xuanxuan0 Update README.md 1b31ed7 · 3 years ago 17 Commits

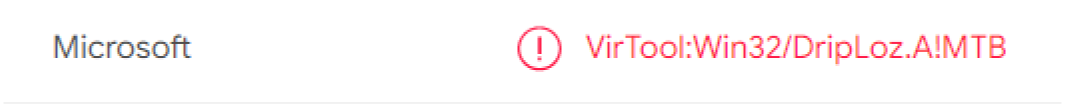
.github/workflows	Create msbuild.yml	3 years ago
DripLoader	Update Helpers.cpp	3 years ago
.gitattributes	Add .gitignore and .gitattributes.	3 years ago
DripLoader.sln	Add project files.	3 years ago
LICENSE	Create LICENSE	3 years ago
README.md	Update README.md	3 years ago

README MIT license

DripLoader (PoC)

MSBuild no status

Evasive shellcode loader for bypassing event-based injection detection, without necessarily suppressing event collection. The project is aiming to highlight limitations of event-driven injection identification, and show the need for more advanced memory scanning and smarter local agent software inventories in EDR.


Microsoft VirTool:Win32/DripLoz.A!MTB

DripLoader evades common EDRs by:

- using the most risky APIs possible like `NtAllocateVirtualMemory` and `NtCreateThreadEx`
- blending in with call arguments to create events that vendors are forced to drop or log&ignore due to volume
- avoiding multi-event correlation by introducing delays

What does DripLoader do

- Identifies a base address suitable for our payload
- Reserves enough `AllocationGranularity` (64kB) sized, `NO_ACCESS` memory segments at the base address
- Loops over those
  - Allocating `PageSize` (4kB) sized, writable segments
  - Writing shellcode
  - Reprotecting as `RX`
- Overwrites prologue of one `ntdll` function in the remote process memory space with a `jmp` to our base
- Drops a thread on that trampoline

About

Evasive shellcode loader for bypassing event-based injection detection (PoC)

blog.redbluepurple.io/offensive-research/...

shellcode shellcode-loader edr shellcode-injector evasion-attacks

Releases 1

v0.9 Latest
on Apr 28, 2021

Packages

No packages published

Languages

● C++ 96.9% ● Assembly 1.9%
● C 1.2%

I'll explain some of the thinking here: https://blog.redbluepurple.io/offensive-research/bypassing-injection-detection

And so

- It's able to fully bypass many EDR injection detections, including Defender ATP.
- Bypasses simple thread-centric scanners like `Get-InjectedThread`. Persisting within a process is another story, and this is up to the payload author.
- It is `sRDI`-compatible, but if your payload creates another local thread you will lose the benefit of thread start address in `ntdll`.

To test it out of the box

- compile/download
- XOR your binary shellcode blob file with default key 0x08, name it `blob.bin`
- place both files in the same directory
- run it and follow the prompts or ./DripLoader.exe <target_pid> <delay_per_step_ms>

I attached an example `MessageBox` blob for your pleasure, be aware though it's size is unrealistically small for a payload.