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

```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;
}```

Example Program

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

Example Program

```c
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);
    }
}
```

Example Program

```
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
- 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.
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, ...
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 printf
0x12 time

> objdump -x player | egrep 'rodata|text|Name'
Name   Size  VMA   LMA   File    File
.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",
        ".align 4 \n",
        ".long 0xb0b5b0b5\n",
        ":L1: \n",
    );
    if (time(0) > 1221011472) die("expired");
    float decoded = (float)decrypted;
    printf(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
```

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

1. `typedef unsigned int uint;`  
2. `typedef uint* waddr_t;`  
3. `uint player_key = 0xbabeca75;`  
4. `uint the_key;`  
5. `uint* key = &the_key;`  
6. `FILE* audio;`  
7. `int activation_code = 42;`  
8. `void FIRST_FUN(){}`  
9. `int hash(waddr_t addr, waddr_t last) {`  
10. `    uint h = *addr;`  
11. `    for(;addr<=last;addr++) h ^= *addr;`  
12. `    return h;`  
13. `}`  
14. `void die(char* msg) {`  
15. `    printf(stderr,"%s!\n",msg);`  
16. `    key = NULL;`  
17. `}`

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)
```

Example Program

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

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

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

- 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
  0x400807   nop
  0x400808   nop
  0x400809   nop
  0x40080a   nop
  0x40080b   nop
  0x40080c   nop
  0x40080d   nop
  0x40080e   nop
  0x40080f   nop
  0x400810   nop
  0x400811   leaveq
  ```

Step 5: Recovering internal data

- Ask the debugger to print out decrypted media data!

```
(gdb) hbreak *0x400a6
(gdb) commands
>x/x -0x8+$rbp
>continue
>end
```

Step 6: Tampering with the environment

- To avoid triggering the timeout, wind back the system clock!
- Change the library search path to force the program to pick up hacked libraries!
- 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?

Example Program

```c
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 0x0b05b0b5\n t" ",L1:\ \n t" );
}
```

A Cracking Example! 33/44

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 0xada5ada5\n t" 
    ",L1:\ \n t" 
);
```

A Cracking Example! 34/44

Example Program

```c
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 0x0b05b0b5\n t" ",L1:\ \n t" );
    if (time(0) > 1221011472) die("expired");
    float decoded = (float)decrypted;
    printf(audio,"%f\n",decoded); flush(audio);
}
```

A Cracking Example! 35/44
Step 9: Decompilation

Example Program

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;
    }
}

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

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;
}
Example Program

```c
typedef unsigned int uint;  // 1
typedef uint* waddr_t;      // 2
uint player_key = 0xbabeca75;  // 3
uint the_key;  // 4
uint* key = &the_key;  // 5
FILE* audio;  // 6
int activation_code = 42;  // 7

void FIRST_FUN(){}  // 8
uint hash (waddr_t addr, waddr_t last) {  // 9
    uint h = *addr;
    for (; addr<=last; addr++) h ^= *addr;
    return h;  // 11
}

void die(char* msg) {  // 14
    fprintf(stderr, "%s!\n", msg);
    key = NULL;
}
```

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.