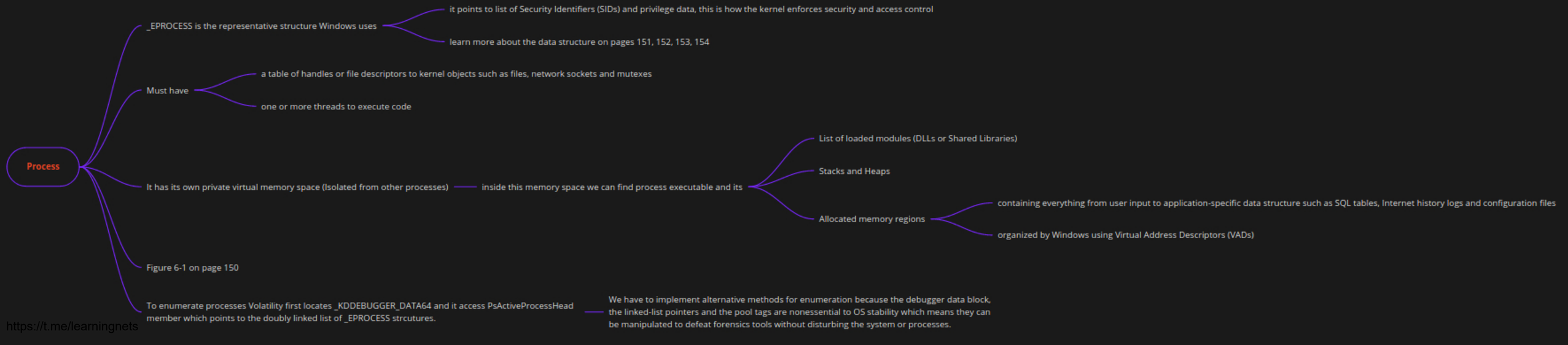
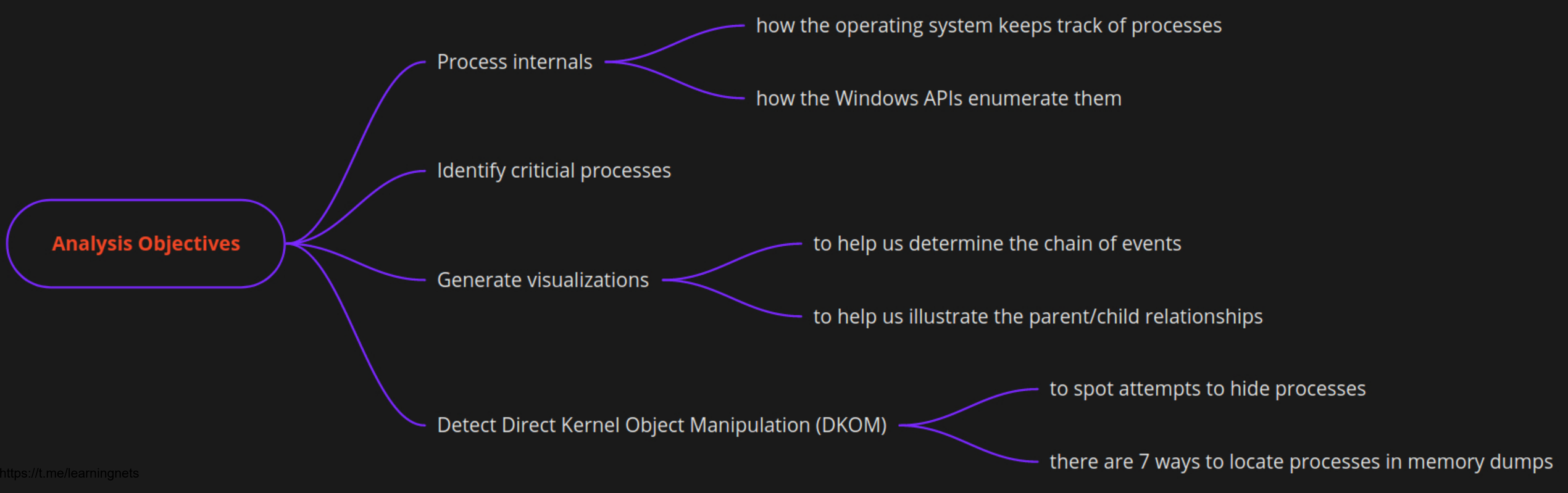
The Art of Memory Forensics

Ch.6 - Processes, Handles and Tokens

0x1411

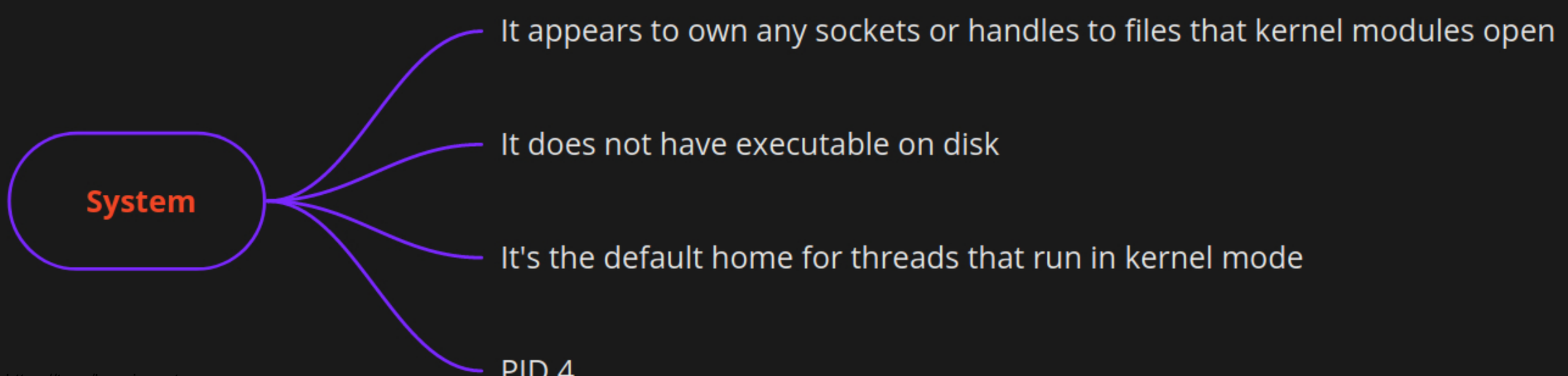


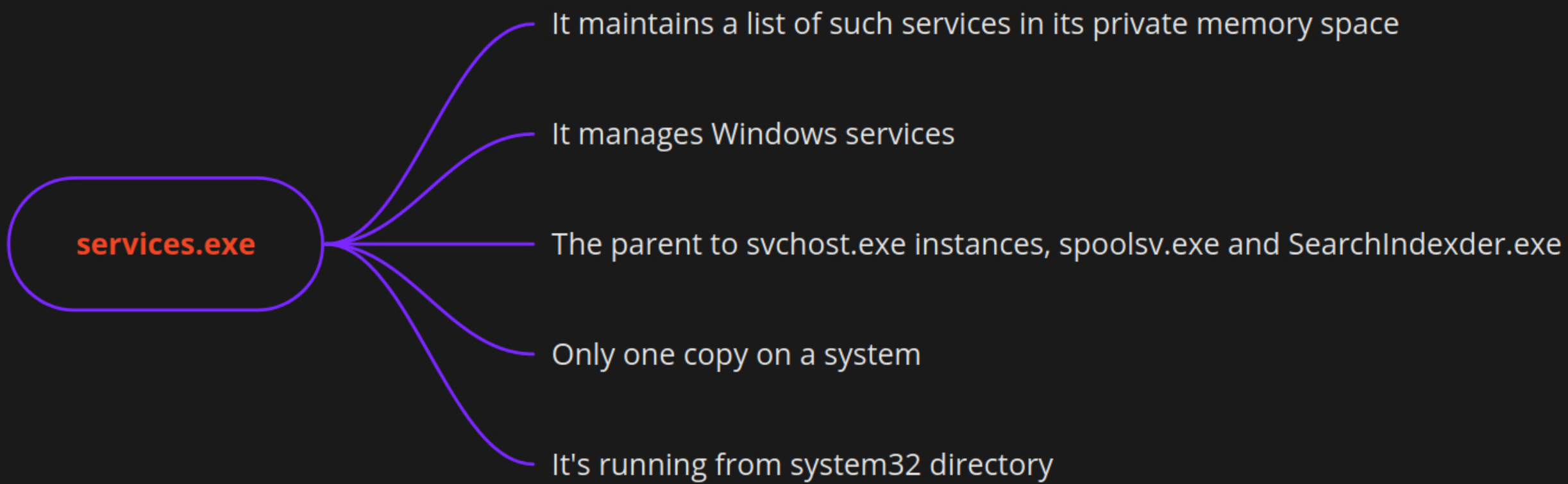


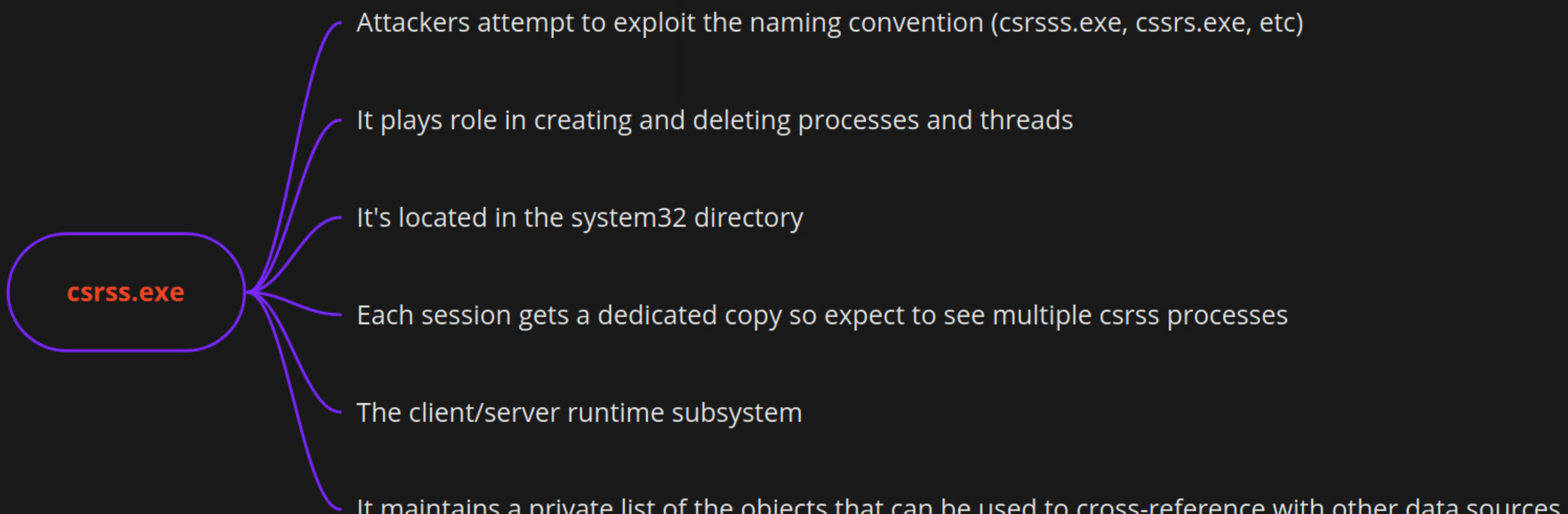
Critical System Processes

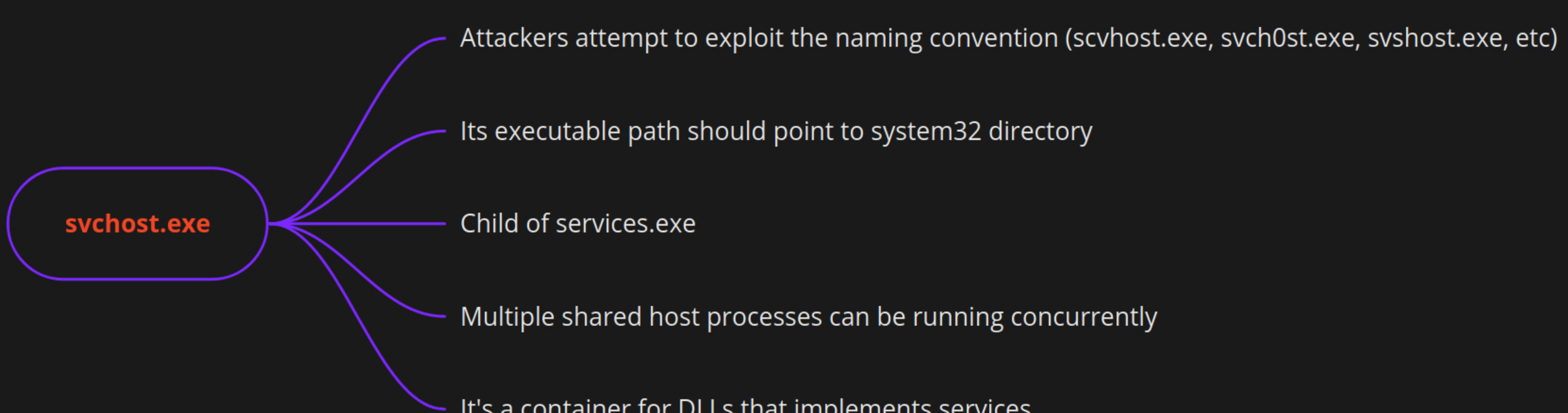
It's a container the kernel uses to charge CPU time for idle threads

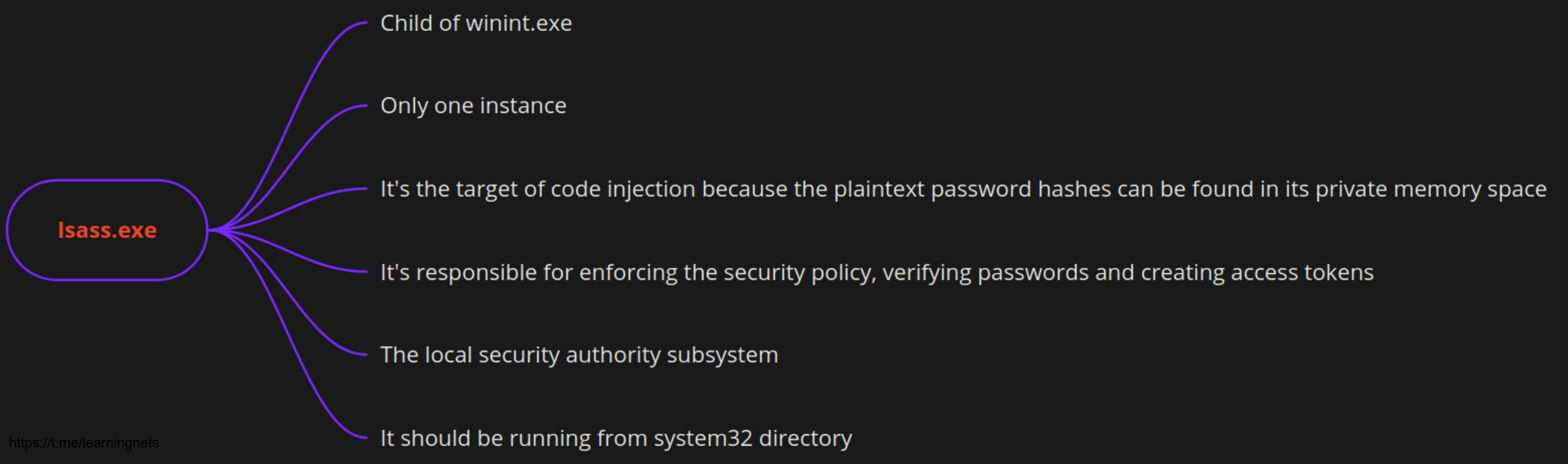
It does not have executable on disk

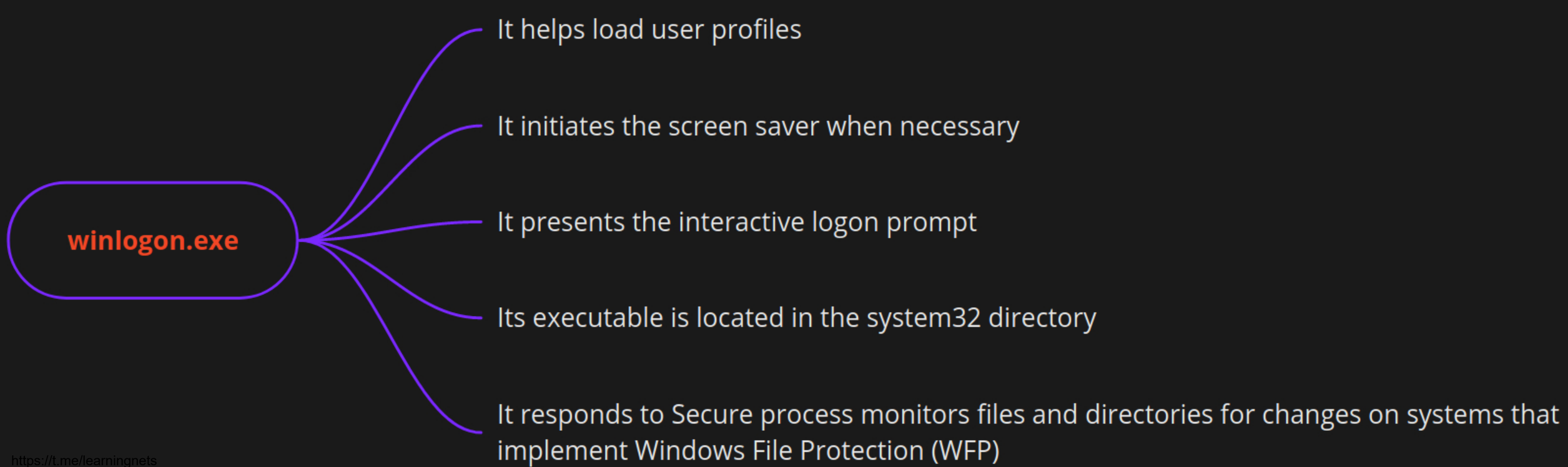


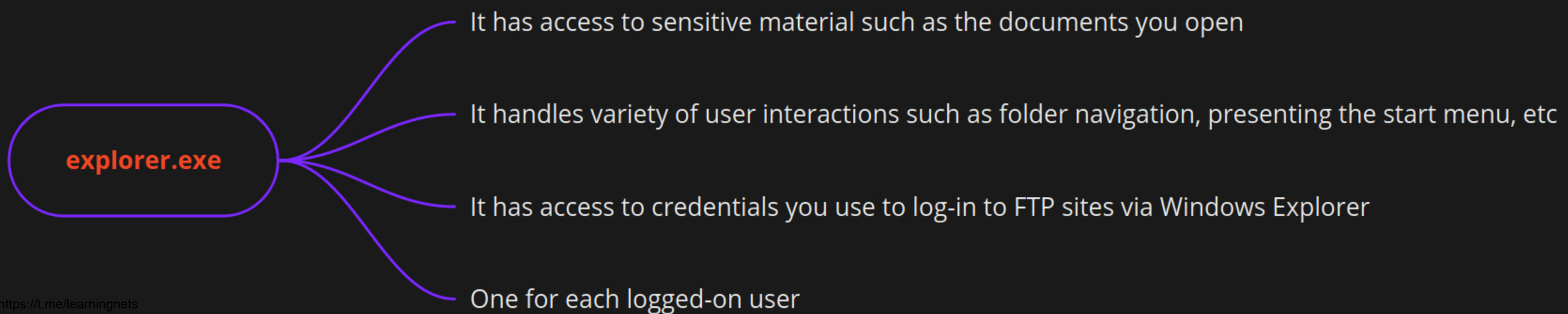


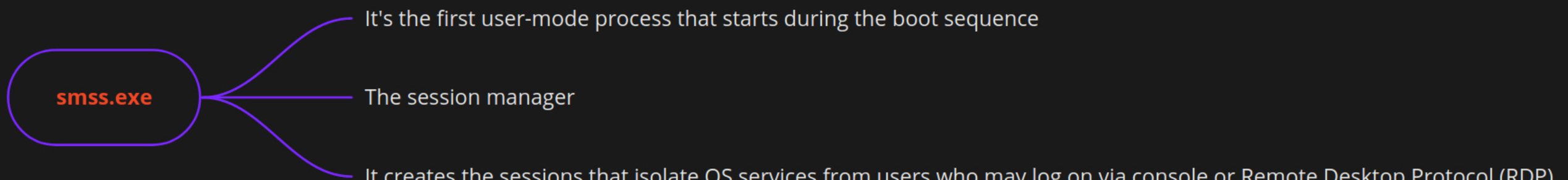


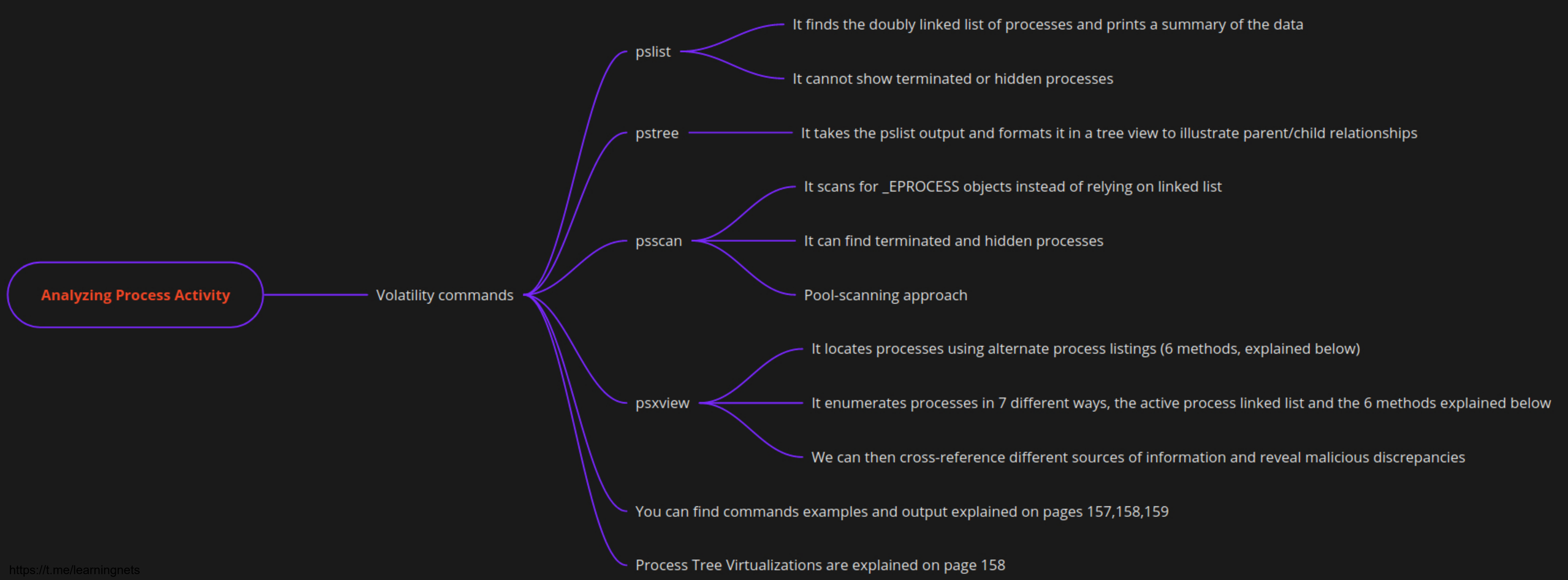


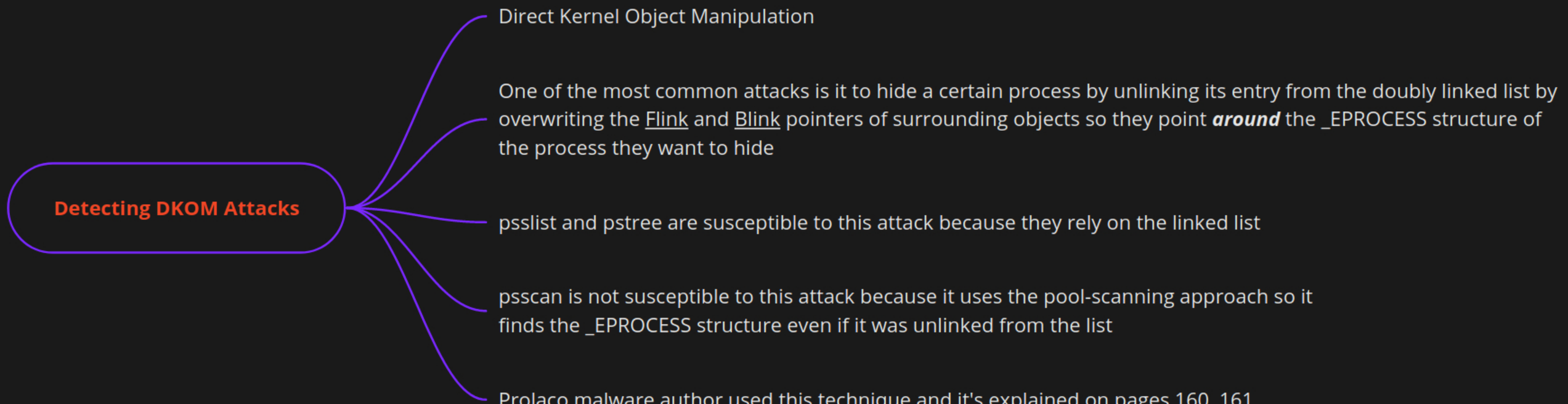






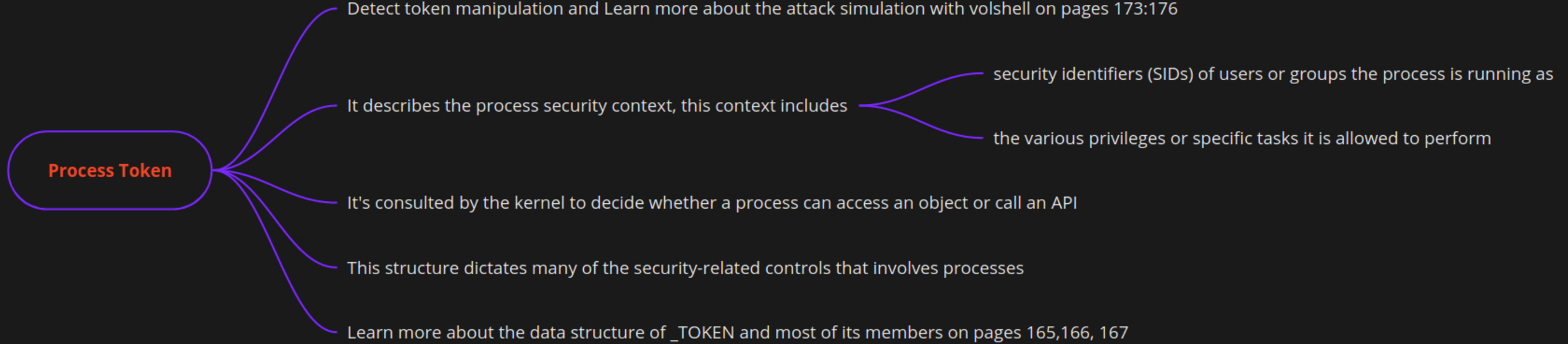


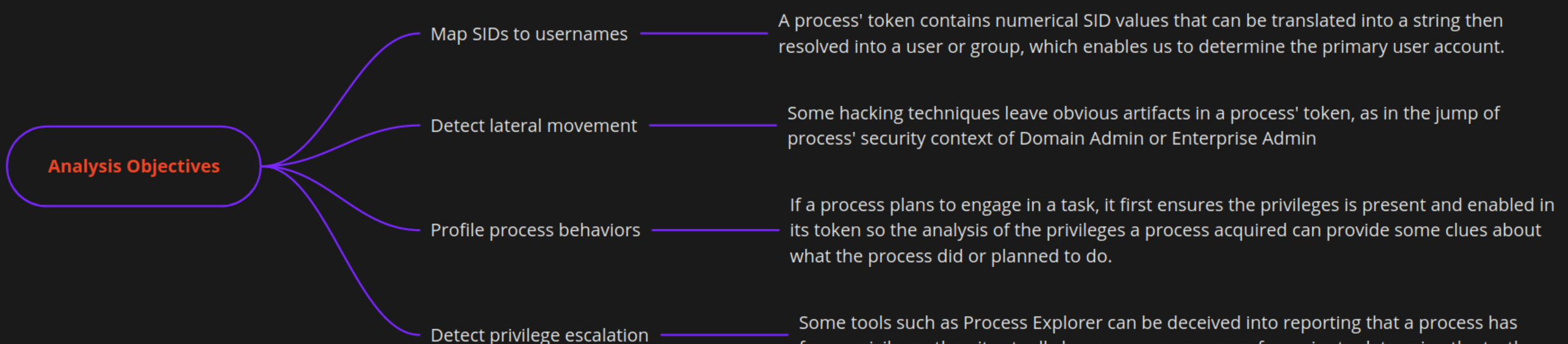


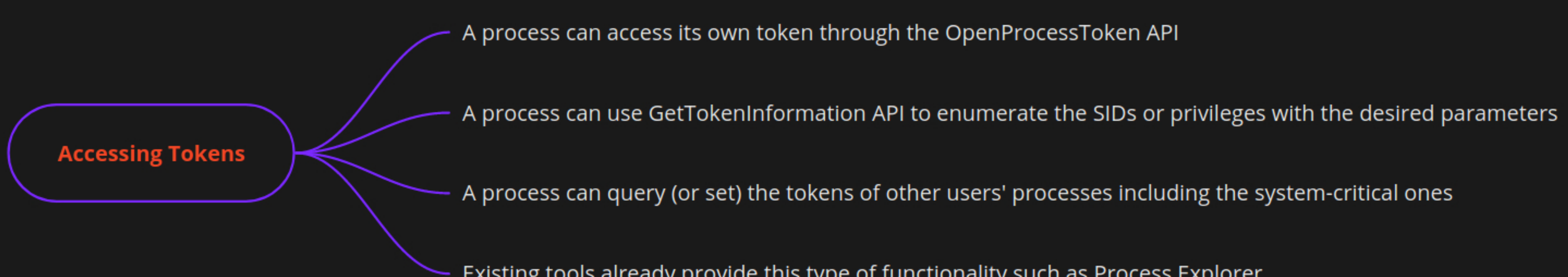


Alternate Process Listings

- Process object scanning
  - As discussed in chapter 5
  - Since the pool tags are not essential, they can manipulated to evade the scanner
- Thread scanning
  - Every process must have at least one thread that execute code
  - Scanner searches for _ETHREAD objects then map them back to their owning processes
  - The member used for mapping is _ETHREAD.Tcb.Process
- CSRSS handle table
  - csrss.exe is involved in creating every process and thread except for itself and the processes started before it
  - Scanner walks the handle table to identify all _EPROCESS objects
- PspCid table
  - It's a special handle table located in kernel memory
  - It stores a reference to all active processes and thread objects
  - The PspCidTable member of the kernel debugger data structure points to the table
- Session processes
  - The SessionProcessLinks member of _EPROCESS associates all processes for every logged-on user session
  - It is not hard to unlink a process from this list, as apposed to the ActiveProcessLinks list
  - Attackers rarely find value in targeting it because live system APIs do not depend on it
- Desktop threads
  - Discussed in details in chapter 14
  - They store list of all threads attached to each desktop
  - The can be easily mapped to their owning process

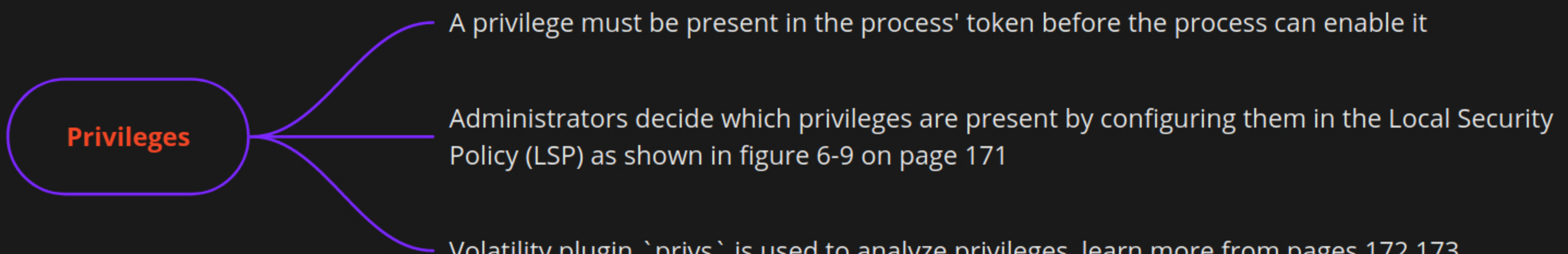


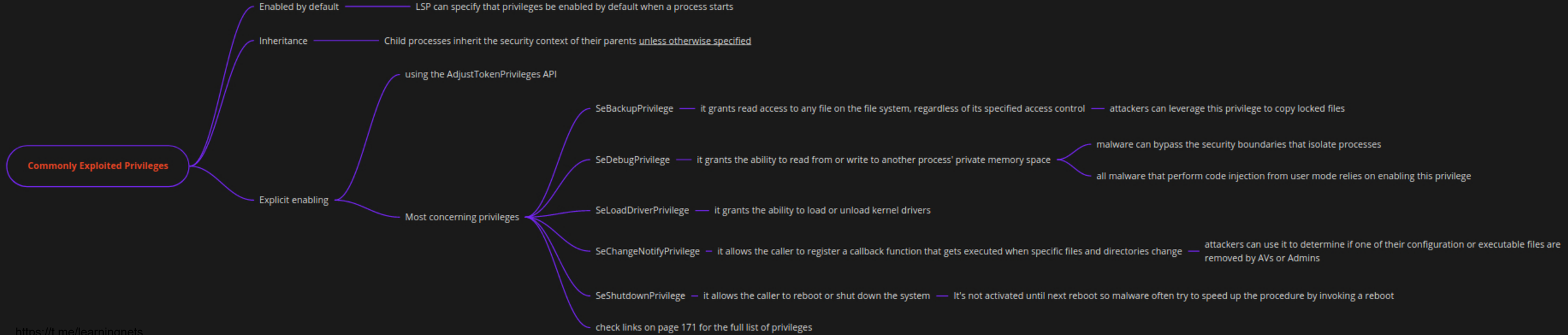




Extracting and Translating SIDs in Memory

- ConvertSidToStringSid API translates the numerical data in _SID structure to human-readable
- LookupAccountSid API returns an account name for a given SID
- getsid volatility plugin can find each process' token, extract the numerical component of _SID structure, translate them into strings, and map the strings to use and group names (learn more on page 168)





Process Handles

- A handle is a reference to an open instance of a kernel object such as files, registry keys, mutexes, processes or threads
- There are close to 40 different types of kernel objects
- Before a process can access an object, it first opens a handle to the object by calling an API such as CreateFile, RegOpenKeyEx, or CreateMutex (example on page 176)
- Kernel Handles example on page 177
- Analysis Objectives
  - Handle table internals — learning more about handles internals can give greater understanding of objects
  - Targeted object attribution — with a given indicator such as registry key, we can trace it back to its creator process
  - Detect registry persistence
    - learn how to analyze open registry handles to determine which key is used to store configurations
    - example on pages 183, 184
  - Identify remote mapped drives — example on pages 185,186
- Handle Table Internals
  - _EPROCESS.ObjectTable member points to a handle table (_HANDLE_TABLE)
  - TableCode serves two critical purposes
    - it specifies the number of levels in the table
    - it points to the base address of the first level
  - All processes start with a single-level table
    - example on page 178,179
    - Figure 6-12 on page 179
  - Data structures on page 179,180
- The volatility plugin `handles` is used to enumerate handles, example on pages 181,182