Reflective Injection Detection – RID.py

Or How I slapped together some python c-types in a week to do what defensive vendors aren't incorporating into main-line products

Or Bringing the Sexy Back - A defensive tool that doesn't fail quite as hard as it could

Pre-talk notes

- I don't own any of the artwork. It's all from google-images without copyright notices.
- I think fault lies on both sides of the fence.
- My statements here don't reflect my past, present, or future employer's point of view. Some thinks were inserted merely for humor.

Who Am I?

This page intentionally left blank

Who Am I?

On a more serious note:

You can check out my CV here on linkedin:

pub/andrew-king/23/432/679

Just a note, defense is first offense is last

Why would I do a talk that's going to make people angry?

If you want to make some enemies

Try to change something

Reflective Injection?

- Load DLL from memory
- VirtualAlloc or Ex
- No heap(might fragment)

Defense

- Programmers are lazy
  - Not just defensive programmers
- So there's probably very little 'request specific and check' going on
- Memory address allocations tend to be pretty predictable
- Possible optimization for scanning

Offense

- So what if they start looking for our PE mapping code?
- Just do the expansion on disk with some utility and now all that relocation code isn't needed.
- Vendors would rather search for reflect inject stager code I think...
- See some AV detects my obfuscation tutorial as malicious even though it prints hello...

FLOPPY DISK

Tha Gangsta way of storing dem filez

Why does it still work?

- Can we detect it at runtime?
- Only if we monitor VirtualAllocEx which seems really doubtful since all memory allocations eventually wind up there.
- Why it's not implemented in my opinion.
- Can we scan memory for it?
- Sure, that's easy.

Defense side of things

Finding reflect injected DLLs

- What does a DLL structurally have that raw data doesn't?
- PE header
- COFF headers
- Section tables
- Permissions
- Predictable layout

So first build a white-list

- Get all processes
- Get modules for all processes
- Build an exclusion map for yourself

VirtualQueryEx

- Find all allocated memory pages and save all the data about them.
- You never know, you might need it later

Process of elimination

- Eliminate all known legitimate pages
- Eliminate Thread areas
- There are more criteria I use to eliminate...go back to those things I said you might need later

ReadProcessMemory

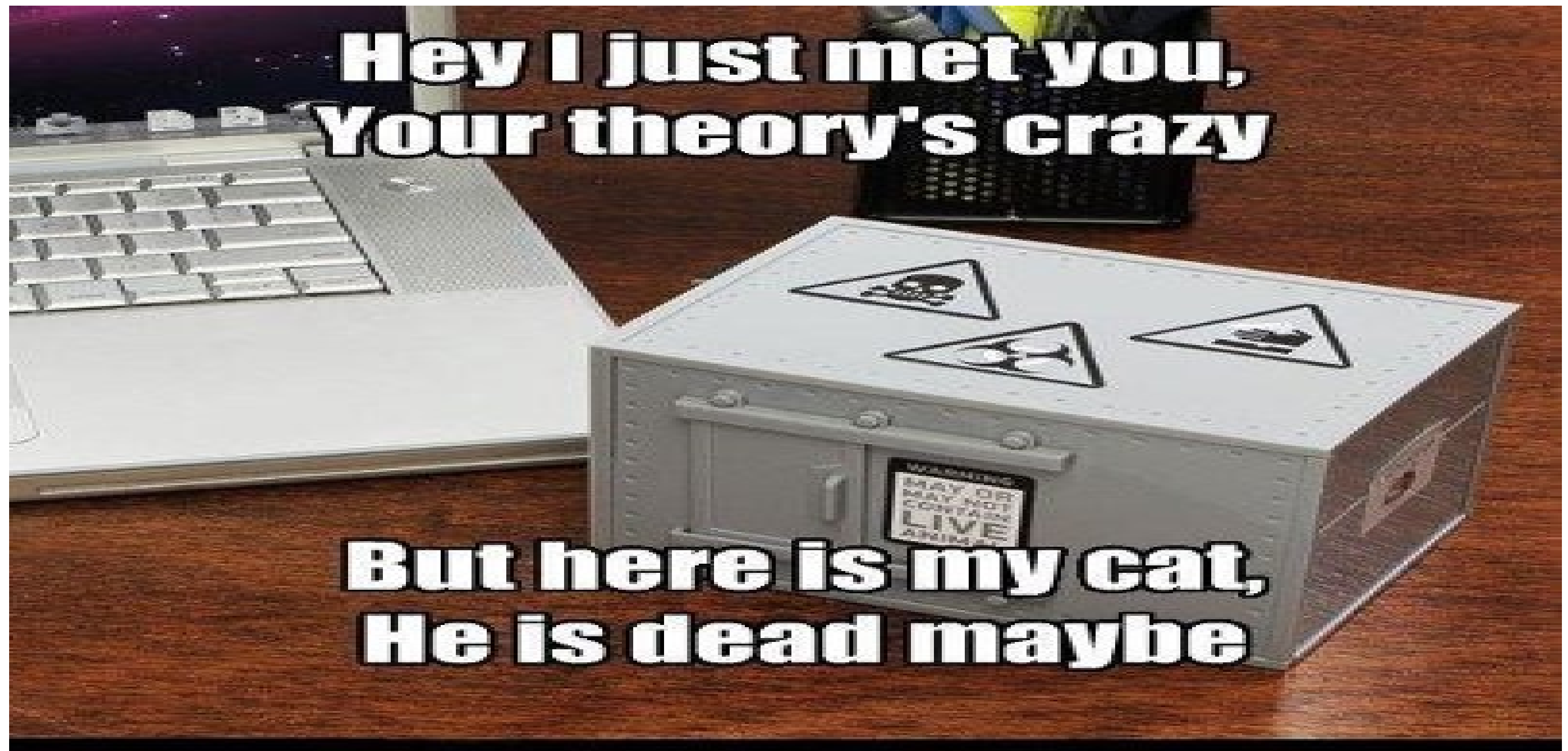
- Find the data in the relevant sections...
- Check for suspicious structures
- Check for fishy permissions
- Could check only probable allocation space...we're talking about shellcode here

So now that we found it

- What to do about it?
- Just flip access permissions so threads die when they try and execute there
- Suspend threads found to be operating in that area
- Dump the DLL
- Reverse the relocations
- Give it to your trusty old AV

Okay so that was easy, and not altogether new

People aren't doing it, but you never know...



Offense side of things

Offense side of things

- So how could we beat this?
- Load a large-ish DLL into memory that the process probably isn't going to use
- Carve it out and do some reflective injection into a targeted area.

Why don't more developers open source?

- Because people like this don't donate:

Hall of Shame

ocsiC

Demo

But then...

- Yes then A/V vendors would see your code.
- SO, you might want to think about run-time obfuscation
- Like I was talking about almost a year ago...

Conclusion

- Both sides of the fence on this one.
- Can it be halted/slowed down?
  - Sure
- Why isn't it?
  - It's kind of processor intensive to catch quickly.
  - Releasing some code.
  - You'll want to shim in the nice-ing up the processor bit

It's not pretty, but it works.

- Only tested on x86
- Doesn't have all the features that may be available in the dev train
- Yes, it's python with Ctypes
  - I have a C port. It's much easier on memory, but much slower...I used lists instead of C++ maps.

The other code

- I stripped out all the lists of possible structures and put in a basic regex for metasploit shellcode instead of section offsets
- I couldn't really in good conscience put a fully weaponized thing out there

What else am I working on?

- Interesting things with python obfuscation
- Shout outs
- Thanks

Questions?

- Yes, I'll be around