SUNBURST, TEARDROP and the NetSec New Normal

research.checkpoint.com/2020/sunburst-teardrop-and-the-netsec-new-normal/

December 22, 2020

December 22, 2020

Foreword

In December 2020, a large-scale cyberattack targeting many organizations – predominantly tech companies, mainly in the United States, but not only there – was discovered to have been going on for several months. The attack was of a degree of sophistication that led to a quick consensus of involvement by a foreign government, and was extraordinary in both the amount of care taken in crafting it and the exotic vector of entry; instead of the usual phishing or even exploitation, the attackers carried out an elaborate supply chain attack. In this post, we share a focused analysis of some choice features of the backdoor used (SUNBURST) and one of its payloads (TEARDROP), including an exhaustive deobfuscation of SUNBURST's hashes encoding strings and an analysis of TEARDROP's control flow and decryption method; and we share our perspective on what these findings say about the attack and the people behind it, as well as what bearing this attack has on the future of network security in general.

Introduction

Here's a story you might have heard already: Mr. Exemplary CISO wakes up early one morning and goes to work as usual, a spring in his step and a bunch of one-time recovery passwords in his wallet that he never ever loses. He reaches the lobby, swipes his smart card which performs an Adi-Shamir-Level challenge-response scheme, and walks past reception where shoulder-surfers are shot on sight. He boots up his laptop, types the BIOS password which is three sentences from *Moby Dick*, presents his retina for scanning and waits patiently as the mail exchange server remotely verifies the integrity of his laptop down to the network card circuit design. A spear-phishing email reaches 40 of his colleagues, all of whom report the incident then delete the email without consciously registering the event. Somewhere on the third floor the signing certificate for a certain device driver expires, and the offending server spontaneously combusts, as per protocol. Just when he thinks life can't get any better, Mr. Exemplary CISO receives one of his favorite things in the world: A software update notification. The updated DLL is signed with the right certificate, its hash had never been seen before, it's almost identical byte-for-byte to the one sent last version, its sandbox run produces no suspicious behavior; and so the update is installed, and Mr. Exemplary CISO's organization is, how goes the parlance, "pwned", because the software supplier's production server was compromised — via social engineering, an unpatched 1-day vulnerability or the admin password being `password123`, pick your favorite — and so a sufficiently clever attacker could access that server and flawlessly arrange all the above.

There are so many ways that sufficiently clever attackers could make all our lives miserable, but usually don't, and this whole ordeal is a somber reminder of that. President of Microsoft, Brad Smith, put it this way: "This is not 'espionage as usual,' even in the digital age [..] this is not just an attack on specific targets, but on the trust and reliability of the world's critical infrastructure". We're not quite as eloquent and will just say that this isn't the Sony hack and it can't be dismissed with "don't click *update later*, don't click *enable macros*". To deflect future attacks of this sort, defenders will have to get technical, get creative, and be willing to make trade-offs that would have seemed wasteful and paranoid before. Somewhere, the author of your favorite banking Trojan just read this news, raised an eyebrow and said "hey, will someone run me a port scan on `notepad-plus-plus.org`". Even if every vendor of every popular piece of software does become hyper-vigilant now, we all can't get too complacent trusting in their hyper-vigilance. That's what we mean by the threat of a "NetSec New Normal": an unsettling step into a future of zero trust.

SUNBURST and the Art of Tactical Retreat

Technical details of the SUNBURST backdoor are widely available now in greater abundance than you will ever require, which puts us at liberty to focus on one feature that interests us and perhaps hasn't been drilled into quite like the others: The backdoor's elaborate evasion scheme.

The evasions employed by SUNBURST are similar in concept to sandbox evasions. Sandbox evasions are engineered to make sure that the malware doesn't run on virtual machines designed to detect malware; SUNBURST's evasions are engineered to make sure that the malware doesn't run on machines belonging to people who have thought of the word "malware" in the last thirty days. We've seen malware that includes blacklists of forensic tools, AV processes and such — but 1. Usually these blacklists were used to violently smother these processes instead of opting not to run the malware at all; and 2. None of them were half as comprehensive as this one. The list is an OCD-level of thorough and can be legitimately used as a resource for reverse engineers to be acquainted with new tools (ever heard of pdfstreamdumper? Well, you have now).

In-line with the overall theme of not wanting to be seen, this blacklist is not given in the form of an array of readable strings. Rather, the readable strings are replaced with FNV-1a hash values. This alone has been an occasional malware feature for years now (except the hipster-ish use of FNV-1a instead of SHA256, or even CRC32 checksums), but the feature that really stands out here is the dedication to maintaining an illusion of code legitimacy even when under direct review. The below code literally attempts to use a Jedi mind trick on the reader: "This is not the malware you are looking for, move along". The list of processes to blacklist is a "service list" belonging to the "Orion Improvement Business Layer", and these aren't hash values of process names associated with AV engines — they are "timestamps".

```
private static readonly OrionImprovementBusinessLayer.ServiceConfiguration[] svcList = new OrionImprovementBusinessLayer.ServiceConfiguration[]
{
    new OrionImprovementBusinessLayer.ServiceConfiguration
    {
        timeStamps = new ulong[]
        {
            5183687599225757871UL
        },
        Svc = new OrionImprovementBusinessLayer.ServiceConfiguration.Service[]
        {
            new OrionImprovementBusinessLayer.ServiceConfiguration.Service
            {
                timeStamp = 917638920165491138UL,
                started = true
            }
        }
    },
    new OrionImprovementBusinessLayer.ServiceConfiguration
    {
        timeStamps = new ulong[]
        {
            10063651499895178962UL
        },
        Svc = new OrionImprovementBusinessLayer.ServiceConfiguration.Service[]
        {
            new OrionImprovementBusinessLayer.ServiceConfiguration.Service
            {
                timeStamp = 16335643316870329598UL,
                started = true
            }
        }
    },
    new OrionImprovementBusinessLayer.ServiceConfiguration
    {
        timeStamps = new ulong[]
        {
            10501212300031893463UL,
            155978580751494388UL
        },
        Svc = new OrionImprovementBusinessLayer.ServiceConfiguration.Service[0]
    },
```

The authors weren't satisfied with just blacklisting processes and services. They also made sure to blacklist some device drivers and entire ranges of IP addresses (by translating the infected machine's IP to a domain name and including domain names in the blacklist), a feature that was used to blacklist all internal Solarwinds domains. This teaches us that not only the attackers decided to use Solarwinds as a Uber to get to their targets, they also learned in-detail the topology of Solarwinds' internal networks to evade the prying eyes of vigilant employees. In total, the list of hash-encoded strings embedded in SUNBURST is a paranoid manifesto of over 200 domains, providers and services that SUNBURST will just flatly refuse to deal with. Mark Russinovich put it tersely, saying that the attackers are "afraid of sysinternals". Which goes to show, even the most advanced and persistent of attackers don't believe themselves to be invincible — they believe in being just invincible enough, and above all, in not tempting fate.

The full list of FNV-1a obfuscated strings included in SUNBURST is available in Addeneum I.

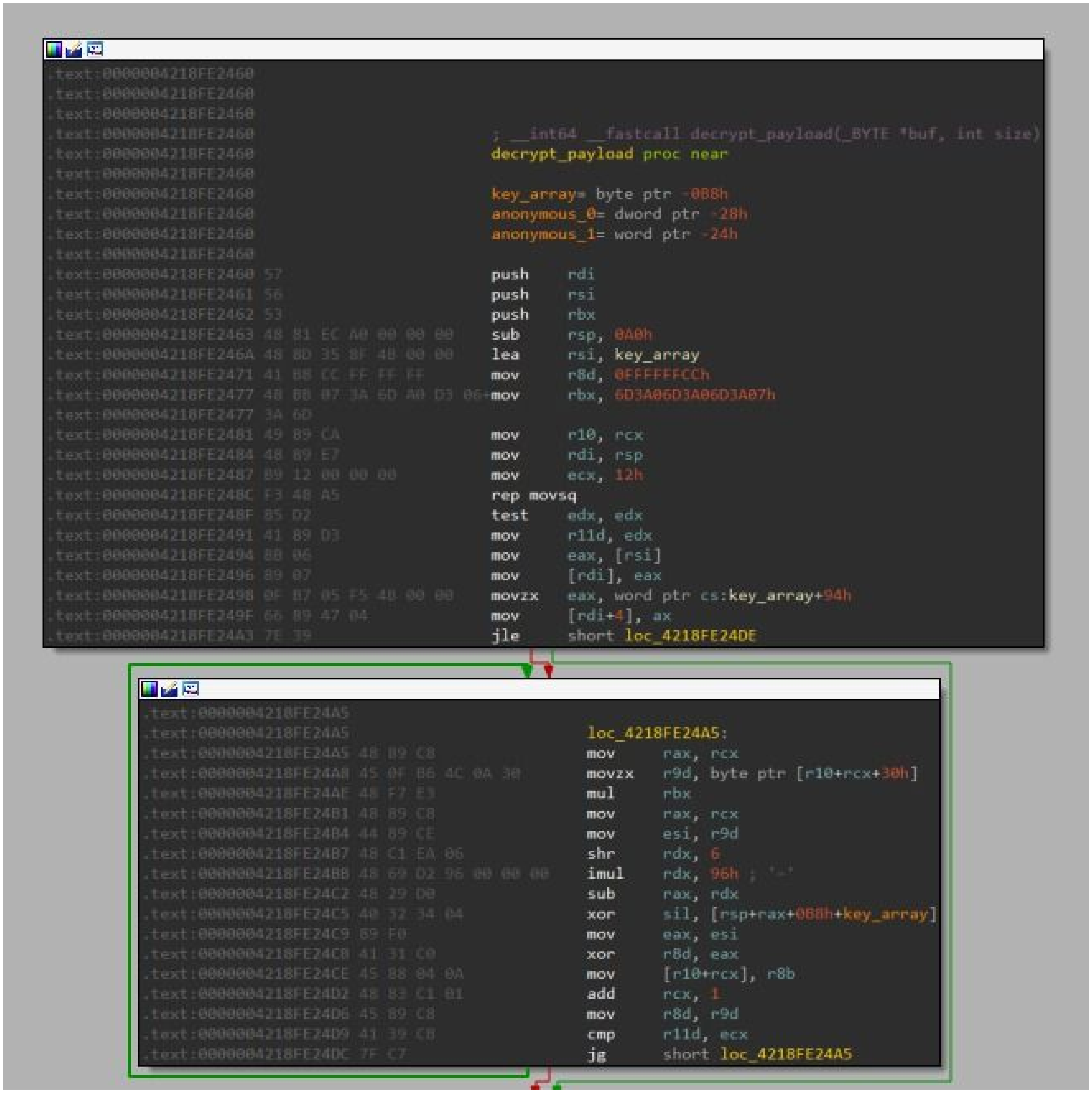
TEARDROP and Settling for the Ordinary

This attack was, no doubt, an incredible technical achievement on a large scale. Check Point Threatcloud telemetry shows over 250 organizations that were infected with the SolarWinds backdoor, half of which are in the United States. The attackers dotted their i's and crossed their t's: they made sure to follow Solarwinds' coding convention when pushing malicious code; they included a "logic bomb" in their initial payload to delay malicious activity a full two weeks from initial infection, and fool dynamic analysis; they limited their lateral movement to legitimate-seeming operations made with stolen, but valid, user credentials. For all these reasons, it's noteworthy that this Übermensch-tier attack was used to deploy TEARDROP, a merely human malware dropper.

At the time of discovery TEARDROP was a novel concoction: never-before-seen, possibly even tailor-made for this attack. It was only deployed against a select few targets. If you're eager to feel its bits and bytes, there's hashes courtesy of Talos and Sophos, as well as YARA rules by FireEye. TEARDROP runs in-memory but it does register a Windows service, which involves editing the registry.

TEARDROP's control flow is straightforward. One of the DLL exported functions, `Tk_CreateImageType`, is called during the service's execution. This function writes a JPEG image to the current directory, the name of which varies; Symantec reports having come across `upbeat_anxiety.jpg` and `festive_computer.jpg`, and FireEye has seen a `gracious_truth.jpg`. To the untrained eye, these might seem to have been named by a poet; but more likely the image name is randomly generated by concatenating two words from a hard-coded word list that's out there somewhere, on whatever machine was used to compile this piece of malware.

TEARDROP then performs decryption using a homebrew cipher and a hardcoded key of length `0x96`. The process is implemented using the following gem of disassembly:



At a high level, this reads like some sort of homebrew PRNG deciding which key byte to use each time, except the more you attempt to follow the actual process, the less sense it makes. Amazingly, when run dynamically, via some dark magic the generated key indexes simply map to `0, 1, 2, ..., 149, 0, 1, ...` and so on; that's some new level of "pseudo" in "pseudo-random"! As it turns out, this isn't a PRNG — it's a compiler-optimized implementation of the modulo operation. Feast your eyes on its underlying reasoning, which is somewhat reminiscent of the Quake Fast Inverse Square Root Hack. If anything, this is mainly a testament to the power of dynamic analysis if we ever saw it. You weren't going to statically reverse-engineer *that*. (Alternatively, it is a testament to the power of hex-rays decompiler, which sees through it immediately).

Once the optimization is understood, the decryption code is equivalent to the following:

```
CTXT_START_OFFSET = 0x30
KEY_LENGTH = 0x96
PREV_CTXT_BYTE_INITIAL_DEFAULT = 0xcc

prev_ctxt_byte = PREV_CTXT_BYTE_INITIAL_DEFAULT
for i, ctxt_byte in enumerate(ciphertext[CTXT_START_OFFSET:]):
        ptxt_byte  = ctxt_byte ^ (prev_ctxt_byte ^ key[i % KEY_LENGTH])
        plaintext[i] = ptxt_byte
        prev_ctxt_byte = ctxt_byte
```

So, the original encryption was a simple rotating XOR, followed by also XORing every ciphertext byte with the previous ciphertext byte. There's probably no purer distillation than this of "homebrew cipher thrown together in five minutes for a piece of malware". This is a perfectly good obfuscation scheme, mind you, but for the thousandth time, there is no reason for that extra XOR to be there. No one is randomly launching the Kasiski attack against in-memory binary blobs in hopes of encountering rotating XOR ciphertexts.

The decrypted payload has the following custom header format, which reads like the tl;dr of a proper PE header:

```
00000000 payload_header  struc ; (sizeof=0xC4, mappedto_24)
00000000 AddressOfEntryPoint dd ?
00000004 ImageBase       dq ?
0000000C SizeOfImage     dd ?
00000010 import_directory_rva dd ?
00000014 import_directory_size dd ?
00000018 relocation_directory_rva dd ?
0000001C relocation_directory_size dd ?
00000020 num_of_sections dd ?
00000024 SectionHeaders  section_struct 10 dup(?)
000000C4 payload_header  ends
```

And here's a taste of the payload code itself. The First image shows the code of the decrypted BEACON payload found on TEARDROP while the second image shows the code of a known BEACON sample we picked randomly. We won't fault you for not being able to find the differences between this picture and that picture. Even the PE base address is the same.

TEARDROP’s BEACON payload

```
sub_6BAC1605 proc near

dwCreationFlags= dword ptr -48h
lpThreadId= qword ptr -40h
var_38= dword ptr -38h
var_30= dword ptr -30h
var_28= dword ptr -28h
var_20= dword ptr -20h
var_18= dword ptr -18h
var_10= dword ptr -10h

sub     rsp, 68h
call    cs:GetTickCount
mov     ecx, 26AAh
xor     edx, edx
mov     r9d, 5Ch ; '\'
div     ecx
lea     rcx, Buffer     ; Buffer
mov     r8d, 5Ch ; '\'
mov     [rsp+68h+var_18], 5Ch ; '\'
mov     [rsp+68h+var_20], 65h ; 'e'
mov     [rsp+68h+var_28], 70h ; 'p'
mov     [rsp+68h+var_30], 69h ; 'i'
mov     [rsp+68h+var_38], 70h ; 'p'
mov     dword ptr [rsp+68h+lpThreadId], 5Ch ; '\'
mov     [rsp+68h+dwCreationFlags], 2Eh ; '.'
mov     [rsp+68h+var_10], edx
lea     rdx, Format     ; "%c%c%c%c%c%c%c%c%cMSSE-%d-server"
call    sprintf
lea     r8, StartAddress ; lpStartAddress
xor     ecx, ecx        ; lpThreadAttributes
mov     [rsp+68h+lpThreadId], 0 ; lpThreadId
mov     [rsp+68h+dwCreationFlags], 0 ; dwCreationFlags
xor     r9d, r9d        ; lpParameter
xor     edx, edx        ; dwStackSize
call    cs:CreateThread
xor     ecx, ecx
add     rsp, 68h
jmp     sub_6BAC15B2
sub_6BAC1605 endp
```

Cobalt Strike's BEACON (sha256: 3cfbf519913d703a802423e6e3fb734abf8297971caccc7ae45df172196b6e84)

The way TEARDROP is built, it could have dropped anything; in this case, it dropped BEACON, a payload included with Cobalt Strike (a "penetration testing" tool based on the well-known Metasploit framework). According to the Cobalt Strike website, BEACON's purpose is to model advanced attackers. It supports network lateral movement across a variety of protocols, "passive" and "active" modes for C2 check-in, and a configurable C2 communication scheme that can be made to imitate other malware or blend in with the target network's legitimate traffic.

This really bears consideration. These attackers were riding on the tail of a network breach of almost unprecedented sophistication, and now they had to pick their weapon of choice for conducting lateral movement and data exfiltration. Armed with boundless ambition and abundant resources, they looked over their options and picked… Cobalt Strike? Even Dton, the Nigerian hustler who was covered here earlier this year and objectively ranks in the top 50 of least competent cybercriminals of all time, had an intuition that using well-known commodity malware will cost him in detection rates. We can't argue with success, and this decision clearly paid off for the attackers, but we're sure curious about the reasoning behind it. Possibly it was meant to make attribution harder, and we can't rule out the use of higher-tier payloads for higher-tier targets.

Conclusion: Where to from here?

If we had to pick one actionable pithy phrase in the wake of this breach, it would be "Defense in Depth". It seems like a cliché that has been with us since forever ago, but it apparently originates with a 2012 paper by the NSA, and the principle behind it is sound and relevant: don't spend all your energy building a single wall. There are no perfect walls, and someday, someone is going to get through to the other side. When configuring a component, imagine an ongoing attack that is within reach of it now — what will help secure the component? Or an attack that has compromised the component already — how best to pre-empt the attack from propagating further? A lot of principles and practices go into this; the Principle of Least Privilege, to name one.

We're not Naïve: organizations want to Get Stuff Done, and the incentives they set effectively mandate a Principle of *Most* Privilege. Employees the world over are constantly demanding, "Just let me do this thing! Don't make me do something 'more secure' that's 4 times as complicated!". Even as we rush to zealously Secure Everything, these concerns should be taken seriously. We couldn't put it better than Avi Douglen has: "how often does strict password complexity policy enforced by IT [..] result in the user writing down his password, and taping it to his screen? That is a direct result of focusing too much on the computer aspect, at the expense of the human aspect. [..] Security at the expense of usability comes at the expense of security."

Looking at the binaries for SUNBURST and TEARDROP, we've learned that even this wildly successful operation had its rough edges. Far from a worry-free power trip, the attackers were wary all the while of having their activity seen at all, never mind recognized for what it was; extensive blacklists of domains and processes had to be created to make sure of that. We've learned that even a campaign on this level will not consist purely of ingenuous rabbit-pulls, textbook solutions and tour-de-forces; even while pulling off an astounding network security coup like this, at some points an actor will say "eh, it'll do" and reach for the ole-reliable forgettable loader, rotating XOR encryption and used-to-death commodity tool. There's something comforting about that; the attackers won this round, but maybe the game in general is not so hopeless — if defenders step up.

For full technical details on our response to the SolarWinds attack click here

Addendum I: List of FNV-1a Obfuscated Strings Included in SUNBURST

```
Processes:
	2597124982561782591 = apimonitor-x64
	2600364143812063535 = apimonitor-x86
	13464308873961738403 = autopsy64
	4821863173800309721 = autopsy
	12969190449276002545 = autoruns64
	3320026265773918739 = autoruns
	12094027092655598256 = autorunsc64
	10657751674541025650 = autorunsc
	11913842725949116895 = binaryninja
	5449730069165757263 = blacklight
	292198192373389586 = cff explorer
	12790084614253405985 = cutter
	5219431737322569038 = de4dot
	15535773470978271326 = debugview
	7810436520414958497 = diskmon
	13316211011159594063 = dnsd
	13825071784440082496 = dnspy
	14480775929210717493 = dotpeek32
	14482658293117931546 = dotpeek64
	8473756179280619170 = dumpcap
	3778500091710709090 = evidence center
	8799118153397725683 = exeinfope
	12027963942392743532 = fakedns
	576626207276463000 = fakenet
	7412338704062093516 = ffdec
	682250828679635420 = fiddler
	13014156621614176974 = fileinsight
	18150909006539876521 = floss
	10336842116636872171 = gdb
	12785322942775634499 = hiew32demo
	13260224381505715848 = hiew32
	17956969551821596225 = hollows_hunter
	8709004393777297355 = idaq64
	14256853800858727521 = idaq
	8129411991672431889 = idr
	15997665423159927228 = ildasm
	10829648878147112121 = ilspy
	9149947745824492274 = jd-gui
	3656637464651387014 = lordpe
	3575761800716667678 = officemalscanner
	4501656691368064027 = ollydbg
	10296494671777307979 = pdfstreamdumper
	14630721578341374856 = pe-bear
	4088976323439621041 = pebrowse64
	9531326785919727076 = peid
	6461429591783621719 = pe-sieve32
	6508141243778577344 = pe-sieve64
	10235971842993272939 = pestudio
	2478231962306073784 = peview
	9903758755917170407 = peview
	14710585101020280896 = ppee
	13611814135072561278 = procdump64
	2810460305047003196 = procdump
	2032008861530788751 = processhacker
```

```
27407921587843457 = procexp64
6491986958834001955 = procexp
2128122064571842954 = procmon
10484659978517092504 = prodiscoverbasic
8478833628889826985 = py2exedecompiler
10463926208560207521 = r2agent
7080175711202577138 = rabin2
8697424601205169055 = radare2
7775177810774851294 = ramcapture64
16130138450758310172 = ramcapture
506634811745884560 = reflector
18294908219222222902 = regmon
3588624367609827560 = resourcehacker
9555688264681862794 = retdec-ar-extractor
5415426428750045503 = retdec-bin2llvmir
3642525650883269872 = retdec-bin2pat
13135068273077306806 = retdec-config
3769837838875367802 = retdec-fileinfo
191060519014405309 = retdec-getsig
1682585410644922036 = retdec-idr2pat
7878537243757499832 = retdec-llvmir2hll
13799353263187722717 = retdec-macho-extractor
1367627386496056834 = retdec-pat2yara
12574535824074203265 = retdec-stacofin
16990567851129491937 = retdec-unpacker
8994091295115840290 = retdec-yarac
13876356431472225791 = rundotnetdll
14968320160131875803 = sbiesvc
14868920869169964081 = scdbg
106672141413120087 = scylla_x64
79089792725215063 = scylla_x86
5614586596107908838 = shellcode_launcher
3869935012404164040 = solarwindsdiagnostics
3538022140597504361 = sysmon64
14111374107076822891 = sysmon64
7982848972385914508 = task explorer
8760312338504300643 = task explorer-x64
17351543633914244545 = tcpdump
7516148236133302073 = tcpvcon
15114163911481793350 = tcpview
15457732070353984570 = vboxservice
16292685861617888592 = win32_remote
10374841591685794123 = win64_remotex64
3045986759481489935 = windbg
17109238199226571972 = windump
6827032273910657891 = winhex64
5945487981219695001 = winhex
8052533790968282297 = winobj
17574002783607647274 = wireshark
3341747963119755850 = x32dbg
14193859431895170587 = x64dbg
17439059603042731363 = xwforensics64
17683972236092287897 = xwforensics
700598796416086955 = redcloak
3660705254426876796 = avgsvc
```

```
        12709986806548166638 = avgui
        3890794756780010537 = avgsvca
        2797129108883749491 = avgidsagent
        3890769468012566366 = avgsvcx
        14095938998438966337 = avgwdsvcx
        11109294216876344399 = avgadminclientservice
        1368907909245890092 = afwserv
        11818825521849580123 = avastui
        8146185202538899243 = avastsvc
        2934149816356927366 = aswidsagent
        13029357933491444455 = aswidsagenta
        6195833633417633900 = aswengsrv
        2760663353550280147 = avastavwrapper
        16423314183614230717 = bccavsvc
        2532538262737333146 = psanhost
        4454255944391929578 = psuaservice
        6088115528707848728 = psuamain
        13611051401579634621 = avp
        18147627057830191163 = avpui
        17633734304611248415 = ksde
        13581776705111912829 = ksdeui
        7175363135479931834 = tanium
        3178468437029279937 = taniumclient
        13599785766252827703 = taniumdetectengine
        6180361713414290679 = taniumendpointindex
        8612208440357175863 = taniumtracecli
        8408095252303317471 = taniumtracewebsocketclient64
Services:
windows Defender:
    5183687599225757871 = msmpeng
    917638920165491138 = windefend
Windows Sense:
    10063651499895178962 = mssense
    16335643316870329598 = sense
Windows Sensor:
    10501212300031893463 = microsoft.tri.sensor
    155978580751494388 = microsoft.tri.sensor.updater
NIST:
    17204844226884380288 = cavp
Carbon Black:
        5984963105389676759 = cb
    11385275378891906608 = carbonblack
    13693525876560827283 = carbonblackk
    17849680105131524334 = cbcomms
    18246404330670877335 = cbstream
CrowdStrike:
    8698326794961817906 = csfalconservice
    9061219083560670602 = csfalconcontainer
    11771945869106552231 = csagent
            9234894663364701749 = csdevicecontrol
        8698326794961817906 = csfalconservice
FireEye:
    15695338751700748390 = xagt
    640589622539783622 = xagtnotif
    9384605490088500348 = fe_avk
```

```
    6274014997237900919 = fekern
    15092207615430402812 = feelam
            3320767229281015341 = fewscservice
ESET:
    3200333496547938354 = ekrn
    14513577387099045298 = eguiproxy
    607197993339007484 = egui
    15587050164583443069 = eamonm
    9559632696372799208 = eelam
    4931721628717906635 = ehdrv
        2589926981877829912 = ekrnepfw
    17997967489723066537 = epfwwfp
    14079676299181301772 = ekbdflt
    17939405613729073960 = epfw
F-SECURE:
    521157249538507889 = fsgk32st
        14971809093655817917 = fswebuid
    10545868833523019926 = fsgk32
    15039834196857999838 = fsma32
    14055243717250701608 = fssm32
    5587557070429522647 = fnrb32
    12445177985737237804 = fsaua
    17978774977754553159 = fsorsp
    17017923349298346219 = fsav32
    17624147599670377042 = f-secure gatekeeper handler starter
    16066651430762394116 = f-secure network request broker
            13655261125244647696 = f-secure webui daemon
    3421213182954201407 = fsma
    14243671177281069512 = fsorspclient
    16112751343173365533 = f-secure gatekeeper
    3425260965299690882 = f-secure hips
    9333057603143916814 = fsbts
    3413886037471417852 = fsni
    7315838824213522000 = fsvista
    13783346438774742614 = f-secure filter
    2380224015317016190 = f-secure recognizer
    3413052607651207697 = fses
    3407972863931386250 = fsfw
    10393903804869831898 = fsdfw
    3421197789791424393 = fsms
    541172992193764396 = fsdevcon
Drivers:
            17097380490166623672 = cybkerneltracker.sys
    15194901817027173566 = atrsdfw.sys
    12718416789200275332 = eaw.sys
    18392881921099771407 = rvsavd.sys
    3626142665768487764 = dgdmk.sys
    12343334044036541897 = sentinelmonitor.sys
    397780960855462669 = hexisfsmonitor.sys
    6943102301517884811 = groundling32.sys
    13544031715334011032 = groundling64.sys
    11801746708619571308 = safe-agent.sys
    18159703063075866524 = crexecprev.sys
    835151375515278827 = psepfilter.sys
    16570804352575357627 = cve.sys
```

```
    16144657739388842903 = brfilter.sys
    12679195163651834776 = brcow_x_x_x_x.sys
    2717025511528702475 = lragentmf.sys
    17984632978012874803 = libwamf.sys
domain names:
    1109067043404435916 = swdev.local
    15267980678929160412 = swdev.dmz
    8381292265993977266 = lab.local
    3796405623695665524 = lab.na
    8727477769544302060 = emea.sales
    10734127004244879770 = cork.lab
    11073283311104541690 = dev.local
    4030236413975199654 = dmz.local
    7701683279824397773 = pci.local
    5132256620104998637 = saas.swi
    5942282052525294911 = lab.rio
    4578480846255629462 = lab.brno
    16858955978146406642 = apac.lab
HTTP:
            8873858923435176895 = expect
            6116246686670134098 = content-type
            2734787258623754862 = accept
            6116246686670134098 = content-type
            7574774749059321801 = user-agent
            1475579823244607677 = 100-continue
            11266044540366291518 = connection
            9007106680104765185 = referer
            13852439084267373191 = keep-alive
            14226582801651130532 = close
            15514036435533858158 = if-modified-since
  16066522799090129502 = date
```