Lec 17: Anti-Malware 2

CS492E: Introduction to Software Security

Sang Kil Cha

Recap

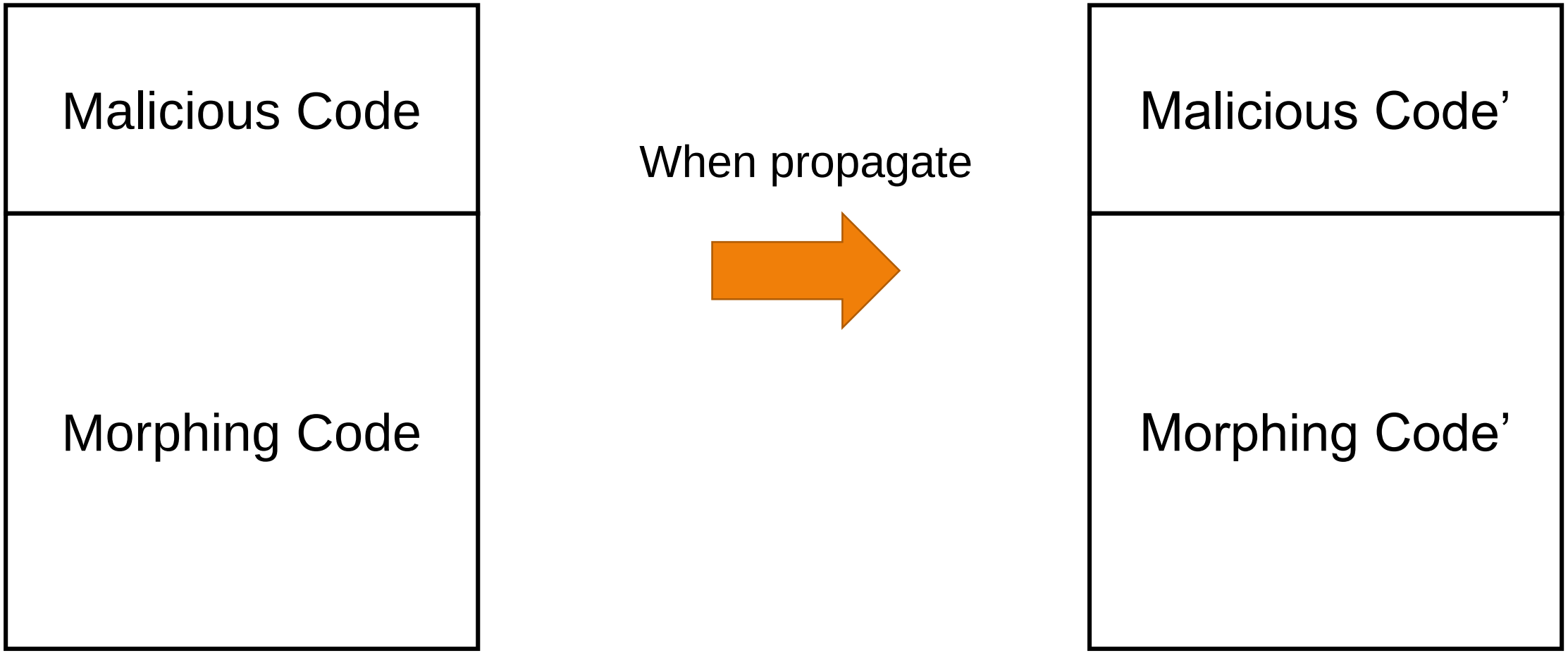
- Polymorphism
- Polymorphic encryption

Metamorphic Malware

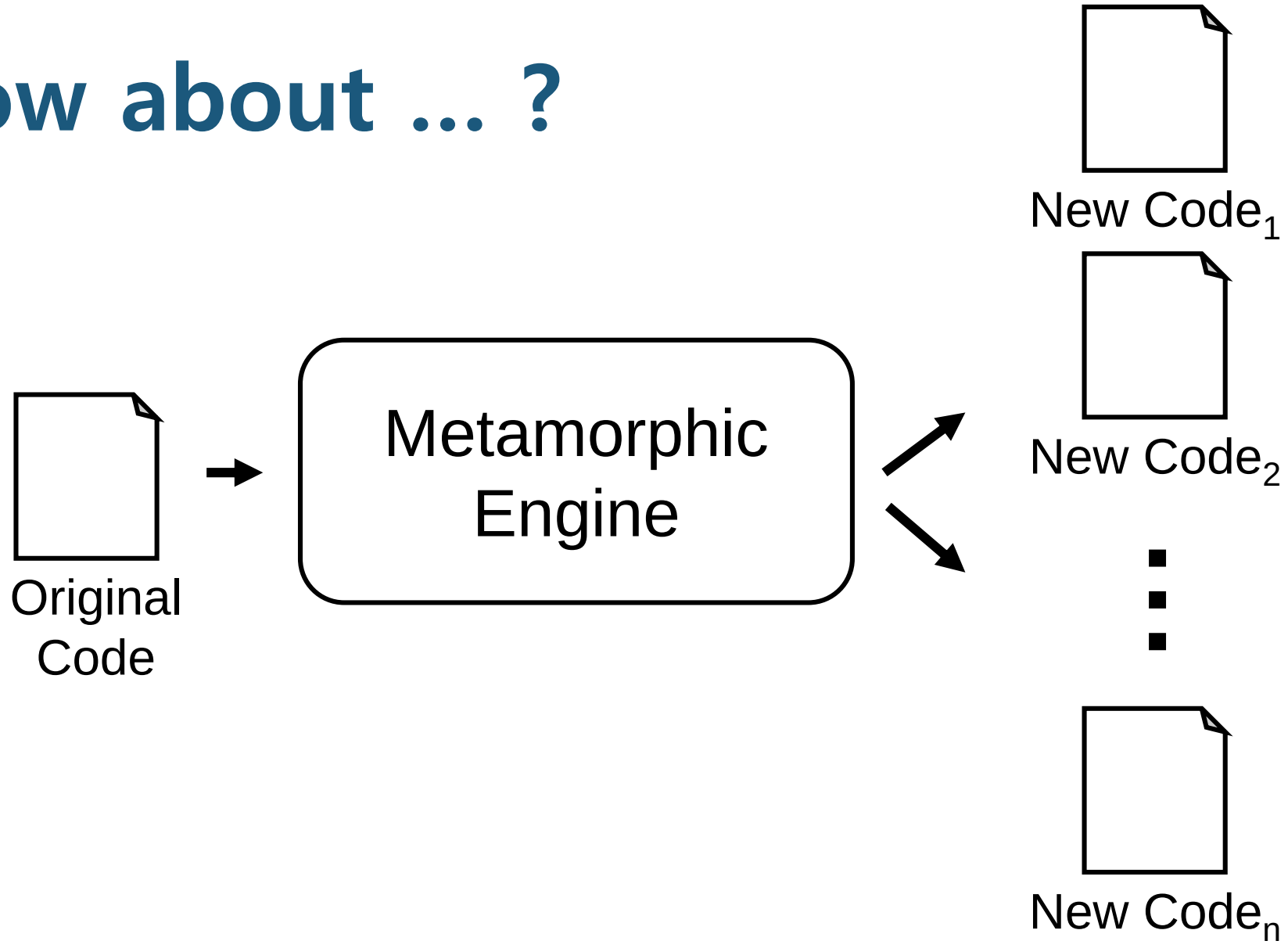
- No pack/unpack code

- Automatically change the code itself ***each time it propagates***

Metamorphic Malware (cont'd)



How about … ?


Original Code
Metamorphic Engine
New Code$_1$
New Code$_2$
New Code$_n$

Techniques for Metamorphism

- Add some dead code in random places in the code
- Reallocate registers
- Function reordering
- And many more …

Dynamic Analysis

- Behavioral analysis

- Run the program/system and observe behavior

Whether it is polymorphic or metamorphic, it will show the same behavior

Two Categories of Behavioral Detection

- Heuristic-based or Rule-based: detect malicious behavior
  - Remote shell is spawned from a process
  - Malware-specific behavior

- Anomaly-based: detect abnormal behavior
  - Define what normal (benign) behavior is
  - When your system behaves abnormally, raise an alarm

Which one is better? And why?

Heuristic-based Approach: SNORT

- Observe network behaviors
- Consist of a large collection of rules

Anomaly-based Approach

Try to define normal (or expected) behavior in order to identify malicious behavior!

Reference: Anomaly Detection: A Survey, ***CSUR 2009***

3 Types of Anomalies

- ***Point anomalies***: defined with an individual data point
- ***Contextual anomalies***: defined in a certain context
- ***Collective anomalies***: defined with a collection of related data

Point Anomalies

If an individual data instance can be considered as anomalous with respect to the rest of data, then the instance is termed as a point anomaly.

From Anomaly Detection: A Survey, CSUR 2009

Example: Credit Card Fraud Detection

Customer X typically spends 1,000 won ~ 100,000 won per transaction.

A transaction for which the amount spent is 10,000,000 won is anomalous.

Contextual Anomalies

If a data instance is anomalous in a specific context (but not otherwise), then it is termed as a contextual anomaly.

a.k.a. conditional anomalies

From Anomaly Detection: A Survey, CSUR 2009

Example: Temperature

30 °C in ***winter of Daejeon*** is abnormal

Example: Credit Card Fraud Detection

Customer X typically spends 100,000 won per week.

Weekly bill of 1,000,000 won ***during Chuseok holiday*** is normal.

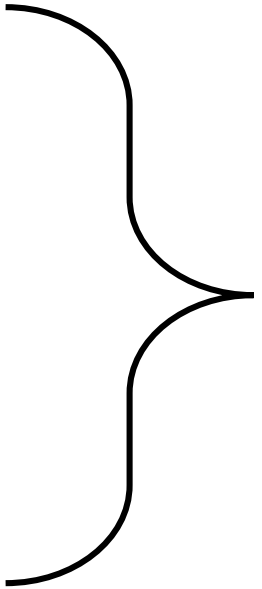
Collective Anomalies

If a collection of related data instances is anomalous with respect to the entire data set, it is termed as a collective anomaly.

From Anomaly Detection: A Survey, CSUR 2009

Example: Money Transfer

A transfers 100,000 won to X: normal
B transfers 100,000 won to X: normal
C transfers 100,000 won to X: normal
D transfers 100,000 won to X: normal
…
Y transfers 100,000 won to X: normal
Z transfers 100,000 won to X: normal



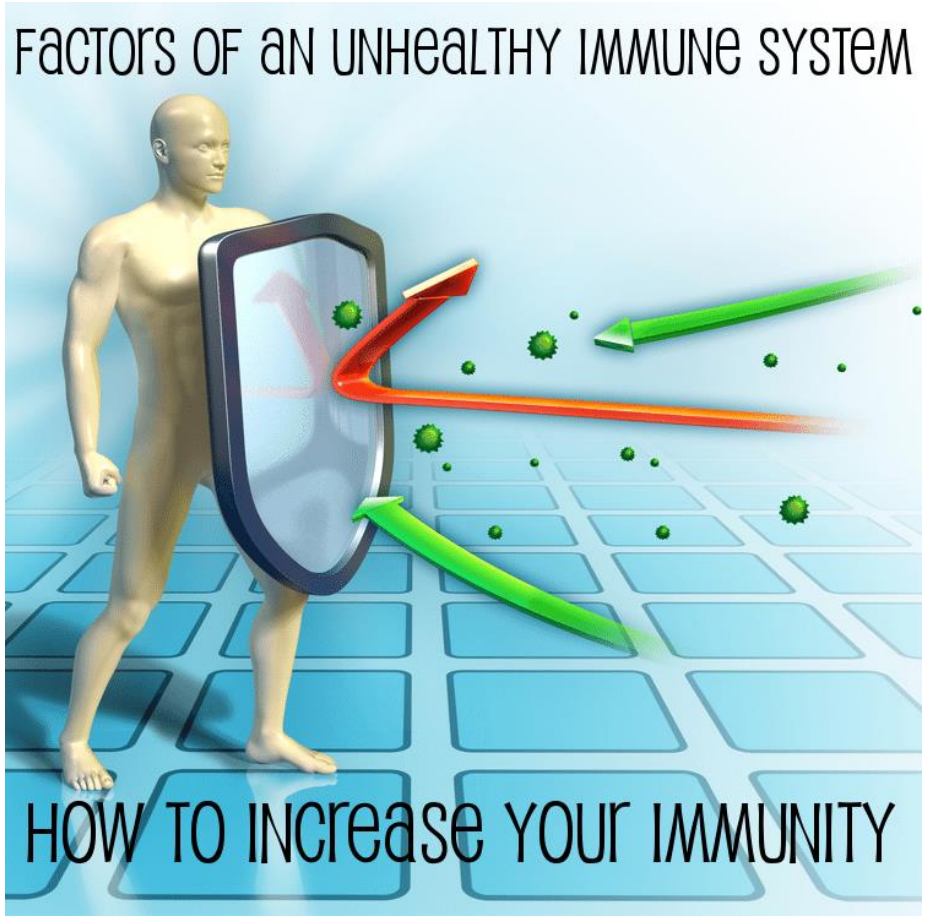
Abnormal

Behavioral IDS

Collective anomaly detection for HIDS

A sense of self for UNIX processes, ***IEEE S&P 1996***

Natural Immune System



Can we build a malware detection system that is as good as natural immune system?

Definition of Self

- 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 treat them as anomalies

Building a Pairwise Profile

- Sliding window of size 4
- Normal execution example:

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

call	position 1	position 2	position 3
open	read, getrlimit	mmap	mmap, close
read	mmap	mmap	open
mmap	mmap, open, close	open, getrlimit	getrlimit, mmap
getrlimit	mmap	close	
close			

Detecting Anomaly

- Sliding window of size 4
- Abnormal execution example:

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

In total 4 mismatch out of 18 (3 x 5 + 2 + 1) possible pairwise mismatches = 22% miss rate

If the miss rate is above a certain threshold, we say the system is abnormal

Obtaining Execution Profile?

- Ptrace
- Attaching debugger to a running process
  - GDB, LLDB, WinDbg, etc.
  - Single stepping: context switching for every single execution
- Instrumentation
  - Pin, DynamoRio, Valgrind, etc.

Defeating Behavior-based Detection

Mimic normal system call sequences!

Mimicry Attacks on Host-based Intrusion Detection Systems,
CCS 2002

More Fundamental Question

- How can we trick dynamic analysis?

- How can we hide execution behavior of a program?

Platform-Independent Programs

Common Assumption

A single executable program runs only on a ***specific platform***.

Common Assumption (cont'd)

A single executable program runs only on a ***specific platform***.

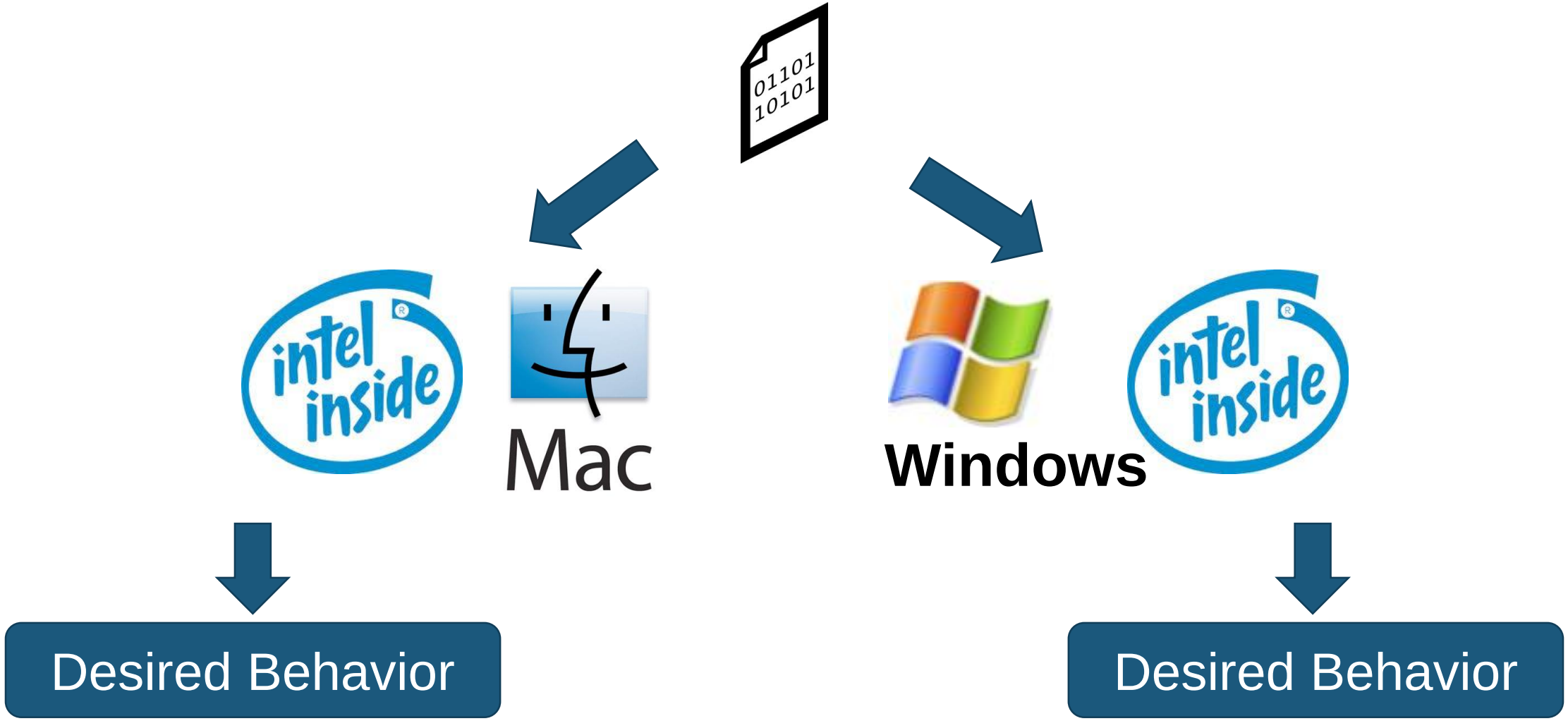
A program can run only on one platform

Automatically generate single binary string that is valid on multiple platforms

A Platform is …

- ISA (Instruction Set Architecture)
  - ARM, MIPS, Intel
- OS (Operating System)
  - Linux, macOS, Windows

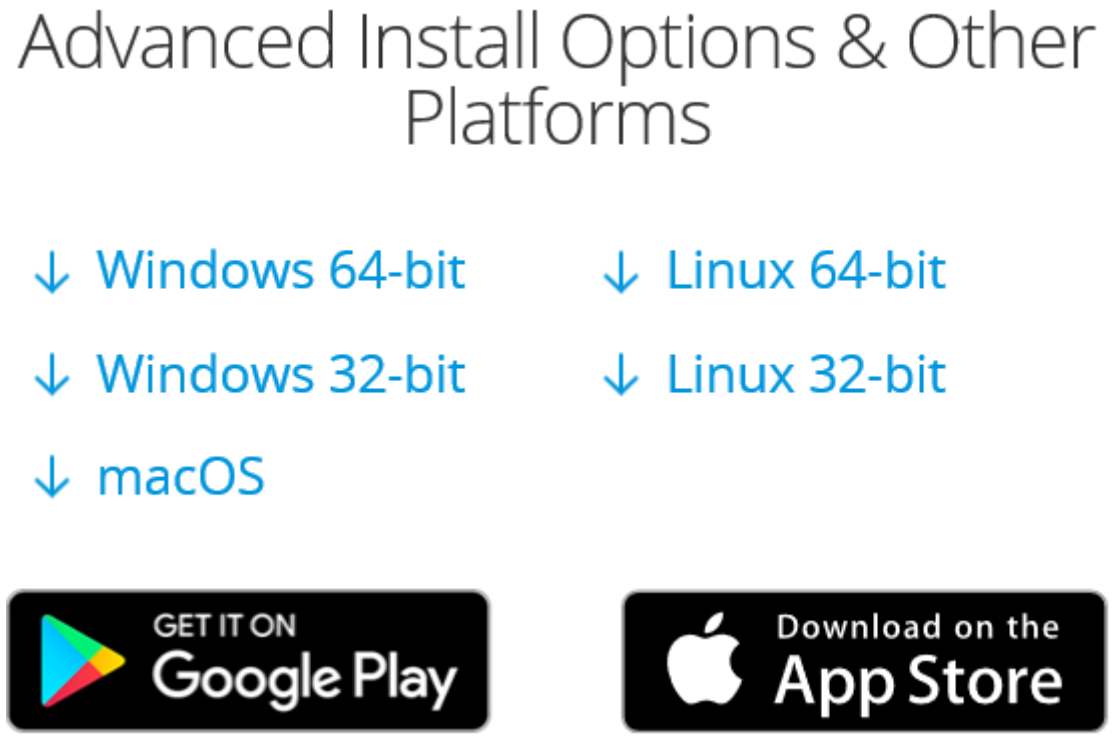
Platform-Independent Program (PIP)



So, Why PIP?

Cool, new paradigm!

Programmer's Perspective



Attacker's Perspective

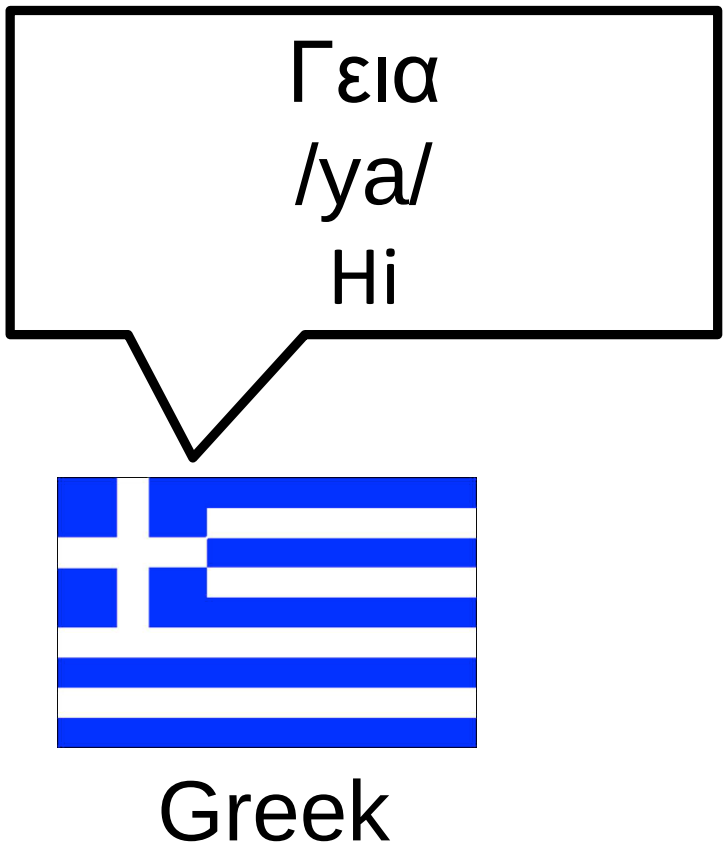
- Platform-independent exploit (shellcode)

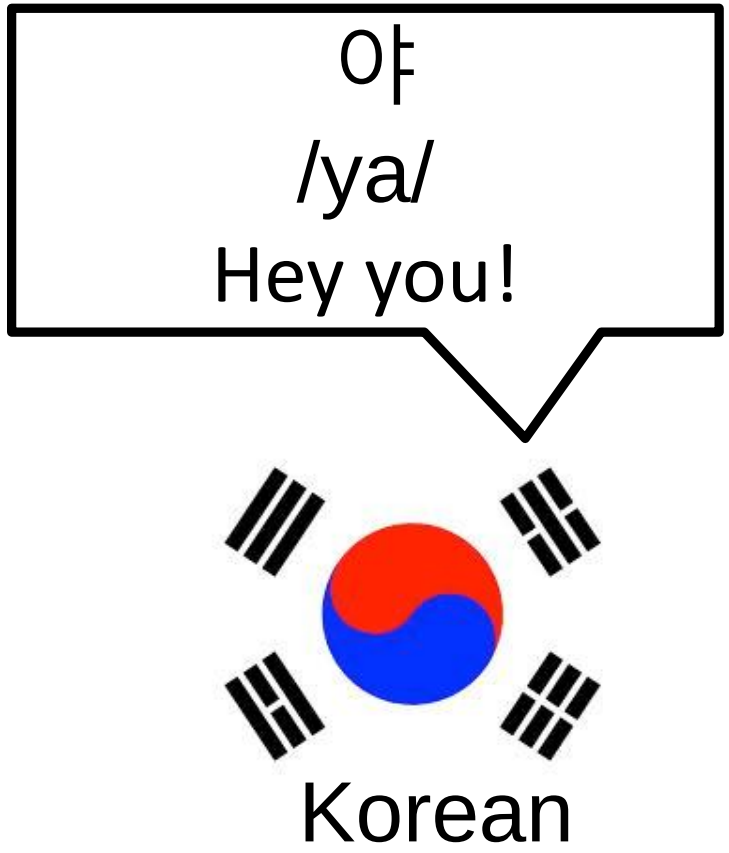
- Platform-independent malware

Execution-based Steganography

Hide runtime behavior of the program!

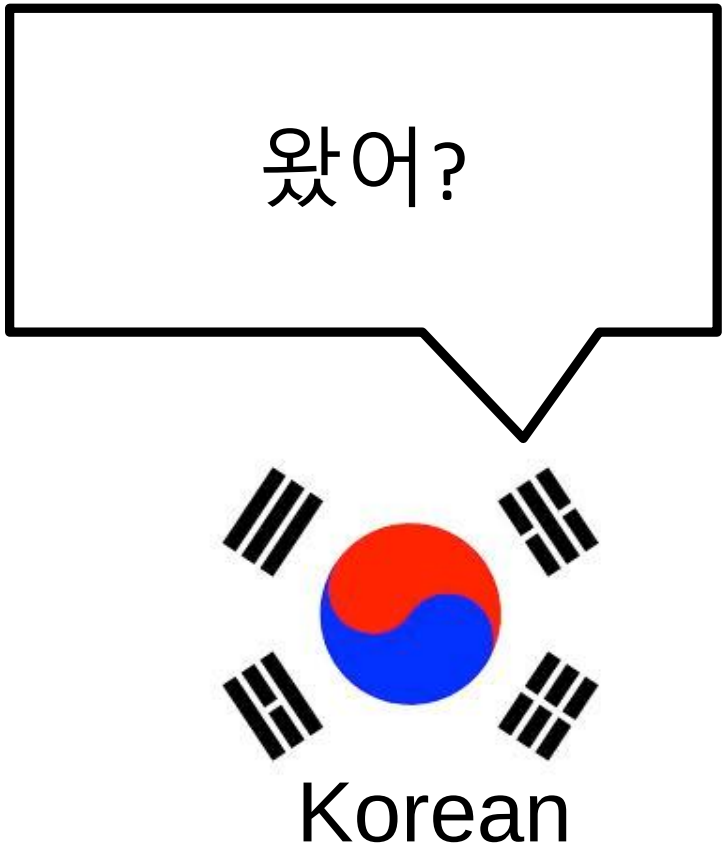
Intuition: False Friends





Intuition: False Friends

English



Korean

Instruction Overlap

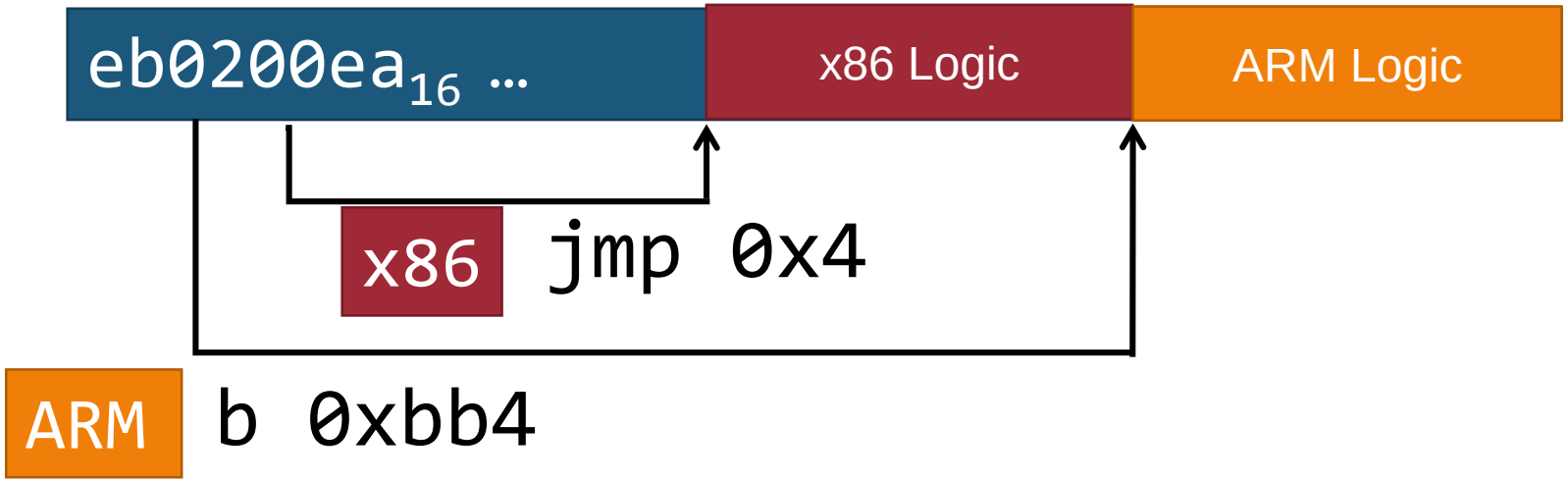
56565656_{16}


```
push esi
push esi
push esi
push esi
```

MIPS TECHNOLOGIES

```
bnel $r18,$r22,0x1595c
```

Basic Construction: Finding Overlaps between Jump Instructions

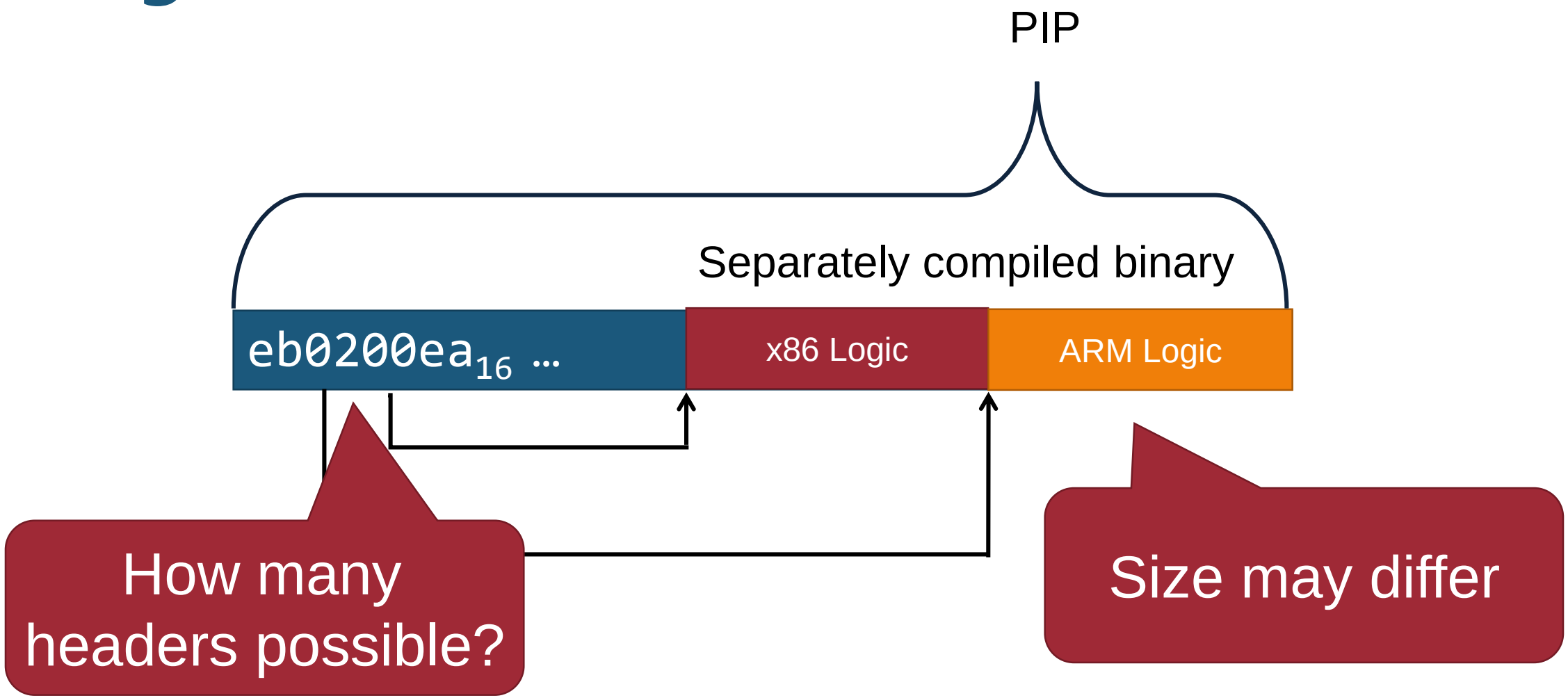


Challenges

- Automatically constructing PIPs

- Turing-complete language
  - PIP meta-language for generating PIPs

Single PIP Header



Over billions of PIP Headers Possible!

- For x86, ARM, and MIPS
- Various jump offsets

But, each binary string should be compiled separately!

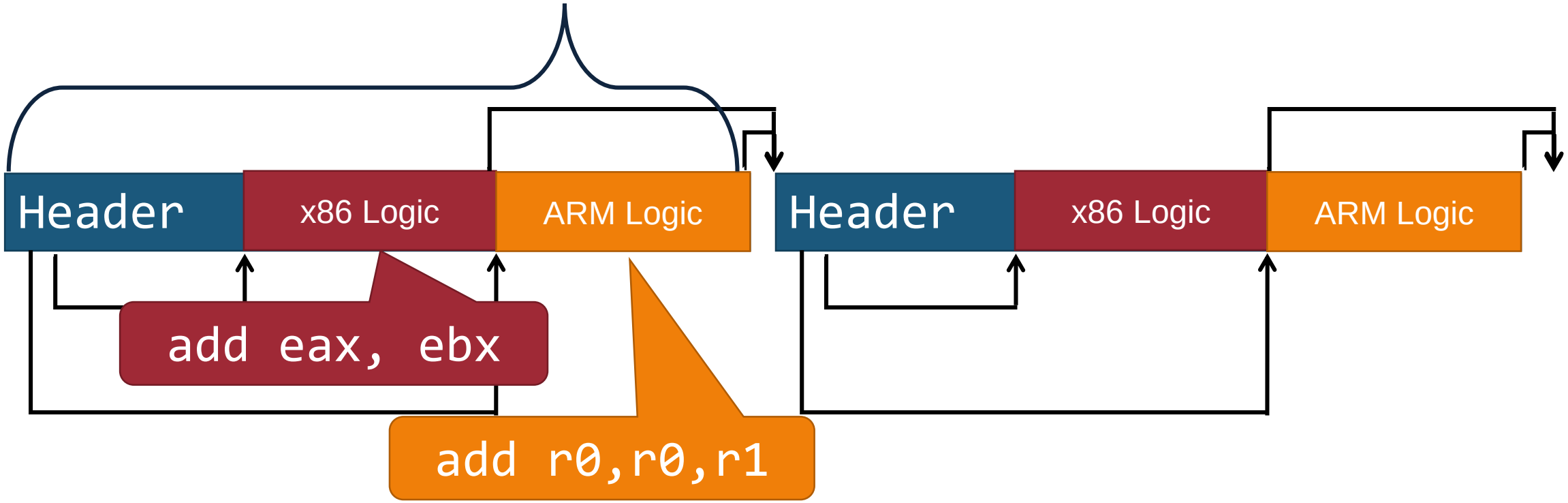
Turing-Complete PIP?

- Construct platform-independent instructions
  - A ***platform-independent gadget*** is a platform-independent instruction

- Splice platform-independent instructions using jump instructions

Turing-Complete Language with Platform-Independent Gadgets

A platform-independent Instruction



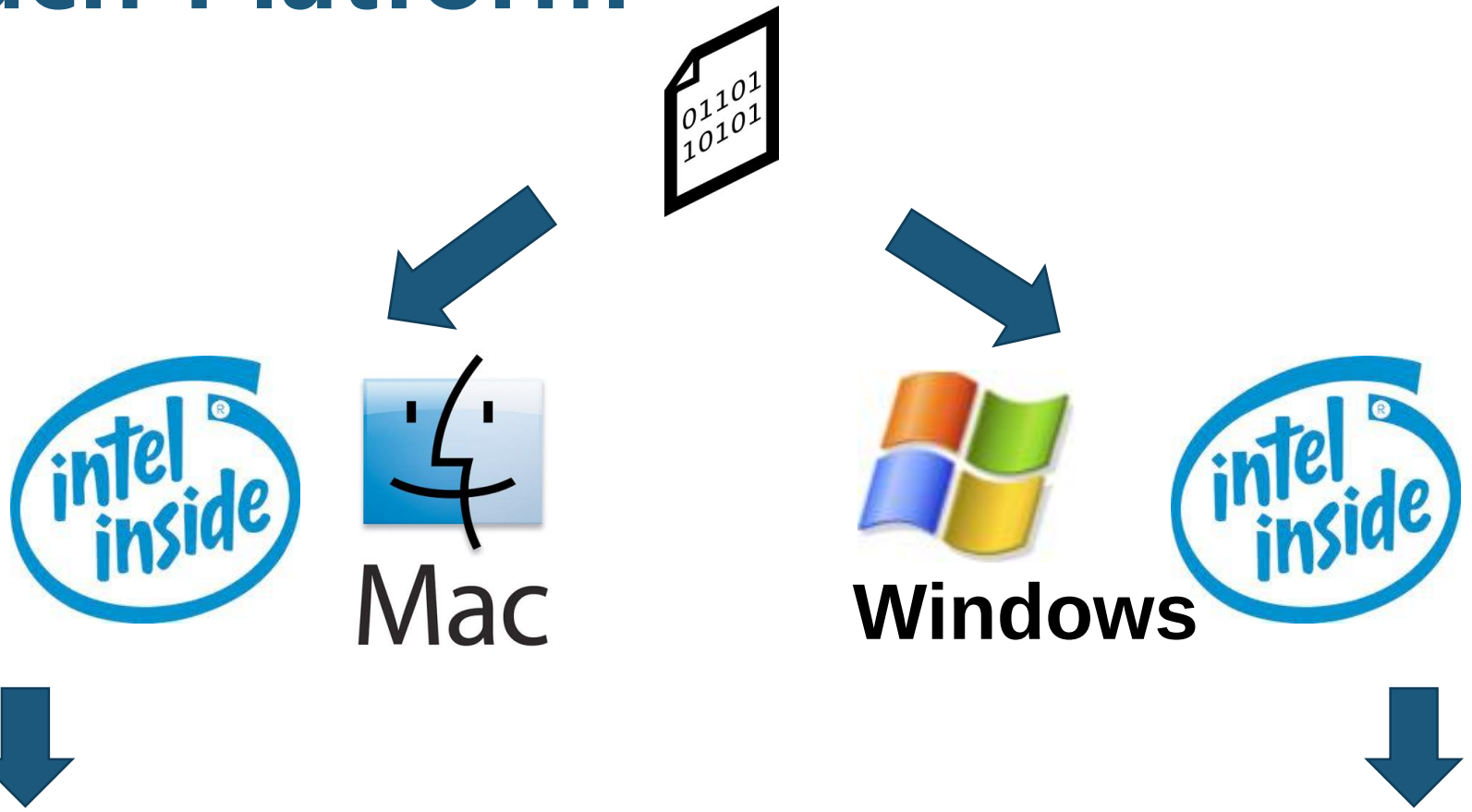
Finding Gadget Headers

- Headers must be side-effect free
- For all platforms, a gadget header is decoded for each platform as in a form of

  (nop*)(branch)(.*)

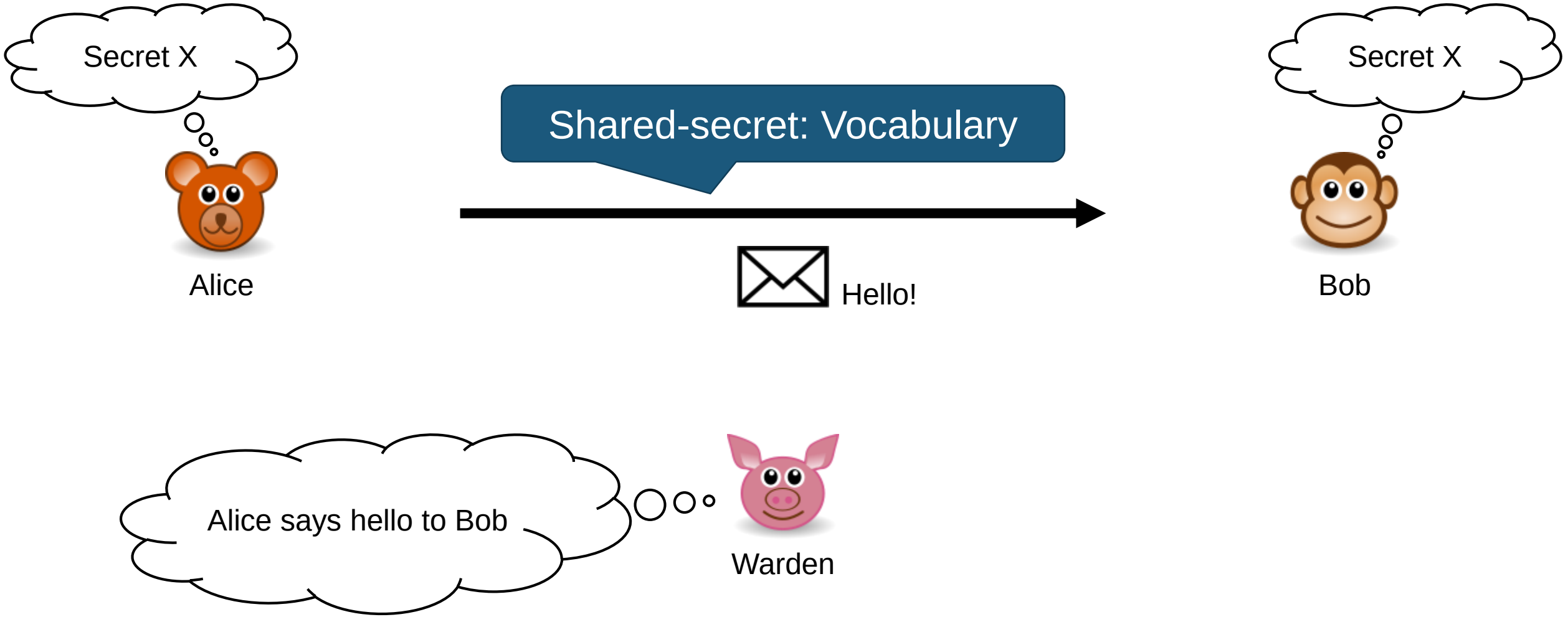
- **Example:** $\texttt{eb0200ea}_{16}$
  - **ARM**: `b 0xbb4`
  - **x86**: `jmp 0x4`
- ≫ billions of 12-byte overlaps for x86,ARM,MIPS

PIP Allows Different Logic for Each Platform

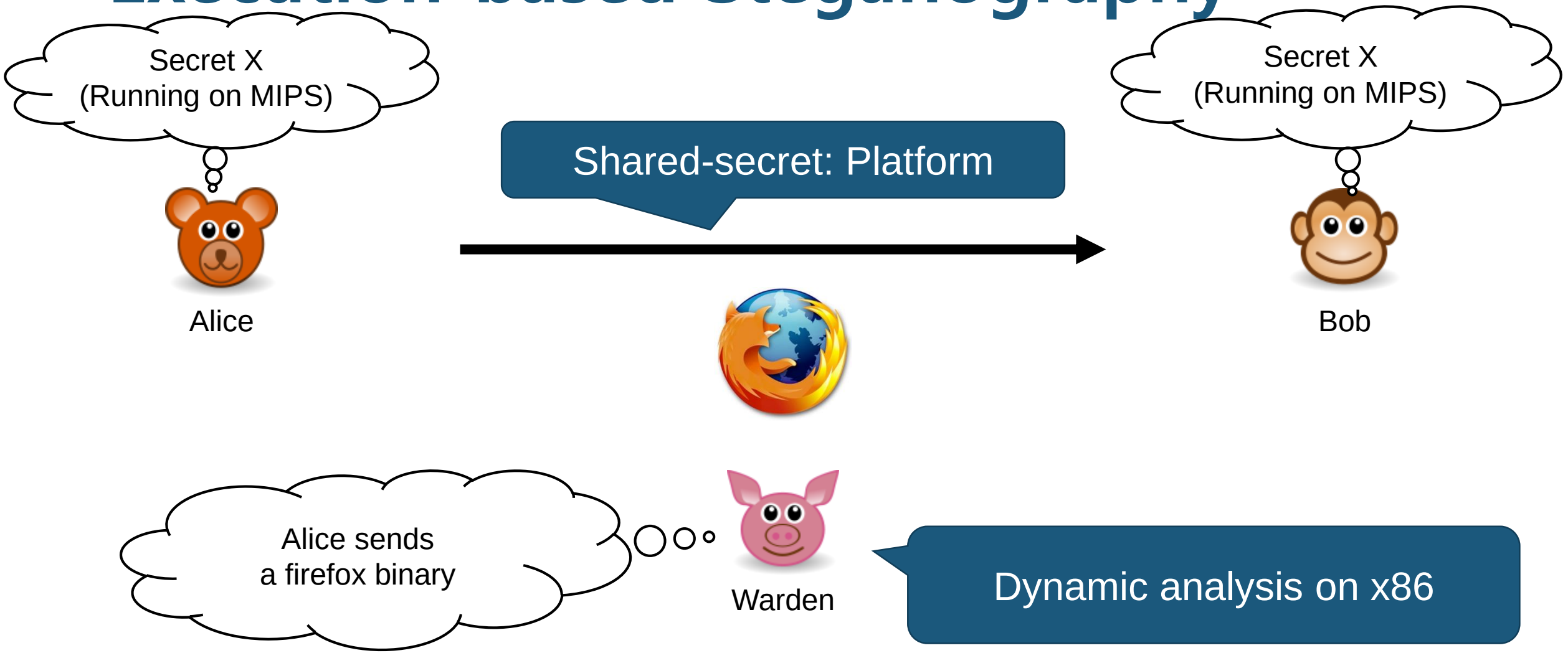


Execution-based Steganography

Classic Steganography



Execution-based Steganography



Other Results

- 8 platform-independent shellcode (x86, ARM, and MIPS)
  - Confirmed with 2 real-world exploits
- Platform-independent malware
  - A virus that spreads over NFS
- Platform-independent shellcode for OSes
  - FreeBSD, Linux, and Mac OS X

Discussion

```
7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
02 00 03 00 01 00 00 00  54 80 04 08 34 00 00 00  |........T...4...|
```

1-byte field representing architecture

- Some OS rejects a program if the file format of the program contains wrong architecture information.
  - Some executable file format does not include architecture information (e.g., COFF).

- Architecture checks are important against PIP, even though they were likely not intended as a security measure.
  - Embedded archs, emulators may all be vulnerable to PIP attacks

x86 vs. x86-64

15-byte code: `\x31\xc9\x41\xe2\x08\x90 ...`

x86

```
xor ecx,ecx
inc ecx
loop 0xd
nop
…
```

x86-64

```
xor ecx,ecx
loop 0xd
nop
…
```

Conclusion

- Metamorphism: harder to break than polymorphism
- Dynamic analysis (behavior-based analysis) for the rescue?
- Mimicry attack
- Execution-based steganography (with PIP)

Questions?