1. Implement a system that stores secret messages in encoded code. The system would be an alternative to encryption; instead of some countries I would send a file that looks like Java code, but which actually encodes a secret message.

2. Implement a system that destroys steganographic messages stored in software, using, for example, various forms of obfuscation.

3. Implement a system that determines whether a piece of software is likely to contain a steganographic message, using, for example, statistical analysis.

---

Steganographic software II

- Peter Wayner, Minic Functions, CRYPTOLOGIA, Volume XVI.
- Mark Chapmen, George Davide, Hiding the Hidden: A Software System for Concealing Ciphertext as Innocuous Text. Springer Verlag, LNCS 1318.
Project 5–6: Software Watermarking I

5. Design and implement a software watermarking technique.
   - Pick one of the techniques discussed in the literature, or invent your own.
   - Watermark Java class files (preferred?), Java source, or Sparc binaries.
   - Evaluation: How well does the watermark stand up to de-watermarking techniques? How much slower/larger is the watermarked program than the original? Is it possible to tell (using statistical analysis) that the program is watermarked?

6. Design and implement a software de-watermarking technique.
   - Target a watermarking technique that some other team is implementing.
   - Evaluation: Does the watermark remain? How much slower/larger is the de-watermarked program?

Software Watermarking II

- Collberg, Thomborson, Software Watermarking: Models and Dynamic Embeddings (POPL'99)
  http://www.cs.arizona.edu/~collberg/Research/Publications/CollbergThomborson99a/index.html
- G. Qu, Fingerprinting IPs Using Constraint-Addition: Approach and Graph Coloring Case Study.
  http://www.cs.ucla.edu/~gangqu/ipp/fpgc.ps.gz

Project 4: Crippled-C I

4. AlgoVista is my latest project. It is a web-based search engine for computer scientists. See cgi.cs.arizona.edu/~collberg/AV.html.
   An important part of AlgoVista is that users can upload executable code (Java class files) into the database server.
   - For users who don’t know Java, I would like them to be able to upload C as well. Obviously, this isn’t safe.
   - Design Crippled-C, a safe subset of C. Implement a compiler for Crippled-C based on lcc, Fraser and Hansen’s free C compiler. Efficiency doesn’t matter that much, security does.
   - Use Software Fault Isolation to check stores.
   - Integrate into the AlgoVista server (maybe).
   - Evaluation: How much of C did you have to tear out to make it safe? How slow is it?

Crippled-C II

- Robert Wahbe, Steven Lucco, Thomas Anderson, Susan Graham, Efficient Software-Base Fault Isolation,
- Steven Lucco, Oliver Sharp, Robert Wahbe,


Software Obfuscation II


Project 9: Name De-obfuscation

9. Advanced name de-obfuscation. Take the output from any of the Java name obfuscators available on the net (or use your own from Assignment 1) and try to regenerate reasonable names.
   - Use all the type information available in the class files.
   - Java program make many references to standard libraries whose semantics are well-known. Use this information to generate good names: Example: `Hashtable x19 = new HashTable();` What are you going to rename x19 to?
   - Can you make use of information gathered from various standard compiler analysis? Which ones? Alias analysis, shape analysis?
   - Read up on Microsoft's Hungarian naming scheme.
   - Evaluation: Design a good evaluation scheme!
   - Extra credit: Generate good comments as well!

Software Watermarking III


Project 7-8: Software Obfuscation I

7. Design and implement a software obfuscation technique.
   - Pick one of the techniques discussed in the literature, or invent your own.
   - Evaluation: How well does the obfuscation stand up to de-obfuscation techniques? How much slower/larger is the obfuscated program than the original? Is it possible to tell (using statistical analysis) that the program has been obfuscated?

8. Design and implement a software de-obfuscation technique.
   - Target an obfuscation technique that some other team is implementing.
   - Evaluation: How close is the de-obfuscated program to the original? How much slower/larger is the de-obfuscated program than the obfuscated one?
Project 12: Tamperproofing

12. Design a library/compiler that allows a program to die gracefully and imperceptibly when it detects that it has been tampered with.
   - The idea is that whenever a program is executed it must check itself to see that it contains exactly the same code as when it was shipped.
   - Can you use Java’s “reflection”?
   - Often, we’re just interested in protecting one particularly important routine. Maybe it contains a password that we want to make sure no one changes.
   - Evaluation: How stealthy is the new code? How much slower is the tamperproofed program?

Project 10: Automatic Degrading of Programs

10. Automatic degrading of demo programs. Companies that sell images off the web will often allow you to download a degraded (low-resolution) version for inspection. Write a program that takes another program $P'$ as input, and produces $P''$. $P''$ has the same functionality as $P$, but is much slower or uses more memory.
   - A bit like obfuscation, but we want the program to be slower.
   - Stealthiness is important. Just inserting null loops is no good; they can easily be found and removed. You’ll want the program to be uniformly slower.
   - Can either be done on Java class files or binary executables.
   - Evaluation: How stealthy is the inserted code? How much larger is the program? How much control does the user have over the amount of downgrading?

Project 13–14: Aucsmith’s Tamper-Resistance

13. Implement and evaluate the cost of Aucsmith’s Tamper-Resistant Software Technique.
   - Can you do it for Java?
   - Evaluation: How much slower/larger is the tamperproofed program?

14. Implement a system that untangles a program that has been protected by Aucsmith’s Tamper-Resistant Software Technique.
   - I can probably get you samples of tamperproofed programs generated internally by Intel’s Aucsmith implementation.
   - This would be a great way to get your foot in the door at Intel!

Project 11: Design a Safe Language

11. Design and implement a “safe” version of a popular interpreted programming language (SafeAWK, SafeIcon...), to be used for code down-loading.
   - Write a Netscape plug-in to execute the safe program. Use a free implementation (gawk for Awk, Icon for Icon) as a starting point.
Project 17-18: Comparative Studies

17. Compare and contrast as many Java verifiers as you can get your hand on, with respect to correctness, elegance, and speed.

18. Compare and contrast as many Java obfuscators as you can get your hand on, with respect to correctness, elegance, and speed.

- Plead with the manufacturers to give you free evaluation copies of the software.

Reminders

- Remember: if you ever turn your project into a product the University of Arizona (and I!) will expect royalties! This is no joke; the UofA owns anything that either you or I produce.

- If you intend to work on an idea that’s been patented:
  1. Don’t make your code available to anyone, even for free. You’d be violating intellectual property rights. You can’t afford to be sued by Microsoft.
  2. If you do want to sell the product or give it away, make sure you contact the patent owner first. Or, better yet, let UofA’s lawyers do it for you.

Project 15: Software Birthmarks

15. Does each programmer have his/her own style? Can you measure it? Can you take two programs and determine whether they’ve been written by the same programmer?

- Write a program that reads in two programs A and B and determines whether they’ve been written by the same programmer.
- This is a bit like plagiarism detection.
- Court cases have been settled based on this kind of argumentation.

- Brenda S. Baker, Udi Manber, Deducing Similarities in Java Sources from Bytecodes.
  http://glimpse.cs.arizona.edu/javadup.html
- Read up on “Software Metrics.” I have lots of references.

Project 16: Software Fault Isolation

16. Implement SFI.

- Modify a Java interpreter to use Software Fault Isolation rather than verification.
- Evaluation: How much slower is SFI than verification?