

Exploit Developer Spotlight: The Story of PlayBit

research.checkpoint.com/2020/graphology-of-an-exploit-playbit/

October 26, 2020



October 26, 2020

Research By: Eyal Itkin and Itay Cohen

Introduction

Exploits have always been an important and integral part of malicious attacks. They allow attackers to gain capabilities that are not easy to achieve otherwise. Whether attackers strive to gain higher privileges on a given computer, or laterally move inside a network, exploits often play a key role in their plan. While in-the-wild exploits are used by many malware families, the malware authors get most of the attention, and the exploit developers remain out of the spotlight.

As part of our effort to focus on the exploit developers themselves, we previously shared our methodology and technique of fingerprinting and tracking exploit developers. In our [earlier publication](#), we thoroughly explained our methodology and focused on *Volodya*, a prominent exploit developer we tracked using the unique fingerprints left in their exploits. Now our focus is on another exploit developer known as *PlayBit* or *luxor2008*.

Initial Sample – CVE-2018-8453

As our research technique of fingerprinting exploit writers exceeded our initial expectations, we were on the lookout for more exploits to investigate. Soon enough, we came across [this blog post](#) from Kaspersky detailing how Sodin (a.k.a Sodinokibi, or REvil), an infamous ransomware, is using a 1-Day exploit for CVE-2018-8453.

CVE-2018-8453 is an interesting case, as it was formerly [caught in the wild](#) by Kaspersky when used by FruityArmor. From their report, it was clear that this exploit was reimplemented by **another** actor. While we would prefer to investigate an exploit developed by the actor behind the 0-Day exploit, we had to settle for the exploit used in REvil.

Even from this one sample, it was clear that this new actor uses a totally different exploit template than the one used by Volodya. Luckily for us, the actor chose to implement new features that weren't in Volodya's exploit template, which gave us a wider choice of artifacts to hunt for. After we created a few hunting rules, we set out to pursue our quarry.

Our actor's exploits

All of the exploits that we found related to this actor were 1-Day exploits for Local Privilege Escalation (LPE) vulnerabilities in Windows. This list of 5 different CVEs that were exploited eventually led us to our actor's identity: *PlayBit* (a.k.a *luxor2008*). A full profile of the actor can be found later on in this blog post, under the "Intelligence Report" section.

CVE-2013-3660

Classification: 1-Day

Basic Description: Uninitialized kernel pointer in `E_PATHOBJ::pprFlattenRec`

Used by the following malware families: Dyre, Ramnit

Background on this vulnerability:

This vulnerability was originally [found](#) by Google's Tavis Ormandy, and made [headlines](#) due to an unusual disclosure process as Microsoft was not notified about the vulnerability before the full disclosure.

```
1 int print_exploit_banner()
2 {
3     printf("\n\n");
4     printf("#####\n");
5     printf("#           Privilege Escalation Exploit           #\n");
6     printf("#           Version(x86) %d.%d.%04d           #\n", 1, 1, 2055);
7     printf("#           #\n");
8     printf("# Vulnerability: CVE-2013-3660 (Published: July 09, 2013) #\n");
9     printf("# Supported versions: XP/2003/Vista/2008/W7/W8/2012 #\n");
10    printf("# Supported architecture: x86/x64 #\n");
11    printf("# #\n");
12    printf("#####\n");
13    printf("#           coded by PlayBit (c) 2013           #\n");
14    printf("#           xmp: playbit@jabbim.cz           #\n");
15    return printf("#####\n\n");
16 }
```

Figure 1: Decompiled code showing PlayBit's exploit banner, as seen in IDA Pro.

CVE-2015-0057

Classification: 1-Day

Basic Description: Use-After-Free in `win32k!xxxEnableWndSBArrows`

Used by the following malware families: Dyre, Evotob

Background on this vulnerability:

Also known as MS15-010, this vulnerability was found by Udi Yavo, the CTO of enSilo. The [public disclosure](#) of the vulnerability included a technical explanation of the vulnerability as well as the exploitation plan. In addition, the authors specifically mentioned they were able to exploit this vulnerability on all Windows versions, and even supplied a demo video.

Even though the disclosure didn't share any exploit code, it was enough to draw our actor's attention. Three months after the patch, PlayBit already had a working exploit, and a month later this exploit was [used by the Dyre Banking Trojan](#). The exceptional case behind this vulnerability is described in more detail in FireEye's Black Hat [presentation](#) on CVE-2015-0057.

CVE-2015-1701

Classification: 1-Day

Basic Description: `CreateWindow` callback validation error

Used by the following malware families: Locky

Background on this vulnerability:

Also known as MS15-051, this vulnerability was originally exploited as a **0-Day** in an [operation attributed to APT28](#).

CVE-2016-7255

Classification: 1-Day

Basic Description: Memory corruption in `NtUserSetWindowLongPtr`

Used by the following malware families: LockCrypt

Background on this vulnerability:

Originally [exploited as a 0-Day](#), once again attributed to APT28. This 0-Day was found and exploited by Volodya, later to be exploited again and sold by PlayBit.

CVE-2018-8453

Classification: 1-Day

Basic Description: Use-after-free in `win32kfull!xxxDestroyWindow`

Used by the following malware families: REvil (Sodinokibi), Maze, Neshta

Background on this vulnerability:

Originally exploited as a **0-Day**, and [attributed to FruityArmor](#).

As [bitdefender reported](#), Maze ransomware uses two different LPE exploits: CVE-2016-7255, and CVE-2018-8453. What's interesting is that they use Volodya's exploit for CVE-2016-7255 (like GandCrab, and many other malware families), while transitioning to using PlayBit's exploit for CVE-2018-8453. Not only do we see both Volodya and PlayBit selling their exploits to the same malware actor, we can take this opportunity to learn about the dynamics involved. While PlayBit sells their own exploit for CVE-2016-7255, Maze chose to use Volodya's exploit even though they also purchased an exploit from PlayBit (CVE-2018-8453).

It seems a bit unusual that when purchasing their LPE exploits, the actors behind Maze decided to buy from two different sellers, especially as PlayBit sells both of these exploits. However, an important fact to remember is that Maze ransomware was first discovered in 2019. Therefore, our theory is that at least in some campaigns, the operators behind Maze simply "inherited" the first exploit, later purchasing another one to target more victims.

Mindset of a 1-Day exploit seller

When going over the list of CVEs that were exploited by PlayBit, we found a unique pattern that they all share: All of the vulnerabilities were already "famous" before PlayBit decided to implement an exploit for them.

- **CVE-2013-3660** – Made headlines due to an unusual disclosure process.
- **CVE-2015-0057** – The public disclosure of the vulnerability included a detailed explanation of the vulnerability and the exploitation plan, and to this day is still highly regarded as a novel exploitation technique.
- **CVE-2015-1701** – Originally exploited as a 0-Day.
- **CVE-2016-7255** – Originally exploited as a 0-Day.
- **CVE-2018-8453** – Originally exploited as a 0-Day.

As we can see, all of the CVEs that our actor chose to exploit as 1-Days were already well known. While we didn't find a similar tactic for Volodya when choosing which 1-Day vulnerabilities to exploit, it does seem that this helped PlayBit advertise their exploits in underground forums.

One more characteristic that we found in all of PlayBit's exploits that were lacking in Volodya's, is the exploitability check. PlayBit supplies the customer with a thin wrapper around the exploit, which checks whether the target computer is indeed vulnerable. Although the check varies a bit between different exploits and versions, the basics are the same: the modification date of the vulnerable win32k driver is checked, to detect if a patch was installed.

```

snwprintf(&SourceString[-var4 - 2], 0x104 - (-var4 - 2), "%S", "win32k.sys");
RtlInitUnicodeString(&DestinationString, SourceString);
ObjectAttributes.ObjectName = &DestinationString;
ObjectAttributes.Length = 48;
ObjectAttributes.RootDirectory = 0;
ObjectAttributes.Attributes = 64;
ObjectAttributes.SecurityDescriptor = 0;
ObjectAttributes.SecurityQualityOfService = 0;
// Open win32k.sys
iVar2 = NtOpenFile(&FileHandle, 0x100080, &ObjectAttributes, &IoStatusBlock, 3, 0x60);
if (-1 < iVar2) {
    // Query for its modification date
    iVar2 = NtQueryInformationFile(FileHandle, &IoStatusBlock, FileInformation, 0x28, FileBasicInformation);
    NtClose(FileHandle);
    if (-1 < iVar2) {
        RtlTimeToTimeFields(&Time, &TimeFields);
        log_print(1, (int64_t)"%s last write date: %04d/%02d/%02d", "win32k.sys",
            TimeFields.Year,
            TimeFields.Month,
            TimeFields.Day);
        // Check if it was updated after the release of the patch
        if (TimeFields.Year < 2015 || TimeFields.Year == 2015 &&
            (TimeFields.Month < 2 || TimeFields.Month == 2 && TimeFields.Day < 10)) {
            iVar5 = 1;
        }
    }
}

```

Figure 2: Checking if `win32k.sys` was modified after February 10, 2015 (CVE-2015-0057). Screenshot from Cutter.

This operational decision was very useful from our perspective, as the exploit effectively tells us in which Patch Tuesday the exploited vulnerability was fixed:

- **CVE-2013-3660** – The first sample of this exploit didn't yet contain such a check.
- **CVE-2015-0057** – `win32k.sys` checked for a modification date of February 10, 2015.
- **CVE-2015-1701** – `win32k.sys` checked for a modification date of May 13, 2015.
- **CVE-2016-7255** – `win32k.sys` checked for a modification date of November 8, 2016.
- **CVE-2018-8453** – `win32k.sys` / `win32kfull.sys` are checked for a modification date of September 11, 2018.

The author's fingerprints

Now that we found 5 different exploits from PlayBit, we can review them in greater detail and familiarize ourselves with the actor's work habits. As was the case with Volodya, it was clear to us from the beginning that PlayBit probably has a simple template to deploy for the different exploits.

As some of the actor's characteristics were already used in the previous [comparison to Volodya](#), we chose to focus instead on other implementation decisions, some of which aren't even included in Volodya's exploits.

Hash-Based Imports

In every exploit, PlayBit picks a handful of important functions and obfuscates their use. Instead of just importing these functions in the PE level, or using their cleartext strings and importing them using `GetProcAddress()`, PlayBit devised their own import technique.

```
84: load_imports ();
push    esi
mov     esi, ecx
mov     ecx, 0x198f48f8
mov     edx, esi
call   load_import_by_crc
mov     edx, esi
mov     dword [PsInitialSystemProcess], eax
mov     ecx, 0xe294ed35
call   load_import_by_crc
mov     edx, esi
mov     dword [PsReferencePrimaryToken], eax
mov     ecx, 0x4aacc974
call   load_import_by_crc
mov     edx, esi
mov     dword [PsLookupProcessByProcessId], eax
mov     ecx, 0xbc41b0a9
call   load_import_by_crc
mov     dword [ObDereferenceObject], eax
xor     eax, eax
cmp     dword [PsInitialSystemProcess], eax
pop     esi
setne  al
ret
```

Figure 3: Loading the addresses of `Ps*` symbols based on their hash/CRC.

Here is a short Python snippet that performs this “hash” calculation (more like a checksum / CRC):

```
from malduck import ror

def calc_hash(export_str):
    crc_value = 0
    for c in export_str:
        cur_value = ord(c)
        # convert lower case back to upper case
        if cur_value >= ord('a'):
            cur_value -= 0x20 # 'a' - 'A' = 0x20
        crc_value = ror(crc_value, 13) + cur_value
    return crc_value
```

Not only is this hash-based import technique used in all of the actor’s exploits, we were also able to find it in other tools they sold over the years, going back to 2012.

When examining the functions imported using this mechanism, we found out they all have a common trait. All of these functions are used by the “shellcode”, the exploitation part that is executed with high privileges and is responsible for elevating the permissions of the target

process. This also means that it is relatively easy to locate the shellcode in each of the exploits, as it references the global variables that store the addresses of the specially imported functions.

OS Fingerprinting

Like any other experienced exploit developer attempting to target as many OS versions as possible, our actor needs to fingerprint the exact version of the immediate target. This means that the exploit identifies and calibrates itself to the target's Windows version. In contrast to Volodya, it seems that PlayBit is interested in everything they can learn about the target:

- OS major number
- OS minor number
- Service pack
- Server or a standalone computer
- Windows 10 build number

```
// Parse OS versions
pPEB = NtCurrentPeb();
major_version = pPEB->OSMajorVersion;
minor_version = pPEB->OSMinorVersion;
if ( major_version == 6 )
{
    if ( !minor_version )
    {
        if ( is_pc_not_server == 1 )           // Vista
        {
            if ( service_pack )
            {
                if ( service_pack == 1 )
                    global_os_version_enum = 8;           // 6.0, SP1
                else
                    global_os_version_enum = 9;           // 6.0, SP2
            }
            else                                     // 6.0, SP0
            {
                global_os_version_enum = 7;
            }
        }
    }
}
```

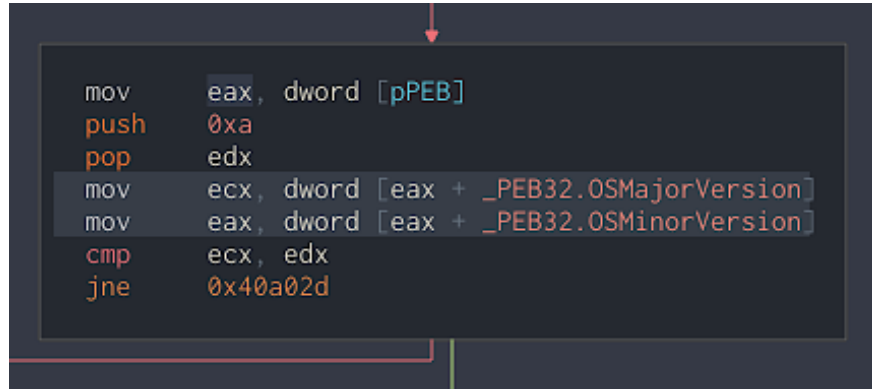
Figure 4: Querying for the exact OS version, as can be seen in Cutter.

GetVersionEx()

While this API call is used in all of the exploits, it was used exclusively only in the exploit of CVE-2013-3660. As this API became deprecated, the exploits from 2015 and later only use it when querying for the service pack and the OS type. Both the major number and minor number of the Windows OS are now queried using a different technique.

Accessing the PEB

Switching from the now deprecated `GetVersionEx()`, PlayBit chose to query the Process-Environment-Block (PEB).



```
mov     eax, dword [pPEB]
push   0xa
pop    edx
mov     ecx, dword [eax + _PEB32.OSMajorVersion]
mov     eax, dword [eax + _PEB32.OSMinorVersion]
cmp     ecx, edx
jne     0x40a02d
```

Figure 5: Extracting the major and minor versions from the PEB, as can be seen in Cutter.

This is a clear difference in the *modus operandi* of PlayBit as compared to Volodya. Not only do they extract the same information in different ways, but Volodya is also interested in far less information than PlayBit, even when they both exploit the same vulnerability (CVE-2016-7255).

In general, both actors hold detailed version-specific configurations from which they load the relevant information once the OS version is determined. The main difference between the two is that the code flow in Volodya's exploits rarely depends on the OS version, while PlayBit incorporates multiple twists and turns using various if-checks that depend on the OS version.

Leaking Kernel Addresses

In contrast to what we initially thought, a closer examination of PlayBit's exploits showed that they do indeed contain a kernel-pointer-leak primitive. Not only do the exploits contain such a leak, the chosen leak primitive hints at a high level of understanding of the internals of Windows. PlayBit directly accesses the Desktop Heap stored in user-mode, via the `Win32ClientInfo` field stored in the Thread-Environment-Block (TEB).

```
pTEB = NtCurrentTeb();
pUserDesktopHeap = (int32_t *)pTEB->Win32ClientInfo[6];
ulClientDelta = pTEB->Win32ClientInfo[7];
desktop_kernel_address = *pUserDesktopHeap;
pCtx->ulClientDelta = ulClientDelta;
pCtx->desktop_kernel_address = desktop_kernel_address;
pCtx->desktop_user_address = desktop_kernel_address - ulClientDelta;
```

Figure 6: Extracting information on the Desktop Heap, using TEB's `Win32ClientInfo`.

As Microsoft gradually fixed the design issue that allowed for this pointer-leak, PlayBit had to implement some updates along the way to allow this technique to continue to work. For instance, Creators Update removed the `ulClientDelta` field from `Win32ClientInfo`, mandating that the exploit for CVE-2018-8453 calculate it manually. This technique was finally patched by Microsoft in Windows 10 RS4.

The leak primitive enables PlayBit to learn the kernel addresses of the windows created during the exploitation. In turn, this information is used in several phases during the exploitation:

- The addresses are used for pointing to / creating fake kernel objects.
- The kernel memory shaping is measured by the distance between the leaked addresses.
- The exploits usually invoke a kernel shellcode, and we need to know where it is stored.

The last point is explained in more detail in the next section.

SMEP Bypass

Generation 0 – No Bypass

In the initial versions of the earlier exploits, such as the exploit for CVE-2013-3660 and some versions of CVE-2015-0057, the exploit caused the kernel to execute a token-swapping shellcode stored in user-mode.

```
__fastcall token_swap(int32_t *pScanHead, int32_t self_token, int32_t system_token)
{
    undefined4 uVar1;
    uint32_t aligned_self_token;
    uint32_t search_loop_index;
    int32_t var_4h;

    aligned_self_token = self_token & 0xffffffff8;
    search_loop_index = 0;
    do {
        if ((*pScanHead & 0xffffffff8U) == aligned_self_token) {
            LOCK();
            aligned_self_token = *pScanHead;
            *pScanHead = system_token;
            uVar1 = 1;
            goto return_statement;
        }
        search_loop_index = search_loop_index + 1;
        pScanHead = (int32_t *)((uint32_t *)pScanHead + 1);
    } while (search_loop_index < 0x300);
    uVar1 = 0;
return_statement:
    return CONCAT44(aligned_self_token, uVar1);
}
```

Figure 7: PlayBit exploit scanning the `EPROCESS` in search for the token, as can be seen in Cutter.

The downside of this form of privilege-elevation is that SMEP prohibits the kernel from executing code that is stored in user-mode. Instead of changing the way the token-swap is performed, PlayBit decided to add an additional layer that turns off SMEP. This way, the problem is reduced to the same token-swap problem that earlier exploits already solved. A classic academic solution.

Generation 1 – Kernel shellcode

In this set of exploits, an additional kernel shellcode was added. This bootloader code is copied to kernel-space and stored as the “window name” of one of the windows corrupted by the exploit. Upon execution, the code modifies CR4 to disable the SMEP support, and then jumps back to the original user-mode payload.

```
cpuid_Extended_Feature_Enumeration_info(7);
if ( (EBX & 0x80) != 0 )
{
    val_cr4 = readcr4();
    writecr4(val_cr4 & 0xFFEFFFFF);
}
```

Figure 8: Checking for kernel-mode, and masking out the SMEP-related bit out of CR4.

The exploit copies the payload to the kernel using the `NtUserDefSetText` syscall, which explains why this syscall is included in the per-version configuration of all of the relevant exploits.

Note that this SMEP bypass is based on the fact that the above code snippet can be executed, in kernel-mode, even though it should have been a window name. The fact that memory pools were flagged as “executable” (or more specifically, weren’t flagged as “non-executable”), was used by many attackers. Eventually, Microsoft noticed this design flaw in the kernel’s memory and introduced Non-eXecutable (NX) pools to the kernel.

Once again, PlayBit decided to overcome this added restriction using yet another reduction, implemented in the exploit for CVE-2018-8453.

Generation 2 – Disabling the NX bit from the kernel shellcode

This additional change was more complicated than the previous one, as it also demanded the use of Arbitrary-Read and Arbitrary-Write kernel primitives. While the use of these primitives allows for a cleaner exploit to begin with, as Volodya was doing since 2015, PlayBit only added it as an additional step in the reduction to the basic user-mode token-swap payload.

During this step, the page table entries are traversed in search of the entry associated with the kernel shellcode. Once found, the entry is masked-out of the XD (eXecute Disable) bit, and stored back in the table. From this point on, the memory page containing the kernel shellcode is executable, and thus the privilege-escalation chain can start.

```
    pCurrentAddress = ArbitraryRead(pctx, pCurrentAddress);
    if (pCurrentAddress) &&
    {
        cpCurrentAddress = ArbitraryRead(pctx, pCurrentAddress + *(_global_os_version_enum * 0x13 + 0x140002720 + 0x389))
        if (6 < _global_os_version_enum) {
            if (_global_os_version_enum < 0x16) {
                cpCurrentAddress = 0xffffffff6800000000; // PTE Pages address (Before Randomized)
            } else {
                cpCurrentAddress = ArbitraryRead(pctx, *(int64_t *)0x1400063c0);
            }
        }
        if (cpCurrentAddress) {
            CR4_patched_page_table_entry = (pctx[72] >> 9 & 0x7fffffff8) + cpCurrentAddress;
            cpCurrentAddress = ArbitraryRead(pctx, CR4_patched_page_table_entry);

            if (cpCurrentAddress)
            {
                // Updating the Page Table Entry (removing SMEP)
                cpCurrentAddress = ArbitraryWrite(pctx,
                    cpCurrentAddress & 0x7fffffffffffffff,
                    cpCurrentAddress,
                    1);
            }
        }
    }
}
```

Figure 9: Traversing the Page Table and removing the XD bit from the record of the kernel shellcode.

An exploit framework

After finding enough samples that adhere to PlayBit's exploit template for Windows memory corruption LPEs, we decided to perform one more check. Using exploit-specific artifacts from each of the exploits, we created an additional set of hunting rules and did one more search.

In this additional search, we found Ramnit samples which contain an exploit for CVE-2013-3660, but with some changes:

- When checking if the target is vulnerable or patched, instead of looking for the driver's modification date, the exploit searches for a specific patch id in the registry, as can be seen in Figure 10 below.
- PlayBit's hash-based imported functions are no longer used; `GetProcAddress()` is used instead.

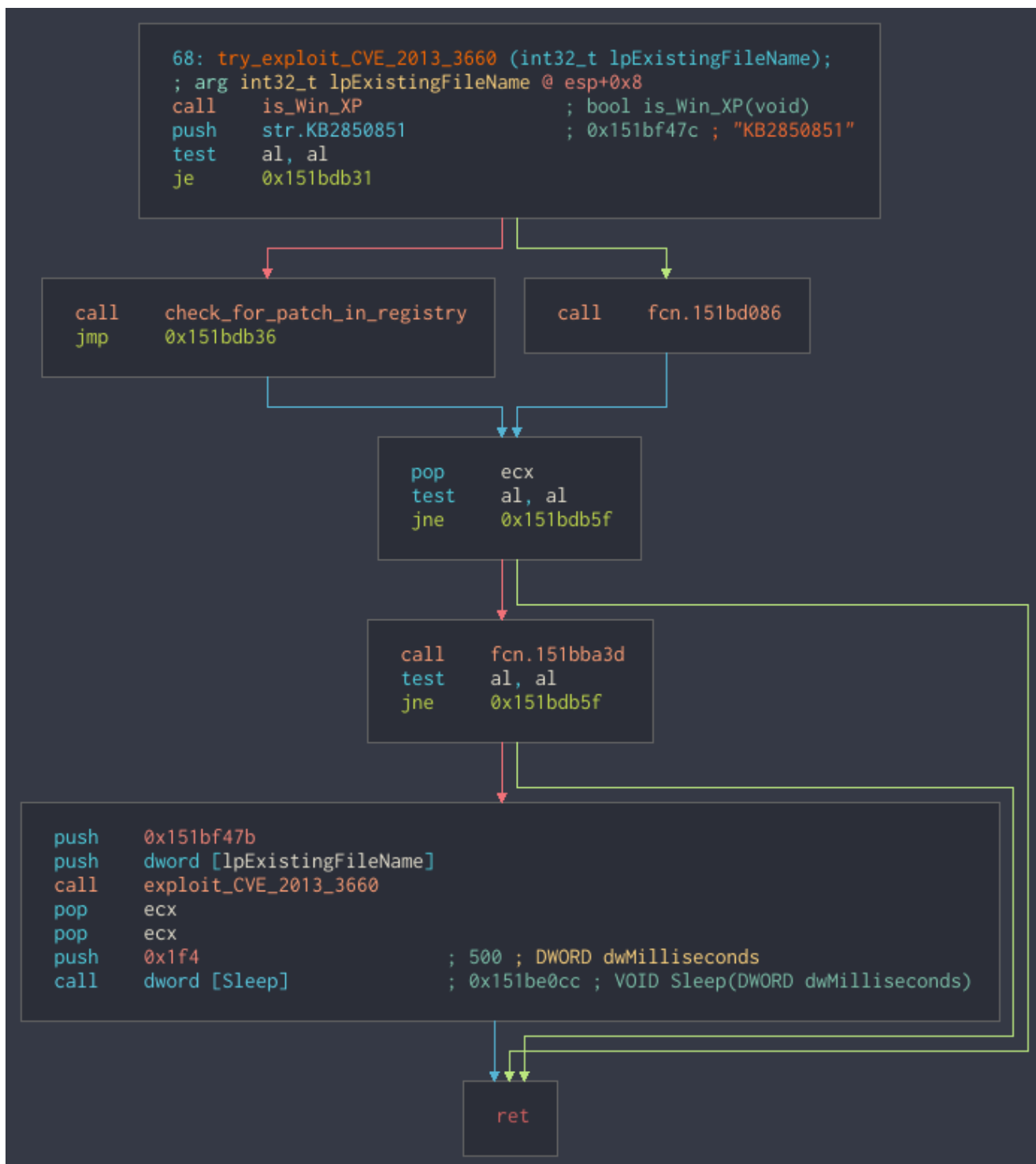


Figure 10: A check for KB2850851 before exploiting CVE-2013-3660.

Our Ramnit sample exploits both CVE-2013-3660 (by PlayBit) and CVE-2014-4113 (using the same exploit code originally found as a 0-Day). The original exploit for CVE-2014-4113 was part of an exploit framework in which the API passes a command-line argument, and that command is executed as SYSTEM. As that wasn't the original API for PlayBit's exploit, some adjustments were made and PlayBit's exploits were re-adjusted to receive a command-line argument to be executed once elevated.

When further investigating this new framework, we saw that it matched FireEye's report on the Dyre Banking Trojan. As Dyre exploited CVE-2013-3660 and CVE-2015-0057, both of which were written by PlayBit, this means that during the lifetime of this exploit framework, it included at least the following 3 exploits:

- CVE-2013-3660 – PlayBit.
- CVE-2014-4113 – 1-Day use of the original 0-Day exploit. Unknown author.
- CVE-2015-0057 – PlayBit.

Aside from connecting PlayBit to this now defunct exploit framework, we can deduce additional conclusions regarding our exploit-based hunting technique:

- Our hunting is mainly based on an actor's given exploit template.
- When working with other associates, and collaborating on a given exploitation framework, this template is replaced with the agreed-upon design modifications.
- Given an initial exploitation framework sample, it is easier to find similar framework-related samples, than it is to find independent exploits of one of the authors who contributed to this framework.

Essentially, both the attribution and the hunting are more complicated as the exploit framework masks many telltale-signs of the individual exploit developer.

Intelligence Report

In contrast to the previous blog post on Volodya, we decided this time to perform an additional "Background Check" on PlayBit. Being unfamiliar with this actor, we thought it would be a good chance to better understand how they work. In addition, we could test how well our exploit-based hunting technique works by comparing the found list of advertisements to the actual samples we caught.

PlayBit (a.k.a luxor2008)

Funnily enough, attributing the exploits to our exploit writer was quite simple. It turns out that alongside the use of numerous underground forums, there are also public YouTube channels advertising the actor's exploits.

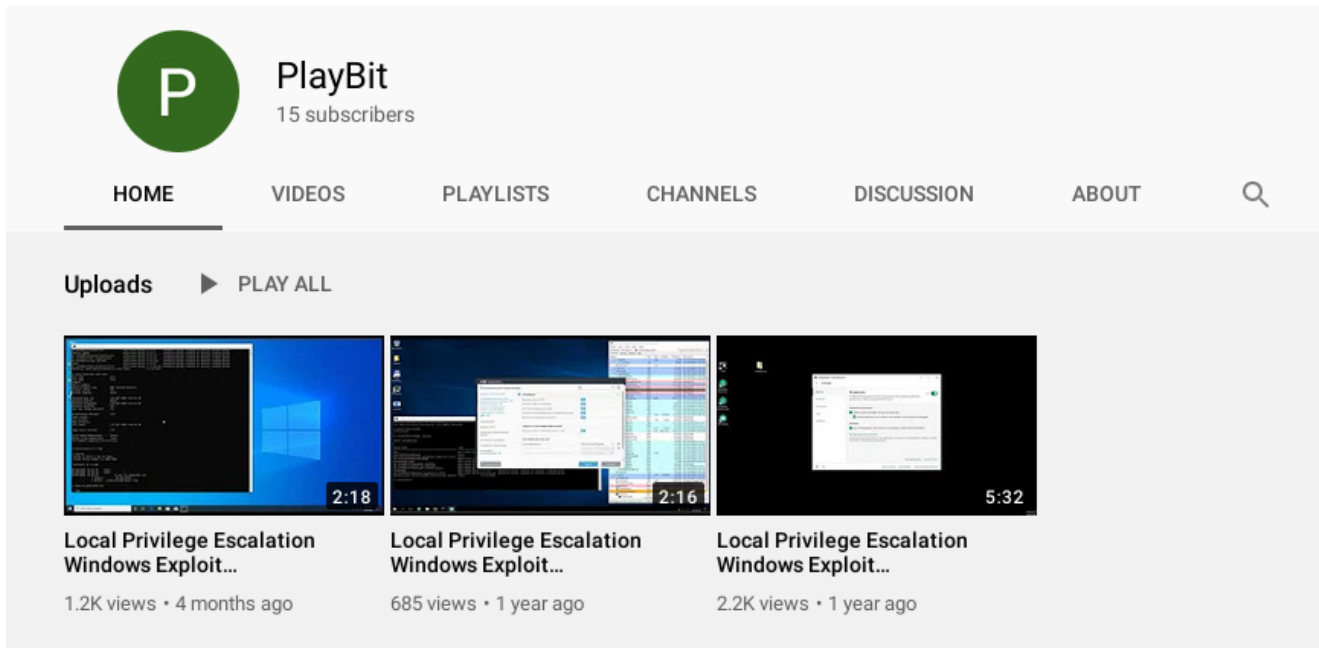


Figure 11: A screenshot from one of PlayBit's YouTube channels.

For some unknown reason, there are actually two YouTube channels: one for earlier exploits, and another one (still active) for more recent exploits. As the videos publicly state which CVE is being exploited, the attribution process was very short.

When trying to learn more about this actor, we only found a single [slim report](#). Therefore, we decided to include a detailed profile of the actor in this blog post.

The Actor's Advertisements

Aside from the YouTube channels we mentioned earlier, the actor used multiple platforms to advertise the vulnerabilities.

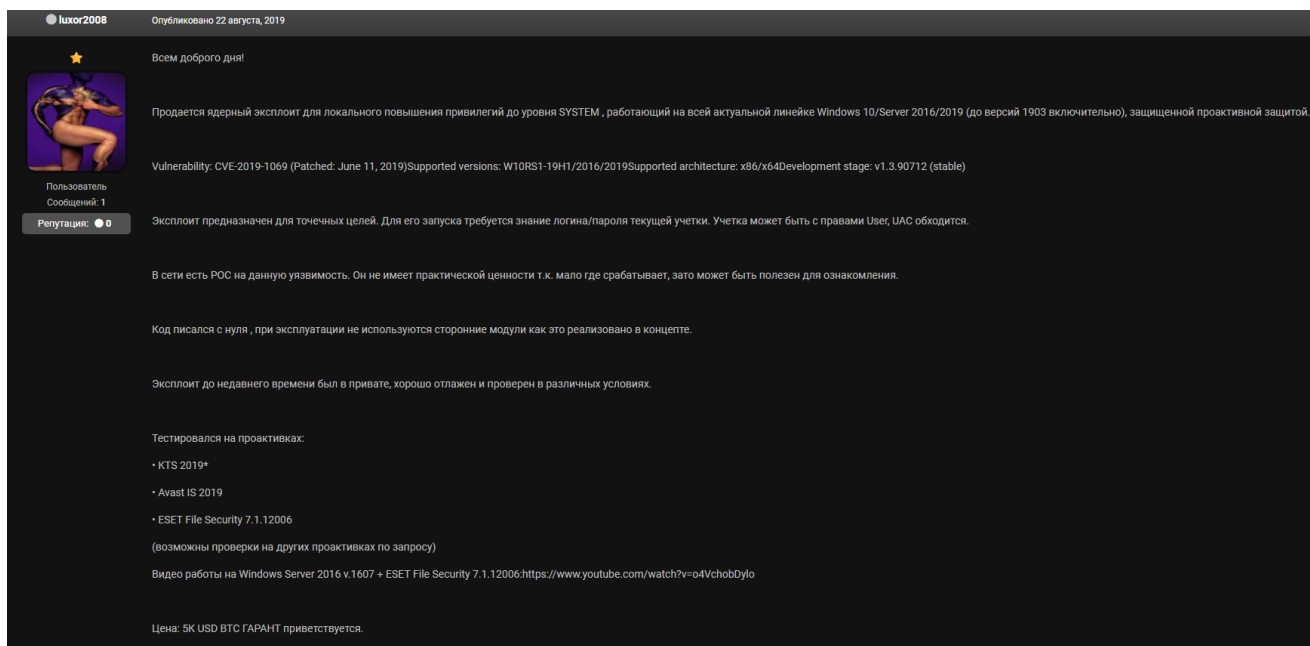


Figure 12: Forum advertisement for CVE-2019-1069.

Whether the ads were placed on underground forums, YouTube, or even Pastebin, they all shared a common template:

“

Ring0 LPE Exploit CVE-2015-0057 [All win versions]

Vulnerability: CVE-2015-0057 (Published: February 10, 2015)

Supported versions: XP/2003/Vista/2008/W7/W8/2012/W8.1/2012R2/W10TP

Supported architecture: x86/x64

Development stage: v1.1.1900 (stable)

Shellcode size x86: 13Kb

Shellcode size x64: 16Kb

Bypass all possible Windows security defences:

- *SMEP*
- *Kernel DEP*
- *KASLR*
- *Integrity Level (escape from Low)*
- *NULL Dereference Protection*
- *UAC*

There are no public POCs on this vulnerability. Shellcode has ready for immediate use in your projects. The test demo sources are supplied. Exploit is extremely stable, no bsods or error messages. Can be run under Guest account from Low Integrity Level.

Successfully bypasses proactive defences of:

- *KIS 2015*
- *Avast IS 2015*
- *ESET Smart Security 8*

“

These extensive ads emphasize the fact the exploit isn't detected by AV vendors, and that public POCs do not exist or are of poor quality when compared to the exploit. The wide range of supported Windows versions is also evident: If a given Windows version is vulnerable to the exploited vulnerability, you can be sure that PlayBit's exploit supports it.

Years of activity

We successfully traced PlayBit to various exploits and tools going as far back as 2012. On average, the actor sells a single Windows LPE 1-Day exploit per year, and that includes recent exploits for both CVE-2019-1069 (a [SandboxEscaper](#) vulnerability) and CVE-2020-0787 (yet another logical vulnerability).

Transition to exploit logical vulnerabilities

While we caught samples of 5 memory-corruption Windows LPE vulnerabilities, the ads show that in the last year or so the actor shifted their interest. Both of the recently sold exploits are now more logical in nature, and may indicate a trend in the exploitation market. Knowing that PlayBit's previous exploits relied on a design flaw that was fixed by Microsoft in Windows 10 RS 4 (released on April 30, 2018), it could be that Microsoft's mitigations are one reason behind this shift in focus.

Pricing

As [previously reported](#) by Kaspersky, Volodya sold 0-Day exploits at prices ranging from \$85,000 (exploit from 2016) up to \$200,000. While we don't know what the selling price was for 1-Day exploits, we expected prices roughly in the same neighborhood. However, when going over the different ads published by PlayBit, we saw a wide pricing gap when compared to Volodya.

All Windows LPE exploits were advertised with a price tag ranging between \$5,000 and \$10,000. We even found the actor using a fake nickname and claiming to sell a 0-Day exploit, which was in fact the 1-Day exploit for CVE-2016-7255. The asking price for this "0-Day" was still \$35,000, way below the \$85,000 for which Volodya sold the real 0-Day exploit of the same vulnerability. In addition, we couldn't find any pricing trend for the different exploits, as all pretty much sold for the same price from 2015 to 2020.

Intelligence Report Wrap-up

Although not intuitive at first, the fact that we didn't see ads for other CVEs (of the same exploitation template) is actually good news. It means that our technique worked better than expected: A fully technical hunt for PlayBit's exploits, without any intelligence, still managed to cover all of their exploits that originated from the exploit template with which we initially started.

Aside from Windows LPEs, we found two more interesting tools that PlayBit is developing/collaborating with.

Avatar Rootkit

- ESET mentioned this in [2013](#).
- Uses the same hash-based imports that is still a feature of PlayBit today:
`get_module_by_hash()` / `get_func_by_hash()` .
- This [Pastebin ad](#) specifically mentions PlayBit as an associate.

EternalBlack

- Self-implemented [EternalRomance](#)
- Again, uses features seen in all of the rest of the samples.

```
#####  
#                                     #  
#           ETERNALBLACK             #  
#           Version 1.0.1820         #  
#                                     #  
#   Using: eb.exe <target_address> <payload32.exe> [<payload64.exe>] #  
#                                     #  
#####  
#           coded by PlayBit (c) 2017 #  
#           xmpp: playbit@exploit.im, playbit@hacklab.li #  
#####
```

Figure 13: Message printed when executing EternalBlack, developed by PlayBit.

The Customers

The “customers” i.e. malware who use PlayBit's exploit, either directly or by using an exploit framework, are all crimeware. Most prominent are the popular ransomware that use PlayBit's exploits to escalate their privileges before encrypting the victim's disk. These ransomware include Maze, Locky, LockCrypt, and REvil (Sodin, Sodinokibi). Other malware are popular Trojans like Ramnit and Dyre.

Conclusion

In this article, we demonstrated another case in which we were able to fingerprint an exploit developer, without prior knowledge about the developer or any public profiles. All we started with was a single sample. We showed how PlayBit, similar to [Volodya](#), has a unique set of choices, approaches, methodologies and combinations of implementation decisions. By gathering all the pieces, we managed to understand and profile PlayBit, as well as attribute samples to the actor. We also took the opportunity to compare PlayBit to Volodya, and highlight the differences between their coding styles and preferences.

Aside from the technical aspects, this is the first time that PlayBit has been thoroughly described by researchers. We took a look at the exploit market, the advertisements, YouTube channels and collaborations between exploit developers and malware authors. Developing an exploit is just the beginning. The next step is to monetize the “product” and sell the customers a high-quality piece of software that is relatively stable and supports as many versions as possible.

Following our success with profiling both PlayBit and Volodya, we believe that our research methodology can be used to identify additional exploit writers as well. We therefore recommend that other researchers try our approach and add it to their toolbox.

Recommendation for Protection

Check Point [Threat Emulation](#) provides protection against this threat:

Wins.Generic.G

Appendix – IOC Table

CVE-2013-3660:

9f1a235eb38291cef296829be4b4d03618cd21e0b4f343f75a460c31a0ad62d3

CVE-2015-0057:

1b3524fd57e4e836778d4af4579b6d986e7475ee6a1a7818ead0fc59efbdc2ac

CVE-2015-1701 (Also contains an exploit for CVE-2015-0057):

8869e0df9b5f4a894216c76aa5689686395c16296761716abece00a0b4234d87

CVE-2016-7255:

5c27e05b788ba3b997a70df674d410322c3fa5e97079a7bf3aec369a0d397164

CVE-2018-8453:

50da0183466a9852590de0d9e58bbe64f22ff8fc20a9ccc68ed0e50b367d7043

Avatar Rootkit:

d1a8d74aadb10bff4bfda144e68db3e087ec4fee82cd22df22839fd5435d0d37

EternalBlack:

effa8e38838ba7d2c58b2731c086ac72d639f9d2ab8184bc8cf05d72c5444dd1