Lec 20: Anti-malware

CSED415: Computer Security

Spring 2025

Seulbae Kim

POSTECH

POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Administrivia

- Lab 4 is due tomorrow!
- No in-class meeting next week!
  - Tue, May 6: Children’s day (National holiday)
  - Thu, May 8: POSTECH festival
- Lecture videos will be posted
  - Lab 3-4 review, Mid-term exam review

Recap

- Malware
  - Malicious program that is covertly inserted into a system with the intent of compromising CIA
- Malware of our interests:
  - Virus, worm, trojan, rootkit, backdoor, spyware, bots, and ransomware
- Anti-malware / Anti-virus (AV) is a software or technique that aims to protect our systems from malware

Anti-virus (AV) / Anti-malware

Fred Cohen's paradox

- Given an arbitrary program, can we design a function that determines whether the program is malicious or not?
  - $f(prog) = \begin{cases} 1 & \text{if } prog \text{ is malicious} \\ 0 & \text{otherwise} \end{cases}$

Fred Cohen's paradox

- Define function `is_virus`
  - Input: A program
  - Output: `1` if the program is a virus, `0` if not

```
def is_virus(prog):
    # test prog and return 1 or 0
```

Assume such a function actually exists

Fred Cohen's paradox

- Construct a program named `real_virus`:

```
# real_virus.py

if is_virus("real_virus.py"):
    return # do nothing

else:
    infect_other_prog() # viral activity
    destroy_user_data()
    return
```

real_virus is a self-contradictory program!

Fred Cohen's conclusion

- Virus detection is an undecidable problem
  - Undecidable: Proved to be impossible to construct an algorithm that always correctly determines the answer
- Since the detection is an undecidable problem, the removal of virus can never be guaranteed
  - You must first detect before you can remove
  - AV research focuses on practical but incomplete solutions

Then, how can we detect malware?

Naïve approach for malware detection

- Goal: Check whether a file is identical to a known malware
  - If a file matches a known malware byte-for-byte, it should be classified as malware
- Method: Signature (hash) matching
  - Collect known malware samples (e.g., worm binaries)
  - Create signature DB by computing hash for each sample
  - Hash the target file
  - Compare hashes
    - Match found → classify the file as malware
    - No match → unknown (need more investigation)

Naïve approach for malware detection

- Problem of hash matching: Too many ways to bypass
  - Add dummy code
    - A function that is not called
    - A function that does nothing significant
    - nop instructions
  - Change code order (e.g., define function A after B / B after A)
  - Replace instructions with semantically equivalent ones
    - e.g., `inc eax` → `add eax, 1`

A difference in a single bit results in totally different hash values

Another naïve approach for malware detection

- Pattern matching
  - Match using regular expression (RE)
  - e.g., bytecode of an `execve("/bin/sh");` shellcode
    - ```
      ...
      6a 0b   push 0xb
      58      pop eax
      cd 80   int 0x80
      ```
    - RE pattern: `(\x6a\x0b\x58)(.*)(\xcd\x80)`

(1) push 0xb
pop eax

(2) anything

(3) int 0x80

Matches any bytecode that has (1), (2), and (3)
      ```

Another naïve approach for malware detection

POSTECH

- Problem of RE matching: Still easy to bypass
  - RE pattern: `(\x6a\x0b\x58)(.*)(\xcd\x80)`
  - Easy to generate semantically identical code to `push 0xb; pop eax;`
    - `mov eax, 0xb;`
    - `mov eax, 0xa; inc eax;`
    - …

    → The above RE pattern misses these

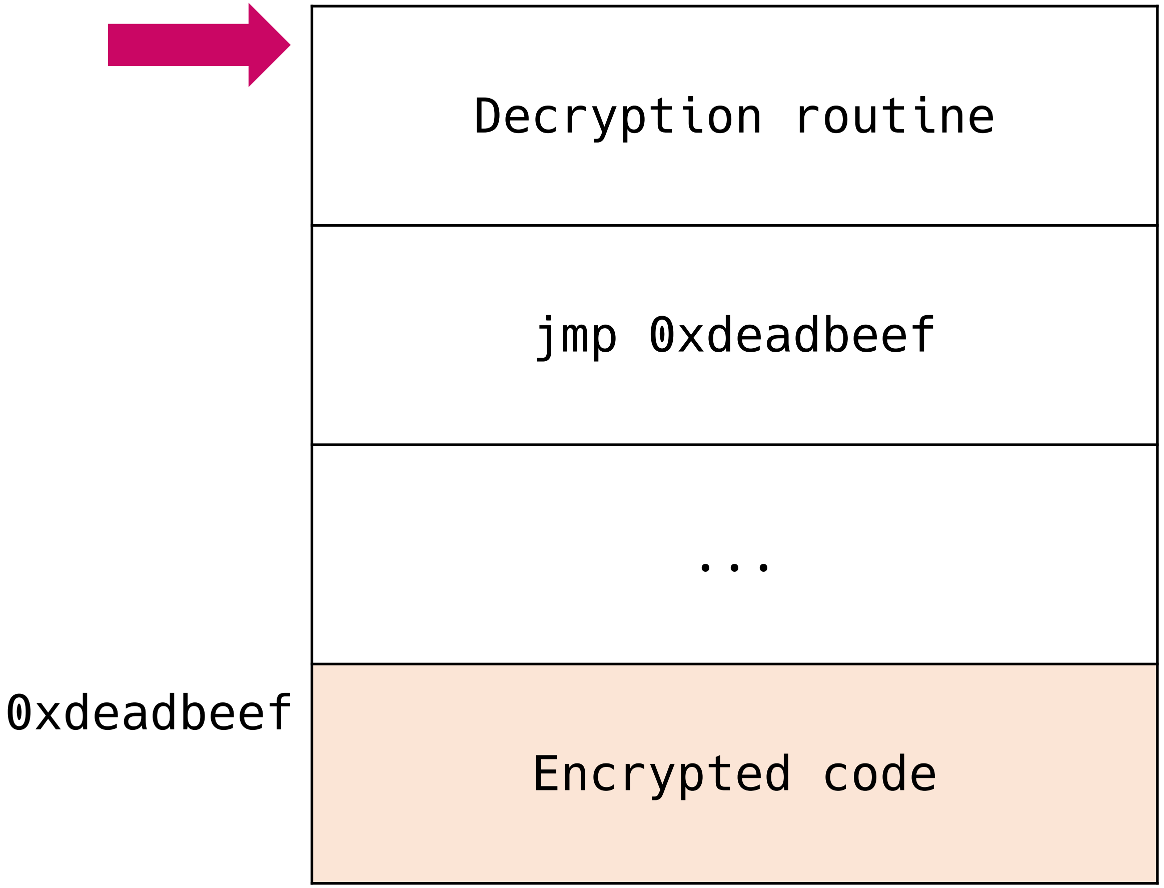
Recent malware utilize "self-modifying code" to make pattern-based detection even more challenging

Polymorphism

Polymorphic code

- Definition:
  - A code that mutates itself to change its appearance while keeping the original algorithm intact
  - Malware often employ polymorphism to bypass signature matching- or pattern matching-based Avs
- How?
  - Malware encrypts its malicious code with a random key and carries it as a payload

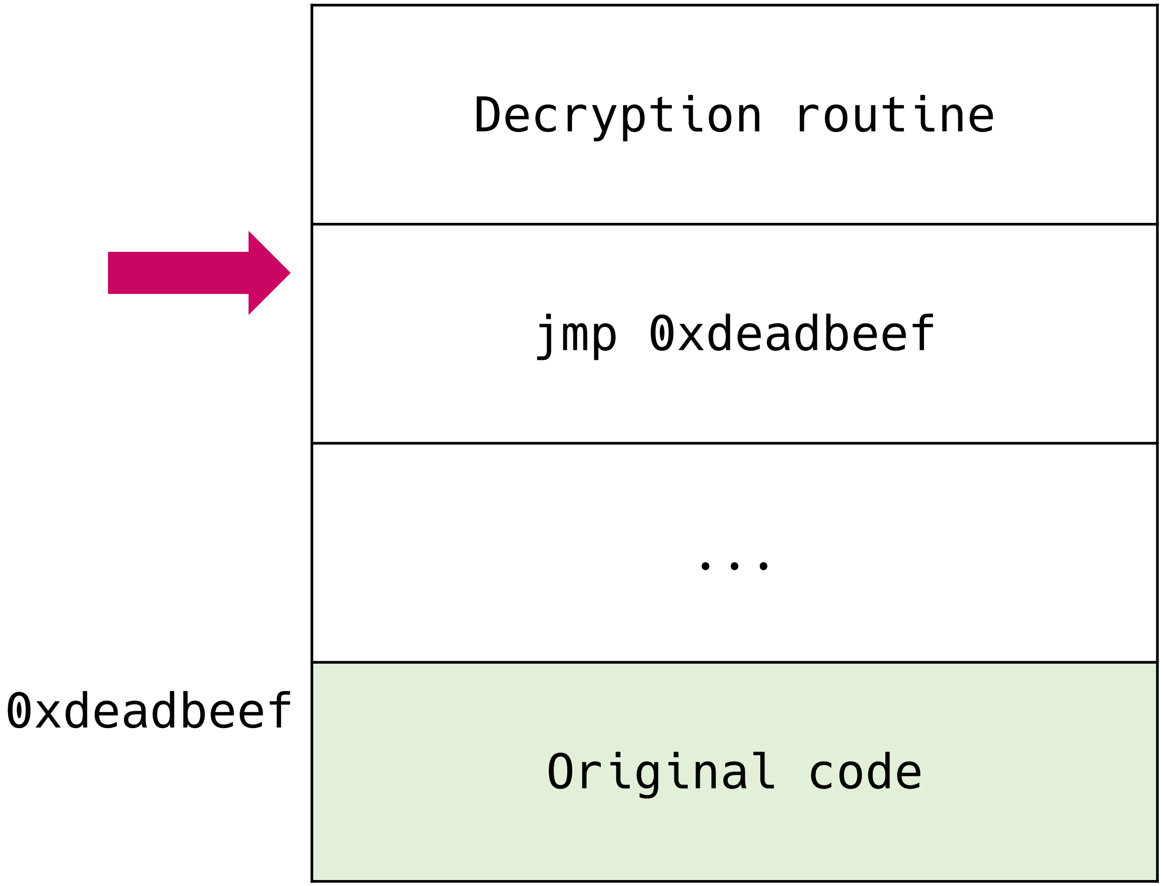
Polymorphism example



Reads the encrypted code and decrypts it

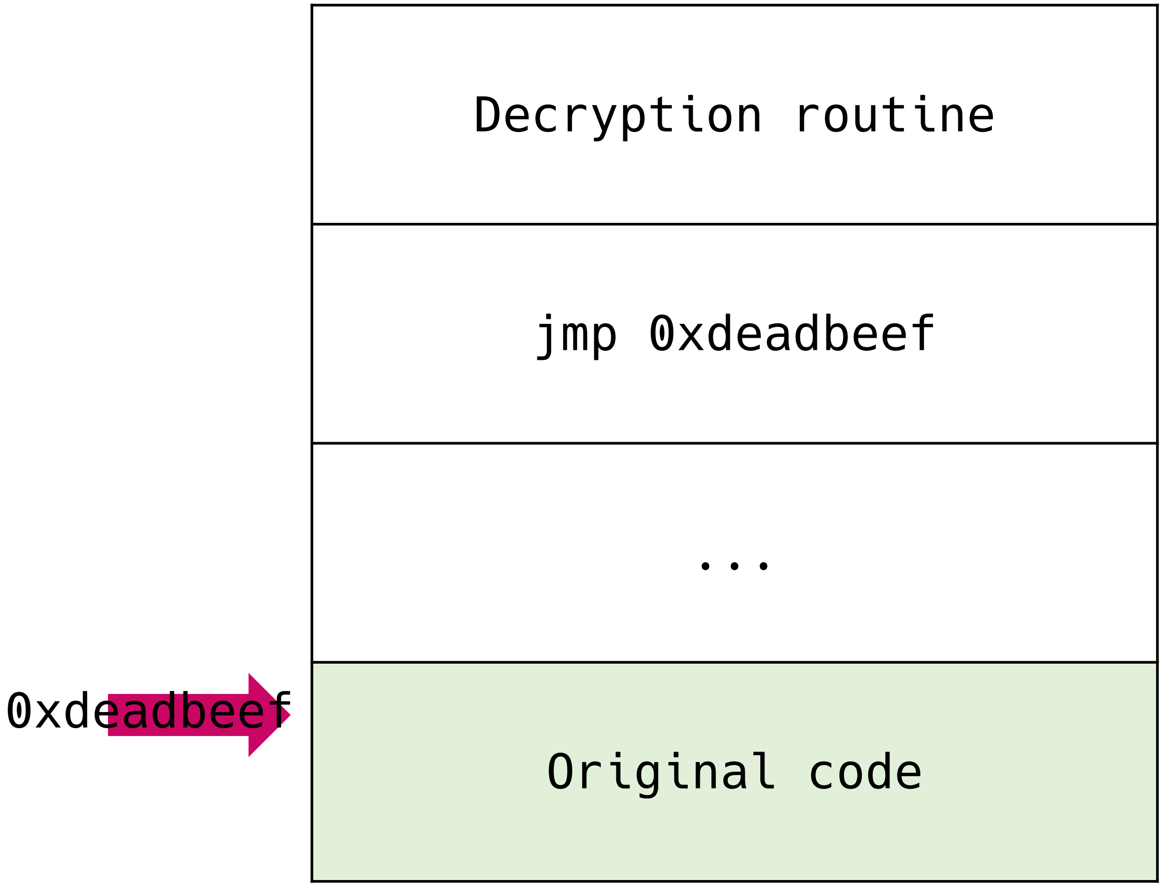
Stores the result at the location where the encrypted code was stored

Polymorphism example



Jumps to `0xdeadbeef`,
i.e., original entry point

Polymorphism example

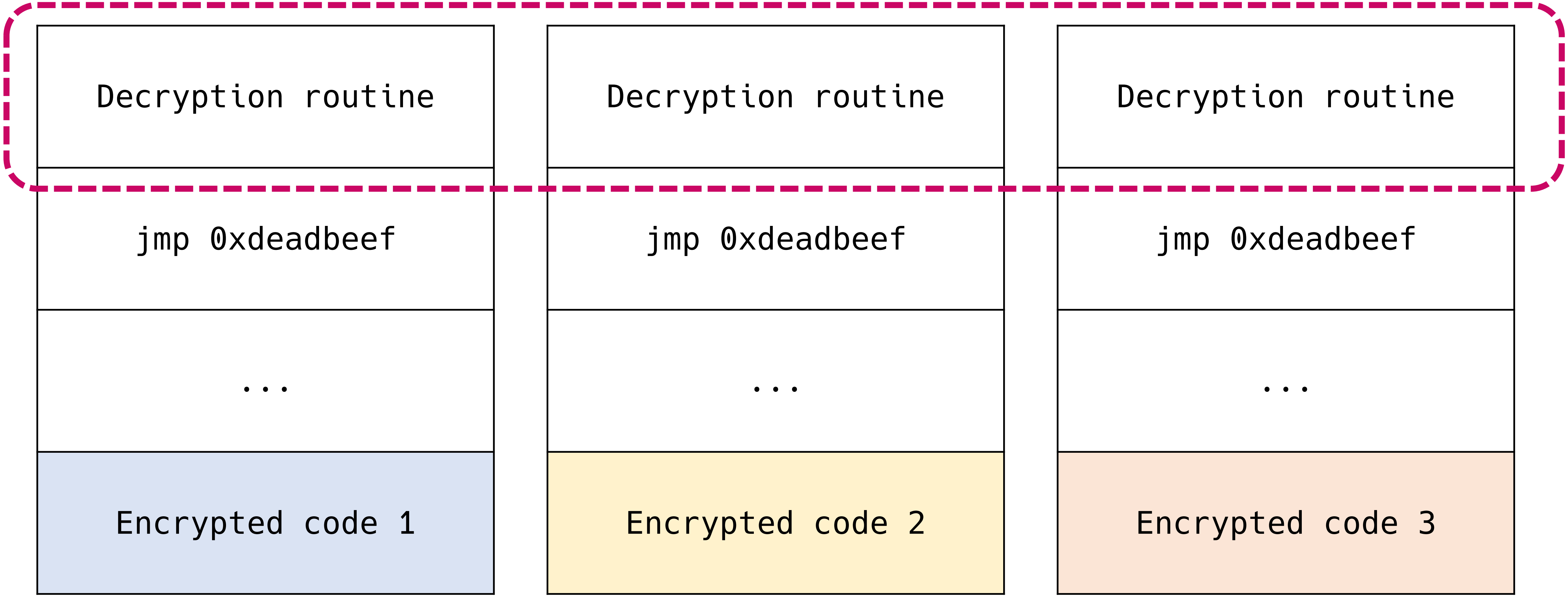


We can produce unlimited number of semantically identical binaries that have different signatures (e.g., hash) by just changing the encryption key

Original malware code is executed

Creating partial signatures

This part does not change → AVs can create signatures of the decryption routine



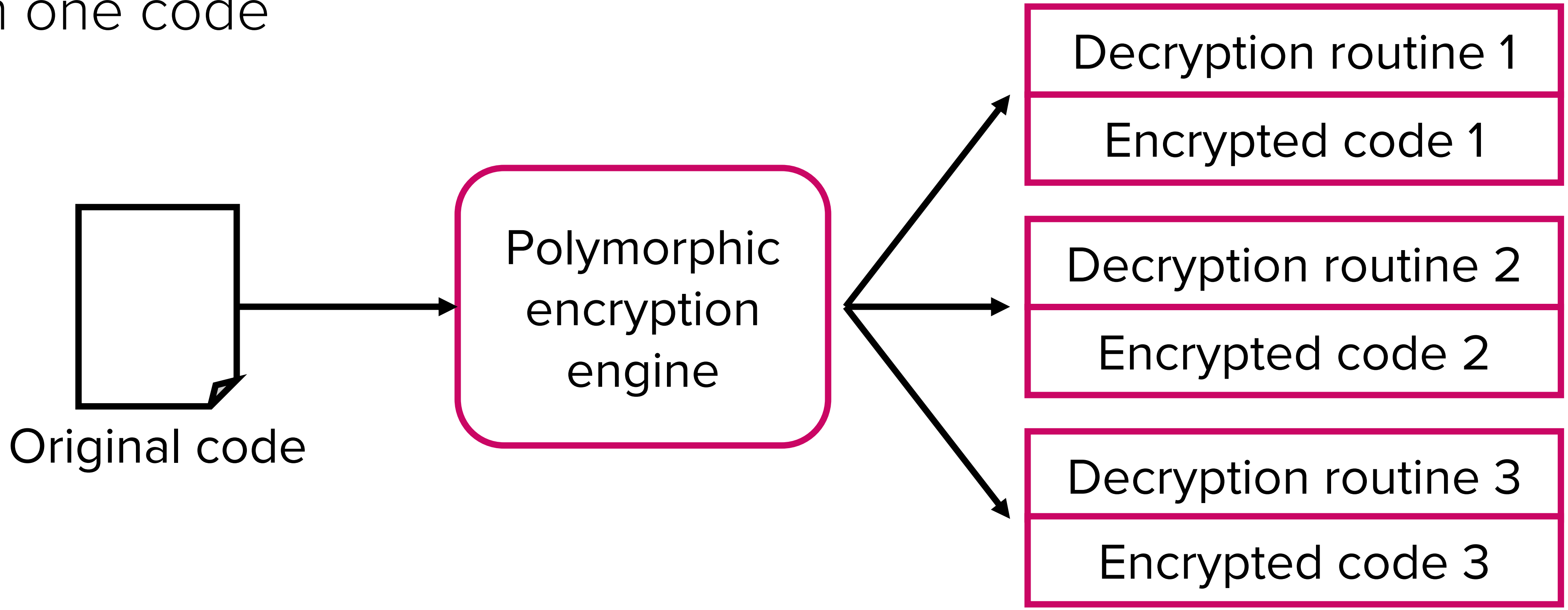
Creating partial signatures

`Decryption routine`	`Decryption routine`	`Decryption routine`

Can polymorphism be applied even to the decryption routine?

Polymorphic encryption

- Goal:
  - Creating multiple unique pairs of encryption and decryption routines from one code



Polymorphic encryption example

```c
for (int i = 0; i < code_len / 4; ++i) {
  v = obc[i]; // obc: int array containing the original bytecode
  key[i] = random_int(); // random 4-byte integer
  op[i] = random_op(); // randomly select encryption operation
  switch (op[i]) {
    case ADD: v += key[i]; break;
    case SUB: v -= key[i]; break;
    case XOR: v ^= key[i]; break;
    /* ... */
  }
  enc[i] = v; // enc: int array containing the encrypted code
}
```

Polymorphic decryption example

```
for (int i = 0; i < code_len / 4; ++i) {
  v = enc[i]; // for every 4-byte of the encrypted code
  k = key[i]; // retrieve the key
  switch (op[i]) { // decrypt each encrypted byte by inverting op
    case ADD: v -= k; break;
    case SUB: v += k; break;
    case XOR: v ^= k; break;
    /* ... */
  }
  dec[i] = v; // store decrypted (original) code in dec
}
```

→ Unroll (i.e., flatten) the loop and embed to malware as decryption routine

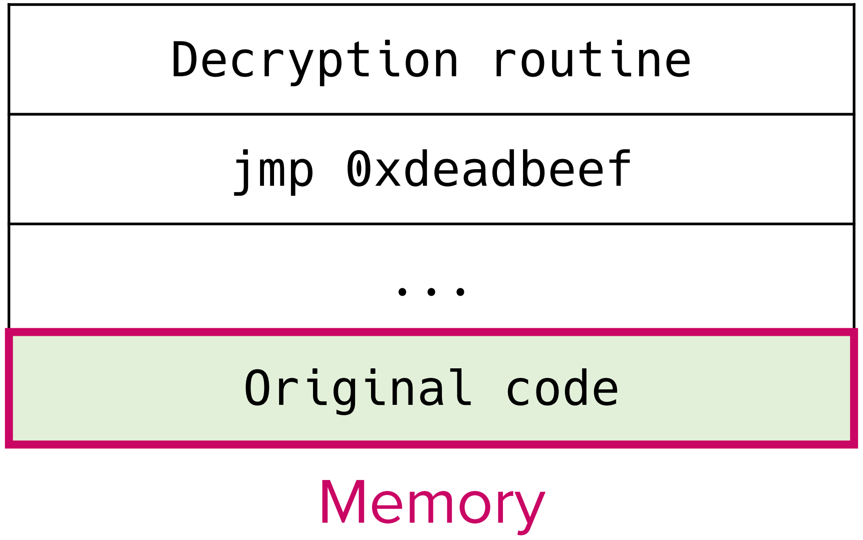
Signatures for polymorphic encryption

- Using polymorphic encryption, millions of variants can be created from a single malware
- Signature database of an AV will rapidly expand if all possible variants are considered
- Signature-based pattern matching does not help anymore

What can be done?

Potential countermeasure

- Memory scanning
  - After decryption, the original code has to be “unpacked” and stored in the memory to be executed
  - By scanning the **memory** for the original malware code pattern, we can detect malware



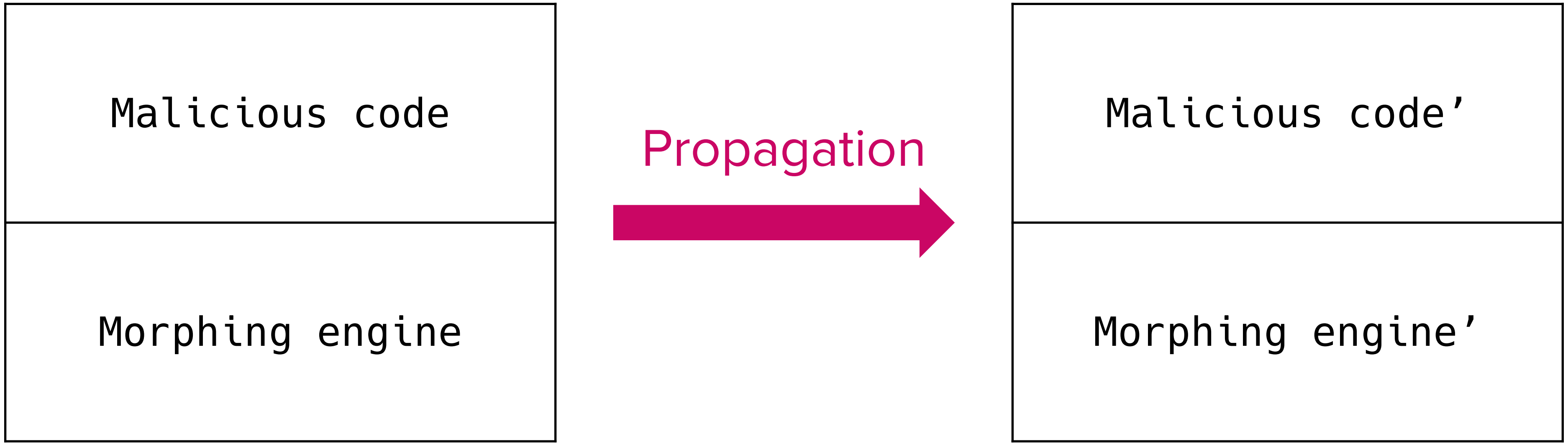
Potential countermeasure

- Polymorphic malware ends up exposing the unpacked code
  - Attacker: Can we completely remove packing/unpacking to bypass detection?

Metamorphism

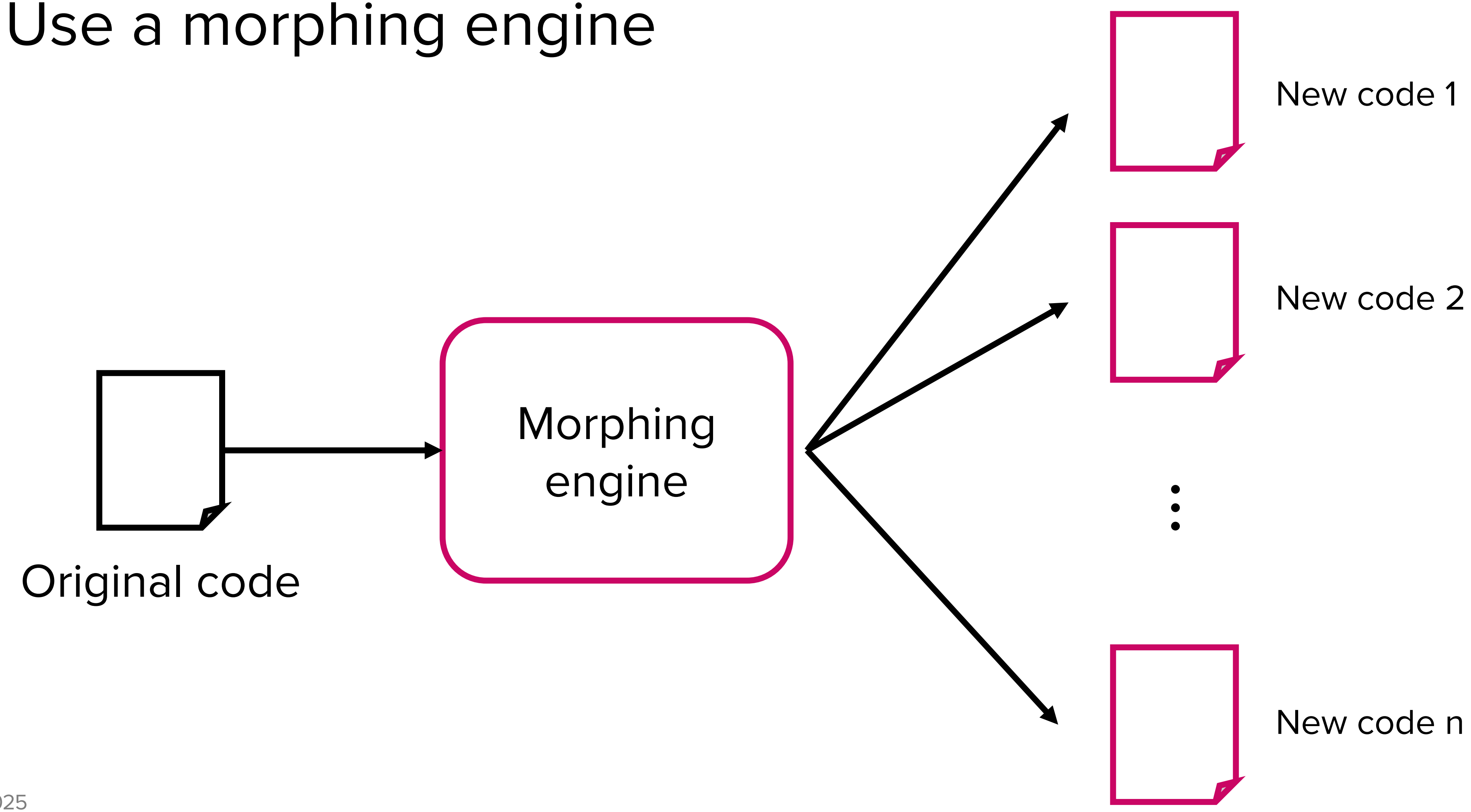
Metamorphic malware

- Concept
  - No encryption, no decryption routine
  - Malware rewrites its entire code to a functionally equivalent but syntactically different code each time it propagates



Metamorphic malware

- Idea: Use a morphing engine



Metamorphic malware

- Typical transformation
  - Adding dead code

```
6a 0b   push 0xb
58      pop eax
cd 80   int 0x80
```

Original code
(invoking execve syscall)

```
6a 0b   push 0xb
90      nop
58      pop eax
cd 80   int 0x80
```

```
6a 0b   push 0xb
43      inc ebx
4b      dec ebx
58      pop eax
cd 80   int 0x80
```

Metamorphic malware

- Typical transformation
  - Instruction substitution

```
31 c9   xor ecx, ecx
6a 04   push 4
59      pop ecx
```

Original code
(setting args for execve syscall)

→

```
31 c9   xor ecx, ecx
6a 04   push 4
5a      pop edx
89 d1   mov ecx, edx
```

⋮

Metamorphic malware

- Typical transformation
  - Function reordering
    - Reorder the order of invocations for functions that do not affect each other

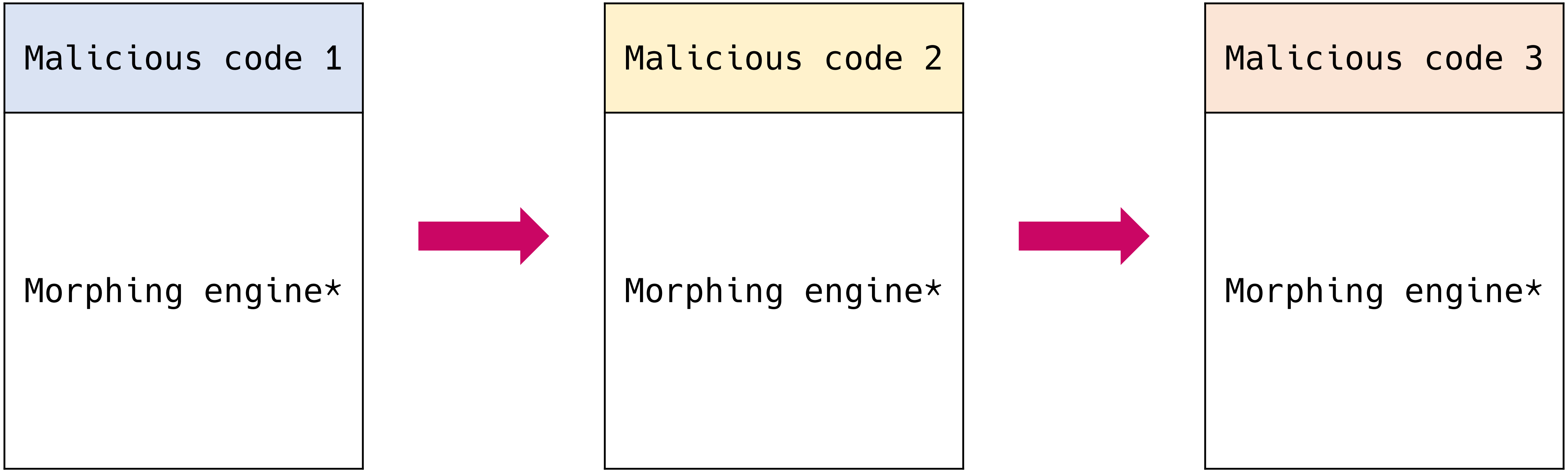
```
setvbuf(stdin, NULL, _IONBF, 0);
setvbuf(stdout, NULL, _IONBF, 0);
```

→

```
setvbuf(stdout, NULL, _IONBF, 0);
setvbuf(stdin, NULL, _IONBF, 0);
```

  - Code permutation
  - Randomizing
  - Compressing and decompressing
  - …

Metamorphic malware



* The morphing engine itself can also be metamorphic

Memory scanning-based detection no longer works!
Malicious code 1, 2, 3, … (not the unpacked original code)
are loaded onto the memory and get executed

Dynamic Analysis

Dynamic Analysis

- Problem:
  - Static analysis (e.g., pattern matching) cannot reliably detect the signatures of self-changing (metamorphic) code
- Idea:
  - Malware will eventually exhibit malicious behavior regardless of whether it is polymorphic or metamorphic
  - We can execute the program and **observe the behavior** to see if it matches malicious behaviors

Two categories of behavioral analysis

- Rule-based approach (== heuristic-based)
  - Detect malicious behavior
  - e.g., malware-specific behavior (reading sensitive files)

- Anomaly-based approach
  - Detect abnormal behavior
  - “Normal” and “Abnormal” behaviors should be defined

Rule-based dynamic analysis

- Monitor malicious behaviors with a set of rules
  - Attempts to open, view, delete, and/or modify files
  - Attempts to wipe out disk drives
  - Modifications to the logic of executable files
  - Modification of critical system settings, e.g., start-up scripts
  - Initiation of network communications

→ Many anti-virus software have their own collection of rules

Anomaly-based dynamic analysis

POSTECH

- Idea:
  - Define normal (== expected) behavior to identify malicious behavior
- Three types of anomalies
  - Point anomalies: Single extreme events
  - Contextual anomalies: Only anomalous under certain conditions
  - Collective anomalies: Benign individually, suspicious in aggregate

Anomaly-based dynamic analysis

- Point anomalies
  - If an individual data instance can be considered as anomalous with respect to the rest of data, then the instance a point anomaly
  - Example: Credit card fraud detection
    - Alice typically spends 5-40 USD per transaction
    - A transaction for a 20,000 USD product is anomalous

Anomaly-based dynamic analysis

- Contextual anomalies
  - If a data instance is anomalous only in a specific context, then it is a contextual (or conditional) anomaly
  - Example: Temperature
    - 30 ℃ (86 °F) at Pohang in December is abnormal
    - Same temperature in December is normal in Singapore or Abu Dhabi (hot all year round)

Anomaly-based dynamic analysis

- Collective anomalies
  - If a collection of related data instances is anomalous with respect to the entire dataset, it is a collective anomaly
  - Example: Money transfer
    - Alice transfers 200 USD to Mallory - normal
    - Bob transfers 200 USD to Mallory - normal
    - Claire transfers 200 USD to Mallory - normal
    - Dave transfers 200 USD to Mallory - normal
    - …
    - Zuckerberg transfers 200 USD to Mallory - normal

} Abnormal

Anomaly-based dynamic analysis

- Example: Self-immune system
  - Collect a sequence of system calls for normally operating programs
  - Build a profile of normal behavior based on the sequence
  - When we observe discrepancies, we flag them as anomalous

Anomaly-based dynamic analysis

- Example: Self-immune system
  - System call sequences of normal execution

```
open-read-mmap-mmap-open-getrlimit-mmap-close
```

```
open-getrlimit-close
```

```
open-getrlimit-mmap-close
```

```
open-read-mmap-mmap-open
```

Anomaly-based dynamic analysis

POSTECH

- Example: Self-immune system
  - Pairwise syscall profile using sliding window of 4

Syscall	pos 1	pos 2	pos 3
`open`	`read`	`mmap`	`mmap`
	`getrlimit`	-	`close`
`read`	`mmap`	`mmap`	`open`
`mmap`	`mmap`	`open`	`getrlimit`
	`open`	`getrlimit`	`mmap`
	`close`	-	-
`getrlimit`	`mmap`	`close`	-

Anomaly-based dynamic analysis

- Example: Self-immune system
  - Checking a behavior against the profile

Syscall	pos 1	pos 2	pos 3
`open`	`read`	`mmap`	`mmap`
	`getrlimit`	-	`close`
`read`	`mmap`	`mmap`	`open`
`mmap`	`mmap`	`open`	`getrlimit`
	`open`	`getrlimit`	`mmap`
	`close`	-	-
`getrlimit`	`mmap`	`close`	-

Behavior to check: `open-read-mmap-open-open-getrlimit-mmap-close`

No match

Anomaly-based dynamic analysis

- Example: Self-immune system
  - Checking a behavior against the profile

Syscall	pos 1	pos 2	pos 3
open	read	mmap	mmap
	getrlimit	-	close
read	mmap	mmap	open
mmap	mmap	open	getrlimit
	open	getrlimit	mmap
	close	-	-
getrlimit	mmap	close	-

Behavior to check: `open-read-mmap-open-open-getrlimit-mmap-close`

No match

Anomaly-based dynamic analysis

POSTECH

- Example: Self-immune system
  - Checking a behavior against the profile

Syscall	pos 1	pos 2	pos 3
open	read	mmap	mmap
	getrlimit	-	close
read	mmap	mmap	open
mmap	mmap	open	getrlimit
	open	getrlimit	mmap
	close	-	-
getrlimit	mmap	close	-

Behavior to check: `open-read-mmap-open-open-getrlimit-mmap-close`

No match

Anomaly-based dynamic analysis

- Example: Self-immune system
  - Checking a behavior against the profile

Syscall	pos 1	pos 2	pos 3
open	read	mmap	mmap
	getrlimit	-	close
read	mmap	mmap	open
mmap	mmap	open	getrlimit
	open	getrlimit	mmap
	close	-	-
getrlimit	mmap	close	-

Behavior to check: `open-read-mmap-open-open-getrlimit-mmap-close`

No match

Anomaly-based dynamic analysis

- Example: Self-immune system
  - Checking a behavior against the profile

Syscall	pos 1	pos 2	pos 3
open	read	mmap	mmap
	getrlimit	-	close
read	mmap	mmap	open
mmap	mmap	open	getrlimit
	open	getrlimit	mmap
	close	-	-
getrlimit	mmap	close	-

Behavior to check: `open-read-mmap-open-open-getrlimit-mmap-close`

Match

Mismatch rate: 4/5 = 80% → Anomalous!

Anomaly-based dynamic analysis

POSTECH

- How to obtain execution profile?
  - Using tracers
    - Tracers allow you to observe and/or intercept syscalls
    - ptrace, strace, ltrace, …
  - Attaching debuggers to running process
    - GDB, LLDB, WinDbg, …
  - Code instrumentation
    - Inject additional code into programs to track behavior
      - Adding `printf()` for debugging is a naïve form of instrumentation!
    - Pin, DynamoRio, Valgrind, …

Anomaly-based dynamic analysis

- Beware: Running potential malware is a bad idea
  - Sandboxing is recommended to avoid host compromise
    - e.g., Dynamically analyze a file in an isolated virtual machine

Summary

- Malware detection is an undecidable problem
- Static analysis
  - Fast – pattern matching w/o execution
  - Safe – does not require execution
  - Prone to false negatives – may miss self-modifying malware
- Dynamic analysis
  - Slow – need to execute
  - Potentially unsafe – need to execute potential malware
  - Better detection – resilient to poly/metamorphism

Questions?