Exploit Development

Vulnerability Development as Program Composition

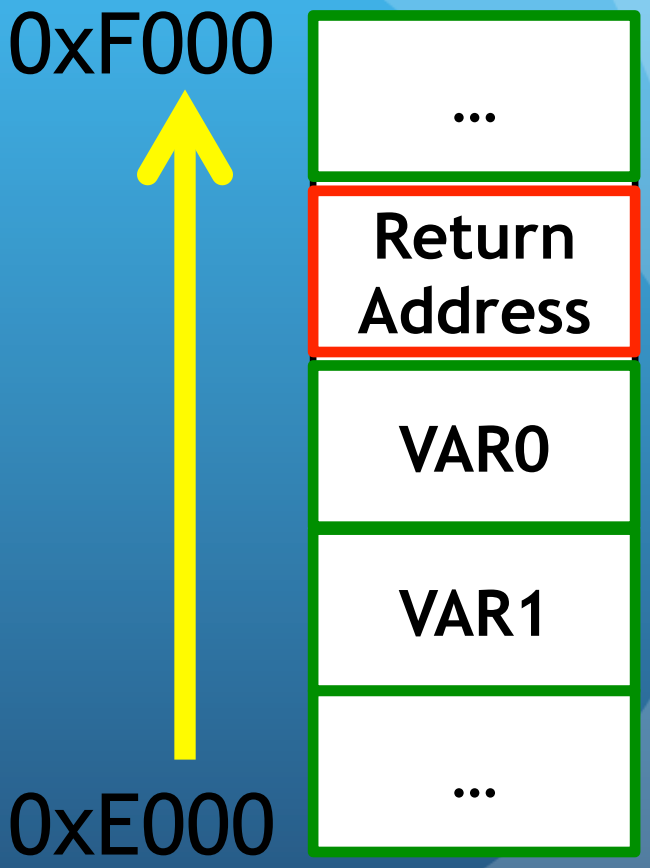
Stack Overflows

- For an explanation of stack overflows, I will use some slides from my friend Artem Dinaburgs guest lecture at GATech.
- Artem is a cool guy and you should look him up online later

What is a Stack Overflow?

- Write past the end of a stack buffer
- Overwrite return address
- Hijack control flow on return

Write direction in yellow.



Exploiting a Stack Overflow: 1996

- Control Flow
  - Overwrite return address (always static)
  - Jump to shellcode (static location)
- Shellcode
- Persist
- Knowledge: None
- Bugs Needed: 1

Exploiting a Stack Overflow: 1996

Defense: Stack Cookies

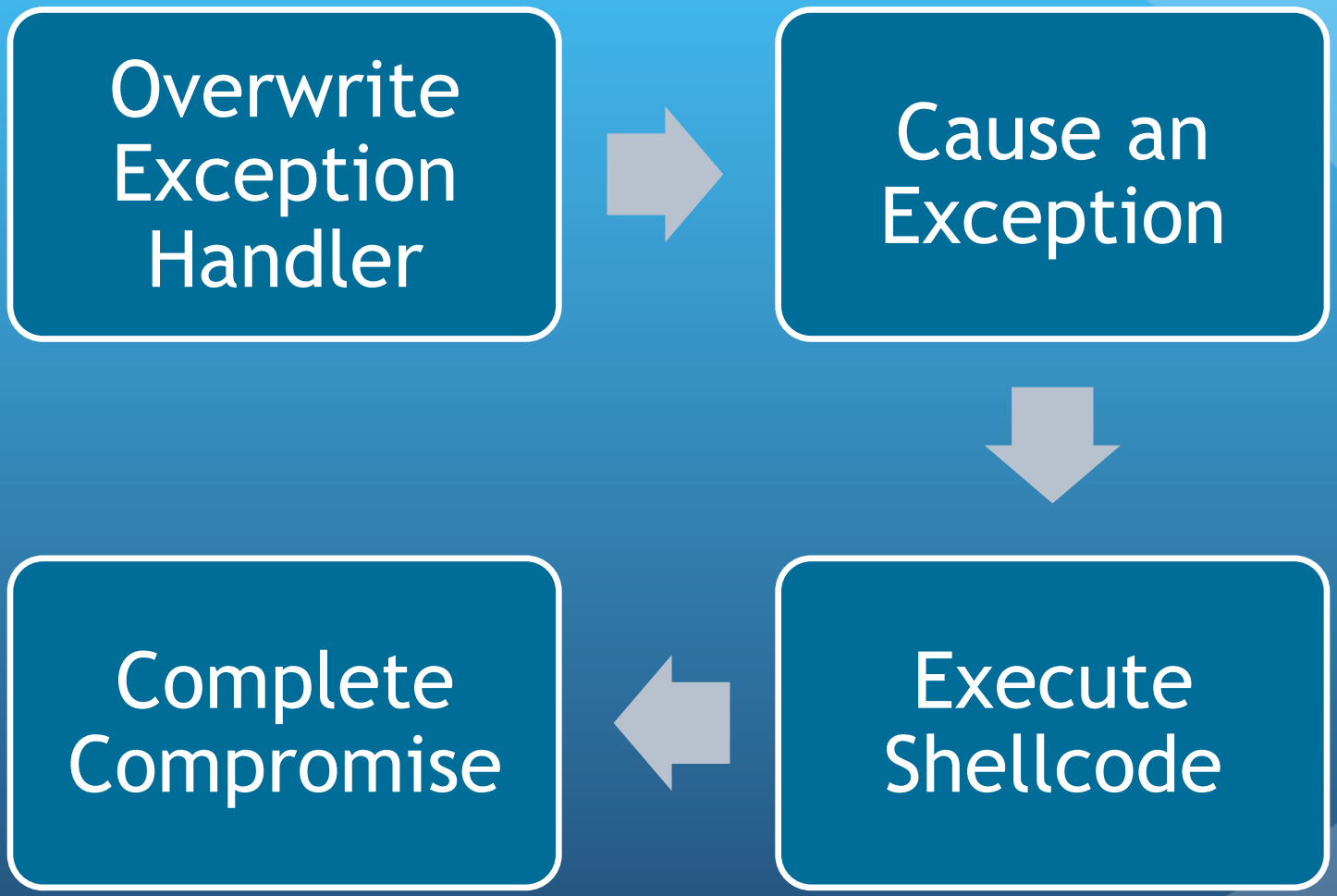
- Goal: Prevent control flow hijacking via return address overwrite.
- Add secret value before return address
- Verify the value before jumping to return address
- First described in 1998
- Introduced in GCC and MSVC in 2003.

Exploiting a Stack Overflow: 2003 StackCookies

- Control Flow:
  - Guess stack cookie (very hard)

  OR
  - Exception handler (Windows)
  - Overwrite function pointer (rare)
- Shellcode
- Persist
- Knowledge Needed: Stack Cookie
- Bugs Needed: 1 - 2

Exploiting a Stack Overflow: 2003



Defense: DEP

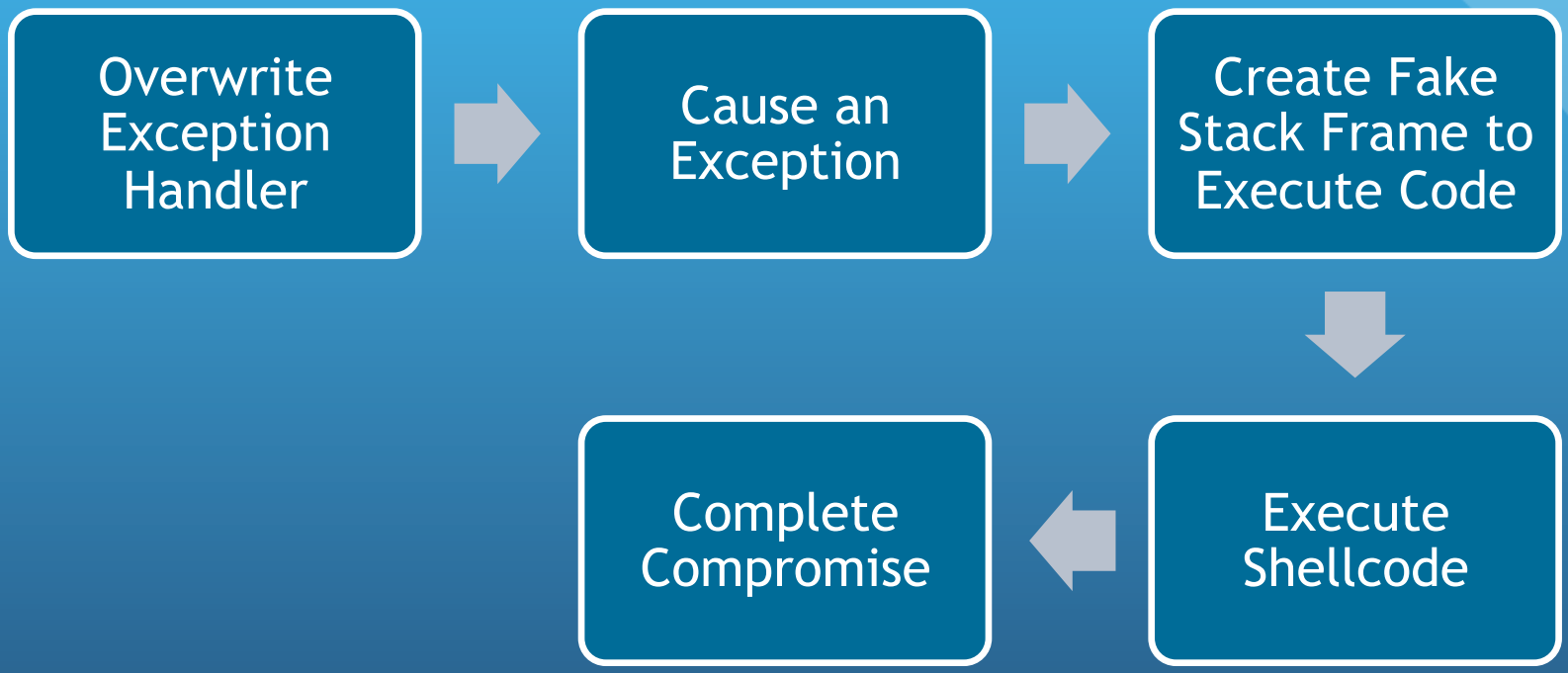
- Goal: prevent introduction of foreign code.
- Mark memory pages as non-executable by default.
- Protections always existed, just not enforced.
- New hardware development.
- Mainstream Windows & Linux support in 2004.

Exploiting a Stack Overflow: 2004 DEP

- Control Flow:
  - Guess stack cookie (very hard)

    OR
  - Exception handler (Windows)
  - Overwrite function pointer (rare)
- Shellcode:
  - Find address of APIs (easy – if they are imported)
  - Create stack frame to +X buffer (easy)
  - Execute shellcode
- Persist
- Knowledge Needed: Stack Cookie
- Bugs Needed: 1 to 2

Exploiting a Stack Overflow: 2004



Defense: ASLR

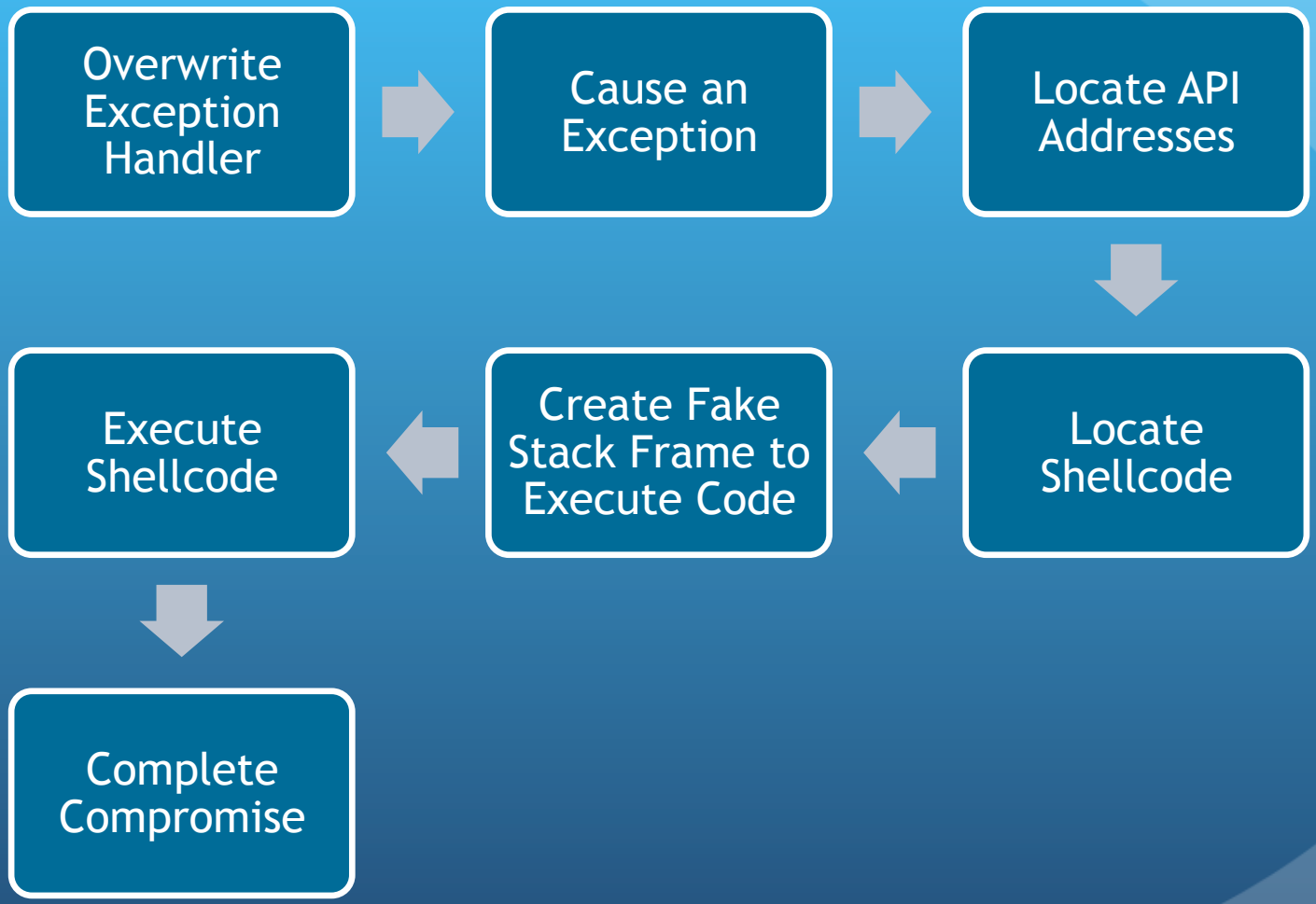
- Goal: prevent the attacker from re-using code and locating their own malicious payload.
- Randomize load locations of libraries and the program image.
- Randomize stack addresses
- Randomize heap addresses

Exploiting a Stack Overflow: 2007 ASLR

- Control Flow:
  - Guess stack cookie (very hard)

    OR
  - Exception handler (Windows)
  - Overwrite function pointer (rare)
- Shellcode:
  - Find address of APIs (hard – if they are imported)
  - Create stack frame to +X buffer (hard)
  - Execute shellcode
- Persist
- Knowledge Needed: Stack Cookie, Module Address, Shellcode Location
- Bugs Needed: 2

Exploiting a Stack Overflow: 2007



SafeSEH/SEHOP

- Goal: prevent control flow hijacking via exception handlers.
- Two different technologies to verify integrity and semantics of exception handlers.

Exploiting a Stack Overflow: 2008 SafeSEH/SEHOP

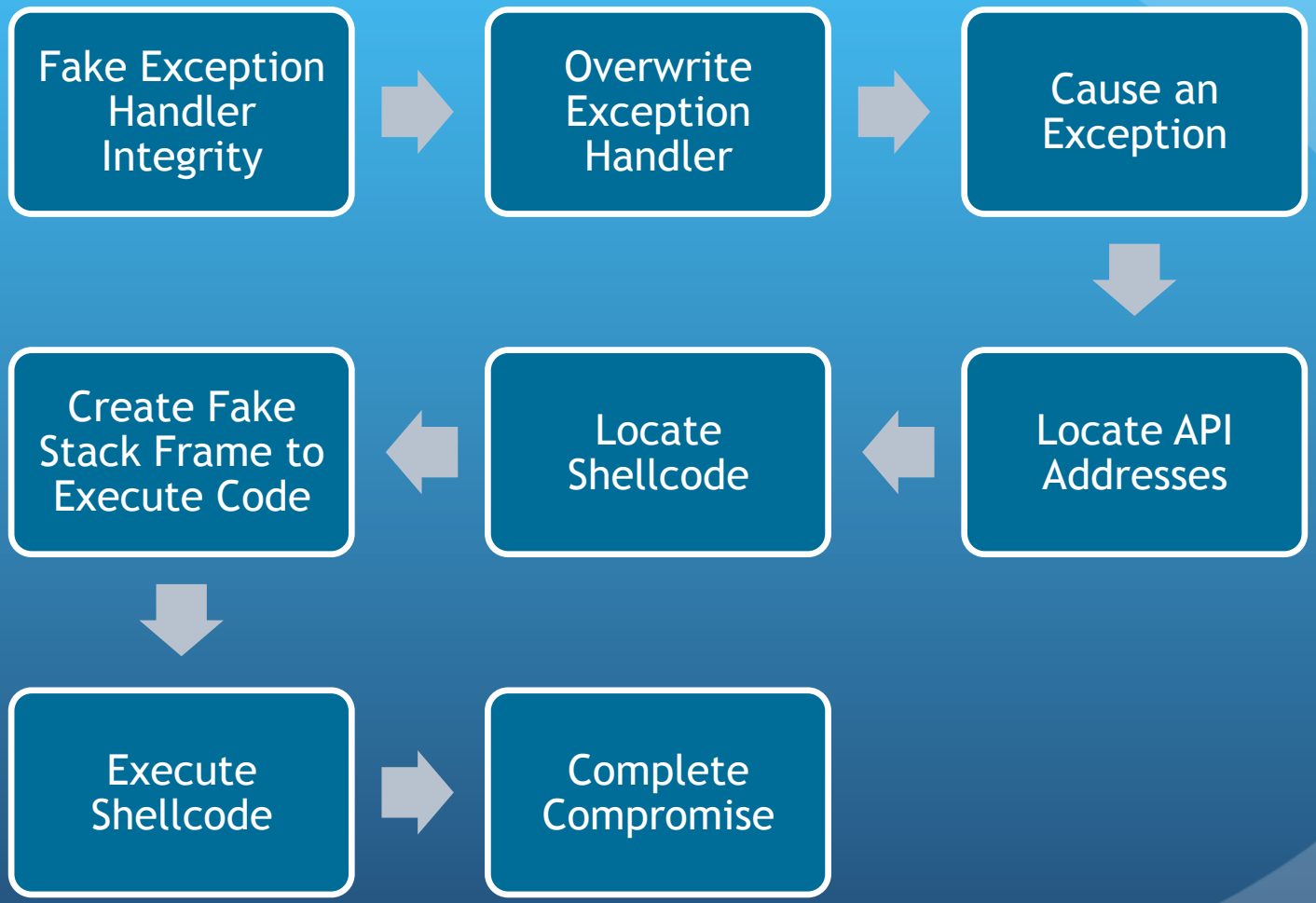
- Control Flow:
  - Guess stack cookie (very hard)

  OR
  - Overwrite function pointer (rare)
- Shellcode:
  - Find address of APIs (hard – if they are imported)
  - Create stack frame to +X buffer (hard)
  - Execute shellcode
- Persist
- Knowledge Needed: Stack Cookie, Base Address, Shellcode Location
- Bugs Needed: 2

Exploiting a Stack Overflows: Interlude

- In reality, attackers just stopped exploiting stack overflows.
- Moved on to greener, more exploitable, pastures.
- But, hypothetically speaking…

Exploiting a Stack Overflow: 2008



Sandboxing

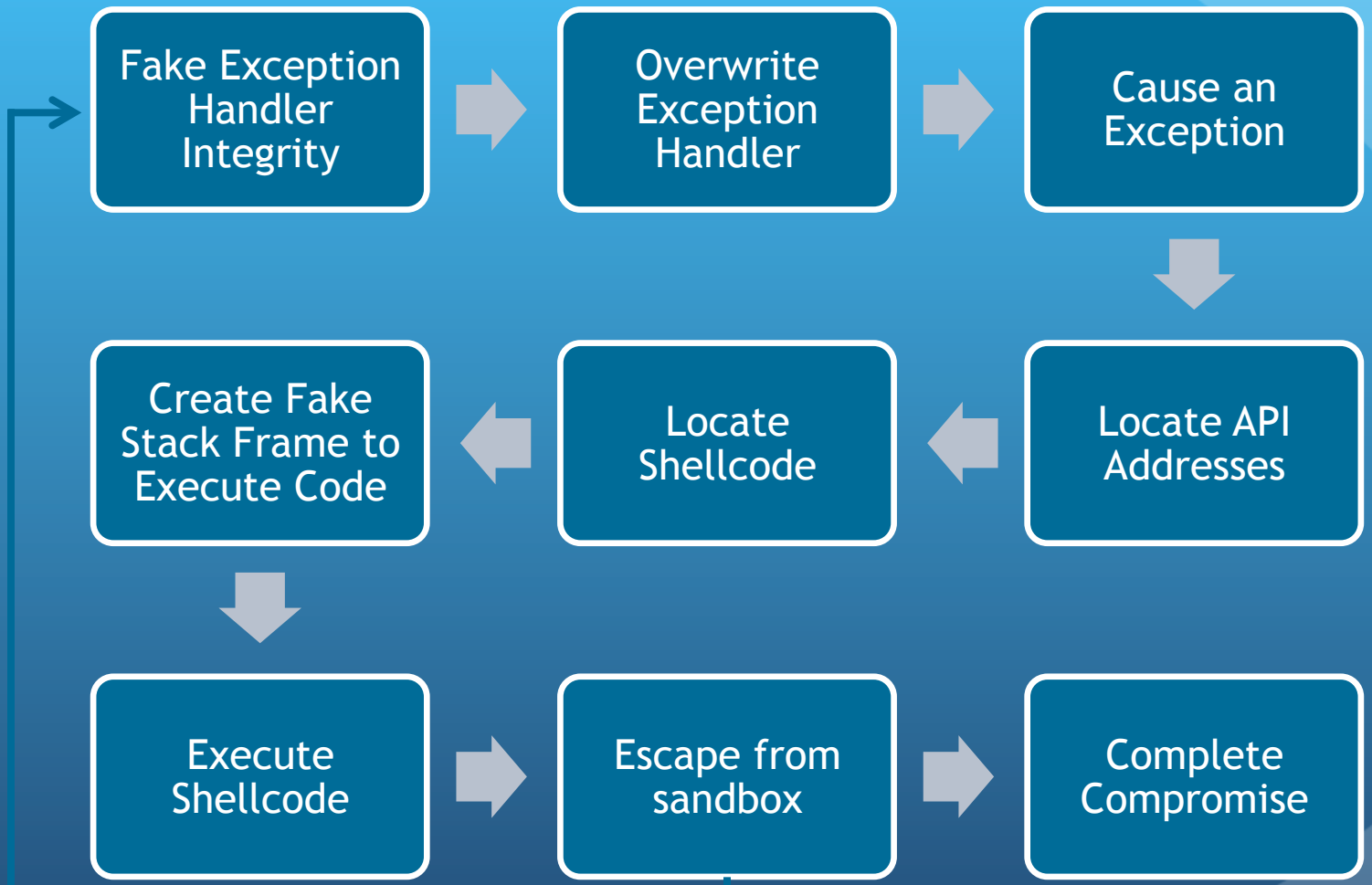
- Goal: Prevent application compromise from leading to complete system compromise.
- Separate application privileges
- Limit what APIs applications may call
- Mandatory Access Control

Exploiting a Stack Overflow: Present Sandboxing

- Control Flow:
  - Guess stack cookie (very hard)

  OR
  - Overwrite function pointer (rare)
- Shellcode:
  - Find address of APIs (hard – if they are imported)
  - Create stack frame to +X buffer (hard)
  - Execute shellcode
- Persist
  - Sandbox Escape
- Knowledge Needed: Stack Cookie, Base Address, Shellcode Location, Sandbox Escape
- Bugs Needed: 3

Exploiting a Stack Overflow: Present

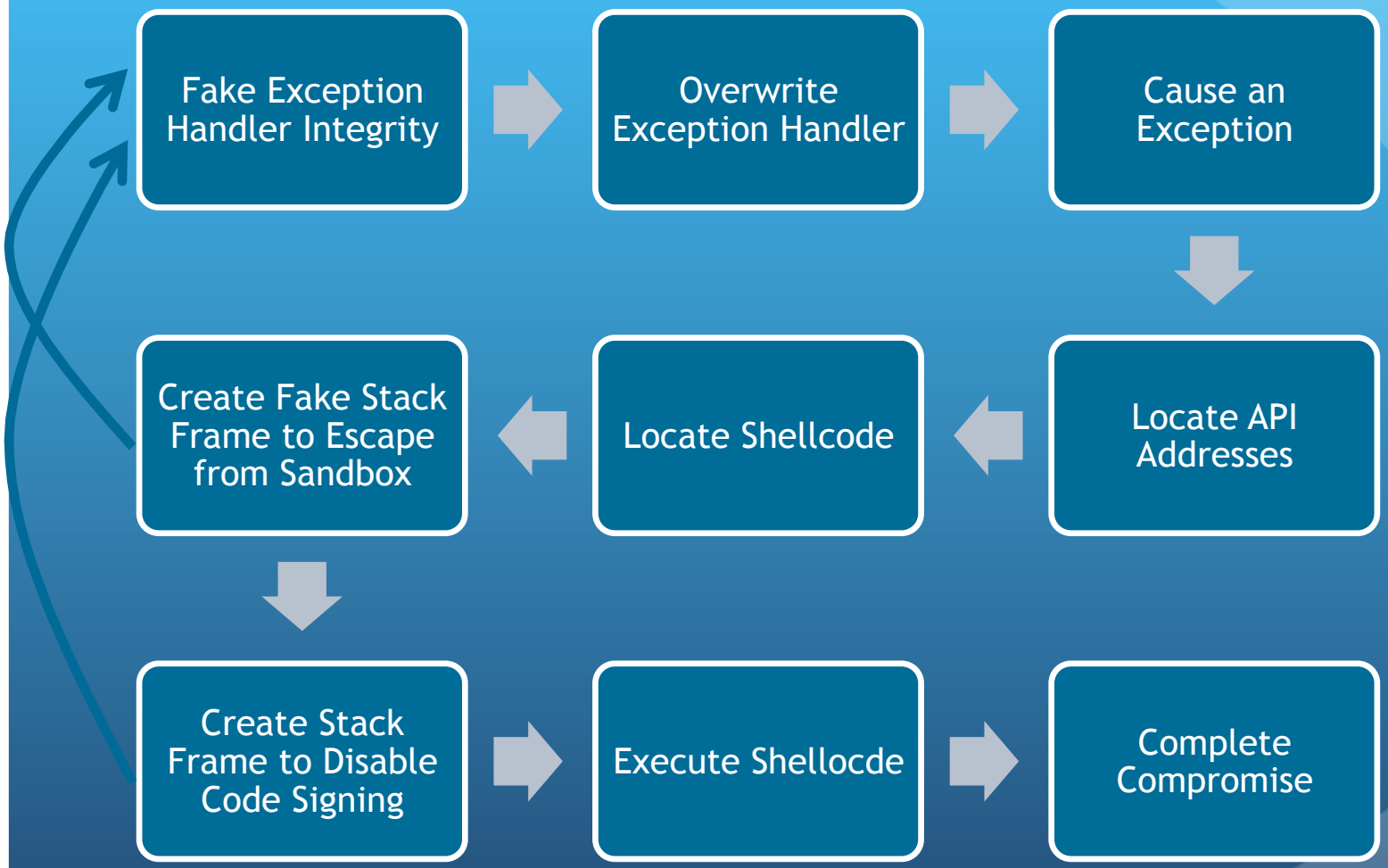


Exploiting a Stack Overflow: Soon Mandatory Code Signing

- Control Flow:
  - Guess stack cookie (very hard)

  OR
  - Overwrite function pointer (rare)
- Shellcode:
  - Find address of APIs (hard – if they are imported)
  - Create stack frame to disable code signing (very hard)
  - Allocate executable buffer
  - Execute shellcode
- Persist
  - Sandbox Escape
- Knowledge Needed: Stack Cookie, API Address, Shellcode Location, Code Signing Exploit, Sandbox Escape
- Bugs Needed: 4+

Exploiting a Stack Overflow: Soon



Stack overflows: wrap-up

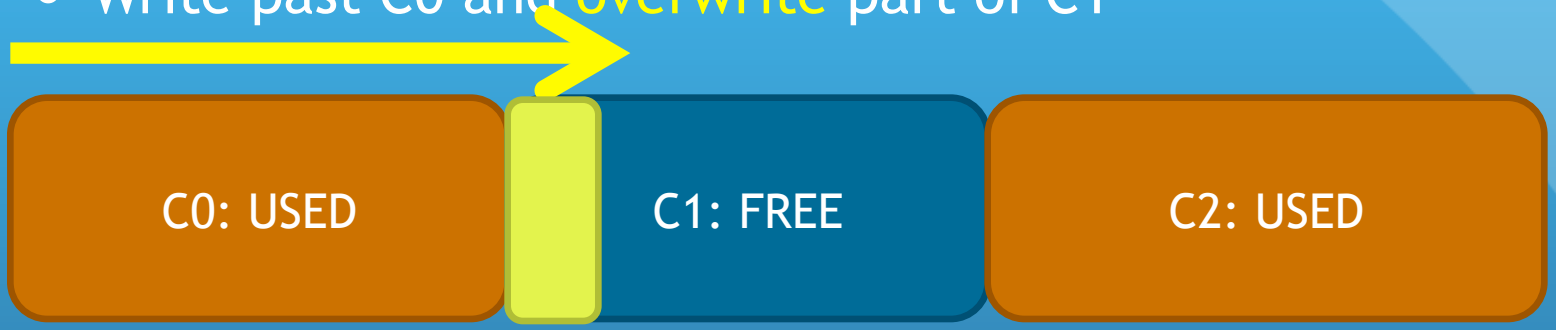
- This is the progression of stack overflow stuff on Windows
- Let's look at some of the telltale signs of stack canaries
- What are some ways that you could do a stack canary incorrectly?
- Would this be an amusing CTF challenge?
- What is the concept that this all reduces to?

What about heap overflows?

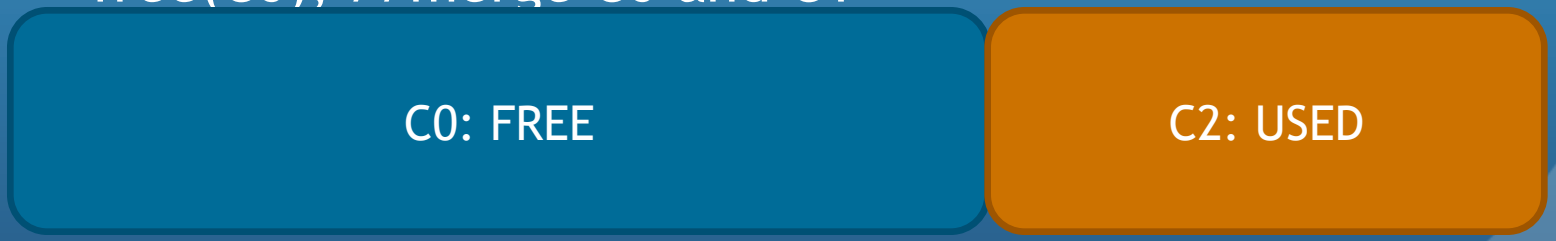
- Oh boy
- Inherently more complicated
  - An additional layer of indirection
- Again, some help from Artem

Heap Overflows

- Write past C0 and overwrite part of C1



- free(C0); //merge C0 and C1



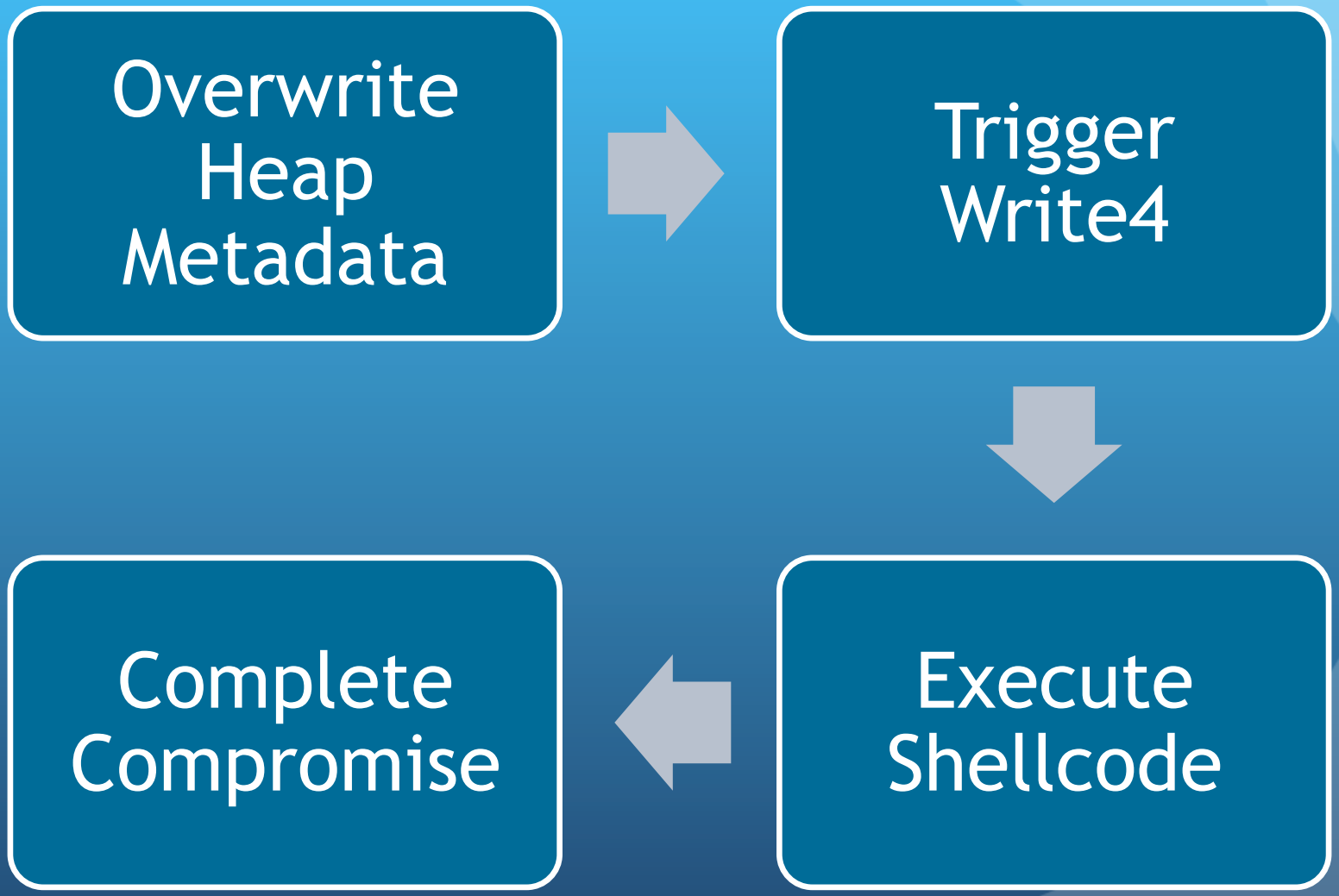
Heap Overflows

- Overwrite heap metadata to gain code execution.
- Get a write-what-where
  - C0->blink->flink = C1->flink
  - C1->flink->blink = C0->blink
- *(what+4) = where
  *(where+0) = what

Exploiting Heap Overflows 2004

- Control Flow
  - Chunk Coalesce Write-What-Where
- Shellcode
- Persist
- Knowledge:
  - Memory Layout (relatively static)
  - Writeable Address (data section)
  - Function Pointer (static via import table)
- Bugs Needed: 1

Exploiting a Heap Overflow: 2004



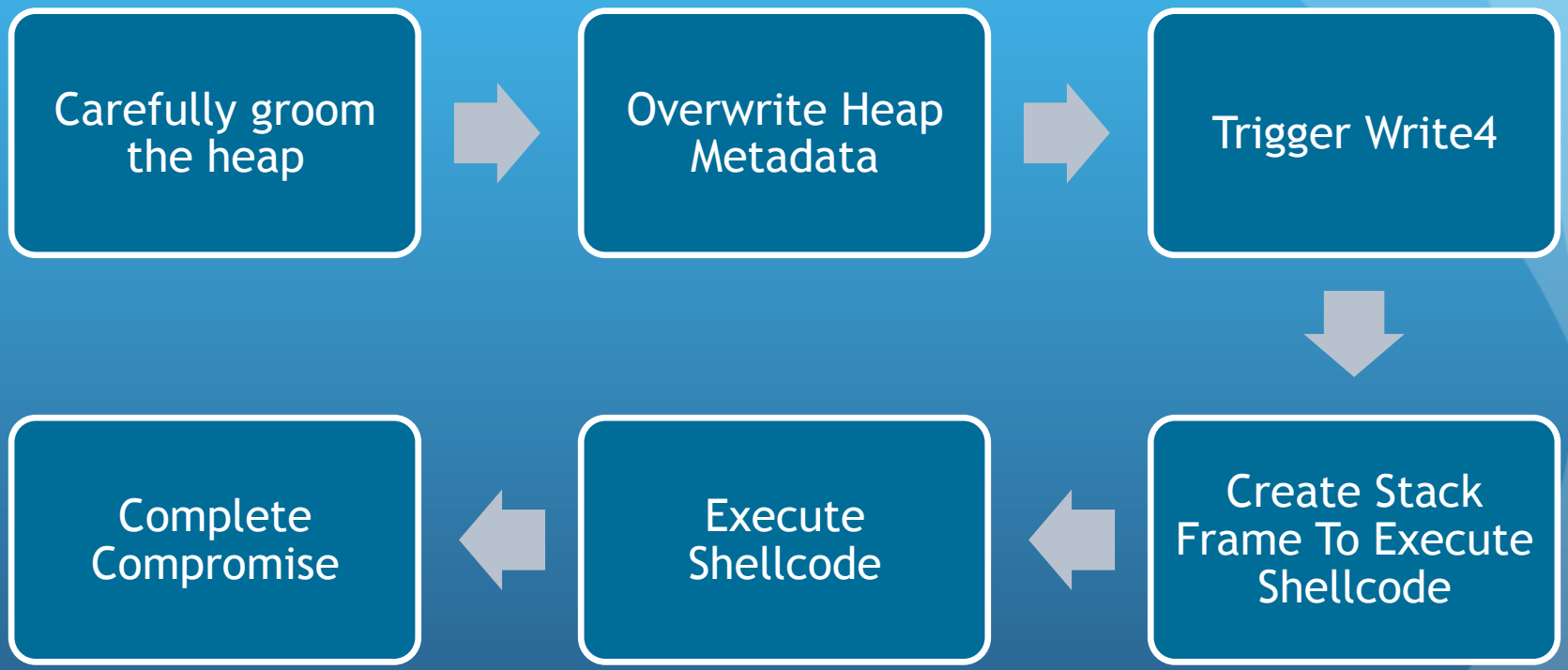
Heap Protections

- Safe Unlinking
  - Not in popular use until 2007
  - entry->blink->flink == entry
  - entry->flink->blink == entry
- Non-Executable Heap (DEP)
- Heap Cookie

Exploiting Heap Overflows 2007

- Control Flow
  - Complex Metadata Overwrites
  - Function Pointer
- Shellcode
  - Need stack pivot (control of address to control of stack)
- Persist
- Knowledge:
  - Location of stack pivot (static, if it exists)
  - Memory Layout (relatively static)
  - Writeable Address (data section)
  - Function Pointer (static via import table)
- Bugs Needed: 1-2

Exploiting a Heap Overflow: 2007



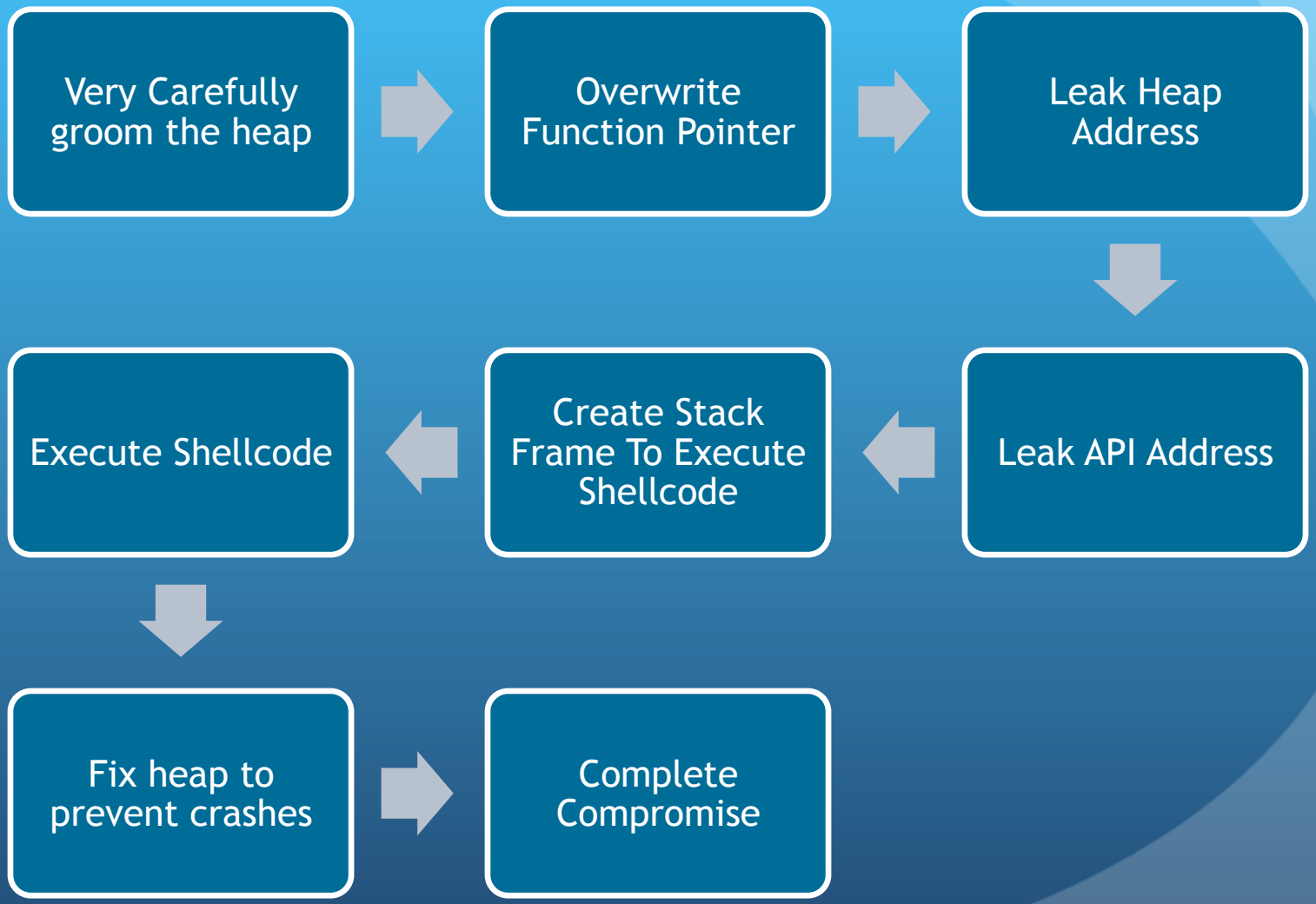
Heap Protections

- Heap Address Randomization
- More Metadata Verification
- Termination on Metadata Inconsistency
- Metadata Encoding
- Pointer Encoding
- ASLR

Exploiting Heap Overflows 2012

- Control Flow
  - Ridiculously Difficult Metadata Overwrites
  - Function Pointers
- Shellcode
  - Need stack pivot
- Persist
- Knowledge:
  - Address of stack pivot (random)
  - Very Specific Memory Layout (very hard)
  - Writeable Address (random)
  - Function Pointer (random)
- Bugs Needed: 2

Exploiting a Heap Overflow: 2007



Current Heap Protections

- Separate data and metadata
- Guard pages between allocations
- Add random offsets to allocated data
- Randomize allocation algorithms to prevent deterministic heap layouts.

Exploiting a Heap Overflow: Current

- The diagram would be too big to show here, but its really really hard.
- Hope for application data overwrite or a function pointer overwrite.
  - C++ objects come with a lot of function pointers.

In CTFs

- CTFs frequently use reduced versions of real-world problems
- Having contestants attack real-world software would be
  - Boring
  - Very very difficult
  - Too much like real work
- Upside: Don’t have to work through ginormous attack trees
- Downside: exploiting is more solving a “puzzle” that is mostly in the head of the problem author

Exploit development workflow

- Exploit development starts when you get a crashing input and a desire to get a shell
- Use a debugger
- Write things down and draw diagrams
- Account for things you know about the conditions in the target program
- In the real world, we don’t like to leave things to chance
- In CTF, 60% of the time, it works every time
- Now is better than later

Exploit development workflow

- Have some notes
- Have a script that produces the crash
  - Describe as much as you can, to yourself, about how input data reacts with the vulnerable program
- Work iteratively to get control over crashes
- Vulnerability identification is generally a solo activity, exploit development can be more conductive to pairing up

Thought process

- Crash – read or write?
- What object was being acted upon?
- How much control do I have over the source / destination?
- What objects are adjacent to where I can control reading / writing?

General theme

- Control flow hijacking exploits have a common theme
  - Find a way to write into something you shouldn't
  - Find how to connect that write to some code sequence that does what you want
  - If no single primitive exists to do last two, find composition of primitives that does
  - If no such combination of primitive exists either
    - You are looking at the wrong thing
    - You are not doing CTF and have started doing this for a living. My condolences