**6.858 Fall 2014 Lab 7: Final project**

|  |  |
| --- | --- |
| **Piazza idea discussions due:** | Thursday, October 23, 2014 |
| **Proposals due:** | Friday, October 31, 2014 |
| **Presentations due:** | Wednesday, December 10, 2014 (in class) |
| **Code and write-up due:** | Friday, December 12, 2014 (5:00pm) |

**Introduction**

In this lab, you will work on a final project of your own choice. Unlike in previous labs, you will work in groups of 3-4 for the final project. You will be required to turn in both your code and a short write-up describing the design and implementation of your project, and to make a short in-class presentation about your work. We will post your write-up and code on the web site after the end of the semester, unless you explicitly talk to us about why you want to keep yours confidential.

The primary requirement is that your project be something interesting. Your project should also have something to do with security, but that's relatively easy, and it's much more important for your project to be interesting.

We encourage students to choose any project idea that you might think is interesting. If you are not sure, we provides you with two kinds of ideas. First, we present two reasonably well-defined starting points for projects. Second, we give a list of half-baked ideas that we think could turn into an interesting project, but we haven't given them too much thought.

We encourage final projects that leverage multiple classes you might be taking, or that involve other research or projects you are already working on. For example, if you are also taking 6.828 or 6.S897 (elections and voting technology), it would be fine with us to have a single project that counts for both 6.858 and another class. Same for other upper-level course-6 classes or MEng and AUP projects.

Most of the final projects from last year are posted [here](http://css.csail.mit.edu/6.858/2013/projects.html). Several final projects from previous years ended up being subsequently published as research papers (e.g., [UserFS](http://people.csail.mit.edu/nickolai/papers/kim-userfs.pdf), [BStore](http://people.csail.mit.edu/nickolai/papers/chandra-bstore.pdf), and [LXFI](http://people.csail.mit.edu/nickolai/papers/mao-lxfi.pdf)). If this sounds interesting to you, try to pick an ambitious class project that you might want to continue working on afterwards!

**Deliverables**

There are four concrete steps to the final project, as follows:

**Form a group.** Decide on the project you would like to work on, and post short summary of your idea (one to two paragraphs) on Piazza; use the lab7 tag/folder on Piazza.

Discuss ideas with others in comments on their Piazza posting. Use these postings to help find other students interested in similar ideas for forming a group. Course staff will provide feedback on project ideas on Piazza; if you'd like more detailed feedback, come chat with us in person.

**Project proposal.** Discuss your proposed idea with course staff over the next week, before the proposal deadline, to flesh out the exact problem you will be addressing, how you will go about doing it, and what tools you might need in the process. By the proposal deadline, you must submit a one-to-two-page proposal describing: your **group members** list, **the problem** you want to address, **how you plan to address it**, and what are you proposing to **specifically design and implement**.

Submit your proposal to [the submission web site](http://css.csail.mit.edu/6.858/2014/labs/handin.html).

**Project presentation.** Prepare a short in-class presentation about the work that you have done for your final project. We will provide a projector that you can use to demonstrate your project. Depending on the number of groups and the kinds of projects that each group chooses, we may decide to limit the total number of presentations, and some groups might end up not presenting in class.

**Write-up and code.** Write a document describing the design and implementation of your project, and turn it in along with your project's code by the final deadline. The document should be about 2-3 pages of text that helps us understand what problem you solved, and what your code does. The code and writeups will be posted online after the end of the semester. Take a look at the list of writeups from past years ([2012](http://css.csail.mit.edu/6.858/2012/projects.html) and [2013](http://css.csail.mit.edu/6.858/2013/projects.html)) to get a sense of what this writeup should look like.

**Well-defined starting points**

We provide two starting points if you want a well-defined project. One involves building a defensive system, namely an encrypted file system that allows users to share data through an untrusted server. The other involves finding vulnerabilities in existing systems, namely in various MIT services that IS&T runs.

**Encrypted file system**

Your goal for this project idea is to develop a file system that allows users to store data on an untrusted file server. The file server should not be able to obtain the user's plaintext data (i.e., your file system should encrypt the data), and the file server should not be able to corrupt the data either (i.e., your file system should authenticate the data it gets back from the file server).

Your file system should support many users, and allow users to share files with one another. For each file, it should be possible to control the set of users who can read, and who can write, to that file.

At a minimum, your file system should meet the following requirements:

* Users can create, delete, read, write, rename files.
* The file system should support directories, much like the Unix file system.
* Users should be able to set permissions on files and directories, which also requires that your file system be able to name users.
* File names (and directory names) should be treated as confidential.
* Users should not be able to modify files or directories without being detected, unless they are authorized to do so.
* If the server is not malicious, unauthorized users should not be able to corrupt the file system.
* The file server should not be able to read file content, file names, or directory names.
* The server should not be able to take data from one file and supply it in response to a client reading a different file.
* A malicious file server should not be able to create or delete files or directories without being detected.

The final result of this project should be a functional file system implementation that meets the above requirements. You can implement your prototype in any language you want, such as Python or C++ or Go. You can decide how the file system client and server should be run. One reasonable design would be to have the file system client provide a minimal shell environment that allows users to perform the operations described in the above requirements.

As a challenge, you may want to think about providing freshness guarantees: that is, that a client should always see the latest version of a file, or at least that a client should never see an older version of a file after it sees a newer one. The [SUNDR file system](https://www.usenix.org/legacy/events/osdi04/tech/full_papers/li_j/li_j.pdf) may provide some inspiration for this, although it is a rather complicated system. You can choose to implement some limited freshness guarantees, unlike SUNDR's ideal fork consistency.

As another challenge, you may want to think about integrating your file system with the OS of your choosing. You may find [FUSE](http://fuse.sourceforge.net/) helpful for this.

**Looking for vulnerabilities**

If you are interested in a more attack-oriented final project, your goal for this project is to pick an interesting service provided by MIT's IS&T (or any other computing service at MIT, such as those provided by SIPB), and try to find vulnerabilities in it. We will judge your project based on what kinds of vulnerabilities you find. Beware that there's no guarantee of success with this (or any other attack-oriented) project, because you may accidentally choose a very secure service, and might end up finding no vulnerabilities, which can potentially result in a failing grade. However, if you have an interesting approach to finding vulnerabilities (e.g., you have designed a new tool for finding bugs), you may receive a good grade even without finding real vulnerabilities.

**IMPORTANT:** In any attack-oriented project, you must be very careful to avoid disrupting existing services, inconveniencing users of those services, compromising the security of that service, or taking advantage of any vulnerabilities you find. If you are ever in doubt, please get in touch with us. If you discover real vulnerabilities in a service, please get in touch both with the operators of that service and with us. Please don't exploit the vulnerability to gain any additional privileges on a service, and don't announce it widely before the service operators have a chance to understand it.

Your grade in this project depends on how interesting the vulnerabilities are that you have found, or how interesting the techniques are that you used to discover those vulnerabilities. For example, obtaining passwords through traditional brute-forcing techniques is not interesting. The general scale of work expected should be comparable to the amount of work involved in building a file system (see project above). Of course, much of your work will involve reading and understanding an existing system, and carefully constructing proof-of-concent exploits to demonstrate the vulnerabilities that you discovered, and the total amount of resulting code may be minimal; think back to how much time you spent on lab 1, and how many lines of code your final exploits were.

Here is a list of services that might be interesting targets for security evaluation as part of this project:

* AFS
* Kerberos
* Moira
* MIT's wireless network
* MIT CA infrastructure
* [Touchstone](https://idp.mit.edu/)
* [scripts.mit.edu](http://scripts.mit.edu/)
* [sql.mit.edu](http://sql.mit.edu/)
* [Stellar](http://stellar.mit.edu/)
* [WebSIS](http://student.mit.edu/)
* [IS&T Web APIs](http://ist.mit.edu/web-api) and [Mobile APIs](http://mobile-dev.mit.edu/docs/)
* [MIT mobile](https://github.com/MIT-Mobile/MIT-Mobile-for-iPhone)
* [XVM](http://xvm.mit.edu/)
* Athena dialup machines
* MIT's Exchange mail servers
* [The 6.858 submission web site](https://6858.scripts.mit.edu/submit/handin.py/)
* [MIT OpenID Connect](https://oidc.mit.edu/)
* MIT ID cards, though check out [prior work](http://css.csail.mit.edu/6.858/2014/readings/mit_prox_vulns.pdf)

If you are interested in looking for bugs in MIT's newly developed APIs or OpenID Connect, several folks at IS&T have volunteered to help you get started: Chris Giles <csgiles@mit.edu>, Stephen Buckley <sbuckley@mit.edu>, and Thomas Hardjono <hardjono@mit.edu>.

**Half-baked project ideas**

If you are interested in working on something other than the two projects proposed above, here are some half-baked ideas. Use them to come up with more well-defined projects; we expect the final work to be comparable in terms of total effort to the two well-defined projects listed above. Of course, we encourage you to come up with your own ideas for what you would like to work on; there's no need to restrict yourself to this list.

* Get the concolic execution system from lab 3 to work on real-world Python web applications, and try to find real bugs.
* Write an interesting web application in Ur/Web.
* Modify Linux to keep track of the last process and user that modified each file. Perhaps log a warning or raise an alarm when a file is modified by an application that hasn't written to that file before.
* Build a static analysis tool to find bugs in real web applications. It would be great to have a static analysis tool for Python programs, even if it's not perfectly sound, much like the PHP static analysis tool we discussed in lecture. In addition to the standard XSS bugs, can you find bugs where application developers forget to perform permission checks, or inadvertently leak sensitive data? Such a tool might also be useful to find non-security bugs in Python programs. One existing tool that's quite limited is [PyLint](http://pypi.python.org/pypi/pylint).
* Find bugs in real C code. Take a look at some recent research papers on finding real bugs in the Linux kernel and other C-based programs, involving [integer errors](http://people.csail.mit.edu/nickolai/papers/wang-kint.pdf) or[undefined behavior](http://people.csail.mit.edu/nickolai/papers/wang-stack.pdf). We can give you access to our source code to these tools; you can apply them to existing software to see what bugs you can find, and also think about ways to extend the tools to make them more accurate, to make them find other kinds of bugs, etc.
* Extend program analysis tools used by Linux kernel developers, such as [sparse](https://sparse.wiki.kernel.org/index.php/Main_Page) and [smatch](http://repo.or.cz/w/smatch.git), to catch different kinds of bugs in the kernel.
* Use the [KLEE](http://klee.llvm.org/) symbolic execution system to build an analysis tool that finds interesting bugs in C programs.
* Explore the extent to which covert channels / side channels matter, e.g. in shared VMs like EC2, vs. shared OS, vs. other environments. See [this paper](http://css.csail.mit.edu/6.858/2014/readings/get-off-my-cloud.pdf) for some background information.
* Add Capsicum support to Linux. There has been some [recent work](http://lwn.net/Articles/604015/) toward capsicum on linux proposed [here.](http://lwn.net/Articles/603929/)
* Port an interesting application to use Capsicum (on FreeBSD).
* Build an interesting application on top of Webathena ([site](https://webathena.mit.edu/), [code](https://github.com/davidben/webathena)), or extend it to support non-DES enctypes (AES, etc).
* Implement Kerberos for Android. We can put you in touch with some folks at the Kerberos Consortium that are working on this. It would be interesting to implement a Kerberos ticket caching service, accessed via Android intents, so that an application can use Kerberos tickets for a specific service without being able to steal the user's entire TGS ticket.
* Implement progressive authentication: instead of requiring the user to log in to do anything, require different levels of credentials for specific tasks. For example, on Athena, perhaps running a web browser should not require any credentials, but accessing your personal files (or your browser accessing your google.com cookie that gives access to gmail) should require your Kerberos password, etc. The same would apply on an Android phone: checking the weather or the map should not require any credentials, but accessing the mail app should prompt for a PIN, swipe pattern, or password. [A paper from Microsoft Research](http://research.microsoft.com/pubs/168102/pa.pdf) might give you some things to think about, although it may not be worthwhile to implement all of the environment monitoring done in that work.
* Examine the security of Android applications. Look at some previous studies for inspiration ([one](http://usenix.org/events/sec11/tech/full_papers/Enck.pdf), [two](http://usenix.org/events/sec11/tech/full_papers/Felt.pdf)).
* Improve sandboxing for Android applications. Is there something that you could do to prevent a malicious application from exploiting kernel bugs? Consider [recent improvements to seccomp](http://lwn.net/Articles/441232/), using [Linux KVM](http://www.linux-kvm.org/), or using Native Client.
* Look for weaknesses in random number generation, or improve the random number generation. Some hints for where to start: [Mining your Ps and Qs](https://factorable.net/weakkeys12.extended.pdf), [Boot-time Entropy](http://cseweb.ucsd.edu/~hovav/dist/earlyentropy.pdf),[Ted T'so on Linux PRNG design](https://news.ycombinator.com/item?id=6548893), [Theoretical weaknesses in the Linux PRNG](http://eprint.iacr.org/2012/251.pdf).
* Audit interactions between Android applications, perhaps by tracing all intents sent via the reference monitor. When something goes wrong, can your system tell the user why an application might be broken?
* Implement an [environmental key generation system](http://www.schneier.com/paper-clueless-agents.html).
* Build a password manager for Android applications, perhaps by implementing a virtual keyboard that can automatically enter passwords into an application. The virtual keyboard can know exactly what application it's entering passwords into, which can guard against phishing-style attacks.
* Provide more fine-grained network access control in Android, to protect an internal corporate network from possibly-malicious applications on a phone.
* Implement an efficient version of Baggy bounds checking on top of [Clang](http://clang.llvm.org/) and [LLVM](http://llvm.org/).
* Use DynamoRIO's binary instrumentation to implement some cool security mechanism for unmodified binaries. For example, you could implement a binary-level taint tracking system that prevents secret files from being sent out over the network. You can look at a previous lab we used to have involving DynamoRIO from a few years ago[here](http://css.csail.mit.edu/6.858/2010/labs/lab2.html).
* Use [Resin's taint tracking for Python](http://pdos.csail.mit.edu/resin/) to enforce some interesting properties in zoobar.
* Find an interesting use for trusted hardware, and figure out how to expose trusted hardware safely to applications. Linux should already have a basic device driver for a TPM.
* Write a tool to help privilege-separating Python applications, and apply it to the Zoobar labs or other real applications.
* Implement more flexible protection mechanisms for Linux (so that any user can create additional protection domains -- sub-users -- to run code with less privileges, without having to be root). You can build upon a class project from a previous year, [UserFS](http://people.csail.mit.edu/nickolai/papers/kim-userfs.pdf).
* Improve the security of HTTPS in web browsers in the face of possibly compromised CAs. For example, define a new URL syntax that includes the server's certificate public key in the URL itself, so that one site can unambiguously include a link to another site without relying on CAs. Or, include the CA name in the URL, so that another CA cannot subvert security. See [SSL observatory](https://www.eff.org/observatory), [perspectives](http://perspectives-project.org/), and [CertPatrol](http://patrol.psyced.org/).
* Auditing support for web applications. For instance, suppose someone broke into your blog or forum, added a user account for themselves, changed permissions, and posted garbage messages. How can you track down all of the changes made by the attacker? Could be done with the help of a language runtime, such as Resin.
* The Tor Project has some ideas for possible Tor-related projects [here](https://www.torproject.org/getinvolved/volunteer.html.en#Projects).
* Analyze the Bitcoin transaction graph. How hard is it to anonymize Bitcoin exchanges? Here's one recent [paper](http://eprint.iacr.org/2012/584.pdf) analyzing the Bitcoin data set, which might give you ideas for other things to try.
* Implement one of Daniel J. Bernstein's ECC curves (e.g., Curve3617) in OpenSSL/OpenSSH. See [this page](http://safecurves.cr.yp.to/) for more information.