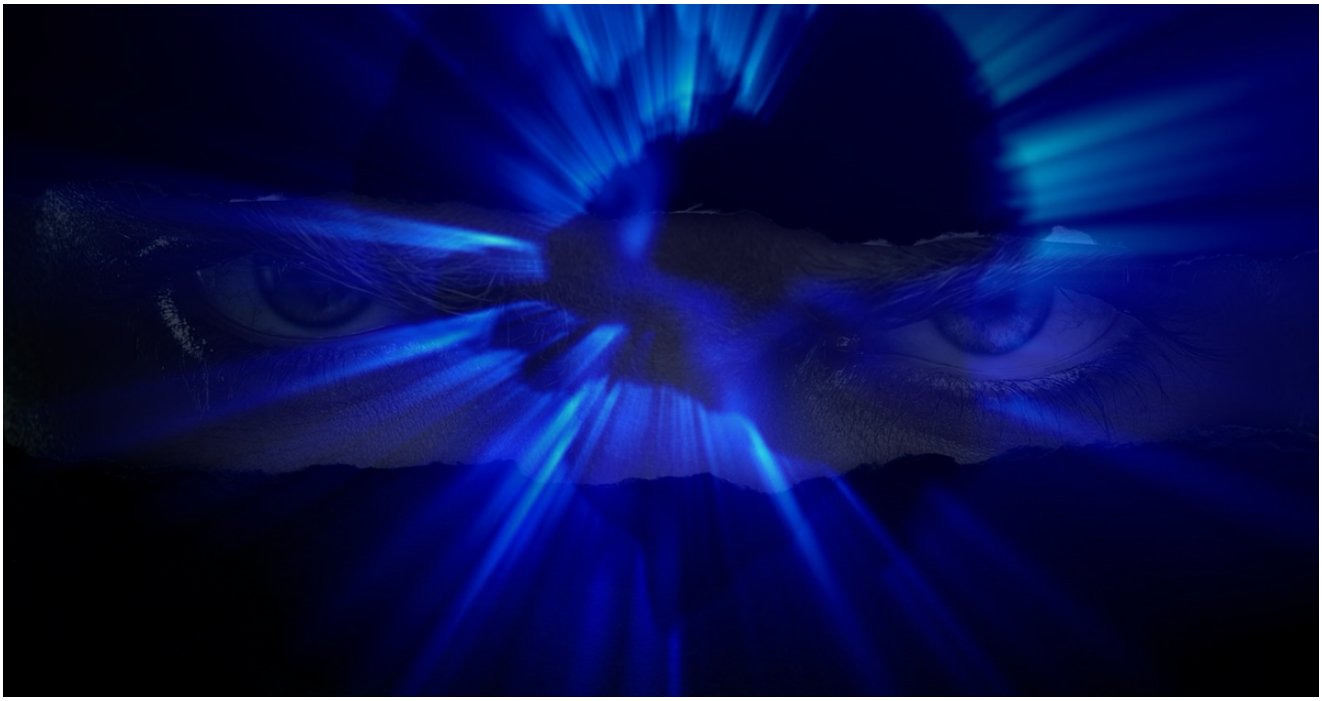
Olympic Destroyer: A new Candidate in South Korea

lastline.com/labsblog/olympic-destroyer-south-korea/

February 21, 2018



Posted by Alexander Sevtsov and Stefano Ortolani ON FEB 21, 2018
Authored by: Alexander Sevtsov
Edited by: Stefano Ortolani

A new malware has recently made the headlines, targeting several computers during the opening ceremony of the Olympic Games Pyeongchang 2018. While Cisco Talos group, and later Endgame, have recently covered it, we noticed a couple of interesting aspects not previously addressed, we would like to share: its taste for hiding its traces, and the peculiar decryption routine. We also would like to pay attention on how the threat makes use of multiple components to breach the infected system. This knowledge allows us to improve our sandbox to be even more effective against emerging advanced threats, so we would like to share some of them.

The Olympic Destroyer

The malware is responsible for destroying (wiping out) files on network shares, making infected machines irrecoverable, and propagating itself with the newly harvested credentials across compromised networks.

To achieve this, the main executable file (sha1: 26de43cc558a4e0e60eddd4dc9321bcb5a0a181c) drops and runs the following components, all originally encrypted and embedded in the resource section:

- a browsers credential stealer (sha1: 492d4a4a74099074e26b5dffd0d15434009ccfd9),
- a system credential stealer (a Mimikatz-like DLL – sha1: ed1cd9e086923797fe2e5fe8ff19685bd2a40072 (for 64-bit OS), sha1: 21ca710ed3bc536bd5394f0bff6d6140809156cf (for 32-bit OS)),
- a wiper component (sha1: 8350e06f52e5c660bb416b03edb6a5ddc50c3a59).
- a legitimate signed copy of the PsExec utility used for the lateral movement (sha1: e50d9e3bd91908e13a26b3e23edeaf577fb3a095)

A wiper deleting data and logs

The wiper component is responsible for wiping the data from the network shares, but also destroying the attacked system by deleting backups, disabling services (Figure 1), clearing event logs using wevtutil, thereby making the infected machine unusable. The very similar behaviors have been previously observed in other Ransomware/Wiper attacks, including the infamous ones such as BadRabbit and NotPetya.

```
result = OpenSCManagerW_0(0, L"ServicesActive", 0x80000000);
hSCManager = result;
if ( result )
{
  v11 = a1;
  v2 = *EnumServicesStatusW;
  pcbBytesNeeded = 0;
  ServicesReturned = 0;
  ResumeHandle = 0;
  EnumServicesStatusW(result, 0x13Fu, 3u, 0, 0, &pcbBytesNeeded, &ServicesReturned, &ResumeHandle);
  v3 = *GetProcessHeap_0;
  v4 = GetProcessHeap_0();
  v13 = RtlAllocateHeap(v4);
  if ( v2(hSCManager, 0x13Fu, 3u, v13, pcbBytesNeeded, &pcbBytesNeeded, &ServicesReturned, &ResumeHandle) )
  {
    v16 = 0;
    if ( ServicesReturned > 0 )
    {
      v5 = *QueryServiceConfigW_0;
      v17 = v13;
      do
      {
        v6 = OpenServiceW_0(hSCManager, *v17, 0x10000000u);
        hService = v6;
        if ( v6 )
        {
          cbBufSize = 0;
          v5(v6, 0, 0, &cbBufSize);
          v7 = v3();
          lpServiceConfig = RtlAllocateHeap(v7);
          ChangeServiceConfigW_0(hService, 0xFFFFFFFF, SERVICE_DISABLED, 0xFFFFFFFF, 0, 0, 0, 0, 0, 0, 0);
          if ( (v5)(hService, lpServiceConfig, cbBufSize, &cbBufSize) )
            PathRemoveArgsW(lpServiceConfig->lpBinaryPathName);
          v8 = lpServiceConfig;
          v9 = v3();
          HeapFree_0(v9, v8, v12);
          GetLastError_0();
          CloseServiceHandle_0(hService);
        }
        ++v16;
        v17 += 9;
      }
      while ( v16 < ServicesReturned );
    }
    v10 = v3();
    HeapFree_0(v10, v11, v12);
  }
  result = CloseServiceHandle_0(hSCManager);
}
return result;
```

Figure 1. Disabling Windows services

After wiping the files, the malicious component sleeps for an hour (probably, to be sure that the spawned thread managed to finish its job), and calls the InitiateSystemShutdownExW API with the system failure reason code (SHTDN_REASON_MAJOR_SYSTEM, 0x00050000) to shut down the system.

An unusual decryption to extract the resources

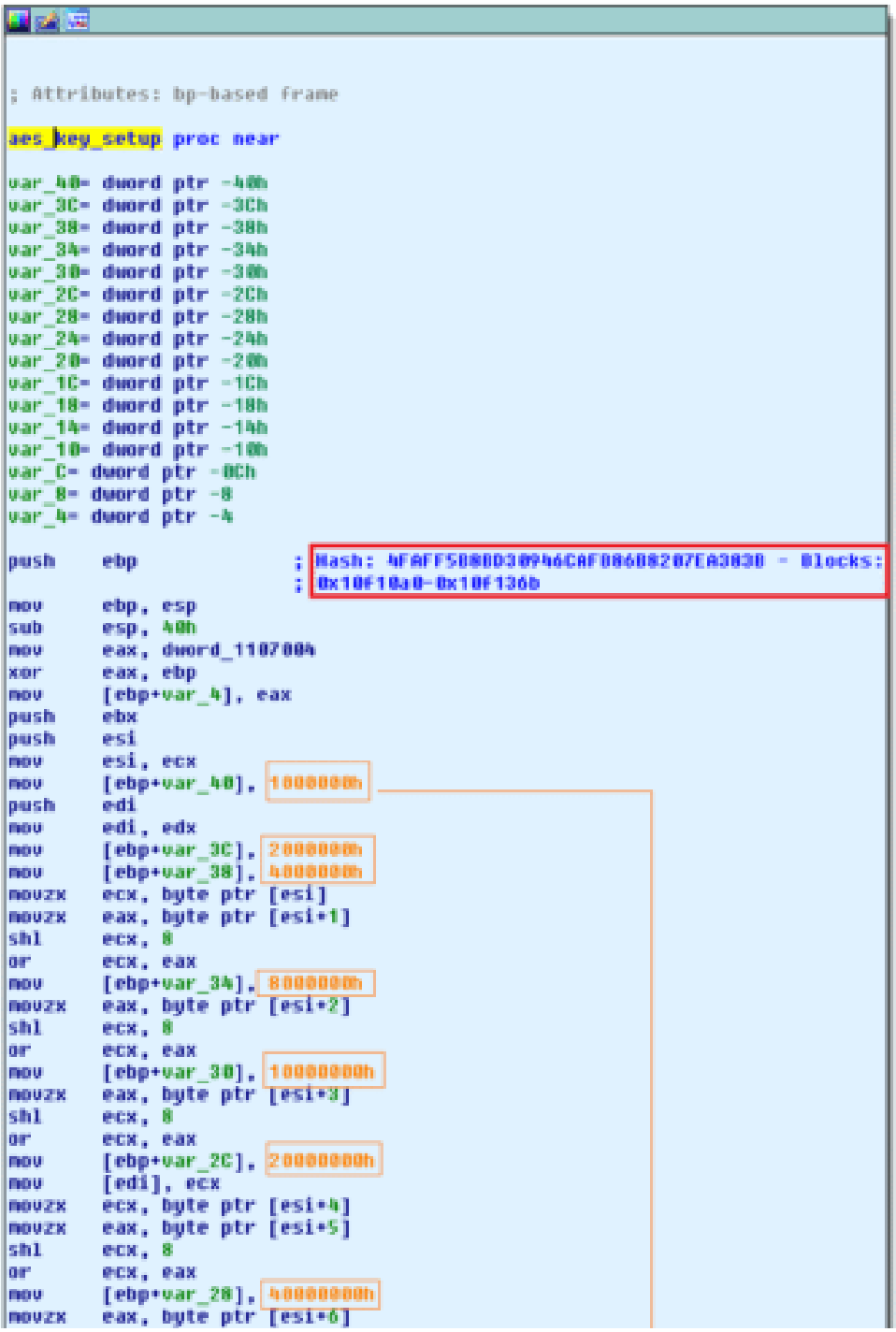
As mentioned before, the executables are stored encrypted inside the binary's resource section. This is to prevent static extraction of the embedded files, thus slowing down the analysis process. Another reason of going "offline" (compared with e.g. the Smoke Loader) is to bypass any network-based security solutions (which, in turn, decreases the probability of detection). When the malware executes, they are loaded via the LoadResource API, and decrypted via the MMX/SSE instructions sometimes used by malware to bypass code emulation, this is what we've observed while debugging it. In this case, however, the instructions are used to implement AES encryption and MD5 hash function (instead of using

standard Windows APIs, such as CryptEncrypt and CryptCreateHash) to decrypt the resources. The MD5 algorithm is used to generate the symmetric key, which is equal to MD5 of a hardcoded string “123”, and multiplied by 2.

The algorithms could be also identified by looking at some characteristic constants of

1. The Rcon array used during the AES key schedule (see figure 2) and,
2. The MD5 magic initialization constants.

The decrypted resources are then dropped in temporary directory and finally, executed.



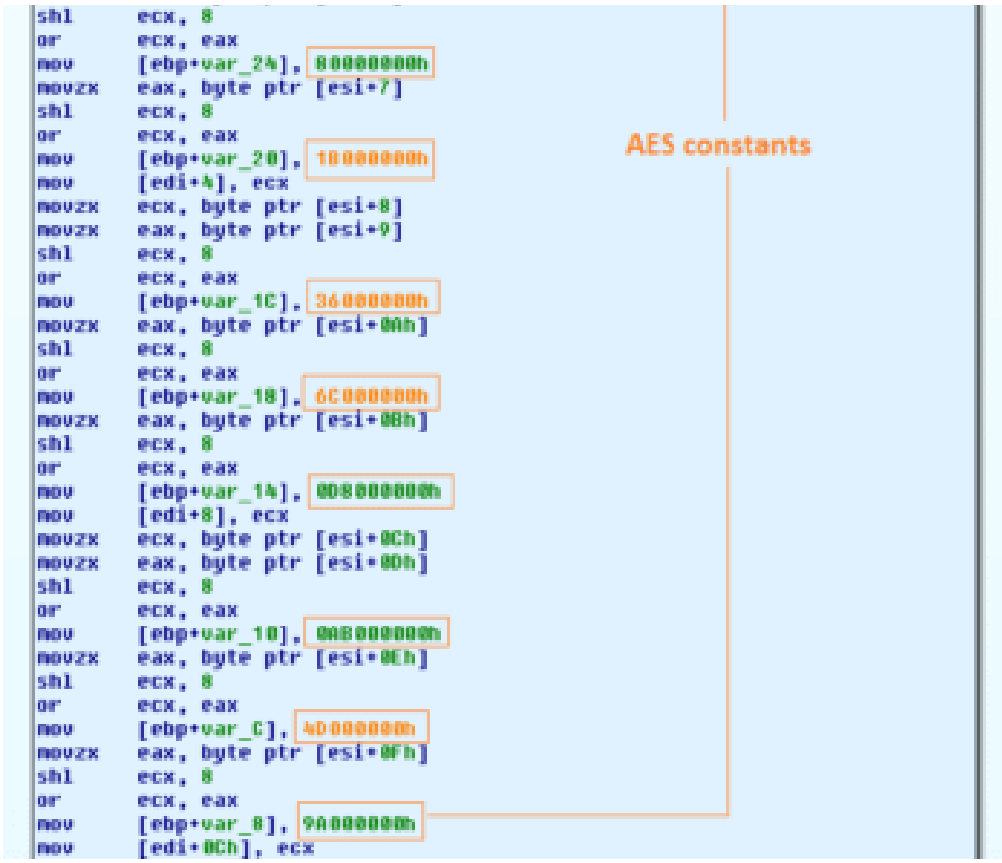


Figure 2. AES key setup routine for resources decryption

Hunting

An interesting aspect of the decryption is its usage of the SSE instructions. We exploited this peculiarity and hunted for other samples sharing the same code by searching for the associated codehash, for example. The later is a normalized representation of the code mnemonics included in the function block (see Figure 3) as produced by the Lastline sandbox, and exported as a part of the process snapshots).

Another interesting sample found during our investigation was (sha1: 84aa2651258a702434233a946336b1adf1584c49) with the harvested system credentials belonging to the Atos company, a technical provider of the Pyeongchang games (see here for more details).

Figure 3. Hardcoded credentials of an Olympic Destroyer targeted the ATOS company

A Shellcode Injection Wiping the Injector

Another peculiarity of the Olympic Destroyer is how it deletes itself after execution. While self-deletion is a common practice among malware, it is quite uncommon to see the injected shellcode taking care of it: the shellcode, once injected in a legitimate copy of notepad.exe, waits until the sample terminates, and then deletes it.

```
shellcode:                              ; DATA XREF: sub_405CB0+E↑o
                                        ; sub_405CB0+1B↑o
                mov     eax, 0DEADBEEFh
                mov     ebp, esp

loop:                                   ; CODE XREF: sub_405CB0+93↓j
                                        ; sub_405CB0+FB↓j
                mov     eax, [ebp+4]
                mov     ebx, [eax+24h]
                mov     eax, [eax]
                push    ebx
                call    eax             ; Sleep
                nop
                nop
                nop
                mov     eax, [ebp+4]
                mov     ebx, eax
                add     ebx, 28h
                mov     eax, [eax+0Ch]
                push    ebx
                call    eax             ; GetFileAttributesW
                cmp     eax, INVALID_FILE_ATTRIBUTES
                jz      end
                mov     ebx, eax
                and     eax, 10h
                jnz     end
                and     ebx, 400h
                jnz     end
                nop
                nop
                nop
                mov     eax, [ebp+4]
                mov     ebx, eax
                add     ebx, 28h
                mov     eax, [eax+10h]
                push    NULL
                push    0
                push    OPEN_EXISTING
                push    NULL
                push    3
                push    0C0000000h      ; Access = GENERIC_READ|GENERIC_WRITE
                push    ebx
                call    eax             ; CreateFileW
                cmp     eax, INVALID_HANDLE_VALUE
                jz      short loop
```

Figure 4. Checking whether the file is terminated or still running

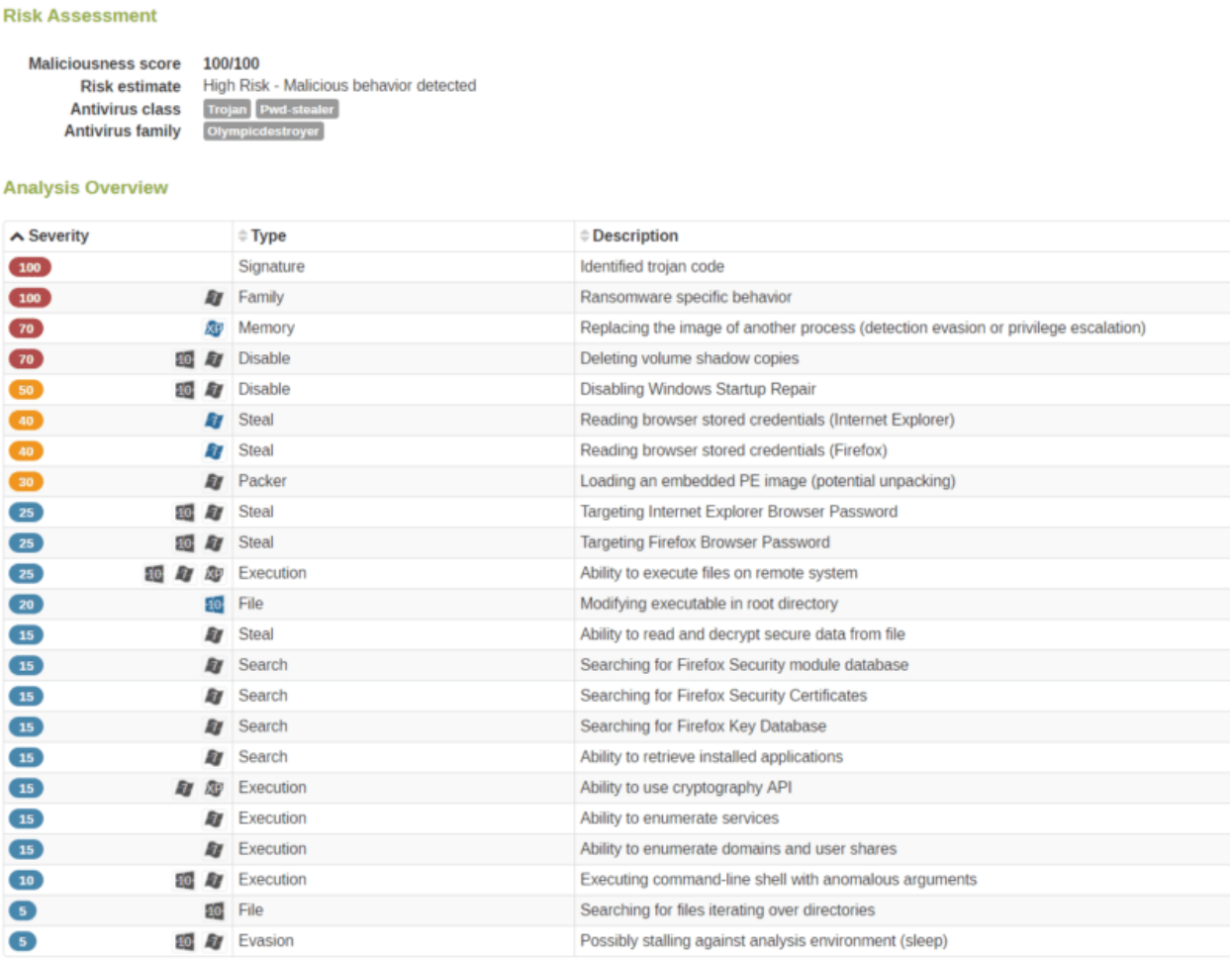
This is done first by calling CreateFileW API and checking whether the sample is still running (as shown in Figure 4); it then overwrites the file with a sequence of 0x00 byte, deletes it via DeleteFileW API, and finally exits the process.

The remainder of the injection process is very common and it is similar to what we have described in one of our previous blog posts: the malware first spawns a copy of notepad.exe by calling the CreateProcessW function; then allocates memory in the process by calling VirtualAllocEx, and writes shellcode in the allocated memory through WriteProcessMemory. Finally, it creates a remote thread for its execution via CreateRemoteThread.

Figure 5. Shellcode injection in a copy of notepad.exe

Lastline Analysis Overview

Figure 6 shows how the analysis overview looks like when analyzing the sample discussed in this article:



Risk Assessment

Maliciousness score	100/100
Risk estimate	High Risk - Malicious behavior detected
Antivirus class	Trojan, Pwd-stealer
Antivirus family	Olympicdestroyer

Analysis Overview

Severity	Type	Description
100	Signature	Identified trojan code
100	Family	Ransomware specific behavior
70	Memory	Replacing the image of another process (detection evasion or privilege escalation)
70	Disable	Deleting volume shadow copies
50	Disable	Disabling Windows Startup Repair
40	Steal	Reading browser stored credentials (Internet Explorer)
40	Steal	Reading browser stored credentials (Firefox)
30	Packer	Loading an embedded PE image (potential unpacking)
25	Steal	Targeting Internet Explorer Browser Password
25	Steal	Targeting Firefox Browser Password
25	Execution	Ability to execute files on remote system
20	File	Modifying executable in root directory
15	Steal	Ability to read and decrypt secure data from file
15	Search	Searching for Firefox Security module database
15	Search	Searching for Firefox Security Certificates
15	Search	Searching for Firefox Key Database
15	Search	Ability to retrieve installed applications
15	Execution	Ability to use cryptography API
15	Execution	Ability to enumerate services
15	Execution	Ability to enumerate domains and user shares
10	Execution	Executing command-line shell with anomalous arguments
5	File	Searching for files iterating over directories
5	Evasion	Possibly stalling against analysis environment (sleep)

Figure 6. Analysis overview of the Olympic Destroyer

Conclusion

In this article, we analyzed a variant of the Olympic Destroyer, a multi-component malware that steals credentials before making the targeted machines unusable by wiping out data on the network shares, and deleting backups. Additionally, the effort put into deleting its traces shows a deliberate attempt to hinder any forensic activity. We also have shown how Lastline found similar samples related to this attack based on an example of the decryption routine, and how we detect them. This is a perfect example of how the threats are continuously improving making them even stealthier, more difficult to extract and analyze.

Appendix: IoCs

Olympic Destroyer
26de43cc558a4e0e60eddd4dc9321bcb5a0a181c (sample analyzed in this article)
21ca710ed3bc536bd5394f0bff6d6140809156cf
492d4a4a74099074e26b5dffd0d15434009ccfd9
84aa2651258a702434233a946336b1adf1584c49
b410bbb43dad0aad024ec4f77cf911459e7f3d97
c5e68dc3761aa47f311dd29306e2f527560795e1
c9da39310d8d32d6d477970864009cb4a080eb2c
fb07496900468529719f07ed4b7432ece97a8d3d

Alexander Sevtsov

Alexander Sevtsov is a Malware Reverse Engineer at Lastline. Prior to joining Lastline, he worked for Kaspersky Lab, Avira and Huawei, focusing on different methods of automatic malware detection. His research interests are modern evasion techniques and deep document analysis.

Latest posts by Alexander Sevtsov (see all)

- Evasive Monero Miners: Deserting the Sandbox for Profit - June 20, 2018
- I Hash You: A Simple But Effective Trick to Evade Dynamic Analysis - April 10, 2018
- Olympic Destroyer: A new Candidate in South Korea - February 21, 2018

- About
- Latest Posts

Stefano Ortolani

Stefano Ortolani is Director of Threat Intelligence at Lastline. Prior to that he was part of the research team in Kaspersky Lab in charge of fostering operations with CERTs, governments, universities, and law enforcement agencies. Before that he earned his Ph.D. in Computer Science from the VU University Amsterdam.

Latest posts by Stefano Ortolani (see all)

- Evolution of Excel 4.0 Macro Weaponization - June 2, 2020
- InfoStealers Weaponizing COVID-19 - May 11, 2020
- Nemty Ransomware Scaling UP: APAC Mailboxes Swarmed by Dual Downloaders - February 18, 2020

Tags:

Advanced Malware, browser stealer, code similarity, codehash, Lastline Sandbox, Mimikatz, Olympic Destroyer, PsExec, Ransomware, shellcode injection, Wiper