# COMP9447

15s2 - LECTURE 1B

# Bug, Threat, Vulnerability, Exploit

**Bug:** An error, flaw, mistake, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways.

**Threat:** A person or group **able to** (intending on?) doing harm or performing some hostile action.

**Vulnerability:** A software or system weakness which allows the system to be used in a manner which would be considered a breach of the intended or extended result.

Irony[1] :a state of affairs or an event that seems deliberately contrary to what one expects and is often wryly amusing as a result.

*In a way, vulnerabilities are computer irony ;-)*

**Exploit:** A tool or code created to exploit one or more vulnerabilities for some purpose. In reality, good exploits for some vulnerabilities take weeks or months of high-skill

You can discover, find, and fix **vulnerabilities** in software! If you “fix an exploit”, you make the attack more reliable. If you “fix a threat”, you may have used a baseball bat. ☺ Confusing these terms is the quickest way of letting someone in security know you don’t know what you are talking about.

# Two Broad Types of Vulnerabilities

- **Design or Architecture Vulnerabilities:**

These are major mistakes in how something is designed, or how components interact with each other at a high-level.

Often very hard to fix – require redesigning protocol, or reworking lots of components, etc.

- **Implementation Vulnerabilities**

These are more minor mistakes in the actual implementation of the program; Highly varied, a lot of these are often more subtle, and harder to find.

# Bugs must be reachable to be vulnerabilities

- The attack surface of a program or system is essentially all of the interfaces which someone can attack.
- Imagine a function which has a stack buffer overflow (potentially):

```
char *newString(char *s) {
  char* m = malloc(128);
  if (m != NULL) strcpy(m, s);
  return m;
}
```

If newString is only called on either non-user controlled data, or data that is checked somewhere previously to make sure it is not less than 128, there is no vulnerability! This is fine. There's only a vulnerability if an attacker can somehow get this function to

However, it's important if you are writing this code that you document this constraint (i.e. we assume only strings below a certain size are passed – dangerous otherwise)

# Vulnerability / Exploit “Range” (TODO)

(from least control over environment to highest)

- **File Format:** File format vulns/exploits are when you can provide a file which will then trigger an exploit. You have a varying level of control – things like Adobe PDF which lets you use a scripting language give you a fair bit of control, so would fit in this sense into the client-side below. Exploits in video files or office files without using a scripting language, or Anti Virus software (hehehe).
- **Remote:** A remote exploit is one that can compromise a system or otherwise perform it’s malicious payload remotely, over a network. There is some argument about this but for the purposes of this course, when we say remote, we mean remote ‘server-side’. These exploits are becoming rather rare for major pieces of software, and they generally give the attacker the least control over their environment.
- **Client-Side:** Same as remote but instead of targeting a server/service, you are targeting a client who connects to you. Sometimes (like with a web browser), you end up with a huge amount of control, able to nicely arrange memory etc.
- **Local (privilege escalation):** When you are actually running code on a system (i.e. you are logged into something, or providing executables) but you want to attack another user or the system itself to elevate your privileges.

# Exploit Mitigations

There are lots of technologies which make exploits harder to write / get reliable, and sometimes they even make it impossible (although, a lot less than advertised..)

We need to disable a lot of these to ***teach*** exploit writing. In this course, we start with none enabled, and then we enable them again one-by-one to teach how to bypass them.

We don't go all the way through all the mitigations on current, hardened devices (such as iPhones or grsecurity linux boxes) – but we give you all the tools you need to go and do your own research and work it out.

# What people do with vulnerabilities + exploits

Vulnerabilities:

- Public (full) disclosure
- Responsible vendor
- (Responsible) non-disclosure

Exploits:

Simple PoC or low quality – public disclosure or pack (i.e. metasploit)

Medium quality – commercial exploit pack

High Quality – sell, or pwn2own, or fame (i.e. Google)

DO NOT MAKE THE MISTAKE OF ASSUMING THAT “Good intentions” WILL KEEP YOU SAFE.

# Common Tools – use these

Code review

- Source Insight is good if you want something fancy
- Vim + CTAGS + CScope is the standard open source way, and I don't mind it too much
- There are some pretty decent webapps that are handy for reading code in, such as OpenGrok

Reverse Engineering

- IDA Pro, IDA Pro, IDA Pro.
- Download the free version

Fuzzing

- Generally, python or ruby, or a framework written in those two (AFL, Sulley)

Debugging

- GDB

# Threat Modelling / Attack Surface

- The attack surface of a program or system is essentially all of the parts which can be attacked.
- Defining a set of possible attacks to consider”
  - Where is it exposed to untrusted data?
  - What could a potentially bad person manipulate?
  - What has the developer assumed? You know what happens when you make assumptions? ;-)
- Attacker centric vs design/software centric
- You have to know and understand the system to have an idea about what is possible
- Asymmetry of attack
- Can apply other known attacks
- Let’s threat model attacking something ourselves

# Risk Assessment

- The common way that governments/businesses assess the actual “threat” of a vulnerability in a system etc.
- Two factors are used, 1st one is Likelihood, or difficulty: How likely the event is to happen.
- Good risk assessments, in terms of vulnerabilities, base this purely on technical difficulty (ie. Days required to develop an exploit), resources required (extremely skilled people? Access to local network? 100s of good machines to break the crypto? Profit of attacker?)
- Second factor is impact: What’s the damage if it happens?