

Design and Use of a Secure Testing Environment on Untrusted Hardware

Martin C. Carlisle and Leemon C. Baird III

Abstract—Dedicated laboratories have been used to provide a secure testing environment. Given the prevalence of laptop computers, the cost of maintaining a computer lab simply for testing seems prohibitive. We present the RAPTOR assessment environment, which allows students to take an exam on their own laptop, without being able to access their hard drive or network resources. We evaluate threats, and the level of security provided by the assessment environment.

Index Terms—D.4.6 Security and Privacy Protection, D4.6c Cryptographic controls, D4.6d Information flow controls, D.2.6.b Graphical environments, D.1.7 Visual Programming.

I. INTRODUCTION

PROVIDING an accurate assessment of a student's mastery of course material is an important part of higher education. Traditionally, this has been done through homework and testing. Since homework is not done within a controlled environment, it may not provide an accurate reflection of the submitter's understanding (either because of plagiarism, or receiving excessive help on the assignment). The controlled environment of tests yields a greater confidence that they accurately assess student ability.

Programming tests are often done on computers, as students are then able to check their answers by compiling and running them. Previously, dedicated laboratories were used for student testing. These machines could be specially configured to guarantee that students did not have access to unauthorized resources. Given the prevalence of laptops, many universities have determined it is no longer cost-effective to maintain dedicated labs. Therefore, we are faced with the problem of guaranteeing that students do not have the ability to bring extra materials into a test when they are doing it on their own computer.

Manuscript received March 21, 2007.

Martin C. Carlisle is with the Department of Computer Science, United States Air Force Academy, USAFA, CO 80840 USA (phone: 719-333-3590; fax: 719-333-3338; e-mail: carlisle@acm.org).

Leemon C. Baird III is with the Department of Computer Science, United States Air Force Academy, USAFA, CO 80840 USA (email leemon@leemon.com).

Our secure assessment environment is built on top of RAPTOR [1], a visual programming environment. It allows tests where the student can write RAPTOR programs and test them, but cannot access any other programs during the test, and also cannot use any files on their hard drive or email or other network resources.

Section 2 describes previous work on secure testing software. We describe how to use the RAPTOR Assessment Environment in Section 3. In Section 4, we describe the threat model. Section 5 describes how our assessment environment addresses these threats. Section 6 gives conclusions and possibilities for future work.

II. PREVIOUS WORK

At least two companies distribute secure testing software. Both Secureexam™ [2] and Respondus Lockdown Browser™ [3] work on the same principle. They provide a web browser that has been specifically configured to:

- fill the entire screen
- not allow switching applications
- not allow access to other network resources beside the exam

We examined the documentation for both Respondus Lockdown Browser and Secureexam and evaluated the Secureexam demo [4]. Both of these systems rely on network access, and are run from within a regular Windows session. Our approach instead uses a restricted version of Windows with no network drivers. We believe this restricted environment has fewer moving parts, and is therefore easier to secure. The Secureexam FAQ claims "in the unlikely event of a breakout, [Secureexam] will detect and record the violation on the encrypted exam." We did not attempt to break the encryption to validate this claim.

We did notice these features:

- disable the right-mouse button
- disable the function keys
- disable the task manager button from the screen that appears when Ctrl-Alt-Del is pressed.

These changes make it difficult for a user to break out of the testing environment.

Both of these packages require a per student, per semester fee. Our solution is entirely based on software that is available to universities at no cost (with the obvious exception

of Microsoft Windows, which we assume each student will have already bought). There is no cost to the university or the student. Also, the environment will run on a wide range of student laptops, requiring only a Windows machine with at least 512MB of RAM.

Another possibility might be virtualization. VMWare ACE [11] is an example of a software product that allows you to create secure virtual desktops that are centrally managed. While this would allow complete control of the virtual environment, there is nothing that would prevent students from accessing a non-virtual environment during the exam.

Finally, classroom management software such as SynchronEyes [12] allows an instructor to monitor what students are doing on their computers, as well as allowing the instructor to block particular applications. SynchronEyes also provides a testing capability. This testing is limited, however, as it would not allow students to compile and run programs. Further, if you unblock an IDE to allow this, then students will be able to access any programs they have stored on their hard drives.

III. USING THE RAPTOR ASSESSMENT ENVIRONMENT

Our goal was to allow the instructor to hand out a test on paper, which asks the student to write several small programs, then give the students an electronic environment in which to write those programs during the test. In this way, there is no additional burden on the instructor to create the test questions in any particular format, or to enter them into any particular electronic system. Since the test environment is not specific to a particular test, it can be reused throughout the semester.

To start the RAPTOR Assessment Environment, the student simply inserts the CD in the computer. The autoplay feature immediately starts a 15 second countdown and reboots the machine. If the student has autoplay disabled, or it otherwise fails to reboot, they should simply restart their machine. Once the machine reboots, it should boot off the CD into the assessment environment. If not, they will need to set the BIOS to boot from the CD first, or select the CD from a boot menu. After it loads, it will automatically start a copy of RAPTOR:

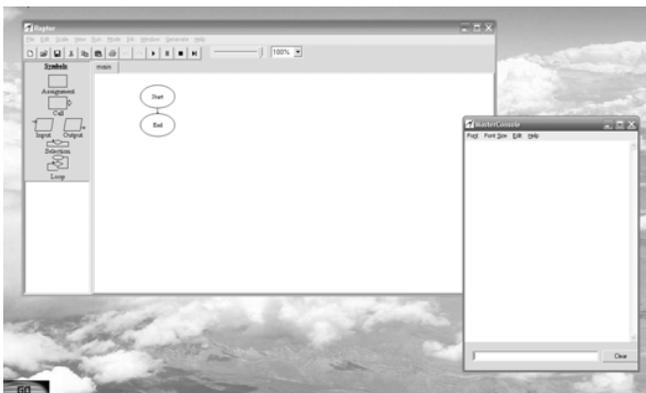


Figure 1: RAPTOR Assessment Environment

As shown in Figure 1, when the environment starts, it displays a distinctive background; there are no desktop icons and no “Start” button or taskbar (a “Go” button appears in the lower left corner). This makes it easier for a proctor to ensure that a student is using the environment.

Students complete assignments as usual. The main differences a student will observe are that the file open and file save dialogs have been replaced. Figure 2 shows the open dialog, which allows the student to browse only the RAM disk:



Figure 2: RAPTOR Assessment Open Dialog

The Save dialog similarly restricts the user to a single folder, and allows only letters and digits in a filename. This dialog is shown in Figure 3. Files are saved unencrypted to the RAM Disk (so that the student can open and close files during the test), but are also stored encrypted on the hard drive. For example, problem1.rap would be saved on the hard drive as “c:\problem1.rap.aes”. A message indicating where the encrypted file was saved is displayed in the console window each time the student saves their file.

Once the assessment is over, the student reboots into ordinary Windows and submits their files either over the network, or by giving them to their instructor on a USB flash drive. In our setting, students may have to rush to their next class, and therefore submit the files later. This creates a period of time where students have access to the encrypted solutions in a non-proctored setting. It also raises the possibility that students might attempt to create encrypted solution files after the exam but before turning it in.

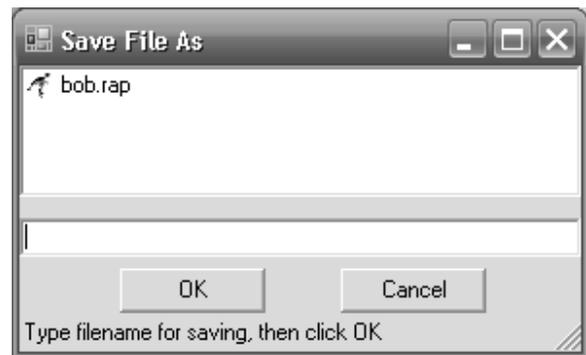


Figure 3: RAPTOR Assessment Save Dialog

IV. THREAT MODEL AND ASSUMPTIONS

We are particularly interested in guarding against the following:

- Students being able to access their hard drive or network resources during the exam
- Students being able to give their solutions to another student to turn in
- Students being able to modify their solutions after the assessment but before submitting them to the instructor
- Students being able to take the test at a later time and then submitting that work instead of what they did during class

We assumed that students would take the test in a proctored environment, but that network services would not necessarily be available. Their solutions would be saved to their hard drive, and they would later connect to the network and submit their files.

We also assume that students will not be able to reverse engineer and modify compiled executables, and will not be able to crack either RSA keys [5] or AES encryption in electronic codebook mode (ECB) [6]. Additionally, despite MD5 having been broken by researchers [7], we still consider it sufficiently secure for detecting if the environment files have been tampered with. (We believe it would be simpler to reverse engineer the compiled executable and remove the hash checks than to create working system files that have an MD5 hash collision).

V. ADDRESSING THE THREATS

We now turn our attention to how students might attempt to subvert the assessment environment, and what steps we take, either through software or through the administration process, to counter these threats. These techniques are not unique to RAPTOR, and we anticipate very similar changes could be made to other software programs to allow them to be used in a testing environment.

A. Accessing Unauthorized Resources

We do not want students to be able to access old programs from their hard drive, interact with each other through email or instant messaging, or look for algorithms or information on the internet. We provide a restricted version of Windows by using Bart's Preinstall Environment (BARTPE) [8]. BARTPE allows us to create our own live CD version of Windows, with custom menus and programs. Our restricted version of Windows has no network drivers, which should prevent network access. USB autoplay is disabled, which prevents access to files on a USB flash drive. Additionally, we replace the start menu with a menu that contains only options for RAPTOR, killing a locked RAPTOR process, changing the screen resolution, shutdown, and restart. This prevents the user from starting any other programs. Also, since we have rebooted into this environment, we are guaranteed that no

previously running programs will put windows on top of the assessment environment while it is running.

There are three additional possibilities that need to be considered. The first is whether the RAPTOR program has features that would allow the student to either start other programs or access the hard drive. The second is whether the assessment environment is subject to tampering. The third is whether virtualization can be used to defeat the system.

We prevent two ways that RAPTOR could be used to access unauthorized resources. First, as described in Section 3, we replace the File Open and Save dialogs. This is essential as the default dialogs allow browsing filenames (which could be a source of information) and also allow other programs to be executed (such as cmd.exe, answers.doc or iexplore.exe) by right clicking on their names. This is why Secureexam and Respondus Lockdown Browser both disable right-clicking. Also, by restricting filenames to only letters and digits, we avoid attacks such as entering "c:" as a filename and escaping the controlled environment. Second, we disable the Help feature. It is unfortunate not to have help in the environment, but HTMLHelp is insecure. The most obvious insecurity is the "Jump to URL" menu option, which can be used to access any arbitrary file (using a file URL).

The RAPTOR Assessment Environment does provide a backdoor for instructor use to resolve issues that may arise during administration. This allows an instructor to generate a command prompt. To open the backdoor, one types "emergency" in the textbox at the bottom of the console window. This generates a challenge, as shown in Figure 4.

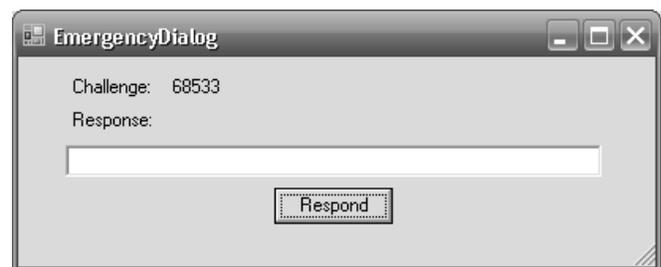


Figure 4: RAPTOR Assessment Open Dialog

If the correct response is entered, a command prompt window will be opened. If not, a different random challenge will be generated. The instructors are given instructions and code for generating the response.

The risk of tampering is lessened by using MD5 hashes of critical system files. When the environment starts, it computes the MD5 hash of several critical system files. If any of the hashes do not match, then the environment will not start. This is only secure if we assume the students will not be able to reverse-engineer the compiled executable, locate the stored hash values and change them to match their tampered files. This is obviously not foolproof, but we consider this an acceptable level of risk. Additional countermeasures that can be deployed against tampering include binary obfuscation,

maintaining physical security on the Assessment Environment CDs (thus reducing the probability someone will be able to get a copy of the environment to reverse engineer) and changing the RSA keys frequently (so that solutions created with tampered copies of environments with the old key will be detected).

Another possibility for defeating the system is to run it from within a virtual machine. The RAPTOR Assessment Environment contains code (from [9]) that detects the two most prevalent virtual machine environments, VMWare and Microsoft Virtual PC. We have verified this also detects Microsoft Virtual Server. If run from within a virtual machine, the program will still run, but will not encrypt any files. The files will also have additional markers that indicate the assessment environment was not used.

B. Providing Solutions to Others

The solutions a student creates during the examination are stored encrypted on the hard drive and unencrypted only on the RAM Disk. Once the test is completed, students are required to reboot into normal Windows. The contents of the RAM Disk will be lost, and they will only have encrypted copies of their files. These files are encrypted using 128 bit AES encryption in ECB mode. Rather than having a single secret (which would then be present in the compiled executable), the environment instead randomly generates a 128-bit key for each save. These are encrypted with an RSA public key, and stored in the header of the saved file. Instructors have access to a program containing the private key, which first reads the key, decrypts it, and then decrypts the remainder of the file using AES.

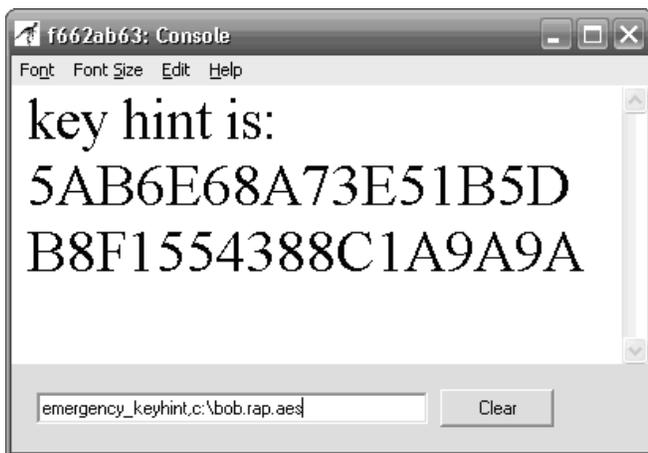


Figure 5: Recovering Files During the Exam

The main advantage to this hybrid scheme is that it allows instructors to more easily recover student files if the student has to reboot during the middle of the exam (e.g. to replace the battery). In such a case, the instructor can use the "emergency_keyhint" command, as shown in Figure 5, to find the encrypted AES key. They can then decrypt it with the

"emergency_decrypt" command.

For example, to restore an encrypted file bob.rap, "emergency_keyhint,c:\bob.rap.aes" will find the encrypted AES key and decrypt this key. The command "emergency_decrypt,c:\bob.rap.aes,b:\bob.rap,BA6DAB3028BCDFA9999F196D3C324EA7" will restore the decrypted file on the RAM Disk so the student can continue working on it. A program on the instructor machine takes in the encrypted AES key and returns the decrypted key.

The security is primarily dependent on the security of the RSA private key. Since each file has a randomly generated AES key, obtaining one of these keys only allows reading that single file. If, however, the RSA private key is compromised, it can be used to decrypt all of the encrypted files.

It is possible that a tampered version of the assessment environment could be designed that stores the unencrypted data on the hard drive. This would allow access only to solutions created using the tampered version. The cryptography would still protect files generated with an unmodified copy. As previously mentioned, MD5 hashes are used to detect possible tampering.

C. Modifying the Completed Test

Once the exam is over, students only have access to encrypted versions of their solutions. Since these solutions have been encrypted using a randomly generated 128-bit AES key, it should not be possible for a student to make any changes without corrupting the file.

D. Redoing the Test Later

If students could obtain a copy of the testing environment, they could reset their system clock and then take the test again later, but before submitting their work.

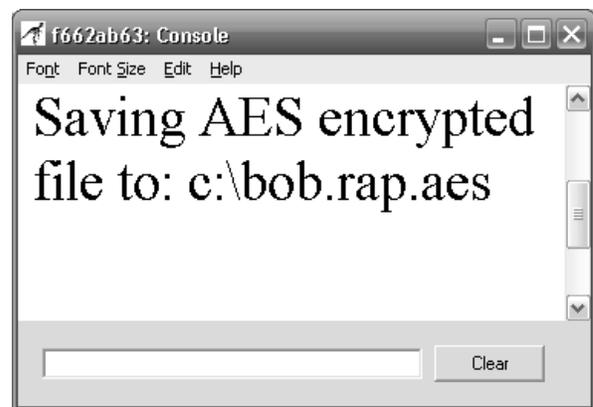


Figure 6: Preventing Time-Shifting the Test

RAPTOR associates a globally unique identifier, or GUID, (Microsoft's implementation of the Universally Unique Identifier standard [10]) with each file. We take advantage of this feature to prevent this time shifting. As shown in Figure 6, a 32 bit section of the GUID is displayed in the titlebar of the console window. Students are required to copy these

numbers on the paper exam, which they then turn in before they leave. On average, that means they would have to hit the new button 2 billion times before they generate a file with the same string.

Tampering is again a concern for this threat. If someone reverse engineered the executable, they could replace the generation of a random GUID with code that prompts for a desired GUID. There are other features of the binary file format that are unique to the assessment environment, but they are not beyond a significant reverse-engineering effort.

VI. EXPERIENCE AND LESSONS LEARNED

The RAPTOR test environment was used in the core computer science course throughout the Spring 2007 semester. This is a required course for every student, regardless of major. Over 600 students in 26 different sections each used the environment for 3 separate in-class assessments. A number of lessons were learned from this extensive real-world testing of the environment.

Perhaps the most significant results were due to the fact that the students were able to run and test their code during the assessment. In the past, they have been required to write their code on paper and hand it in. There were a number of advantages that became apparent for allowing the students to run their programs as they wrote them.

First, it is difficult for beginning students to get a program right on the first try, especially when they have only had a few lessons of instruction and have no prior programming experience. Simple errors in the code that would lead to large point deductions may not be visible to them at first glance, but may be apparent after they've run the code and observed its behavior. This doesn't just lower the grades of all students uniformly. There is a large amount of randomness involved. It appears that the assessment was better able to judge the students' actual knowledge when they were allowed to run the program during the assessment.

Second, an assessment better reflects what a student has learned if it is similar to the homework exercises that preceded it. When students were given homework assignments, they were asked to write small programs, and of course they were able to run them as they wrote them. Therefore, it was better to have an assessment with a similar environment. Otherwise, we would be testing a slightly different skill set than what was taught.

Third, even for small problems, it is useful to teach the students a spiral method of software development. They were encouraged to write the smallest program that would execute, and then repeatedly add one feature at a time, until the full program was done. They were taught to test the software after each spiral, which is good software engineering, but is impossible to do when the entire program must be written out on paper. Furthermore, when writing on paper, it is difficult to insert new code into the middle of old code. This discourages a spiral approach to development. So once again,

an assessment on paper ended up testing slightly different skills than what were taught. An assessment in an executable environment was much more effective.

Fourth, there was a noticeable difference in student attitudes with the interactive environment. There was less frustration, and the assessment had the appearance of being more "fair". The first three points show how the assessment did a better job of assessing, but this fourth point regarding morale is a useful side effect. The students are likely to work harder when they perceive the experience as more enjoyable.

In these ways, the interactive assessment proved to be better than the paper assessments that had been used in the past. In addition, there were several lessons learned.

First, we discovered that it is useful to leave more time before and after the assessment. Initially, we tried a 45-minute assessment during the 53-minute class period. Those 8 minutes of non-assessment time turned out to be barely enough for the students to come in, set up their computers, and boot into the assessment environment. Any students who had technical problems with their machines would find themselves losing time. In the later assessments we switched to a 40-minute assessment with 13 minutes extra. This allowed most students to get set up without any problem.

Second, we found it was useful to encourage the students to submit their results during class. There is a wireless network in the classroom. It isn't always reliable, so we had to design the assessment environment to allow the students to submit their encrypted (and unchangeable) files many hours after the assessment. However, it was often difficult for them to remember to do so, which caused frustration for the both the students and the instructors. When we encouraged the students to submit their files immediately after the test, that made the entire process run more smoothly. Most of them were able to use the wireless network. Most of the others were able to use a USB flash memory to transfer the files to the instructor's computer. The very few students who could do neither in the allotted time were able to submit them later that evening. But since only a few students had to resort to that, there were far fewer problems with people forgetting. This was a large improvement.

Third, we found that it was very useful that the test environment had a recognizable appearance to the START menu button and the desktop image. This meant that after all the students had booted into the environment, the instructor could stand at the back of the room and see that all the computers were in the special environment rather than in normal windows. This fact, combined with the fact that the CDs were always collected and counted at the end of the period, made it significantly more difficult for a student to make copies of the environment that could later be modified. All indications were that the environment was secure in practice. No security problems were seen for any of the 600 students during any of the 3 assessments.

In addition to these educational benefits and lessons

learned, this environment reduced the workload for the instructors.

The instructors were able to run a script that automatically ran each student's program, and reported whether it gave the correct outputs for various inputs. This wasn't sufficient by itself to assign a grade. A program might be almost entirely correct and yet get the answer wrong for every possible input. However, it did save the instructor time, by focusing attention on the important parts. If the program ran correctly, the instructor could focus on programming style without worrying about correctness. If the program ran incorrectly, the instructor again didn't have to untangle spaghetti code to determine whether it was correct. The instructor would already know it was incorrect, and could grade it more quickly. Overall, this partial automation made grading more than twice as fast as it would be with written programs on paper.

In addition, both the instructors and students had tablet computers. The instructors were able to write comments in the margin of the code using a stylus on the tablet. It is very convenient to be able to write comments and draw arrows to the parts of the code being discussed. This made electronic grading as easy as paper grading in that respect. Also, the graded programs could be returned to the students electronically, which saved the instructors some effort, and allowed the students to get their feedback more quickly, rather than having to wait until the next time the class met.

Overall the assessments using the new environment were much better than the old paper-based assessments. They were more accurate, they were less frustrating for the students and instructors, and there were no security problems. The major lessons learned were that it is useful to leave a little more time before and after the assessment, and that it is useful to encourage the students to submit their files as soon as possible before they forget.

VII. CONCLUSIONS AND FUTURE WORK

We have described a secure solution for allowing students to be tested on their own laptops by using a live CD to boot into a restricted environment. We have provided a threat model, and assessed how our countermeasures address these risks. Students are prevented from accessing the network or their hard drives, giving their solutions to others, modifying them after the exam, or taking the exam at a later date. This particular environment supports only creating RAPTOR programs, but could be extended to provide more general testing.

REFERENCES

- [1] Carlisle, M., T. Wilson, J. Humphries and S. Hadfield. "RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving", 36th SIGCSE Technical Symposium on Computer Science Education, Saint Louis MO, February 2005.
- [2] SoftwareSecure. Secureexam Browser. Online. Available at: <http://www.softwaresecure.com/browser.htm>.

- [3] Respondus. Respondus Lockdown Browser. Online. Available at: <http://www.respondus.com/products/lockdown.shtml>.
- [4] WebAssign. Secureexam Demo. Online. Available at: <http://www.webassign.net/securebrowser>.
- [5] Rivest, R., Shamir, A. and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, Vol. 21 (2), pp.120-126. 1978.
- [6] Daemen, J. and V. Rijmen. The Design of Rijndael: AES - The Advanced Encryption Standard, Springer Verlag, 2002.
- [7] Wang X. and H. Yu. "How to break MD5 and other hash functions", Eurocrypt 2005.
- [8] Wikipedia. BARTPE. Online. Available at: <http://en.wikipedia.org/wiki/BartPE>.
- [9] CodeProject. Detect If Your Program is Running Inside a Virtual Machine. Online. Available at: <http://www.codeproject.com/system/VmDetect.asp>.
- [10] International Telecommunication Union. Study Group 17 X.667. "Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components". Online. Available at: <http://www.itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf>.
- [11] VMWare ACE description. Online. Available at: <http://www.vmware.com/products/ace>.
- [12] SynchronEyes description. Online. Available at: <http://www2.smarttech.com/st-en-US/Products/SynchronEyes+Classroom+Management+Software/>.