CMSC 414 — Computer and Network Security

Buffer Overflows and Other Vulnerabilities

Dr. Michael Marsh

September 6, 2017

Logistics

1. We will continue to use Clicker, but it will not be graded.
2. Assignments will be Wednesday-to-Wednesday, not lecture-to-lecture.
3. First exam will be October 9th instead of October 2nd.
4. Please back up your private keys!
5. Senior-level Class $\equiv$ You are expected to figure some things out for yourself

Preventing Buffer Overflow Vulnerabilities

Programmers are going to write bad programs.

How do we keep systems safe in spite of this?

In the case of buffer overflows, there are some techniques. We've disabled some of these for our exercises.

Stack Canaries

Add special values at certain places on the stack.

If these values change $\Rightarrow$ something is wrong!

If these are predictable, still possible to defeat, but harder.

We disabled this, along with bounds checking, with `-fno-stack-protector`.

Non-Executable Stacks

Our shellcode has to be somewhere. Usually, this is a buffer on the stack.

`%eip` should never point to a stack location, but that's exactly what we're doing.

If we mark the stack as *non-executable*, we can defeat this type of attack.

We disabled this with `-z execstack`.

Address Randomization

Disassembling in gdb gives us instruction addresses.

Inspecting variables and memory locations in gdb gives us more information.

If this information is stable, we can construct an exploit that will always work.

If this information changes, our work is much harder, because we have to try many times before we're likely to succeed.

Address-Space Layout Randomization (ASLR) randomizes where things are in memory every time you run.

We disabled this with

```
sudo sysctl -w kernel.randomize_va_space=0
```

Question 1

If ASLR is enabled, how might we improve our chances of a successful exploit?

A. Filling the stack with many random addresses, instead of repeating just one

B. Repeating our shellcode multiple times

C. Adding more NOP instructions before our shellcode

D. Disabling ASLR

Return-Oriented Programming

If we can't execute instructions on the stack, what can we do?

1. "Groom" the stack
   - Set registers to useful values
   - This includes arguments as well as `%esp`
2. Set `%eip` to a "gadget"
   - Existing library code, with predictable (or computable) address
   - *Does not* need to be normal function entrypoint
   - One or more instructions using register values we've groomed
   - Ends with a `ret` to take us to the next gadget
3. Repeat

We aren't going to go into this further, but there are many references on the Internet.

Question 2

What's one thing that makes ROP particularly challenging?

A. Determining what instructions you want to run

B. Overwriting a vulnerable buffer

C. Finding gadgets that do something useful

D. Removing NULLs from the exploit

Using the Shell with Malicious Intent

`your_fcn()` is obviously a toy example. We'd never let users hand us the code to execute. (Actually, sometimes we do!)

Most programs these days are *dynamically* linked. Important functions are pulled in at load time from other files. How these files are found is based on searching a path. We can control this path with an environment variable called `LD_LIBRARY_PATH`.

There is also an environment variable called `LD_PRELOAD` that lets us load a dynamic library of our choosing before anything else.

For extra fun, most people have "`.`" in their executable search path, often as the first entry. Nothing stops you from creating a malicious program named "`ls`", and a sloppy sysadmin might be tricked into running it...

Group Exercise 1

Discuss with your group how you would find ROP gadgets in a library. Write some simple code (do one simple thing, like add two numbers), look at the instructions that are generated, and see if you can find that as a gadget in a standard library.

You might want to use `disassem/r` in **gdb** and the **xxd** command.

Principle of Least Privilege

Any principal should have access to

- Resources it needs for legitimate actions
- Nothing else

$\Rightarrow$ Access Control (mandatory/discretionary)

Most software violates this

Take a look at the permissions apps on your phone have...

The Gold Standard

Properties that any secure system should have:

- **Au**thentication
  Establish the *Identity* of a *Principal*
- **Au**thorization
  Establish the *Permissions* of an *Identity*
  This includes Access Control
- **Au**ditability
  Ensure that all operations can be inspected/verified/validated later

CAN Bus

Cars have had computers for years

Controller Area Network bus provides communications between components.

This was great, until...

- Navigation systems
- Internet hotspots
- In-car “infotainment”
- Vehicle-to-vehicle communications (coming soon!)

DNA Malware

This isn't DNA-specific

Larger problem: domain-specific programs written by experts in that domain, not security

Basic assumption: input is well-formed and well-behaved

Vulnerabilities will be common

Use of higher-level languages can reduce these types of errors

Autonomous Vehicles

All the problems of CAN bus, plus:

- Vehicle must be connected to the network
- Reliance on automated processing of local sensors
- Algorithms can only be tested on what the testers think of
- No human-in-the-loop to override poor decisions

Vulnerability Equities

Two groups of people who look for vulnerabilities:

1. People who want to fix those vulnerabilities
2. People who want to exploit those vulnerabilities

National intelligence agencies are unusual, in that they may do both

- Often, separate groups with conflicting goals
- Don't necessarily talk to one another often
- Attack is more fun, less stressful than defense, tends to divide talent asymmetrically

Election Systems

Have special place in society, with important properties required:

- Ballot secrecy
- Resistance to vote-buying/coersion/fraud
- High reliability
- Authentication vs. Disenfrachisement
- Auditable/recountable

Correctness not only important thing — voters must have **confidence** in results

Really good example for various security topics, so we'll return to it throughout the course

Group Exercise 2

Discuss some of the trade-offs in election system properties. How many of these properties do we really have? Where do we compromise on some of them, and why?

Please limit yourselves to technical issues. These trade-offs rarely have objective solutions, and reality often forces us to make imperfect decisions.