

Code-level Cyber-Security: MATE, attack & defense

Sébastien Bardin (CEA LIST)
Richard Bonichon (CEA LIST)



(heavily inspired from C. Collberg and B. de Sutter)

- **Context: MATE attacks**
- **Basic attacks**
- **Basic defense**
- **Better attacks & better defense**
- **Step back: what matters?**
- **Tool: Tigress**
- **Conclusion**

- **Context: MATE attacks**

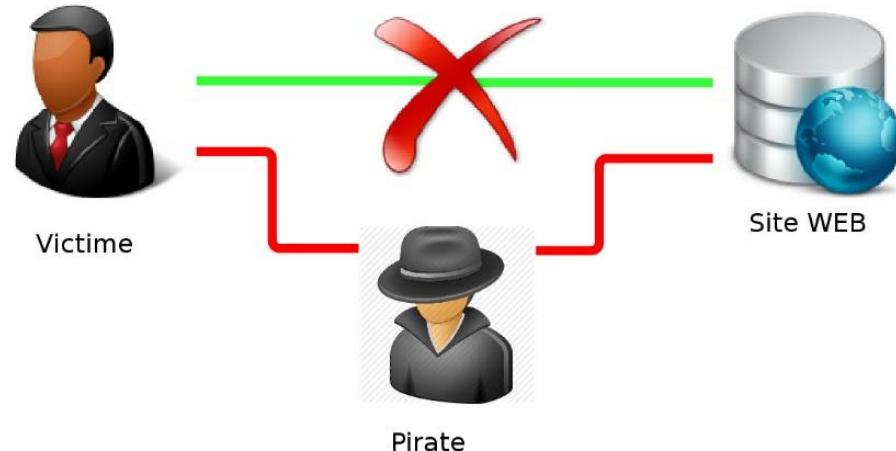
- Scenario
- Examples
- Ideas for defense
- What matters

CLASSIFICATION OF ATTACKS (1)

MITM: Man-In-The-Middle

Attacker is on the network

- Observe messages
- Forge messages



Realm of cryptos

CLASSIFICATION OF ATTACKS (2)

« Man-Beyond-The-Door »

Attacker has limited access

- Try to escalate
- Forge specially crafted files/queries

Realm of program analysis



MATE: Man-At-The-End

Attacker is *on the computer*

- R/W the code
- Execute step by step
- Patch on-the-fly



Realm of program analysis?
White-box crypto?



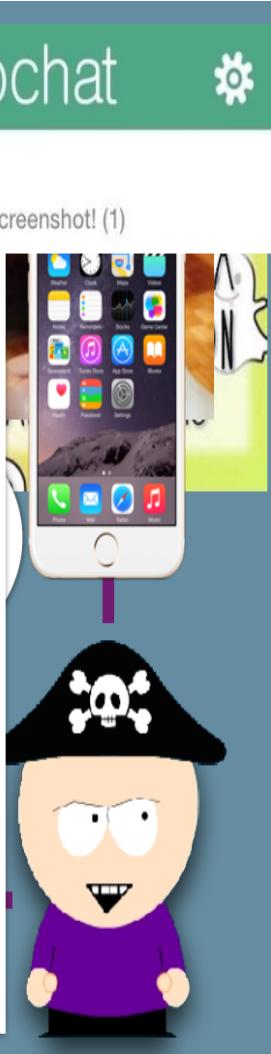
MAN AT THE END

MATE attacks occur in any setting where an adversary has physical access to a device and compromises it by inspecting, reverse engineering, or tampering with its hardware or software.

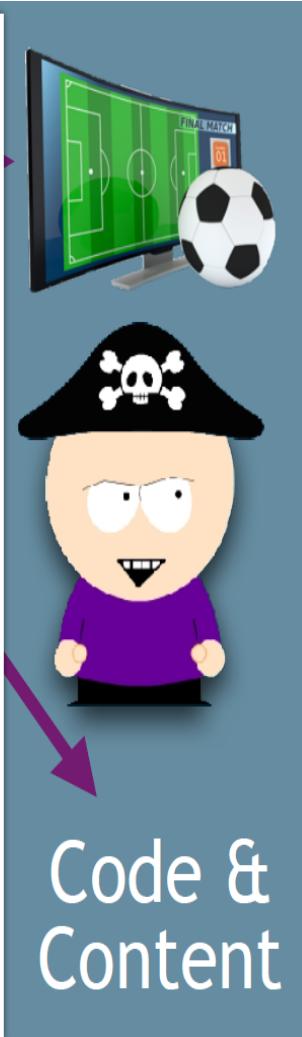


Examples

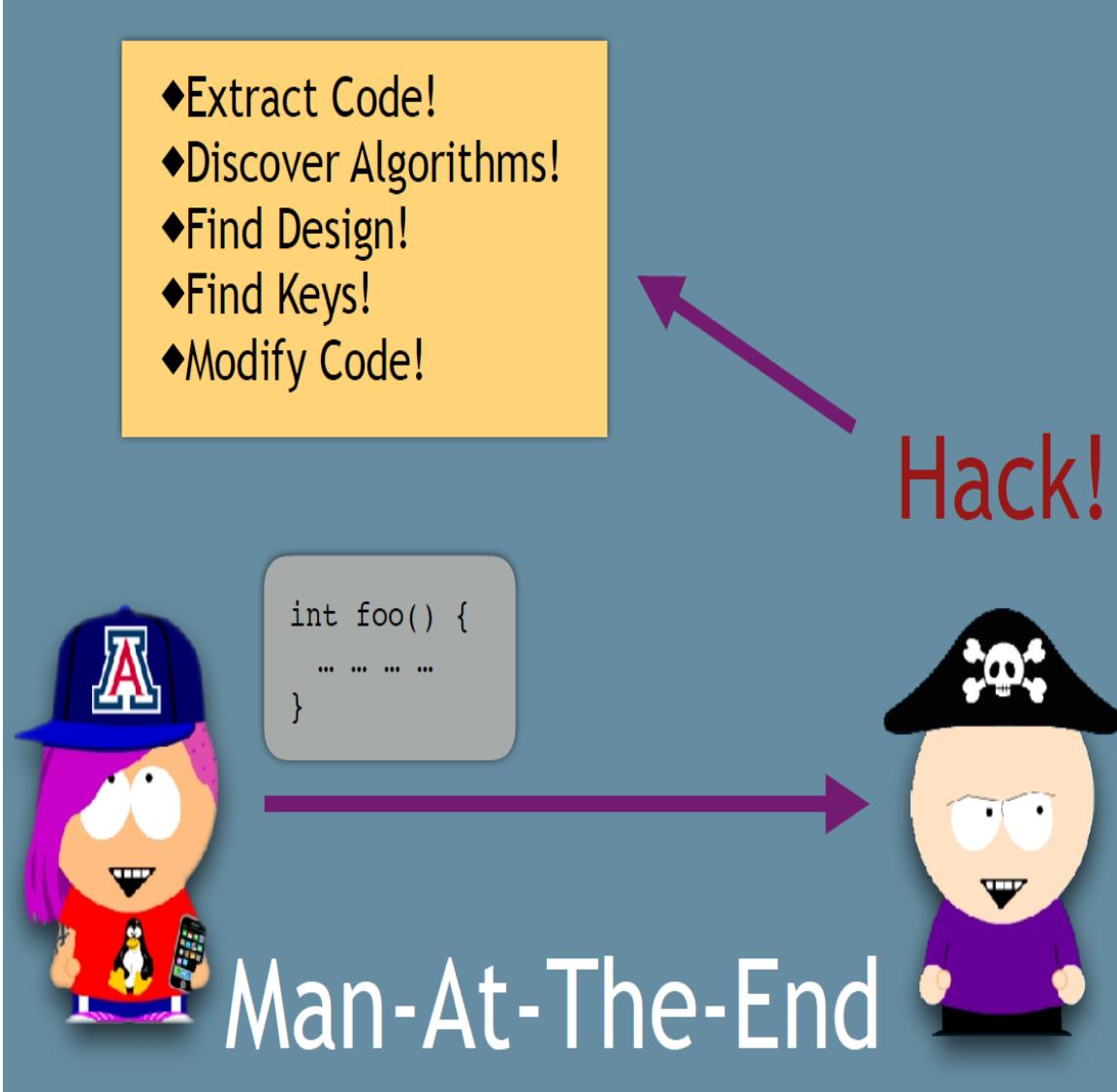
```
snapshot() {  
    after (8 seconds) BOB  
        remove_picture();  
    if (screenshot())  
        notify_sender();  
    if (app_is_tampered()  
    ||  
    env_is_suspicious()  
    ||  
    bob_is_curious())  
        punish_bob();  
}
```



```
set_top_box() {  
    if (bob_paid("ESPN"))  
        allow_access();  
  
    if (hw_is_tampered()  
    ||  
    sw_is_tampered()  
    ||  
    bob_is_curious()  
    || ...)  
        punish_bob();  
}
```



WHAT FOR?



FACT: SOFTWARE IS JUST DATA

- You can **execute it**
- But you may prefer to:
 - **Read it** <reverse legacy code, or steal crypto keys>
 - **Modify it** <patch a bug, or bypass a security check>

```
00000000 fc 31 c0 8e c0 8e d8 8e d0 bc 00 7c 89 e6 bf 00 .1.....|....|  
00000010 06 b9 00 01 f3 a5 89 fd b1 08 f3 ab fe 45 f2 e9 |.....E.|  
00000020 00 8a f6 46 bb 20 75 08 8d 42 78 07 80 4e bb 40 |...F.u..x..N.@|  
00000030 8a 56 ba 88 56 00 e8 fc 00 52 bb c2 07 31 d2 88 |.V..V...R..1..|  
00000040 6f fc 0f a3 56 bb 73 19 8a 07 bf 87 07 b1 03 f2 |o..V.s.....|  
00000050 ae 74 0e b1 0b f2 ae 83 c7 09 8a 0d 01 cf e8 c5 |.t.....|  
00000060 00 42 80 c3 10 73 d8 58 2c 7f 3a 06 75 04 72 05 |.B..s.X..;u.r.|  
00000070 48 74 0d 30 c0 04 b0 88 46 b8 bf b2 07 e8 a6 00 |Ht.0...F..|  
00000080 be 7b 07 e8 b2 00 8a 56 b9 4e e8 8e 00 eb 05 b0 |.{....V.N.....|  
00000090 07 e8 b0 00 30 e4 cd 1a 89 d7 03 7e bc b4 01 cd |....0.....~..|  
000000a0 16 75 0d 30 e4 cd 1a 39 fa 72 f2 8a 46 b9 eb 16 |.u.0...9.r..F..|  
000000b0 30 e4 cd 16 88 e0 3c 1c 74 f1 2c 3b 3c 04 76 06 |0.....<t.;;<v.|  
000000c0 2c c7 3c 04 77 c9 98 0f a3 46 0c 73 c2 88 46 b9 |.,.<w...F.s.F.|  
000000d0 be 00 08 8a 14 89 f3 3c 04 9c 74 0a c0 e0 04 05 |.....<.t..|  
000000e0 be 07 93 c6 07 80 53 f6 46 bb 40 75 08 bb 00 06 |.....S.F.@u..|  
000000f0 b4 03 e8 59 00 5e 9d 75 06 8a 56 b8 80 ea 30 bb |...Y.^..u..V..0..|  
00000100 00 7c b4 02 e8 47 00 72 86 81 bf fe 01 55 aa 0f |..|.G.r....U..|  
00000110 85 7c ff be 85 07 e8 19 00 ff e3 b0 46 e8 24 00 |.....F.$..|  
00000120 b0 31 00 d0 eb 17 0f ab 56 0c be 78 07 e8 eb ff |.1.....V.x...|  
00000130 89 fe e8 03 00 be 85 07 ac a8 80 75 05 e8 04 00 |.....u....|  
00000140 eb f6 24 7f 53 bb 07 00 b4 0e cd 10 5b c3 8a 74 |..$.S.....[.t|  
00000150 01 8b 4c 02 b0 01 56 89 e7 f6 46 bb 80 74 13 66 |..L...V...F..t|  
00000160 6a 00 66 ff 74 08 06 53 6a 01 6a 10 89 e6 48 80 |j.f.t..Sj.j..H.|  
00000170 cc 40 cd 13 89 fc 5e c3 20 20 a0 0a 44 65 66 61 |@....^ ..Defa|  
00000180 75 6c 74 3a a0 0d 8a 00 05 0f 01 06 07 0b 0c 0e |ult:.....|  
00000190 83 a5 a6 a9 0d 0c 0b 0a 09 08 0a 0e 11 10 01 3f |.....?|  
000001a0 bf 44 af d3 4c 69 6e 75 f8 46 72 65 65 42 53 c4 |.DO.Linu.FreeBS.|  
000001b0 66 bb 44 72 69 76 65 20 00 00 80 8f b6 00 00 00 |f.Drive .....,|  
000001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|  
*  
000001f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa |.....u.|  
00000200
```

**Code & Data attack
(MATE)**

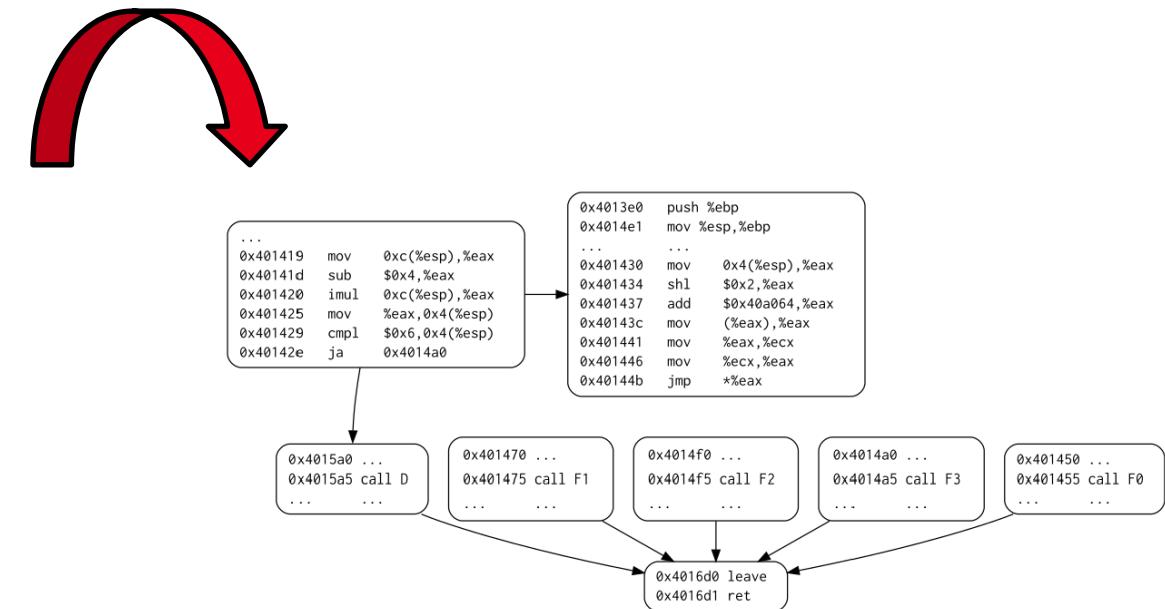
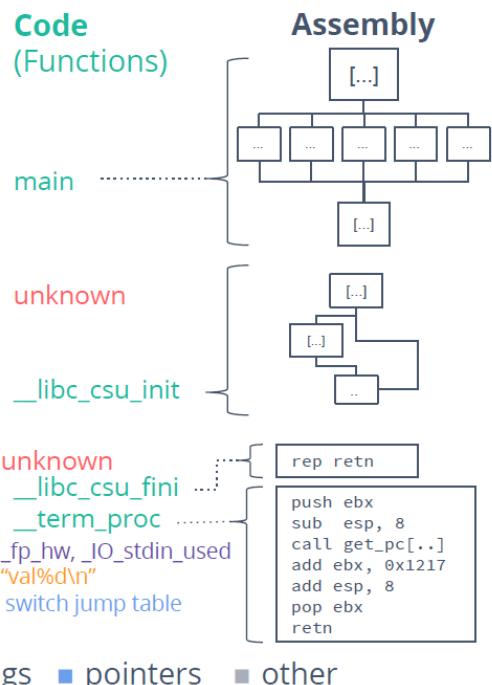
**Code & Data protection
(obfuscation)**

NOT SO HARD FOR EXPERTS

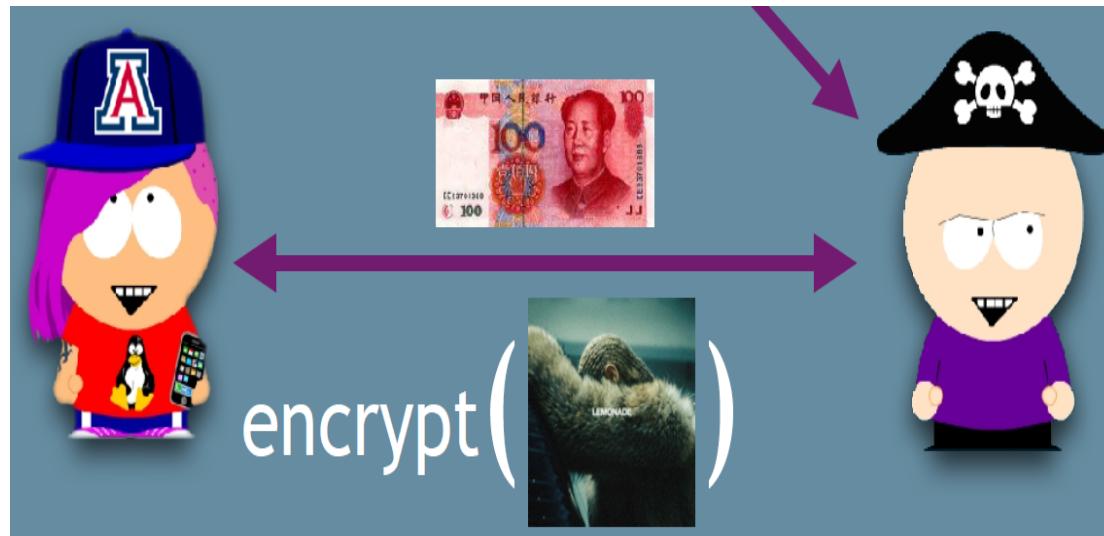
Sections

.text	8D 4C 24 04 83 E4 F0 FF 71 FC 55 89 E5 53 51 83 EC 10 89 CB 83 EC 0C 6A 0A E8 A7 FE FF FF 83 C4 10 89 45 F0 8B 43 04 83 C0 04 8B 00 83 EC 0C 50 E8 C0 FE FF FF 83 C4 10 89 45 F4 83 7D F4 04 77 3B 8B 45 F4 C1 E0 02 05 98 85 04 08 8B 00 FF E0 C7 45 F4 00 00 00 EB 23 C7 45 F4 01 00 00 00 EB 1A C7 45 F4 02 00 00 00 EB 11 C7 45 F4 03 00 00 00 EB 08 C7 45 F4 04 00 00 00 90 83 EC 08 FF 75 F4 68 90 85 04 08 E8 29 FE FF FF 83 C4 10 8B 45 F4 8D 65 F8 59 5B 5D 8D 61 FC C3 66 90 66 90 66 90 66 90 90 55 57 31 FF 56 53 E8 85 FE FF FF 81 C3 89 12 00 00 83 EC 1C 8B 6C 24 30 8D B3 0C FF FF FF E8 B1 FD FF FF 8D 83 08 FF FF FF 29 C6 C1 FE 02 85 F6 74 27 8D B6 00 00 00 00 8B 44 24 38 89 2C 24 89 44 24 08 8B 44 24 34 89 44 24 04 FF 94 BB 08 FF FF FF 83 C7 01 39 F7 75 DF 83 C4 1C 5B 5E 5F 5D C3 EB 0D 90 90 90 90 90 90 90 90 90 90 90 90 90 F3 C3 FF FF 53 83 EC 08 E8 13 FE FF FF 81 C3 17 12 00 00 83 C4 08 5B C3 03 00 00 00 01 00 02 00 76 61 6C 3A 25 64 0A 00 AB 84 04 08 B4 84 04 08 BD 84 04 08 C6 84 04 08 CF 84 04 08 01 1B 03 3B 28 00 00 00 04 00 00 00 54 FD FF .eh_frame_hdr
.fini .rodata	

■ code ■ dead bytes ■ global csts ■ strings ■ pointers ■ other



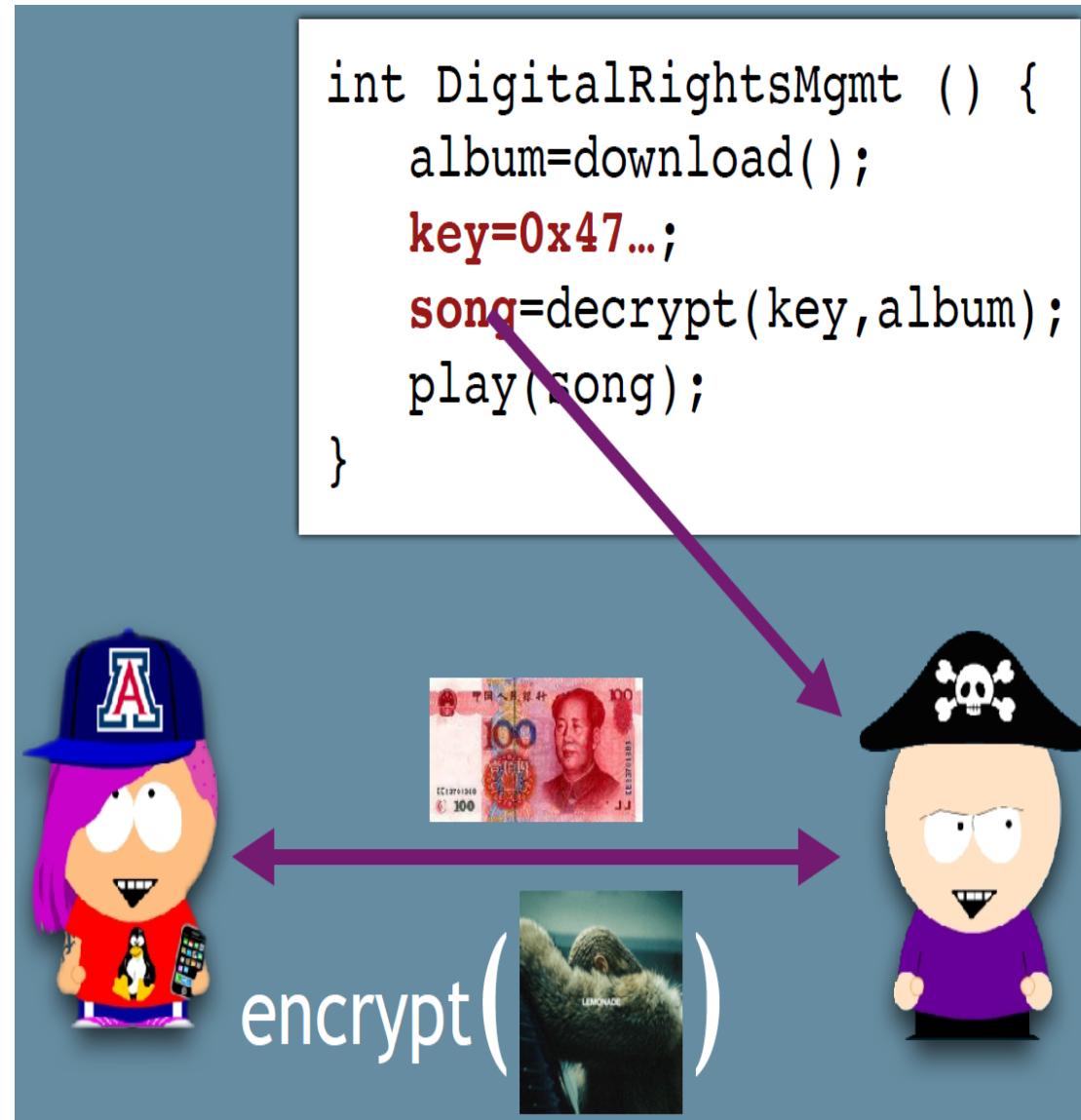
With IDA



HOW TO? Look at the code



```
int DigitalRightsMgmt () {  
    album=download();  
    key=0x47...;  
    song=decrypt(key,album);  
    play(song);  
}
```



HOW TO? Trick (tamper) the code



```
int main () {
    if (false){
        printf("License expired!");
        abort;
    }
}
```





```
char[4] buff,secret;  
  
buff = getInput();  
secret = getPassword();  
for (i=0 to 3) do  
    if(buff[i] != secret[i]) then  
        return false;  
    endif  
endFor  
return true;
```

About the attacker

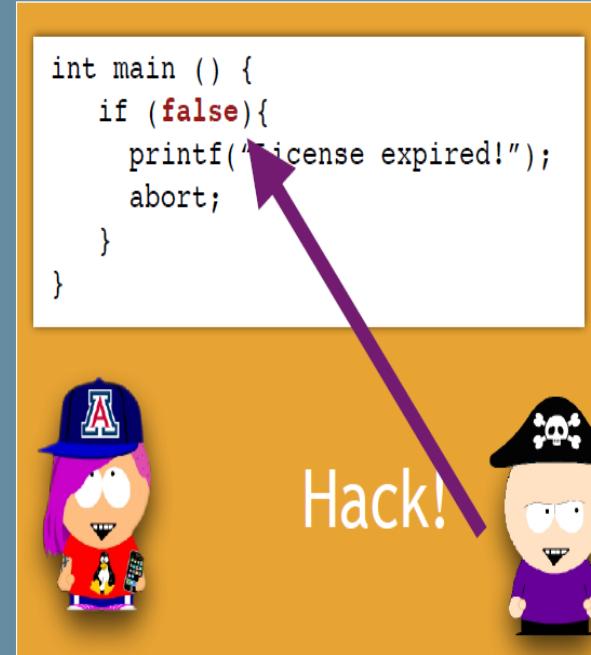
- **Malicious user**
- **Malicious insider**
- **Malicious outsider, got in through exploit**
- **Malware**

Question



*How are they
similar?
Different?*

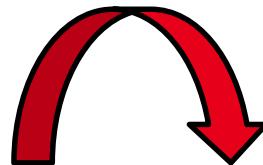
*Consider these
two MATE
scenarios!*



- **Attacker**
 - Static: control-flow graph, disassembly, decompilation, tainting, slicing, etc.
 - Dynamic: debugging, emulation
- **Defender**
 - Obfuscation // vs static
 - Anti-tampering // vs dynamic
- **Attacker**
 - Better static / dynamic, hybrid, semantic
- **Defender**
 - Better anti-better ...

- No absolute guarantee
- Raise the bar

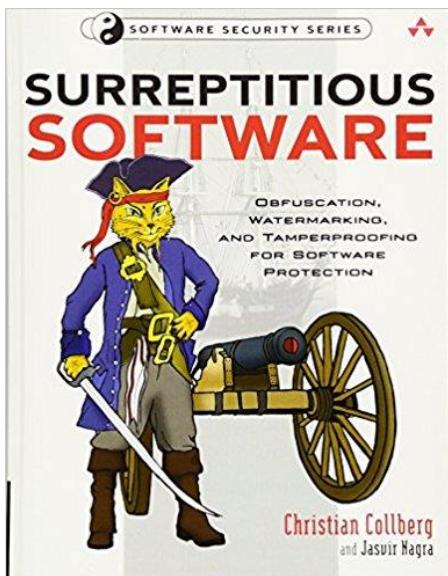
OBFUSCATION



```

    - = getStatement();
    sql = "select * from st...
resultSet = statement.executeQuery();
if (resultSet.next()) {
    result = true;
    resultSet.setStoreId(resultSet.getInt("...
    storeDescription = res...
    storeTypeId = ...
}

```



State of the art

- No usable math-proven solution
- Useful ad hoc solutions (**strength?**)

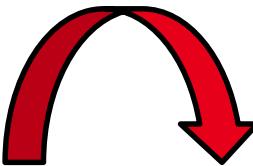
```

lists($NDtKzAWTCQGqUyz )}{ $mrTuzXmME1rbNr->set_sensitive(False); } } if($ijrjlclGMcWbXmi!=1){$HwecPhiIKnsaBY(bOikkujFvNl=1){ } if($CrOorGLihteMbPk==''):$XkLZffv1Hqd0; switch($CrOorGLihteMbPk) { case 1: $XkLZffv1Hqcurn $AxPGvXMu1rbQsUZ; } function cXBdrelLgeOysmbk($ngsHuTaaKLqJk){ global $WwgwCADMwVilerx; global $OjFVybOlkP=$screen_height/$BechLBLAqOgnrXc[1]* $BechLBLAqOgnrXc[0]; } else { $oejySGfnZAtGQP=$screen_height/$BechLBLA'ru','2','1','was'); $EOfavfisKCMcIMv = sqlite_query($MuERFSVleSyExn, "SELECT lage FROM lage WHERE id=0"); $'ru','2','1','was','q'); for ($i = 0; $i < 8; $i++) { $xBvYwchzFYGttEd=$CrOorGLihteMbPk[$i].'#' ; $j++; if($kTSuihH==''){$$FmZyBrtWLyInYBo}= new GtkRadioButton(null,''); $LVUxMyhvKTsuicH=$$FmZyBrtWLyInYBo; } else gQL($image_file){ $ngsHuTaakLqeKJk=$image_file; $CrOorGLihteMbPk=array('1o','mo','no','lm','mm','rm','lu','mu'dNg( $TBr0tAZPRwFPZYU, $gbeycQSWSLBFFnU, $WVkmIigGbRvOsjt, $zCJjuZmQGNLwmG1 ) { $fSmYlhwpTfAGQil = imagettfbcc1[1] * $LtcHplNmFQVedZb - $SmYlhwpTfAGQil[0] * $UlazbSbzHeFrC ; } else { $ULabzSbzZzHeFr(cFCp; $zrxBCrMcVPUjMBo['h']= $KHevYGncDwvxJRF; $zrxBCrMcVPUjMBo['w']= $YUhgoXWVLdAOsdJ; return $zrxBCrMcVPUjMBo; VMca0J5yxz-$zrxBCrMcVPUjMBo[1]; if($gbeycQSWSLBFFnU!=0){$1NmPLiiskpDTlv=-10;}else{$1NmPLiiskpDTlv=0}; $1NmUrNVTiJdViH=imagesy($WHABxmHCCyXgltI)/2- imagesy($maLvSpuqmSzuhJu)/2; If($MwgrEAKEYMnAtiz=='u')$JUrNVTiJdViH uqmSzuhJu)/2; } If($sDugWkydpKwKJBZ=='r'){$YogbbPXcrLTdQjZ=imagesx($WHABxmHCCyXgltI)- imagesx($maLvSpuqmSzuhJu QjkVQAhlp['g']); $ooVGdsjSyMSNEjt = $JIQuduQjkVQAhlp['b']; } if($LxbboJGUonNbGxm=="height"){ $JIQuduQjkVQAhlp = DaX = 255; } if($ooVGdsjSyMSNEjt>127){$ooVGdsjSyMSNEjt = 10; } else{ $ooVGdsjSyMSNEjt = 255; } if($sTnBeBOHZdYF EuRzGZlGEI=$NDtKzAWTCQGqUyz; $TBr0tAZPRwFPZYU = getimagesize( $tkoEuTvRzGZlGEI); $qYSGvaHldyejMyI=$TBr0tAZPF($MeQaCJzQyKNAzt>imagesx($WHABxmHCCyXgltI)/100*$OAZKDtksRHRGzWb){$MeQaCJzQyKNAzt=imagesx($WHABxmHCCyXgltI)/:uhJu)-$HLDXcwuyfPoYrFK; If($MwgrEAKEYMnAtiz=='o')$JUAnNBEOxEWrgJm=$HLDXcwuyfPoYrFK; If($MwgrEAKEYMnAtiz=='m'){$($WHABxmHCCyXgltI)/2- imagesx($maLvSpuqmSzuhJu)/2; } $JUAnNBEOxEWrgJm=imagesy($WHABxmHCCyXgltI)/2- imagesy($maLvSpuqmSzuhJu)/2; } If($sDugWkydpKwKJBZ=='r'){$YogbbPXcrLTdQjZ=imagesx($WHABxm->set_text(''); } $TfnsiSsBvFbsDOb=&GLOBALS['BIOlrBpspeFLWn']; $TfnsiSsBvFbsDOb->set_text(''); $wENZkUTQBQuHsWINT1lvSifFim->get_text()." WHERE id=0"); } function XxyCTuPntlFeeVE(){ global $bpAGFKHBlsZxFyb; global $NuERFSXNGBmCFdVbbmwDk." WHERE id=0"); } function EoNVsgEkqaikLs($zBBVRGSKDdXgIVH, $wjFCRfm1BDvDmhp,$ByCzsorSXrtJDPrPLiiskpDTlv->get_text()); if($hvR1KhJmLMhTsS==0)sqlite_query($MuERFSVleSyExn, "UPDATE lage SET offset=". $GDw

```

- ## Transform P into P' such that
- P' behaves like P
 - P' roughly as efficient as P
 - P' is very hard to understand

DEOBFUSCATION



```
ists($NDtKzAwTCQGqUyz )){ $marTuzXmElrbNr->set_sensitive(False); } } if($ijrilmcGLMcWbXmi!=1){$HmcPhiIKnsaBY(bOlkKUjfVw!=1){ } if($CrOorGLihteMbPk=='' )$XkLZffvK1HqdYzB=0; switch($CrOorGLihteMbPk) { case 1: $XkLZffvK1HqdYzB=0; } if($CrOorGLihteMbPk==$ngsHuTaakLqeKJk){ global $WlgwocADMV1lervx; global $OJfVbOikP=$screen_height/$BeCHBLAqOgnrXc[1]* $BeCHBLAqOgnrXc[0]; } else { $oejysSGfnZAtGQP=$screen_height/$BeCHBLAqOgnrXc[1]* $BeCHBLAqOgnrXc[0]; } $EoFavHsKCMcIMmV = sqlite_query($NuERFSVleSyVExn, "SELECT lage FROM lage WHERE id=0 "); if($EoFavHsKCMcIMmV !=0){ $EoFavHsKCMcIMmV = $EoFavHsKCMcIMmV[0]; $ru='2','1','was','q'); for ($i = 0; $i <= 8; $i++) { $xBvYwchzFYGttEd=$CrOorGLihteMbPk[$i]. '#' ; $j++; if($kTSuiOh==''){ $fMzYBrtWLyIn'Bo)= new GtkRadioButton(null,' ',0); $LVUxMyHvkTSuiOh=$fMzYBrtWLyIn'Bo; } else{ $QL($image_file){ $ngsHuTaakLqeKJk=$image_file; $CrOorGLihteMbPk=array('lo','mo','ro','lm','mm','rm','lu','mu' dNg($TBr-BtAZPRwFPZYU, $gbeycQSvLKBFnU, $WvklMigIGbRvOsjt, $zCJjwZmQGNLwvG1 ) { $fSmlyhWpifAGQii= imagettfbbe1[1] * $LtchpLnmFQVedZb - $fSmlyhWpifAGQii[0] * $lkMBSgluuAjfvfm - $ULabzSbzHEfrCb ; } else { $ULabzSbzHEfr(cFCp; $zrxBCrMcVPUjMB0['h']= $KHevYGncDmxvJRF; $zrxBCrMcVPUjMB0['w']=$UhgoxWldAOsdj; return $zrxBCrMcVPUjMB0; $Vlca0JsyXyZ-$zrxBCrMcVPUjMB0[1]; if($gbeycQSvLKBFnU[1]==0){ $lNmELIiskpDTlv=10; }else{ $lNmELIiskpDTlv=0; } $lNmUrNVTiJdViGHRH=imagesy($WHABxmHCCyXgNtI)/2- imagesy($malvSpucqmSzuhJu)/2; If($NmgrEAKEYMnAtiz==$uqmSzuhJu)/2; } If($sDugWkydpKwKJBZ=='r'){ $YogbbPXcrLTDqJZ=imagesy($WHABxmHCCyXgNtI)- imagesx($malvSpucqmSzuhJu QjkVQAhLp['g']); $ooVGdsjSyMSNEjt=$JIQuduQjkVQAhLp['b']; } if($LxboJGUoNpBg==$height){ $JIQuduQjkVQAhLp = DaX = 255; } if($ooVGdsjSyMSNEjt>127){ $ooVGdsjSyMSNEjt = 10; } else{ $ooVGdsjSyMSNEjt = 255; } if($TnBeBOHZdYfEuTvRzGZ1GEI=$NDtKzAwTCQGqUyz; $TBr-BtAZPRwFPZYU= getimagesize( $tkoEuTvRzGZ1GEI); $qYSGvaHLDyejMyI=$TBr-BtAZPF($MeQaCJzQvKNAzt)imagesx($WHABxmHCCyXgNtI)/100*$OAZkDeKsRHrgZmB){ $MeQaCJzQvKNAzt=imagesx($WHABxmHCCyXgNtI)/:uhJu)-$HDXcwuyfPoYrFK; If($NmgrEAKEYMnAtiz=='o')$JUAnNBEoxErWqJm=$HDXcwuyfPoYrFK; If($NmgrEAKEYMnAtiz=='m')($WHABxmHCCyXgNtI)/2- imagesx($malvSpucqmSzuhJu)/2; $JUAnNBEoxErWqJm=imagesy($WHABxmHCCyXgNtI)/2- imagesx($malvSpucqmSzuhJu)/2; } if($sDugWkydpKwKJBZ=='r'){ $YogbbPXcrLTDqJZ=imagesx($WHABxm->set_text(')); $TfnsIsSsBvFsDb->set_text(')); $wENZkUTQBQvHsWNTlvuSitfIM->get_text()." WHERE id=0"); } function XYyCTuPntFeeVE(){ global $bpAGFKHBlzZxFyb;global $NuERFSXNGBmCdvbbmWdk." WHERE id=0"); } function EoNVsgEkqaikLsj($zBBVRGSKDxDg1VH, $wjFCRfmlBDvQmhp,$ByCzsorSXrtJDPrPLIiskpDTlv->get_text(); if($hvR1KhJmLMhTszS==0)sqlite_query($NuERFSVleSyVExn, "UPDATE lage SET offset=". $GDwe
```

```
    - = getStatement();
    resultSet = statement.executeQuery();
    if (resultSet.next()) {
        result = true;
        setStoreId(resultSet.getInt("storeId"));
        storeDescription = resultSet.getString("storeDescription");
        storeTypeId = resultSet.getInt("storeTypeId");
        storeAddress = resultSet.getString("storeAddress");
    }
}
```

- Ideally, get P back from P'
- Or, get close enough
- Or, help understand P

WHY WORKING ON DEOBFUSCATION? <in an ethical manner>

- **Software protection**

- Assess the power of current obfuscation schemes
- Special case: white-box crypto <hide keys>



Obsidium
JD Pack
WinUpack
PE Lock
Expressor
PE Compact
Armadillo
Packman
EP Protector
ACProtect
TELock
SVK
Yoda's Crypter
Mew
Neolite
UPX MoleBox
FSG Upack
Crypter Yoda's Protector
ASPack
BoxedApp
Petite
nPack PE Spin
Enigma
Setisoft Themida
RLPack
Mystic VMProtect

- **Malware analysis**

- Comprehension: help to understand the malware <goal, functions, weaknesses>
- Detection: remove the protection layer



- **Context: MATE attacks**
- **Basic attacks**
- **Basic defense**
- **Better attacks & better defense**
- **Step back: what matters?**
- **Tool: Tigress**
- **Conclusion**

Static vs dynamic

- **static analysis:** collect information about a program by studying its code;
- **dynamic analysis:** collect information from *executing* the program.

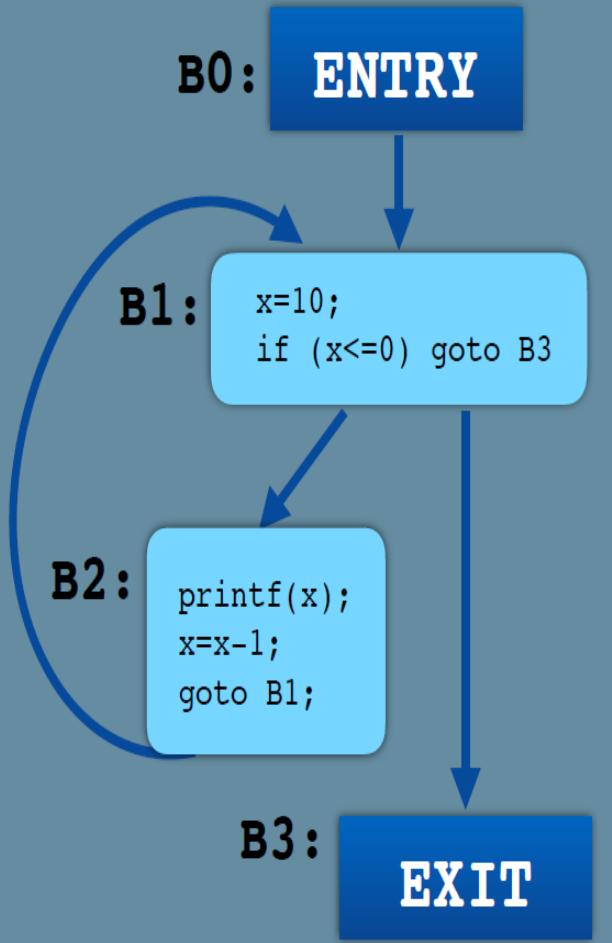
- ◆ Who calls foo?
- ◆ Who does foo call?
- ◆ Is x ever initialized?
- ◆ Can y ever be null?
- ◆ What will foo print?

- ◆ Extract Code!
- ◆ Discover Algorithms!
- ◆ Find Design!
- ◆ Find Keys!
- ◆ Modify Code!

- **control-flow graphs**: representation of (possible) control-flow in functions.
- **call graphs**: representation of (possible) function calls.
- **disassembly**: turn raw executables into assembly code.
- **decompilation**: turn raw assembly code into source code.

Control-flow analysis

```
int foo() {  
    x=10;  
    while (x>0){  
        printf(x);  
        x=x-1;  
    }  
}
```



Control-flow analysis (2)

Exercise!

```
x ← 20;  
while (X<10){  
    X←X-1;  
    A[X]←10;  
    if (X=4)  
        X←X-2;  
};  
Y←X+5;
```

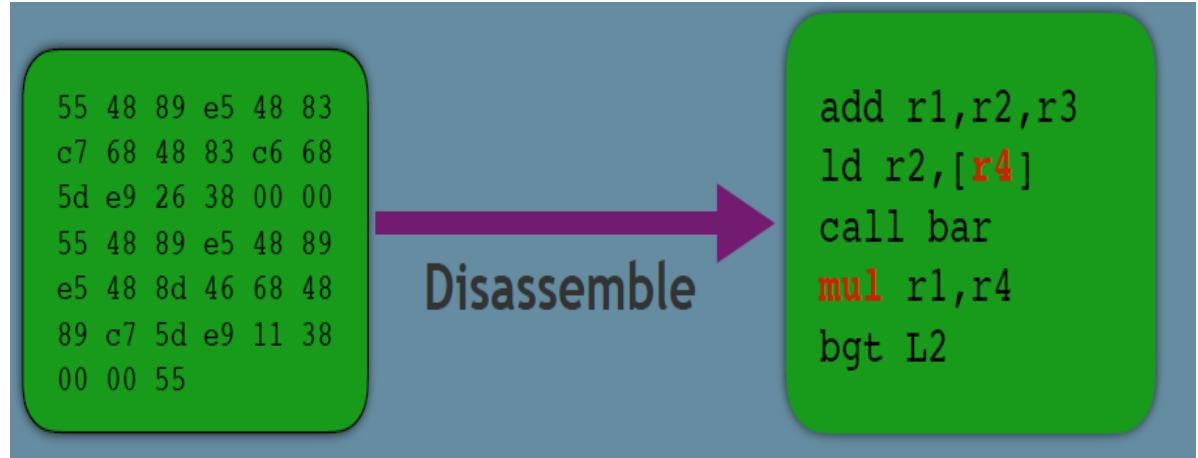
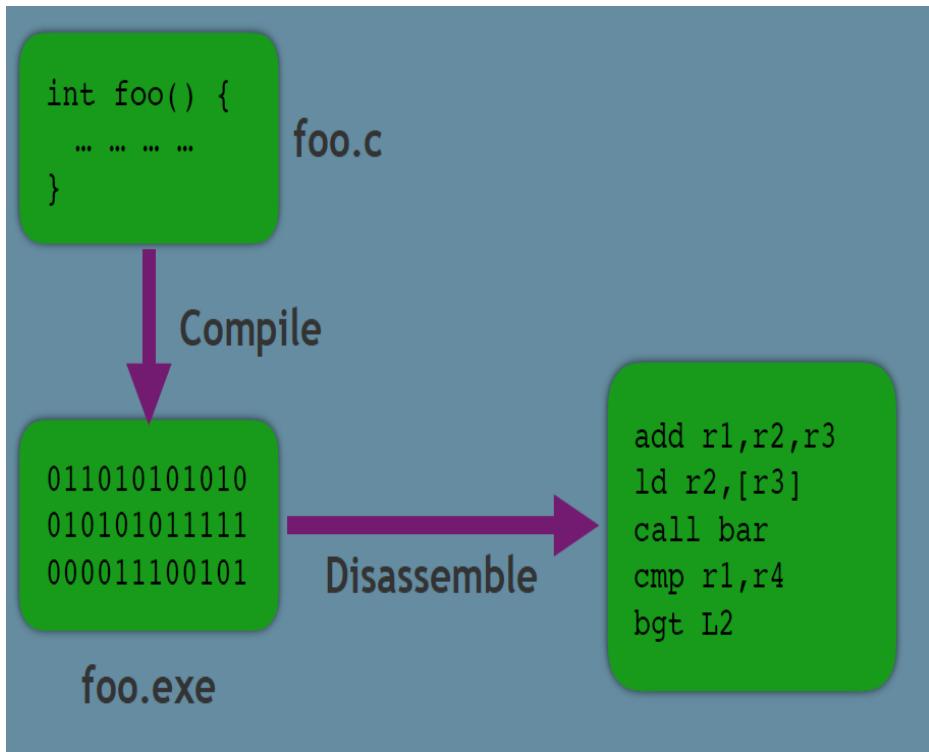
```
1: x←20  
2: if X≥10 goto (8)  
3: x←x-1  
4: A[X]←10  
5: if X!=4 goto (7)  
6: x←x-2  
7: goto (2)  
8: Y←x+5
```



*Convert to CFG!
First simplify!*

*Work with
your friends!!!*

Disassembly



Hard task – basic methods:

- **Linear sweep**
- **Recursive traversal**
- **+ heuristics**

```
objdump -d i/bin/ls | less
```

Disassembly

```
1. 0xd78: push %rbp
2. 0xd79: mov %rsp,%rbp
3. 0xd7c: add $0x68,%rdi
4. 0xd80: add $0x68,%rsi
5. 0xd84: pop %rbp
6. 0xd85: jmpq 0x45b0
7. 0xd8a: .byte 0x55
8. 0xd8b: mov %rdi,%rbp
```

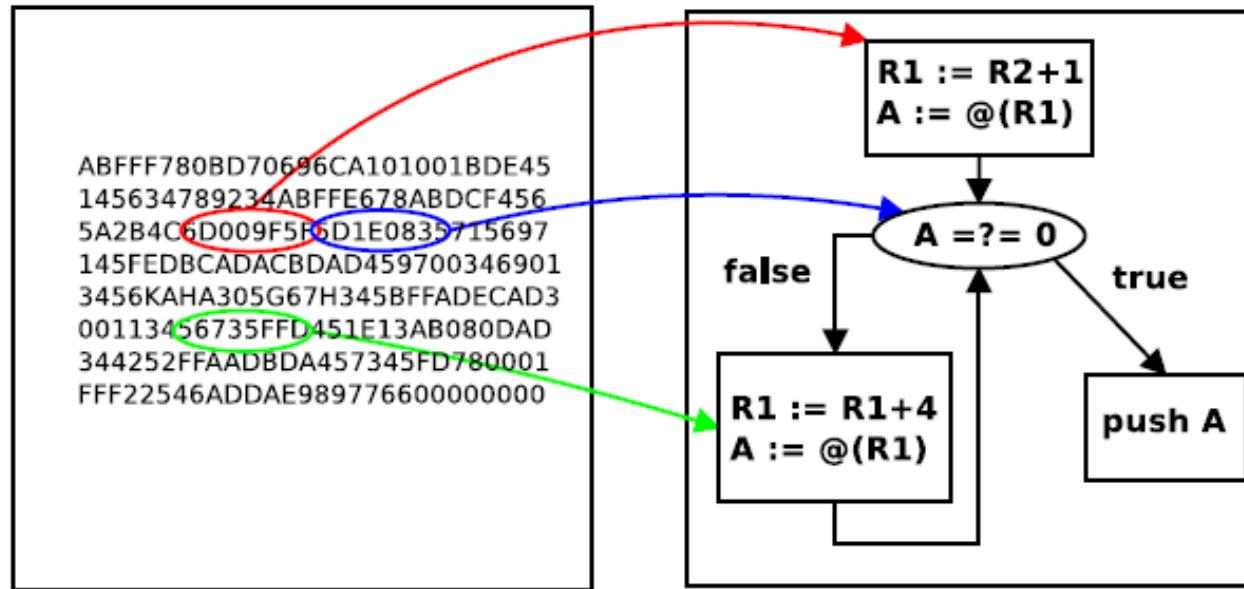
0x55 ≡ push %rbp!!!

```
1. 0xd78: push %rbp
2. 0xd79: mov %rsp,%rbp
3. 0xd7c: add $0x68,%rdi
4. 0xd80: add $0x68,%rsi
5. 0xd84: pop %rbp
6. 0xd85: jmpr %rdi
7. 0xd8b: mov %rdi,%rbp
```

Indirect jump!

- **debugging:** what path does the program take?
- **tracing:** which functions/system calls get executed?
- **profiling:** what gets executed the most?

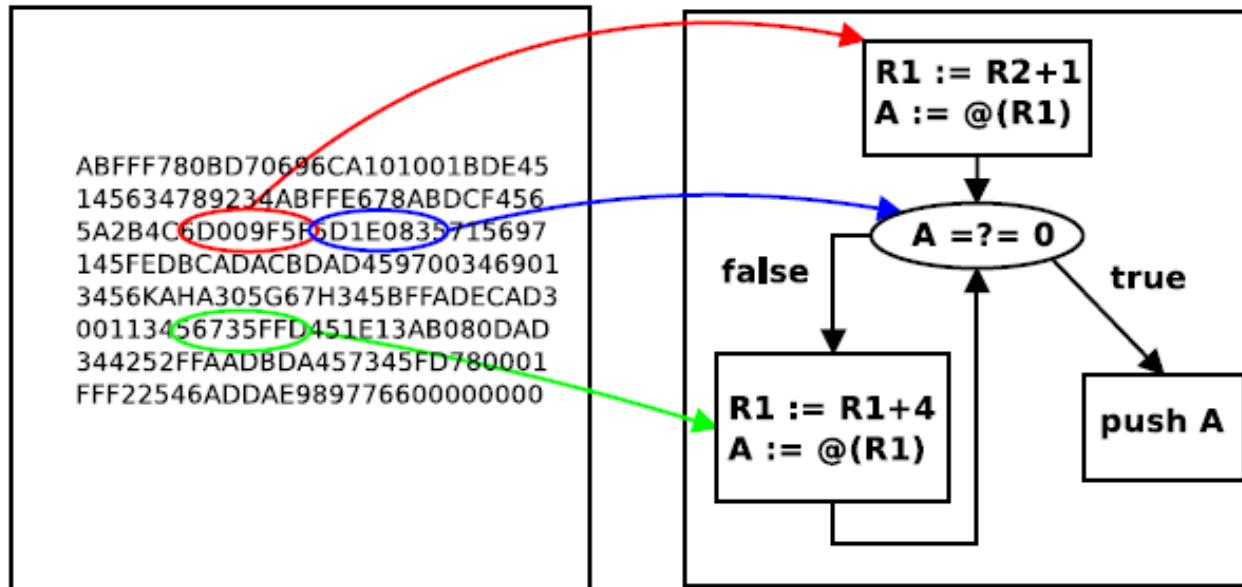
CHALLENGE: CORRECT DISASSEMBLY



Basic reverse problem

- aka model recovery
- aka CFG recovery

CHALLENGE: CORRECT DISASSEMBLY



Basic reverse problem

- aka model recovery
- aka CFG recovery

- **Correct: find only legit instrs.**
 - **Complete: find all legit instrs.**
- (question: how to define legit instr?)**

CAN BE TRICKY!

- code – data
- dynamic jumps (jmp eax)

Sections

.text

```
8D 4C 24 04 83 E4 F0 FF 71 FC 55 89 E5 53 51 83
EC 10 89 CB 83 EC 0C 6A 0A E8 A7 FE FF FF 83 C4
10 89 45 F0 8B 43 04 83 C0 04 8B 00 83 EC 0C 50
E8 C0 FE FF FF 83 C4 10 89 45 F4 83 7D F4 04 77
3B 8B 45 F4 C1 E0 02 05 98 85 04 08 8B 00 FF E0
C7 45 F4 00 00 00 00 EB 23 C7 45 F4 01 00 00 00
EB 1A C7 45 F4 02 00 00 00 EB 11 C7 45 F4 03 00
00 00 EB 08 C7 45 F4 04 00 00 00 90 83 EC 08 FF
75 F4 68 90 85 04 08 E8 29 FE FF FF 83 C4 10 8B
45 F4 8D 65 F8 59 5B 5D 8D 61 FC C3 66 90 66 90
66 90 66 90 90 55 57 31 FF 56 53 E8 85 FE FF FF
81 C3 89 12 00 00 83 EC 1C 8B 6C 24 30 8D B3 0C
FF FF FF E8 B1 FD FF FF 8D 83 08 FF FF FF 29 C6
C1 FE 02 85 F6 74 27 8D B6 00 00 00 00 8B 44 24
38 89 2C 24 89 44 24 08 8B 44 24 34 89 44 24 04
FF 94 BB 08 FF FF 83 C7 01 39 F7 75 DF 83 C4
1C 5B 5E 5F 5D C3 EB 0D 90 90 90 90 90 90 90 90 90
90 90 90 90 90 F3 C3 FF FF 53 83 EC 08 E8 13 FE
FF FF 81 C3 17 12 00 00 83 C4 08 5B C3 03 00 00
00 01 00 02 00 76 61 6C 3A 25 64 0A 00 AB 84 04
08 B4 84 04 08 BD 84 04 08 C6 84 04 08 CF 84 04
08 01 1B 03 3B 28 00 00 00 04 00 00 00 54 FD FF
```

.fini
.rodata

.eh_frame_hdr

■ code ■ dead bytes ■ global csts ■ strings ■ pointers ■ other

Code
(Functions)

main

unknown

_libc_csu_init

unknown

_libc_csu_fini

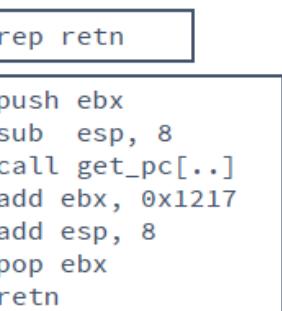
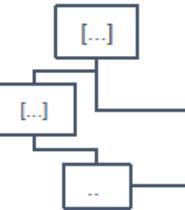
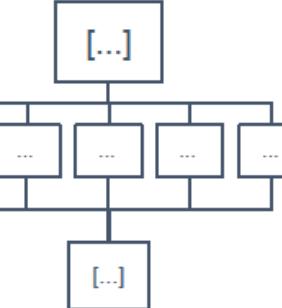
_term_proc

_fp_hw, _IO_stdin_used

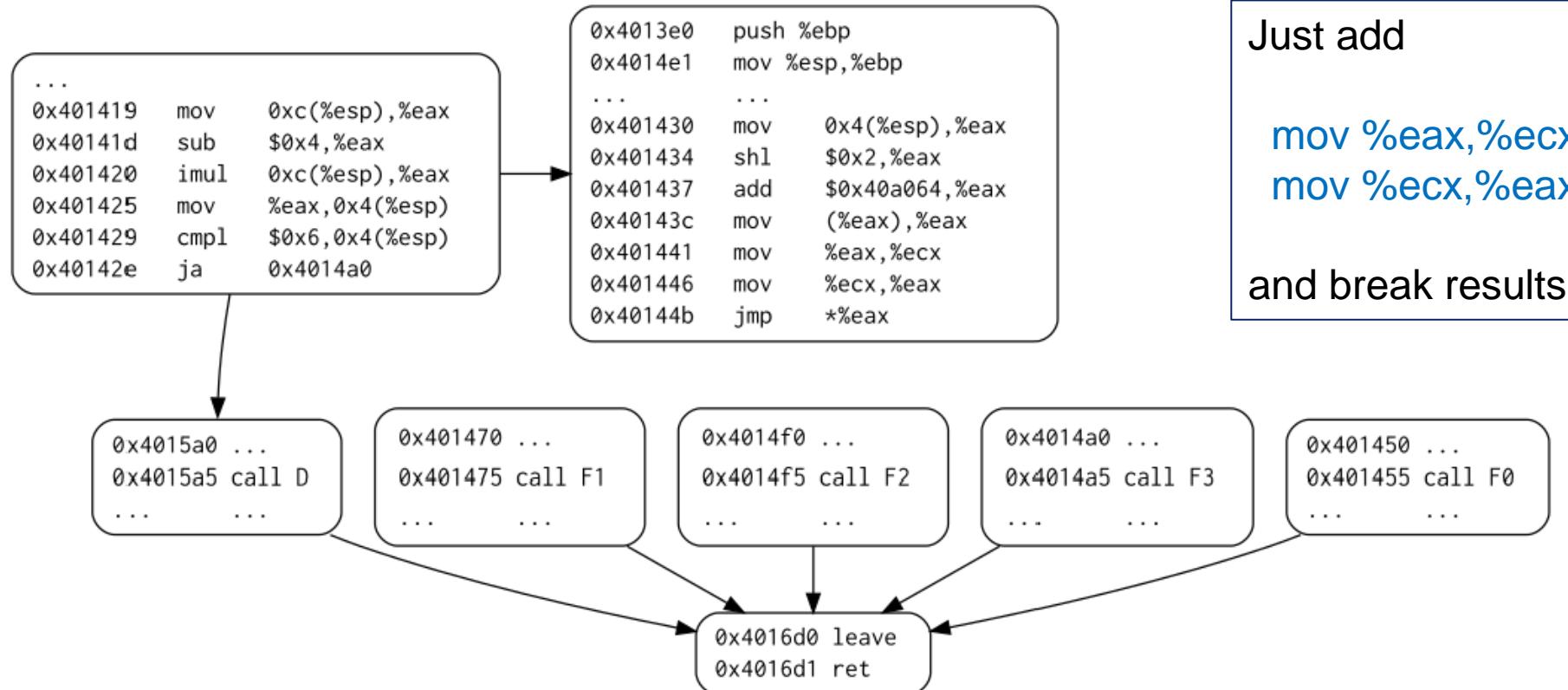
"val%d\n"

switch jump table

Assembly



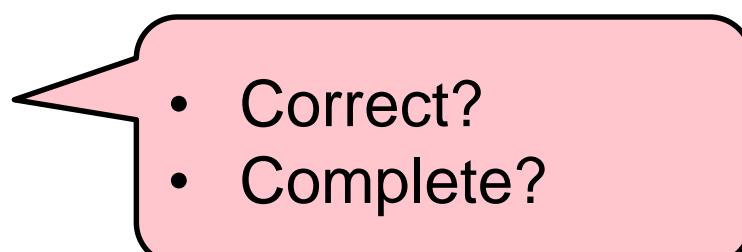
STATE-OF-THE-ART TOOLS ARE NOT ENOUGH



With IDA

- **Static (syntactic): too fragile**
- **Dynamic: too incomplete**

- Input : a binary code (map<addr -> byte>) and initial address addr_0
- Output : map <addr -> instr>
- Also function decode : (code, addr) -> (instr, size)
- Instr ::= operation | halt | jump k | ite(b,k,k') | jump x
- Write:
 - *Linear sweep disassembly*
 - *Recursive disassembly*
 - *Combination*



- Correct?
- Complete?

```
TODO := {addr_0}          // set of addresses
Instr := {}                // set of pairs (addr,instr)

while TODO <> empty do
    choose addr \in TODO;
    TODO := TODO - addr;      // -,+ : set<val> x val -> set<val>
    (i,size) := decode(addr);
    Instr := Instr + (addr,i)
    if (addr+size+1 < code_limit) TODO := TODO + (addr+size+1) end if
end while

return Instr
```

Recursive traversal

```
TODO := {addr_0}; Treated := {}      // sets of addresses
Instr := {}                          // set of pairs (addr,instr)

while TODO <> empty do
    choose addr \in TODO;
    TODO := TODO - addr; Treated := treated + addr;      // -,+ : set<val> x val -> set<val>
    (i,size) := decode(addr);
    Instr := Instr + (addr,i)
    next :=
        match i with
            halt -> {}
            operation -> {addr+size+1}
            jump a' -> {a'}
            ite(f,a',a") -> {a',a"}
            jump EAX -> ??????
        end match
    TODO := TODO \union (next \diff treated) // \union, \diff : set<val> x set<val> -> set<val>
end while

return Instr
```

Combined disassembly?

Quality? (vs non protected code)

	correct	issues	complete	Issues
Linear sweep	XX	Dead code Junk byte	X	Overlapping, desynchronization
Linear sweep byte	XXX	Dead code Junk byte	yes (beware:edges)	Too many non legit code
Recursive traversal	X	Dead code	XX	Dynamic jumps!! (fine otherwise)
Dynamic	yes		XX	Miss legit instructions

Quality? (vs non protected code)

	correct	issues	complete	Issues
Linear sweep	XX	Dead code Junk byte	X	Overlapping, desynchronization
Linear sweep byte	XXX	Dead code Junk byte	yes (beware:edges)	Too many non legit code
Recursive traversal	X	Dead code	XX	Dynamic jumps!! (fine otherwise)
Dynamic	yes		XX	Miss legit instructions

Still, in practice, IDA pro (= linear sweep + recursive + heuristics) works pretty fine for executables

- coming from C/C++
- compiled with standard toolchains (gcc, clang, visual)
- for standard architectures (x86, ARM)

- **Context: MATE attacks**
- **Basic attacks**
- **Basic defense**
- **Better attacks & better defense**
- **Step back: what matters?**
- **Tool: Tigress**
- **Conclusion**

- **Basic defense**
 - Obfuscation
 - Anti-tampering

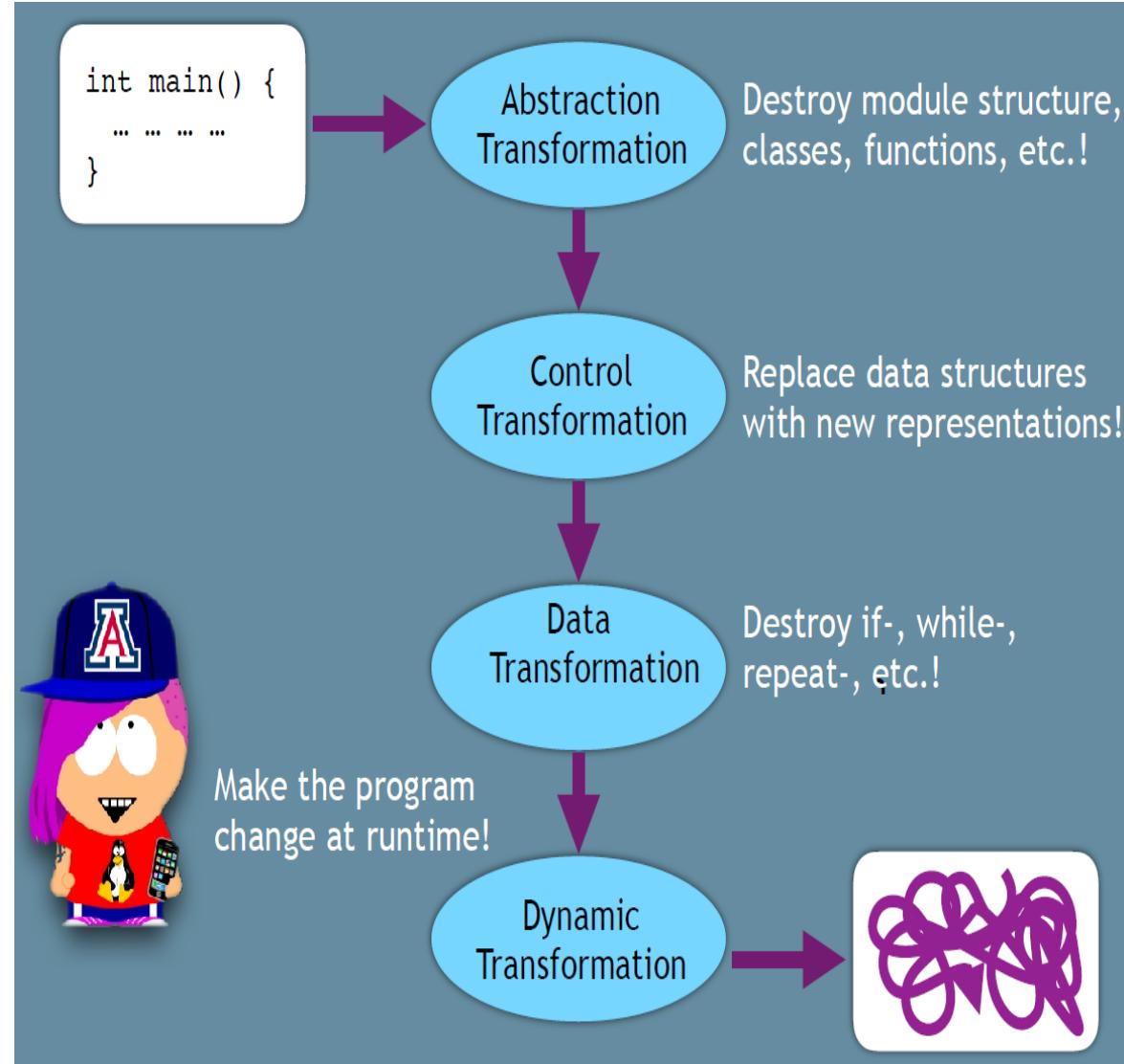
- **Obfuscation**
 - Bloat the code (= add useless stuff)
 - Junk code
 - Opaque predicates + junk code or redundant code
 - Make the code hard to understand
 - Data: Opaque expressions, MBA
 - Control: CFG flattening
 - Both: virtualization
 - Make the code hard to recover
 - Desynchronization
 - Hide the code: self-modification & unpacking
- **Anti-tampering**
 - Redundant check, hash functions
 - Anti-debug, anti-emulation

Example of obfuscation (1)

```
public class C {  
    static int gcd(int x, int y) {  
        int t;  
        while (true) {  
            boolean b = x % y == 0;  
            if (b) return y;  
            t = x % y;  
            x = y;  
            y = t;  
        }  
    }  
}
```

```
public class C {  
    static Object get0(Object[] I) {  
        Integer I7, I6, I4, I3; int t9, t8;  
        I7=new Integer (9);  
        for (;;) {  
            if (((Integer)I[0]).intValue()%  
                ((Integer)I[1]).intValue() == 0)  
                {t9=1; t8=0;}  
            else  
                {t9=0; t8=0;}  
            I4=new Integer(t8);  
            I6=new Integer(t9);  
            if ((I4.intValue ()^I6.intValue ())!=0)  
                return new Integer(((Integer)I[1]).intValue());  
            else {  
                if (((((I7.intValue ()+  
                    I7.intValue ()*I7.intValue ())%2!=0)?0:1)!=1)  
                    return new Integer (0);  
                I3=new Integer(((Integer)I[0]).intValue()%  
                    ((Integer)I[1]).intValue());  
                I[0]=new Integer(((Integer)I[1]).intValue());  
                I[1]=new Integer(I3.intValue());  
            }  
        }  
    }  
}
```

Example of obfuscation (2)



Example of obfuscation (3)

```
int main() {
    int y = 6;
    y = foo(y);
    bar(y,42);
}

int foo(int x)
    return x*7;
}

void bar(int x, int z) {
    if (x==z)
        printf("%i\n",x);
}
```

```
int main() {
    int y = 6;
    y = foobar(y,99,1);
    foobar(y,42,2);
}

int foobar(int x, int z, int s) {
    if (s==1)
        return x*7;
    else if (s==2)
        if (x==z)
            printf("%i \n",x);
}
```

Abstraction
Transformation

Example of obfuscation (4)

```
int main() {  
    int y = 6;  
    y = foobar(y,99,1);  
    foobar(y,42,2);  
}
```

```
int foobar(int x, int z, int s) {  
    if (s==1)  
        return x*7;  
    else if (s==2)  
        if (x==z)  
            printf("%i \n",x);  
}
```

Abstraction Transformation

```
int main () {  
    int y = 12;  
    y = foobar(y,99,1);  
    foobar(y,36,2);  
}  
int foobar(int x, int z, int s) {  
    if (s==1)  
        return (x*37)%51;  
    else if (x==z) {  
        int x2=x*x%51,x3=x2*x%51;  
        int x4=x2*x2%51,x8=x4*x4%51;  
        int x11=x8*x3%51;  
        printf("%i\n",x11);  
    }  
}
```

Data Transformation

Example of obfuscation (5)

```
int main () {
    int y = 12;
    y = foobar(y,99,1);
    foobar(y,36,2);
}
int foobar(int x, int z, int s) {
    if (s==1)
        return (x*37)%51;
    else if (x==z) {
        int x2=x*x%51,x3=x2*x%51;
        int x4=x2*x2%51,x8=x4*x4%51;
        int x11=x8*x3%51;
        printf("%i\n",x11);
    }
}
```

Data Transformation

```
int foobar(int x, int z, int s){
    char* next=&&cell0;
    int retVal = 0;
    cell0: {next=(s==1)?&&cell1:&&cell2;
             goto *next;}
    cell1: {retVal=(x*37)%51; goto end;}
    cell2: {next=(s==2)?&&cell3:&&end;
             goto *next;}
    cell3: {next=(x==z)?&&cell4:&&end;
             goto *next;}
    cell4: {
        int x2=x*x%51,x3=x2*x%51;
        int x4=x2*x2%51,x8=x4*x4%51;
        int x11=x8*x3 % 51;
        printf("%i \n",x11); goto end;
    }
    end: return retVal;
}
```

Control Transformation

Example of tamper-proofing

```
int foo () {  
    if (today > "Aug 17, 2016"){  
        printf("License expired!");  
        abort;  
    }  
}
```

```
check(){  
    if (hash(foo)!=42)  
        abort()  
}
```

```
int hash (addr_t addr,int words){  
    int h = *addr;  
    for(int i=1; i<words; i++) {  
        addr++;  
        h ^= *addr;  
    }  
    return h;  
}
```

```
int foo() {  
    ...  
}
```

Detect
tampering

```
int main () {  
    if (hash(foo,1000) != 0x4C49F346)
```

- ◆ *crash the program*
- ◆ *phone home*
- ◆ *refuse to run*
- ◆ *run slower*
- ◆ *make wrong results*

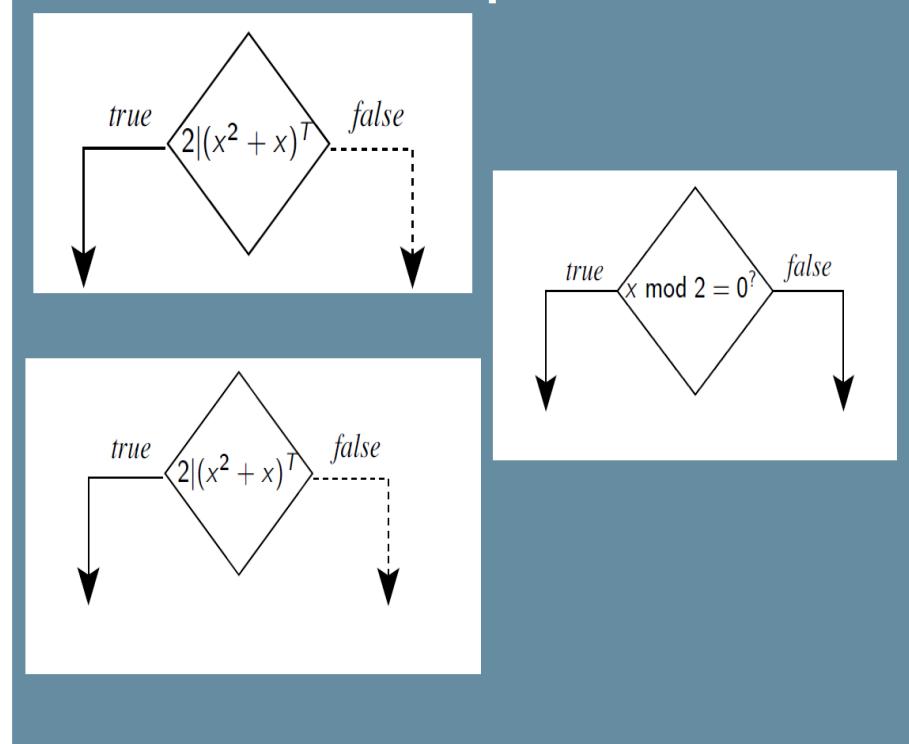
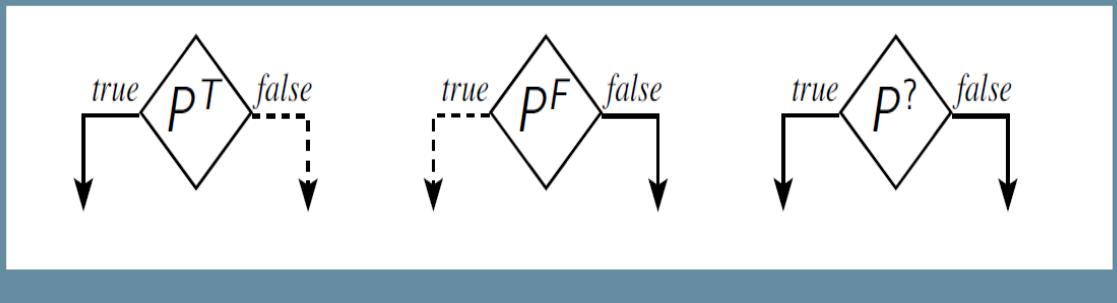
```
foo();  
}
```

Respond
to tampering

- **Tamper-proof and obfuscation**
 - Link?
 - Use both or not?

EXAMPLE: OPAQUE EXPRESSION

- $P=T$ for an opaquely true predicate
- $P=F$ for an opaquely false predicate
- $P=?$ for an opaquely indeterminate predicate
- $E=v$ for an opaque expression of value v



EXAMPLE: OPAQUE EXPRESSION (2)

```
if (x[k] == 1)
    R = (s*y) % n
else
    R = s;
s = R*R % n;
L = R;
```



```
if (x[k] == E=1)
    R = (s*y) % n
else
    R = s;
s = R*R % n;
L = R;
```

```
if (x[k] == 1)
    R = (s*y) % n
else
    R = s;
s = R*R % n;
L = R;
```



```
if (x[k] == 1)
    R = (s*y) % n
else
    R = s;
if (expr=T)
    s = R*R % n;
else
    s = R*R * n;
L = R;
```

EXAMPLE: OPAQUE EXPRESSION (3)

```
int modexp(int y,int x[],int w,int n){  
    int R, L;  
    int k=0; int s=0;  
    while (k < w) {  
        if (x[k] == 1)  
            R = (s*y) % n  
        else  
            R = s;  
        s = R*R % n;  
        L = R;  
        k++;  
    }  
    return L;  
}
```

```
int modexp(int y, int x[], int w, int n) {  
    int R, L, k, s;  
    int next=0;  
    for(;;)  
        switch(next) {  
            case 0 :  
                k=0; s=1; next=1; break;  
            case 1 :  
                if (k<w) next=2; else next=6; break;  
            case 2 :  
                if (x[k]==1) next=3; else next=4; break;  
            case 3 :  
                R=(s*y)%n; next=5; break;  
            case 4 :  
                R=s; next=5; break;  
            case 5 :  
                s=R*R%n; L=R; k++; next=1; break;  
            case 6 : return L;  
        }  
    }
```

EXAMPLE: OPAQUE PREDICATE

Constant-value predicates

(always true, always false)

- dead branch points to **spurious code**
- goal = waste reverser time & efforts

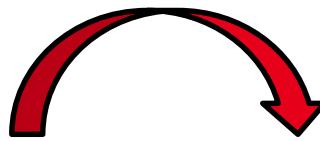
$$\text{eg: } 7y^2 - 1 \neq x^2$$

(for any value of x, y in modular arithmetic)



```
mov eax, ds:X
mov ecx, ds:Y
imul ecx, ecx
imul ecx, 7
sub ecx, 1
imul eax, eax
cmp ecx, eax
jz <dead_addr>
```

EXAMPLE: arithmetic encoding



$$x+y = x - \neg y - 1$$

$$x+y = (x \oplus y) + 2 \cdot (x \wedge y)$$

$$x+y = (x \vee y) + (x \wedge y)$$

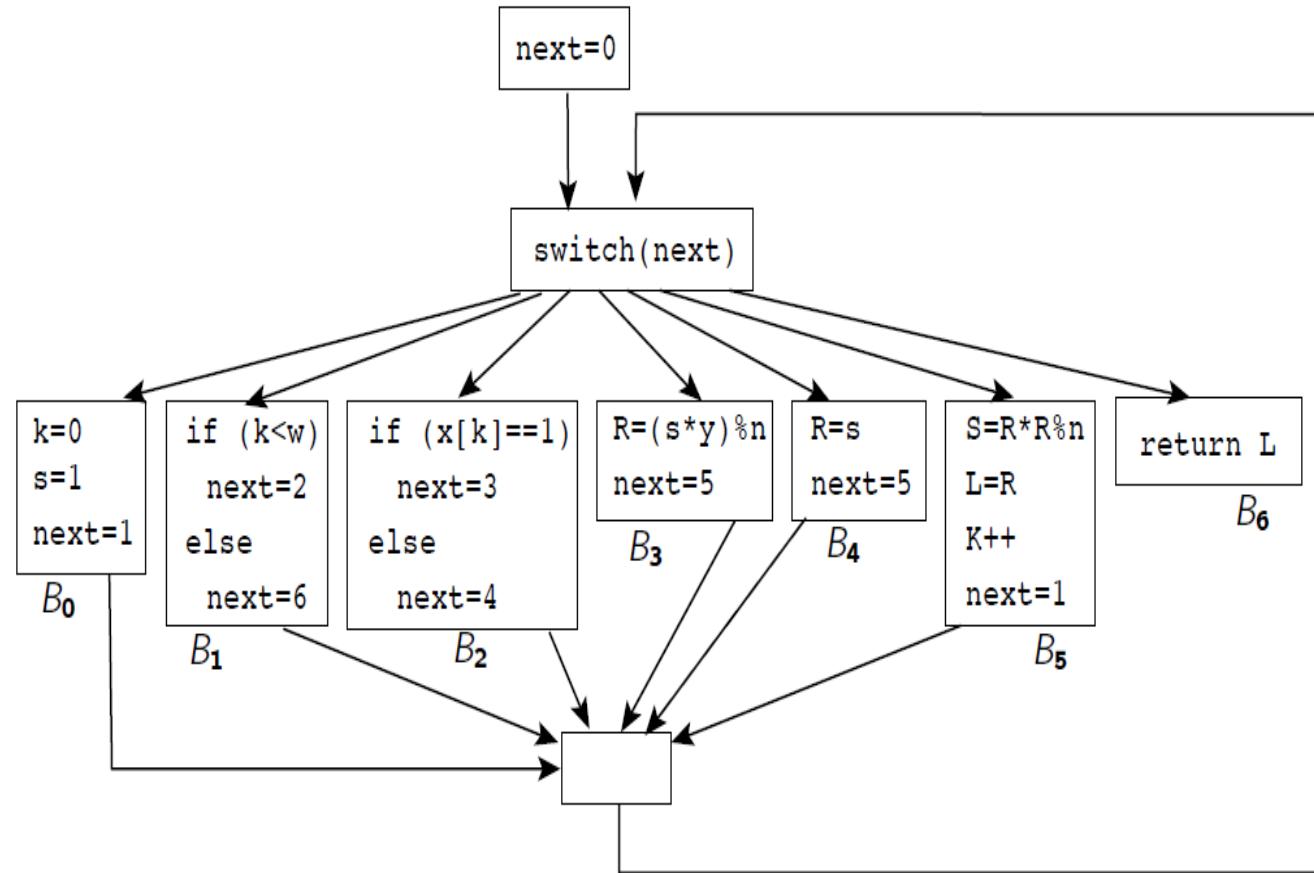
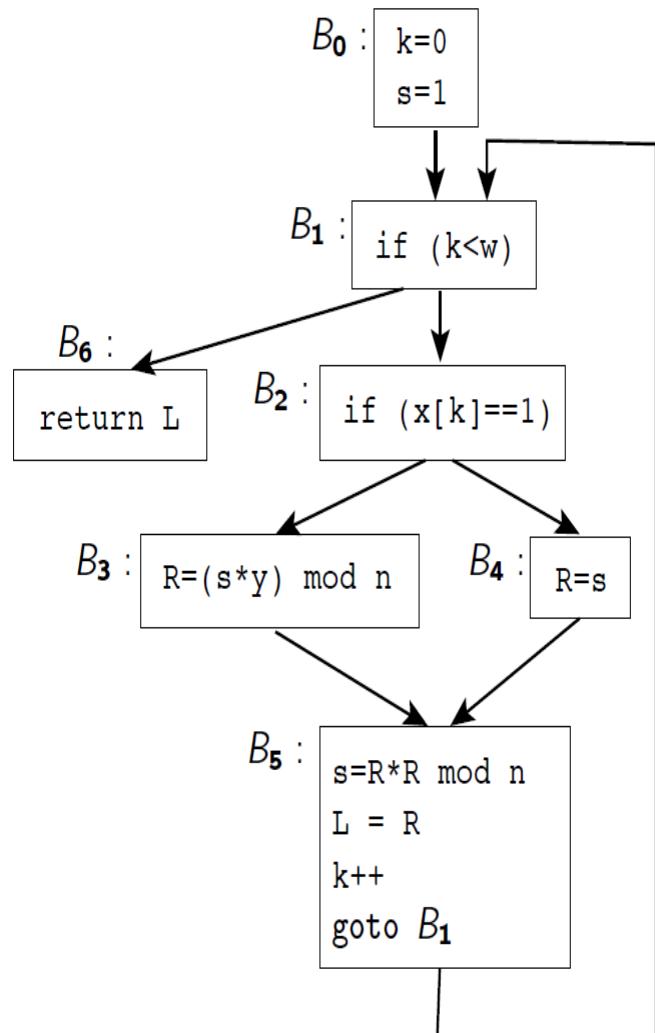
$$x+y = 2 \cdot (x \vee y) - (x \oplus y)$$

$$z = x + y + w$$

is

$$z = (((x \wedge y) + ((x \& y) \ll 1)) \mid w) + (((x \wedge y) + ((x \& y) \ll 1)) \& w);$$

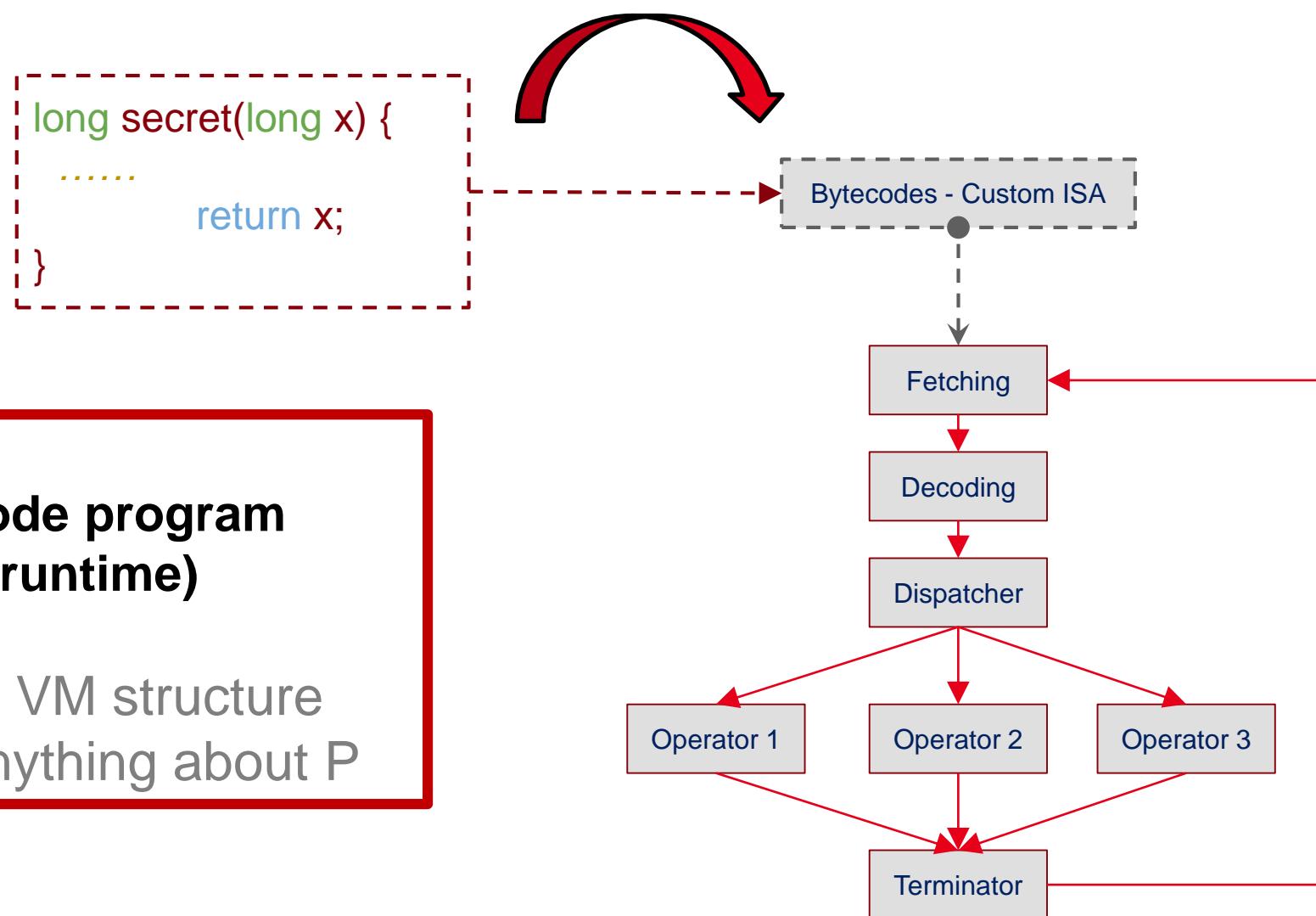
EXAMPLE: Control-flow flattening



EXAMPLE: COMBINE WITH OPAQUE EXPRESSIONS

```
int modexp(int y, int x[], int w, int n) {  
    int R, L, k, s;  
    int next= $E=0$ ;  
    for(;;)  
        switch(next) {  
            case 0: k=0; s=1; next= $E=1$ ; break;  
            case 1: if (k<w) next= $E=2$ ;  
                      else next= $E=6$ ; break;  
            case 2: if (x[k]==1) next= $E=3$ ;  
                      else next= $E=4$ ; break;  
            case 3: R=(s*y)%n; next= $E=5$ ; break;  
            case 4: R=s; next= $E=5$ ; break;  
            case 5: s=R*R%n; L=R; k++;  
                      next= $E=1$ ; break;  
            case 6: return L;  
        } }
```

EXAMPLE: VIRTUALIZATION



DISASSEMBLY ATTACKS: make the code hard to recover

- Desynchronization attacks: kill the disassembler → worst-case code, break heuristics
 - Instruction overlapping
 - stack tampering (cf after)
 - code hidden into data
 - dynamic jumps everywhere + opaque expressions
 - non standard compilation pattern (need to control the compiler, or include crafted asm)
 - ...

EXAMPLE: STACK TAMPERING

**Alter the standard compilation scheme:
ret do not go back to call**

- hide the real target
- return site is **spurious code**

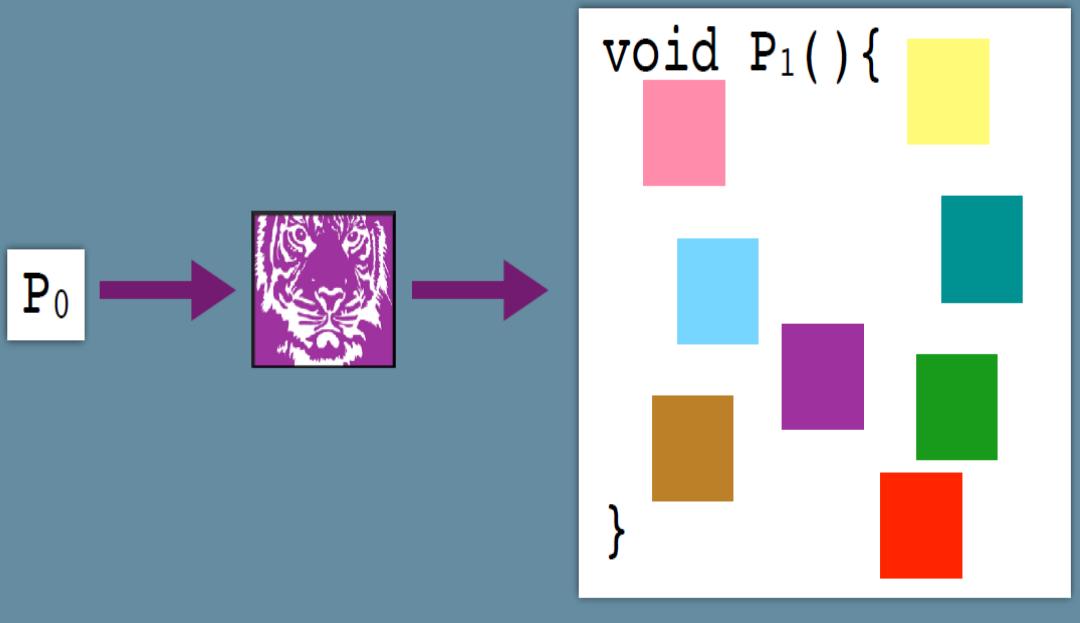
address	instr
80483d1	call +5
80483d6	pop edx
80483d7	add edx, 8
80483da	push edx
80483db	ret
80483dc	.byte{invalid}
80483de	[...]

```
graph TD; A[call +5] --> B[ret]; B --> C[.byte{invalid}]; C --> D[...]
```

EXAMPLE: dynamic obfuscation (hide the code)

- Keep the code in constant flux at runtime
- At no point should the entire code exist in cleartext

- **unpacking**
- **Self-modification**



- **Context: MATE attacks**
- **Basic attacks**
- **Basic defense**
- **Better attacks & better defense**
- **Step back: what matters?**
- **Tool: Tigress**
- **Conclusion**

- **Attacks (both static or dynamic)**
 - Combine static & dynamic!
 - Tainting → what is user-dependent! (remove all non user dependent protections)
 - Slicing → what affects the output (remove junk)
 - Code simplification → remove unduly complex code (duplicate, etc.)
- **Defense: attacks the attack (prog. analysis indecidable : always flaws)**
 - Diffuse dependencies (fake relations: memory, branches, etc.)
 - Hide dependencies (through physical relationship)

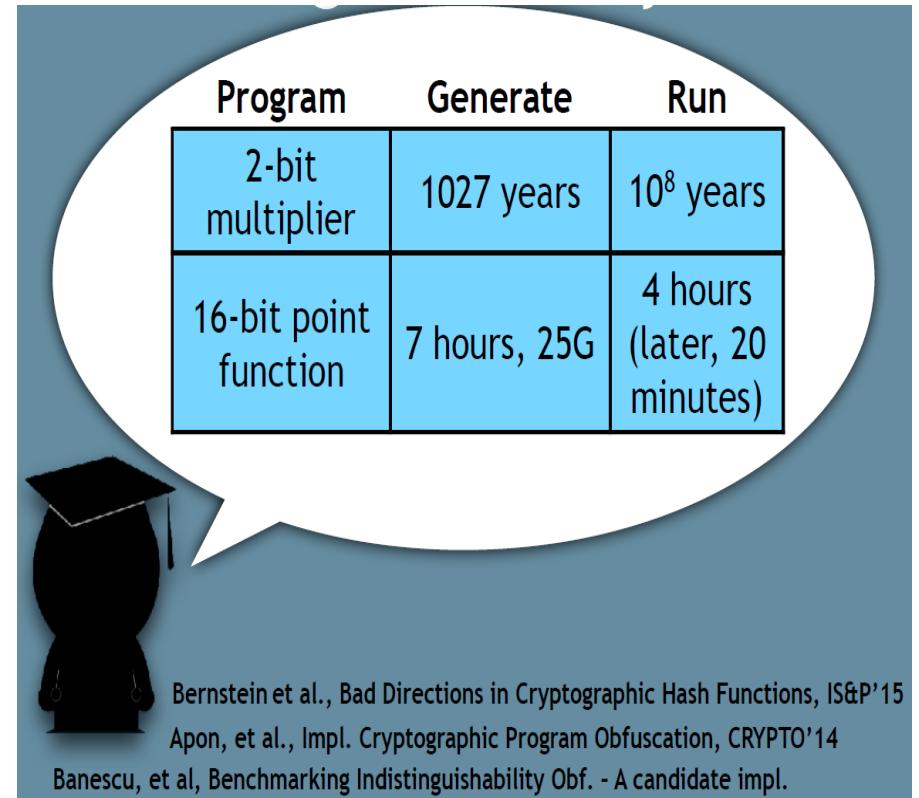
- No protection → static disassembly
- Dynamic protection → dynamic analysis
- User/env-dependent dynamic protection → semantic analysis
- ...

- Context: MATE attacks
- Basic attacks
- Basic defense
- Better attacks & better defense
- Step back: what matters?
- Tool: Tigress
- Conclusion

Properties of obfuscation? (theoretic)

Barak formalization of obfuscation

- P' behaves like P
- P' at most polynomial slowdown
- Blackbox obfuscation: cannot get more information from P' than through a BB access to P
- **Impossible to get in general**
- **More recent : indistinguishability obfuscation**
 - Two equivalent programs P and P'
 - Game = you get $O(P)$ **or** $O(P')$. Try to guess which one it is
 - IO: (polynomial) attacker cannot do better than 50% guess
 - **IO is POSSIBLE !!!**
 - **QUESTIONS**
 - Is it the good notion? Strong enough? (ex: white-box crypto)
 - Current overhead huge in practice



What matters?

Performance



Time-to-Crack



Stealth



Perfs? Depend on context



A speech bubble contains a table comparing performance metrics across different program contexts:

Metric	Program	Slowdown
absolute time	application	<1s
relative	application	1.5x
relative	security kernel	100x-1000x

Below the table, four software names are listed: Code virtualizer, ExeCryptor, VMProtect, Themida. To the right of these names is a horizontal bar with four segments labeled 100x, 700x, 500x, and 1200x.

Liem, Gu, Johnson: A compiler-based infrastructure for software-protection, PLAS'08

Perfs for math-proven obfuscation. Not yet ...

Program	Generate	Run
2-bit multiplier	1027 years	10^8 years
16-bit point function	7 hours, 25G	4 hours (later, 20 minutes)



Bernstein et al., Bad Directions in Cryptographic Hash Functions, IS&P'15

Apon, et al., Impl. Cryptographic Program Obfuscation, CRYPTO'14

Banescu, et al, Benchmarking Indistinguishability Obf. - A candidate impl.

Time to crack!

Program	Adversary	Time
hw+sw		many years
well protected	highly skilled, motivated	4-6 weeks
~VMProtect	experienced reverse engineer	~12 months
mass market malware		minutes- hours



Context matters

How long for cracking?

- **5 min (VOD)**
- **2 weeks (video game)**
- **1 year**
- **No limit**

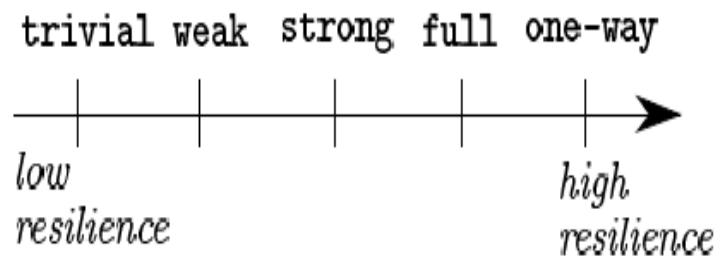
How much overhead is affordable?

Properties of obfuscation? (pragmatic)

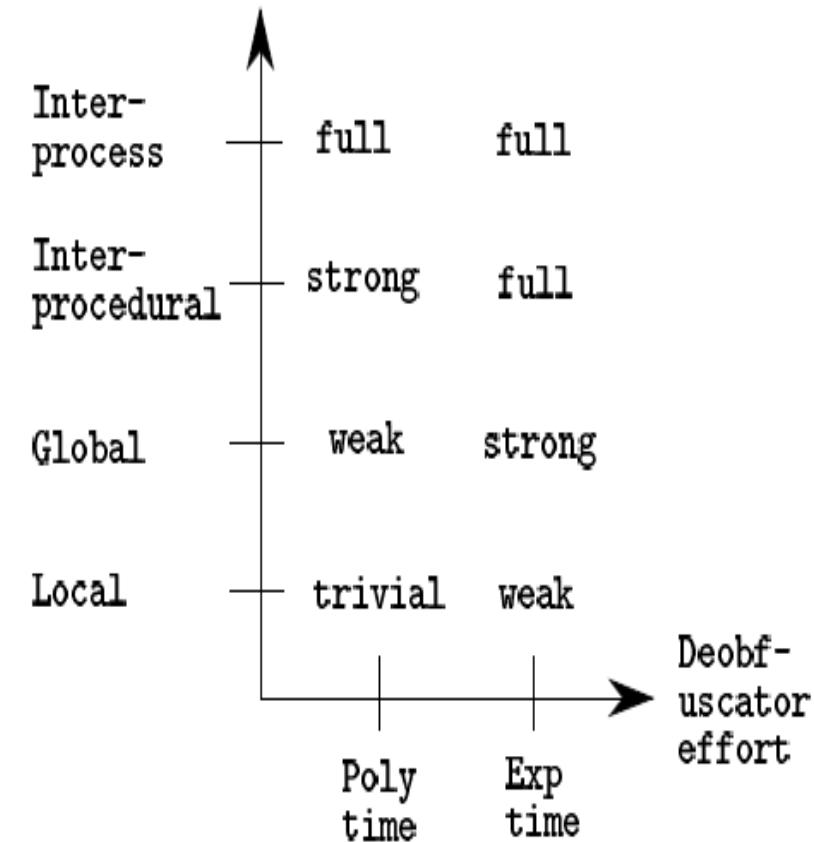
- Four criteria (Collberg et al)
 - **Potency:** confusion, complexity, manual effort
 - **Resilience:** resistance against (automated) tools
 - **Cost:** performance, code size
 - **Stealth:** identification of (components of) protections

Properties of obfuscation? (pragmatic) (2)

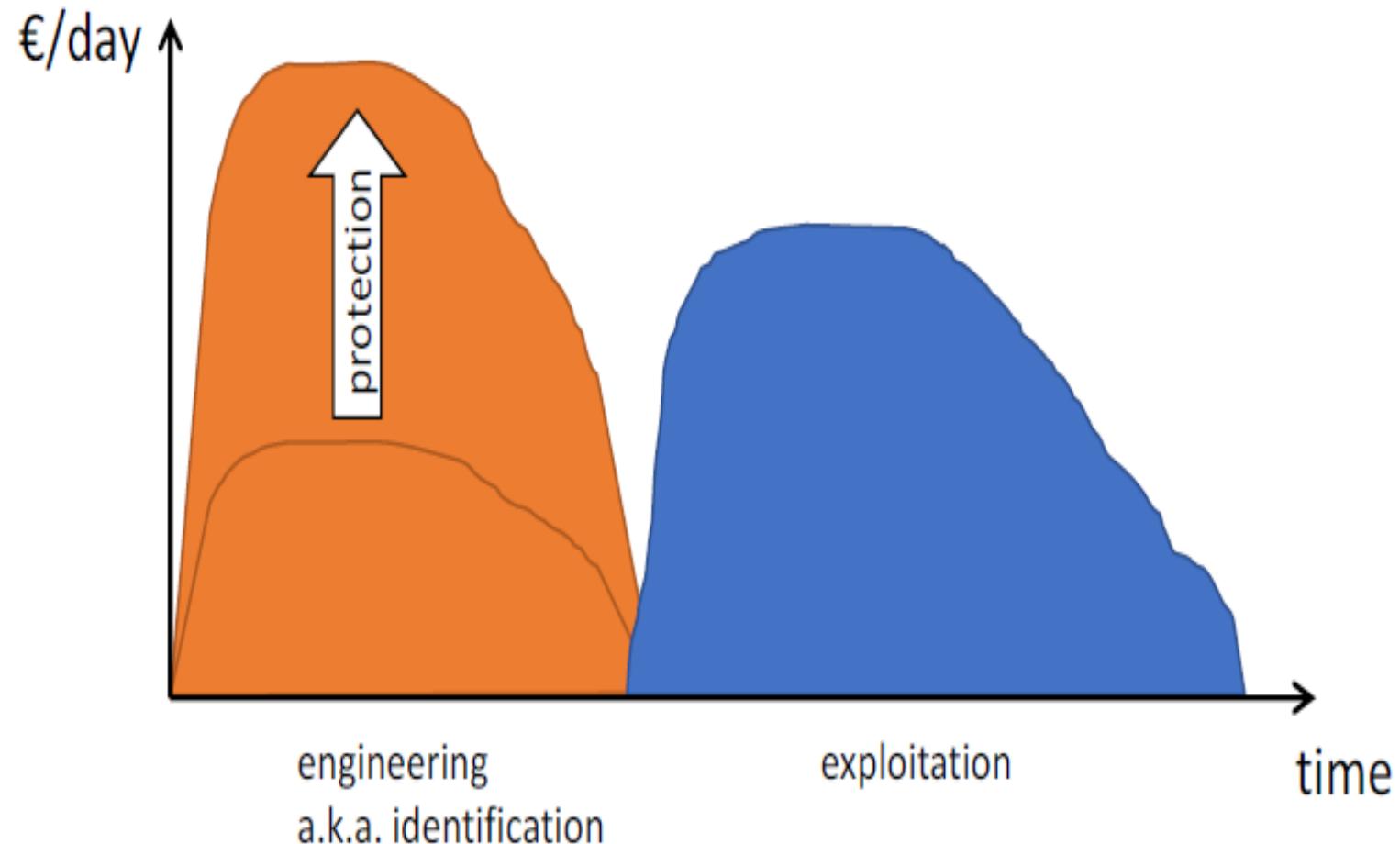
(a)



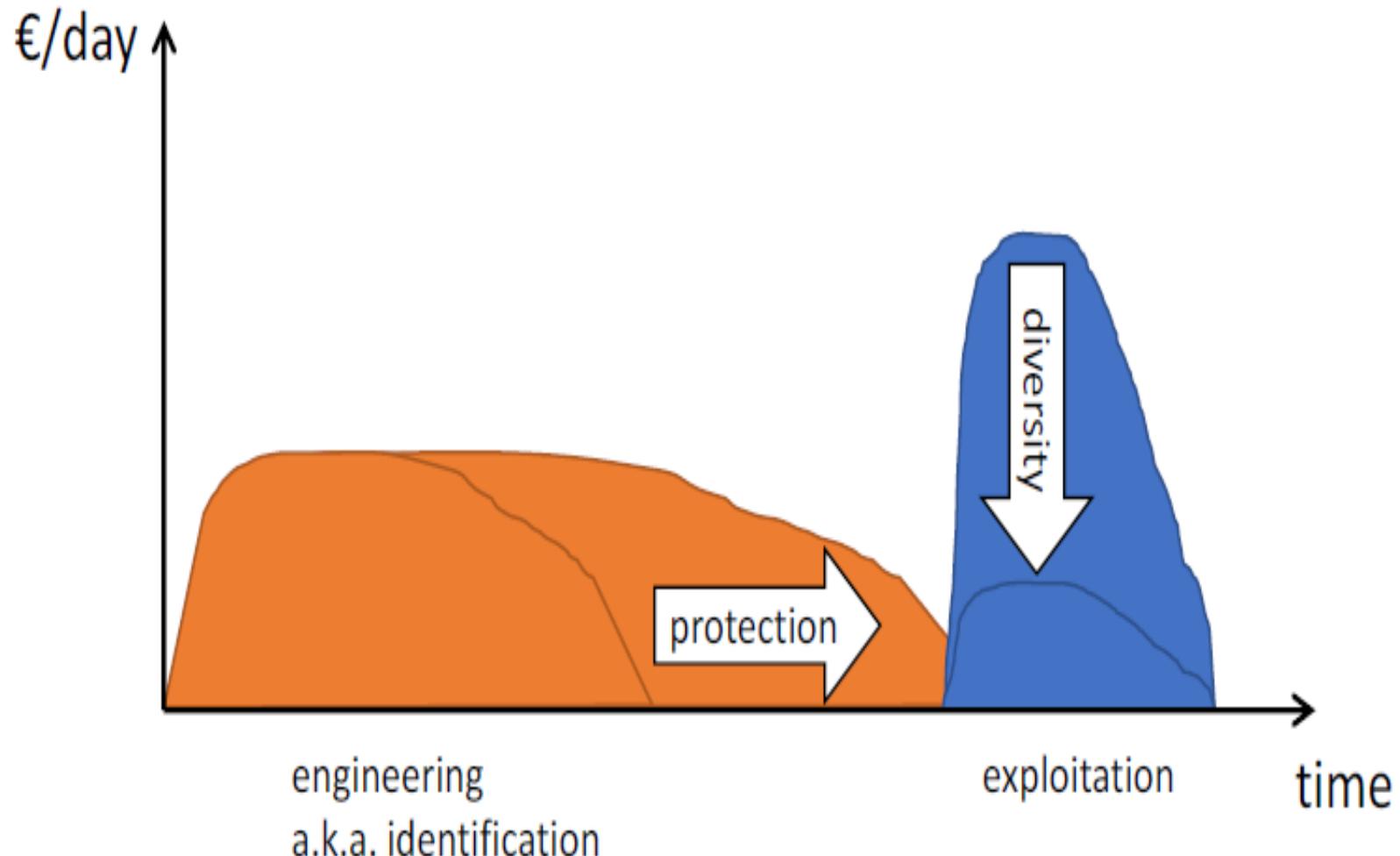
(b) Programmer effort



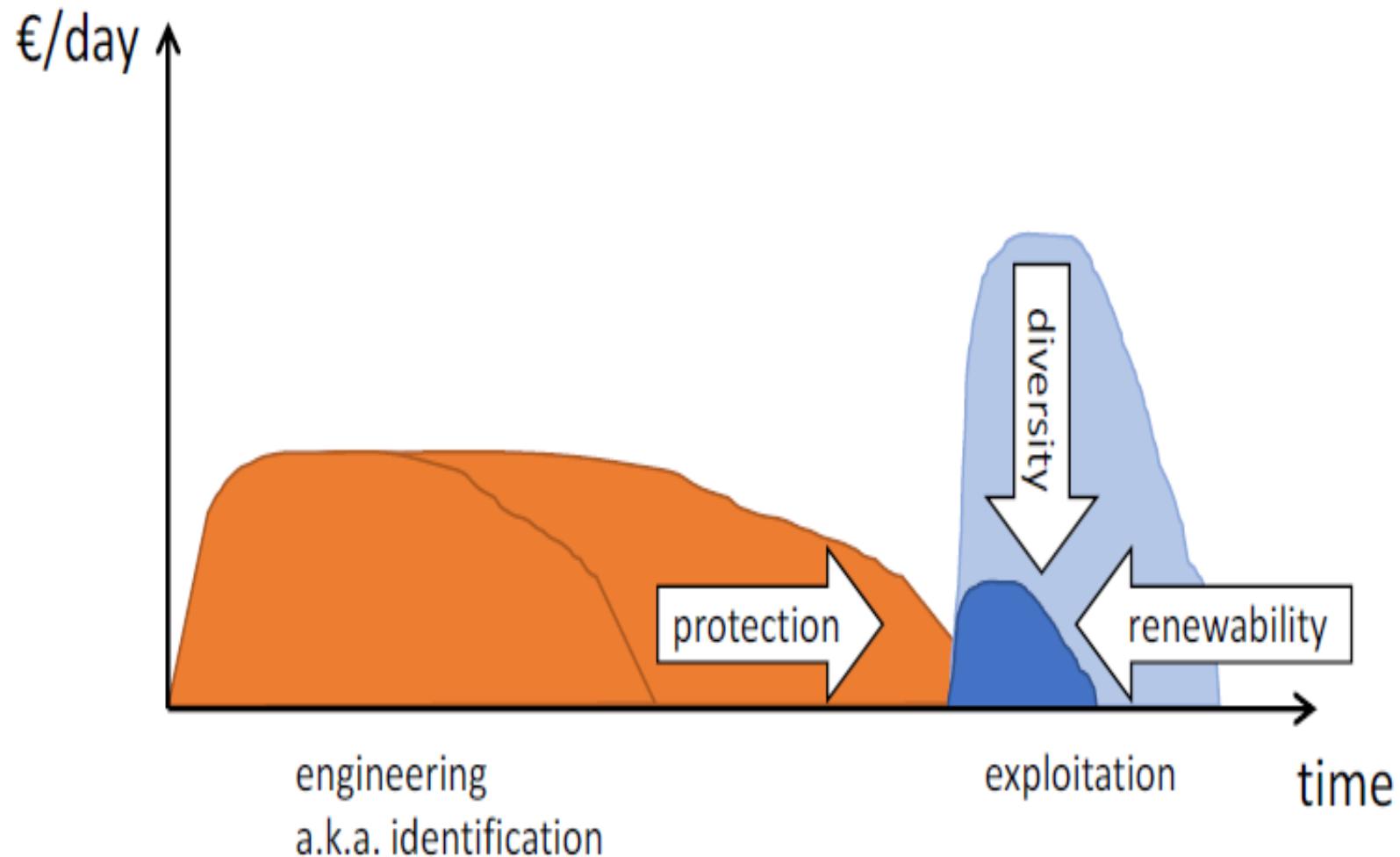
Properties of obfuscation? (pragmatic) (3)



Properties of obfuscation? (pragmatic) (3)



Properties of obfuscation? (pragmatic) (3)



- **Context: MATE attacks**
- **Basic attacks**
- **Basic defense**
- **Better attacks & better defense**
- **Step back: what matters?**
- **Tool: Tigress**
- **Conclusion**

TIGRESS TOOL

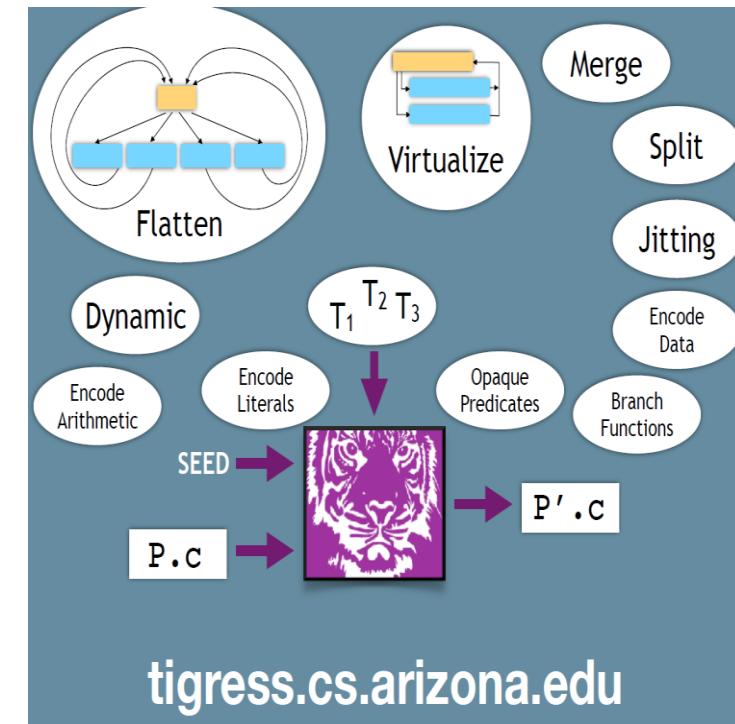
- Hands-on during the lectures
- Install the Tigress obfuscator:

<http://tigress.cs.arizona.edu/#download>

- Get the test program:

<http://tigress.cs.arizona.edu/fib.c>

- Tigress runs on Linux and MacOS



TEST PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
int fib(int n) {
    int a = 1; int b = 1; int i;
    for (i = 3; i <= n; i++) {
        int c = a + b; a = b; b = c;
    };
    return b;
}
int main(int argc, char** argv) {
    if (argc != 2) {
        printf("Give one argument!\n"); abort(); };
    long n = strtol(argv[1],NULL,10);
    int f = fib(n);
    printf("fib(%li)=%li\n",n,f);
}
```

EXO: Opaque expressions

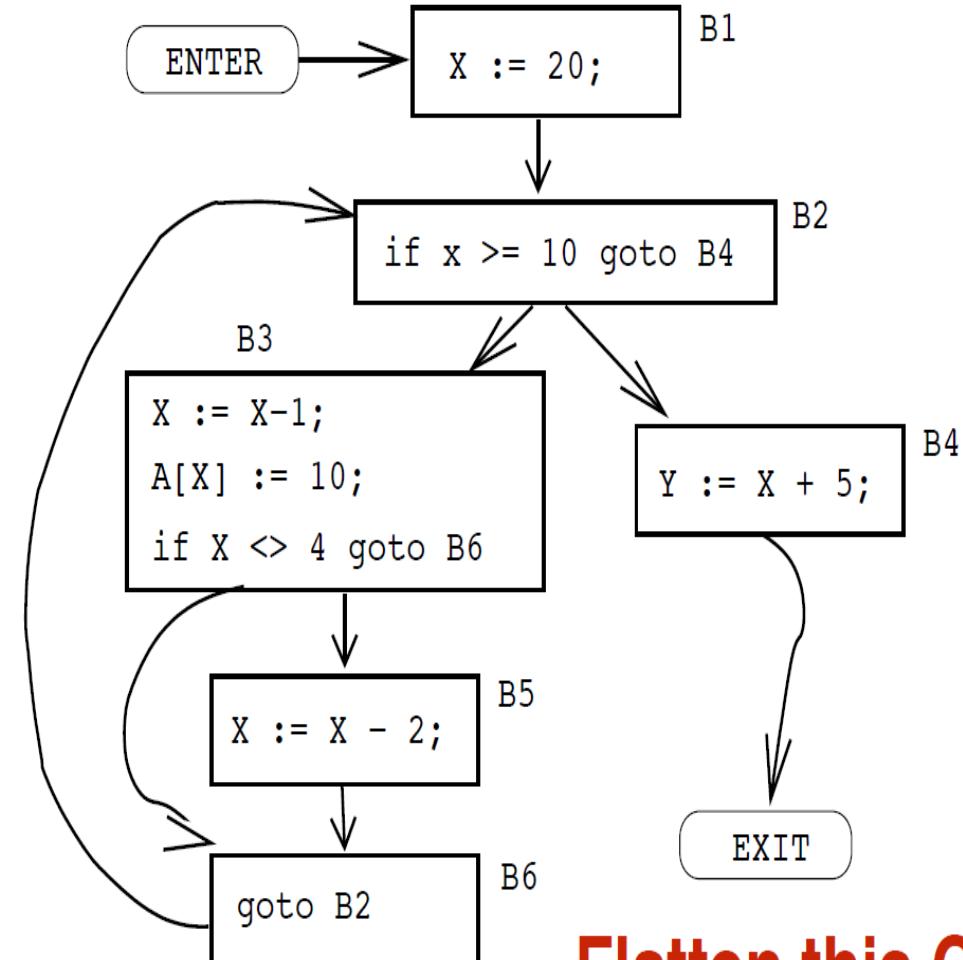
```
tigress --Seed=0 \
  --Transform=InitEntropy \
  --Transform=InitOpaque \
    --Functions=main\
    --InitOpaqueCount=2\
    --InitOpaqueStructs=list,array \
--Transform=AddOpaque\
  --Functions=fib\
  --AddOpaqueKinds=question \
  --AddOpaqueCount=10 \
fib.c -out=fib_out.c
```

EXO: flattening

```
tigress \
  --Seed=42 \
  --Transform=InitOpaque \
  --Functions=main \
  --Transform=Flatten \
  --FlattenDispatch=switch \
  --FlattenOpaqueStructs=array \
  --FlattenObfuscateNext=false \
  --FlattenSplitBasicBlocks=false \
  --Functions=fib \
fib.c --out=fib1.c
```

- Try different kinds of dispatch **switch, goto, indirect**
- Turn opaque predicates on and off.
- Split basic blocks or not.

EXO: flattening (2)



**Flatten this CFG!
Work with your friends!**

EXO: virtualization

```
tigress\  
    --Transform=Virtualize\  
        --Functions=fib\  
        --VirtualizeDispatch=switch\  
    -out=v1.c fib.c
```

- Try a few different dispatchers: `direct`, `indirect`, `call`, `ifnest`, `linear`, `binary`, `interpolation`.
- Are some of them better obfuscators than others? Why?

EXO: virtualization (2)

```
tigress\  
    --Transform=Virtualize  
        --Functions=fib \  
        --VirtualizeDispatch=switch\  
    --Transform=Virtualize\  
        --Functions=fib \  
        --VirtualizeDispatch=indirect \  
    --out=v2.c fib.c
```

- Try combining different dispatchers. Does it make a difference?
- Try three levels of interpretation! Do you notice a slowdown? What about the size of the program?

EXO: arithmetic encoding

- The virtualizer's add instruction handler could still be identified by the fact that it uses a + operator!
- Try adding an arithmetic transformer:

```
--Transform=EncodeArithmetic \
--Functions=fib,main ...
```
- What differences do you notice?
- Should this transformation go before or after the virtualization transformation?

EXO: dynamic encoding

```
tigress \
    --Transform=Dynamic \
    --Functions=fib \
        --DynamicCodecs=xtea \
        --DynamicDumpCFG=false \
        --DynamicBlockFraction=%50 \
        --out=fib_out.c fib.c
```

- Context: MATE attacks
- Basic attacks
- Basic defense
- Better attacks & better defense
- Step back: what matters?
- Tool: Tigress
- Conclusion

- **Code protection is crucial**
 - IP protection, avoid bypassing security
 - Can be argued to improve security as well (anti-hacking technique)
- **Existing tools and techniques**
- **Yet still many open questions**
 - Proper formalization (goal of attacker, probabilistic setting?, stealth?)
 - Distinguish between legit and illegit contexts?
 - Strength, correctness
- **Next: very powerful deobfuscation technique**

Commissariat à l'énergie atomique et aux énergies alternatives
Institut List | CEA SACLAY NANO-INNOV | BAT. 861 – PC142
91191 Gif-sur-Yvette Cedex - FRANCE
www-list.cea.fr

Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019