A Survey of Obfuscations in Prevalent Packer Tools

Kevin Roundy
Paradyn Project

Paradyn / Dyninst Week
Madison, Wisconsin
March 26, 2012

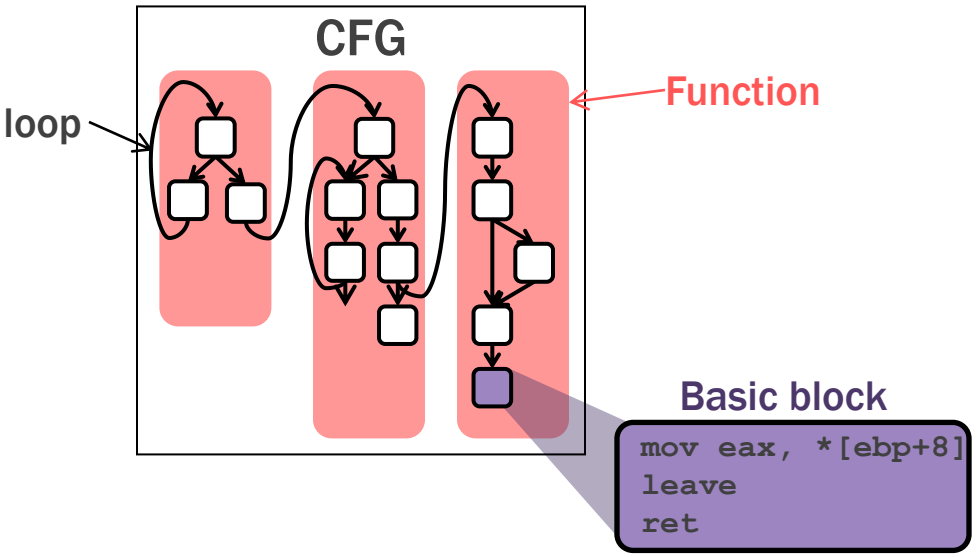
Types of program analysis

Source code

Friendly binary

Uncooperative binary

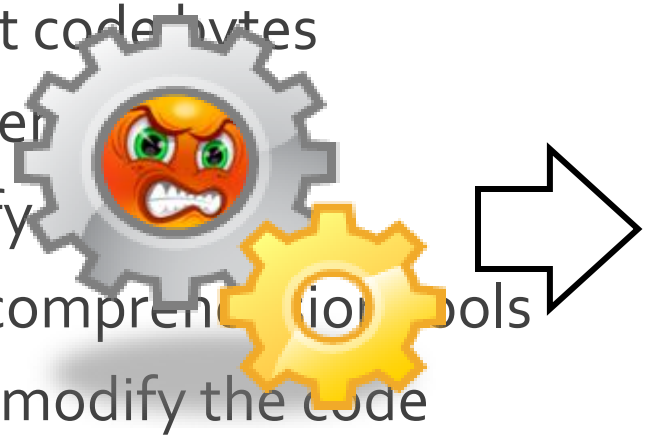
Hostile binary



Analysis building blocks

Analysis steps

1. Extract code bytes
2. Disassemble
3. Identify
4. Build comprehension tools
5. Patch/modify the code
6. Trace code's execution



Toolkit:

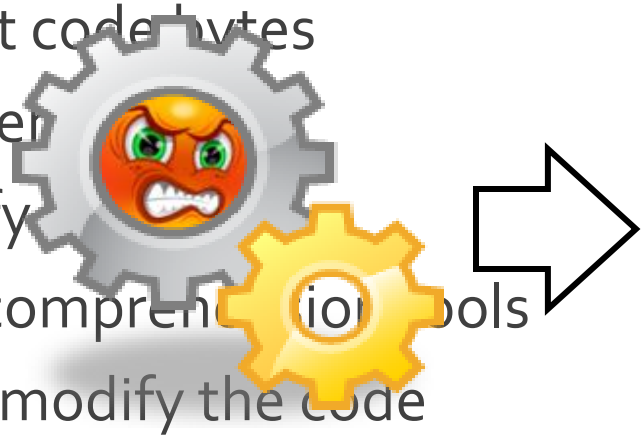
Defensive-mode Dyninst

- Interactive debuggers
- Automated testing
  - Combinatorial testing
  - Code-coverage test generation
- Fault localization
  - Backwards slicing
  - Correlation of statement executions and test failures
- Vulnerability analysis
  - Taint analysis
  - Symbolic evaluation

Analysis building blocks

Analysis steps

1. Extract code bytes
2. Disassemble
3. Identify
4. Build comprehension tools
5. Patch/modify the code
6. Trace code's execution



Toolkit:

Defensive-mode Dyninst

- Interactive debuggers
- Automated testing
  - Combinatorial testing
  - Code-coverage test generation
- Fault localization
  - Backwards slicing
  - Correlation of statement executions and test failures
- Vulnerability analysis
  - Taint analysis
  - Symbolic evaluation

Binary packing tools

Open source cross-platform

Fast, **Small**, Good

Anti-reverse engineer-ing

Packer	Malware market share*
OVERALL	75%-80%
UPX	9.45%
PolyEnE	6.21%
PECompact	2.59%
Upack	2.08%
nPack	1.74%
ASPack	1.29%
FSG	1.26%
Nspack	0.89%
ASProtect	0.43%
Armadillo	0.37%
Yoda's Prot.	0.33%
WinUpack	0.17%
MEW	0.13%

Paradyn

* Packer (r)evolution. Panda Research, 2008. Two-month average Feb-March 2008.

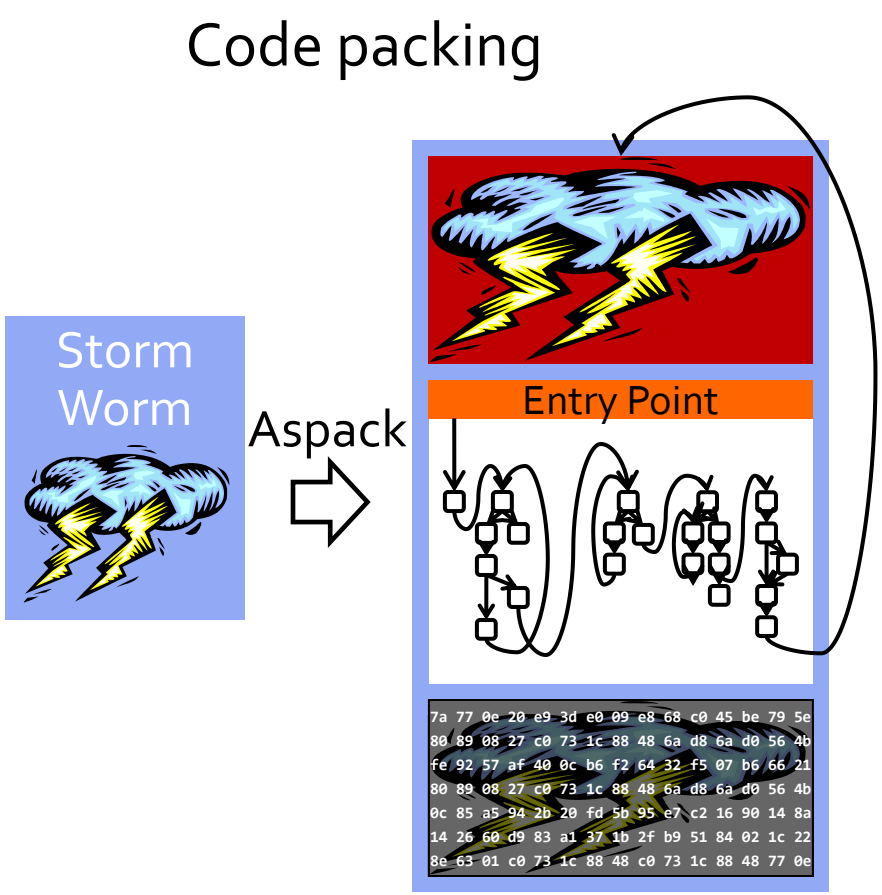
Dyninst

Outline

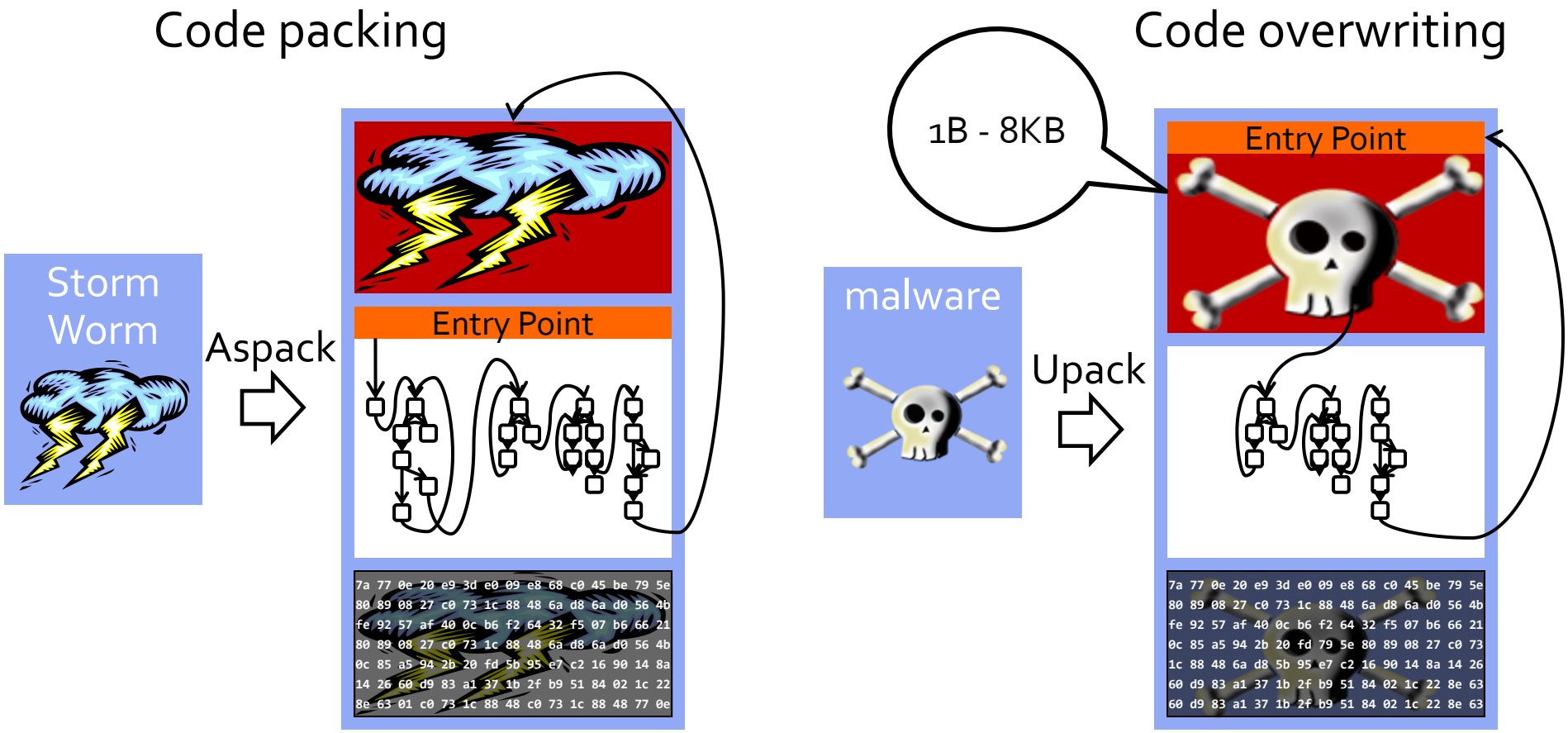
Analysis steps

1. Extract code bytes
   a. Code packing
   b. Code overwriting
2. Disassemble
3. Identify functions
4. Build comprehension tools
5. Patch/modify the binary
6. Trace code's execution

Code packing



Code overwriting



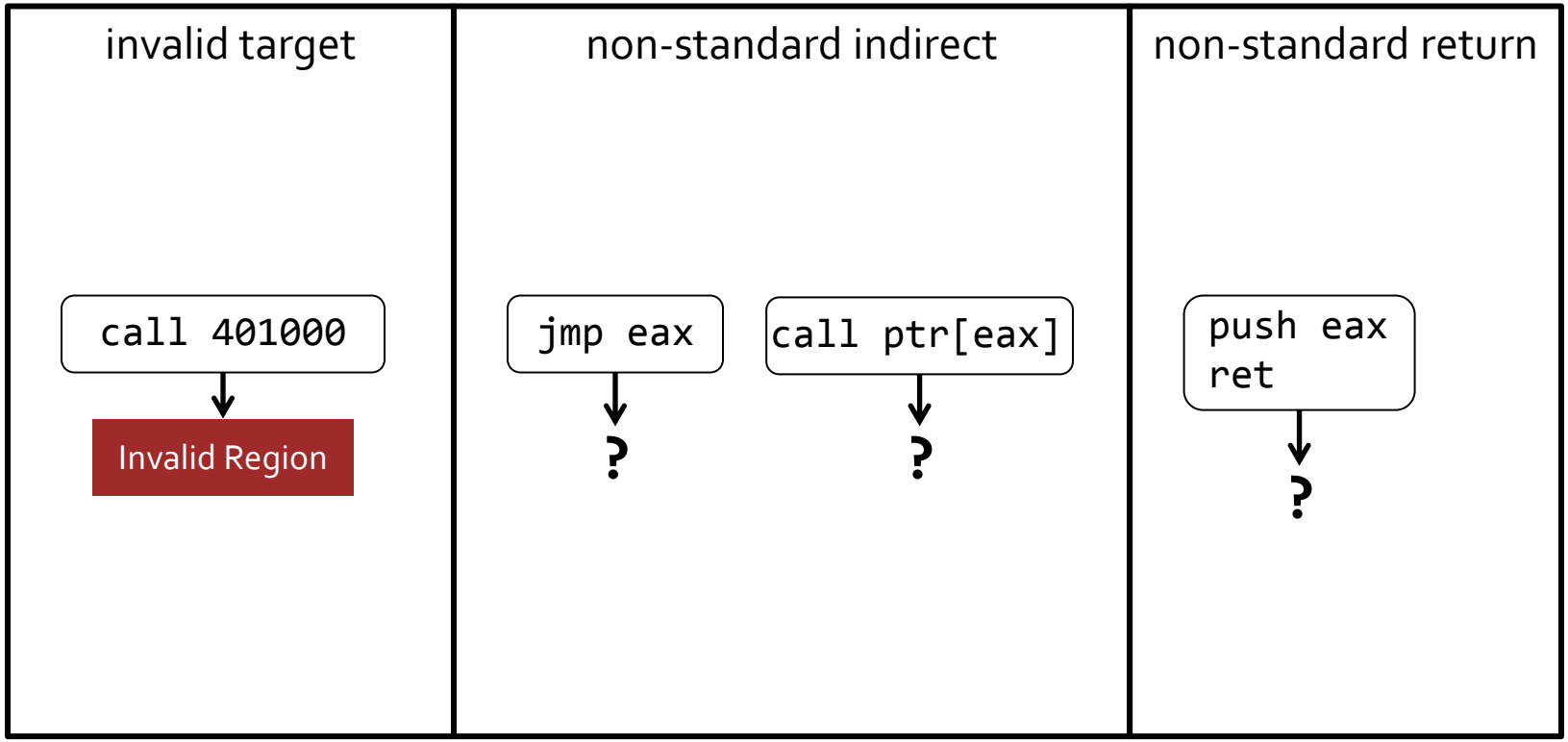
Outline

Analysis steps

1. Extract code bytes
2. Disassemble
3. Identify functions
4. Build comprehension tools
5. Patch/modify the binary
6. Trace code's execution

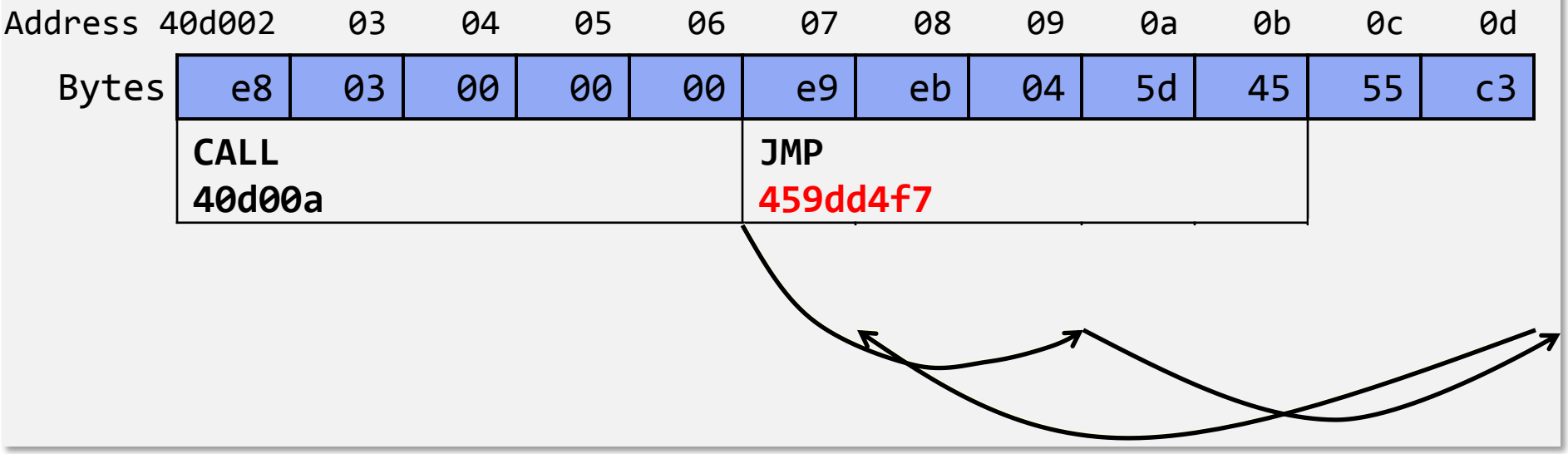
a. Unresolvable control flow
b. Call-stack tampering
c. Signals and exceptions
d. Ambiguous code & data
e. Disassembler fuzzing

Unresolvable control flow



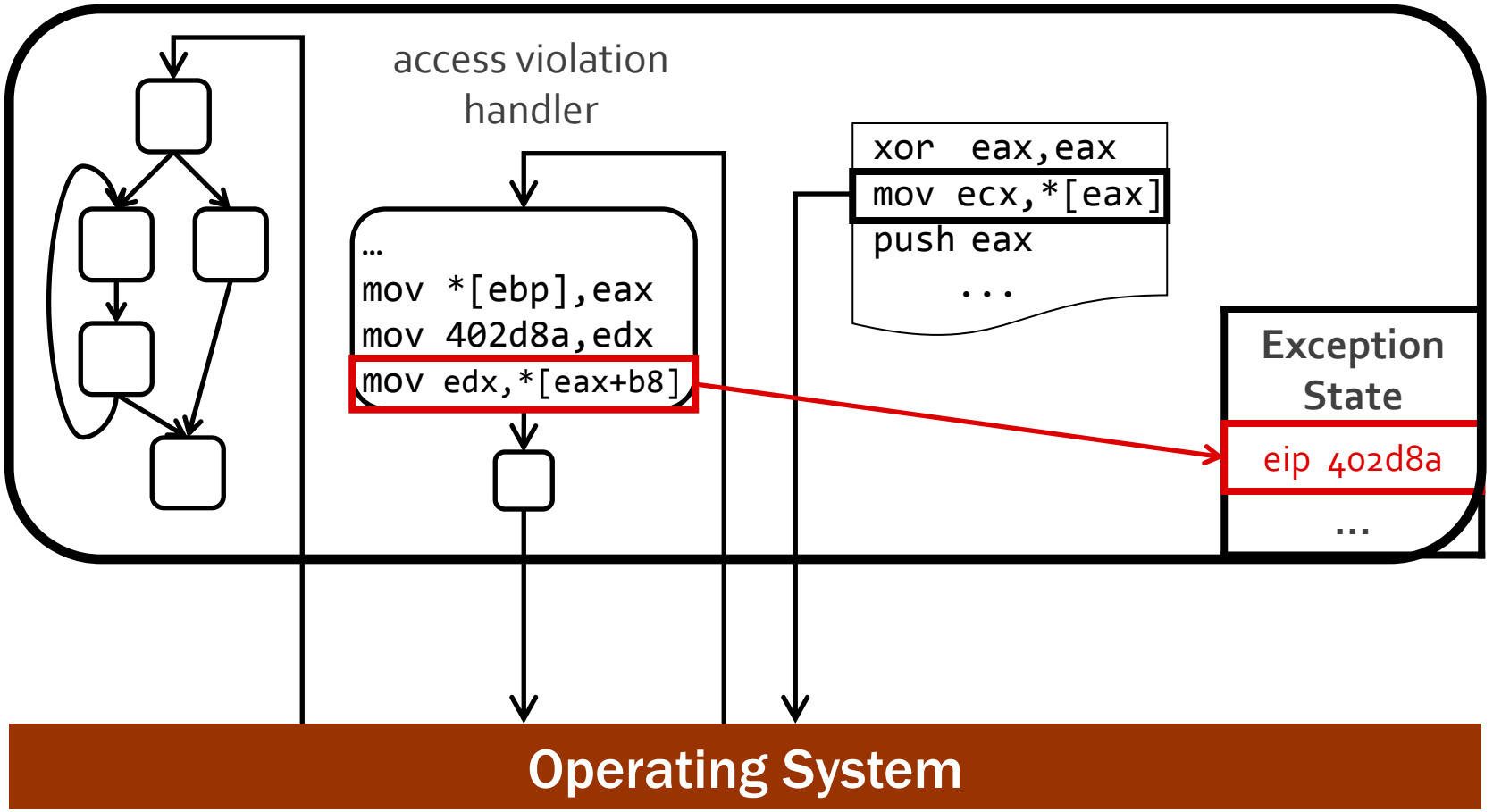
Call-stack tampering

Storm Worm



Exception-based control flow

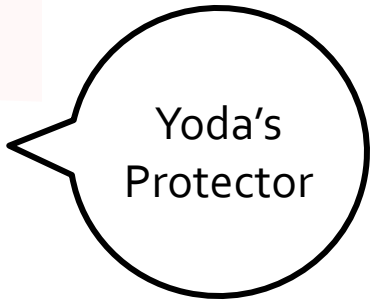
Popov, Debray, Andrews. Usenix 2007.
Danekhar. http://www.codeproject.com/KB/system/inject2exe.aspx 2005.



Ambiguous code and data

- Bytes after call instructions
- Junk after exception-raising instruction
- In-place decryption of unpacked code

```
0101D843   04 37            ADD AL,37
0101D845   04 6A            ADD AL,6A
0101D847   F8               CLC
0101D848   AA               STOS BYTE PTR ES:[EDI]
0101D849 ^E2 9C             LOOPD SHORT notepad_.0101D7E7
0101D84B   E8 D4D99C9F      CALL A09EB224
0101D850   F0:A7            LOCK CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[EDI]
0101D852   24 89            AND AL,89
0101D854   1C 64            SBB AL,64
0101D856 ^73 F9             JNB SHORT notepad_.0101D851
0101D858   7B FD            JPO SHORT notepad_.0101D857
0101D85A   0355 CA          ADD EDX,DWORD PTR SS:[EBP-36]
0101D85D   37               AAA
0101D85E   8C0F             MOV WORD PTR DS:[EDI],CS
0101D860 vE1 7F             LOOPDE SHORT notepad_.0101D8E1
0101D862   DB98 A29064E7    FISTP DWORD PTR DS:[EAX+E76490A2]
0101D868   45               INC EBP
0101D869   D93F             FSTCW WORD PTR DS:[EDI]
0101D86B   59               POP ECX
0101D86C   D4 7A            AAM 7A
0101D86E   8C3D 4245489A    MOV WORD PTR DS:[9A484542],SEG?
0101D874   50               PUSH EAX
0101D875   6223             BOUND ESP,QWORD PTR DS:[EBX]
0101D877   C2 1DDA          RETN 0DA1D
0101D87A   1862 A6          SBB BYTE PTR DS:[EDX-5A],AH
0101D87D   2987 2BD2B459    SUB DWORD PTR DS:[EDI+59B4D22B],EAX
0101D883   CA 80D4          RETF 0D480
0101D886   1D 780B0E23      SBB EAX,230E0B78
0101D88B   3B6B CE          CMP EBP,DWORD PTR DS:[EBX-32]
0101D88E   ED               IN EAX,DX
```



Outline

Analysis steps

1. Extract code bytes
2. Disassemble
3. Identify functions
4. Build comprehension tools
5. Patch/modify the binary
6. Trace code's execution

a. Missing `call/ret` instructions
b. Extra `call/ret` instructions
c. Overlapping functions
d. Overlapping basic blocks

Outline

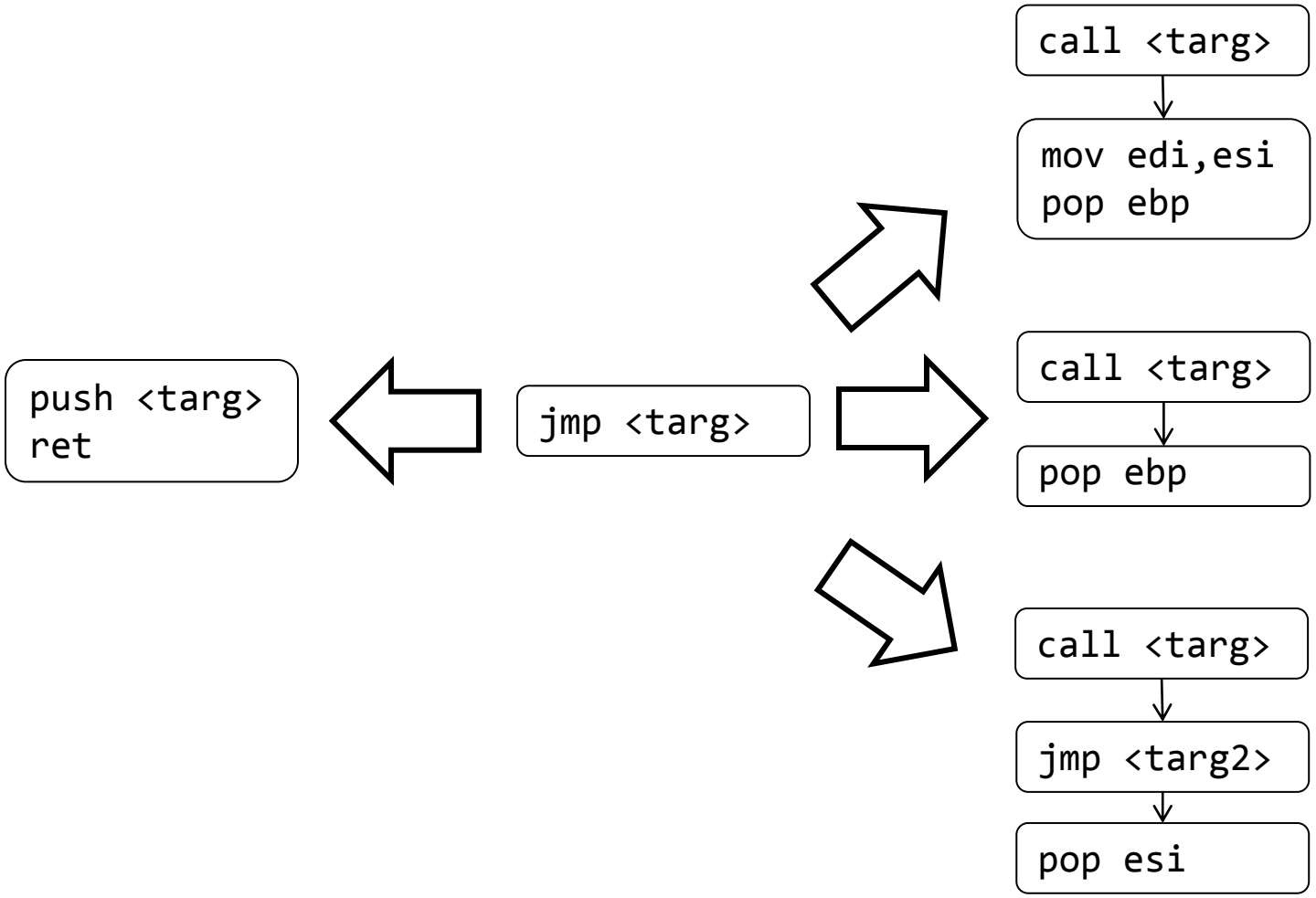
Analysis steps

1. Extract code bytes
2. Disassemble
3. Identify functions

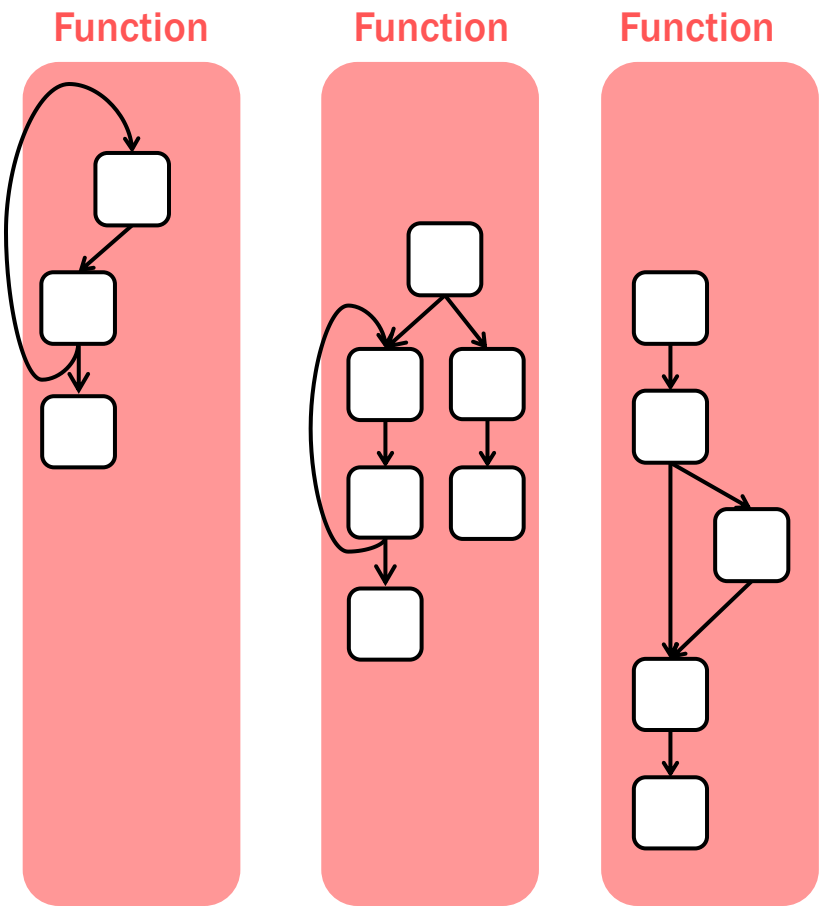
   a. Missing `call`/`ret` instructions
   b. Extra `call`/`ret` instructions
   c. Overlapping functions
   d. Overlapping basic blocks

4. Build comprehension tools
5. Patch/modify the binary
6. Trace code's execution

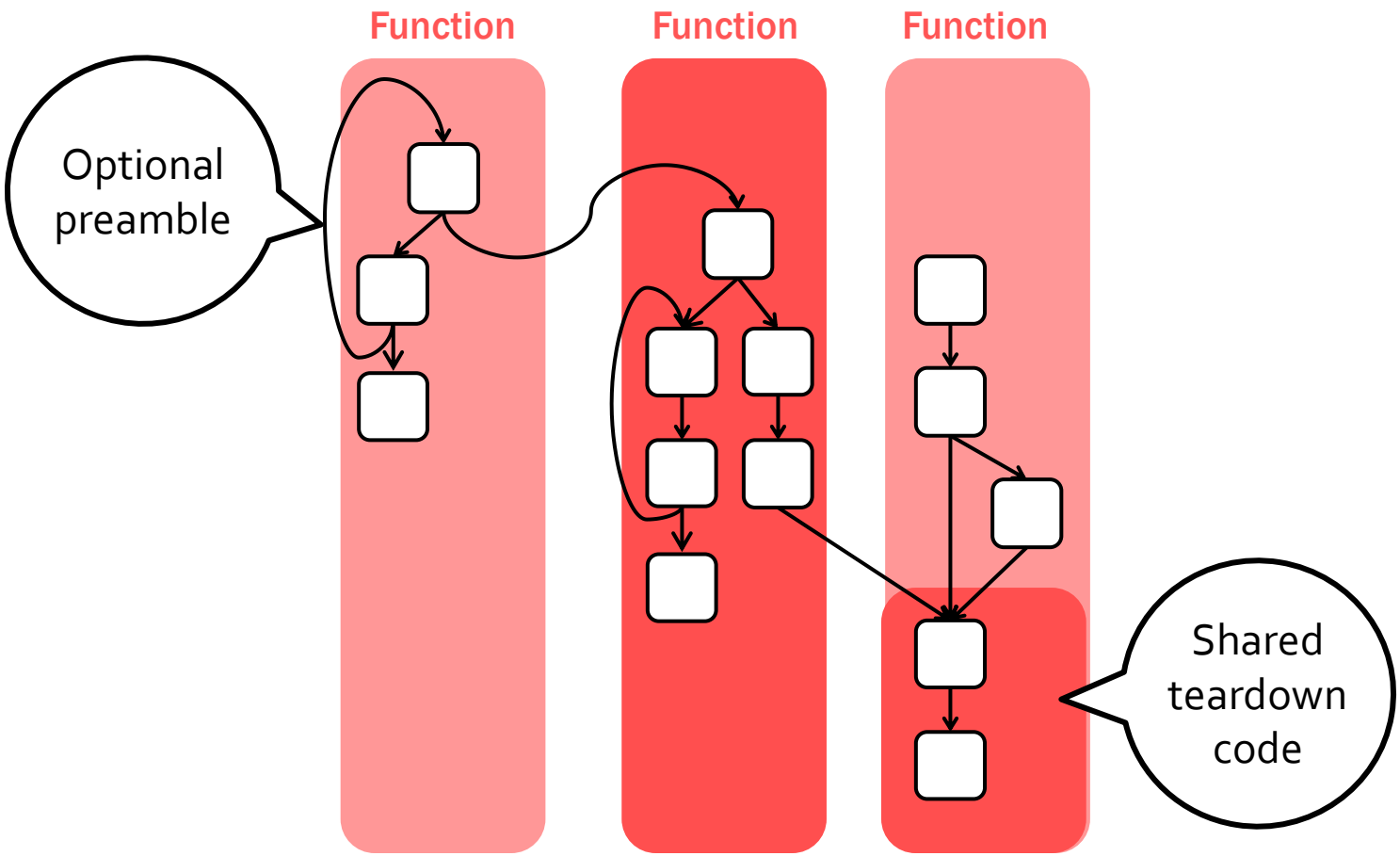
Extra call/ret instructions



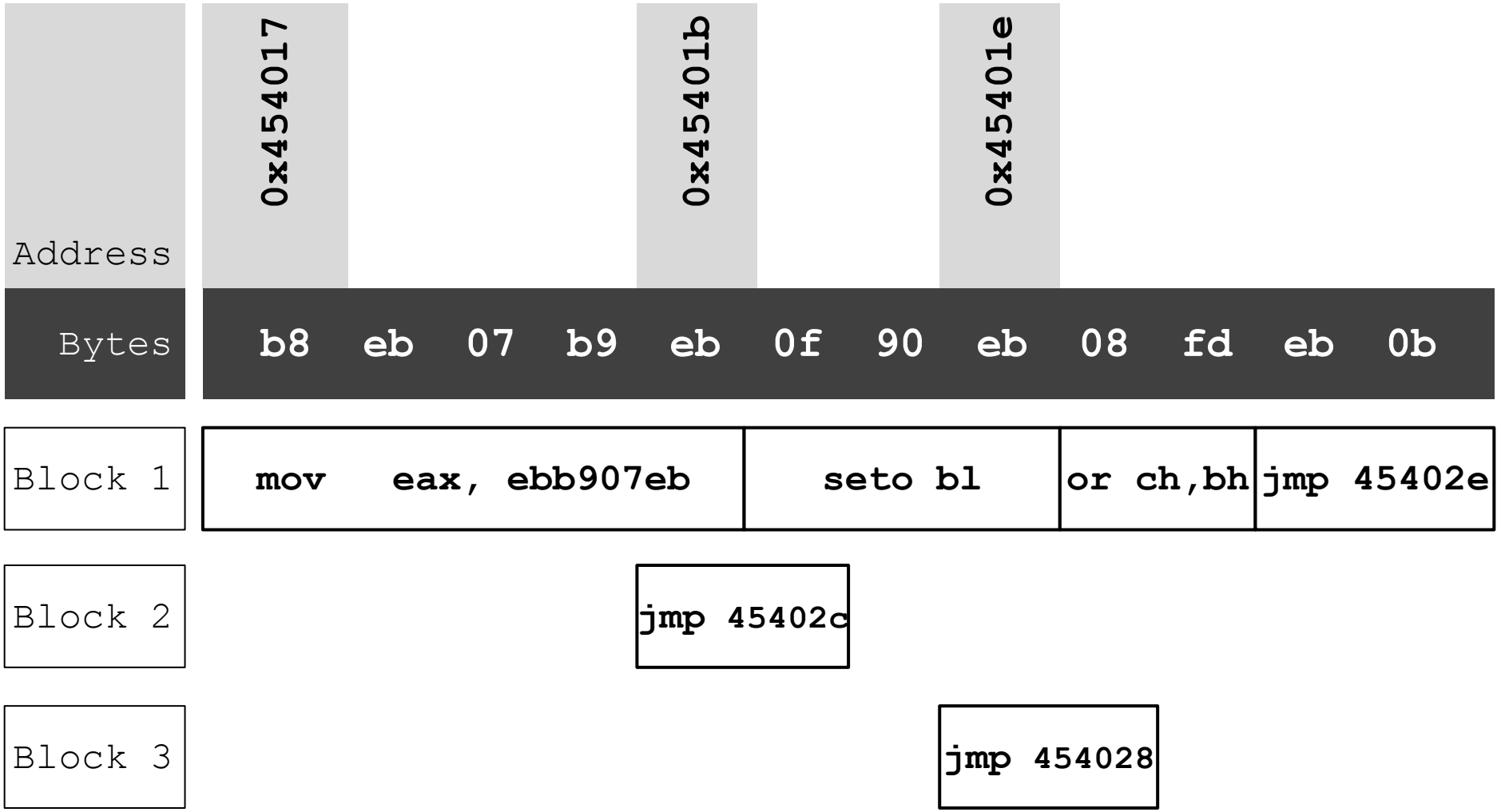
Overlapping functions



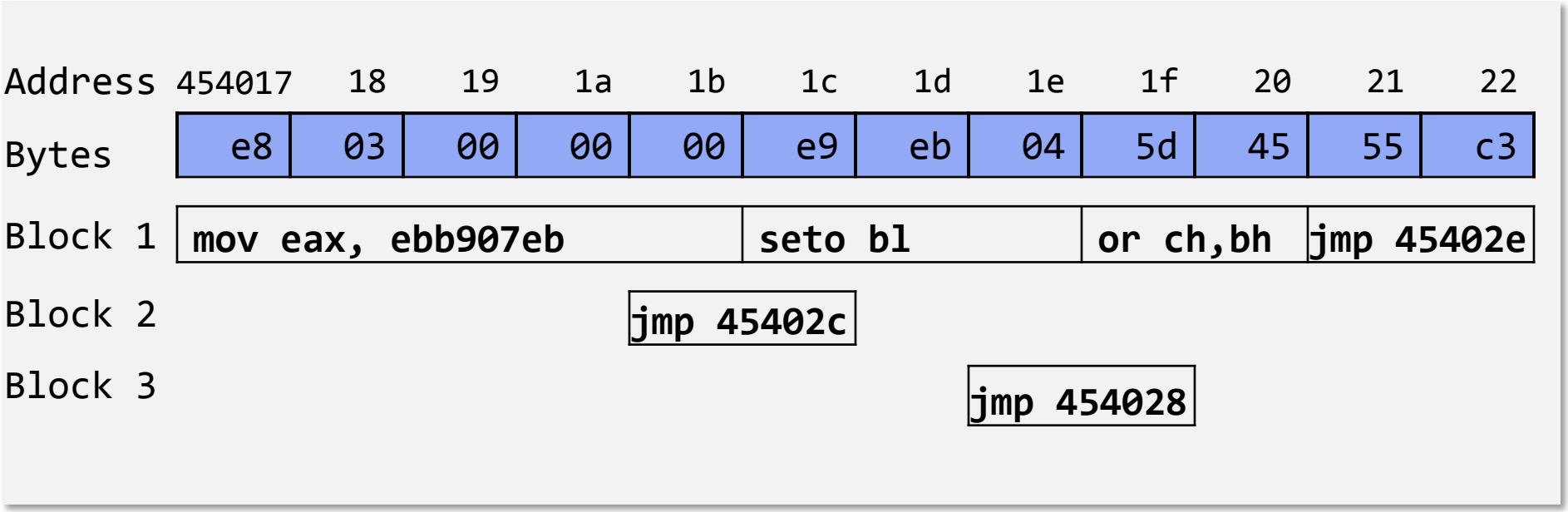
Overlapping functions



Overlapping blocks



Overlapping blocks



Outline

Analysis steps

1. Extract code bytes
2. Disassemble
3. Identify functions
4. Build comprehension tools
   a. Obfuscated constants
   b. ABI violations
   c. Do-nothing code
5. Patch/modify the binary
6. Trace code's execution

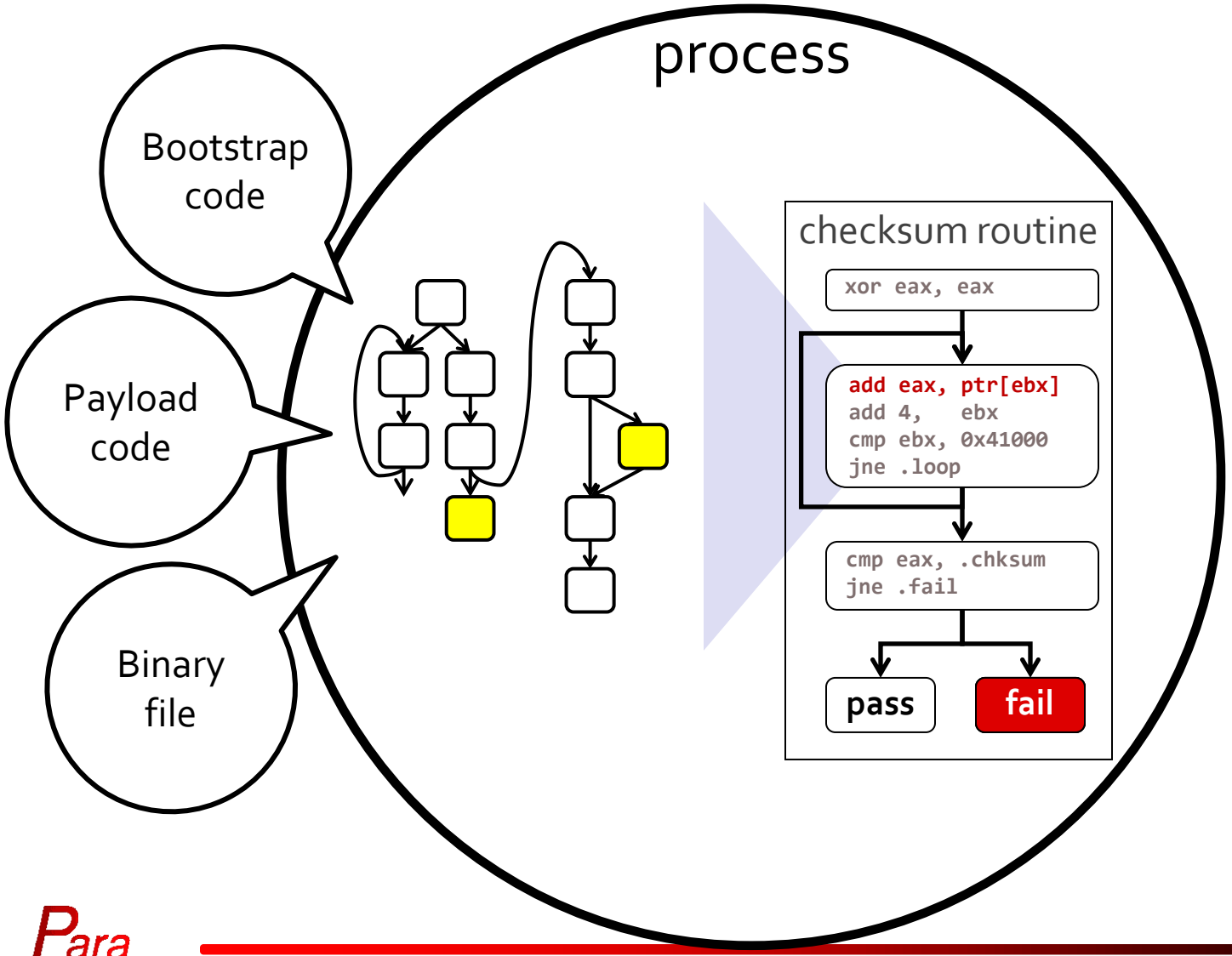
Outline

Analysis steps

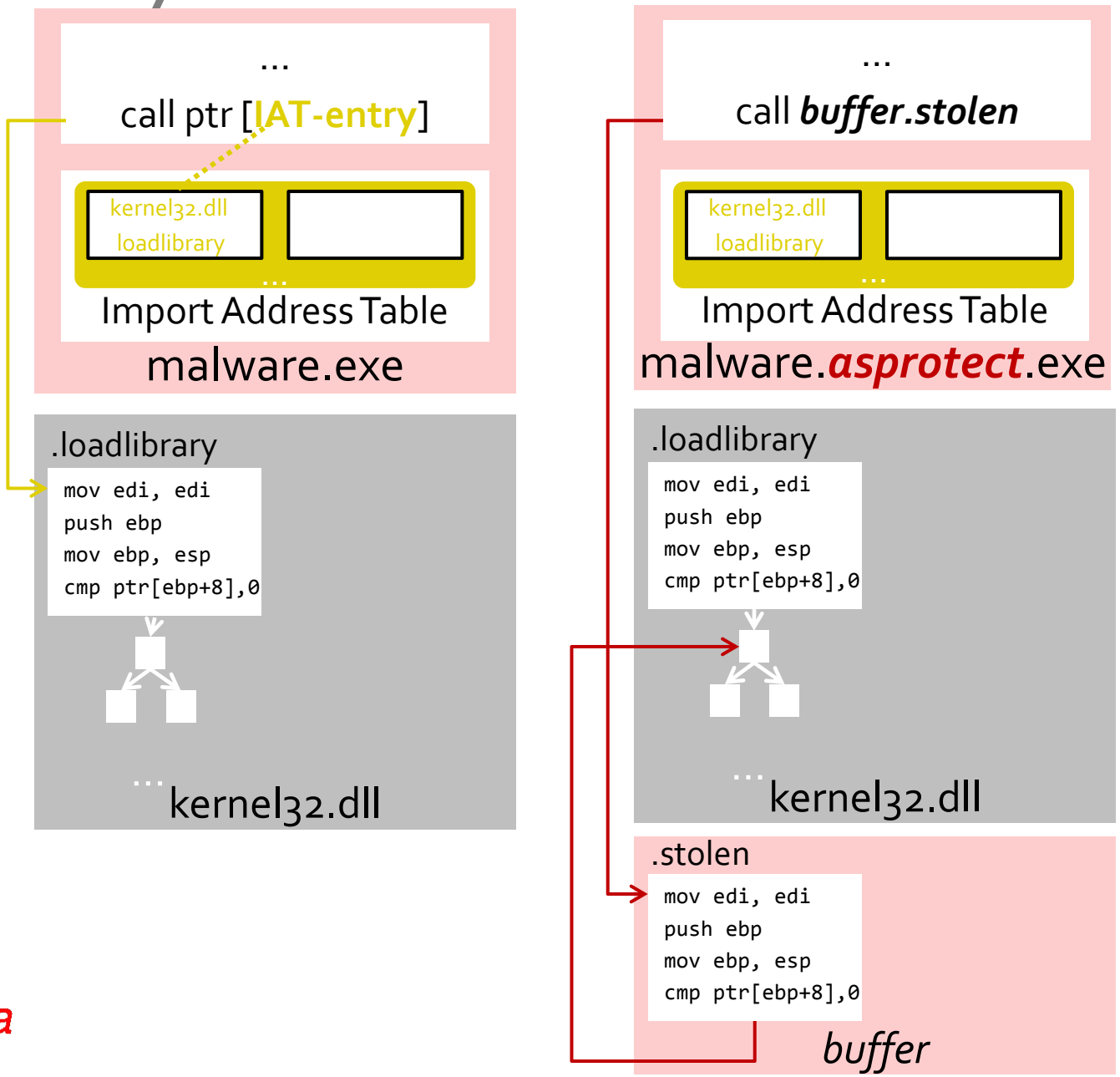
1. Extract code bytes
2. Disassemble
3. Identify functions
4. Build comprehension tools
5. Patch/modify the binary
6. Trace code's execution

a. Self-checksumming
b. Stolen bytes
c. Anti-unpacking

Self-checksumming



Stolen bytes



Outline

Analysis steps

1. Extract code bytes
2. Disassemble
3. Identify functions
4. Build comprehension tools
5. Patch/modify the binary
6. Trace code's execution

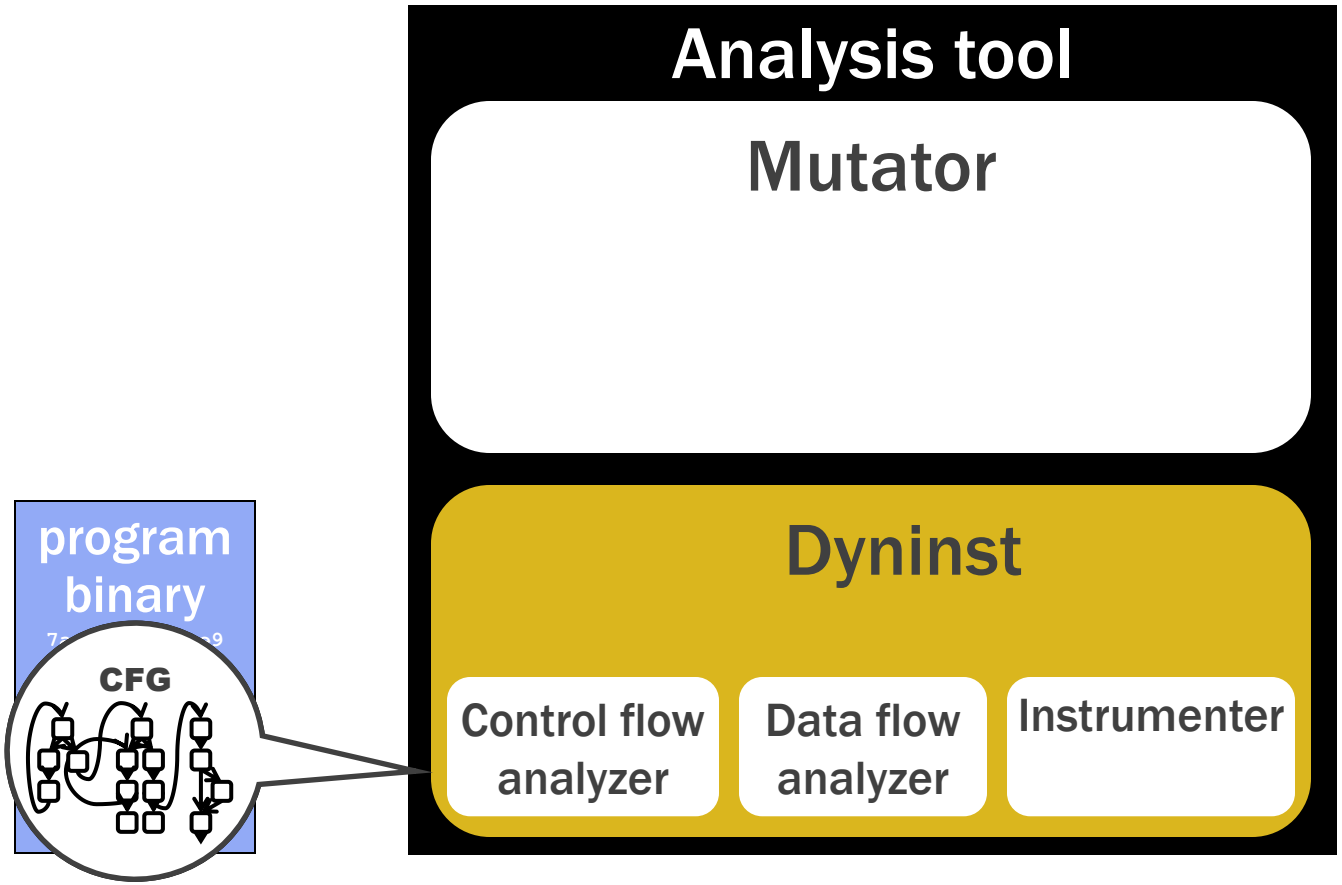
   a. Stolen bytes
   b. Non-standard API calls
   c. Anti-debugging

Outline

Analysis steps

1. Extract code bytes
2. Disassemble
3. Identify functions
4. Build comprehension tools
5. Patch/modify the binary
6. Trace code's execution

Adapting Dyninst for Malware



Adapting Dyninst for Malware