1. Introduction

Computer Security has quickly become one of the most important areas of modern computing. In both the academic and industrial worlds new computer scientists are expected to have an understanding and competence with computer security and to understand how to create software which will be both effective and safe. This recognition of the importance of Computer Security goes all the way up to the White House as shown by President Obama’s active work in cybersecurity\(^1\). To try and gain an understanding of the level of security knowledge within these new software engineers, we have attempted to do an audit of the lower-level undergraduate courses taught at the University of Washington. After reviewing this, it is clear to us that security is a topic lightly covered in the 100 or 300 level courses (Note: UW-CSE does not offer any 200 level courses, only 100 level for introductory courses and 300 level for core Sophomore and Junior courses).

Trying to gain a complete understanding of how well our undergraduates understood security coming out of the 300-level courses is a very challenging task. We recognized that it would be difficult, if not impossible given our time restrictions, to sit in on every single course and do an individual study of its material. As a result, we decided to approach this challenge by studying the textbooks assigned to the students within each course. By reading through these books, we hoped to gain an understanding of what is expected of the students and how well they are being prepared. Within each text we looked for specific markers: is the Security

---

\(^1\) The White House Blog post on the new cybersecurity coordinator.
side of a topic taught/mentioned/warned-about/etc., are any topics taught or demonstrated in an insecure way, and are there any topics taught in such a way as to leave the student open to creating insecure code. Finally, we worked to analyze the results of the above markers to determine how well the undergraduates were truly being prepared.

This choice of only studying the textbooks was also done in the hopes of this work being more influential than just one undergraduate curriculum. We recognized that large populations of software engineers learn their craft by purely learning from the texts. Therefore, we wanted this study to apply to any individuals who may learn by purely working from textbooks and not participating in any formal in classroom education.

2. Background

The University of Washington Computer Science and Engineering Department has been consistently ranked in the top ten Computer Science Departments in the country. The department itself is made up of roughly five hundred undergraduates who are required to complete a wide variety of courses in order to graduate with a CSE degree. In planning our study, we chose to focus only on the 100 and 300 level undergraduate courses. The major reason we chose to stop at the 300 level courses is that the UW CSE majors are given a lot of control over which 400+ level courses they take. As a result, we would have no guarantee that a graduate would complete a course which we evaluated. On the other hand, the 100 and 300 level courses (or the ‘lower level courses’) are required for all undergraduate students. By focusing on these courses, we hoped to get a strong picture of where each student’s understanding of security will be during a specific point in their education.

3. Prompts for this work

Every day something appears in the news which relates to the security vulnerabilities of a piece of software. These security issues cause millions of dollars of damage and can even

2 U.S. News Rankings
result in the loss of life. From the 1980’s Morris Worm\textsuperscript{3} to the modern day Stuxnet Virus\textsuperscript{4} the importance of the effectiveness and security of software is pervasive throughout our lives.

Looking at the level of understanding of security from the standard undergraduate Computer Science/Engineering major, it became clear to us that the lower-level courses should be covering much more. The interest to learn about security is definitely in the student body. The department’s upper level Computer Security course is one of the most sought-after classes in the undergraduate curriculum. By evaluating and updating the lower level courses, we hope to advance the department into producing graduates with a much stronger understanding of security.

4. What is currently taught

As mentioned above, the lower level Computer Science courses are split into two different categories: 100 level courses which are the introductory series and 300 level courses which consist of a wider range of topics and a more advanced set of topics. Both of these groups of courses do a wonderful job of preparing their students for a future in Computer Science and we have no disagreements with their successes at teaching computer software fundamentals.

The 100 level courses, broken up into a series of two classes, teach all of the basics of Java programming. Students are expected to have no prior programming experience and work from the ground up. Topics covered in these courses include (amongst many others):

- proper formatting
- java style
- object oriented programming
- basics of data structures and algorithms

These areas of study are a great way to begin teaching the students about how to program and

\textsuperscript{3} A look back on the Morris Worm
\textsuperscript{4} An overview of Stuxnet
are used as a gateway into the more advanced 300 level courses.

The 300 level courses, made up of seven total classes, are much more in depth study of Computer Science and work to greatly increase the student’s knowledge base. These courses’ topics include:

- design and construction of software systems
- software engineering experience
- effective testing
- building CUI’s vs GUI’s
- data structures and algorithms
- development in C and C++
- memory management
- human-computer interactions

The courses go on to teach the basics of computer hardware and assembly language as well as the theory of computing.

As can be seen in the above lists, there is a substantial amount of information that is taught in these lower level courses. Unfortunately, security is a minor (and sometimes non-existent) topic in all of the 100 and 300 level courses. We feel strongly that its presence in these lower level courses needs to be increased, otherwise the student’s first real security experience won’t occur until they take an optional 400 level security course.

5. Security Metric

There are currently many different ways in which common computer security issues are rated and analyzed. One method is to create a list of security threats or security standards on which to base your work off of. Two such examples are the Common Weakness Enumeration
(or CWE) list⁵ and CERT⁶. We have chosen to extend the current CWE metric by adding our own entries to the list of the most common and/or most worrisome security concerns within the undergraduate curriculum. By doing this, we hope to build on the success of the CWE metric by applying it more specifically to the area of Computer Security education.

We began our metric by using the CWE’s “Top 25 Most Dangerous Software Errors”. We then gradually updated the metric to also include the new topics brought forth by our results from the analysis of the undergraduate textbooks and where we feel changes need to be made. Finally, we went back through the metric and removed any of the original CWE topics which we felt did not apply in our case. This left us with one metric which was able to categorize all of the new issues we discovered. We have worked to include everything from very technical and specific programming issues all the way up to the theoretical choices of a developer in how to design a system.

We feel that our metric⁷ does a good job of recognizing where the threats are and what areas need to be updated. As you will see further along in this paper, we have marked each concern found within the textbooks with a unique identifier. This identifier correlates to an entry in our metric, allowing you to see how common a concern arises.

6. Result Explanation

As stated above, we chose to focus purely on the information portrayed in the textbooks for each of the 100 and 300 level courses. Each text was then analyzed by comparing what we read with our metric of security concerns. If we discovered a new concern then it would be noted and a new entry would be added to our metric if necessary. If this was a repeated issue, then we would simply mark that it is a recurring concern either throughout a text or across multiple texts. We made our way through the courses, one at a time, reading each text with two

---

⁵ A list of the top 25 most Dangerous Software Errors
⁶ A group working to create secure coding standards.
⁷ See Appendix for metric
different viewpoints in order to get the most accurate portrayal of each book.

The first view was of a student who is entirely new to the material portrayed. By reading the textbook with this in mind, we were able to absorb the material in an entirely free manner in the same way a new student would. We hoped that by doing this we could remove some of the bias imparted by our own knowledge of the material. This allowed us to ‘learn’ a topic for the first time and then imagine how we would implement a real world version of the same type. By doing so, we were able to see how a student would extrapolate the examples found in the text into larger projects in an internship or job.

The second view was of a security conscious researcher (a.k.a. us) who would be able to find security concerns that a beginner might miss. Specifically, we built on top of the larger project ideas and tried to see where a reader could make a mistake while trying to implement the example at that time or in the future. These mistakes could be due to a poor understanding of a potential concern or even a direct issue with the manner a topic is taught.

By combining these two ‘readers’, we hoped to get the most out of every textbook and to discover as many security concerns as we could. Naturally, we can never claim to have found all of the security issues in each book. This is especially true because Computer Security, along with Computer Science in general, is continually changing with new exploits and bugs being discovered every day. As a result, we can only provide a list concerns that we feel are currently the most important topics to cover in terms of security.

7. The 100 level courses (Introductory)

“Building Java Programs” - By Stuart Reges and Marty Stepp

This text is used as the introduction to programing course for all Computer Science undergraduates and it has a very successful track record as such. It is used every year by over a thousand students at the University of Washington, not to mention elsewhere in the country,
and we feel confident that it is a great way to begin teaching students how to program in Java or any other language. That being said, we also noted an absence of any discussion of security or the importance of secure programming.

We recognize that this is an introductory course and as such cannot be expected to go into the depth of security needed to produce security conscious developers. However, we do feel that some of the topics we are concerned about could be easily addressed with simple rewrites in how the topic is taught within the text. We concede that other topics would require a substantial amount of work to teach the security side effectively and that they should not be a part of these introductory courses.

Of the areas we were concerned about with this text, the most severe areas are:

<table>
<thead>
<tr>
<th>Topic of Concern</th>
<th>ID on our Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>The idea of overflowing a primitive is never taught or even mentioned.</td>
<td>UWS-04</td>
</tr>
<tr>
<td>The danger of ‘public’ variables and methods.</td>
<td>UWS-07</td>
</tr>
<tr>
<td>Timing attacks (what they are, how they can be used, etc.) and what can make a program vulnerable to one (i.e. early exit efficiencies).</td>
<td>UWS-08</td>
</tr>
<tr>
<td>Assuming a well-meaning user and how/why to sanitize a user’s input.</td>
<td>UWS-10</td>
</tr>
<tr>
<td>The dangers of scanners and of allowing them access to files/folders on the host machine.</td>
<td>UWS-11</td>
</tr>
</tbody>
</table>

These topics were the most common issues that we found in the text and also those which we felt, if left unaddressed, could cause the most harm. Should a student go on to an internship or job and develop a public facing program with any one of these vulnerabilities, they would leave their users and their company open to massive damages.

These concerns are only the beginning of the areas within this text which we found concerning. Many of the topics in our metric are relevant to this text in one way or another. This made us realize that readers need additional knowledge than what they get from an
introductory textbook. If they used only that to create commercial software then they are a potential security threat and need further education. This is especially important as many students in other departments, engineering or otherwise, will take the 100 level courses, but that will be their only official Computer Science training. They will then go on to a job in their field where, if they are required to write software, they could cause a lot of damage.

8. The 190M course (Web Programming)

“Web Programming Step by Step” - By Marty Stepp, Jessica Miller, and Victoria Kirst

An optional course offered by the CSE Department, 190M is an introduction to web programming. It is designed to orient the students to the world of developing software for the web. The textbook required for this course goes along with that ideal. It is meant as a basic guide to creating anything a web developer might need. This includes everything from html and css all the way to JavaScript and Ajax. Due to the broad level of topics, the book offers reminders of common areas where a developer might make a mistake. These areas include forgetting an ‘=’ within a PHP expression to mismatching brackets in html.

Comparing this text to general web security, it is clear that this text was not meant to provide the reader with a strong understanding of web security. It was purely meant to create a broad level of understanding of how the web works so that the students can continue on and learn more in the future if they so wish. As a result of this, there are many security topics left out of this text such as cross-site scripting attacks, cross-site request forgeries, malicious links, etc. In order to confidently have a student build a full web app/system, that student would need to continue their education in the web security field and not rely simply on this course.

9. The 331 course (Software Design and Implementation)

“The Pragmatic Programmer” - By Hunt and Thomas
This textbook is a very different from the one for the introductory series. It is designed to create capable software engineers who understand what is needed to create reliable and maintainable software systems. This includes good development practices, testing techniques, work distribution skills via software-team experience, and much more.

Due to the differences in topics being covered, we approached this book with a different focus. We wouldn’t be able to discover technical details which were taught in an insecure way, but instead we strove to find software engineering practices which we felt could leave a developer open to making a security mistake.

Of the areas we were concerned about with this text, the most severe areas are:

<table>
<thead>
<tr>
<th>Topic of Concern</th>
<th>ID on our Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>The threat of only using a ‘temporary fix’ rather than a well-tested and complete update.</td>
<td>UWS-12</td>
</tr>
<tr>
<td>Allowing a user to experiment with an early version of the product with ‘rough edges’. This still needs to be secure and can’t have any more vulnerabilities than a complete product.</td>
<td>UWS-10, UWS-13</td>
</tr>
<tr>
<td>The dangers of storing data in plain text either on the back-end system or servers or on the user’s side.</td>
<td>UWS-14</td>
</tr>
<tr>
<td>They do teach the basics of debugging (what to look for, what a bug is, etc.), but they do not teach how to create effective tests to find them.</td>
<td>UWS-01</td>
</tr>
<tr>
<td>Preconditions are not enough to guarantee user input. This is very dangerous as they imply that a precondition is sufficient to assume that all input will be of the correct format. (also a continuation of the same issue from the introduction book)</td>
<td>UWS-16</td>
</tr>
</tbody>
</table>

These topics pose a direct threat to industry and the users of any software product as this textbook is designed to prepare the reader to be an effective software engineer. The threat is if a company should hire an applicant who based all of their software engineering work upon this text alone. There is a decent chance that this engineer would create a product with dangerous security holes that could otherwise have been prevented.
10. The 332 course (Data Abstractions)

“Data Structures and Algorithms Analysis in Java” - By Weiss

This textbook is purely a data structures and algorithms text and does not go into the implementation or use of any data structure or algorithm beyond the standard best-practices methods. The average reader would also choose to implement these data structure or algorithms via the libraries associated with the programming language in question, therefore negating any security issues that might be hidden in the text’s implementation. Due to it not going beyond this standard implementation, we did not discover any security concerns which we felt needed to be addressed.

11. The 333 course (Systems Programming)

“C++ for Java Programmers” - By Mark Allen Weiss

The transition from developing in Java to using C or C++ brings with it many new security concerns. These concerns are unique to the new language as well as due to a confusion of the differences from the language you already know to the new one. This book is proof of this as there are many instances where a reader could easily have a lacking level of understanding with respect to the security of a specific topic. That being said, this book is not different in its numerous security issues that carry on directly from the other textbooks. Two such concerns are:

<table>
<thead>
<tr>
<th>Topic of Concern</th>
<th>ID on our Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sanitizing user input</td>
<td>UWS-10</td>
</tr>
<tr>
<td>The dangers of editing, overwriting, and deleting files</td>
<td>UWS-15, UWS-17</td>
</tr>
</tbody>
</table>

Both of these concerns, especially the first one, are relevant to our review of almost every textbook we considered in this study. The concept of sanitizing a user’s input is a security
concern which cannot be warned about enough and yet it is left out time and again. The dangers of editing, overwriting, and deleting files also seems like a very general concern which could be an obvious topic to anyone. However, the nuances that go along with this and larger software projects are an important topic to mention. Specifically, what happens when one person modifies a file unbeknownst to another developer who uses the same file. It is possible that the changes will go unnoticed, even by an experienced and talented developer, until a bug appears that could be massively damaging. As one can see in these examples, and many others, these issues could result in seemingly effective software being riddled with bugs which put the company and users as risk.

This text stands out from the others in that it mentions many of the areas that a developer would need to be cautious about when writing their code. Unfortunately, even though the text mentions that one must be careful, it does not go into the details of what could happen should the developer make a mistake. Specific example of this are:

<table>
<thead>
<tr>
<th>Topic of Concern</th>
<th>ID on our Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>They show how to pass a 'long' into an 'int' variable, but not what could happen if the value is out of scope of an 'int'.</td>
<td>UWS-04</td>
</tr>
<tr>
<td>They explain how a stale pointer is created, but not what an attacker could do with one.</td>
<td>UWS-09</td>
</tr>
<tr>
<td>They mention how C++ doesn’t check if you are trying to access beyond the end of an array, but not the dangers of this vulnerability (i.e. a Buffer Overflow attack).</td>
<td>UWS-09.1</td>
</tr>
</tbody>
</table>

As you can see, many of the ideas mentioned in this table are important details about developing in C/C++ that a Java developer might be unaware of. This text does a wonderful job in showing the reader many of these areas, but it fails to explain why they are dangerous. A developer may understand that a stale pointer could be created in certain situations, but would have no inclination to care about such a thing if they are not warned about its dangers. This applies to all of the concerns mentioned here along with many others found in this text.
12. Review Conclusion

For each of these textbooks, and presumably many others, we have discovered countless occurrences of topics being taught or demonstrated in such a way as to create security concerns. We feel that these are security issues because if a reader were to go out directly after reading these books and write commercial software then there is a chance that they would produce insecure and dangerous products. We have mentioned above a subset of concerns from each textbook, but there are many more that we have found in all of the texts. Our metric, found in the Appendix, is a good example of this as it shows the other key concerns which were found in many of the books. Many of these concerns arise multiple times with one text and reinforce our worry about how a reader might be taught these topics. As such, we feel strongly that something needs to be done to repair these security issues and prepare future developers as well as we can.

13. What We Want

After reviewing the results from all of the textbooks, we feel that there are two possible ways that this lack of security education could be fixed. The first would be to update all of the texts with the necessary warnings of the security concerns with each section of the textbooks. The second option would be to create an entirely new undergraduate course, with matching textbook, which would focus on covering the types of security concerns which are on our metric.

Although the first solution seems the most logical, i.e. fix what is ‘broken’, we do not think this is the best solution. Many of these textbooks are already exceedingly lengthy and the courses which use these books are filled to capacity with topics to teach. If the textbooks tried to expand their descriptions to include the security portion then we feel that they would grow too far out of control and be much too difficult for readers to digest. This carries over to the courses that use the textbooks as we do not think that these courses would be able to teach the security aspects of each topic without dropping another important area of the course. This concern is
especially true for the introductory textbook and courses as brand new students might have a lot of difficulty understanding the security concerns raised above.

The second solution is much more appealing to us as we feel it will solve the issue of preparing students or readers and it does not have the same issues as the textbook expansion idea. By creating a new course, we would hope to try and patch any holes in the security knowledge imparted on the students who read the above textbooks. This course, and the potential accompanying book, would be able to show the students how each topic they were taught in the earlier courses has a security component to it and how to implement that topic securely. By taking this course, the students would be able to produce software which implemented many of the methods taught in the 100 and 300 level courses, but with the added benefit of being secure and hopefully security bug free. The only issue with this solution is that any student or reader who chose not to take advantage of this course or textbook would be right back in the same position as before. There is no way for us to require any reader, who isn’t a student, to read this new text in addition to the book they have already selected so we feel that this is a subset that we cannot fix.

14. The New Course

For our work, we decided that we would be much more successful at designing a rough plan for this new course than we would if we tried to plan out an entirely new textbook. As such, this section will be describing what this new course would be relative to the University of Washington’s Computer Science and Engineering Department. One specific thing to note is that the CSE 484 course is the University of Washington’s senior level Computer Security course. Our proposition is to create a new CSE 384 course which would be designed to be a lower level security course and that could potentially be required of all undergraduates. The focus of this course would be to ensure that all graduates left the school with at least a basic understanding of Computer Security in terms of software development.
This new course would be used to fill the gap in security education at the 300 level, focusing on undergraduate sophomores and juniors as the main student basis. Its primary goal would be to educate these students on the security portions of much of the material they already learned in their other 100 and 300 level courses. We would work to show them how the old methods they learned are insecure and where they would need to be most careful in the future. We would build our syllabus off of our metric from the analysis of the other textbooks and attempt to cover as much of it as we can, if not all of it.

A secondary goal of this course would be to prepare the students for a more advanced, in depth, and hands-on 400 level course. This updated 484 would be able to build off of what the students learned in our new 384, and not spend nearly as much time updating old knowledge. Much of the currently material from 484 would be pulled out and used in the 384 course to continue their education on how to be effective software engineers, just as the current 484 does. As a result, the new 484 could go much further in depth into Computer Security, and become a far more hands on course. We feel that this would benefit the students even more than the current 484 course does.

Much of the 384 course material would be pulled from the metric we have come up with, but, as we mentioned above, we can also pull a lot of it from the current 484 course. Specifically, we could teach:

- How to avoid timing attack vulnerabilities
- How to create robust code which prevents attackers from corrupting a system
- How to protect against a buffer overflow attack
- How to protect against a malicious user
- How to defend the rest of the system from being corrupted due to your program’s bugs.
- etc.

We feel that all of these topics would be great examples of what could be covered in a 300 level
course. If all undergraduates took this course, then the department could be confident in its graduate’s security knowledge and the 484 course could be greatly expanded.

15. Conclusion

There is much to be improved upon with the lower level courses with respect to Computer Security. Looking at the material covered in the standard textbooks and how the students are prepared, we feel that they are not being given the information they need from an undergraduate Computer Science and Engineering Department. We feel strongly that the students are being trained very well, but this gap in their understanding of how to create secure software will leave their future employer’s systems and products vulnerable.

By focusing on the textbooks and not just the courses, we have worked to expand our study beyond just the University of Washington. These are very standard choices of textbooks used by universities and individual alike and we hoped to expand our result to all of them. By trying to keep ourselves at this broader level, we are working to raise the awareness of this issue and to begin the design of a new course, and possibly textbook, which will work to repair these issues. If this new course were to be implemented, we hope to be able to have the students complete their undergraduate education with a much more complete view of software engineering and the importance of security. Finally, our long term goal would be to expand beyond just a new course and to write a new textbook which would allow these goals to apply outside of just the University of Washington.
Bibliography:


Appendix:

7. Our table:
   a. **Code** - The ID code we have assigned to the rule for our metric
      i. **UWS** - Stands for ‘University of Washington Security’ and is used to represent our own identification of a new rule.
   b. **Rule** - The definition of the Security Issue we are concerned about
   c. **Other ID** - The ID number of another location (either in our UWS or the CWE metric) of the same or a similar rule

<table>
<thead>
<tr>
<th>Code</th>
<th>Rule</th>
<th>Other ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UWS-01</td>
<td>No explanation of testing; especially testing to find/fix bugs.</td>
<td>CWE-798</td>
</tr>
<tr>
<td>UWS-02</td>
<td>Bad code development styles (i.e. copy + paste, hard-coded values)</td>
<td></td>
</tr>
<tr>
<td>UWS-03</td>
<td>Common places in code to create bugs (i.e. losing the decimal place going from int → double → int)</td>
<td>CWE-190</td>
</tr>
<tr>
<td>UWS-04</td>
<td>Primitive overflow/wraparound; also applies to variable controlling a loop</td>
<td>UWS-28</td>
</tr>
<tr>
<td>UWS-05</td>
<td>Printing excessive information (parameters of loops, number of times run, variables used, data stored, etc.).</td>
<td></td>
</tr>
<tr>
<td>UWS-06</td>
<td>Security surrounding pseudocode (physical security of the pseudocode; it gives away just as much info as code).</td>
<td></td>
</tr>
<tr>
<td>UWS-07</td>
<td>Public vs Private vs Protected variables/methods</td>
<td></td>
</tr>
<tr>
<td>UWS-08</td>
<td>Timing attacks.</td>
<td></td>
</tr>
<tr>
<td>UWS-08.1</td>
<td>Make code as efficient as possible (i.e. use early exits, etc.).</td>
<td></td>
</tr>
<tr>
<td>UWS-09</td>
<td>Issues which apply to other languages (especially C after being taught Java).</td>
<td></td>
</tr>
<tr>
<td>UWS-09.1</td>
<td>Buffer overflow or buffer size issues.</td>
<td></td>
</tr>
<tr>
<td>UWS-09.2</td>
<td>Format string attack.</td>
<td></td>
</tr>
<tr>
<td>UWS-10</td>
<td>Assuming a well-meaning user; sanitizing user input.</td>
<td></td>
</tr>
<tr>
<td>UWS-11</td>
<td>Scanners can be dangerous (They can read/write to any file on your machine if you're not careful).</td>
<td></td>
</tr>
<tr>
<td>UWS-12</td>
<td>Temporary fix to a problem (i.e. dummy message 'Not Implemented' or a small untested patch).</td>
<td></td>
</tr>
<tr>
<td>UWS-13</td>
<td>Rush out a product with 'rough edges'</td>
<td></td>
</tr>
<tr>
<td>UWS-14</td>
<td>Missing encryption of sensitive data (storing data in plain text)</td>
<td></td>
</tr>
<tr>
<td>UWS-15</td>
<td>How often to check for accidental or malicious changes (with diff or other tools)</td>
<td></td>
</tr>
<tr>
<td>UWS-16</td>
<td>Over-confidence in use of pre-conditions (they aren't checked/tested by the code unless you build in tests).</td>
<td></td>
</tr>
<tr>
<td>UWS-17</td>
<td>Refactoring working code and not testing it to make sure that it still works.</td>
<td></td>
</tr>
<tr>
<td>UWS-18</td>
<td>Giving the end-user too much access/control/information of the product (i.e. hot-key to open diagnostics window).</td>
<td></td>
</tr>
<tr>
<td>UWS-19</td>
<td>Improper neutralization of input during web page generation (i.e. Cross-Site Scripting)</td>
<td></td>
</tr>
<tr>
<td>UWS-20</td>
<td>Missing Authentication for Critical Function</td>
<td></td>
</tr>
<tr>
<td>UWS-21</td>
<td>Missing Authorization</td>
<td></td>
</tr>
<tr>
<td>UWS-22</td>
<td>Unrestricted interaction (especially uploading) of file with dangerous type.</td>
<td></td>
</tr>
<tr>
<td>UWS-23</td>
<td>Execution with unnecessary privileges.</td>
<td></td>
</tr>
<tr>
<td>UWS-24</td>
<td>Cross-site request forgery.</td>
<td></td>
</tr>
</tbody>
</table>

CWE-120, CWE-131, CWE-134  
CWE-120, CWE-131  
CWE-134  
CWE-89  
CWE-22; CWE-434 [semi-related; see UWS-22]  
CWE-311, UWS-33  
CWE-78; CWE-807  
CWE-250  
CWE-352
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>CWE</th>
</tr>
</thead>
<tbody>
<tr>
<td>UWS-25</td>
<td>Use of external code that you don't fully understand.</td>
<td>CWE-494</td>
</tr>
<tr>
<td>UWS-26</td>
<td>Incorrect authorization.</td>
<td>CWE-863</td>
</tr>
<tr>
<td>UWS-27</td>
<td>Inclusion of functionality from untrusted control sphere.</td>
<td>CWE-829</td>
</tr>
<tr>
<td>UWS-28</td>
<td>Incorrect permission assignment for critical resources</td>
<td>CWE-732</td>
</tr>
<tr>
<td>UWS-29</td>
<td>Use of potentially dangerous function or tool.</td>
<td>CSE-676</td>
</tr>
<tr>
<td>UWS-30</td>
<td>Use of broken or risky cryptographic algorithm (don't create your own).</td>
<td>CWE-327</td>
</tr>
<tr>
<td>UWS-31</td>
<td>Improper restriction of excessive authentication attempts</td>
<td>CWE-307</td>
</tr>
<tr>
<td></td>
<td>(i.e. prevent brute force).</td>
<td></td>
</tr>
<tr>
<td>UWS-32</td>
<td>URL redirection to untrusted site ('open redirect')</td>
<td>CWE-601</td>
</tr>
<tr>
<td>UWS-33</td>
<td>Use of a one-way hash without a salt.</td>
<td>CWE-759</td>
</tr>
</tbody>
</table>

References:
- List of Other Weaknesses: [http://cwe.mitre.org/top25/cusp.html](http://cwe.mitre.org/top25/cusp.html)

Below you will find the full list of concerns noted per book. They are broken up by Chapter and then labeled by page number.

8. **CSE 142/143 Book:**

**Chapter 1:**
- 9, 12, 27, 33 - No explanation of the code; specifically what public, static, void, main, etc. mean.
- 11 - No explanation of what ‘public’ means and how open it is. Only says ‘open to anyone to use’, but doesn’t explain why that could be a problem.
- 14 - No explanation of how to escape to get double quotes: \\
- 15 - Teaches ‘System.out.println”, but no explanation of what is actually happening and what they are.
- 19 - Explains legal names, but not the issues with repeating a name (more of a C issue since Java compiler catches it).
- 20, 27 - List of Java keywords, but not what they actually are or what they do.
- 24 - Explains type of errors, but doesn’t specify that ‘logic errors’ are of the biggest concern for security.
- 28 - Doesn’t teach how to find/fix a logic error.
- 31 - Teaches iterative coding (breaking up a problem into sub-parts), but only has you test that each part works individually, not that they work together correctly.
- 31 - Copy and pasting of code is dangerous.
- 40 - Doesn’t teach how to do additional error testing

**Chapter 2:**
- 62, 79 - No explanation of the potential issues of going between int and double
- 63 - No explanation of how the code handles multiple variable types (i.e. 4 + 8.2)
- 72, 98 - No mention of how a variable is stored in memory (more important for C)
74 - No mention of what happens when you access an uninitialized variable (more important for C)
75 - printf and it's complexity is not mentioned, even though the student could find it easily having been taught print and println.
77, 97 - teaches variable increment/decrement, but doesn’t mention overflow
94 - no mention of off by one errors when looping
94 - no mention of infinite loops by creating a bad loop
105 - Teaches pseudocode, but not the security risk of pseudocode becoming public.
108 - teaches global class variables, but makes them public.

Chapter 3:
138 - timing attack if the attacker could call the method and observe the result.
140, 148 - At this point the way objects are passed as parameters hasn’t been taught which could lead to a misuse of objects as parameters. Same issue with returning an object vs. a primitive (page 148).
153 - Even if you ignore a return value, it is still stored in the registers (at least in C) and is therefore accessible to an attacker.
159 - No mention of accessing beyond the end of the string (definitely applies to C, but not sure about Java)
165 - This will check that the user enters the right value type (int,double,string, etc.), but there is no mention of sanitizing user inputs.

Chapter 4:
241 - Teaches how to use ‘&&’, but if you’re using an object you always need to see if it’s null first
241 - timing attack if using if/else to early exit
244 - If you create a large variable within early loop then a timing attack could detect how many times you loop.
261 - No instruction on how not to use printf and how it’s dangerous (at least in C).
265 - They teach you how to throw an exception, but not how to catch it. This could let an attacker easily crash your program.

Chapter 5:
308 - no sanitizing of user data
323 - Use of ‘||’, but no mention of ensuring that the user can’t pass a ‘true’ value (SQL attack).
   - Also no mention of incorrect use of ‘||’ by making everything pass: (x < 6 || x >= 6)
     (could be an easily made typo).
338 - Letting a user keep trying values and giving them feedback about what they entered could easily give away a lot of information about what you’re looking for.
   - also a timing attack for how long it takes you to deny a value.

Chapter 6:
378 - Teaches how to scan a file on your system, but no mention of the possibility that the file has been altered/corrupted.
390 - The idea that the scanner keeps it state over method calls (i.e. remembers where in the file it currently is as it gets passed between methods) also holds true for objects, but that isn’t mentioned.
392 - No check that the user provides a valid file name.
393 - If you’re code allows this to happen, then you’ve now opened up every file on your entire system to be read/edited by an attacker.
   - No mention of this threat or of how scanners provide this large opening.
402, 406 - Teaches you how to write to a file, but doesn’t mention the threat of writing to the wrong file
   - If the user can select the file, they can destroy your entire OS.
407 - If you ask the user to select which file to use and then inform them if they gave a bad name, they could now determine exactly what files are on your machine and where they are located.

Chapter 7:
434 - First mention of security
   - buffer-overflow attack, and how Java is more secure with array-bounds check
442 - No check on the ‘size’ that the array is set to; if made too large the program could run out of memory.
451 - timing attack; early exit from loop when you find a match.

Chapter 8:
506 - still public global variables
511, 514 - no error checking on the values of x, y, dx, or dy; no overflow check; if x*x + y*y results in an overflow, the int will flip to the negative and crash the Math.sqrt.
524 - No mention of the information given away by the public constructors and how many parameters they take.

Chapter 9:
565 - If an attacker could replace one of your sub-classes with their own, your code would still call it allowing them access to your system with their code. (is this a high possibility?)
566 - Single point of failure; if a super-class is compromised then all subclasses have the same vulnerability/problem.
579 - If an attacker could pass in their own object which is a sub-type of ‘Employee’, they could see what fields you’re using in your public/private methods.

Chapter 10:
638 - timing attack to determine how many elements in a list
639 - Object manipulation becomes an issue if multiple parts of code use the same object at the same time (multi-threading). An attacker could potentially force a crash with this if the code isn’t thread-safe.
658 - Comparable gets dangerous when combined with something else (ex: allowing a scanner to compare a provided file name with what’s on the machine).

Chapter 11:
683 - In other languages you must build the linked list yourself which could lead to mistakes being made.
694 - With a predictable hash, an attacker could force hash-collisions and result in the programing slowing/crashing and/or running out of memory.

Chapter 12:
725 - timing attack: recursion always takes the early exit.
725 - no error checking, what if n < 0, then it’d keep looping until n flips back to the positive side and counts all the way down to zero.
736 - Since recursion always has the backtracking step, an attacker could using a timing attack to detect recursion vs loop iteration.
   - similarly recursion would require more memory for the extra method calls and parameters.
Chapter 13:

778 - By entering a value you know isn’t in the list, you can use a timing attack to approximate the size of the list since the Binary Search algorithm will always check up to a max number of time given the length of the list (2*\log(n))?

780 - If you use Quicksort, an attacker could lock up your system if they pass you a large sorted list.

- A potential threat since Arrays.sort uses quicksort on lists of primitives.

783 - A public comparator gives away to an attacker how you’re doing your comparisons.

799 - I’m assuming that there are no issues with the Binary Search or recursive binary search code.

801 - An attacker could use the return value of Binary Search to analyze your list.

- it returns (-(insertion point) - 1), where ‘insertion point’ is the position that the value should be inserted to keep the list sorted.

806 - I’m assuming no issues with Selection Sort.

Chapter 14:

826 - First instance of sanitizing user input.

835 - everything is still public.

[844 - Assuming no GUI layout security issues.]

Chapter 15:

892, 894, 896, 899 - no error checking on size and going out of bounds on the array; no checking on the validity of the passed index (i.e. index >0 && index <= size && size < list.length).

902 - fixes these issues, but never explains them, just makes them preconditions.

906 - An exception seems strong because it’ll crash the program unless the coder knows to catch the exception.

- an attacker could use this by forcing the program to crash and exit.

Chapter 16:

934 - The global variables are still public.

935 - Other languages don’t have the garbage collector, so the coder must free the memory before losing the pointer to it.

935 - In other languages, if you don’t free a node before losing reference to it, it’ll just sit in memory and take up space since there is no garbage collector.

956 - Timing attack on an ‘addSorted’ for a list.

Chapter 17:

985 - All global variables are still public.

995 - no check that n doesn’t overflow the int and loop to the negative value.

- if ‘max’ is almost 2^31 and then 2 * n would be > 2^31, then n loops to a negative number and makes this an infinite recursive method.
1002 - no check that the sum doesn't overflow int.
1004 - no mention of the idea/possibility of a cycle which would make this an infinite recursive method.

9. **CSE 190M Book:**

Due to this textbook being a purely basic overview of web programming, there is not a reason to do a full security audit of its contents. It is clear that this text is meant to provide the reader with a general understanding of the many pieces of web development, but not to be a complete end-to-end tutorial.

10. **CSE 331 Book:**

**Chapter 1:**
5 - if you ‘board up’ a problem (i.e. put in a temporary fix or display a dummy message like “Not Implemented”), make sure to label who is responsible; code left in place like this can create a problem if never returned to. (UWS-12)
11 - Even if you’re producing a product with ‘rough edges’ for the users to get their hands on, be sure to not skimp on security. A product can have plenty of rough edges without hindering the user, but ‘rough edges’ in security cause major issues. (UWS-13)

**Chapter 2:**
27 - Great point on the DRY principle and why it avoids problems.
63 - any ‘internal company security concerns’ apply here. These ‘languages’ give away a ton of information about your code (i.e. don’t let them get distributed).

**Chapter 3:**
72 - new tools are great, but they don’t mention that you must be careful with all tools as they each have new security concerns (especially if used incorrectly). (UW-29)
74 - When discussing using plain text to store data, there is no discussion of the security concerns that go along with it. Storing information in plain text means that any person (of any technical ability) can interpret the data. (UWS-14)
74 - They mention using MD5, but wasn’t it cracked?
76 - mention using ‘diff’ tools to check to changes whether accidental or malicious, but no mention of when/how-often to check for these changes or how to prevent them occurring. (UWS-15)
97 - They teach how to debug (what to look for, what a bug is, etc.), but not how to effectively test for it (i.e. complete testing of all cases). (UWS-01)

**Chapter 4:**
111 - Great point that pre-conditions are useful, but should not be ever used for such things as user-input validation. (UWS-16)
115 - They give responsibility for checking the validity of data to the ‘caller’ portion of the code; often your code is the caller as you use libraries or your own functions, but they don’t teach how to do good validation.

**Chapter 5:**
144 - By making your code dynamic and configurable, are you giving the users too much control? This could be exploited if done in a sloppy manner by the developer. (UWS-18)
Chapter 6:
188 - Refactoring code to make it shorter/cleaner is great, but you must be careful that in doing so you don’t break any of your security measures which the ‘dirtier’ code fulfilled. (UWS-17)
196 - They discuss putting a ‘hot-key combination’ into the program which would open up a diagnostics window to give status messages if something goes wrong. This is VERY dangerous if an attacker discovers it (and they will). (UWS-18)

Chapter 7:
Discussions of the planning phases of a project. No mention of planning out security concerns/threats and how to implement the appropriate measures to prevent them.

Chapter 8:
General clean-up and finishing up the book. Includes working as a team and general summary of the book. (Not relevant for this study)

11. CSE 332 Book:
Due to this textbook not going beyond the standard implementations of algorithms and data structures, we did not discover any security concerns which we felt needed to be addressed.

12. CSE 333 Book:

Chapter 0:
5 – GREAT! Mentions the vulnerabilities of C++ not checking array indexes which can result in buffer overflows. (UWS-09.1) (UWS-03)
9 – GREAT! Mentions how C++ does not throw compile or runtime errors with the above issues. (UWS-09)

Chapter 1:
12 – Mentions that they fail to handle input errors, but states that i/o is handled in Chapter 9. (UWS-10)
[Note: chapter 9 doesn’t discuss sanitizing user input]
12 – The range and size of the int can vary from machine to machine which means that there is no guarantee on the size and using large numbers could therefore be very dangerous. (UWS-04)
14 – Great that they mention that C++ has operator issues that are machine dependent (i.e. the same code reacts differently depending on where it is run), but they don’t give any instruction on how to write code that is safe even with this. (UWS-09)

Chapter 2:
20 – Explains how C++ allows you to pass a long into an int, but this could easily crash a program if the long is out of the range of int on that machine. (UWS-04) (UWS-09)
25 – Teaches the vector array rather than the built in array and mentions how the primitive string is dangerous. This is great, but programmers of C++ will at some point need to use the primitive array/string objects when manipulating old C++ code or C code, so it is potentially dangerous that these aren’t taught until very late in the book. (UWS-09)
26 – Mentions how you can access memory beyond the end of an array allowing you to
overwrite some areas of the buffer. Good that they’ve mentioned it, but they don’t discuss what
the true danger is and what are good ways of avoiding it. (UWS-09.1) (UWS-03)

Chapter 3:
38 – Shows how by incorrectly changing a pointer you could accidentally overwrite other areas
in memory. Doesn’t mention how an attacker might want to get your code to do this so that they
could then manipulate what you have stored. (UWS-09.1)

Chapter 4:
71 – Explains how stale pointers are created, but not about any of the issues associated with
them and how an attacker might be able to use them. (UWS-09)
73 – GREAT! Teaches you how to ‘friend’ another piece of code and explains how if you ‘friend’
a very public piece of code, anyone could gain access to information you’re trying to keep
private. (UWS-07) (UWS-09)
86 – Mentions how variables do not have a default initialization variable, so it is not safe to
assume that they will be a value before you assign one to them. This could create strange
scenarios for a developer used to using Java. (UWS-09)

Chapter 5:
91 – Teaches how to overload operators, but doesn’t warn the developers to be careful when
using other people’s objects. Depending on whether an operator was overloaded or not could
make major changes to how you use their code. (UWS-02) (UWS-09)
94 – Teaches how to use BigInteger, but by repeated wrapping objects in BigInteger (such as in
a loop), you could leave your code open to a timing attack due to the slowness of the BigInteger
construction or you could crash your program by using up all of the memory. (UWS-08) (UWS-
02)

Chapter 6:
124 – Because C++ doesn’t have the runtime checks that Java does, using incorrect type
conversions can result in strange values being manipulated and even accessing other areas of
memory that you didn’t mean to use. (UWS-09)

Chapter 7:
147 – Teaches the developer how to create/use templates for code development. It would be
good to mention though that just as your source code needs to be kept private and secured,
these templates need to have the same level of security. They give out a lot of information
about your code to anyone who reads them. (UWS-06)

Chapter 8:
156 – Mentions that the code they have written will have ‘bizarre output’, but doesn’t explain
that it is because they are accessing unallocated memory which could give information to an
attacker about what is stored elsewhere on the machine. (UWS-05) (UWS-09)
157 – Buffer overflow! It mentions how C++ doesn’t check that you haven’t gone beyond the
end of the array, but not how dangerous this can be. (UWS-09.1) (UWS-03)

Chapter 9:
177 – No mention of being careful which files are read or written to. If you don’t do checking on the filename your system is VERY vulnerable. (UWS-10)
177 – They mention how their code can lead to buffer overflow, which is GREAT, but they don’t mention to sanitize input (especially input from the user). (UWS-09.1) (UWS-10)
178 – They mention how the ‘offset’ variable can get you beyond the buffer (in the positive or negative directions even!), but they don’t explain how this is dangerous or how to avoid it. (UWS-09.1)

Chapter 10:
197 – Doing excess calls to mcp[k] (where mcp is a map and ‘k’ is a key not in the map) could result in greatly slowing down access to the map and even filling up memory with the default value. (UWS-02) (UWS-09)
200 – They mention how the default algorithm is quicksort rather than mergesort and how O(N log N) worst-case is not guaranteed, but they don’t mention how to prevent the awful worst case of quicksort! (UWS-29) (UWS-09)

Chapter 11:
208 – Mentions again how there is no index checking on arrays. Doesn’t mention the dangers associated with that. (UWS-09.1) (UWS-03)

Chapter 12:
230 – The simple text substitution could create HUGE issues should someone get sloppy with their formatting (such as leaving out parenthesis), and it’s great that they mention it. It would be nice though if they had really pushed how big of an issues this was. (UWS-08.1) (UWS-09)
235 – They teach printf and how it isn’t type safe, but they don’t mention anything about vulnerabilities (such as format string attacks) that are prevalent with printf. (UWS-09.2)
241 – GREAT! They mention how sprint is dangerous because it could cause a buffer overflow and how sscanf is dangerous especially if the target tokens are strings. However, it would be better if they mentioned what sorts of values/situations create issues and how to prevent them. (UWS-09.1)
241 – They say that if you allocate memory with ‘new’ you must release it with ‘delete’ and not with ‘free’ otherwise “havoc results”. What is this ‘havoc’? (UWS-09)
244 – They show how you need to have a ‘NULL’ at the end of your variable number of arguments, but nothing about what might happen should you forget it. (UWS-09)

Chapter 13:
248 – GREAT! This paragraph covers a lot of important concerns.
256 – Native methods have the power to alter the values in fields, even final fields, and an attacker could definitely take advantage of this should they gain access to your C++ code. (UWS-10) (UWS-09)