APPENDING SECURITY THEORIES TO PROJECTS IN UPPER-DIVISION CS COURSES

Vahab Pournaghshband[1] and Hassan Pournaghshband[2]

[1]Computer Science Department, University of San Francisco, USA
[2]Software Engineering & Game Development Department
Kennesaw State University, USA

ABSTRACT

Software systems have been under continued attacks by malicious entities, and in some cases, the consequences have been catastrophic. To tackle this pervasive problem, the academic world has significantly increased the offering of computer security-related courses during the past decade. In fact, offering these courses has become a standard part of the curriculum for many computing disciplines. While many proposals suggest adding this appealing topic into the non-security CS courses, many faculties do not entirely support the idea for a convincing reason. They rightfully claim that each one of these courses is already packed with concepts and materials developed toward that course, leaving not much room for other topics. In this study, we show how exposing students to security concepts can be incorporated into upper-division CS courses without increasing the normally required efforts needed by students as well as the instructor. We show how to develop a project of this nature that can be appended to an already existing course project. We have successfully employed our proposed approach in two of our core CS courses and present them in this paper as case studies.

KEYWORDS

Computer Science Education, Computer Security, Security Mindset.

1. INTRODUCTION

In recent decades, software has become a critical element in our lives. In parallel, the hackers have become increasingly capable of interrupting these technological connections by breaching software security. New threats are further emerging as computers become more embedded and play more intimately a big role into our environment and daily lives, as for the recent security vulnerabilities found in mobile medical devices [1]. In 2017, an attack was reported to a connected computer globally for every 39 seconds on average [2]. In the US, the number of data breaches increased by 900% in 2019 compared to 2005 [3, 4]. In the past year, over 500,000 Zoom account credentials were hacked and made available on the Dark Web [5]. In another instance, in the past year, a critical security flaw in WhatsApp was exploited, enabling hackers to install surveillance software on users' smartphones. This incident may have impacted WhatsApp's 1.5 billion users [6].

Computer security problems of this kind, and a variety of different ones, support the fact that it is critical for our students to gain the necessary skills and knowledge for handling them [7]. With this in mind, incorporating security into CS courses would have a positive effect on software security vulnerabilities, the most common cause of software security breaches. Consequently, a national-scale and critical unmet need exist for including fundamental principles of security in the

David C. Wyld et al. (Eds): CSTY, SIGI, MaVaS - 2021
pp. 27-37, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.112203

development of algorithms, software system implementation, and networked and mobile systems in primary undergraduate instruction. These days, undergraduate computer science students are not typically exposed to the rigorous constraints that security must impose in software development. The lack of experience and preparedness by students creates an increasing cost to the industry for not only training software engineers, also dealing with large risks associated with security weakness in their deployed products. Computing security is a central constraint in design, implementation, and usage of operating systems, database technology, networking, and other disciplines. Therefore, it must be integrated into each component of their curriculum. Among the reasons for the absence of security fundamentals in core undergraduate computing courses and curricula, one is the fact that there are relatively few faculties who bring a secure computing research background to a typical computer science department.

This paper discusses the challenges in teaching computer security and proposes a novel approach that can be effectively employed in teaching CS courses. This is a straightforward approach and is mainly based on introducing a security addition to an already existing project in non-security courses. It is important to note that, while the curriculum may include an elective upper-division security course, or have it incorporated in an introductory course, our intention is not to replace those courses, but rather to supplement them.

While basic security concepts can be taught in foundational programming courses (CS1), learning and absorbing the security concepts creditably requires advanced knowledge of computer science. This is why we believe that teaching them in upper-division courses complements what has been taught in introductory CS courses. Also, it is important to note that since security is not a standalone topic, its concept should be taught in context to be effective. These reasons are behind our strong belief that security should be taught in conjunction with the respective primary topics in non-security CS courses.

In this study, we show how exposing students to security concepts can be incorporated into upper-division CS courses without ever-increasing the efforts normally required by students and the actions needed by the instructor. We show how to develop a project of this nature that can be appended to an already existing course project. We have successfully employed our proposed approach in two of our core CS courses and present them in this paper as case studies.

The paper is organized as follows: Section 2 presents Related Work. Methodology is presented in Section 3 followed by Case Studies in Section 4. Section 5 concludes the paper.

2. RELATED WORK

There have been numerous works in literature focusing on computer security education [8-14]. In our prior work [8], we pragmatically examined major issues of teaching the security mindset in the early stage of programming by demonstrating how to teach the security mindset through a set of carefully crafted examples in lectures. Additionally, there have been studies emphasizing security concepts incorporation in early computer programming courses [8, 15, 16]. In this regard, educators have conducted various studies dealing with the challenge of how to incorporate security education into the undergraduate curriculum of the computer science or related discipline [17]. While some propose approaches for teaching a single course covering security concepts [17], some recommend effective track methods where a sequence of specialized security courses is offered [18]. As stated in [19], teaching security concepts and security mindset should be emphasized throughout the student's undergraduate program. Authors in [8, 20] and others have echoed this view, offering various teaching approaches.

3. METHODOLOGY

In this section, we present and discuss the methodology applied for our approach. We show how exposing students to security concepts can be incorporated into upper-division CS courses without increasing the required efforts normally needed by students as well as the instructor. We achieve this goal by developing an additional design phase that is security in nature, to an already existing course project. The project design phase is a combination of implementing both attack and defense mechanisms to the system implemented by the students in the original project. Since the project specification should be self-contained, it must include all the security terminologies, definitions, and preliminaries relevant to and required for the project. The main reason behind the project specification being self-contained is not requiring in-class discussions on security topics. This is because (1) the instructors might not be security experts, and (2), as the instructors rightfully claim, each one of these courses is already packed with concepts and materials developed toward that course, leaving not much room for other topics. The project specification also provides additional resources pertaining to security which enables students to conduct independent research in learning additional security concepts they may need in order to complete the project. Moreover, by appending a security phase to existing projects, our approach is easily adaptable to course projects currently employed in various upper-division courses.

The followings are the underlying rationale behind why appending a design project phase of security is naturally effective:

- It allows the students to make design choices. This way, they understand why one approach or algorithm is better than the other. For instance, through these projects, the students learn that AES encryption is more secure than DES encryption. As another example, they learn when to use public-key vs. private key encryption.
- Most curricula in introductory programming courses give students the impression that software is mostly correct, and security needs to be added to systems later, which overlooks the ‘secure first’ view. However, in our approach, we have a different objective by making the security design phase the last phase. We want the students to learn the principle of keeping security in mind from the beginning by understanding the complexity of adding it afterward.
- Design problems naturally lead to various solutions to the same problem based on inherently different design choices. Through end-of-semester in-class presentations, the students learn about other attack prevention or mitigation approaches presented by their classmates.

4. PROJECT DESIGN PHASE EXAMPLES: CASE STUDIES

In this section, we present four design projects that we have developed for the implementation of our proposed design project approach. These may be used as examples of how to add security components to already existing projects. We have recently incorporated these projects into our upper-division courses. We have done it for a senior-level operating system course and a junior-level computer network course. These courses were selected since they are presented to students who will be entering the industry workforce or transitioning to graduate computer science programs.

4.1. Operating Systems: Shell

4.1.1. Attack and defend your shell

The shell you implemented in the first phase of this project is a relatively functional shell compared to bash, but nevertheless has underlying security flaws. Identify these vulnerabilities and consequent potential attacks on the system running your shell that can be launched in your shell by a malicious user. Find them, identify to what extent these vulnerabilities can harm the system, and propose and/or implement a fix or mitigation technique to defend against such attacks. Some of these vulnerabilities are rather straightforward compared to subtle ones. For instance, use of printf and strcpy in your code already pose risks into your shell.

Extra credit: Exploit the vulnerabilities you have found in your lab1ab implementation to do something extremely harmful to the system running your shell (you may want to run it in a virtual machine.).

4.1.2. Process-Overload Attack

In an overload attack, a shared resource or service is overloaded with requests to such a point that it is unable to satisfy requests from other users. One of the simplest denial of service attacks is a process attack. In a process attack, one process or a user makes a computer unusable for other processes or users to run. The following program will probably paralyze or crash your system if your shell executes it, specially in root.

```
while(1)
      fork();
```

You will need to design a way to mitigate such attacks or perhaps not even let them happen. Find a way to secure the system using your shell from such attacks. (Hint: killing the rouge parent process will not solve the problem. You can try it).

4.1.3. Code Shell Injection

You will extend your implementation to introduce shell variables arguments to your ./timetrash, similar to how shell scripts handle arguments.

In that case, your ./timetrash (without -p or -t options) can have more arguments, which will be accessed by some of the commands listed in the script.sh.

Here is an example: `./timetrash script.sh myfile.txt cs`

Where the content of script.sh is: `cat $1 sort < $1 echo $2`

However, this feature can lead to security problems. In fact, a malicious user can inject and execute arbitrary commands. For instance, it can do some harm by running ./timetrash with the following arguments:

```
./timetrash script.sh "myfile.txt;rm -f /" cs
```

Or,

```
./timetrash script.sh "myfile.txt;mail evil@evilmail.com < /etc/shadow" cs
```

You are to implement variable shell arguments in a safe way to not let arbitrary commands to be executed.

4.2. Operating Systems: File Systems

4.2.1. Race Condition Attack

File systems are susceptible to race condition attacks on file systems when the following flaw in process behaviour is exploited: when a process performs a sequence of operations on a file, it assumes that the file does not change between any two successive operations. In this case, an attack can manifest itself by changing the file during the time window between two successive operations on it by a victim process. This temporal window is known as race window. There are two scenarios in which race condition in file systems may lead to potential damages: (a) the victim process operates on the changed file, leading to damage, or, (b) information is leaked/written from/into the file illegally if the victim's operations allow the attacker to get permissions on the file during the race window. Dictionary redirection attack, filelogger attack, and readfile attack are a few scenarios of race condition attacks on filesystems [21].

One way to prevent race condition is to use locks, which in this case could be called file locking. You are to design and implement file locking or any other approach that would prevent the attacks presented above in your implemented filesystem.

4.2.2. Filesystem-Level Encryption

As long as the operating system is running on a system without file encryption, access to the files will have to go through OS-controlled user authentication and access control lists. If an attacker gains physical access to the computer, however, this barrier can be easily circumvented. One way would be to remove the disk and put it in another computer with an OS installed that can read the filesystem, or simply reboot the computer from a boot CD containing an OS that is suitable to access the local filesystem.

The most widely accepted solution is to store the files encrypted on the physical media (disks, USB pen drives, tapes, CDs and so on). Recent operating systems have allowed users to store data encrypted on the filesystem (e.g., eCryptfs and EFS). Such systems provide both data and filename encryption on per-file basis. This is in contrast to full disk encryption where the entire partition or disk, in which the file system resides, is encrypted. In filesystem-level encryption, the file or folder is readable and writable only if the user provides the right password associated to that file or folder when they want to access it. Once implemented, this would enable files to be transparently encrypted to protect confidential data from attackers with physical access to the computer. By default, no files are encrypted, but encryption can be enabled by users on a per-file or per-directory basis.

You may need to add some information to the metadata of files in the underlying filesystem. This metadata describes the encryption for that particular file that is to be encrypted. It should provide both data and filename encryption on per-file basis.

You should decide which forms of encryption to support. In addition, you should address the security problem of keeping the key used for encryption in plain-text format in RAM, and how your approach mitigates this problem. In your approach of handling the encryption key in RAM problem, you should also consider the usability of your proposed security system.

4.2.3. Denial-of-Service Attacks on File systems

Denial-of-Service (DoS) attack is an attempt to make a machine unavailable to its intended users. Although the means to carry out, motives for, and targets of a DoS attack may vary, it generally consists of efforts to temporarily or indefinitely interrupt or suspend services on that system. One class of DoS attacks are those targeted for filesystems, specially that in Unix, files are more than just information storage. For instance, devices and sockets are also files.

You are going to propose a design an approach for all of the following particular filesystem DoS attacks and show why your design protects against such attacks.

4.2.3.1. Free space illusion attack

Open files that are unlinked continue to take up space until they are closed. The space that these files take up will not appear with the du or find commands, because they are not in the directory tree; however, they will nevertheless take up space, because they are in the filesystem.

For example:

```
int ifd;
char buf[8192];
ifd = open("./attack", O_WRITE|O_CREAT, 0777);
unlink("./attack");
while (1)
      write (ifd, buf, sizeof(buf));
```

Files created in this way can't be found with the ls or du commands because the files have no directory entries.

Hint: To recover from this situation and reclaim the space, you must kill the process that is holding the file open.

4.2.3.2. Deep directory attack

It is also possible to attack a system by building a tree structure that is made too deep to be deleted with the rm command. Such an attack could be caused by something like the following shell file:

```
$!/bin/ksh
$
$
Don't try this at home!
while mkdir anotherdir
do
      cd ./anotherdir
      cp /bin/cc fillitup
done
```

On some systems, rm -r cannot delete this tree structure because the directory tree overflows either the buffer limits used inside the rm program to represent filenames or the number of open directories allowed at one time.

4.2.3.3. Empty file attack

The UNIX filesystem uses inodes to store information about files. One way to make the disk unusable is to consume all of the free inodes on a disk, so no new files can be created. A person might inadvertently do this by creating thousands of empty files. This can be a perplexing problem to diagnose if you're not aware of the potential because the df command might show lots of available space, but attempts to create a file will result in a "no space" error. In general, each new file, directory, pipe, FIFO, or socket requires an inode on disk to describe it. If the supply of available inodes is exhausted, the system can't allocate a new file even if disk space is available.

You can tell how many inodes are free on a disk by issuing the df command with the -i option:

```
$ df -i
Filesystem    Inodes      IUsed       IFree     %IUse       Mounted on
/dev/sdb5    7569408     170084     7399324        3%       /
```

The output shows that this disk has lots of inodes available for new files.

Now if you run

```
$ touch empty_file
```

And run df -i again, you will see that it will increase the number of used inodes, even though it is just an empty file.

4.3. Operating Systems: Ramdisk

4.3.1. Encrypted Ramdisk

Recent operating systems have allowed users to store data encrypted on the filesystem. The filesystem is readable and writable only if the user provides the right password when they log in. Implement a software-encrypted ramdisk, where data is stored on the ramdisk in encrypted format. If a user opens the ramdisk normally, they should see encrypted gobbledegook. But if a user provides the right password at open time, then read operations on that open file should transparently decrypt the disk's data. Furthermore, if the user writes to the file, the data they write should be encrypted before it is sent to the ramdisk.

You will need to implement ramdisk-specific read and write operations for this design problem. (You probably need to change the osprd_blk_fops structure's read and write operations to point to your code.) You should decide which forms of encryption to support.

In addition, you should address the security problem of keeping the key used for encryption in plain-text format in RAM, and how your approach mitigates this problem. In your approach of handling the encryption key in RAM problem, you should also consider the usability of your proposed security system.

4.3.2. Partitioned Ramdisk

RAM was recently shown to be vulnerable to attacks exposing the totality of memory, including sensitive user data and encryption keys.

One way to mitigate data exposure is to divide the RAM into two partitions: protected partition to store sensitive data and a public partition to store everything else. This somewhat resembles partitioning a harddisk, with one difference: There is a hard boundary between the two partitions such that read and write accesses to the protected partition is only allowed by privileged processes.

This does not necessarily mean that the protected partition has to be encrypted. In fact, sometimes it is preferred not to be. One reason is that on-the-fly encryption and decryption often slows down access to memory (which defeats the purpose of using ramdisk for performance and speed in the first place). Also, some sensitive information should not be encrypted when in use anyways, such as encryption keys and passwords.

In your design, you should think about where and how in memory the information about the partitions you define is going to be kept. You should also not let the protected area of RAM to be swapped to disk, since this partition is designed to hold a variety of highly sensitive data, which will be exposed if it is swapped to harddisk. Even by adding such security, some form of physical attack, called cold boot attack, has proven to expose data in the memory even when the system is powered off.

By incorporating your design can we stop unprivileged processes from reading the content of the protected memory using “memory viewers”? Is it possible to protect against cold boot attack?

(Hint: One design to mitigate this is to introduce privileged read and write system calls)

4.4. Computer Network: Peer-to-Peer File Sharing

4.4.1. Authentication and Authorization

Authentication is the process carried out by an entity to confirm the identity of another entity or to confirm that a data is indeed from whom it claims to be. Passwords, digital signatures or message authentication codes are the common techniques of authentication. Authorization on the other hand, is the process of granting to the users some privileges on the access to a set of resources according to what is permitted to them. The most popular techniques to enforce authorization are to maintain access control lists (ACL) listing the access rights of entity.

The peer is happy to serve other peers any data contained in its current directory. Fancier programs, such as real web servers, allow users to specify which files in a directory can be served. For example, this syntax tells Apache to refuse to serve the osppeer.c file:

```
<Files "osppeer.c"> Order allow,deny Deny from all </Files>
```

This syntax would usually go in a file called .htaccess, in the directory containing osppeer.c.

Design access control syntax for our peers. Will you support Apache-style .htaccess files, or something else? What type of syntax will you support? For full credit, you should design a very flexible access control syntax. Consider such issues as limiting access for some files to limited sets of peers, defined based on (say) network address; symbolic links; and so forth.

In addition to authorization, your design should also authenticate the peer via some ACL mechanism you will design, so that you serve the file only if the peer is authenticated and is authorized to download the file from you.

4.4.2. Transmitting Encrypted Files

Extend our current design to allow peers to send encrypted data. You should consider three types of encryption. In increasing order of safety:

1. Hiding file contents from network snoopers.
2. Hiding file contents from unauthorized peers. I.e., if a peer does not know the right key(s), then the peer will not be able to understand a download file.
3. Hiding the existence of a file from unauthorized peers. I.e., if peer 1 does not know the right key(s), then peer 1 cannot tell which files peer 2 has made available. (Perhaps peer 1 will be able to tell that peer 2 has registered 5 files with weird scrambled names, but peer 1 cannot tell what those files' true names are, and peer 1 cannot download their data -- encrypted or not -- from peer 2.)

For each of the items mentioned above, you should address what key management mechanism you put in place in this peer-to-peer network. In doing so, your design should handle scenarios such as where the key is compromised or a scenario where peer 1 would like to revoke peer 2's access to a particular file it is sharing, while peer 3 and peer 4 would still be able to access the file, as they originally have. How would your encryption system differ in such cases?

Extra credit: Confidentiality is reached when data is protected from unauthorized disclosures, whereas integrity is means that data is safe from unauthorized modifications. Encryption is a powerful guarantee of confidentiality, while data integrity can be achieved using digital signatures and message authentication codes. Replay attacks involve a malicious user injecting old data on the system. In order to guarantee freshness, timestamps or nonces can be used, but these tools require a certain degree of synchronization between the entities. How can you extend your cryptographic protocol to ensure authenticity of content? (Hint: one way to achieve that is through signed public-key certificates by trusted publishers)

5. CONCLUSIONS

While software has become a crucial element in our lives in recent decades, and consequently, software systems have been under continued attacks by malicious entities. As a result, in recent years, the offering of computer security courses in universities has increased. While many proposals suggest adding this appealing topic into the non-security CS courses, there are several challenges in implementing them effectively. We addressed those challenges by presenting a novel approach to show how to develop a project of security nature that can be appended to an already existing course project. We employed our proposed approach in two of our Operating Systems and Computer Networks courses and present the details as case studies.

REFERENCES

[1] Vahab Pournaghshband, Majid Sarrafzadeh, and Peter Reiher (2012) “Securing legacy mobile medical devices”, *In Proc. of 3rd International Conference on Wireless Mobile Communication and Healthcare* (MobiHealth ’12). Springer.

[2] The Security Magazine (2017) “Hackers Attack Every 39 Seconds.”, https://www.securitymagazine.com/articles/87787-hackers-attack-every-39-seconds.

[3] DataStealth (2020) “110 Must-Know Cybersecurity Statistics for 2020”, https://www.datex.ca/blog/110-must-know-cybersecurity-statistics-for-2020.

[4] Statista (2021) “Number of compromised data records in selected data breaches as of January 2021”, https://www.statista.com/statistics/290525/cyber-crime-biggestonline-data-breaches-worldwide/.

[5] Forbes.com (2020) "Hacked Zoom Accounts Given Away For Free On The Dark Web", https://www.forbes.com/sites/leemathews/2020/04/13/500000-hackedzoom-accounts-given-away-for-free-on-the-dark-web/#6e67407a58c5.

[6] BBC Technology (2019) "WhatsApp discovers targeted surveillance attack", https://www.bbc.com/news/technology-48262681.

[7] Computing Curricula (2017) "Curriculum guidelines for post-secondary degree programs in cybersecurity" *IEEE Computer Society Association for Computing Machinery,* https://doi.org/10.1145/3184594

[8] Vahab Pournaghshband (2013) "Teaching the security mindset to CS1 students", *In Proceeding of the 44th ACM technical symposium on computer science education* (Denver, Colorado, USA) (SIGCSE '13). ACM, 6. http://doi.acm.org/10.1145/2445196.2445299

[9] Debarati Basu, Harinni K. Kumar, Vinod K. Lohani, N. Dwight Barnette, Godmar Back, Dave McPherson, Calvin J. Ribbens, and Paul E. Plassmann (2020) "Integration and Evaluation of Spiral Theory Based Cybersecurity Modules into Core Computer Science and Engineering Courses" *In ACM SIGCSE '20* Association for Computing Machinery, 9–15.

[10] Trajce Dimkov,Wolter Pieters, and Pieter Hartel (2011) "Training students to steal: a practical assignment in computer security education", *In Proceedings of the 42nd ACM technical symposium on computer science education*. ACM, 21–26.

[11] Tadayoshi Kohno and Brian D Johnson (2011) "Science fiction prototyping and security education: cultivating contextual and societal thinking in computer security education and beyond", *In Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 9–14.

[12] Peter Peterson and Peter L Reiher (2010) "Security Exercises for the Online Classroom with DETER", *In Proceedings of the 3rd international conference on Cyber security experimentation and test* (CSET'10).

[13] Valdemar Šv.benský, Jan Vykopal, and Pavel Čeleda (2020) "What Are Cybersecurity Education Papers About? A Systematic Literature Review of SIGCSE and ITiCSE Conferences", *In Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) (SIGCSE '20). 2–8.

[14] Bernard Yett, Nicole Hutchins, Gordon Stein, Hamid Zare, Caitlin Snyder, Gautam Biswas, Mary Metelko, and Ákos Lédeczi (2020) "A Hands-On Cybersecurity Curriculum Using a Robotics Platform" *In ACM SIGCSE'20*. Association for Computing Machinery, 1040–1046.

[15] Vahab Pournaghshband and Hassan Pournaghshband (2021) "Teaching Security Notions in Entry-Level Programming Courses", *In Proc.* of *IEEE International Conference on Teaching, Assessment, and Learning for Engineering* (IEEE TALE '21). IEEE.

[16] K. Nance (2009) "Teach Them When They Aren't Looking: Introducing Security in CS1" *In IEEE Security Privacy* 7, 5 (Sept 2009), 53–55. https://doi.org/10.1109/MSP.2009.139

[17] Luiz Felipe Perrone, Maurice Aburdene, and Xiannong Meng (2005) "Approaches to undergraduate instruction in computer security", *In Proceedings of the American Society for Engineering Education Annual Conference and Exhibition,* ASEE.

[18] S. Azadegan, M. Lavine, M. O'Leary, A. Wijesinha, and M. Zimand (2003) "An Undergraduate Track in Computer Security", *SIGCSE Bull.* 35, 3 (June 2003), 207–210. https://doi.org/10.1145/961290.961568

[19] Ambareen Siraj, Nigamanth Sridhar, John A. Drew Hamilton, Latifur Khan, Siddharth Kaza, Maanak Gupta, and Sudip Mittal (2021) "Is There a Security Mindset and Can It Be Taught?" *In Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy* (CODASPY'21). Association for Computing Machinery, New York, NY, USA, 335–336. https://doi.org/10.1145/3422337.3450358

[20] Kara Nance and Brian Hay (2009) "Computer Security-focused Programming Assignments in Foundational CS Courses" *In Proceedings of the 13th Colloquium for Information Systems Security Education*.

[21] Prem Uppuluri, Uday Joshi, and Arnab Ray. (2005) "Preventing race condition attacks on file-systems", *In Proc. of the ACM symposium on Applied computing* (SAC '05). Association for Computing Machinery, New York, NY, USA, 346–353.

AUTHORS

Vahab Pournaghshband is an Associate Professor in the Computer Science department at University of San Francisco. He received his Ph.D. in Computer Science at University of California, Los Angeles (UCLA) in 2014. He received his M.Sc. in Computer Science from University of California, Berkeley. Also from UC Berkeley, he received his B.Sc. in Electrical Engineering and Computer Science. Dr. Pournaghshband was a recipient of Fulbright US Scholar Award and has published numerous papers and has made numerous national and international professional presentations at different conferences. His current area of research are Computer Networks, Computer Security, and Computing Education.

Hassan Pournaghshband, recipient of the Kennesaw State University 2021 Outstanding Teaching Award, is a professor of software engineering in the department of software engineering and game development at KSU. Before joining KSU he taught at City University of New York, Hofstra University, University of Tehran, and University of Missouri. He earned his bachelor's degree in Economics from University of Tehran, his Masters of Computer Science from Northwestern University, and his Ph.D. in Computer Science from University of Oklahoma. He has published numerous papers and has made numerous national and international professional presentations at different conferences. His current areas of research are Software Architecture, Software Project management, Software QualityAssurance and testing, and Database Management Systems.