CS395T: Securing real-world systems

1. Course Contact

Dr. Shravan Narayan
Office Hours: 2:30 to 3:30 at GDC 6.430 on Tuesday and 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)

(Syllabus last Updated: Aug 21st, 2023)

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.

3. Grading / Attendance

Grade breakdown is as follows:

- Class attendance – 10%
- Paper writeups – 30%
- Paper presentations – 10%
- Class project midterm presentation and writeup – 20%
- Class project final presentation and writeup – 30%
- Bonus: class participation – 5%

Each of these are broken down below.

4. Class attendance

Attendance in this class is required. 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 allowed.

Note one class on September 7th is expected to be remote over zoom. The link for this will be posted on Canvas.

5. Paper writeups

The list of papers that will be discussed in the course are listed in the calendar below. Each class, we will discuss one (or in some cases two) research paper(s). Students are expected to read the paper 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, and discussion points. 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.

If multiple papers are listed for a given class, submit a single writeup summarizing both papers.

These writeups may be submitted on Gradescope and are due at 3:30pm 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 four skips that you can use either for classes or paper writeups without any explanation/email/note to me. These skips can be used for emergencies, sickness etc.

What does this mean?

- You didn’t do the writing assignment (in time): use up a skip
- You can’t show up to class: use up a skip

If you need to skip classes or paper writeups beyond the four skips, please follow standard UT guidelines for excused absences. Suggestion: try to reserve this skips for emergencies.

7. Paper presentations

Each student will be expected to lead one or two paper discussions in class. On the 24th and 29th of August, I will lead a discussion as an example for students. Starting the 31st of August, students will lead discussions. Each paper will have either one or two discussion leads, depending on the size of the class. I will lead the discussion on 14th September as this is a slightly tricky paper to fully put in context.

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 selection and brainstorming will occur in class on 28 September 2023, 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 presentation

Project groups are expected to provide a two-page single-spaced writeup on the progress of their project as well as a 10 to 15 minute presentation in class of their project on 24 October 2023 or 26 October 2023. The exact day of the presentation will be decided in class during project selection.

10. Class project final presentation

Project groups are expected to provide a five-page single-spaced writeup on the progress of their project as well as a 15 to 20 minute presentation in class of their project on 28 Nov 2023 or 30 Nov 2023. 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 will either be in open free-form discussions or Q&A style discussions where each student will be given the chance to answer questions raised by the paper discussion lead. 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 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

  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. In particular, clang provides annotations in C/C++ that can be used to modify code to leverage segmentation instructions. Using this, modify Chrome to implement this optimization and measure the performance difference.
  Skills: Comfortable with C++ and clang. Experience working with Chrome will help but is not necessary.
  Level of difficulty: Moderate/Hard.
- In the style of SFI, modify Firefox to ensure all memory accesses to its JavaScript heap are masked to remain in the JavaScript heap. However, unlike traditional SFI tools, the Firefox

browser creates multiple tiny discontinuous arenas for its JavaScript heap. Thus, it is not easy to check if a pointer being dereferenced is outside the current arena. To adjust this, modify the arena to be aligned to its size, and store the size in the style of low-fat pointers (in the top bits of the pointer). Then during pointer arithmetic, check that a pointer is not modified to point to a location outside the arena.
Skills: Comfortable with C++. Experience working with Firefox will help but is not necessary.
Level of difficulty: 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.
  Skills: Comfortable with Rust.
  Level of difficulty: Moderate/Hard.
- Measure the cost of each of clang's sanitizers. Clang provides a number of sanitizers to help identify memory safety errors as well as other programming errors. These include the address sanitizer (which we will read about in class), the undefined behavior sanitizer, memory sanitizer etc. Execute each of these sanitizers on standard benchmarks and measure the performance. Reproduce the experimental results from FloatZone – an optimization for sanitizers recently developed by researchers. FloatZone makes its modified clang sanitizers available open source, so you would need to build their modified Clang compiler and run standard benchmark suites like SPEC.
  Required experience: Comfortable using Clang and building C/C++ code.
  Level of difficulty: Easy.
- Measure the cost of clang's CFI in standard benchmarks and appliations. Clang currently provides a few simple compiler flags to compile an application with different forms of CFI. The configuration flags of CFI include "Forward-Edge CFI for Virtual Calls", "Bad Cast Checking", "Non-Virtual Member Function Call Checking", "Indirect Function Call Checking", "Member Function Pointer Call Checking". Measure the performance cost of different combinations of these CFI checks both in terms of runtime overheads and memory overheads on both standard benchmark suites like SPEC as well as one large application like a web browser or web server.
  Required experience: Comfortable using Clang.
  Level of difficulty: Easy.
- 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.
Skills: Comfortable with RISC-V or ARM. Familiarity with the tiny C compiler would help as well.
Level of difficulty: Moderate/Hard.

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

By UT Austin policy, you must notify me of your pending absence for a religious holy day as far in advance as possible of the date of observance. If you must miss a class, an examination, a work assignment, or a project in order to observe a religious holy day, you will be given an opportunity to complete the missed work within a reasonable time after the absence.

17. Class Recordings

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
Tuesday, 22 Aug 2023	Introduction	Welcome, introduction. How to read a paper S. Keshav (2007) Time permitting: Paper discussion assignment
Thursday, 24 Aug 2023		**Instructions: no paper writeup for this week** Quick recap from CS380s: How Memory Safety Violations Enable Exploitation of Programs - M. Payer (2018) Paper discussion assignment
Tuesday, 29 Aug 2023	Privilege separation	Preventing Privilege Escalation - N Provos, M Friedl, P Honeyman (2003)
Thursday, 31 Aug 2023		Site isolation: Process separation for web sites within the browser - C Reis, A Moshchuk, N Oskov (2019)
Tuesday, 5 Sep 2023	Memory Safety	SoftBound: Highly compatible and complete spatial memory safety for C - Santosh Nagarakatte, Jianzhou Zhao, Milo M. K. Martin, Steve Zdancewic (2009)
Thursday, 7 Sep 2023	**(Class will be remote. Zoom link on canvas.)**	**Instructions: Class will be remote. Zoom link on canvas.**

		Low-Fat Pointers: Compact Encoding and Efficient Gate-Level Implementation of Fat Pointers for Spatial Safety and Capability-based Security - Albert Kwon, Udit Dhawan, Jonathan M. Smith, Thomas F. Knight, Andre DeHon (2013)
Tuesday, 12 Sep 2023	SFI	Native client: A sandbox for portable, untrusted x86 native code - Bennet Yee, David Sehr, Gregory Dardyk, J. Bradley Chen, Robert Muth, Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar (2009)
Thursday, 14 Sep 2023		Bringing the web up to speed with WebAssembly - Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, JF Bastien (2017) **Instructions:** Skip Sections 3 and 4. Paper discussion will be led by instructor.
Tuesday, 19 Sep 2023	Securing applications	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)
Thursday, 21 Sep 2023		Google Chrome V8 Protections mechanisms V8 pointer compression V8 Ubercage V8 Pointer Compression 2 V8 MiraclePointer - Samuel Groß et. al, Google (2021)
Tuesday, 26 Sep 2023	Potpourri	AddressSanitizer: A Fast Address Sanity Checker - Konstantin Serebryany, Derek Bruening, Alexander Potapenko, Dmitry Vyukov (2012)
Thursday, 28 Sep 2023		No paper discussion. Brainstorm projects ideas and form project groups.
Tuesday, 3 Oct 2023	Kernel / kernel-powered security	KSplit: Automating Device Driver Isolation - Yongzhe Huang, Vikram Narayanan, David Detweiler, Kaiming Huang, Gang Tan, Trent Jaeger, and Anton Burtsev (2022)
Thursday, 5 Oct 2023		Dune: Safe User-level Access to Privileged CPU Features - Adam Belay, Andrea Bittau, Ali Mashtizadeh, David Terei, David Mazières, and Christos Kozyrakis (2012)
Tuesday, 10 Oct 2023	Challenges of security hardening (CFI)	Control-Flow Integrity: Principles, Implementations, and Applications M. Abadi, M. Budiu, Úlfar Erlingsson, and J. Ligatti (2009) V8 CFI - Google (2021)

Thursday, 12 Oct 2023		Control-Flow Bending: On the Effectiveness of Control-Flow Integrity - Nicolas Carlini, Antonio Barresi, Mathias Payer, David Wagner, Thomas R. Gross (2015)
Tuesday, 17 Oct 2023	Challenges of security hardening (MPK)	Hodor: Intra-Process Isolation for High-Throughput Data Plane Libraries - Mohammad Hedayati, Spyridoula Gravani, Ethan Johnson, John Criswell, Michael L. Scott, Kai Shen and Mike Marty (2019)
Thursday, 19 Oct 2023		PKRU-safe: automatically locking down the heap between safe and unsafe languages - Paul Kirth, Mitchel Dickerson, Stephen Crane, Per Larsen, Adrian Dabrowski, David Gens, Yeoul Na, Stijn Volckaert, Michael Franz (2022)
Tuesday, 24 Oct 2023		Midpoint presentation for projects, part 1
Thursday, 26 Oct 2023		Midpoint presentation for projects, part 2
Tuesday, 31 Oct 2023	Spectre attacks	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)
Thursday, 2 Nov 2023		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)
Tuesday, 7 Nov 2023	Spectre-aware defenses	Swivel: Hardening WebAssembly against Spectre - Shravan Narayan, Craig Disselkoen, Daniel Moghimi, Sunjay Cauligi, Evan Johnson, Zhao Gang, Anjo Vahldiek-Oberwagner, Ravi Sahita, Hovav Shacham, Dean Tullsen, Deian Stefan (2021)
Thursday, 9 Nov 2023		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)
Tuesday, 14 Nov 2023	Unallocated classes	Will be used in case of any class cancellations / reorganizations. If unused, we will pick a paper based on class interest.
Thursday, 16 Nov 2023		Will be used in case of any class cancellations / reorganizations. If unused, we will pick a paper based on class interest.
Tuesday, 21 Nov 2023		Thanksgiving break, no class
Thursday, 23 Nov 2023		Thanksgiving break, no class

Tuesday, 28 Nov 2023	Project presentation	Final presentation for projects, part 1
Thursday, 30 Nov 2023		Final presentation for projects, part 2

Possible extra papers

1. Some thoughts on security after ten years of qmail 1.0 - Daniel J. Bernstein (2007)
2. Everything Old is New Again: Binary Security of WebAssembly - Daniel Lehmann, Johannes Kinder, Michael Pradel (2020)
3. Towards a verified range analysis for JavaScript JITs - Fraser Brown, John Renner, Andres Nötzli, Sorin Lerner, Hovav Shacham, Deian Stefan (2020)
4. Summary: MTE As Implemented (+ all three subparts linked) - Mark Brand, Project Zero (2023)
5. Language-independent sandboxing of just-in-time compilation and self-modifying code - J Ansel, P Marchenko, U Erlingsson, E Taylor, B Chen, DL Schuff, D Sehr, CL Biffle, B Yee (2011)
6. Language support for fast and reliable message-based communication in Singularity OS
7. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems