CS-501

Introduction to Malware, Threat Hunting & Offensive Capabilities Development

Lecture 7: More YARA. More Processes

-

Modules:

Allows for extension of features

Want to implement rules on custom data structures/types?

Want to write rules to express complex conditions and package them for public use?

Want to create “dry code” but for yara rules?

Modules: PE

- Fine grained rules for Portable Executables and DLLs
- PE files contain a lot of information that we can parse out using YARA’s PE module
- Imphash : returns the impash of a pe (md5 of the import address table)
- section_index(name) : useful for parsing sections
- exports(function_name): check if the PE exports a function

Modules: Hash

Module containing functionality supporting cryptographic hash functions

This is usually best used with the PE module, to identify and hash sequences of data

Windows System Programming

Program vs Process

A program is a static collection of instructions. A process is a container for a set of resources used when executing **an instance** of a program

What Makes a Process

- A **private** virtual address space
- A program mapped in the virtual address space
- A collection of open **handles to objects**
- A Process ID
- >=1 thread
- *Security/token information.

Processes Vs Threads

- Processes are (usually) independent, and are containers for threads
  - Non Example: python multiprocessing, Chrome.exe + sandbox
- Threads in the same process share process state, memory and other resources
- Processes have separate address spaces, and threads in the same process share their address space
- Processes can only interact with each other via system-provided Inter Process Communication (IPC) mechanisms
  - Pipes, sockets, files, ...etc
- Threads have per-process shared storage (TLS)
- Context switching is more expensive between processes than it is between threads in a process.

Threads

An entity within a process that that actually executes code

- Threads are scheduled
- Threads have access to the contents of multiple CPU registers
- Threads hace private memory for shared objects (Thread Local Storage)
- Threads can have a security context that is different from other threads within a process. Security is weird with Windows.
- Users can schedule their own threads via Fibers

Kernel Objects

- Kernel object (KOs): a single run-time instance of a statically defined object type
- Object types are system-defined data types.
- Each object type has its own attributes, and functions to interact with it
- For example, an object of type process is an instance of a process object
- A file object is an instance of a file...etc

If each process gets its Virtual Address space, how do we interact with shared objects?

Objects and Handles

A handle is an abstract reference to an object. This could be an actual pointer to the object, or a reference to a GUID that references an object

This allows us to abstract away direct management of objects in memory, and instead work with with references. This is also a security control.

It also allows gives us an API to interact with system resources, share resources among processes, and protect resources from unauthorized access.

Sharing Objects

We share objects by sharing handles to objects

In order to share a handle, we can either copy a reference to the same handle, or duplicate the handle (this creates a new GUID)

The Registry

The registry is a system database that is used to control the configuration of your environment.

Screensavers, desktop background, startup tasks, runkeys...these are all stored in the registry

References counters to handles are stored in the registry

Process Creation

- Recall that process creation is handled by the kernel
- An executable image is loaded from disk
- Its sections are parsed
- All imports are resolved
- And once that is complete, execution is passed to the main thread

There are many ways to actually start a process

Starting a Process in C++ (POSIX)

- system()
- _popen()

Starting Programs (Win API)

- ShellExecute
- CreateProcess
  - CreateProcessWithToken, CreateProcessAsUser, CreateProcessWithLogon
  - CreateProcessInternal, NtCreateProcess,
- Schtask
- RunKeys

Digging into CreateProcess’ Arguments

```
BOOL CreateProcessW(
  LPCWSTR               lpApplicationName,
  LPWSTR                lpCommandLine,
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  BOOL                  bInheritHandles,
  DWORD                 dwCreationFlags,
  LPVOID                lpEnvironment,
  LPCWSTR               lpCurrentDirectory,
  LPSTARTUPINFOW        lpStartupInfo,
  LPPROCESS_INFORMATION lpProcessInformation
);
```

https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw

Starting Processes and Getting Output

- Example: What if we want to run powershell or a CMD command?

Python Analogy

Directing output to a File

```
In [1]: import os

In [2]: os.system("whoami")
laptop-\user
Out[2]: 0

In [3]: os.system("whoami > a.txt")
Out[3]: 0

In [4]: with open("a.txt" ,"r") as f:
   ...:     result = f.read()
   ...:

In [5]: print(result)
laptop-\user
```

Directing Output to a Pipe

```
In [8]: import subprocess

In [9]: result = subprocess.Popen('whoami', stdout=subprocess.PIPE)

In [10]: print(result.stdout.read())
b'laptop-4rn9eipi\\user\r\n'
```

system/popen C++

System: output can be retrieved by redirecting output to a file

popen(_ “r”) can be used to retrieve output from stdout using a pipe

CreateProcess - Option 1

- Call create process
- Pass a handle to a FILE that will be written to disk
- This is accomplishes the job, **but** touches disk
- What if we don’t want to touch disk?
- This is where PIPes and Handle inheritance come into play

Handle Inheritance

There is a 3rd way to share a handle to an object

In the special case where a processes is created by another process, the parent process can choose to share its handles with the child process

Once CreateProcess is called with the 5th argument set to TRUE, the child process will inherit all handles owned by the parent process that have their inheritance bit set.

Inheritance actually works under the hood by duplicating the handles and passing them to the child processes

Note that the handle values are the same as in the parent process

Pipes

A pipe is a block of shared memory that processes can use for communication and data exchange. They behave in the same easy as a file, where multiple processes can read and write from the file, just (hopefully) not at the exact same time.

Pipes can either be anonymous or named. Anonymous pipes are only accessible from the processes they were created in and will get a random identifier

Named pipes exist in memory and can be accessed via //pipes//...

Named Pipes enables two unrelated processes to exchange data between themselves, even if the processes are located on two different networks.

A named pipe server can open a named pipe with some predefined name and then a named pipe client can connect to that pipe via the known name. Once the connection is established, data exchange can begin.

https://www.ired.team/offensive-security/privilege-escalation/windows-namedpipes-privilege-escalation#:~:text=Overview,located%20on%20two%20different%20networks.

The plan:

- Create a Process, and pass in two write handles to Pipes
  - One for STDOUT
  - One for STDERR
    - Alternatively, (and this is usually easier!) we can make them the same
- We allow the child processes to write to these pipes, setting the correct attributes to enable handle inheritance
- From there, we wait for the process to terminate
- Once the process terminates, we use the Read handles for our pipes, and check the the contents.
- Cleanup as needed

HTTP (s)

- Hypertext Transfer Protocol (secure)
- Built on top of TCP
- Sends data in structured envelopes called HTTP Requests
- Data in the requests are separated by a carriage return and a newline
- Headers allow for arbitrary Key-value data pairs
    - In fact, you can have more than that
- Conventions are standardized

HTTP Methods

- GET
- POST
- PUT
- DELETE
- TRACE
- HEAD

Response Codes

- 1xx -- Information
- 2xx -- Success
- 3xx -- Redirect
- 4xx -- Client Error
- 5xx -- Server Error

200

:thumbsup:

200: Everything worked out.

3xx

Not here

MARIO
058700
×20
WORLD
1-4
TIME
200
THANK YOU MARIO!
BUT OUR PRINCESS IS IN
ANOTHER CASTLE!

4xx

You f*cked up

- 401 -- Unauthorized
- 402 -- Payment Required
- 403 -- Forbidden
- 404 -- Not Found
- 405 Method (verb) not allowed
- 418: I'm a teapot

5xx

I F*cked up

500 -- Internal Server Error

501 -- Not implemented

502 -- Bad Gateway

503 -- Service Unavailable

For more on statuses, see

For more see https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/

S: Secure

- We can strip this using a MITM proxy, by trusting a bogus CA cert.
- PKI Infrastructure is beyond the scope of this class
-

Setting up Burp Suite

DEMO

Process Enumeration

- EnumProcesses
- ToolHelp Functions
- Using WTS Functions
- Native API

Network I/O

- WinSOCK, WinTCP
- BSD sockets
- Boost