CS380S: Graduate Computer Security

1. Course Contact

Dr. Shravan Narayan
Classes: 2:00pm to 3:30pm at GDC 2.210 on Tuesday and Thursday.
Office Hours: At GDC 6.430, 3:30pm to 4:30pm on Tuesday and 4:00pm to 4:45pm on Wednesday.
(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: Aashish Gottipati agottipati@utexas.edu
Canvas: https://utexas.instructure.com/courses/1402120

(Syllabus last updated: Oct 14th, 2024)

2. Course requirements

It is expected that all students have taken CS 361s or equivalent. This is not a hard requirement, but students without this experience 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 have some familiarity 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)
- Paper writeups – 20% (twice a week)
- Paper presentations – 20% (once or twice a semester)
- Class project midterm writeup – 15% (once on Oct 26th)
- Class project final presentation – 15% (once on Dec 3rd or Dec 5th)
- Class project final writeup – 20% (once on Dec 6th)
- **Bonus:** class participation – 5% (throughout the semester)

Each of these are broken down below.

4. Class attendance

Attendance in this class is required starting week 2 (Sept 3rd). See "Skipping attendance or paper writeups" 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.

5. Paper writeups

The list of papers that will be discussed in the course are listed in the calendar below. Starting week 2 (Sept 3rd), we will discuss one or two research papers each class. Students are expected to read the papers prior to attending class and submit paper writeups prior to each class. A paper writeup is a short summary of each paper along with pros, cons, discussion points, and specific questions about the paper. These may not be submitted late unless you have a UT approved reason (sickness, emergency etc.). See "Skipping attendance or paper writeups" below if you need to skip classes.

These writeups may be submitted on Gradescope and are due at 2pm on Tuesday and Thursday (If Gradescope does not cooperate, this may change to submission in physical form during the class).

Note that some papers may be modified based on student interest/new security news etc. Papers that are replaced will be done so with one week's notice.

6. Skipping attendance or paper writeups

This class has a total of 4 skips for paper writeups and 2 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.

Note that each paper skip only counts for one paper, not for all papers in a class. If you want to skip both papers in a class this counts as two paper skips.

If you need to skip classes or paper writeups beyond the 4 paper skips + 2 class skips, please follow standard UT guidelines for excused absences.

Suggestion: try to reserve these skips for emergencies.

7. Paper presentations

Each student is expected to present one paper (with powerpoint or other suitable presentation aides) starting Sept 3rd. The presentation is expected to be 30 mins with of content with 10 mins of Q&A during or after the presentation. Presenters should be well versed in both the paper as well as background material. Bonus points for including some background material in the presentation.

Signups for paper presentations here.

Possible modifications: We may modify paper presentations to have two presenters working as a team, depending on the size of the class. We may also require students to present up to papers in the class.

8. Project

The goal for the course project is to first develop or build on 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 can be done individually or in groups of up to 3. If you are a group, you 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 the area of systems security and discuss this with me. Alternately, you can build on, or implement one of the existing project ideas that I will share in class.

Project group and topics selection should be complete by 25th September 2024, although you are welcome to select your project earlier as well. 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 execute an easy project well will score the same as students who pick an ambitious project but only have partial success.

9. Class project midterm writeup

Project groups are expected to provide a two-page single-spaced writeup on the progress of their project on 26th October 2024 via Canvas.

10. Class project final writeup and presentation

Project groups are expected to provide a five-page single-spaced writeup on their project on 6th December via Canvas as well as a 15 minute presentation in class of their project on 3rd December and 5th December. The exact day of the presentation will be decided in class during project selection.

11. Class participation

This class is powered by discussion and thus students should participate in discussions. It is thus very important you read the papers and submit paper writeups so you can take part in the class discussions. The discussions and Q&A will be directed to the presenter of the paper. Students who make a point of participating in discussions are eligible for a bonus score of up to 5% of the course grade.

12. 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 the area of 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 you could sandbox:
    - 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.
- Modify wasm2c to trace all memory accesses, record it to file, and calculate the diff of two different memory traces of a program. Wasm2c is a compiler that compiles Wasm binaries to native code by first compiling WebAssembly files to C code, and then compiling C code with a standard C compiler. You can modify Wasm2c to record every memory load and store operation to the WebAssembly heap, and save this to a file. By running this modified Wasm2c to two different executions of a program, you can identify memory accesses that differ between two traces of a program. Such tools allow developers to identify where program executions differ for different inputs. Build this tool that records two traces and then diffs the two traces, and identify the memory access that differs between the two traces.
  Skills: Comfortable with C++.
  Level of difficulty: Easy/Moderate.
- 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.

13. 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.

14. 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.

15. Artificial intelligence

The use of artificial intelligence tools (such as ChatGPT) in this class is strictly prohibited. This includes using AI to generate ideas, outline an approach, answer questions, solve problems, or create original language. All work in this course must be your own or created in group work, where allowed.

16. 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 accommodations due to that, please let me know as soon as possible. Email is an acceptable form of communication. In order 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.

17. 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.

18. Class Calendar

Date	Theme	Class contents	Presenters
Tuesday, 27 Aug 2024	Introduction	Introduction, syllabus etc. How to read a paper S. Keshav (2007) Paper discussion assignments **Time permitting:** Discuss material from next class	Instructor
Thursday, 29 Aug 2024	Background	**Instructions: Background catchup. No paper writeup for this week** **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) **Side-channels:** Sections 1 to 4 from Transient-Execution Attacks: A Computer Architect Perspective – Luís Fiolhais, Leonel Sousa (2023)	Instructor

		On giving talks: (An Opinionated Talk) On Preparing Good Talks – Ranjit Jhala (2018)	
Tuesday, 3 Sep 2024	Side-channel attacks & defenses	**Instructions: paper writeups start this week and must be submitted on Canvas** **For the below paper. Skip section 4.2 to 4.8** **Paper 1:** 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) **AND** **Paper 2:** Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86 - Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, David Kohlbrenner (2022)	Paper 1: <fill in> Paper 2: <fill in>
Thursday, 5 Sep 2024		**Paper 1:** Opening Pandora's Box: A Systematic Study of New Ways Microarchitecture Can Leak Private Data - Jose Rodrigo Sanchez Vicarte, Pradyumna Shome, Nandeeka Nayak, Caroline Trippel, Adam Morrison, David Kohlbrenner, Christopher W. Fletcher (2021)	Paper 1: <fill in>
Tuesday, 10 Sep 2024		**Paper 1:** Trusted browsers for uncertain times - David Kohlbrenner, Hovav Shacham (2016) **AND** **Paper 2:** Dynamic Process Isolation - Martin Schwarzl, Pietro Borrello, Andreas Kogler, Kenton Varda, Thomas Schuster, Daniel Gruss, Michael Schwarz (2021)	Paper 1: <fill in> Paper 2: <fill in>
Thursday, 12 Sep 2024		**Paper 1:** Ileakage: Browser-based timerless speculative execution attacks on apple devices - J Kim, S van Schaik, D Genkin, Y Yarom (2023) **AND** **Paper 2:** GoFetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers - Boru Chen, Yingchen Wang, Pradyumna Shome, Christopher W Fletcher, David Kohlbrenner, Riccardo Paccagnella, Daniel Genkin (2024)	Instructor or guest lecture.

Tuesday, 17 Sep 2024	Memory safety	**Paper 1a:** Smashing the Stack for Fun and Profit - Aleph One (1996) – reformatted in 2017 **Paper 1b:** Exploiting Format String Vulnerabilities – scut (2001) **Paper 2a:** The advanced return-into-libc exploits - Nergal (2001) **Paper 2b:** Return-Oriented Programming: Systems, Languages, and Applications - R. Roemer, E. Buchanan, H. Shacham and S. Savage (2012)	Paper 1a and 1b: <fill in> Paper 2a and 2b: <fill in>
Thursday, 19 Sep 2024		**Paper 1:** Evaluating fuzz testing - George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, Michael Hicks (2018)	Paper 1: <fill in>
Tuesday, 24 Sep 2024		**Reserved for project discussions** **Project description due by 25th September: 1 paragraph writeup on canvas** **Paper 1:** Exploiting the DRAM rowhammer bug to gain kernel privileges - Mark Seaborn with contributions by Thomas Dullien (2015)	Paper 1: Instructor
Thursday, 26 Sep 2024		**Paper 1a:** AddressSanitizer: A Fast Address Sanity Checker - Konstantin Serebryany, Derek Bruening, Alexander Potapenko, Dmitry Vyukov (2012) **Note: the presentation of this is premade in the link. The challenge is to understand what this is doing in a complex application – Google Chrome** **Paper 1b:** MiraclePtr - the UaF slayer - Keishi Hattori, Bartek Nowierski (2022) Additional resources for MiraclePtr	Paper 1a and 1b: <fill in>
Tuesday, 1 Oct 2024		**Paper 1:** SoftBound: Highly compatible and complete spatial memory safety for C - Santosh Nagarakatte, Jianzhou Zhao, Milo M. K. Martin, Steve Zdancewic (2009) **AND** **Paper 2:** CETS: compiler enforced temporal safety for C - Santosh Nagarakatte, Jianzhou Zhao, Milo M.K. Martin, Steve Zdancewic (2010)	Paper 1: <fill in> Paper 2: <fill in>

Thursday, 3 Oct 2024		**Paper 1:** Sys: A {Static/Symbolic} Tool for Finding Good Bugs in Good (Browser) Code - Fraser Brown, Deian Stefan, Dawson Engler (2020)	Paper 1: <fill in>
Tuesday, 8 Oct 2024	Probabilistic memory safety	**Paper 1a:** Memory Tagging: A Memory Efficient Design - Aditi Partap, Dan Boneh (2022) **Paper 1b:** Security analysis of memory tagging - J. Bialek, K. Johnson, M. Miller, and T. Chen (2020)	Paper 1a and 1b: <fill in>
Thursday, 10 Oct 2024		**Reserved some time for ad-hoc topics, incomplete prior discussions and project discussions** **Paper 1:** Hacking blind - Andrea Bittau, Adam Belay, Ali Mashtizadeh, David Mazieres, Dan Boneh (2014)	Paper 1: <fill in>
Tuesday, 15 Oct 2024	Control flow integrity	**Paper 1:** Control-Flow Integrity: Principles, Implementations, and Applications - M. Abadi, M. Budiu, Úlfar Erlingsson, and J. Ligatti (2009) **AND** **Paper 2:** Security Analysis of Processor Instruction Set Architecture for Enforcing Control-Flow Integrity - Vedvyas Shanbhogue, Deepak Gupta, Ravi Sahita (2019)	Paper 1: <fill in> Paper 2: <fill in>
Thursday, 17 Oct 2024		**Paper 1:** Control-Flow Bending: On the Effectiveness of Control-Flow Integrity - Nicolas Carlini, Antonio Barresi, Mathias Payer, David Wagner, Thomas R. Gross (2015)	Paper 1: <fill in>
Tuesday, 22 Oct 2024	Coarse-grain defenses	**Paper 1:** Lightweight Fault Isolation: Practical, Efficient, and Secure Software Sandboxing - Zachary Yedidia (2024) **AND** **Paper 2:** 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)	Paper 1: <fill in> Paper 2: <fill in>
Thursday, 24 Oct 2024		**Paper 1:** The Security Architecture of the Chromium Browser - Adam Barth, Collin Jackson, Charles Reis, Google Chrome Team (2008) **Project midterm writeup due by 26th October on canvas**	Paper 1: <fill in>

Tuesday, 29 Oct 2024	JIT Security	**Paper 1:** The Art of Exploitation: Attacking JavaScript Engines - saelo (2016) **Paper 2:** The Art of Exploitation: Compile Your Own Type Confusion: Exploiting Logic Bugs in JavaScript JIT Engines - saelo (2019)	Paper 1: <fill in> Paper 2: <fill in>
Thursday, 31 Oct 2024	Formal verification	**Paper 1:** Icarus: Trustworthy Just-In-Time Compilers with Symbolic Meta-Execution - Naomi Smith, Abhishek Sharma, John Renner, David Thien, Fraser Brown, Hovav Shacham, Ranjit Jhala, Deian Stefan (2024)	Paper 1: <fill in>
Tuesday, 5 Nov 2024	Hardware defenses	**Paper 1:** Keystone: An open framework for architecting trusted execution environments - Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, Dawn Song (2020) **AND** **Paper 2:** Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems - Yuanzhong Xu, Weidong Cui, Marcus Peinado (2015)	Paper 1: <fill in> Paper 2: <fill in>
Thursday, 7 Nov 2024		**Paper 1:** Going Beyond the Limits of SFI: Flexible and Secure Hardware-Assisted In-Process Isolation with HFI - Shravan Narayan, Tal Garfinkel, Mohammadkazem Taram, Joey Rudek, Daniel Moghimi, Evan Johnson, Chris Fallin, Anjo Vahldiek-Oberwagner, Michael LeMay, Ravi Sahita, Dean Tullsen, Deian Stefan (2023)	Paper 1: <fill in>
Tuesday, 12 Nov 2024	Redesigning OSes	**Paper 1:** Multiprogramming a 64 kB Computer Safely and Efficiently - Amit Levy, Bradford Campbell, Branden Ghena, Daniel B Giffin, Pat Pannuto, Prabal Dutta, Philip Levis (2017) **Paper 2:** Simple and Precise Static Analysis of Untrusted Linux Kernel Extensions - Elazar Gershuni, Nadav Amit, Arie Gurfinkel, Nina Narodytska, Jorge A. Navas, Noam Rinetzky, Leonid Ryzhyk, Mooly Sagiv (2019)	Paper 1: <fill in> Paper 2: <fill in>
Thursday, 14 Nov 2024	Protocol security	**Paper 1:** RADIUS/UDP Considered Harmful - Sharon Goldberg, Miro Haller, Nadia Heninger, Mike Milano, Dan Shumow, Marc Stevens, Adam Suhl (2024)	Paper 1: <fill in>
Tuesday, 19 Nov 2024	Ecosystem security	**Paper 1:** Click Trajectories: End-to-End Analysis of the Spam Value Chain - Kirill Levchenko, Andreas Pitsillidis, Neha Chachra, Brandon Enright, Márk Félegyházi, Chris Grier,	Paper 1: <fill in>

		Tristan Halvorson, Chris Kanich, Christian Kreibich, He Liu, Damon McCoy, Nicholas Weaver, Vern Paxson, Geoffrey M. Voelker, Stefan Savage (2011) **AND** **Paper 2:** Re: CAPTCHAs – Understanding CAPTCHA-Solving Services in an Economic Context - Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M. Voelker and Stefan Savage (2010)	Paper 2: <fill in>
Thursday, 21 Nov 2024	Usable security	**Paper 1:** Alice in warningland: a {Large-Scale} field study of browser security warning effectiveness - Devdatta Akhawe, Adrienne Porter Felt (2013)	Paper 1: <fill in>
Tuesday, 26 Nov 2024		**Thanksgiving break, no class**	NA
Thursday, 28 Nov 2024		**Thanksgiving break, no class**	NA
Tuesday, 3 Dec 2024	Project presentation	**Final presentation for projects, part 1**	Group 1: <fill in> Group 2: <fill in> …
Thursday, 5 Dec 2024		**Final presentation for projects, part 2** **Project midterm writeup due by 6th December on canvas**	Group 1: <fill in> Group 2: <fill in> …