Name: Alex Groce

NAU login ID (e.g. adg326 for me):

SUMMARY: The mean grade was just over a 7. The median and mode were also 7. There were 4 students with perfect scores, and two who missed 6 questions (nobody missed more than 6). I'll probably end up curving a little once I get assignment 3 graded and all late assignment 2s handled.

1. Which of the following involves the "I" in "CIA"?

A Protecting a website against denial of service attacks
B Ensuring that doctors at a hospital, other than those treating you, cannot read your medical history
C Using very strong cryptography to discourage the NSA
D Preventing a student from changing the timestamp on an assignment submission

Correct answer: D (it's INTEGRITY of data, which D is the only instance of an attack on).
2 people missed this question (about 6%).

2. A general description of the high-level approach of many static analysis tools is:

A Parse; build annotated CFG; walk CFG to check properties; prioritize/compress results and present to user
B Compile; compute product of CFG and specification automata; determine if language intersection is empty
C Mark variable initializations as "def"s; mark variables in expressions as "use"s; check all defs are before uses
D Check for proper coding style; check for tainted user input; check for use of deprecated functions

Correct answer: A. B is off, since you don't need to compile at all, and you don't always use automata/language intersection (UNO does, but not everybody, and we didn't). C is just one example of a static analysis, not a high-level description of the algorithms. Similarly, D is a mishmash of some things some tools might do, not a high-level approach.
3 people missed this question (about 10%).

3. The end-goal of a security protocol is essentially to:

A Authenticate data received from an unreliable source over the Internet
B Force the use of sufficiently strong cryptography, even in embedded systems with low compute power
C Formulate a security policy that is agreed on by users of a software system
D Instill in various participants certain justified beliefs about other participants or data

The answer is D. It can't be A, because some security protocols don't even involve the Internet (the garage door opener example in Anderson). It can't be B, because many protocols don't really discuss the cryptography, and certainly don't force embedded systems to use very strong crypto. C was a popular answer, but again a protocol is a mechanism for doing something: you might analyze it with respect to a security policy, but it isn't itself a way to formulate such a policy. Instead if often grows out of a policy (also users may not agree). That leaves us D. See the Anderson chapter for how the BAN logic analyzes protocols in terms of beliefs. Authentication etc. are about *justified belief* that something is true. This is a really important point. For example, in a simple password protocol, the core idea is that your sole possession of a password entitles a system to believe that someone who can enter that password correctly is you. The Needham-Schroeder Public Key Protocol is about exchanging nonces so Alice and Bob can believe certain messages originate from each other. The logic is about what black box crypto (assumed correct) lets you believe as a result of the impossibility of decryption without keys. In a sense, protocols are *epistemological* devices using crypto mechanisms. Since secrets are involved, the epistemology is a bit tricky, of course!

28 people missed this, over 90% of the class.

4. A major cause for a large number of security vulnerabilities over the last 30 years is:

A Open source software is not developed using a highly-disciplined waterfall approach
B Low-level systems software is usually written in C, a language in which it is easy to write vulnerable code
C Modern computer architectures make the order of operations hard for programmers to predict
D Cloud computing is inherently insecure, because you do not have physical control of the machine

Since nobody missed question #4, I don't see any need to discuss it. You all know C is a powerful but very very dangerous tool. Good job, gang.

5. The following tools are all widely used in finding and exploiting software vulnerabilities:

A fuzzer like afl, SQL database for SQL injection attacks, and disassembler like IDA Pro

B fuzzer like afl, debugger like gdb, and natural language processing tool to scan for suspicious comments

C fuzzer like afl, debugger like gdb, and disassembler like IDA Pro

D fuzzer like afl, debugger like gdb, and code coverage tool to check test suite quality

Answer: C. This is from the *Bug Hunter's Diary* reading. afl is common to all answers, and while a SQL database itself is relevant, you don't really need one on hand to search for SQL injections, just an application that might talk to one (you don't even have to know for sure). Code coverage is not really used by much of anyone (except afl and other fuzzers that include their own) in typical security vulnerability searches; it's not even widely used in security research (vs. testing), and it wouldn't be that helpful. A disassembler for binaries is really nice, though, for the low level work, as the bug hunter's diary points out in the tool chapter.

16 of you missed this, over half the class.

6. What is the MIG-in-the-middle attack?

A Using a Russian fighter jet to intimidate an employee into giving away secret data

B An example of a World War II era cryptographic system

C An example of not using good timestamps to make sure that a message is fresh

D An instance of phishing to obtain a response to a challenge-response in authentication

Answer: D. It's in the Anderson protocol chapter. You get a challenge from an IFF system, beam it at an enemy fighter, and then watch their response and replay it, roughly. May not have happened, but nicely illustrates the man-in-the-middle concept, with bombs and fighter jets.

Only 4 people missed this one. I felt like giving the one who put A a point, but didn't.

7. SQL injection does NOT involve which of the following:

A Defense using static analysis of taint in the flow of data

B Enforcing of quotation rules to make sure that data is parsed correctly in database queries

C Exploitation of a timing or electromagnetic side channel

D Execution of code provided by an untrusted user

Answer: C. Static analysis can sometimes detect these (the paper on SQL injections we covered in class compares to such tools), so A cannot be the answer. Similarly, B is almost a description of the problem/attack, so is not the answer; and D *is* the high level description of the problem and many problems taint addresses (or attacks in general, really, if you let them get ahold of the program counter). So it's C, which has nothing to do with SQL injections at all.

A lot of people (11) missed this, to my surprise – over 1/3 of the class!

8. The end goal of most attempts to exploit software bugs is:

A To introduce nondeterminism into the behavior of a software system
B To corrupt the heap and cause a crash due to a null pointer
C To decrypt encrypted email messages
D To be able to take control of the instruction pointer/program counter

Answer: D. Nondeterminism itself is pretty useless. Crashes are more useful (denial of service) but still not what we call "exploitable" – and usually a crash just starts a new instance of the code under attack up. Nobody thought it was C. D is when you can execute arbitrary code, the most valuable kind of exploit.

Again, more people than expected (12, nearly 39%) missed this.

9. Symbolic execution is

A The simplest kind of software vulnerability analysis
B Not considered a useful technique in security, only in performance analysis
C Static analysis of the symbol table of a compiled program
D Executing a program with some values left partly undetermined (solved for)

Answer: D. Symbolic execution is D, it's certainly not simple (much more complex than fuzzing or static analysis), it's widely used in security but not in performance, and it has nothing to do with symbol tables (I'm pretty sure C was the answer of people who had no idea what symbolic execution might be and thought "hey, symbol!").

Yes, another D. 13 people missed this, almost 42% of the class.

10. American Fuzzy Lop works by

A Injecting new faults into a software system and seeing if it can detect them
B Mutating inputs that it has found to cover interesting code paths
C Watching network traffic for passwords and plaintext data
D Using valgrind to look for memory-safety problems

Answer: B

7 people missed this one, and I believe you all put A. Nope, afl doesn't mess up a program, it just sends it weird inputs.