COMP8050 – Security for Software Systems

Dr. David Stynes

Lecturer

- **David Stynes**
- E-mail: david.stynes@mtu.ie

- Berkeley Building Room 118,
- Comp Sci Technical Support Room.

- Office Hours: Fridays 13:00 – 14:00

Lectures

Semester 1

- Tuesday 17:00 – F1.2
- Thursday 09:00 – A123L

Labs

- SDH4-A Thursday 11:00 – IT2.3
- SDH4-B Friday 14:00 – IT1.2

- *Starting Week 2, Sept 22nd*

Recommended Books (not required)

Mark Dowd, John McDonald, Justin Schuh 2007, "*The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*", Addison-Wesley
[ISBN: 9780321444424]

Assessment:

Assignment	–	20%	(Week 6 approx.)
Assignment	–	20%	(Week 11 approx.)
Final Exam	–	60%	(End of Semester)

Plagiarism

1. Plagiarism is presenting someone else's work as your own. It is a violation of CIT Policy and there are strict and severe penalties.
2. You must read and comply with the CIT Policy on Plagiarism http://www.cit.ie/contentfiles/Jill%20Exams%20Office/CIT%20Student%20Reg%20Plagiarism%20-%20Cheating.pdf
3. The Policy applies to *all* work submitted, including software.
4. You can expect that your work will be checked for evidence of plagiarism or collusion.
5. In some circumstances it may be acceptable to reuse a small amount of work by others, but *only* if you provide explicit acknowledgement and justification.
6. If in doubt ask your module lecturer *prior* to submission. Better safe than sorry!

Plagiarism

- Assignments are to be completed *individually*.
- Discussion with friends is encouraged.
- But ***never*** share code/content.

Disrupting Lectures

- Phones set to *silent* mode.
- Be considerate of those around you.
- *Do* ask questions in lectures!!

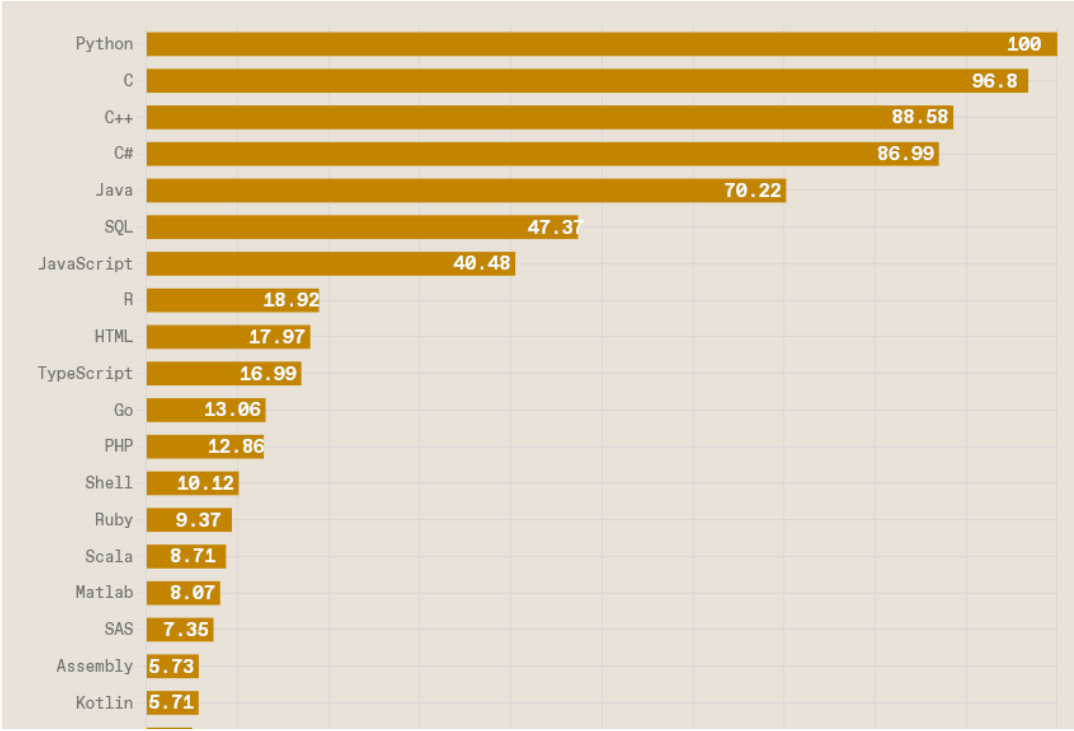
Canvas – Course Support Site

- https://cit.instructure.com
- (Most) Lecture Notes will be available online.

http://www.learn-c.org/

- For learning, C's approach to memory management using the stack and heap is excellent
- These concepts are critical to really understanding the certain sorts of pernicious security vulnerabilities.

C is still very popular!



https://spectrum.ieee.org/top-programming-languages-2022

Target Audience

- People involved or interested in developing secure software.

- This includes people
  - Who design software systems.
  - Who write code to implement those systems.
  - Who review code and designs that aim to be secure.
  - Who test software to make sure it is secure.

Written in C

Most OS Kernels and utilities:

- fingerd, X windows server, shell

Many high-performance servers

- Microsoft IIS, Apache httpd, nginx
- Microsoft SQL server, MySQL, redis, memcached

Many embedded systems

- Aeroplanes, industrial control systems, cars

Black Hat vs. White Hat

- We're going to take on two points of view when looking at software security, Black Hat, and White Hat.
  - *Black Hat* takes on the point of view of the adversary.
  - *White Hat* of the defender.
- So a *Black Hat* is going to ask, what are the security relevant defects that constitute vulnerabilities in our software, and how can those defects be exploited?
- As a *White Hat*, we're going to ask, how do we prevent security relevant defects before we deploy?

Black Hat View

- We will look at low level vulnerabilities in programs written in C, and C++.
- In particular, buffer overflows which can take place on the stack or the heap or due to integer overflow, or over-writing, or over-reading buffers in memory.
- We'll also look at format string mismatches and dangling pointer dereferences.
- Where accesses to memory via pointers go to other parts of the program.

- These vulnerabilities lead to attacks like stack smashing or format string attacks, or return oriented programming.

Memory Safe and Type Safe

- All of them are violations of a property called memory safety.
- The easiest way to ensure memory safety and thereby avoid these different sorts of attacks is to use what's called a memory-safe programming language.
- Or better yet a type-safe programming language.
- If you still want to use C and C++, which are not memory safe, then there are several automated defences that will help prevent or mitigate attacks.

Memory Safe and Type Safe

- We'll discuss several such as
  - Stack canaries,
  - Non-executable data,
  - Address space layout randomization,
  - Memory-safety enforcement, and
  - control-flow integrity.

- We will also consider how to augment these defences using safe programming patterns and libraries

- We will also consider how to **validate untrusted input** and, therefore, prevent certain sorts of attacks.

Web Security

We will consider:

- SQL injection, Cross-site scripting, Cross-site request forgery, and Session hijacking.
- We'll also look at the defences against these attacks.

Security in the software development process

- We will consider the different phases of software development lifecycles including Requirements, Design, Implementation, and Testing and Assurance.
- We'll look at the corresponding activities that take security into account
- For example,
  - Define security requirements and define abuse cases.
  - Perform architectural risk analysis and threat modelling
  - Use a security conscious design
  - Conduct code reviews,
  - Perform risk-based security testing.
  - Perform penetration testing to make sure that the software that we have designed and built truly is secure.

Language Specific Issues

- We will consider secure code development in a high level language.

Requirements & Design

- We will look at how to identify sensitive data and resources, and define security requirements for them.
- Then, apply principles for secure software design to prevent, mitigate, and detect possible attacks.

3 main categories of rules:

1. Favour simplicity in your design and code.
2. Trust components with reluctance.
3. Defence in depth, relying not on one defence but many

Implementation & Testing

- Focus on Rules and Tools
- Apply coding rules to implement our secure design.
- Apply automated code review techniques to find potential vulnerabilities
- We will discuss a technique called static analysis that is able to analyze a program and consider all of it's possible executions when making a judgement.

Penetration Testing

- Goal : to find potential flaws in systems in a deployment environment.
- We will consider different attack patterns as enabled by different sorts of pen testing tools.
- We will discuss a technique called **fuzz testing** for trying to find failure scenarios in software programs.

Summary

1. Memory attacks
2. Memory defences, looking at low-level software.
3. Web security.
4. Secure design and development.
5. Code review, via static analysis in symbolic execution.
6. Techniques for penetration testing, notably fuzzing.