

Quick-Analysis/SmokeLoader/SmokeLoader.md at main · vc0RExor/Quick-Analysis

By vc0RExor

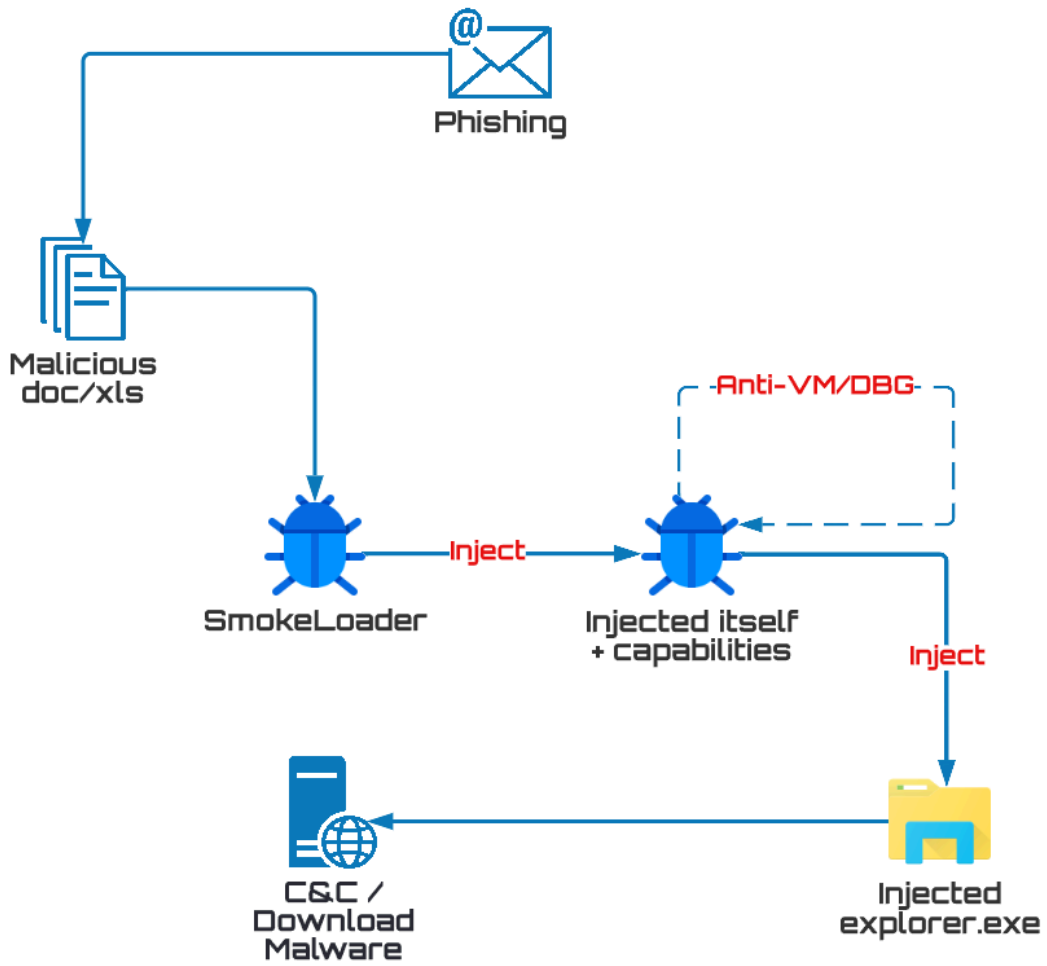
Archived: 2026-04-05 21:22:45 UTC

Overview

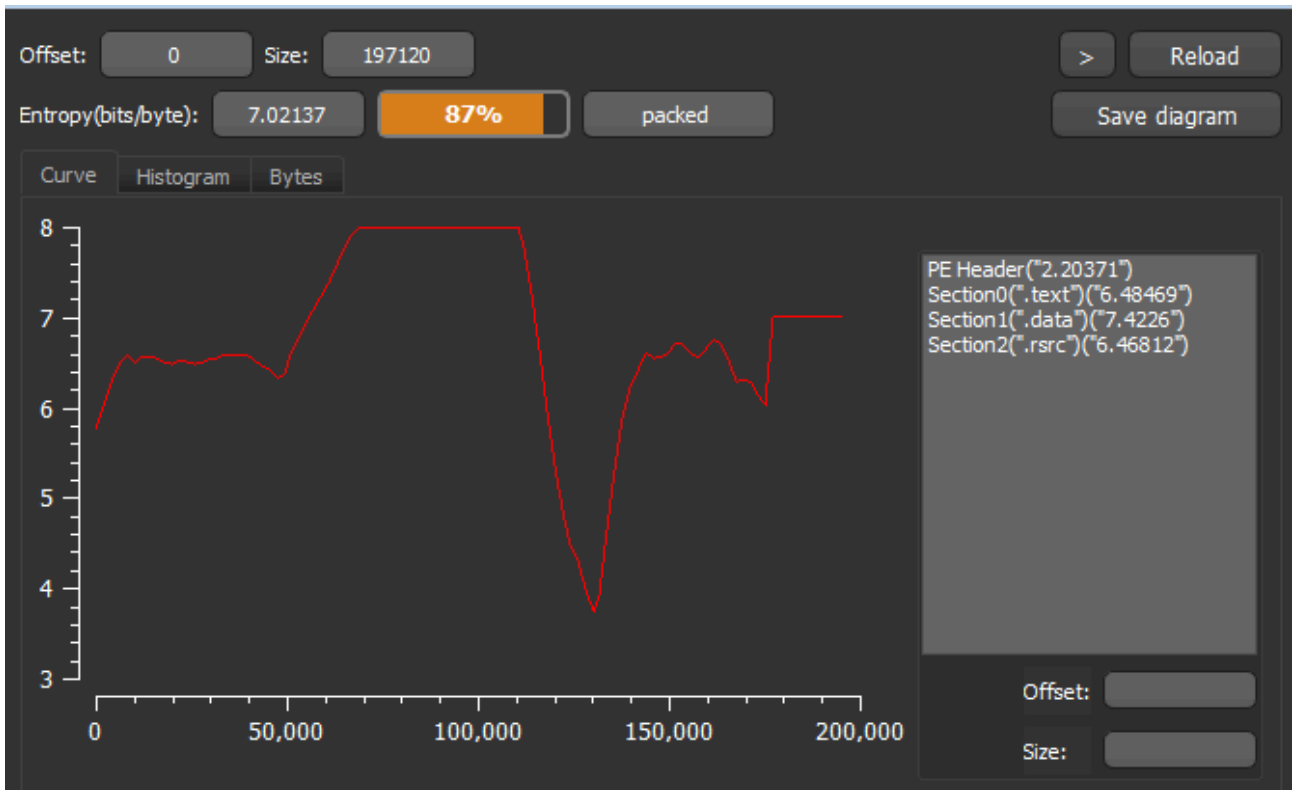
SmokeLoader is a malware that generally acts as a backdoor and is commonly used as a loader for other malware. Attributed to the criminal group Smoky Spider, a group that uses SmokeLoader and Sasfis, loader and downloader respectively. SmokeLoader has been used as a bot in infrastructures and contains strong evasion capabilities as well as Anti-Analysis, Anti-VM and Anti-DBG techniques.

Technical Analysis

SmokeLoader appears on systems usually through phishing, although it can be loaded by other PUP/PUA or malware. The main execution will revolve around a document that will spawn the SmokeLoader which will run, in most of its versions, a version of itself in a suspended state to inject code, after which it will execute an *explorer.exe* that it will inject again in order to perform the malicious C&C actions or download other files using legitimate software.



The samples that have been found have in most cases been detected as packed, due to the high level of entropy contained in their sections.



At the initial point, we see how it tries to load libraries in RunTime, something really useful since it prevents us from being able to discern its intentions if we perform a basic static analysis, so it will obtain new functionalities during its execution.

Dll runtime load

```

sub_403910 proc near
var_8= dword ptr -8
var_4= byte ptr -4

sub     esp, 8
push   offset dword_4790E0 ; lpLibFileName
call   ds:LoadLibraryM
push   offset ProcName ; lpProcName
push   eax ; hModule
mov    dword_4790C8, eax
mov    word_4233C0, 74h
mov    ProcName, 74726956h
mov    dword_4233C4, 506C6175h
mov    dword_4233C8, 65746F72h
mov    byte_4233CC, 63h
call   ds:GetProcAddress
mov    dword_4768C8, eax
mov    [esp+var_8], 20h
add    [esp+var_8], 20h
mov    ecx, [esp+var_8]
mov    edx, dwBytes
lea   eax, [esp+var_4]
push  eax
mov    eax, dword_4768E0
push  ecx
push  edx
push  eax
call  dword_4768C8
add   esp, 8
retn
sub_403910 endp
    
```

00403913	68 E904700	push	6b48d5999d04db64c7f91fa311bfff6caee938dd50095a7a5fb7f2229	4790E0:L"kernel32.dll"
00403918	FF15 30104000	call	dword ptr ds:[4&LoadLibraryM]	
00403921	66 C034200	push	6b48d5999d04db64c7f91fa311bfff6caee938dd50095a7a5fb7f2229	
00403923	50	push	eax	
00403924	A3 C8904700	mov	dword ptr ds:[4790C8],eax	
00403929	66C705 C0334200 740	mov	dword ptr ds:[4233C0],74	74:'t'
00403932	C705 C0334200 566972	mov	dword ptr ds:[4233C4],74726956	
0040393C	C705 C4334200 75616C	mov	dword ptr ds:[4233C8],506C6175	
00403946	C705 C8334200 726F74	mov	dword ptr ds:[4233CC],65746F72	
00403950	C605 C0334200 63	mov	byte ptr ds:[4233CC],63	63:'c'
00403957	FF15 58104000	call	dword ptr ds:[6&GetProcAddress]	
0040395D	A3 C8684700	mov	dword ptr ds:[4768C8],eax	
00403962	C70424 20000000	mov	dword ptr ss:[esp],20	[esp]:L"kernel32.dll"
00403969	830424 20	add	dword ptr ss:[esp],20	[esp]:L"kernel32.dll"
0040396D	8B0C24	mov	ecx,dword ptr ss:[esp]	[esp]:L"kernel32.dll"
00403970	8B15 DC904700	mov	edx,dword ptr ds:[4790C1]	
00403976	8D4424 04	lea	eax,dword ptr ss:[esp+4]	
0040397A	50	push	eax	

004038B5	FF15 84104000	call	dword ptr ds:[6&GetProcAddress]	
004038B6	A3 E0684700	mov	dword ptr ds:[4768C8],eax	
004038B8	E8 B5FDFFFF	call	6b48d5999d04db64c7f91fa311bfff6caee938dd50095a7a5fb7f2229	

004034F5	0D C9334200	or	eax,6b48d5999d04db64c7f91fa311bfff6caee938dd50095a7a5fb7f2229	4233C9:"11"
004034FA	66C705 C1334200 736	mov	word ptr ds:[4233C1],6973	004233C1:"smg32.dll"
00403503	A2 C0334200	mov	byte ptr ds:[4233C3],a1	004233C0:"msmg32.dll"
00403508	C705 C3334200 606733	mov	dword ptr ds:[4233C3],3233676D	004233C3:"mg32.dll"
00403512	66C705 C4334200 6C0	mov	word ptr ds:[4233C4],6C	6C:'l'
0040351B	66C705 C7334200 2E6	mov	word ptr ds:[4233C7],642E	004233C7:".dll"
00403524	C3	ret		

In some of the techniques used to hinder the analysis, such as code obfuscation, we find different hidden calls, as well as abuses of RET to reach calls that we will not see statically.

Hidden calls

```
EIP EAX → 005D8D9  E8 01000000  call 5D8DF
             005D8DE  C3          ret
             005D8DF  55          push ebp
             005D8E0  8BEC       mov ebp,esp
             005D8E2  8D45 C4    lea eax,dword ptr ss:[ebp-3C]
             005D8E5  83EC 3C    sub esp,3C
             005D8E8  50          push eax
             005D8E9  E8 0D000000 call 5D8FB
```



```
EIP EAX → 005D8D7  4D          dec ebp
             005D8D8  7A E8      jp 5D8C2
             005D8DA  0100      add dword ptr ds:[eax],eax
             005D8DC  0000      add byte ptr ds:[eax],al
             005D8DE  C3          ret
             005D8DF  55          push ebp

EAX → 005D8D7  4D          dec ebp
             005D8D8  7A E8      jp 5D8C2
             005D8DA  0100      add dword ptr ds:[eax],eax
             005D8DC  0000      add byte ptr ds:[eax],al
             005D8DE  C3          ret
EIP → 005D8DF  55          push ebp
             005D8E0  8BEC       mov ebp,esp
             005D8E2  8D45 C4    lea eax,dword ptr ss:[ebp-3C]
             005D8E5  83EC 3C    sub esp,3C
             005D8E8  50          push eax
             005D8E9  E8 0D000000 call 5D8FB
```

As mentioned above, it fetches libraries during runtime and is dedicated to resolving APIs that it could use later on

Resolving APIs

0050D9A9	8B32	mov esi,dword ptr ds:[edx]	esi:"AcquireSRWLockShared", edx:"c\\f"
0050D9AB	58	pop eax	
0050D9AC	50	push eax	
0050D9AD	03F0	add esi,eax	esi:"AcquireSRWLockShared"
0050D9AF	6A 01	push 1	
0050D9B1	FF75 0C	push dword ptr ss:[ebp+C]	
0050D9B4	56	push esi	esi:"AcquireSRWLockShared"
0050D9B5	E8 23000000	call 50D9D0	
0050D9BA	85C0	test eax,eax	
0050D9BC	74 08	je 50D9C6	
0050D9BE	83C2 04	add edx,4	edx:"c\\f"
0050D9C1	83C3 02	add ebx,2	
0050D9C4	EB E3	jmp 50D9A9	
0050D9C6	58	pop eax	

0050D9A9	8B32	mov esi,dword ptr ds:[edx]	esi:"ActivateActCtx", edx:"x\\f"
0050D9AB	58	pop eax	
0050D9AC	50	push eax	
0050D9AD	03F0	add esi,eax	esi:"ActivateActCtx"
0050D9AF	6A 01	push 1	
0050D9B1	FF75 0C	push dword ptr ss:[ebp+C]	
0050D9B4	56	push esi	esi:"ActivateActCtx"
0050D9B5	E8 23000000	call 50D9D0	
0050D9BA	85C0	test eax,eax	
0050D9BC	74 08	je 50D9C6	
0050D9BE	83C2 04	add edx,4	edx:"x\\f"
0050D9C1	83C3 02	add ebx,2	
0050D9C4	EB E3	jmp 50D9A9	
0050D9C6	58	pop eax	

```

75002F48 6C 6C 00 41 63 71 75 69 72 65 53 52 57 4C 6F 63 11.AcquireSRWLoc
75002F58 68 45 78 63 6C 75 73 69 76 65 00 41 63 71 75 69 kExclusive.Acqui
75002F68 72 65 53 52 57 4C 6F 63 68 53 68 61 72 65 64 00 reSRWLockShared.
75002F78 41 63 74 69 76 61 74 65 41 63 74 43 74 78 00 41 ctivateActCtx.A
75002F88 64 64 41 74 6F 60 41 00 41 64 64 41 74 6F 60 57 ddAtomA.AddAtomW
75002F98 00 41 64 64 43 6F 6E 73 6F 6C 65 41 6C 69 61 73 .AddConsoleAlias
75002FAB 41 00 41 64 64 43 6F 6E 73 6F 6C 65 41 6C 69 61 A.AddConsoleAlia
75002FB8 73 57 00 41 64 64 49 6E 74 65 67 72 69 74 79 4C sw.AddIntegrityL
75002FC8 61 62 65 6C 54 6F 42 6F 75 6E 64 61 72 79 44 65 abelToBoundaryDe
75002FD8 73 63 72 69 70 74 6F 72 00 41 64 64 4C 6F 63 61 scriptor.AddLoca
75002FE8 6C 41 6C 74 65 72 6E 61 74 65 43 6F 6D 70 75 74 lAlternateComput
75002FF8 65 72 4E 61 6D 65 57 00 41 64 64 4C 6F 63 61 6C erNameA.AddLocal
75003008 41 6C 74 65 72 6E 61 74 65 43 6F 6D 70 75 74 65 lternateCompute
75003018 72 4E 61 6D 65 57 00 41 64 64 52 65 66 41 63 74 rNameW.AddRefAct
75003028 43 74 78 00 41 64 64 53 49 44 54 6F 42 6F 75 6E Ctx.AddSIDToBoun
75003038 64 61 72 79 44 65 73 63 72 69 70 74 6F 72 00 41 daryDescriptor.A
75003048 64 64 53 65 63 75 72 65 4D 65 6D 6F 72 79 43 61 ddSecurMemoryCa
75003058 63 68 65 43 61 6C 6C 62 61 63 68 00 41 64 64 56 cheCallback.AddV
75003068 65 63 74 6F 72 65 64 43 6F 6E 74 69 6E 75 65 48 ectorContinueH
75003078 61 6E 64 6C 65 72 00 41 64 64 56 65 63 74 6F 72 andler.AddVecto
75003088 65 64 45 78 63 65 70 74 69 6F 6E 48 61 6E 64 6C edExceptionHandl
75003098 65 72 00 41 64 6A 75 73 74 43 61 6C 65 6E 64 61 er.AdjustCalenda
750030A8 72 44 61 74 65 00 41 6C 6C 6F 63 43 6F 6E 73 6F rDate.AllocConso
    
```

At all times, it has control over what is running on the machine, as it subsequently performs various Anti-Vm and Anti-dbg techniques, so having all running processes mapped is always a good technique.

Control of running processes

```
74F7735F 6A 40          push 40
74F77361 68 D874F774   push kernel32.74F774D8
74F77366 E8 85A2FDFF   call kernel32.74F515F0
74F7736B 834D D0 FF    or dword ptr ss:[ebp-30],FFFFFFFF
74F7736F 834D CC FF    or dword ptr ss:[ebp-34],FFFFFFFF
74F77373 33F6         xor esi,esi
74F77375 8975 E4       mov dword ptr ss:[ebp-1C],esi
74F77378 8975 D8       mov dword ptr ss:[ebp-28],esi
74F7737B 8975 D0       mov dword ptr ss:[ebp-20],esi
74F7737E 8975 DC       mov dword ptr ss:[ebp-24],esi
74F77381 8975 C0       mov dword ptr ss:[ebp-40],esi
74F77384 8975 C8       mov dword ptr ss:[ebp-38],esi
74F77387 3975 0C       cmp dword ptr ss:[ebp+C],esi

0050E0C6 57          push edi
0050E0C7 FF56 2C      call dword ptr ds:[edi+2C]
0050E0CA 85C0        test eax,eax
0050E0CC 74 07       je 50E0D5
0050E0CE 56          push esi
0050E0CF E8 6AFCFFFF  call 50D03E
0050E0D4 59          pop ecx
0050E0D5 57          push edi
0050E0D6 FF56 30     call dword ptr ds:[edi+30]
0050E0D9 5F          pop edi
0050E0DA 5E          pop esi
0050E0DB 5B          pop ebx
0050E0DC C9          leave
0050E0DD C3          ret
0050E0DE D873 00     fdiv st(0),dword ptr ds:[edi+0]
0050E0E1 0001       add byte ptr ds:[ecx],a
0050E0E3 0000       add byte ptr ds:[eax],a
0050E0E5 8001 A0     add byte ptr ds:[ecx],A
0050E0E8 8900       mov dword ptr ds:[eax],a
0050E0EA 0000       add byte ptr ds:[eax],a
0050E0EC 0000       add byte ptr ds:[eax],a
0050E0EE 0000       add byte ptr ds:[eax],a
0050E0F0 0000       add byte ptr ds:[eax],a
0050E0F2 0000       add byte ptr ds:[eax],a
0050E0F4 0000       add byte ptr ds:[eax],a
```

After this, it starts loading APIs that will serve it moments later, in which we will see a routine that will be loading from memory and using LoadLibrary + GetProcAddress


```

ExitProcess
SendMessageExtraInfo
WaitForSingleObject
NtUnmapViewOfSection
MessageBoxA
ReadProcessMemory
GetThreadContext
WriteFile
GetModuleFileNameA
GetFileAttributesA
WinExec
SendMessageA
    
```

Once it has the libraries, APIs and processes controlled, it creates a process in suspended state, for this it uses `CreateProcessInternalA` that will call `CreateProcessInternalW` entering `0x04` in `dwCreationFlags` to create the process in suspended state.

CREATE_SUSPENDED
 0x00000004
 The primary thread of the new process is created in a suspended state, and does not run until the `ResumeThread` function is called.

→	74F51072	8BFF	mov edi,edi		CreateProcessA
●	74F51074	55	push ebp		
●	74F51075	8BEC	mov ebp,esp		
●	74F51077	6A 00	push 0		
●	74F51079	FF75 2C	push dword ptr ss:[ebp+2C]		
●	74F5107C	FF75 28	push dword ptr ss:[ebp+28]		
●	74F5107F	FF75 24	push dword ptr ss:[ebp+24]		
●	74F51082	FF75 20	push dword ptr ss:[ebp+20]		
●	74F51085	FF75 1C	push dword ptr ss:[ebp+1C]		
●	74F51088	FF75 18	push dword ptr ss:[ebp+18]		
●	74F5108B	FF75 14	push dword ptr ss:[ebp+14]		
●	74F5108E	FF75 10	push dword ptr ss:[ebp+10]		
●	74F51091	FF75 0C	push dword ptr ss:[ebp+C]		
●	74F51094	FF75 08	push dword ptr ss:[ebp+8]		
●	74F51097	6A 00	push 0		
●	74F51099	E8 19940100	call <kernel32.CreateProcessInternalA>		
●	74F5109E	5D	pop ebp		
●	74F5109F	C2 2800	ret 28		

☰	6b48d5999d04db6b4c791fa311...	0.02	1,332 K	4,024 K	2868
☰	6b48d5999d04db6b4c791fa...	Susp...	808 K	128 K	2272

Once the process is created in a suspended state, it proceeds to introduce the binary inside the previously spawned process, which, through `ProcessHollowing`, will unmap data from itself, to write the binary inside, this is usually done through `ZwUnmapViewOfSection` + `VirtualAlloc` + `ZwWriteVirtualMemory`, once introduced into the memory of the process in suspension, it will stop being suspended and will execute it, so the memory file will be detonated.

Process Following

The image shows a multi-pane view of a debugger. At the top, a taskbar shows a process named '6b48d599d04db64c791fa311' with a status of 'Susp.' and memory usage of 1,332 K. Below this, a memory dump window displays hex and ASCII data, with a blue arrow pointing to a specific address. To the right, an 'Input' window shows a list of memory addresses and their corresponding hex values. Below the memory dump, an assembly window shows instructions like 'mov ecx, dword ptr ds:[ecx*7C]', 'push 30h', and 'push 00000000h'. A second blue arrow points from the assembly window to the 'Input' window. At the bottom, a taskbar shows the process '6b48d599d04db64c791fa311' with a status of '0.01' and memory usage of 3,324 K.

[The binary extracted from memory, which will inject explorer.exe, is very interesting, we will follow soon :)]

_IOC

_SHA256

```

Ebbebba349aba676e9739df18c503ab8c16c7fa1b853fd183f0a005c0e4f68ae
D618d086cdfc61b69e6d93a13cea06e98ac2ad7d846f044990f2ce8305fe8d1b
Ee8f0ff6b0ee072a30d45c135228108d4c032807810006ec77f2fb72856e04a
6b48d599d04db6b4c7f91fa311bfff6cae938dd50095a7a5fb7f222987efa3
B961d6795d7ceb3ea3cd00e037460958776a39747c8f03783d458b38daec8025
02083f46860f1ad11e62b2b5f601a86406f7ee3c456e6699ee2912c5d1d89cb9
059d615ce6dee655959d7feae7b70f3b7c806f3986deb1826d01a07aec5a39cf
5318751b75d8c6152d90bbb2864558626783f497443d4be1a003b64bc2acbc2
79ae89733257378139cf3bdce3a30802818ca1a12bb2343e0b9d0f51f8af1f10
F92523fa104575e0605f90ce4a75a95204bc8af656c27a04aa26782cb64d938d

```

_IP

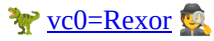
```

216.128.137.31
8.209.71.53

```

_Domains

```
host-file-host6[.]com
host-host-file8[.]com
fiskahlilian16[.]top
paishancho17[.]top
ydiannetter18[.]top
azarehanelle19[.]top
quericeriant20[.]top
xpowebs[.]ga
venis[.]ml
tootoo[.]ga
eyecosl[.]ga
bullions[.]tk
mizangs[.]tw
mbologwuholing[.]co[.]ug
quadoil[.]ru
```



[vc0=RExor](#)



Source: <https://github.com/vc0RExor/Quick-Analysis/blob/main/SmokeLoader/SmokeLoader.md>