1. **Security product under test**

**Assumptions**

The task of this particular security product is to

* identify processes/executables running on OS
* find unique information about each process, such as: pid, parent process, path, description, process group. (vary by OS)
* calculate the hash of this process based on the info above)
* store this hash in cache (memory data structure) for 60 secs along with a capture time stamp and count
* what happens after 60 secs, is the hash moved to the DB?
* constant polling happens, say, every 5 sec
* with every poll the list of processes is evaluated against the list in cache

if the hash from the new list if found in cache, the time stamp is reset, and count incremented

* Another process should run every 1 sec (30 ms) to examine the cache data. The entries with timestamp older that 60 sec. should be moved to perm storage.

1. **To test this security product**
   1. Identify the matrix of OS/App/Web servers to be covered by testing
   2. Identify success/failure criteria for functional, usability, performance, reliability testing
   3. Develop a TEST APP.
   4. Deploy the Product and TEST APP on the identified web server. Note, to evaluate happy path for functional test it’s good enough to use one Linux and one Win webserver.
   5. Create a test suite to validate the Product.
2. **Test Application -**

**should be able to generate various types of system calls on desired OSs (say, Windows and Unix) as**:

* 1. Process Control (example – create and terminate process, allocate&free memory)
  2. File Management (example – create a file, read/update/reposition/delete/change file attributes)
  3. Device Management (request/release a device (printer), logically attach/detach, get/set device attributes
  4. Information Maintenance (example – get/set time/date, get process/device attributes)
  5. Communications (example – create/delete communication connections, help OS to transfer status information, attach/detach remote devices)

**Here comes the difference between Unix and Window, since the systems calls differ on each OS**

| **Categories** | **Windows** | **Unix** |
| --- | --- | --- |
| Process control | CreateProcess() ExitProcess() WaitForSingleObject() | fork() exit() wait() |
| Device manipulation | SetConsoleMode() ReadConsole() WriteConsole() | loctl() read() write() |
| File manipulation | CreateFile() ReadFile() WriteFile() CloseHandle() | Open() Read() write() close!) |
| Information maintanence | GetCurrentProcessID() SetTimer() Sleep() | getpid() alarm() sleep() |
| Communication | CreatePipe() CreateFileMapping() MapViewOfFile() | Pipe() shm\_open() mmap() |
| Protection | SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup () | Chmod() Umask() Chown() |

The TEST APP itself can be multiplatform or 2 flavors of the same RESTful API app for Unix and Windows, which will be deployed on various web servers running on Win and Unix. **Assumption:** API endpoints should be identical on both flavors of the TEST APP.

1. **Test Cases** (given an endpoint generate a flood of processes and all have unique hash values)

**Assumption:** the endpoint is a RESTful endpoint of the TEST APP referenced above.

Question: how to I make a process unique so that it will have a unique hash?

* PID makes a process unique

Create a number of processes – some of them executable, some not – I know the number

Of them

Work with timing also –

Automated test cases:

1. Given an endpoint generate a flood of processes and all have unique hash values.

* Due to the vague requirements, I’m basing my response on the following assumptions:
  + Using my endpoint on the webserver I’m generating the files. Not sure if that will span unique system processes.
  + My test script generates unique processes, with unique PID, which will result in unique HASH.
* Test coverage:

1. Generate a known number of processes with unique hash over time < 60 sec. Say, 10 secs.
   1. Test 1: verify the new number of saved hashes in the DB is 0.
   * Wait 60 sec more and pull the data from DB to verify the new number of saved hashes in the DB is exactly the number of your unique requests.
2. No new hashes generated (only 1 actually) – use test script to generate not-unique processes. Verify DB table after > 60 sec there should be 1 new process saved.
3. Generate a mix of known number of unique processes and non-unique processes (haven’t done it)

**TO SETUP the test:**

* Deploy provided .war file on a webserver running http on 8080
  + The endpoint will look like <http://localhost:8080/sample/command.jsp> If your webserver is running on a different port or you want to change smth, change the url in python test file “generateProcesses.py”
* Make sure you have python3 setup and packages imported
  + To generate unique OS processes with unique hash run test script as : python generateProcesses.py U <number of desired processes> <path to tomcat executable dir>
    - Example: python generateProcesses.py U 10 /usr/local/Cellar/tomcat/9.0.39/libexec/
  + For non-unique processes use: python generateProcesses.py N <number of desired processes> <path to tomcat executable dir>
    - Example: python generateProcesses.py N 10 /usr/local/Cellar/tomcat/9.0.39/libexec/
    - NOTE – this test still will have 1 unique process