University of Arizona, Department of Computer Science

CSc 620 — Assignment 3 — 40%

Christian Collberg
August 27, 2008

1 Introduction

This is your main project for the class.

The project is worth 40% of your final grade, divided (roughly) in four parts:

1. 5-page design document (5%),
2. a 25-minute presentation (5%), and
3. working code (15%),
4. a 10-13 page paper (15%).

Now, obviously, four three categories aren't completely independent! You can't expect to get a good grade by putting on a spectacular presentation and write a brilliant paper about a project that's essentially a POC!

The project should be done in teams of two students.

2 The projects

Table 1 lists the projects and teams. If you want to, you can switch teams with other students.

2.1 Project Wugui

Implement the infrastructure for a remote entrusting system (called biànliǎn$_J$) for Java, using *continuous replacement*.

Read Section 7.5.1 in the book as a starting point and also read *TR08-03:* biànliǎn*: Remote Tamper-Resistance with Continuous Replacement* by Christian Collberg, Jasvir Nagra, Will Snavely, `ftp://ftp.cs.arizona.edu/reports/2008/TR08-03.pdf`.

An undergraduate student, Will Snavely, has previously worked on this, so there's quite a lot of code already to start from. Get the existing Java code base from me, study it in detail, and make a plan for how to finish it up. Talk to Will (`wsnavely@cs.arizona.edu`) to learn about his design and the problems he had.

The system makes use of SOOT to manipulate Java bytecode programs so start by downloading and learning how to use it.

team	project	students	presentation
Wugui	Java infrastructure for a remote entrusting system using continuous replacement.	bhandari,sushanth	Tue Dec 2
Shifu	Java obfuscations for a continuous replacement system for remote entrusting.	ricarlos,robackja	Tue Dec 2
Po	C infrastructure for a remote entrusting system using continuous replacement.	anandp,xingqu	Thu Dec 4
Tigress	C obfuscations for a continuous replacement system for remote entrusting.	kpcoogan,jamyers	Thu Dec 4
Monkey	Measure and compare primitives for tamperproofing and obfuscation.	qingju,tpatki	Thu Dec 4
Mantis	Generate an infinte number of hash functions to prevent pattern matching attacks against tamperproofing systems.	ghigliom,marshall	Tue Dec 9

Table 1: Project assignment

Collaborate with **Project Shifu** to design the plugin system so that new obfuscation/tamperproofing algorithms can easily be added to the system.

Also build a set of testcases that you can use to evaluate the system.

Extra credit if you can get something working and written up for the deadline for the *Compiler Construction Conference* (2 October 2008 for abstracts, 9 October 2008 for full papers)![1]

2.2 Project Shifu

Collaborate with **Project Wugui** by implementing a set of obfuscations/tamperproofing algorithms that can be plugged into the bianliǎn$_J$ system.

For ideas, read Section 7.5.1 in the book and also read *TR08-03:* bianliǎn*: Remote Tamper-Resistance with Continuous Replacement* by Christian Collberg, Jasvir Nagra, Will Snavely, `ftp://ftp.cs.arizona.edu/reports/2008/TR08-03.pdf`.

The system makes use of SOOT to manipulate Java bytecode programs so start by downloading and learning how to use it. SOOT may already have some obfuscation algorithms that you can learn from. You could also check out the SandMark system which implements many obfuscation algorithms, although it uses a different bytecode manipulation system.

[1] Ha!

2.3 Project Po

Project Po mirrors **Project Wugui**, but for C! I.e. you should design the infrastructure for a remote entrusting system using continuous replacement. We'll call this system bianlian$_C$. The basic idea is to break a C program up into functions, apply obfuscations to generate an "infinite" sequence of different versions of each function, and replace these on the fly on the client.

Start by downloading and learning about the CIL C source code manipulation system

- `http://sourceforge.net/projects/cil`
- `http://manju.cs.berkeley.edu/cil`

This involves learning to program in O'Caml (a functional language with objects), too!

Read Section 7.5.1 in the book as a starting point and also read *TR08-03:* bianlian*: Remote Tamper-Resistance with Continuous Replacement* by Christian Collberg, Jasvir Nagra, Will Snavely, `ftp://ftp.cs.arizona.edu/reports/2008/TR08-03.pdf`. How will your design differ from the design of the Java system?

Also build a set of testcases that you can use to evaluate the system.

You may find that you need to do some binary-level manipulations as well, in which case you'll want to learn about a system such as Diablo:

- `http://diablo.elis.ugent.be/obf_download`

2.4 Project Tigress

Project Tigress mirrors **Project Shifu**, but for C! I.e. you should implement the obfuscations/tamper-proofing algorithms for the bianlian$_C$ system.

Start by downloading and learning about the CIL C source code manipulation system:

- `http://sourceforge.net/projects/cil`
- `http://manju.cs.berkeley.edu/cil`

Collaborate with **Project Po** to design the plugin system.

2.5 Project Monkey

It's very difficult to compare two software protection algorithms, even when it comes to performance. The idea for **Project Monkey** is to implement the *primitives* on which many algorithms are based, and compare *those*.

A masters student (Anand Patikkal) started this project last year, implementing the primitives for the Aucsmith dynamic obfuscation algorithm and the Horne tamperproofing algorithm. Start by reading up on these, Sections 7.2.3 and 6.2.2, respectively. Get the code base from me and study it in detail.

Next, look for other interesting algorithms from the book and identify the primitives on which they are based. For example, look at Section 7.2.1 for another tamperproofing algorithm that uses hashing, and 6.3.1 for one that uses encryption. Implement the primitives and measure! It's important to identify the parameters that matter (code size, cache size, etc.) and measure their effect on performance as they are varied.

Study how you can use CPU *performance counters* to get accurate profiles of things like cache effects.

2.6 Project Mantis

The Skype client was cracked in part because it was easy to find the hash functions that checked the code for modifications. See Section 7.5.5 in the book. Section 7.2.2 describes how one algorithm generated 3 million different versions of the same hash function, by obfuscating/randomizing the assembly code:

- `http://citeseer.ist.psu.edu/horne01dynamic.html`
- `www.freepatentsonline.com/20030023856.html`

What we *really* want, however, is a system that can generate an *infinite* number of hash functions, all different enough from each other that a pattern matching attack isn't possible. This could be done on the assembly code level, of course, but nicer might be to do it at the C code level. Or maybe both!

Additionally, it would be cool to be able to merge in the hash computation with existing code, rather than have the hash loop run on its own. This would make a dynamic attack much harder, where you run the program under tracing or emulation to find a tight loop that computes a value over the code segment.

To evaluate the system, look into *code clone* analysis from Software Engineering. This is a technique for finding similar pieces of code in different parts of a program.

3 The design document

The design document should describe

- the overall design of the system,
- a detailed module structure,
- which tools will be needed to build the system,
- a detailed time-line including time for testing,
- an evaluation plan.

Run the design past me. Also, discuss it with the other students in the class, particularly those who are working on projects related to yours.

The timeline should have *milestones*, such as "Nov 2 we will have completed a system which can perform the following." When you hit a milestone you should come to my office and report.

I like projects where "someething always works", i.e. you start with an "empty" (but running) system and gradually add and test features. This way, you always have something to show me when I walk past your desk and say "so, how's the project going?", avoiding potential embarrassment. Also, it makes *you* feel better about your progress!

The design document should be 3-5 pages long and is due Tuesday September 23.

4 The paper

The project report should be 10-13 pages, in a form that would make it possible to submit to a conference. It should be written in LaTeX.

A typical research paper has the following sections.

1. Abstract.
2. Introduction. What problem are we solving? Motivate why this is an important problem! Argue that previous solutions are inadequate. Outline your solution, the major results, shortcomings of the solution. Be very concrete: "In this project we..." Outline the rest of the paper: "In the remainder of this paper we will: ..."
3. Related work. Papers and systems that solve the same problem as you but which are inferior to yours in some way and better in others, systems and papers that are actually different (solve a different problem) but could be confused with yours.
4. System overview. Include a drawing of the major components of the system.
5. Algorithms. Give detailed algorithms where necessary. Describe design pitfalls, things you could have done differently, stuff left out of the system.
6. Evaluation. Measurements, comparisons to other systems.
7. Future work. Yhings you would have done differently had you had more time, ideas for future studies.
8. Conclusion. What did you do. What did you learn? Be very concrete: "We built a system which: .... Our system is 25% faster than ..."
9. Bibliography.

The paper is due the last day of class, Wednesday December 10.

5 The presentation

You have 20 minutes to present your project, plus 5 minutes for questions. It's appropriate to have 15-20 slides (you can use whatever system you want to make them), and prepare a quick demo, if appropriate. The presentation should be similar to what you might see at an academic conference. It's important to spend enough time on the *motivation* for the project: why did you do this, what problem are you solving, what has been done previously? Also, remember to summarize the outcome of your study and point to future work.

6 OK, what should we do???

1. Start by reading up on any relevant papers and sections in the book.
2. Study any tools that you will be needing for the project.
3. Ask me for any code that I might already have from previous projects.
4. Set up a repository (such as CVS) so that you can share code between the team members.
5. Prepare the design document.
6. Start coding!
7. Whenever you hit a milestone on the timeline come see me and show me what you've got.
8. If your timeline is slipping, come see me *immediately* and explain what's going on! Are you slacking off? Did you run into some unexpected problems? Was your timeline too optimistic?
9. Don't wait till the end of the semester to write up the paper. Start writing it immediately at the beginning of the semester, and keep it up to date as the project develops.
10. A week before your presentation it's time to stop coding and start writing! Finish the paper and prepare slides for the presentation.
11. Package everything up neatly, make sure that all files (code, paper, slides) are present, and submit to me.