CSc 466/566

Computer Security

13 : Man-At-The-End — Introduction

Version: 2012/03/27 13:35:35

Department of Computer Science
University of Arizona

collberg@gmail.com

Christian Collberg

Outline

Surreptitious Software

- Protect the secrets contained within computer programs.
- Prevent others from exploiting the intellectual effort invested in producing a piece of software.
- For example,
  - software fingerprinting — trace software pirates,
  - code obfuscation — make it more difficult to reverse engineer a program,
  - tamperproofing — make it harder to remove a license check.

Surreptitious Software

- The fundamental questions in this field are
  - is it possible to hide a secret in a piece of software?
  - and, is it possible to protect this secret from modification?
- There's always a time-dimension: how long can we protect the secret?
- There's always an economic dimension: how valuable is the secret (to us, and to the adversary)?
- There's always a performance dimension: how much slower/larger are you willing to let your program grow, to make it harder to crack?

Why?

- We borrow from
  - security research — but we don't protect against malware!
  - cryptography — but we don't assume the key is hidden!
  - software engineering — but we try to make programs hard to understand!
  - steganogrphy — but for programs, not media!
  - compiler research — but we make programs slower and larger!
- Why????
- There are real-world problems that don't fit neatly into traditional computer security and cryptography research, but which are interesting none-the-less.

Techniques

- Software watermarking
  - embed a unique identifier into your program,
  - use to fight piracy,
  - trace beta copies leaked by partner companies.
- Code obfuscation
  - use to protect a novel algorithm.
- Birthmarking
  - identify your code "reused" by competitor.
- Tamperproofing
  - prevent pirate from removing license checks,
  - prevent music pirate from hacking DRM system.

Strengths and weaknesses

- None of these techniques are fool-proof.
- Our secret will eventually be discovered by a sufficiently determined hacker.
- All we can hope is to can slow down our adversaries.
- Our goal is to slow them down *enough*:
  - they give up on cracking our code because it's too painful, or
  - by the time they've cracked our code, we've already made a profit.

Pay TV

- Personalized set-top boxes.
- A cracker hacks open the box, extracts the code, disassembles it, find the unique indentifier, sells it over the web.
- How do you counter his?
  - tamperproof smart cards.
  - obfuscate the code
  - software tamperproofing
  - all of the above!
- Still, eventually, the box will be cracked.
  - Maybe by then, you've made your $1,000,000!
  - Automatically updatable security.

Who uses Surreptitious Software?

- Microsoft owns several software watermarking, obfuscation,and birthmarking patents.
- Intertrust holds many DRM-related patents, including patents on obfuscation and tamperproofing (Microsoft licensed this for $440M.)
- Microsoft uses PreEmptive Solutions' obfuscator in Visual Studio.
- Arxan sells a tamperproofing system invented by Purdue University researchers.
- Apple holds an obfuscation patent, perhaps intended to protect iTunes?
- Intel spun off a company, Convera, to explore their tamperproofing algorithm for DRM.

Who uses Surreptitious Software?

- Cloakware, which holds patents on HLwhitebox cryptography, how to hide cryptographic algorithms and keys in computer code. Sold for $72.5M to pay-TV company Irdeto.
- Skype's VoIP client is highly obfuscated and tamperproofed. Sold to eBay for $2.6 billion in 2005.
- The military wants to protect embedded software!

- Bad guys want to protect malware!

Academic Research

- Some, like me, come from a compiler and programming languages background. We work on practical code transformation algorithms.
- Some come from cryptography, typically work on fundamental issues, such as "what can be obfuscated?"
- Some come from media watermarking, computer security, software engineering,. . .

The military anti-tamper research (AT)

All U.S. Army Project Executive Offices (PEOs) and Project Managers (PMs) are now charged with executing Army and Department of Defense (DoD) AT policies in the design and implementation of their systems. Embedded software is at the core of modern weapon systems and is one of the most critical technologies to be protected. AT provides protection of U.S. technologies against exploitation via reverse engineering. Standard compiled code with no AT is easy to reverse engineer, so the goal of employed AT techniques will be to make that effort more difficult. In attacking software, reverse engineers have a wide array of tools available to them, including debuggers, decompilers, disassemblers, as well as static and dynamic analysis techniques. AT techniques are being developed to combat the loss of the U.S. technological advantage, but further advances are necessary to provide useful, effective and varied toolsets to U.S. Army PEOs and PMs.

And then, there's the. . .

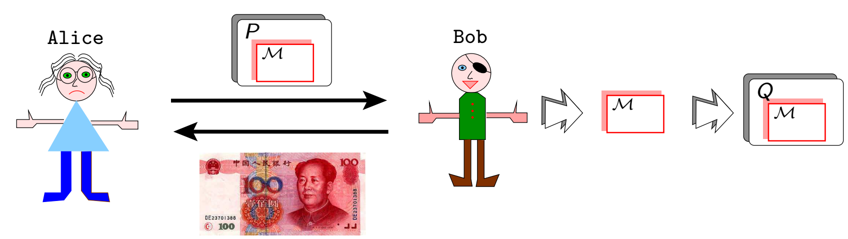
- . . . the bad guys!
- Virus writers use obfuscation to prevent it from being intercepted by virus scanners.
- Hiding root kits requires obfuscation.

What's the goal of this research?

- Invent algorithms
  - slow down our adversaries
  - add little computational overhead
- Devise evaluation techniques
  - algorithm A will force a hacker to use T extra time, adding O amount of overhead, or
  - algorithm A produces code that is faster/smaller/harder to crack than algorithm B.
- Theoretical advances
  - what can be protected?
  - what *cannot* be protected?
- Research still in its infancy!

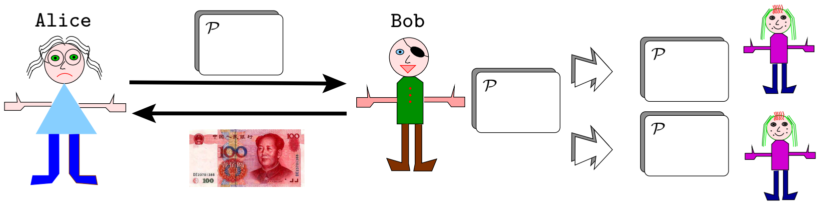
Outline

Scenario: Malicious reverse engineering



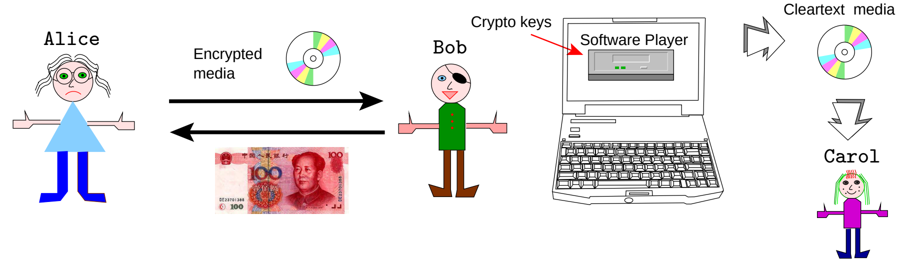
- Alice's program contains a valuable trade secret (a clever algorithm or design).
- Bob, a rival developer, extracts and incorporates module $\mathcal{M}$ into his own program (code lifting).
- Computer games industry: stealing 3$^{\text{rd}}$ party modules for graphics/physics/. . . .

Scenario: Software piracy



- Alice is a software developer.
- Bob buys one copy of Alice's program.
- Bob illegally sells copies to his friends.

Scenario: Digital rights management (DRM)

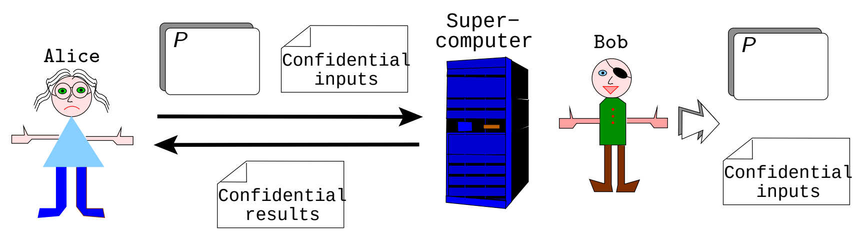


- A DRM media player contains cryptographic keys that unlock and play encrypted music files.

Scenario: Mobile agent attack

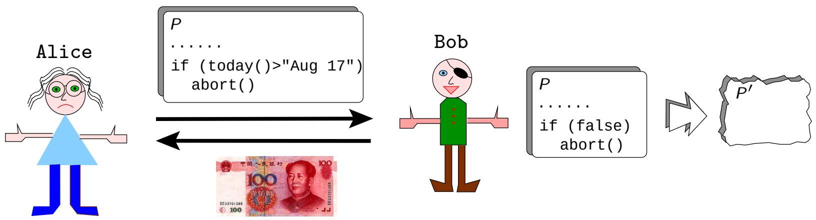
- Alice's mobile shopping agent visits on-line stores to find the best deal for a CD.
- Bob manipulates the agent's code such that it returns his higher price as the best one.

Scenario: Grid computing



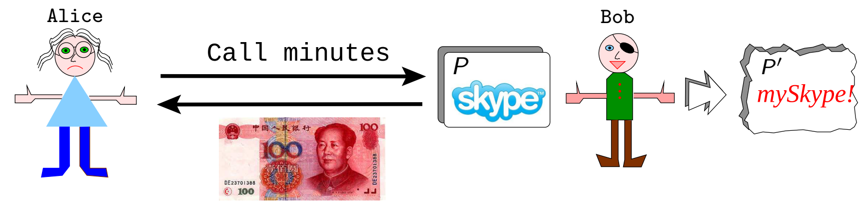
- Alice buys cycles from Bob's supercomputer.
- Bob snoops on confidential data/algorithms or tampers with Alice's program.

Scenario: License check tampering



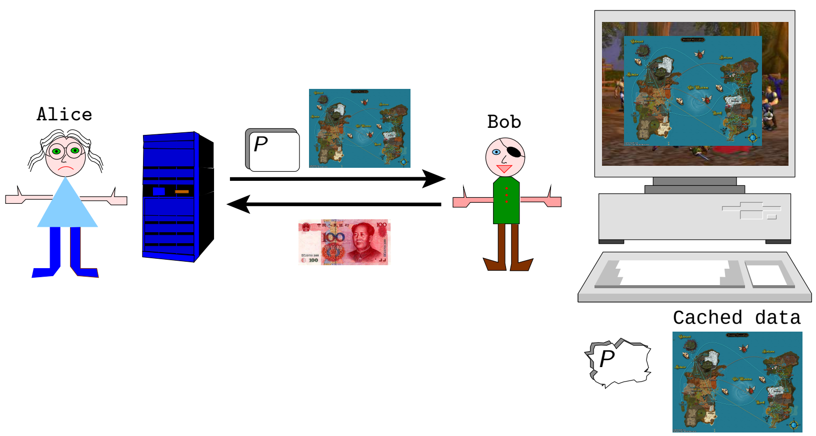
- Bob removes license checks to be able to run the program whenever he wants.
- Alice protects her program so that it won't run after being tampered with.

Scenario: Protocol discovery

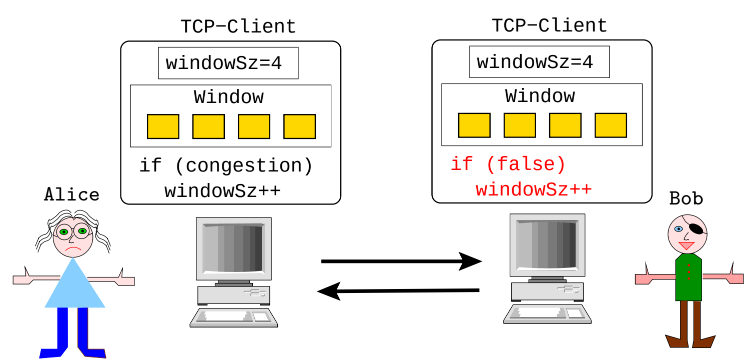


- Alice sells voice-over-IP call minutes.
- Bob examines the VoIP client to discover proprietary protocols to build his own rival client.

Scenario: Protecting networked computer games

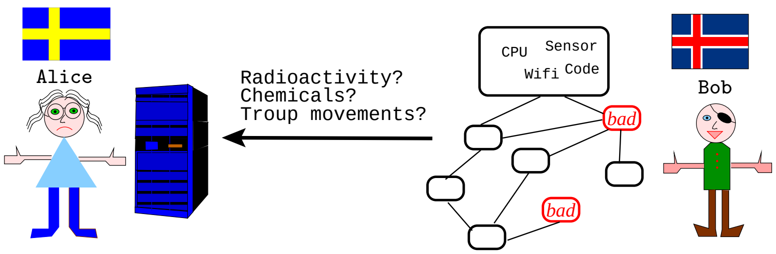


Scenario: Protecting Internet infrastructure



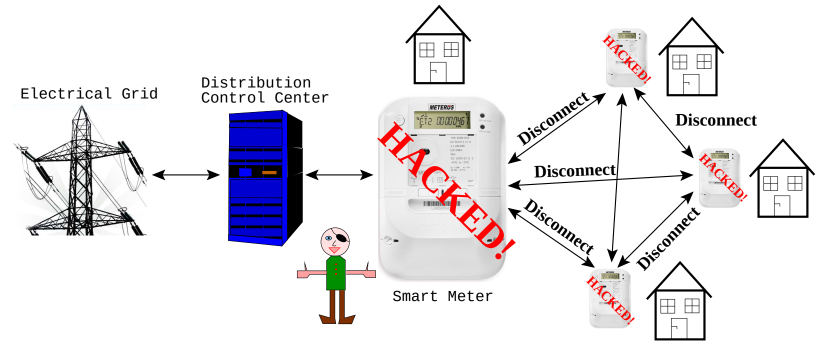
- Ensure well-behaved TCP window flow control.

Scenario: Wireless sensor networks



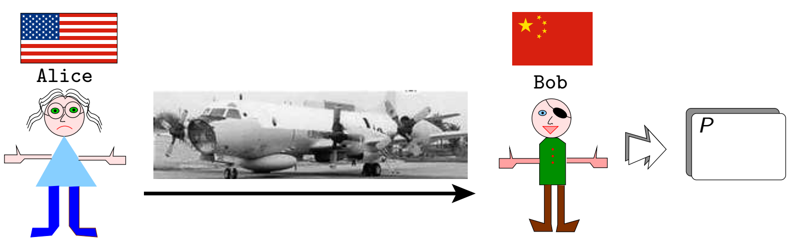
- Sensor networks are common in military scenarios.
- The enemy can intercept/analyze/modify sensors.

Scenario: Advanced Metering Infrastructure



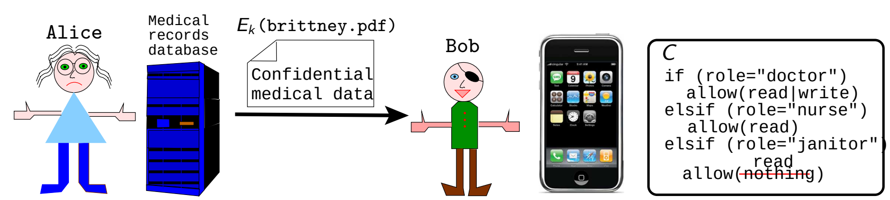
- Selective black-outs, consumers can adjust usage based on current costs, small-scale energy production...
- *What if a smart kid hacker sent out 5 million disconnect commands?*

Scenario: Protecting military software



- The military and intelligence communities would like to be able to track the whereabouts of classified software.
- In 2001, an EP-3 spy/reconnaissance plane landed on Hainan Island in China after a collision. The crew was unable to destroy all equipment.
- Much Air Force anti-tamper funding.

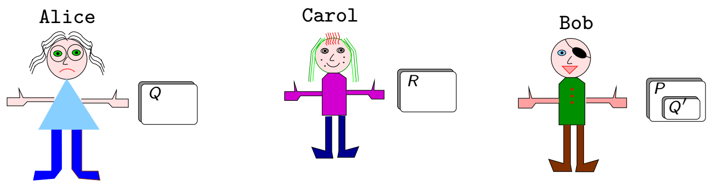
Scenario: Protecting medical records



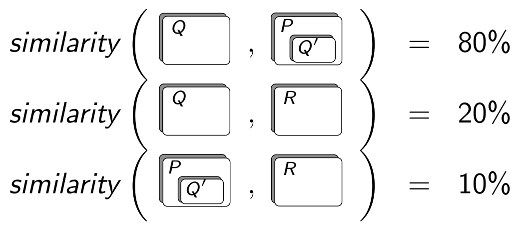
- Medical records must be protected from improper access and improper modification.
- Records are stored on one secure site, accessed from multiple (sometimes mobile) devices.

Scenario: Software plagiarism

- Student Bob copies a piece of Alice's program Q:

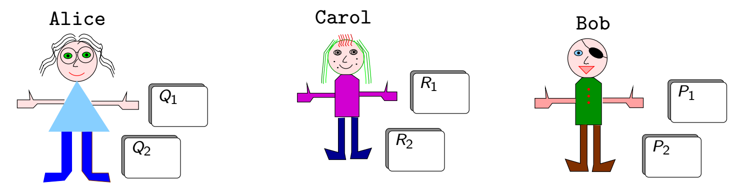


- Who has copied from whom?

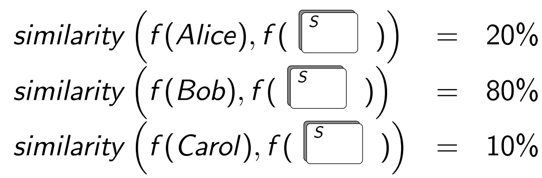


Scenario: Software forensics

- Who wrote program S ?

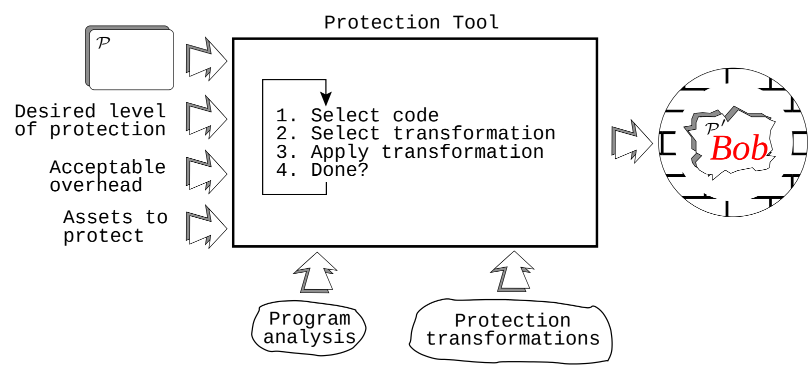


- Trace a malware author by comparing his programming style to those of known viruses.
- Extract features likely to identify each programmer:



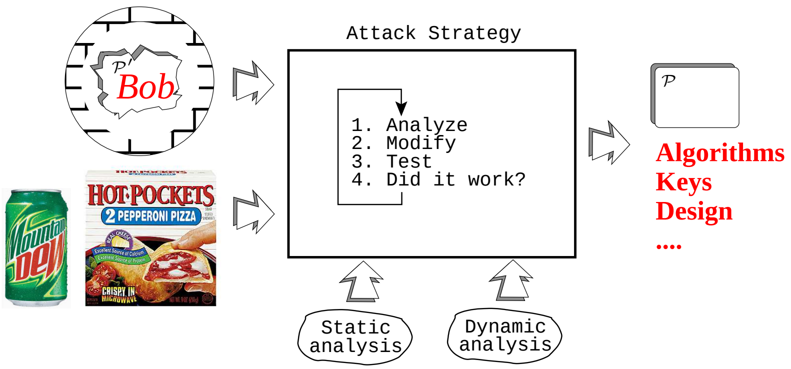
Outline

Protection Tools



- We build tools to protect $\mathcal{P}$ against attack.
- Look, we're building a compiler!
- Optimize for security, not speed! Programs will be larger, slower. . .

Attack Strategies



- The adversary has full access to $\mathcal{P}'$.
- He has static analysis tools: disassemblers, decompilers, slicers, . . .
- And dynamic analysis tools: debuggers, tracers, emulators, . . .
- And infinite energy and patience!

Outline

Attack models

- Attack model: your assumptions about the adversary's abilities and the strategies that he'll use to attack your system.
- For example:
  - the adversary cannot find the secret cryptographic key
  - the adversary won't try to tamper with the tamperproof smartcard.
- The adversary will try to think of ways to attack that are *not* in your model!

Outline

Code obfuscation

- To obfuscate a program means to

  transform it into a form that is more difficult for an adversary to understand or change than the original code.

- Vague definition of *difficult*:

  The obfuscated program requires more human time, more money, or more computing power to analyze than the original program.

Code obfuscation — Example obfuscated code

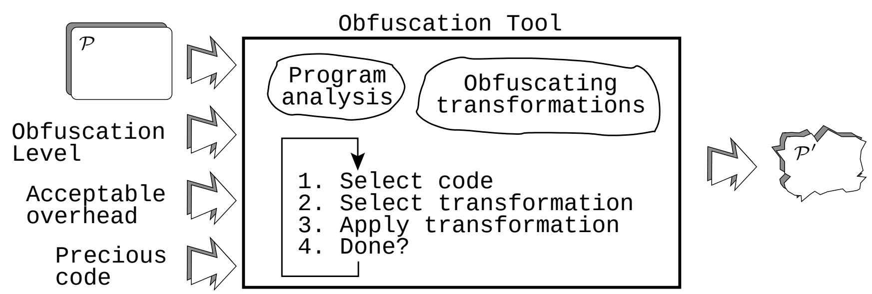
```
public class C {
 static Object get0(Object[] I) {
  Integer I7, I6, I4, I3; int t9, t8;
  I7=new Integer(9);
  for (;;) {
   if (((Integer)I[0]).intValue()%((Integer)I[1]).intValue()==0)
       {t9=1; t8=0;} else {t9=0; t8=0;}
   I4=new Integer(t8);
   I6=new Integer(t9);
   if ((I4.intValue()^I6.intValue())!=0)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


Code obfuscation — Example original code

```
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;
      }
   }
}
```

- An obfuscation tool turns the original code into obfuscated code.
- We want obfuscating transformations that make the program as hard to understand as possible.

Obfuscation Tool



Types of obfuscation

1. Abstraction transformations
   - Destroy module structure, classes, functions, etc.!
2. Data transformations
   - Replace data structures with new representations!
3. Control transformations
   - Destroy if-, while-, repeat-, etc.!
4. Dynamic transformations
   - Make the program change at runtime!

Obfuscation example: original program

```
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);
}
```


Obfuscation example: After abstraction transformation

```
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);
}
```

- It appears as if `main` calls the same function twice!

Obfuscation example: After data 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 (s==2)
        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);
        }
}
```

Obfuscation example: After control transformation

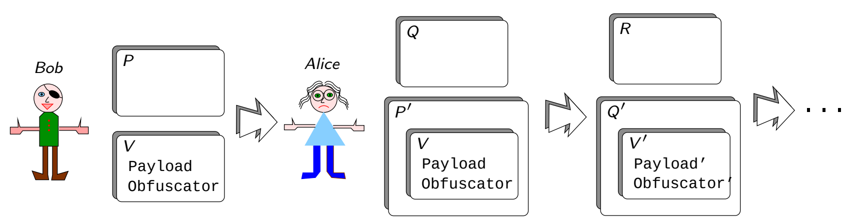
```c
int foobar(int x, int z, int s) {
   char* next = &&cell0;
   int retVal = 0;

   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;
}
```


Outline

Black hat code obfuscation

- Even bad guys can use obfuscation!
  1. Protecting viruses from virus scanners.
  2. Protecting misbehaving electronic voting code from discovery



Black Hat Code Obfuscation

- Even bad guys can use obfuscation!
  1. Protecting viruses from virus scanners.
  2. Protecting misbehaving electronic voting code from discovery
- Here's a program to tally the votes for American Idol:

```
% cat votes-cast.txt
alice
alice
bob
alice
dmitri
bob
zebra
```

```
% java Voting < votes-cast.txt
Total: 7
Invalid: 1
alice: 3
bob: 2
charles: 0
dmitri: 1
```

```
public class Voting {
  final int INVALID_VOTE = -1;
  int invalidVotes, totalVotes = 0;
  String[] candidates = {"alice", "bob", "charles", "dmitri"};
  int[] tally = new int [ candidates.length ];
  BufferedReader in = null; BufferedWriter log = null;
```

```
public class Voting {
  final int INVALID_VOTE = -1;
  int invalidVotes, totalVotes = 0;
  String[] candidates = {"alice", "bob", "charles", "dmitri"};
  int[] tally = new int [ candidates.length ];
  BufferedReader in = null; BufferedWriter log = null;

  public Voting() {
    in = new BufferedReader(new InputStreamReader(System.in));
  }
```

```
public class Voting {
  final int INVALID_VOTE = -1;
  int invalidVotes, totalVotes = 0;
  String[] candidates = {"alice", "bob", "charles", "dmitri"};
  int[] tally = new int [ candidates.length ];
  BufferedReader in = null; BufferedWriter log = null;

  public Voting() {
    in = new BufferedReader(new InputStreamReader(System.in));
  }

  public String readVote() {
    try {return in.readLine();}
    catch(Exception e) {return null;}
  }
```

```
public class Voting {
  final int INVALID_VOTE = -1;
  int invalidVotes, totalVotes = 0;
  String[] candidates = {"alice", "bob", "charles", "dmitri"};
  int[] tally = new int [ candidates.length ];
  BufferedReader in = null; BufferedWriter log = null;

  public Voting() {
    in = new BufferedReader(new InputStreamReader(System.in));
  }

  public String readVote() {
    try {return in.readLine();}
    catch(Exception e) {return null;}
  }

  public boolean isValidTime ( Date today ) {
    SimpleDateFormat time = new SimpleDateFormat("HH");
    int hour24 = Integer.decode(time.format( today)).intValue();
    return !(hour24 < 9 || hour24 > 21);
  }
```

```
public int decodeVote(String input) {
  for(int i=0; i < candidates.length; i++)
    if(candidates[i].equals(input)) return i;
  return INVALID_VOTE;
}
```

```
public int decodeVote(String input) {
  for(int i=0; i < candidates.length; i++)
    if(candidates[i].equals(input)) return i;
  return INVALID_VOTE;
}

public void logVote(Date date, int vote) throws Exception {
  if (log == null)
    log = new BufferedWriter(new FileWriter("log.txt"));
  log.write("TIME: "+ date+" VOTE: "+vote);
}
```

```
public int decodeVote(String input) {
  for(int i=0; i < candidates.length; i++)
    if(candidates[i].equals(input)) return i;
  return INVALID_VOTE;
}

public void logVote(Date date, int vote) throws Exception {
  if (log == null)
    log = new BufferedWriter(new FileWriter("log.txt"));
  log.write("TIME: "+ date+" VOTE: "+vote);
}

public void printSummary() {
  System.out.println("Total:"+totalVotes+
                     "\nInvalid:"+invalidVotes);
  for (int i=0; i < candidates.length; i++)
    System.out.println("Votes for "+candidates[i] +": "+tally[i]);
}
```

```
public void go() {
  while (true) {
    String input = readVote();
    int vote = 0;
    if (input == null)break;
    try {
      Date today = new Date();
      if (isValidTime(today)) vote = decodeVote(input);
      else                    vote = INVALID_VOTE;
      logVote(today, vote);
    } catch(Exception e) {}
    totalVotes++;
    if (vote == INVALID_VOTE) invalidVotes++;
    else                      tally[vote]++;
  }
  printSummary();
}
```

```
  public void go() {
    while (true) {
      String input = readVote();
      int vote = 0;
      if (input == null)break;
      try {
        Date today = new Date();
        if (isValidTime(today)) vote = decodeVote(input);
        else                    vote = INVALID_VOTE;
        logVote(today, vote);
      } catch(Exception e) {}
      totalVotes++;
      if (vote == INVALID_VOTE) invalidVotes++;
      else                      tally[vote]++;
    }
    printSummary();
  }

  public static void main(String args[]) {
    Voting voting = new Voting(); voting.go();
  }
}
```

```
public boolean isValidTime(Date today) {
  ...
  int hour24 = Integer.decode(time.format(today)).intValue();
  ...
}

public void go() {
    ...
    try {
      Date today = new Date();
      if (isValidTime(today)) vote = decodeVote(input);
      else                    vote = INVALID_VOTE;
      logVote(today, vote);
    } catch(Exception e) {}
    ...
}
```

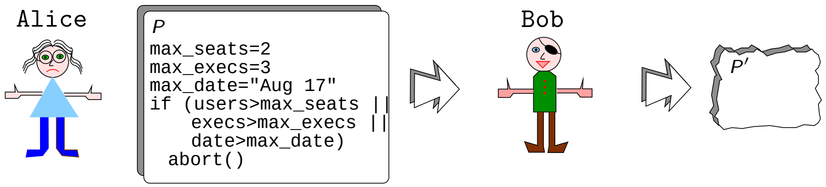
```
public boolean isValidTime(Date today) {
  ...
  int hour24 = Integer.decode(time.format(today)).intValue();
  ...
}

public void go() {
    ...
    try {
      Date today = new Date();
      if (isValidTime(today)) vote = decodeVote(input);
      else                    vote = INVALID_VOTE;
      logVote(today, vote);
    } catch(Exception e) {}
    ...
}
```

- Numbers that start with zero are interpreted as octal.
- Unexpected number-format exception between 8am and 9:59am!
- Alice gets all votes between 9 and 9:59!

Outline

Tamperproofing



- Tamperproofing makes the program useless to Bob if he tries to modify it!
- Necessary for
  1. digital rights management systems,
  2. license checking code

Two phases of tamperproofing

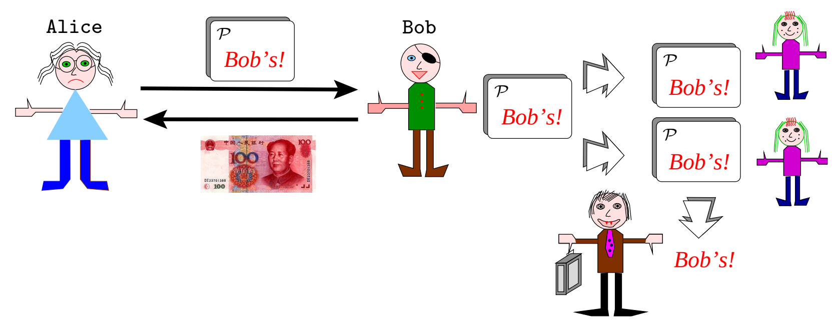
- Tamperproofing has to do two things:
  1. detect tampering
  2. respond to tampering
- Essentially:

```
if (tampering-detected()) abort
```

  but this is too unstealthy!
- Detection:
  1. has the code been changed?
  2. are variables in an OK state?
- Response:
  1. refuse to run,
  2. crash randomly,
  3. phone home, ...

Outline

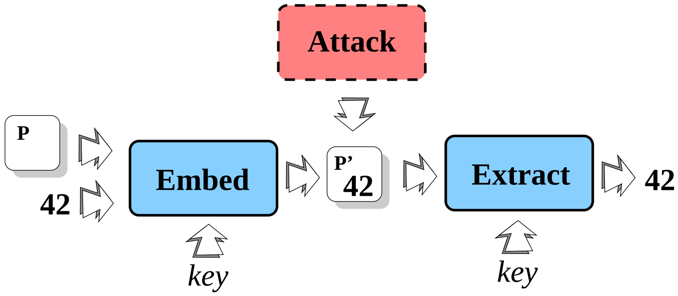
Software Piracy



- Bob buys one copy of Alice's program.
- Bob illegally sells copies to his friends.
- ⇒ Alice watermarks/fingerprints her program.
- Alice uses the fingerprint to trace the program back to Bob.
- Alice's lawyer sues for software piracy!

Watermarking API

- A watermarking system consists of two functions *embed* and *extract*:



- Bob wants destroy the mark before reselling the object!
  - Disturb the *extract* function so that Alice can no longer get the mark.
  - Example: Bob can obfuscate the program to destroy the mark!

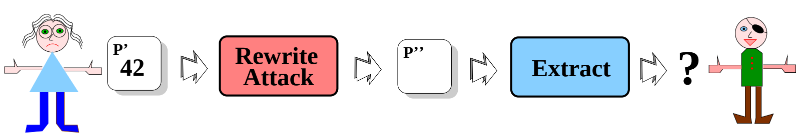
Trivial static watermark

- Embed the watermark as string constants included in the source of a program:

```
public class Fibonacci {
  String copyright = "Copyright ©
Alice";
  public int fibonacci ( int n ) {
    if ( n <= 2 )
      return 1;
    else
      return fib( n - 1 ) + fib( n - 2 );
  }
}
```


Attacks against software watermarks — Rewrite attack

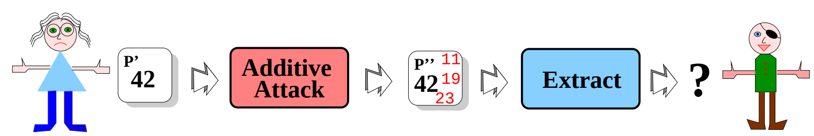
- Alice has to assume that Bob will try to destroy her marks before trying to resell the program!
- One attack will *always* succeed. . .



- Ideally, this is the *only* effective attack.

Attacks against software watermarks — Additive attack

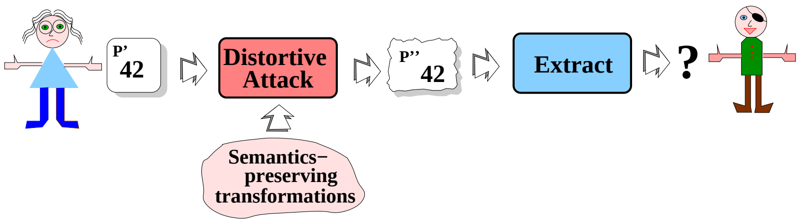
- Bob can also add his own watermarks to the program:



- An additive attack can help Bob to cast doubt in court as to whose watermark is the original one.

Attacks against software watermarks — Distortive attack

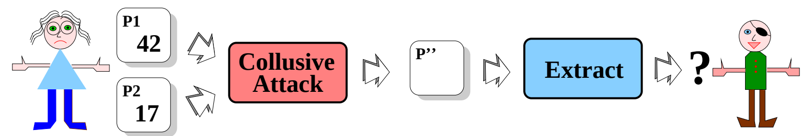
- A HLdistortive attack applies semantics-preserving transformations to try to disturb Alice's recognizer:



- Transformations: code optimizations, obfuscations,. . .

Attacks against software watermarks — Collusive attack

- Bob buys two differently marked copies and compare them to discover the location of the fingerprint:



- Alice should apply a different set of obfuscations to each distributed copy, so that comparing two copies of the same program will yield little information.

Outline

Basic Principles: Defense in Depth

- As in real life, we don't rely on just one means of protection.
- To protect your car, you
  1. lock it;
  2. put on a bar across the steering wheel;
  3. install a vehicle tracking system. . .
- We call this defense in depth.
- To protect a program, you
  - watermark it to protect against piracy;
  - obfuscate it to protect against reverse engineering;
  - tamperproof it to protect against modification;
  - and, you apply several different watermarking/obfuscation/tamperproofing algorithms!

Basic Principles: Renewability

- As in real life no protection lasts forever!
- When thieves get better, you
  1. buy better locks!
  2. buy thicker doors!
  3. buy a bigger gun!
- We call this renewability.
- To protect a program, you
  - monitor the abilites of your attacker;
  - be one step head of your attacker;
  - upgrade your defenses before an attack;
  - constantly invent new watermarking/obfuscation/tamperproofing algorithms!

Basic Principles: Diversity

- Make every
  1. distributed copy of a program different!
  2. instance of a software protection different!
- We call this diversity.
- Harder for the adversary to build scripted attacks.