Outline

1. The Adversary
2. A Cracking Example!

Who's our adversary?

- What does a typical program look like?

Who's our adversary?

- What does a typical program look like?
- What valuables does the program contain?

Who's our adversary?

- What does a typical program look like?
- What valuables does the program contain?
- What is the adversary's motivation for attacking your program?

Who's our adversary?

- What does a typical program look like?
- What valuables does the program contain?
- What is the adversary's motivation for attacking your program?
- What information does he start out with as he attacks your program?

Who's our adversary?

- What does a typical program look like?
- What valuables does the program contain?
- What is the adversary's motivation for attacking your program?
- What information does he start out with as he attacks your program?
- What is his overall strategy for reaching his goals?

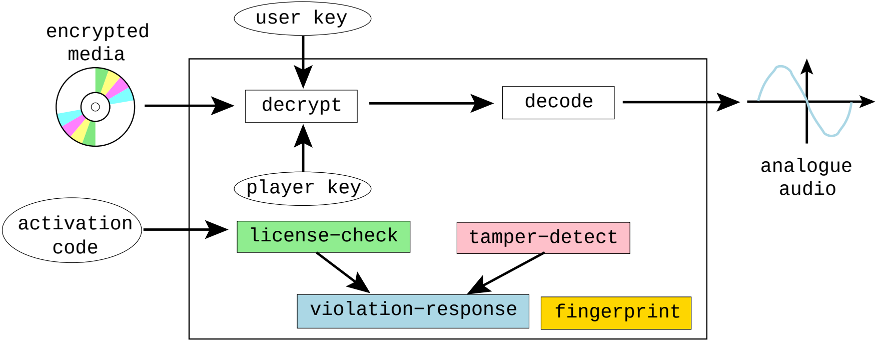
Who's our adversary?

- What does a typical program look like?
- What valuables does the program contain?
- What is the adversary's motivation for attacking your program?
- What information does he start out with as he attacks your program?
- What is his overall strategy for reaching his goals?
- What tools does he have to his disposal?

Who's our adversary?

- What does a typical program look like?
- What valuables does the program contain?
- What is the adversary's motivation for attacking your program?
- What information does he start out with as he attacks your program?
- What is his overall strategy for reaching his goals?
- What tools does he have to his disposal?
- What specific techniques does he use to attack the program?

Example Program



Example Program

```c
typedef unsigned int uint;
typedef uint* waddr_t;
uint player_key = 0xbabeca75;
uint the_key;
uint* key = &the_key;
FILE* audio;
int activation_code = 42;

void FIRST_FUN(){}
uint hash (waddr_t addr, waddr_t last) {
    uint h = *addr;
    for(;addr<=last;addr++) h^=*addr;
    return h;
}
void die(char* msg) {
    fprintf(stderr,"%s!\n",msg);
    key = NULL;
}
```

Example Program

```
uint play(uint user_key,
          uint encrypted_media[],
          int media_len) {
    int code;
    printf("Please enter activation code: ");
    scanf("%i",&code);
    if (code!=activation_code) die("wrong code");

    *key = user_key ^ player_key;
```

Example Program

```
    int i;
    for(i=0;i<media_len;i++) {
        uint decrypted = *key ^ encrypted_media[i];
        asm volatile (
            "jmp L1              \n\t"
            ".align 4            \n\t"
            ".long     0xb0b5b0b5\n\t"
            "L1:                 \n\t"
        );
        if (time(0) > 1221011472) die("expired");
        float decoded = (float)decrypted;
        fprintf(audio,"%f\n",decoded); fflush(audio);
    }
}
```

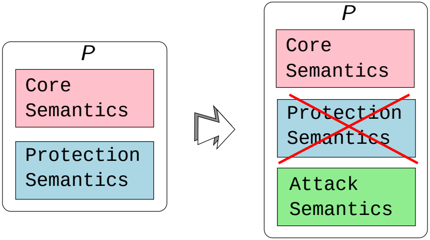
Example Program

```
void LAST_FUN(){}
uint player_main (uint argc, char *argv[]) {
    uint user_key = ···
    uint encrypted_media[100] = ···
    uint media_len = ···
    uint hashVal = hash((waddr_t)FIRST_FUN,
                        (waddr_t)LAST_FUN);
    if (hashVal != HASH) die("tampered");
    play(user_key, encrypted_media,media_len);
}
```

What's the Adversary's Motivation?

The adversary's wants to

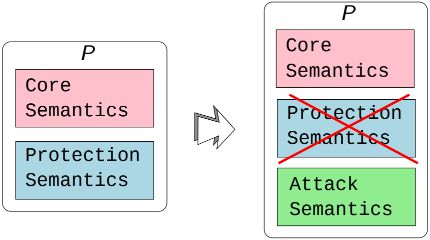
- remove the protection semantics.



What's the Adversary's Motivation?

The adversary's wants to

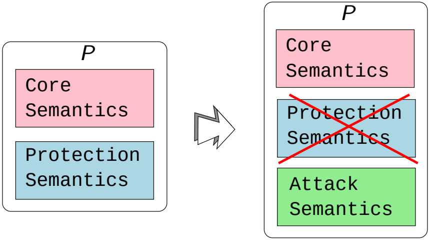
- remove the protection semantics.
- add his own attack semantics (ability to save game-state, print,. . .)



What's the Adversary's Motivation?

The adversary's wants to

- remove the protection semantics.
- add his own attack semantics (ability to save game-state, print,. . .)
- ensure that the core semantics remains unchanged.



What does he want to do to our Player program?

- get decrypted digital media

What does he want to do to our Player program?

- get decrypted digital media
- extract the `player_key`

What does he want to do to our Player program?

- get decrypted digital media
- extract the `player_key`
- use the program after the expiration date
  - remove use-before check
  - remove activation code

What does he want to do to our Player program?

- get decrypted digital media
- extract the `player_key`
- use the program after the expiration date
  - remove use-before check
  - remove activation code
- distribute the program to other users
  - remove fingerprint `0xb0b5b0b5`

What does he want to do to our Player program?

- get decrypted digital media
- extract the `player_key`
- use the program after the expiration date
  - remove use-before check
  - remove activation code
- distribute the program to other users
  - remove fingerprint `0xb0b5b0b5`
- reverse engineer the algorithms in the player

What are the methods of attack?

1. the *black box* phase
   - feed the program inputs,
   - record its outputs,
   - draw conclusions about its behavior.

What are the methods of attack?

1. the *black box* phase
   - feed the program inputs,
   - record its outputs,
   - draw conclusions about its behavior.
2. the *dynamic analysis* phase
   - execute the program
   - record which parts get executed for different inputs.

What are the methods of attack?

1. the *black box* phase
   - feed the program inputs,
   - record its outputs,
   - draw conclusions about its behavior.
2. the *dynamic analysis* phase
   - execute the program
   - record which parts get executed for different inputs.
3. the *static analysis* phase
   - examining the executable code directly
   - use disassembler, decompiler, . . .

What are the methods of attack?

4. the *editing* phase
   - use understanding of the internals of the program
   - modify the executable
   - disable license checks

What are the methods of attack?

4. the *editing* phase
   - use understanding of the internals of the program
   - modify the executable
   - disable license checks
5. the *automation* phase.
   - encapsulates his knowledge of the attack in an automated *script*
   - use in future attacks.

Outline

1. The Adversary
2. A Cracking Example!

Let's crack!

- Let's get a feel for the types of techniques attackers typically use.
- Our example cracking target will be the DRM player.
- Our chief cracking tool will be the `gdb` debugger.

Step 1: Learn about the executable file

```
> file player
player: ELF 64-bit LSB executable, dynamically linked

> objdump -T player
DYNAMIC SYMBOL TABLE:
0xa4    scanf
0x90    fprintf
0x12    time

> objdump -x player | egrep 'rodata|text|Name'
Name           Size     VMA        LMA        File off
.text          0x4f8    0x4006a0   0x4006a0   0x6a0
.rodata        0x84     0x400ba8   0x400ba8   0xba8

> objdump -f player | grep start
start address 0x4006a0
```

Step 2: Breaking on library functions

- Treat the program as a black box
- Feed it inputs to see how it behaves.

```
> player 0xca7ca115 1 2 3 4
Please enter activation code: 42
expired!
Segmentation fault
```

- Find the assembly code equivalent of

```
if (time(0) > some value)···
```

- Replace it with

```
if (time(0) <= some value)···
```

Example Program

```
    int i;
    for(i=0;i<media_len;i++) {
        uint decrypted = *key ^ encrypted_media[i];
        asm volatile (
            "jmp L1                \n\t"
            ".align 4              \n\t"
            ".long      0xb0b5b0b5\n\t"
            "L1:                   \n\t"
        );
        if (time(0) > 1221011472) die("expired");
        float decoded = (float)decrypted;
        fprintf(audio,"%f\n",decoded); fflush(audio);
    }
}
```

Step 2: Breaking on library functions

At 0x4008bc is the offending conditional branch:

```
> gdb -write -silent --args player 0xca7ca115 \
                     1000 2000 3000 4000
(gdb) break time
Breakpoint 1 at 0x400680
(gdb) run
Please enter activation code: 42
Breakpoint 1, 0x400680 in time()
(gdb) where 2
#0  0x400680 in time
#1  0x4008b6 in ??
(gdb) up
#1  0x4008b6 in ??
(gdb) disassemble $pc-5 $pc+7
0x4008b1    callq  0x400680
0x4008b6    cmp    $0x48c72810,%rax
0x4008bc    jle    0x4008c8
```

Step 2: Breaking on library functions

Patch the executable:

- replace the `jle` with a `jg` (x86 opcode 0x7f)

```
(gdb) set {unsigned char}0x4008bc = 0x7f
(gdb) disassemble 0x4008bc 0x4008be
0x4008bc    jg     0x4008c8
```

Step 3: Static pattern-matching

- search the executable for character strings.

```
> player 0xca7ca115 1000 2000 3000 4000
tampered!
Please enter activation code: 99
wrong code!
Segmentation fault
```

Example Program

```c
uint play(uint user_key,
          uint encrypted_media[],
          int media_len) {
    int code;
    printf("Please enter activation code: ");
    scanf("%i",&code);
    if (code!=activation_code) die("wrong code");

    *key = user_key ^ player_key;
```

Step 3: Static pattern-matching

- the code that checks the activation code looks something like this:

```
addr1:    .ascii "wrong code"
          ...
          cmp     read_value,activation_code
          je      somewhere
addr2:    move    addr1, reg0
          call    printf
```

Step 3: Static pattern-matching

1. search the data segment to find address `addr1` where `"wrong code"` is allocated.
2. search through the text segment for an instruction that contains that address as a literal:

```
(gdb) find 0x400ba8,+0x84,"wrong code"
0x400be2
(gdb) find 0x4006a0,+0x4f8,0x400be2
0x400862
(gdb) disassemble 0x40085d 0x400867
0x40085d    cmp    %eax,%edx
0x40085f    je     0x40086b
0x400861    mov    $0x400be2,%edi
0x400866    callq  0x4007e0
```

Step 3: Static pattern-matching

- Replace the jump-on-equal with a jump-always

```
(gdb) set {unsigned char}0x40085f = 0xeb
(gdb) disassemble 0x40085f 0x400860
0x40085f     jmp     0x40086b
```

Step 4: Watching memory

- the program still crashes with a segmentation violation
- the edits cause the tamper detection mechanism to kick in!

```
>  player 0xca7ca115 1000 2000 3000 4000
tampered!
Please enter activation code: 55
Segmentation fault
```

Example Program

```c
typedef unsigned int uint;
typedef uint* waddr_t;
uint player_key = 0xbabeca75;
uint the_key;
uint* key = &the_key;
FILE* audio;
int activation_code = 42;

void FIRST_FUN(){}
uint hash (waddr_t addr, waddr_t last) {
    uint h = *addr;
    for(;addr<=last;addr++) h^=*addr;
    return h;
}
void die(char* msg) {
    fprintf(stderr,"%s!\n",msg);
    key = NULL;
}
```

Example Program

```
    int i;
    for(i=0;i<media_len;i++) {
        uint decrypted = *key ^ encrypted_media[i];
        asm volatile (
            "jmp L1              \n\t"
            ".align 4            \n\t"
            ".long     0xb0b5b0b5\n\t"
            "L1:                 \n\t"
        );
        if (time(0) > 1221011472) die("expired");
        float decoded = (float)decrypted;
        fprintf(audio,"%f\n",decoded); fflush(audio);
    }
}
```

Step 4: Watching memory

1. let the program run until it crashes
2. rerun the program while watching the address
3. find the location which sets it to an illegal value

```
(gdb) run
Program received signal SIGSEGV
0x40087b in ?? ()
(gdb) disassemble 0x40086b 0x40087d
0x40086b    mov    0x2009ce(%rip),%rax   # 0x601240
0x400872    mov    0x2009c0(%rip),%edx   # 0x601238
0x400878    xor    -0x14(%rbp),%edx
0x40087b    mov    %edx,(%rax)
```

Step 4: Watching memory

1. set a watchpoint
2. rerun the program from the beginning

```
(gdb) watch *0x601240
(gdb) run
tampered!
Hardware watchpoint 2: *0x601240

Old value = 6296176
New value = 0

0x400811 in ?? ()

(gdb) disassemble 0x400806 0x400812
0x400806    movq    $0x0,0x200a2f(%rip)   # 0x601240
0x400811    leaveq
```

Step 4: Watching memory

- the instruction at 0x400806 is setting the word at address 0x601240 to 0!
- This corresponds to

```
void die(char* msg) {
    fprintf(stderr,”%s!\n”,msg);
    key = NULL;
}
```

Step 4: Watching memory

- overwrite with a sequence of `nop` instructions (x86 opcode 0x90):

```
(gdb) set {unsigned char}0x400806 = 0x90
        ....
(gdb) set {unsigned char}0x400810 = 0x90

(gdb) disassemble 0x400806 0x400812
0x400806    nop
...
0x400810    nop
0x400811    leaveq
```

Step 5: Recovering internal data

1. ask the debugger to print out decrypted media data!

```
(gdb) hbreak *0x4008a6
(gdb) commands
>x/x -0x8+$rbp
>continue
>end
(gdb) cont
Please enter activation code: 42
Breakpoint 2, 0x4008a6
0x7fffffffdc88: 0xbabec99d
Breakpoint 2, 0x4008a6
0x7fffffffdc88: 0xbabecda5
          ...
```

Step 6: Tampering with the environment

1. To avoid triggering the timeout, wind back the system clock!
2. Change the library search path to force the program to pick up hacked libraries!
3. Hack the OS (we'll see this later).

Step 7: Dynamic pattern-matching

- Pattern-match not on static code and data but on its dynamic behavior.
- What encryption algorithm is this?

```
0x0804860b       cmpl   $0x0,0xfffffff0(%ebp)
0x0804860f       jg     0x8048589

0x08048589       mov    0x8(%ebp),%edx
0x08048592       shl    $0x2,%eax
0x080485a0       shl    $0x2,%eax
0x080485ab       shl    $0x2,%eax
0x080485ba       shr    $0x5,%edx
0x080485c0       shl    $0x2,%eax
0x080485c5       xor    %eax,%ecx
......
```

Step 8: Differential attacks

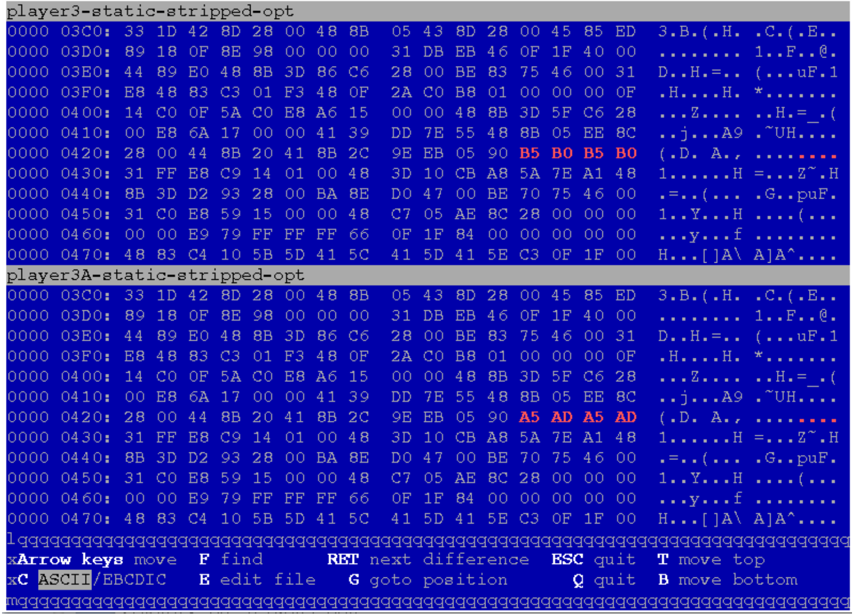
1. Find two differently fingerprinted copies of the program
2. Diff them!

```
asm volatile (
   "jmp L1               \n\t"
   ".align 4             \n\t"
   ".long     0xb0b5b0b5\n\t"
   "L1:                  \n\t"
);
```

```
asm volatile (
   "jmp L1               \n\t"
   ".align 4             \n\t"
   ".long     0xada5ada5\n\t"
   "L1:                  \n\t"
);
```

Example Program

```
    int i;
    for(i=0;i<media_len;i++) {
        uint decrypted = *key ^ encrypted_media[i];
        asm volatile (
            "jmp L1              \n\t"
            ".align 4            \n\t"
            ".long     0xb0b5b0b5\n\t"
            "L1:                 \n\t"
        );
        if (time(0) > 1221011472) die("expired");
        float decoded = (float)decrypted;
        fprintf(audio,"%f\n",decoded); fflush(audio);
    }
}
```



```
player3-static-stripped-opt
0000 03C0: 33 1D 42 8D 28 00 48 8B  05 43 8D 28 00 45 85 ED  3.B.(.H. .C.(.E..
0000 03D0: 89 18 0F 8E 98 00 00 00  31 DB EB 46 0F 1F 40 00  ........ 1..F..@.
0000 03E0: 44 89 E0 48 8B 3D 86 C6  28 00 BE 83 75 46 00 31  D..H.=.. (...uF.1
0000 03F0: E8 48 83 C3 01 F3 48 0F  2A C0 B8 01 00 00 00 0F  .H....H. *.......
0000 0400: 14 C0 0F 5A C0 E8 A6 15  00 00 48 8B 3D 5F C6 28  ...Z.... ..H.=_.(
0000 0410: 00 E8 6A 17 00 00 41 39  DD 7E 55 48 8B 05 EE 8C  ..j...A9 .~UH....
0000 0420: 28 00 44 8B 20 41 8B 2C  9E EB 05 90 B5 B0 B5 B0  (.D. A., ........
0000 0430: 31 FF E8 C9 14 01 00 48  3D 10 CB A8 5A 7E A1 48  1......H =...Z~.H
0000 0440: 8B 3D D2 93 28 00 BA 8E  D0 47 00 BE 70 75 46 00  .=..(... .G..puF.
0000 0450: 31 C0 E8 59 15 00 00 48  C7 05 AE 8C 28 00 00 00  1..Y...H ....(...
0000 0460: 00 00 E9 79 FF FF FF 66  0F 1F 84 00 00 00 00 00  ...y...f ........
0000 0470: 48 83 C4 10 5B 5D 41 5C  41 5D 41 5E C3 0F 1F 00  H...[]A\ A]A^....
player3A-static-stripped-opt
0000 03C0: 33 1D 42 8D 28 00 48 8B  05 43 8D 28 00 45 85 ED  3.B.(.H. .C.(.E..
0000 03D0: 89 18 0F 8E 98 00 00 00  31 DB EB 46 0F 1F 40 00  ........ 1..F..@.
0000 03E0: 44 89 E0 48 8B 3D 86 C6  28 00 BE 83 75 46 00 31  D..H.=.. (...uF.1
0000 03F0: E8 48 83 C3 01 F3 48 0F  2A C0 B8 01 00 00 00 0F  .H....H. *.......
0000 0400: 14 C0 0F 5A C0 E8 A6 15  00 00 48 8B 3D 5F C6 28  ...Z.... ..H.=_.(
0000 0410: 00 E8 6A 17 00 00 41 39  DD 7E 55 48 8B 05 EE 8C  ..j...A9 .~UH....
0000 0420: 28 00 44 8B 20 41 8B 2C  9E EB 05 90 A5 AD A5 AD  (.D. A., ........
0000 0430: 31 FF E8 C9 14 01 00 48  3D 10 CB A8 5A 7E A1 48  1......H =...Z~.H
0000 0440: 8B 3D D2 93 28 00 BA 8E  D0 47 00 BE 70 75 46 00  .=..(... .G..puF.
0000 0450: 31 C0 E8 59 15 00 00 48  C7 05 AE 8C 28 00 00 00  1..Y...H ....(...
0000 0460: 00 00 E9 79 FF FF FF 66  0F 1F 84 00 00 00 00 00  ...y...f ........
0000 0470: 48 83 C4 10 5B 5D 41 5C  41 5D 41 5E C3 0F 1F 00  H...[]A\ A]A^....
lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
xArrow keys move  F find      RET next difference  ESC quit  T move top
xC ASCII/EBCDIC   E edit file   G goto position      Q quit  B move bottom
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
```

Step 9: Decompilation

```
L080482A0(A8, Ac, A10) {
    ebx = A8;
    esp = "Please enter activation code: ";
    eax = L080499C0();
    V4 = ebp - 16;
    *esp = 0x80a0831;
    eax = L080499F0();
    eax = *(ebp - 16);
    if(eax != *L080BE2CC) {
        V8 = "wrong code";
        V4 = 0x80a082c;
        *esp = *L080BE704;
        eax = L08049990();
        *L080BE2C8 = 0;
    }
```

Example Program

```
uint play(uint user_key,
          uint encrypted_media[],
          int media_len) {
   int code;
   printf(”Please enter activation code: ”);
   scanf("%i",&code);
   if (code!=activation_code) die(”wrong code”);

   *key = user_key ^ player_key;
```

```
          eax = *L080BE2C8;
          edi = 0;
          ebx = ebx ^ *L080BE2C4;
          *eax = ebx;
          eax = A10;
          if(eax <= 0) {} else {
              while(1) {
                  esi = *(Ac + edi * 4);
L08048368:        *esp = 0;
                  if(L08056DD0() > 1521011472) {
                      V8 = "expired";
                      V4 = 0x80a082c;
                      *esp = *L080BE704;
                      L08049990();
                      *L080BE2C8 = 0;
                  }
```

Example Program

```c
typedef unsigned int uint;
typedef uint* waddr_t;
uint player_key = 0xbabeca75;
uint the_key;
uint* key = &the_key;
FILE* audio;
int activation_code = 42;

void FIRST_FUN(){}
uint hash (waddr_t addr, waddr_t last) {
    uint h = *addr;
    for(;addr<=last;addr++) h^=*addr;
    return h;
}
void die(char* msg) {
    fprintf(stderr,"%s!\n",msg);
    key = NULL;
}
```

```
            ebx = ebx ^ esi;
            (save)0;
            edi = edi + 1;
            (save)ebx;
            esp = esp + 8;
            V8 = *esp;
            V4 = "%f\n"; *esp = *L080C02C8;
            eax = L08049990();
            eax = *L080C02C8;
            *esp = eax;
            eax = L08049A20();
            if(edi == A10) {goto L080483a7;}
            eax = *L080BE2C8; ebx = *eax;
        }
        ch = 176; ch = 176;
        goto L08048368;
    }
L080483a7:
}
```

```
L080483AF(A8, Ac) {
    ...
    ecx = 0x8048260;
    edx = 0x8048230;
    eax = *L08048230;
    if(0x8048260 >= 0x8048230) {
        do {
            eax = eax ^ *edx;
            edx = edx + 4;
        } while(ecx >= edx);
    }
    if(eax != 318563869) {
        V8 = ”tampered”;
        V4 = 0x80a082c;
        *esp = *L080BE704;
        L08049990();
        *L080BE2C8 = 0;
    }
    V8 = A8 - 2;
    V4 = ebp + -412;
    *esp = *(ebp + -416);
    return(L080482A0());
}
```

Example Program

```
typedef unsigned int uint;
typedef uint* waddr_t;
uint player_key = 0xbabeca75;
uint the_key;
uint* key = &the_key;
FILE* audio;
int activation_code = 42;

void FIRST_FUN(){}
uint hash (waddr_t addr, waddr_t last) {
    uint h = *addr;
    for(;addr<=last;addr++) h^=*addr;
    return h;
}
void die(char* msg) {
    fprintf(stderr,”%s!\n”,msg);
    key = NULL;
}
```

Discussion

Who is our prototypical cracker? He can

- pattern-match on static code and execution patterns,

Discussion

Who is our prototypical cracker? He can

- pattern-match on static code and execution patterns,
- relate external program behavior to internal code locations,

Discussion

Who is our prototypical cracker? He can

- pattern-match on static code and execution patterns,
- relate external program behavior to internal code locations,
- disassemble and decompile binary machine code,

Discussion

Who is our prototypical cracker? He can

- pattern-match on static code and execution patterns,
- relate external program behavior to internal code locations,
- disassemble and decompile binary machine code,
- debug binary code without access to source code,

Discussion

Who is our prototypical cracker? He can

- pattern-match on static code and execution patterns,
- relate external program behavior to internal code locations,
- disassemble and decompile binary machine code,
- debug binary code without access to source code,
- compare (statically or dynamically) two closely related versions of the same program,

Discussion

Who is our prototypical cracker? He can

- pattern-match on static code and execution patterns,
- relate external program behavior to internal code locations,
- disassemble and decompile binary machine code,
- debug binary code without access to source code,
- compare (statically or dynamically) two closely related versions of the same program,
- modify the executable and its execution environment.