processing in the April 2017 Patch Tuesday. The documents explored utilize differing EPS exploits, ROP construction, shellcode, EOP exploits and final payloads. While these documents are detected by

Russian cyber espionage is a well-resourced, dynamic threat

The use of zero-day exploits by Turla Group and APT28 underscores their capacity to apply technically sophisticated and costly methods when necessary. Russian cyber espionage actors use zero-day exploits in addition to less complex measures. Though these actors have relied on credential phishing and macros to carry out operations previously, the use of these methods does not reflect a lack of resources. Rather, the use of less technically sophisticated methods – when sufficient – reflects operational maturity and the foresight to protect costly exploits until they are necessary.

A vibrant ecosystem of threats

CVE-2017-0261's use by multiple actors is further evidence that cyber espionage and criminal activity exist in a shared ecosystem. Nation state actors, such as those leveraging CVE-2017-0199 to distribute FINSPY, often rely on the same sources for exploits as criminal actors. This shared ecosystem creates a proliferation problem for defenders concerned with either type of threat.

CVE-2017-0261 was being used as a zero-day by both nation state and cyber crime actors, and we believe that both actors obtained the vulnerability from a common source. Following CVE-2017-0199, this is the second

MD5	Filename
2abe3cc4bff46455a945d56c27e9fb45	Confirmation (NETWIRE)
e091425d23b8db6082b40d25e938f871	Confirmation (NETWIRE)
006bdb19b6936329bffd4054e270dc6a	Confirmation (NETWIRE)
15660631e31c1172ba5a299a90938c02	st07383.en17 (SHIRIME)
f8e92d8b5488ea76c40601c8f1a08790	Trump's_Atta (GAMEFISH)

Table 1: Source Exploit Documents

Table 2: CVEs related to these attacks

iSIGHT Intelligence Team, FLARE Team, FireEye Labs, Microsoft Security Response Center (MSRC).

Posted in Threat Intelligence—Security & Identity

Related articles

Threat Intelligence

Hybrid Russian Espionage and Influence Campaign Aims to Compromise Ukrainian Military Recruits and Deliver Anti-Mobilization Narratives

By Google Threat Intelligence Group • 10-minute read

Threat Intelligence

Investigating FortiManager Zero-Day Exploitation (CVE-2024-47575)

By Mandiant • 19-minute read

Threat Intelligence

How Low Can You Go? An Analysis of 2023 Time-to-Exploit Trends

By Mandiant • 10-minute read

Threat Intelligence

capa Explorer Web: A Web-Based Tool for Program Capability Analysis

By Mandiant • 6-minute read