Analysis of security in digital examinations

Kim Kling
Adrian Bjugård

TDA602 Language-based Security,
Department of Computer Science and Engineering,
CHALMERS UNIVERSITY OF TECHNOLOGY

May 2016

Contents

1. Background

Digital examinations are a way for institutions to make the work of grading easier, while also giving students a comfortable environment to work in, eliminating problems with student handwriting, and enforcing time restrictions.

Because an academic examination is a test of knowledge that is often awarded with course credits, it is important that those taking the exam do not cheat. In regular written exams this is generally done by having exam guards watching the students for signs of odd behaviour such as, but not limited to, reaching into pockets, visiting the bathroom multiple times, using unauthorised electronic devices, or communicating with other students [1].

By introducing computers into the world of examinations, many new potential ways for students to cheat are made possible. Some of which are not always as obvious to an exam guard as with a written exam. Often it is even necessary for exam computers to be networked in order to access the exam software, opening up even more potential security vulnerabilities, and in certain cases students are even allowed to run parts of the examination suite on their own hardware. One such example is Safe Exam Browser (SEB) which advertises itself with the claim that "The software changes any computer into a secure workstation" [2].

1.1. System architecture

The SEB application is an open source application [3] that implements the client in a server and client structure. The client acts almost exactly like a regular web browser, connecting to the examination server via HTTP or HTTPS depending on the server configuration. The notable difference from a regular browser is that SEB locks down the computer it is running on by disabling certain features:

- Shortcuts/Keys as Win, CTRL+ALT+DEL (Windows XP), ALT+F4, F1, CTRL+P etc.
- Right-click
- Switch to other applications (unless permitted)
- Process monitoring and killing of not allowed processes starting up while SEB is running
- Menu options on Windows Security Screen on Windows 7 removed while SEB is running
- VMware shade when using VMware Horizon View Client for VDI
- Display and system idle sleep
- Print Screen/OS X screen capture

Notably SEB also claims to disallow running inside of a Virtual Machine (VM) [4].

2. Goal

The primary goal of our project is to break out of the controlled environment of a client in a digital examination setup using SEB, and thus make it possible for us to run any application simultaneously. While breaking out, we aim to successfully access the Internet in a web browser while still taking the exam. Since the client is installed on a student computer, we can use any tool at our disposal in our attempts to do so.

We will look at SEB and try to break the enforced restrictions of the application. The restrictions include, but are not limited to, not running inside a VM, blocking access to other applications, blocking multitasking, disabling certain hot-keys and more. See the more extensive list of blocked features in Section 1.1.

2.1. Scope and limitations

We believe there are several possible ways to circumvent the security restrictions of SEB. In order to limit the scope we are focusing on two specific ways; running SEB inside a VM, and modifying the source code to build a version of SEB with the security restrictions removed.

2.2. Testing Environment

In the experiments where a VM is used, we used both OS X 10.11 with VirtualBox version 5.0.10 r104061 and Arch Linux with QEMU version 2.5.1-1. For the experiments where we used SEB natively, OS X 10.11 was utilised. For all experiments in VMs, we used version 2.1.1 of SEB. For all experiments that required us to compile SEB from source, we used the most current version in the official SEB GitHub repository, which at the time (2016-05-10) was at commit 278c760e145173a93cbc4a732d756c53db5cd996.

3. Description of Work

This chapter will explain the method used to reach a result in our investigation.

3.1. Running inside a Virtual Machine

By running SEB in a VM, we have the ability to contain the SEB restrictions to our VM software and the containing guest Operating System (OS). This enables us to run other applications in the host OS without the knowledge of SEB, and multitask between those applications.

To prevent this, SEB has checks to detect if it is being started from inside a VM and refuses to run if that is the case. By looking at the source code for SEB we found the function that performs this check. See Listing 1. [5]

Listing 1: The isInsideVM()-function

```
264 private static bool IsInsideVM()
265 {
266     using (var searcher = new ManagementObjectSearcher("Select * from
            Win32_ComputerSystem"))
267     {
268         using (var items = searcher.Get())
269         {
270             foreach (var item in items)
271             {
272                 Logger.AddInformation("Win32_ComputerSystem Manufacturer: "
                        + item["Manufacturer"].ToString() + ", Model: " +
                        item["Model"].ToString(), null, null);
273
274                 string manufacturer =
                        item["Manufacturer"].ToString().ToLower();
275                 string model = item["Model"].ToString().ToLower();
276                 if ((manufacturer == "microsoft corporation" &&
                        !model.Contains("surface")) ||
                        manufacturer.Contains("vmware")
277                         || manufacturer.Contains("parallels software")
278                         || manufacturer.Contains("xen")
279                         || model.Contains("xen")
280                         || model.Contains("virtualbox"))
281                 {
282                     return true;
283                 }
284             }
285         }
286     }
287     return false;
288 }
```

The check is made on lines 276 to 280 and blacklists specific vendors and products. In particular, all hardware platforms from the manufacturer Microsoft Corporation except Surface, everything from VMware, everything from Parallels Software, everything

from Xen and the specific model/products Xen and Virtualbox. We can not find the VM software manufacturer QEMU mentioned in this list, which makes their virtualisation software a good candidate for our goal of running SEB in a VM. Since a blacklisting technique is used it could be possible to modify the manufacturer and model information of the VM to circumvent this particular check.

3.2. Modify and recompile Safe Exam Browser to remove security checks

Because SEB is open source, theoretically, any potential attempts by the server to identify a modified version of SEB can be intercepted by us and replied to as if the application was unmodified. Therefore one potential way of escaping the restrictions imposed by SEB may be to modify the source code, removing the part of the code which blocks the users ability to switch applications. Except for the removed usability restrictions, a modified version of SEB needs to look and function exactly like an unmodified version, so as to fool both an exam guard, and the exam server being used.

After analysing the source code for the OS X version of SEB we found that most of the security restrictions are activated in the $\mathrm{startKioskModeThirdPartyAppsAllowed()}$ function, found on line 1266 in SEBController.m, shown in Listing 2.

Listing 2: The startKioskModeThirdPartyAppsAllowed()-function [6]

```
1266 - (void) startKioskModeThirdPartyAppsAllowed:[...] {
1267     // Switch to kiosk mode by setting the proper presentation options
1268     // Load preferences from the system's user defaults database
1269     NSUserDefaults *preferences = [NSUserDefaults standardUserDefaults];
1270     BOOL showMenuBar = overrideShowMenuBar || [preferences
             secureBoolForKey:@"org_safeexambrowser_SEB_showMenuBar"];
1271 //    BOOL enableToolbar = [preferences secureBoolForKey:[...]
1272 //    BOOL hideToolbar = [preferences secureBoolForKey:[...]
1273     NSApplicationPresentationOptions presentationOptions;
1274
1275     if (allowSwitchToThirdPartyApps) {
1276         [preferences setSecureBool:NO
                 forKey:@"org_safeexambrowser_elevateWindowLevels"];
1277     } else {
1278         [preferences setSecureBool:YES
                 forKey:@"org_safeexambrowser_elevateWindowLevels"];
1279     }
1280
1281     if (!allowSwitchToThirdPartyApps) {
1282         // if switching to third party apps not allowed
1283         presentationOptions =
1284         NSApplicationPresentationDisableAppleMenu +
1285         NSApplicationPresentationHideDock +
1286         (showMenuBar ? 0 : NSApplicationPresentationHideMenuBar) +
1287         NSApplicationPresentationDisableProcessSwitching +
1288         NSApplicationPresentationDisableForceQuit +
1289         NSApplicationPresentationDisableSessionTermination;
1290     } else {
1291         presentationOptions =
1292         (showMenuBar ? 0 : NSApplicationPresentationHideMenuBar) +
1293         NSApplicationPresentationHideDock +
1294         NSApplicationPresentationDisableAppleMenu +
1295         NSApplicationPresentationDisableForceQuit +
1296         NSApplicationPresentationDisableSessionTermination;
1297     }
1298
1299     @try {
1300         [[MyGlobals sharedMyGlobals]
                 setStartKioskChangedPresentationOptions:YES];
1301
1302         DDLogDebug(@"setPresentationOptions: %lo", presentationOptions);
1303
1304         [NSApp setPresentationOptions:presentationOptions];
1305         [[MyGlobals sharedMyGlobals]
                 setPresentationOptions:presentationOptions];
1306     }
1307     @catch(NSException *exception) {
1308         DDLogError(@"Error. Make sure presentation options are valid.");
1309     }
1310 }
```

3.3. Accessing an exam server without Safe Exam Browser

Since the architecture (as described in Section 1.1) uses HTTP or HTTPS to serve the exam, a possible way to cheat is to access the exam server via a regular web browser. This enables us to circumvent the enforced restrictions of SEB. Problems such as finding the URL and circumventing any required header data must be handled.

The exam server verifies that the student is using SEB by checking an HTTP-header that SEB includes for all requests to the server.

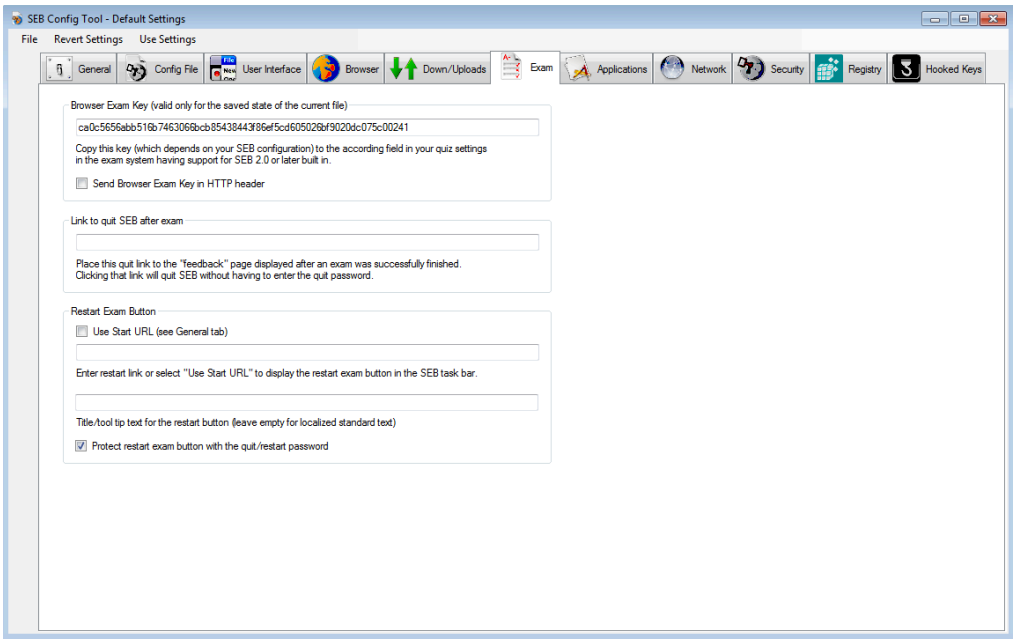

Figure 1: SEB Config Tool screenshot including HTTP-header data for one exam

By analysing the source code of SEB we found that the data for this extra header is available in the '.seb' configuration file downloaded from the exam server when the student wants to connect to the exam server. This file is compressed and encrypted using the passcode students are provided for accessing the exam. SEB could be modified to decrypt this file and extract the header data on launch instead of starting the browser. We could then manually insert it into requests coming from another browser like for example Google Chrome, and by running that browser in full screen presentation mode, we can achieve a visually similar experience.

4. Result

This chapter will detail and discuss the results of our work.

4.1. Running inside a Virtual Machine

While working on this project we succeeded in running an unmodified version of SEB inside a VM using multiple different strategies. These are detailed below.

4.1.1. QEMU

Analysing the source code we found that there is no mention of the QEMU virtualisation software in the blacklist. When tested, we found that running SEB in a QEMU VM worked without any problems, SEB started as expected and accepts an exam. There were no modifications to QEMU required in order for this to work, we simply set up a guest OS and installed SEB .

4.1.2. VirtualBox

At first, the created VM could not successfully start SEB. See Figure 2. We then made the modifications in Listing 3 to configure VirtualBox in order to report a different manufacturer and product to the OS.

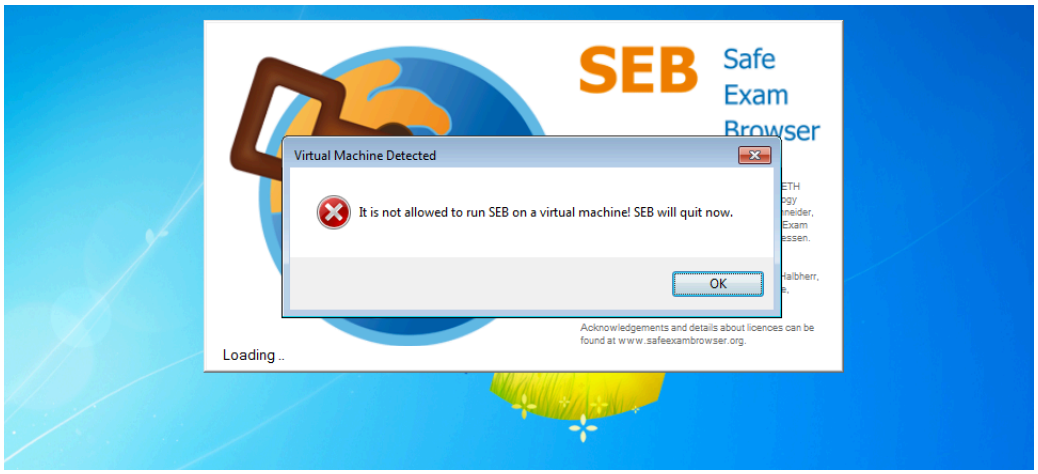

Figure 2: SEB fails to start in a standard VirtualBox VM

In order to fake our manufacturer and product, we used the following terminal commands:

These commands changes what the system reports as the hardware parameters to, in this particular case, ”Other” on both the vendor and product. Since this is not blacklisted by SEB, it successfully starts. See Figure 3.

Listing 3: Modifications to default configuration in VirtualBox

```
1  VBoxManage setextradata "Safe Exam Browser"
       "VBoxInternal/Devices/pcbios/0/Config/DmiSystemVendor" "Other"
2  VBoxManage setextradata "Safe Exam Browser"
       "VBoxInternal/Devices/pcbios/0/Config/DmiSystemProduct" "Other"
```

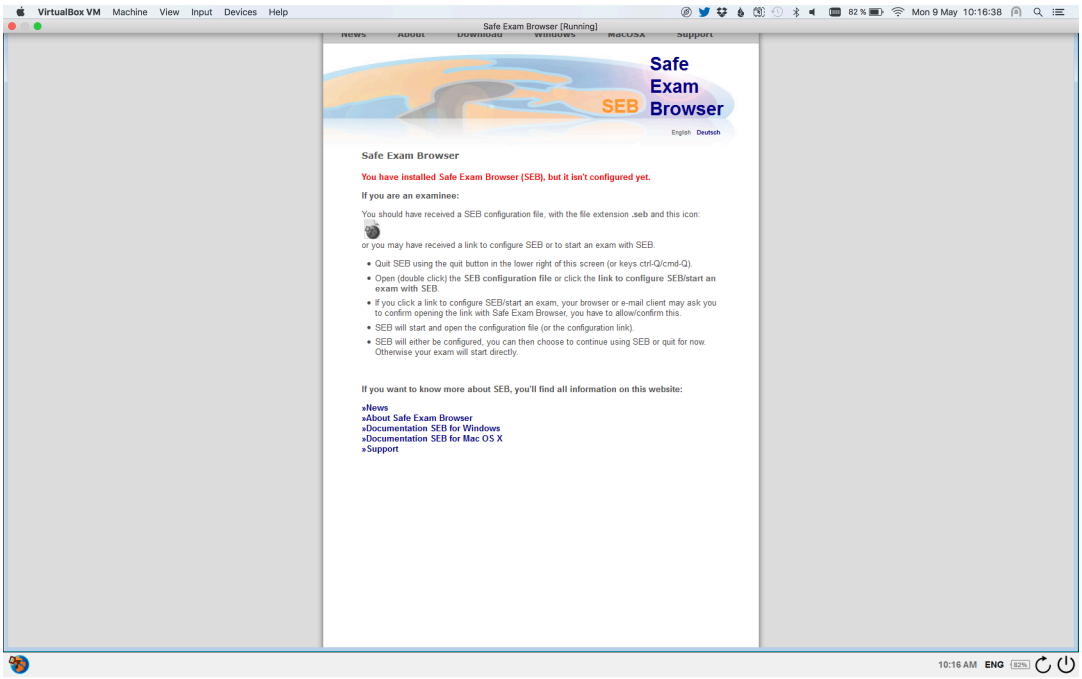

Figure 3: A screenshot of SEB started in a tweaked VirtualBox VM

4.2. Modify and recompile Safe Exam Browser to remove security checks

While investigating the SEB for OS X source code, we found that the majority of the restrictions are set up inside the awakeFromNib and startKioskMode functions. See an extract in Listing 4. The first, awakeFromNib, spawns a number of observers which notifies SEB of OS events relevant to the intended system restrictions. startKioskMode is where SEB actually implements the restrictions, setting running modes like. These restrictions can be evaded fully by simply adding presentationOptions = 0;, as well as making some other minor changes in SEBController.m (full changes can be seen in Listing 4).

Listing 4: The modifications to the startKioskModeThirdPartyAppsAllowed()-function, additions highlighted in green, changes in yellow

```
1281    NSApplicationPresentationOptions evaluatedPresentationOptions;
1282    if (!allowSwitchToThirdPartyApps) {
1283        // if switching to third party apps not allowed
1284        evaluatedPresentationOptions =
1285        NSApplicationPresentationDisableAppleMenu +
1286        NSApplicationPresentationHideDock +
1287        (showMenuBar ? 0 : NSApplicationPresentationHideMenuBar) +
1288        NSApplicationPresentationDisableProcessSwitching +
1289        NSApplicationPresentationDisableForceQuit +
1290        NSApplicationPresentationDisableSessionTermination;
1291    } else {
1292        evaluatedPresentationOptions =
1293        (showMenuBar ? 0 : NSApplicationPresentationHideMenuBar) +
1294        NSApplicationPresentationHideDock +
1295        NSApplicationPresentationDisableAppleMenu +
1296        NSApplicationPresentationDisableForceQuit +
1297        NSApplicationPresentationDisableSessionTermination;
1298    }
1299    presentationOptions = 0;
1300    @try {
1301        [[MyGlobals sharedMyGlobals]
                setStartKioskChangedPresentationOptions:YES];
1302
1303        DDLogDebug(@"setPresentationOptions: %lo",
            evaluatedPresentationOptions);
1304
1305        [NSApp setPresentationOptions:presentationOptions];
1306        [[MyGlobals sharedMyGlobals]
                setPresentationOptions:presentationOptions];
1307    }
1308    @catch(NSException *exception) {
1309        DDLogError(@"Error. Make sure presentation options are valid.");
1310    }
```

With these changes made we can compile a version of SEB that does not block application switching, allowing a student to access cheating information running in the background of SEB. Our modified SEB, still visibly identical to an unmodified version, is now just a full-screen browser.

4.3. Accessing an exam server without Safe Exam Browser

We realised that pursuing this approach was outside of the scope of our analysis, so we did not develop a proof of concept using this approach.

5. Discussion

The worries and concerns that running an exam client on the students own computers have been confirmed by our results. In this section we discuss our results and elaborate on how we think the issues identified impact the future of digital examinations.

5.1. Fundamental Problems

Making any system into a secure exam platform is a grandiose claim to make, and proves too big for SEB to deliver. In general we believe it is a very difficult task, if possible at all, to implement security critical applications such as an exam application in an environment that is ultimately uncontrolled by the application. There is always some degree of user trust in such systems, and requiring the trust of users during an examination should be avoided if possible.

5.2. Running inside a Virtual Machine

The protection SEB has against being executed in VM is decidedly insufficient according to our results. The checks, at the time of writing this report, for a VM environment is only to check for a short list of manufacturers and products as a blacklist. We have shown that it is possible to use either a piece of VM software that SEB does not blacklist, like QEMU, or to simply change certain values in the configuration of another VM software which SEB actually does attempt to blacklist, such as VirtualBox.

From a students point of view, this is a simple and easy way to get around the restrictions. Although some modification is often required, we believe that with instructions any student could utilise this way of cheating.

Unfortunately there is not a clear way to solve this issue, but in general it would be wise to look at a more sophisticated method of detecting a VM environment. Either by extending the blacklist to other hardware parameters such as network controllers (VirtIO), processors (QEMU Virtual CPU), disk devices (VMware Virtual disk), and graphics controllers (VMware SVGA 3D) to name a few or by detecting certain characteristics that a VM introduces. However, in SEBs defense, in reality this is a cat and mouse game, as a user need only look at how the software detects the VM components and adapt their approach accordingly.

5.3. Modify and recompile Safe Exam Browser to remove security checks

The results show that there are no restrictions against modifying and compiling your own version of SEB. We suggest that the detectability of this type of cheating in an exam is even lower than running SEB in a VM, since no middleware can be found upon inspection. It is also very easy as a student to utilise since it only requires one person to do the modification, packaging and then distributing of the binary (potentially under the name *Unsafe Exam Browser*) to other students. The knowledge requirements to use are therefore the same as to use the regular version of SEB.

One idea to improve the system would be to have the server check the integrity of the client software by asking it for some checksum. But that takes us back to the original problem. A server side way of verifying that a student is indeed running the correct version of SEB can be fooled by the response from the client since it is spoofed as well, if the source codes was modified to do that. Solving this particular problem when we have a piece of client side software, that executes such a substantial part of the system logic, is difficult as the user can choose to compile the software themselves.

In order to really solve this issue, a change in the overall architecture is needed. We need to move more of the logic to the server. One way to do this is for the server to not only serve the exam, but serve the binary that restricts the student computer. The client-side implementation is changed into a simple loader which downloads a binary version of SEB for the platform the student laptop is running and then execute this binary. This binary could be bundled with the exam configuration. This would make it possible for an institution conducting digital examinations to include security features in the binary that a student can not prepare for in advance.

5.4. The future of digital examination

As mentioned many times throughout this report, the students computer should not be trusted to behave as desired by the examiner. Modifications to the hardware and OS are always possible when the user has administrative rights. This can be used to trick applications so they do not behave as expected.

We believe that due to this problem, future digital examinations must be performed in controlled environments where the student has limited rights to modify the OS, and are only allowed to run the exam client. The exam client could even be executed at startup, to prevent the user from downloading and starting a modified version.

This dramatically decreases the risk that the chain of trust is ever broken, and that the student can subsequently cheat when using the exam computer.

5.5. Future work

If we had more time to work on this project, the next item to focus on would have been to build a sandbox environment to load SEB in, simulating positive responses to relevant system calls made by SEB. This way we can run an official version of SEB without having to make modifications to the source code, and still have the same behaviour as when security checks are removed.

6. Conclusion

From the beginning, trusting a students computer to play by the rules implies a trust-based system. As expected, breaking out of SEB and accessing other applications while taking an exam was possible in multiple ways. The anti-VM feature in SEB is unsuccessful in stopping most VM software with small modifications to the default configuration, and fails entirely at stopping other VM software, even unmodified.

The whole idea of running exam kiosk software like SEB on students own computers, as a means of saving money, only serves to provide the exam guards with a false sense of security that students can not cheat. In this report we have, in numerous ways, proven the claim that SEB turns any computer into a secure platform false.

Even if the specific loopholes identified in this report are fixed, we are doubtful that the strategy employed by SEB can, on its own, be effective at actually locking down any given computer and transform it into a secure exam kiosk.

Ultimately, since the signs of cheating are generally more difficult to detect when computers are involved, the idea of allowing students to bring their own hardware for an exam, thinking that SEB will turn those machines into secure workstations following a set of rules placed by the institution giving the exam, is fundamentally flawed.

The silver lining is that without any further modifications SEB could be a good fit to use as an exam kiosk on controlled computers. As a means of simply locking out other functions of the operating system while students are taking an exam, the software performs rather well. It just can not make such promises when students own computers are used to run SEB. The hardware and OS running SEB must be carefully inspected to ensure that no modifications are made to either the SEB installation or any other applications prior to the exam.

References

[1] Patricia MacKown Kevin Yee. "Detecting and Preventing Cheating During Exams". In: (2009). Accessed 2016-05-15. URL: https://www.syr.edu/gradschool/pdf/resourcebooksvideos/AIBook/AIYee.pdf.

[2] Safe Exam Browser. *Safe Exam Browser - About SEB*. URL: http://safeexambrowser.org/about_overview_en.html (visited on 9th May 2016).

[3] Safe Exam Browser. *Safe Exam Browser*. URL: https://github.com/SafeExamBrowser/ (visited on 10th May 2016).

[4] Safe Exam Browser. *Safe Exam Browser*. URL: http://safeexambrowser.org/about_overview_en.html#Features (visited on 10th May 2016).

[5] Safe Exam Browser. *seb-win/SebWindowsClientMain.cs at 6591a3dae8ef2311d2... SafeExamBrowser/seb-win*. URL: https://github.com/SafeExamBrowser/seb-win/blob/6591a3dae8ef2311d26f09fbe72ea5365b2ba490/SebWindowsClient/SebWindowsClient/SebWindowsClientMain.cs#L264 (visited on 6th May 2016).

[6] Safe Exam Browser. *seb-mac/SEBController.m at 278c760e145173a93cbc4a732d7... SafeExamBrowser/seb-mac*. URL: https://github.com/SafeExamBrowser/seb-mac/blob/278c760e145173a93cbc4a732d756c53db5cd996/Classes/SEBController.m#L1266 (visited on 15th May 2016).

Appendices

A. Individual Contribution

Both students contributed equally in all parts of the project. On all practical tests both students have been active and trying different things. In the writing of this report, both students have written about the same amount in all sections.