CS380S: Graduate Computer Security

1. Course Contact 2
2. Course requirements 2
3. Grading / Attendance 2
4. Class attendance 3
5. In-class paper presentation 3
6. Paper writeups ..... **Error! Bookmark not defined.**
7. Project 4
8. Class project midterm writeup 5
9. Class project final writeup and presentation 5
10. Class participation ..... **Error! Bookmark not defined.**
11. Project ideas 5
12. Policy on Academic Accommodations 7
13. Academic Integrity 8
14. Artificial intelligence 8
15. Religious holy days 8
16. Class Recordings 8
17. Class Calendar 8

1. Course Contact

Dr. Shravan Narayan
Classes: 12:30pm to 2pm at GDC 2.410 on Tuesday and Thursday.
Office Hours: At GDC 6.430, 1pm to 1:45pm on Wednesday and 2pm to 3pm on Thursday.
(Email to let me know you're coming, or if you need alternate meeting times)
Email: shr@cs.utexas.edu (Expect a response within 48 hours)
TA/Grader: Rohan Gangaraju rgangar@utexas.edu (Always cc the Instructor)
Canvas: https://utexas.instructure.com/courses/1441038

(Syllabus last updated: Jan 12th, 2026)

2. Course requirements

It is expected that all students have taken CS 361s or equivalent. While this is technically not a hard requirement, students without this experience will likely find the course extremely challenging. Students without the requisite background of CS 361s who still want to take this course are expected to look through the readings in CS361s as needed on their own time to keep up with this course.

At a minimum, students are expected to have knowledge of programming with C, compiling C applications, the memory layout of C applications including stack and heap, as well as standard terms and operations from programming and compilation such as how control flow (branches, indirect function calls) work, what a program counter does etc. While the course will introduce the concept of memory safety, this is mainly meant as a refresher. Students should be familiar with this and those unfamiliar with memory safety are expected to go through the additional material listed in this course calendar or from UT's CS361s course.

3. Grading / Attendance

Grade breakdown is as follows:

- Class attendance – 10% (twice a week after the first week)
- In-class paper presentations – 15%
- In-class post presentation quizzes – 15%
- In-person attendance of visiting scholar presentations
    - On Secure Crypto Systems Feb 17th, 11am, GDC 2.216 – 4%
    - Any 3 additional presentations (See details on Canvas) – 6%
- Project selection – 0% (one para writeup due on Feb 10th)
- Class project midterm writeup – 10% (on Mar 26th)
- Class project final
    - Presentation – 15% (on Apr 21st, Apr 23rd)
    - Final writeup – 10% (once on Apr 27th)
    - Project quality – 15%

Each of these are explained in the next few sections.

Grading scheme (Final scores maybe curved by the instructor)

- A : 100% to 94%
- A- : 94% to 90%
- B+ : 90% to 87%
- B : 87% to 84%
- B- : 84% to 80%
- C+ : 80% to 77%
- C : 77% to 74%
- C- : 74% to 70%
- D+ : 70% to 67%
- D : 67% to 64%
- D- : 64% to 61%
- F : 61% to 0%

4. Class attendance

Attendance in this class is required starting week 2 (Jan 20th). See "**Error! Reference source not found.**" below if you need to skip classes. Attending the last week of class is required as you will be required to present your project – only UT excused absences will be accommodated.

This class has a total of 3 skips for class attendance that you can use **without** any explanation/email/note to me. These are for any situations that may come up that is not recognized as a UT-approved reason for skipping class/assignments.

Each subsequent skip will result in a loss of 2% from your total grade. Skipping more than 8 classes in total will automatically result in a failing grade.

Suggestion: try to reserve these skips for emergencies.

5. In-class paper presentation

Each student is expected to present one class topic/research-papers as a team of two starting Jan 20th. The topic will be from a list of preselected papers listed in Class Calendar section in this document. The topic sign-up will be available on the Canvas website and are on a first-come first-serve basis. The presentation can use powerpoint or other suitable presentation aids and is expected to have 50 mins of content with 10 mins of Q&A during or at the end of the presentation. Presenters should be well versed in both the papers as well as background material. Bonus points for including some background or follow up research in the presentation.

Quiz question preparation: Presenters are also expected to make a list of 5 questions that will be answered by the other students in the class. See next section for details.

Note: Topics/Papers may be modified based on student interest/new security news etc. This will be done with at least one week notice, and after discussion with the student who signed up for the topic.

6. In-class post presentation quizzes

Presenters are also expected to make a list of 5 questions which can be answered in a sentence. These will be answered by the other students in the class. The goal here is to test whether everyone was paying attention during the presentations. The top 15 quiz scores for each student will be considered towards this portion of the grade.

7. Visiting scholar presentations

In the spring semester, visiting scholars and graduate students from different universities present talks at UT Austin about their research. These talks are some of the highest quality research talks that are accessible – even better than conference talks as they are one-hour long presentations that are meant for a general CS audience rather than expert practitioners.

The talks are typically held at 11am on specific Tuesdays, Thursdays, and Mondays at 11am on GDC 2.216. A precise schedule will be made available on Canvas.

On Feb 17th, 2026, 11am, a visiting scholar will be presenting their research on employing cryptographic techniques to build secure systems. Since this is a topic we don't cover in this course, attending this talk is a required part of this course and consists of 4% of your grade. Beyond this talk, you may attend any 3 of the remaining talks that are scheduled for this semester.

Proof of attendance: To submit proof that you have attended the talk, take a picture of an interesting slide in the speaker's talk (DO NOT DISTURB THE SPEAKER WHEN DOING THIS). You will be able to upload the picture to Canvas. Each student is expected to take and upload a picture themselves – any re-use of pictures amongst students is prohibited and will be reported as an instance of cheating to the Dean's office.

Note: Even though the talk is available on zoom, you must attend the four talks in-person. Zoom attendance will not be credited, as we don't track zoom attendances in any way.

Can't attend the visiting scholar presentations due to other commitments?
Talk to the instructor and then sign up to present a second topic in the class. The second presentation's score will be used in place of attending the visiting scholar talks.

8. Project Selection

The goal for the course project is to first develop or build upon existing security hardening techniques, next, modify existing applications to use the security technique, and finally evaluate their impact on security and performance. The modifications will need to be evaluated on large popular applications such as browsers, the Linux kernel, frameworks like tensor flow etc.

The projects should be done in groups of up to 3. Groups are expected to clearly document what contributions each member of the team has made to the project.

You are welcome to develop your own project ideas in systems security and discuss this with me. Alternately, you can build on or implement one of the existing project ideas listed in this document under Project ideas and additional ideas I will share in class.

Project group and topics selection should be complete by Feb 10th, although you are welcome to select your project earlier as well. A one paragraph writeup is due in Canvas describing your project (this is not graded). If you choose to develop your own project ideas, you must get my approval before using this as your class project. You are welcome to ask me about possible projects at any time in the course.

You are encouraged to be ambitious and try a challenging project that you think would be fun. Students who completely execute an easy project will score the same as students who have partial success on an ambitious project.

9. Class project midterm writeup

Project groups are expected to provide a two-page single-spaced writeup on the progress of their project on Mar 26th via Canvas.

10. Class project final writeup, presentation, and quality score

Project groups are expected to provide a five-page single-spaced writeup on their project on Apr 27th via Canvas as well as a 15-minute presentation in class of their project on Apr 21st or Apr 23rd. The exact day of the presentation will be decided in class. Additionally, students will receive a score based on project quality – a score assigned based on effort, success, ambition, and clarity of the project.

11. Project ideas

Here are a few project ideas that you are welcome to use as this course's project. You are also welcome to develop your own project ideas in systems security and discuss this with me. If you choose to develop your own project ideas, you must get my approval to use your idea as this course project.

- Use RLBox to sandbox a library in a major application or framework. To sandbox the library, configure RLBox and modify the build scripts to use a WebAssembly sandbox. Compare the performance of this with that of a Native Client sandbox. A tutorial to use RLBox is available here https://rlbox.dev/. Some examples of libraries that have been sandboxed in the past (please find new examples for this project):
    - Sandboxing libjpeg in the TensorFlow framework
    - Sandboxing libjpeg (or any file format parsing library) in ClamAV
    - Sandboxing markdown-to-html libraries in the Apache web server or in standalone apps

  Pre-reading: RLBox: Retrofitting Fine Grain Isolation in the Firefox Renderer
  Skills: Comfortable with C++. Experience using Makefiles will help but is not necessary.
  Level of difficulty: Easy/Moderate.

- Speedup Chrome's compressed pointer heap to use memory accesses based on Intel x86 segmentation. Similar to Native Client or WebAssembly, Chrome uses a contiguous heap for JavaScript code, as you will read about in class. Chrome accesses these contiguous memories using the usual load/store instructions. However, prior research shows that Intel x86 allows a more optimized way to access contiguous memories. This low-level optimization leverages instructions that are part of Intel x86 segmentation – instructions optimized to access contiguous memories. Modify Chrome's access of the contiguous JavaScript heap from using standard x86 load/store instructions to instructions that can use segmentation instructions. This requires two

changes --- changes in Chrome's C++ code and changes in Chrome's JITtedcode. For Chrome's C++ code, clang provides annotations in C/C++ that can be used to modify code to leverage segmentation instructions. For Chrome's JIT code, you can modify Chrome's JIT compiler directly to use segmentation instructions. Modify Chrome to implement this optimization and measure the performance difference.
Pre-reading: Segue & ColorGuard: Optimizing SFI Performance and Scalability on Modern x86
Skills: Comfortable with C++ and clang. Experience working with Chrome will help but is not necessary.
Level of difficulty: Moderate/Hard.

- Implement a version of RLBox's tainted type in Rust. As you will read in the class, the RLBox framework provides "tainted" types to safely handle untrusted data coming from a particular sandbox's heap. When using a tainted integer, RLBox allows arithmetic on the tainted integer but does not allow the tainted integer to be used in place of a regular integer. When dereferencing a tainted pointer, RLBox automatically checks that the pointer being dereferenced is within the sandbox's heap. Recreate this behavior in Rust using the following setup. Use a Rust "Vec<u8>" to represent the sandbox heap. Then provide APIs to access this Rust Vec<u8> that return data wrapped in a new tainted type that you create in Rust. This tainted type must ensure that the tainted data being returned cannot accidentally be misused but continue to allow simple safe operations like arithmetic on a tainted int.
  Pre-reading: RLBox: Retrofitting Fine Grain Isolation in the Firefox Renderer
  Skills: Comfortable with Rust.
  Level of difficulty: Moderate/Hard.
- Develop a scheme that compiler backends can follow to ensure the emitted instructions would simply abort in the presence of a 1-bit flip in any one of instructions. Compilers today emit assembly that when modified by Rowhammer can be used to bypass security checks. For example, assume that we have a load instruction "lw a1, 0(a2)". a1 and a2 are registers. This instruction loads from the memory location a2 and stores the loaded value in register a1. Assuming with Rowhammer, you could flip a single bit in this instruction which causes the instruction to be parsed differently. The instruction could now become "lw a1, 0(a4)" or "lw a1, 0(a8)". Then a simple encoding scheme that keeps this safe would be to emit "lw a1, 0(a2)" only after setting registers a4 and a8 to zero. This is because execution of "lw a1, 0(a4)" or "lw a1, 0(a8)" would fault as it in-effect dereferences a null pointer. You can modify the tiny C compiler to use this new encoding scheme. This project can target the RISC-V or ARM instruction encoding.
  Pre-reading: Exploiting the DRAM rowhammer bug to gain kernel privileges
  Skills: Comfortable with RISC-V or ARM. Familiarity with the tiny C compiler would help as well.
  Level of difficulty: Moderate/Hard.
- Implement HFI in QEMU. HFI is an instruction set extension designed for CPUs so they can allow applications to isolate components. The implementation can be in x86, ARM or RISC-V.
  Pre-reading: Going Beyond the Limits of SFI: Flexible and Secure Hardware-Assisted In-Process Isolation with HFI
  Skills: Comfortable with the chosen CPU architecture. Familiarity with QEMU would help as well.
  Level of difficulty: Easy.

- Sandboxing runtimes isolate components of an application to a subset of the address space by applying bounds checks on all memory accesses. Construct a sandboxing runtime that isolates the components of an application by modifying the page tables entries of an application dynamically. Concretely, these page table modifications will ensure that a component is isolated by restricting all page entries to only point to a fixed range of the virtual address space. For example, component 1 may be restricted to the space 8GB to 16GB while component 2 is restricted from 16GB to 32GB etc. Modifying the page entries of an application can be performed using libraries like PTEditor. One subtle point to note here is that for correct functionality, you must ensure each component has its own allocator and each allocator only allocates memory within the permitted space. You can do this either by using a custom allocator that you modify like dlmalloc, or you can do this by building on an existing sandboxing tool which already provides a runtime for this --- for instance wasm2c or LFI. If you do build on an existing runtime, you can disable any bounds checks these tools add.
  Pre-reading: Lightweight Fault Isolation: Practical, Efficient, and Secure Software Sandboxing
  Skills: Comfortable with C/C++ and the linux kernel. Familiarity with sandboxing will also help.
  Level of difficulty: Moderate
- Reproduce the results of Arabica, a tool that isolates native libraries accessed by Java programs through the JNI interface.
  Pre-reading: Arabica: JVM-Portable Sandboxing of Java's Native Libraries
  Skills: Comfortable with Java and C++. Familiarity with JVM implementation would also help
  Level of difficulty: Moderate/Hard
- RLBox is a sandboxing framework that allows applications to sandbox libraries with WebAssembly (wasm) which isolates the effects of the library from the application. Modify the RLBox sandboxing framework to concurrently execute each function call in a separate process as well. You can then use this mechanism to figure out if sandboxing with Wasm is faster or slower than sandboxing with processes for each function call to the sandboxed library. Modify RLBox to dynamically choose between the two options depending on which approach is faster for each function call. Compare the performance to stock RLBox.
  Pre-reading: RLBox: Retrofitting Fine Grain Isolation in the Firefox Renderer
  Skills: Comfortable with C++.
  Level of difficulty: Easy/Moderate

If you're still having difficulties picking a project, please talk to the instructor to brainstorm.

12. Policy on Academic Accommodations

The university is committed to creating an accessible and inclusive learning environment consistent with university policy and federal and state law. Please let me know if you experience any barriers to learning so I can work with you to ensure you have equal opportunity to participate fully in this course. If you are a student with a disability, or think you may have a disability, and need accommodation please contact Disability and Access (D&A). Please refer to D&A's website for contact and more information: http://diversity.utexas.edu/disability/. If you are already registered with D&A , please deliver your Accommodation Letter to me as early as possible in the semester so we can discuss your approved accommodation and needs in this course.

13. Academic Integrity

Recall the Student Honor Code: "As a student of The University of Texas at Austin, I shall abide by the core values of the University and uphold academic integrity."

Students who violate University rules on academic dishonesty are subject to disciplinary penalties, including the possibility of failure in the course and/or dismissal from the University. Since such dishonesty harms the individual, all students, and the integrity of the University, policies on academic dishonesty will be strictly enforced. For further information, please visit the Student Conduct and Academic Integrity Website.

To detect instances of academic integrity violations in programming assignments we may use 3rd party software.

14. Artificial intelligence

The use of artificial intelligence tools (such as ChatGPT) are allowed, except for the in-class quizzes. However, it is your responsibility to validate any information you receive from them. This applies to all content in the presentations. Any errors due to AI tools will be graded as if they are your errors.

15. Religious holy days

Religion (or lack thereof) is an important part of who we are. If a holy day observed by your religion falls during the semester and you require accommodation due to that, please let me know as soon as possible. Email is an acceptable form of communication. To guarantee accommodation around presentations or other big deadlines, I will need notice of at least two weeks. If you are unable (or forget!) to provide that notice, please contact me anyway in case I can still accommodate you.

University-required language: A student who is absent from an examination or cannot meet an assignment deadline due to the observance of a religious holy day may take the exam on an alternate day or submit the assignment up to 24 hours late without penalty, ONLY if proper notice of the planned absence has been given. Notice must be given at least 14 days prior to the classes which will be missed. For religious holy days that fall within the first two weeks of the semester, notice should be given on the first day of the semester. Notice must be personally delivered to the instructor and signed and dated by the instructor, or sent certified mail. Email notification will be accepted if received, but a student submitting email notification must receive email confirmation from the instructor.

16. Class Recordings

While there are no plans to record this class, this may change, and classes may be recorded. Class recordings, if provided, are reserved only for students in this class for educational purposes and are protected under FERPA. The recordings should not be shared outside the class in any form. Violation of this restriction by a student could lead to Student Misconduct proceedings.

17. Class Calendar

Date	Theme	Class contents	Presenters
Tue, Jan 13		**No class today**	
Thu, Jan 15	Introduction	Introduction, syllabus etc. How to read a paper S. Keshav (2007) Paper discussion assignments **Memory safety:** - Chapter 1 from How Memory Safety Violations Enable Exploitation of Programs - M. Payer (2018) - Sections 30.1 and 30.2 Low-Level Software Security by Example - Úlfar Erlingsson, Yves Younan, and Frank Piessens (2010) **On giving talks:** (An Opinionated Talk) On Preparing Good Talks – Ranjit Jhala (2018)	Instructor
Tue, Jan 20	Memory safety attacks	Smashing the Stack for Fun and Profit - Aleph One (1996) – reformatted in 2017 Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns - Pincus, Baker (2004)	<Student sign up 1> <Student sign up 2>
Thu, Jan 22		Exploiting Format String Vulnerabilities – scut (2001) Basic Integer Overflows – blexim (2002) Understanding Integer Overflow in C/C++ - Will Dietz, Peng Li, John Regehr, Vikram Adve (2015)	<Student sign up 1> <Student sign up 2>
Tue, Jan 27	Memory safety defenses	SoftBound: Highly compatible and complete spatial memory safety for C - Santosh Nagarakatte, Jianzhou Zhao, Milo M. K. Martin, Steve Zdancewic (2009) CETS: compiler enforced temporal safety for C - Santosh Nagarakatte, Jianzhou Zhao, Milo M.K. Martin, Steve Zdancewic (2010) Updated writeup (2024): Full Spatial and Temporal Memory Safety for C - Santosh Nagarakatte (2024)	<Student sign up 1> <Student sign up 2>
Thu, Jan 29		AddressSanitizer: A Fast Address Sanity Checker - Konstantin Serebryany, Derek Bruening, Alexander Potapenko, Dmitry Vyukov (2012)	<Student sign up 1> <Student sign up 2>

		MemorySanitizer: fast detector of uninitialized memory use in C++ - Evgeniy Stepanov, Konstantin Serebryany (2015) UndefinedBehaviorSanitizer - Clang 22.0.0 git documentation ThreadSanitizer – data race detection in practice - Konstantin Serebryany, Timur Iskhodzhanov (2009)	
Tue, Feb 03	Control-flow attacks and defenses	The advanced return-into-libc exploits - Nergal (2001) Return-Oriented Programming: Systems, Languages, and Applications - R. Roemer, E. Buchanan, H. Shacham and S. Savage (2012)	<Student sign up 1> <Student sign up 2>
Thu, Feb 05		**(Skip sections 5 and 6)** Control-Flow Integrity: Principles, Implementations, and Applications - M. Abadi, M. Budiu, Úlfar Erlingsson, and J. Ligatti (2009) Security Analysis of Processor Instruction Set Architecture for Enforcing Control-Flow Integrity - Vedvyas Shanbhogue, Deepak Gupta, Ravi Sahita (2019) Control-Flow Bending: On the Effectiveness of Control-Flow Integrity - Nicolas Carlini, Antonio Barresi, Mathias Payer, David Wagner, Thomas R. Gross (2015)	Instructor
Tue, Feb 10	Memory safety attacks	**Note:** Project selection deadline. Ungraded short writeup due. Hacking blind - Andrea Bittau, Adam Belay, Ali Mashtizadeh, David Mazieres, Dan Boneh (2014)	<Student sign up 1> <Student sign up 2>
Thu, Feb 12		TBD: Analysis of a real attack	<Student sign up 1> <Student sign up 2>
Tue, Feb 17	Bug finding	**Mandatory visiting scholar presentation attendance – Feb 17th, 11am, GDC 2.216** KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs - Cristian Cadar, Daniel Dunbar, Dawson Engler (2008)	<Student sign up 1> <Student sign up 2>

		Sys: A {Static/Symbolic} Tool for Finding Good Bugs in Good (Browser) Code - Fraser Brown, Deian Stefan, Dawson Engler (2020)	
Thu, Feb 19		AFL fuzzer - Design - Docs - Follow up: AFL++: Combining Incremental Steps of Fuzzing Research - Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, Marc Heuse (2020) Evaluating fuzz testing - George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, Michael Hicks (2018)	<Student sign up 1> <Student sign up 2>
Tue, Feb 24	Sandboxing	Evaluating SFI for a CISC Architecture - Stephen McCamant, Greg Morrisett (2006) Bringing the web up to speed with WebAssembly - Andreas Haas, Andreas Rossberg, Derek L. Schuf, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, JF Bastien (2017)	<Student sign up 1> <Student sign up 2>
Thu, Feb 26		RLBox: Retrofitting Fine Grain Isolation in the Firefox Renderer - Shravan Narayan, Craig Disselkoen, Tal Garfinkel, Nathan Froyd, Eric Rahm, Sorin Lerner, Hovav Shacham, Deian Stefan (2020)	Instructor
Tue, Mar 03	Secure Browser architecture	The Security Architecture of the Chromium Browser - Adam Barth, Collin Jackson, Charles Reis, Google Chrome Team (2008) Site isolation: Process separation for web sites within the browser - Charles Reis, Alexander Moshchuk, and Nasko Oskov (2019)	<Student sign up 1> <Student sign up 2>
Thu, Mar 05	JIT Security	The Art of Exploitation: Attacking JavaScript Engines - saelo (2016)	<Student sign up 1> <Student sign up 2>
Tue, Mar 10		The Art of Exploitation: Compile Your Own Type Confusion: Exploiting Logic Bugs in JavaScript JIT Engines - saelo (2019)	<Student sign up 1> <Student sign up 2>

Thu, Mar 12		Fuzzilli: Fuzzing for JavaScript JIT Compiler Vulnerabilities - Samuel Groß, Simon Koch, Lukas Bernhard, Thorsten Holz, Martin Johns (2023) Cover some example bugs uncovered by Fuzzilli. This link also has extra resources for Fuzzilli.	<Student sign up 1> <Student sign up 2>
Tue, Mar 17		**Spring break, no class**	
Thu, Mar 19		**Spring break, no class**	
Tue, Mar 24	Side-channel attacks	**Introduction and background** **Side-channels:** Sections 1 to 4 from Transient-Execution Attacks: A Computer Architect Perspective – Luís Fiolhais, Leonel Sousa (2023)	Instructor
Thu, Mar 26		Spectre attacks: Exploiting speculative execution - Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom (2018) **For the below paper. Skip section 4.2 to 4.8** A Systematic Evaluation of Transient Execution Attacks and Defenses - Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss (2019)	Instructor
Tue, Mar 31		Exploiting the DRAM rowhammer bug to gain kernel privileges - Mark Seaborn with contributions by Thomas Dullien (2015)	<Student sign up 1> <Student sign up 2>
Thu, Apr 02	Hardware-security	SoK: Hardware-supported Trusted Execution Environments - Moritz Schneider, Ramya Jayaram Masti, Shweta Shinde, Srdjan Capkun, Ronald Perez (2022)	<Student sign up 1> <Student sign up 2>
Tue, Apr 07		Memory Tagging: A Memory Efficient Design – Aditi Partap, Dan Boneh (2022)	Instructor

		ARM MTE Performance in Practice (link coming) – Taehyun Noh, Yingchen Wang, Tal Garfinkel, Mahesh Madhav, Daniel Moghimi, Mattan Erez, Shravan Narayan (2026)	
Thu, Apr 09	Web Security	Tor: The Second-Generation Onion Router - Roger Dingledine, Nick Mathewson, Paul Syverson (2004) Top changes in Tor since the 2004 design paper (Part 1) – Nick Mathewson (2012) Top changes in Tor since the 2004 design paper (Part 2) – Nick Mathewson (2012) Top changes in Tor since the 2004 design paper (Part 3) – Steven Murdoch (2012)	<Student sign up 1> <Student sign up 2>
Tue, Apr 14		TBD – CSRF etc	
Thu, Apr 16		Buffer class. Maybe filled with another paper.	
Tue, Apr 21		**Final presentation for projects, part 1**	
Thu, Apr 23		**Final presentation for projects, part 2**	