**LICENSING AND ECONOMICS OF SOFTWARE REUSE****Navninderjit Singh\***

Punjabi University Patiala.

Article Received on 29/10/2019

Article Revised on 19/11/2019

Article Accepted on 09/12/2019

**\*Corresponding Author****Navninderjit Singh**

Punjabi University Patiala.

**ABSTRACT**

Cracking some Adobe products is as simple as replacing a .dll file in the respective app's folder. These software range from hundreds of dollars in price to thousands of dollars in price. Thus, making sure

users have a legitimate copy and are not using pirated software is of utmost importance to many distributed software companies. There is a lot of information concerning this topic on the Internet, since it is related to piracy, most of the technical information is kept behind closed doors. This is where much of our research will have to take place. We plan to solve this problem by inspecting the dll (dynamic link library) files that regulate the product's license. After analyzing the files, we will try and find a way to see if a product's legitimacy can be verified by testing scenarios such as if the dll has been modified in any way.

**KEYWORDS:** DLL, Reverse Engineering, Software Cracking, Adobe Photoshop, Licensing.

**[I] INTRODUCTION**

The pricing of software can fall into the following broad categories: shareware, liteware, freeware, public domain software, and open source (Rouse, Software, n.d.) When using software from major companies, such as Microsoft, Adobe, Autodesk, or Corel, the software ranges from hundreds of dollars to thousands of dollars in price. We will use Adobe's software products and digital forensics to analyze the file used for licensing: the amtlb.dll file, and provide a range of proposed solutions to allow software companies to ensure users are using a legitimate copy.

**[II] LITERATURE REVIEW**

Business Software Alliance conducted a survey for users who confessed to pirating software around the world. The results stated that 57 percent admitted to pirating.<sup>[5]</sup> While researching Adobe, we discovered that with a simple modification to the .dll file, one could take the free trial software, extend the expiration date, and have unlimited use of the current premium software. An article written by a lawyer stated that a company will most likely not take action against an end user of pirated software. Typically, the companies go after the supplier of the pirated software or if the software is being used at a large scale within a specific institution. This raises the question of how much money Adobe might lose from pirated software. Further research found that an article written on gadgetsnow.com stated Microsoft and Adobe lost around \$14 billion in 2011. Given that was 8 years ago, the number should be exponentially larger given today's current state of software technology explosion. To better understand how .dlls work, we found a mutiple Microsoft developer documents that provided a comprehensive understanding of what exactly the files do. Once we had a better understanding of the dll file, we came across a website on DLL file protection. The company was Palo Alto Networks and provided an understanding on how to protect DLL files and how to block malicious DLL files.

**[III] Software License**

Most software used today is licensed in some form or another. A software license is an agreement between the user and the owner that provides rules like where and how often to install/update, what permissions are available to copy, modify, or redistribute, and whether the user can view the source code for the software. Most software also includes a license key. A license key is a long string of letters and numbers that are configured/saved during installation. This method provides a means for software companies to confirm the user is compliant with the software license.

**[IV] Software Price**

The price of the software depends on three main characteristics: the company, function, and availability. Because some software can be quite expensive, the companies offer evaluation or trial copies. The evaluation copies could be limited to time constraint, features, or could run just like the full version, albeit a watermark is embedded into the product. Evaluation or trial copies help users decide if a software is really worth the cost.

**[V] Software Authentication**

Authentication is comparing credentials with the required information stored in a file or database. This information is usually stored on the local operating system or with an authentication server. (Rouse, Authentication, n.d.) It is up to the software company to provide the means to verify that the software users are authenticated correctly. Some products include a file that hosts the key that the software checks every time the program is opened to do so.

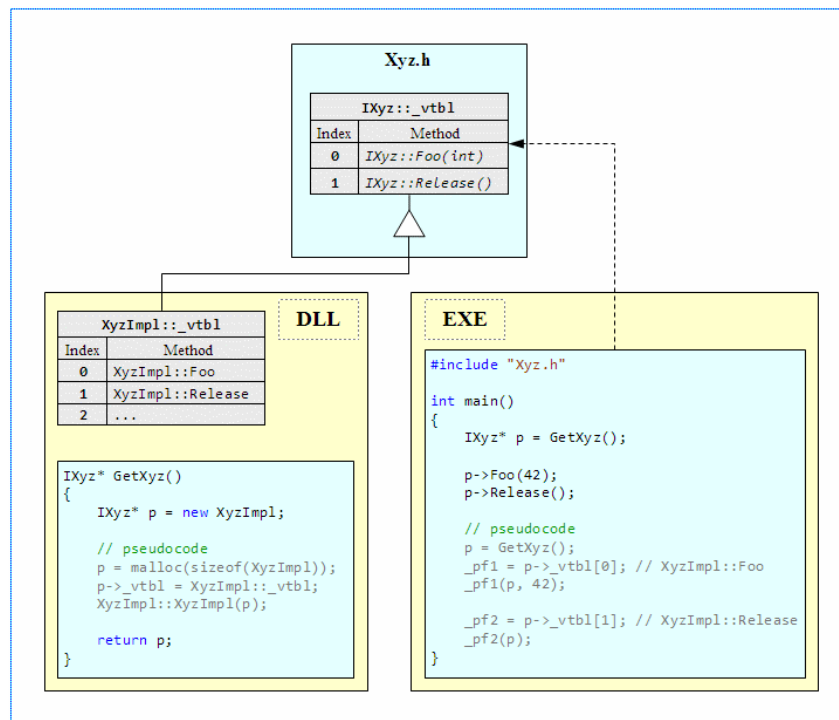
**[VI] Adobe**

Adobe is an American multinational computer software company. The software company focuses on the release of creativity and multimedia products. Well known Adobe software used today are Adobe Acrobat Reader and creative tools such as Adobe Photoshop. Professionals typically use a suite called Adobe Creative Cloud giving them all the tools needed to produce multimedia content. For our research on the Adobe products we will focus on Photoshop, however most Adobe products work the same way in regulating their licensing. (Adobe, n.d.).

**[VII] Adobe Photoshop**

Adobe Photoshop stores the activation licensing code in the amtlib.dll file. When you purchase the software, during the installation process, the activation license key you entered is regulated through the amtlib.dll file. Every time the program is opened, it checks this file for the license and communicates with Adobe servers when connected to the internet. The cost of Photoshop is expensive. In the past, the software could cost you around \$600 for the current version. While there was no timeline for when this product expired, it would cost you around the same every time you upgraded. With the creation of Adobe Creative Cloud, the software is now purchased on a monthly plan that includes upgrades to the newest version (Adobe, n.d.)

## [VIII] Dynamic Link Library (.dll)



**Figure 1: DLL use-case (Stack Overflow answer: relation diagram).**

The dynamic link library (dll) is a file that contains compiled code used in all programs. With DLLs, programs can be separated into different modules. This makes updates and revisions easier as the updates or revisions can be applied to the appropriate file without the need to rebuild or reinstall an entire program. DLLs cannot run on their own; an executable is needed to run a program, then DLLs can be called as needed. DLLs are helpful as multiple programs can share the abilities of the host program in a single file and can be called on by the program only as needed, then released allowing memory to be free for other uses. (What Is a DLL file, n.d.)

## [IX] Digital Certificates

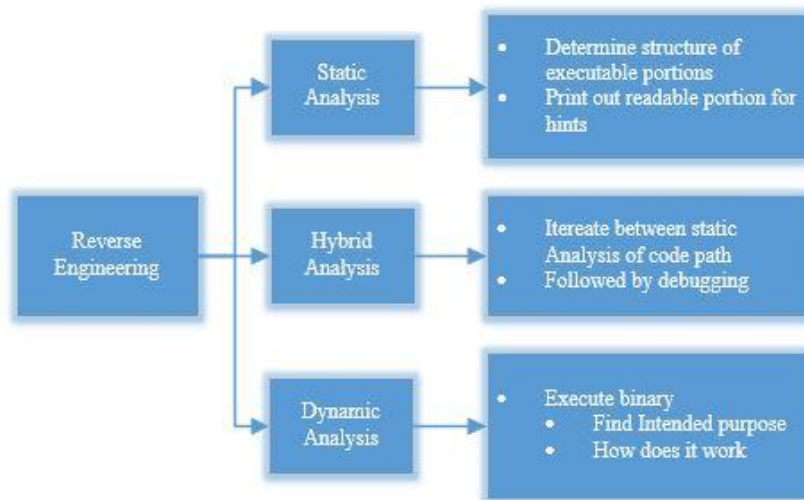
Digital certificates are basically an electronic record that a person or organization needs to exchange data securely over the Internet. Digital certificates use a public key infrastructure. (Digital Certificate, n.d.) As the Internet became popular many people and companies were creating their own webpages, thus opening the world up to criminal minds as they found ways to exploit vulnerabilities and gain access to sensitive information. A security solution was needed, thus the introduction of digital certificates. Digital certificates provide security features such as authentication /identification, confidentiality, integrity, and access control. Digital certificates provide businesses a secure way to communicate with others over the

Internet. For the purpose of this report, digital certificates were used to ensure a file has not been modified.

### [X] Software Cracking

Software cracking, as it suggest is the process of changing the software that eradicates the distinctive attributes those are not required, such as copy protection attribute. (Quora, 2017)

### [XI] Reverse Engineering Binary Assembly



**Figure 2: Reverse engineering hierarchy.**

There are three kinds of reverse engineering analysis. Static analysis, which involves analyzing the structure for hints on how the software is assembled, dynamic analysis which involves determining its purpose by launching it, and hybrid analysis which combines the two. (Getting Started with Reverse Engineering, 2015). Executable software is written in assembly language. DLLs are in the same format as executables. For the `amtlb.dll` file, it is important to reverse engineer the file, determine the location and format of the components of the license agreement, and determine how a hacker can revise the information to gain full rights to the software.

### [XII] dnSpy

There are many tools available for reverse engineering. dnSpy is an open source assembly editor that can be used for reverse engineering. This product includes a decompiler, debugger, and an assembly editor. With this tool, assemblies can be read and written, including obfuscated assemblies using a utility called `dnlib`.

**[XIII] Interactive Disassembler (IDA)**

Another program known as Interactive Disassembler (IDA). IDA takes a program, analyzes it, and provides the instructions in assembly language. IDA Pro make the code readable in a very user-friendly manner with multiple views available. IDA also serves as a debugger. The program can also be used to analyze viruses, worms, or other malware. (IDA: About, n.d.)

**[XIV] OllyDbg**

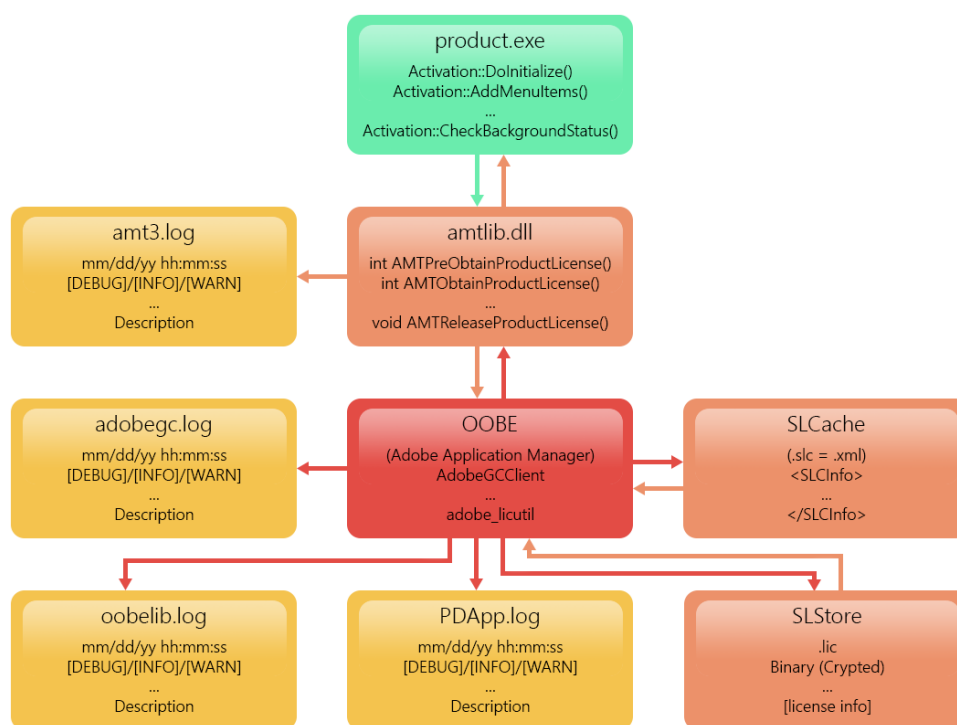
OllyDbg is a 32-bit assembly level debugger, specifically for Microsoft Windows. OllyDbg is a shareware that is available for free. OllyDbg has several features include direct loads and debugging of DLLs. (OllyDbg, n.d.)

**[XV] Methodology**

Software cracks are a very serious problem. A very popular trend in acquiring customers nowadays is the free trial strategy, in which the base trial holds all the features of the premium version, though it only lasts for a limited amount of time (usually one week to a month) as to convince the user to eventually purchase the paid product after exploring its features. Since you do not need to be very technologically savvy in order to abuse a pre-existing software crack, a decent number of abusers could run an app dependent on converting trial users to paid users for revenue out of business. Imagine if common criminals could just use a teleporter to jump in and out of brick-and-mortar stores to steal inventory, it would be total anarchy for the retail businesses. In essence, the exact same thing is happening to apps with fraudulent licenses, making protecting their digital goods a top priority.

To the general public, software cracks often release sporadically via various file-sharing services, which allow their abusers to acquire any number of paid softwares for free. Essentially all you have to do in order to exploit this massive financial threat to a company is search up a tutorial on the web, follow the step by step instructions, and viola: you can instantly steal expensive products from any of the many vulnerable applications within a matter of minutes, with no technical knowledge required. The abusers of software cracks often refer to themselves as hackers, when in reality all they did was employ the work of the real hacker. As computer scientists, we want to be more than just a user in order to understand the ins and outs of how these hacks are carried out, so that we may protect our own apps from similar security breaches in the future.

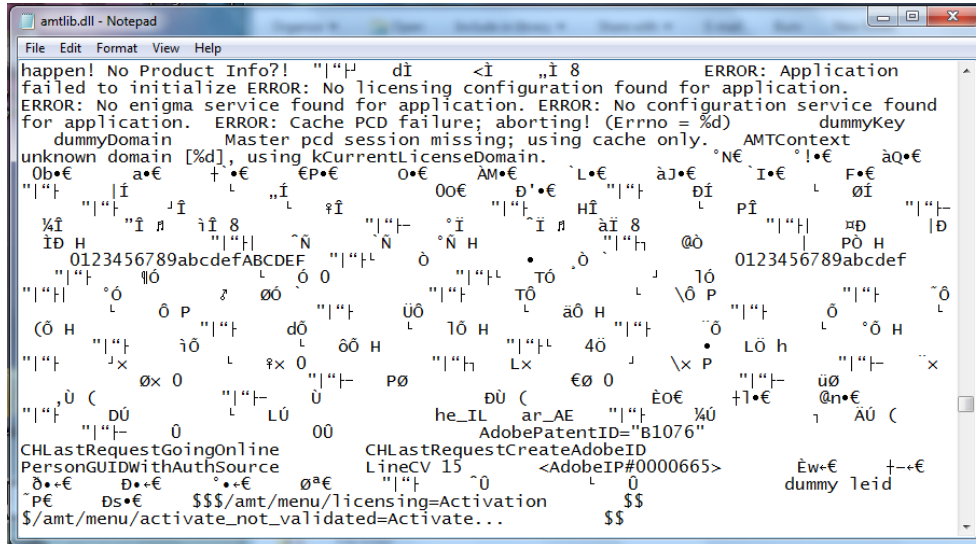
For example, to crack Photoshop, all a user would need to do is download the trial version from the official Adobe website. Once the trial is downloaded, prior to actually opening up the application, all that a user needs to do is disconnect from the internet (so as to stop any communication with Adobe servers that is happening) and replace the old amtlib.dll file with a cracked amtlib.dll, which can be found in under a minute from a quick Google search. The key step lies within the amtlib.dll as this file regulates licensing across all Adobe products. In the following section we will discuss our findings on the matter.



**Figure 3: Supposed structure of the validation check (AMT Emulator).**

Above is a figure laying out the supposed structure of the underlying application interface put into place by Adobe. It was showcased in a blog post concerning AMT Emulator, a universal solution to cracking Adobe products. AMT Emulator is one of those one-click solutions for nontechnical users as explained before: you simply download it, click run, and now you have the paid software for free. In fact, all that AMT Emulator does is replace the original amtlib.dll with a patched amtlib.dll, just as explained before. More so, any type of validation checking distributed software can be bypassed in the same way: by simply disregarding the checks whether or not they are client-sided or server-sided, it does not matter. Now armed with a general understanding on the subject, the software cracking began.

Opening up the amtlib.dll file in any common text editor such as Notepad would result in some readable text due to scattered hex values being translated to ASCII for corresponding bytes of data, but more often than not, the file is just a bunch jumbled garbage as displayed below.



```

happen! No Product Info?!  " |" | d|  <|  „|  8  ERROR: Application
failed to initialize ERROR: No licensing configuration found for application.
ERROR: No enigma service found for application. ERROR: No configuration service found
for application. ERROR: Cache PCD failure; aborting! (Errno = %d)  dummyKey
dummyDomain  Master pcd session missing; using cache only.  AMTContext
unknown domain [%d], using kCurrentLicenseDomain.  ^N€  °!•€  àQ•€
0b•€  a•€  †•€  €P•€  O•€  AM•€  °L•€  àJ•€  †•€  F•€
"|“†  "|“†  |  |  „|  8  „|  8  „|  8  |  |  |  |  |  |  |  |  |  |  |
%|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
Ï  H  0123456789abcdefABCDEF  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
"|“†  °  °  P  z  ø  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
(  H  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
"|“†  j  x  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
"|“†  ø  x  0  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
"|“†  ù  (  DÚ  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
"|“†  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
CHLastRequestGoingOnline  CHLastRequestCreateAdobeID
PersonGUIDWithAuthSource  LineCV 15  <AdobeIP#0000665>  Èw•€  †•€
  ••€  D••€  °••€  ø•€  „|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
PE  Ds•€  $$$/amt/menu/licensing=Activation  $$$  dummy leid
$/amt/menu/activate_not_validated=Activate...  $$$

```

Figure 4: amtlib. dll opened up in a text editor.

From the values that were humanly readable, occurrences of important keywords such as license or valid could be discovered, though any attempt to understand the implementation of the license validation process would be unavailing as its true functionality is hidden beneath the hieroglyphic symbols present. Thus, in order to truly understand the inner workings of the amtlib.dll, a more powerful tool was needed.

Initially a decompiler was used in an attempt to view the original source code of the file, though after many attempts the file was either incompatible or revealed very little information. Below is the result of decompiling the file in dnSpy (usually doing so would result in a hierarchy of valuable functions giving an analyst a well-rounded idea of how the dll was constructed, though here all that was exposed was file metadata).



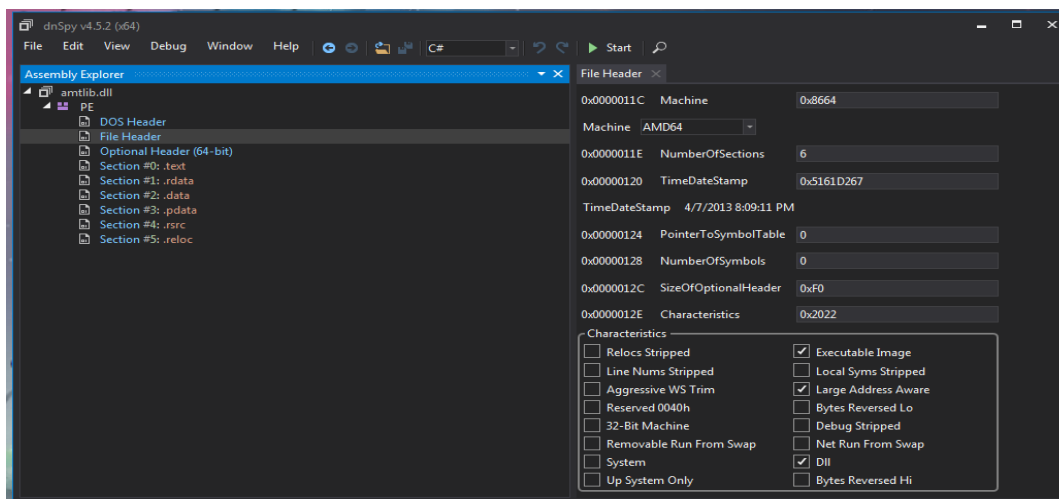


Figure 5: amtlib.dll opened in a decompiler.

As a result of this failure to reveal the inner workings of the amtlib.dll through source code, we must attempt to reveal information at an even lower level: assembly language. This process can be carried out through a disassembler. Below is a screenshot of the text view we saw earlier in Figure 4, from the IDA display.

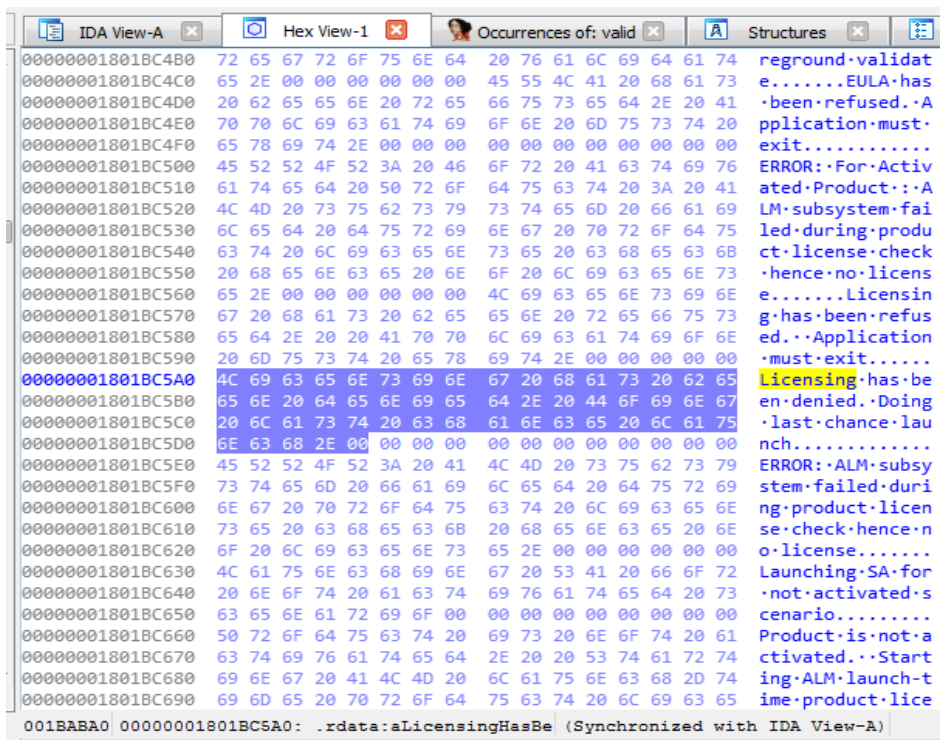
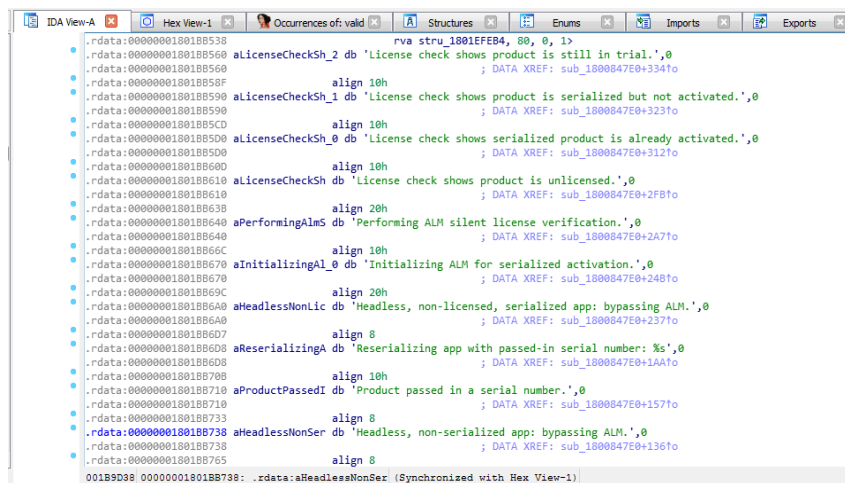


Figure 6: amtlib.dll opened up in a disassembler.

Now this is where things start to get interesting for a digital forensics analyst. Using IDA, we are able to jump to the exact spot in the assembly code that the text comments are referring

to; something that was not possible with a regular text editor. Below is an example of a section of these reference strings, which we can double-click to jump straight to their corresponding methods.



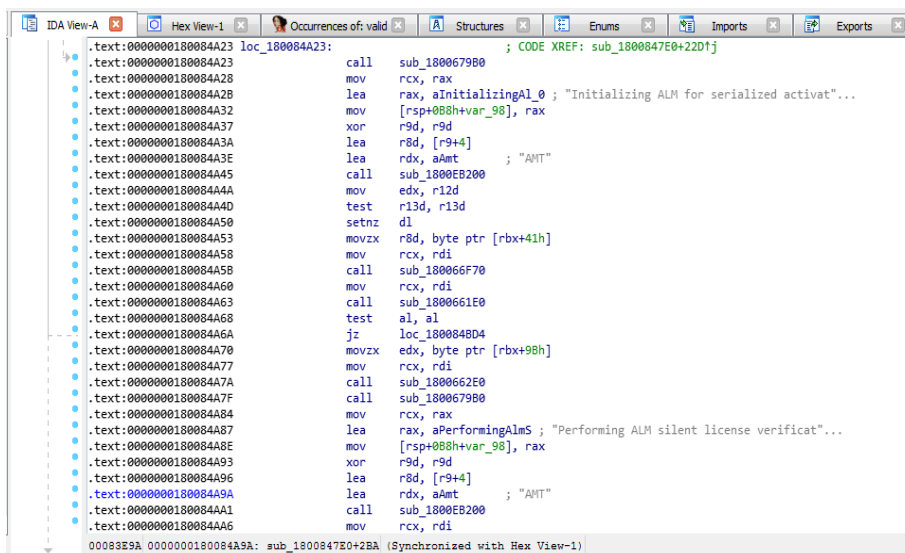
```

.rdata:00000001801BB538      rva stru_1801EFB4, 80, 0, 1>
.rdata:00000001801BB560      aLicenseCheckSh_2 db 'License check shows product is still in trial.',0
                                ; DATA XREF: sub_1800847E0+334f0
.rdata:00000001801BB58F      align 10h
.rdata:00000001801BB598      aLicenseCheckSh_1 db 'License check shows product is serialized but not activated.',0
                                ; DATA XREF: sub_1800847E0+323f0
.rdata:00000001801BB5CD      align 10h
.rdata:00000001801BB5D0      aLicenseCheckSh_0 db 'License check shows serialized product is already activated.',0
                                ; DATA XREF: sub_1800847E0+312f0
.rdata:00000001801BB600      align 10h
.rdata:00000001801BB610      aLicenseCheckSh db 'License check shows product is unlicensed.',0
                                ; DATA XREF: sub_1800847E0+2FBf0
.rdata:00000001801BB63B      align 20h
.rdata:00000001801BB640      aPerformingAlmS db 'Performing ALM silent license verification.',0
                                ; DATA XREF: sub_1800847E0+2A7f0
.rdata:00000001801BB66C      align 10h
.rdata:00000001801BB670      aInitializingAl_0 db 'Initializing ALM for serialized activation.',0
                                ; DATA XREF: sub_1800847E0+248f0
.rdata:00000001801BB670      align 20h
.rdata:00000001801BB6A0      aHeadlessNonLic db 'Headless, non-licensed, serialized app: bypassing ALM.',0
                                ; DATA XREF: sub_1800847E0+257f0
.rdata:00000001801BB6D7      align 8
.rdata:00000001801BB6D8      aReserializingA db 'Reserializing app with passed-in serial number: %s',0
                                ; DATA XREF: sub_1800847E0+1AAf0
.rdata:00000001801BB700      align 10h
.rdata:00000001801BB710      aProductPassedI db 'Product passed in a serial number.',0
                                ; DATA XREF: sub_1800847E0+157f0
.rdata:00000001801BB733      align 8
.rdata:00000001801BB738      aHeadlessNonSer db 'Headless, non-serialized app: bypassing ALM.',0
                                ; DATA XREF: sub_1800847E0+136f0
.rdata:00000001801BB755      align 8
001B5D38 00000001801BB738: .rdata:aHeadlessNonSer (Synchronized with Hex View-1)

```

Figure 7: Commented headers referencing licensing code.

Now we began perusing for keywords of interest, in particular: license. After determining the headers most likely to be related to validation checking, one only needs to double-click the subroutine reference to jump straight into the assembly implementation. Below is a subroutine related to the initialization of the Adobe License Manager for serialized activation.



```

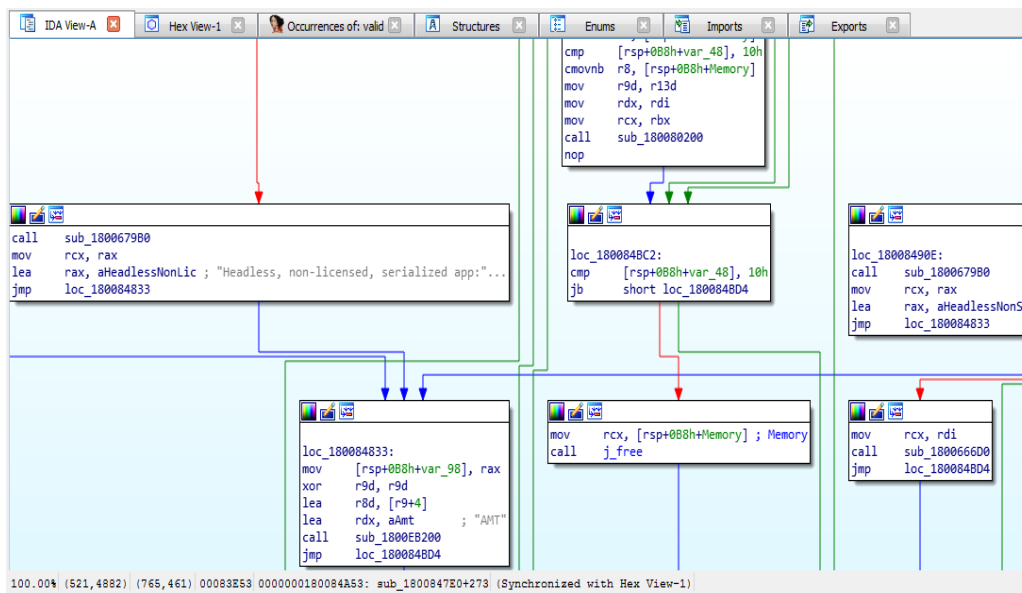
.rtext:0000000180084A23      loc_180084A23:      call     sub_180067980 ; CODE XREF: sub_1800847E0+22D2f1
.rtext:0000000180084A23      mov     rcx, rax
.rtext:0000000180084A28      lea    rax, aInitializingAl_0 ; "Initializing ALM for serialized activat"...
.rtext:0000000180084A32      mov     [rsp+0B8h+var_98], rax
.rtext:0000000180084A37      xor     r9d, r9d
.rtext:0000000180084A3A      lea    r8d, [r9+4]
.rtext:0000000180084A3E      lea    rdx, aAmt ; "AMT"
.rtext:0000000180084A45      call   sub_1800EB200
.rtext:0000000180084A4A      mov     edx, r12d
.rtext:0000000180084A4D      test   r13d, r13d
.rtext:0000000180084A50      setnz  dl
.rtext:0000000180084A53      movzx  r8d, byte ptr [rbx+41h]
.rtext:0000000180084A58      mov     rcx, rdi
.rtext:0000000180084A5B      call   sub_180066F70
.rtext:0000000180084A60      mov     rcx, rdi
.rtext:0000000180084A63      call   sub_1800661E0
.rtext:0000000180084A66      test   al, al
.rtext:0000000180084A6A      jz     loc_1800848D4
.rtext:0000000180084A70      movzx  edx, byte ptr [rbx+98h]
.rtext:0000000180084A77      mov     rcx, rdi
.rtext:0000000180084A7F      call   sub_1800662E0
.rtext:0000000180084A84      call   sub_180067980
.rtext:0000000180084A87      mov     rcx, rax
.rtext:0000000180084A8E      lea    rax, aPerformingAlmS ; "Performing ALM silent license verificat"...
.rtext:0000000180084A93      mov     [rsp+0B8h+var_98], rax
.rtext:0000000180084A96      xor     r9d, r9d
.rtext:0000000180084A99      lea    r8d, [r9+4]
.rtext:0000000180084AA1      lea    rdx, aAmt ; "AMT"
.rtext:0000000180084AA6      call   sub_1800EB200
.rtext:0000000180084AA6      mov     rcx, rdi
00083E9A 0000000180084A9A: sub_1800847E0+2BA (Synchronized with Hex View-1)

```

Figure 8: Actual assembly code referenced by the headers.

This however, is a very narrow viewpoint, and not enough reconnaissance has been conducted in order to fully understand the implementation of the validation check. To successfully reverse engineer the amtlib.dll file we must first take a step back to understand

how this specific function relates to other functions in order to see the big picture. Below is a display of the graph view of IDA, showcasing how different subroutines relate to one another.



**Figure 9: A graph view showcasing subroutine relations.**

After performing substantial review on the implementation of the `amtlb.dll` file the true software cracking began. In order to test the workarounds, the debugger OllyDbg was utilized in order to set breakpoints in the assembly code and step through instructions one by one, even able to view the values loaded in and out of specific registers. It was here we were able to determine which functions to avoid in order to bypass the validation checks. Below is a view of the OllyDbg interface.

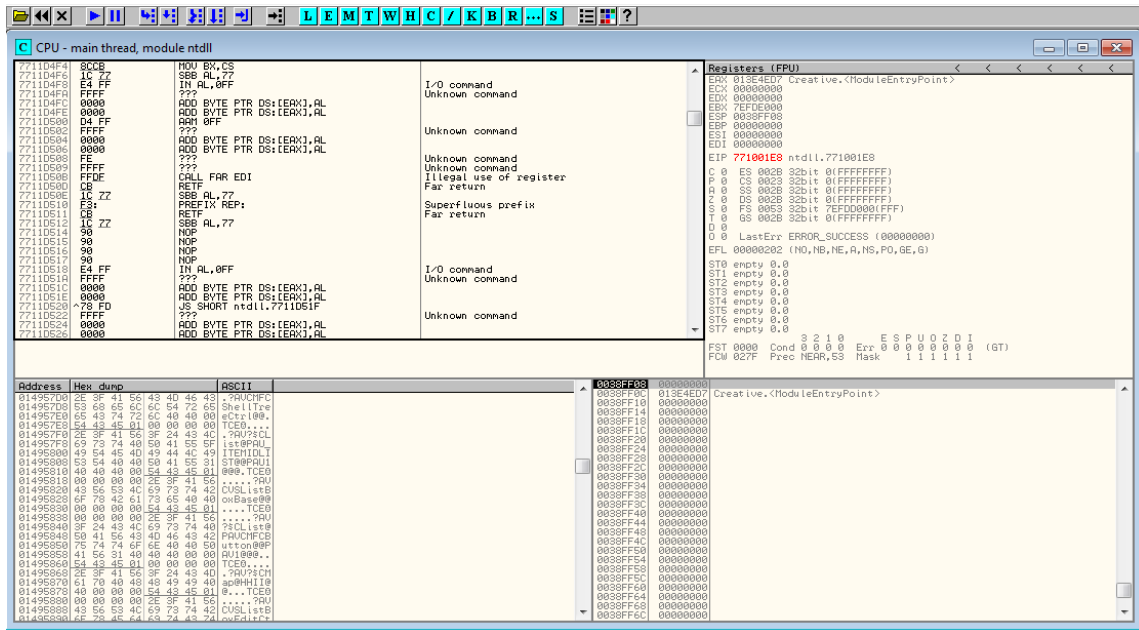


Figure 10: amltib. dll opened up in a debugger.

The true implementation of the validation check bypass is actually quite simple: a few well-placed jump instructions as shown below (you can see all the jumps exemplified by the arrows on the left-hand side of the image).

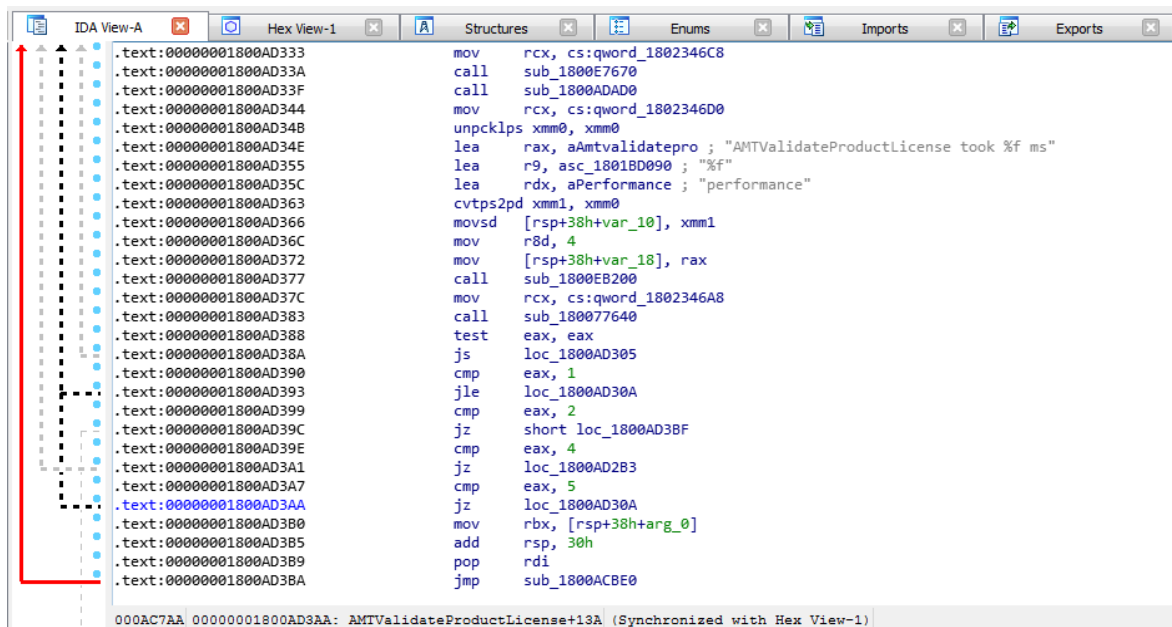
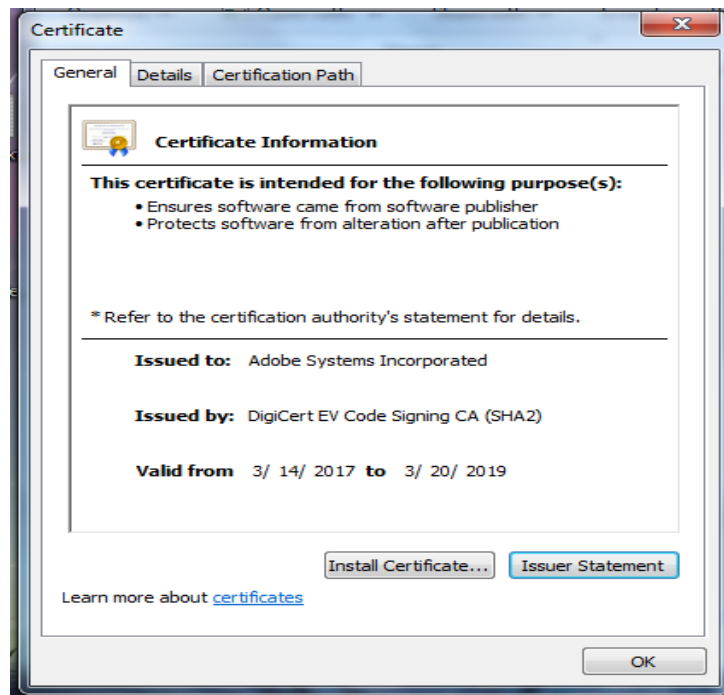


Figure 11: Assembly code handling the license validation.

Now instead of first checking whether a license is valid or not, the jump instruction simply tells the program to continue on as if the license was valid in the first place. We had to jump through a few hoops as to follow the underlying protocol, though in effect, it works as an

instant bypass. However, changing this file does not come without its side-effects. As you can see below, the `amtlb.dll` comes with a digital certificate.



**Figure 12: Digital certificate accompanying the `amtlb.dll`.**

Thus any changes to the `amtlb.dll` file will make the certificate invalid, though this does not really affect the cracking process; it simply states the file has been modified. The main takeaway however is, if you can understand code at a very low level such as in assembly or even binary, you can reverse engineer anything.

The main issue with this process however is not necessarily the `amtlb.dll`, but the fact that the only difference between the trial version of the software and the paid version of the software is that the trial version has a 30 day counter before it expires, meaning it already includes all the premium features. Thus, how can you expect users to pay for what they already have if removing a counter is the only thing they really have to worry about?.

This is because the problem with security in the client/server architecture is that security measures implemented on the client side can almost always be cracked much easier than security measures implemented on the server side. In fact, the assumption that you can never trust the client (guilty until proven innocent) is quite prevalent among developers who understand the value of security. As such, it is important to understand the nature of your

security measures and where they take place, which is precisely why Adobe started pushing their subscription-based model through the Creative Cloud Suite.

Creative Cloud was more of a business-minded fix than a coding-minded fix as hackers will always find a way to break security measures; it will forever be an unwinnable war. However, Adobe could have still make the job of the hacker much harder by removing blatant reference comments to important pieces of code, as well as implementing validation checks across multiple dll files. Regardless, Creative Cloud seems to be doing a decent job at converting more trial users to paid users.

This was done so by switching to a subscription-based model rather than requiring a lump-sum payment. Instead of paying around \$1,000 for a copy of Photoshop like before, you can now receive a suite of Adobe's image-editing software for around \$10 a month through Creative Cloud. From this simple change in price alone, many past abusers have decided to simply switch over to being legitimate paying users rather than pirates, also being able to receive the latest software updates which they could not do before in risk of invalidating their activation patch. On top of this, part of the solution came from the new protocols that implement Creative Cloud. Creative Cloud comes with a lot of extra logging subscribers that constantly monitor and send information about your system to Adobe and its applications. Due to these extra redundancies in place, disabling them may prove to be difficult for nontechnical users as Creative Cloud is still a relatively new aspect of Adobe, and not much information regarding patching its underlying methods are released yet. After all, as stated before, the vast majority of pirates do not truly understand how to crack software, they simply download and run pre-existing patches, so implementing this form of multi-layered security will definitely help.

#### **[XVI] Additional Findings**

It should also be noted, one such finding was the application.xml file which contains important initial configuration information relevant to the startup of the program. As it turns out, this file can be modified to use Adobe trials indefinitely by simply changing the serial number assigned to the product. There are  $10^{24}$  possible serial numbers, with not even  $10^{10}$  humans on this planet, leaving plenty of free trials for malicious users to use up.

Figure 13: application.xml file.

## [XVII] Solutions

The solution for the Photoshop DLL file hacking is three-fold, where applying one or two of the solutions would get some added security, but implementing all three should dramatically cut down on the hacking and pirating of Adobe Photoshop software and in turn other software on the market today. The three tiers in this solution combine standard common sense, standard cryptography, and the use of some new age technology.

## [XVIII] Analysis Step 1

Remove the “License check shows product is still in trial...” that links to the specific location of code to be changed. Adobe will also need to adjust the code to place the reference and license checks in multiple locations.

The Adobe comment section is riddled with issues stemming from telling the user exactly where the reference for trials is located and displaying too much information to allow amateur level hackers to easily navigate and adjust the code inside the DLL file itself. These issues not only allow inexperienced hackers users to find and navigate specific code, but it also allows them to easily search and modify for the specific code that needs to be adjusted in the file. There are usually standards for code comments but unfortunately Adobe comments are too detailed and allow non-technical users to easily adjust code and implement hacks on the DLL files themselves. With this being said, the comment section should include the basic understanding of code but not detailed key words that could be used to search and adjust by any non-technical customer. This problem might also be affected by the linking of read-only data while the program is being compiled, thus making unlinking it in some way even harder.

In combination with this simple step Adobe should also place nonces and license checks in multiple locations throughout different files without comments that tell hackers where to adjust or even place skip functions to circumvent the checks altogether.

### [XIX] Analysis Step 2

Include the user's ID or personal information, or possibly using the public key of the user to encrypt the files and including multiple nonces and multiple license regulations across many files.

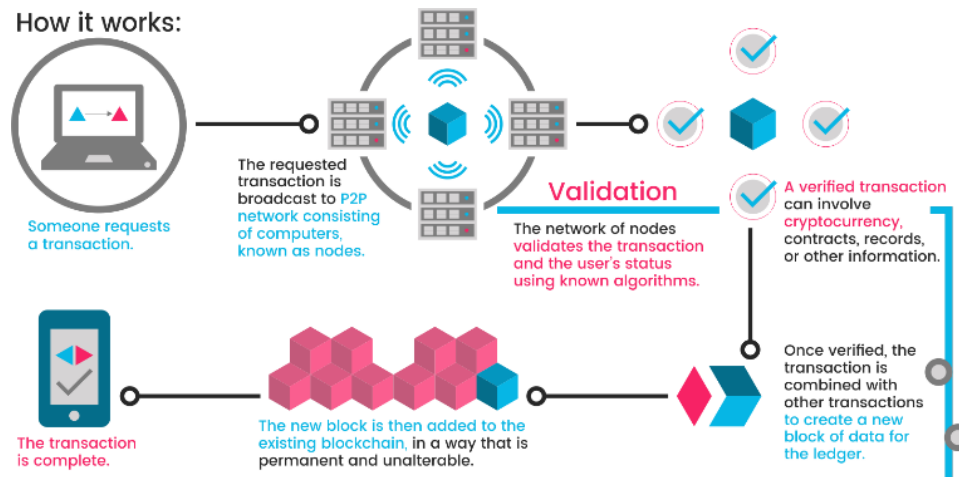
The second step in securing the Adobe DLL files would need to include the users public key in the encryption of these files. It would also benefit Adobe to include multiple nonces inside multiple files. The benefits of these added security parameters would mitigate non-technical customers from easily modifying the DLL files or distributing pirated information. Adobe will have to carefully secure this information and include it in the files sent to the client through a secure API similar to that suggested by Microsoft on their "*Securing your DLL files*" website. Thus, if the customer purchases the software all of the files and the nonces are encrypted with that "customers device public key" restricting them to use the software by unlocking with their private key. No customer will want to hand out their private key to unlock the files and distribute the pirated software, thus exponentially reducing the spread of the `amtlb.dll` files. Adobe would also need to secure the license regulations verification steps and validate the license regulations across multiple locations and files. These regulations will also help deter hackers by making them manipulate multiple locations without code that will help them easily navigate and modify the files, as compared to right now where only one simple file is doing all the work.

Another idea to preventing software piracy is to make the software free for only specific base features where Adobe doesn't give all of the files for the fully licensed version to the user, presented as a "true trial version". This minor step would have prevented millions from circumventing the system and cracking the software without any money being spent. This would have allowed Adobe to monitor the extra subscriptions and individually give access to different features and new updates to only the paying customers, as a connection to the server is required to deliver additional premium features to the base version. There is hope though, with the next step Adobe could implement these updates and future modifications using a BlockChain subscription.



**[XX] Analysis Step 3**

The use of a BlockChain solution where you only get 3 to 5 keys, and each time you use a key the total is reduced.



**Figure 14: How blockchain works (BlockGeeks).**

BlockChain is the latest craze, but it is also a great way to implement a public record or “ledger” on license and subscriptions. Take for example the purchase of Photoshop from Adobe: first you get a product key and then you can use that key on up to 5 devices. Similarly, we can show the example of Adobe having 5 coins representing the 5 devices of the Adobe license given to the customer at time of purchase. The ledger is kept by Adobe and every time the customer downloads the software and runs the subscription the key is checked and that will use one of the coins. When you get to 5 coins being used by 5 devices you have no coins left so you can’t give out another key or in this case another coin. Another benefit to using the BlockChain solution to monitor and regulate the product keys so that when you want to cancel your subscription all you need to do is withdraw your digital signature which would void the ledger and cancel the contract. This would automatically cancel the subscription on all 5 accounts and make the ledger null and void.

In conclusion these steps will not fully secure and protect against hacked DLL files and pirated software, but it will dramatically cut down on the problems due to customers not wanting to give out their private key to other users to unlock the software. It will also reduce the ease of knowing where and how to change and adjust the code in the trial version, and with the new blockchain technology all parties included in one license will have a ledger to see all users of that particular software.

## [XXI] CONCLUSIONS

In this paper we discussed the basic understanding of software, licensing, DLL files, and why Adobe Photoshop is a prime target for DLL cracking, and thus at the forefront of hacked software distribution. We also would like to point out that there is no “golden bullet” or “perfect solution” to counteracting software piracy or DLL hacking for that matter, but we do suggest 3 interesting solutions to building a better barrier to the ever expanding problem. Again, this doesn't mean these steps will solve the issue, but they will start to prevent the widespread public knowledge of easy hacking implementations on Adobe software as a whole.

Note: Please note this paper and the authors in no way condone the piracy of software or hacking of DLL files and distributed software in general, but we do want to make aware the issues with software piracy, the understanding on file architectures, and some possible solutions to fight the never-ending battle of software hacking and piracy.

## REFERENCES

1. Adobe. (n.d.). Retrieved from Adobe: <https://www.adobe.com/>.
2. AMT Emulator: Universal Adobe Patcher, SAYS. (12/22/2017).
3. BlockGeeks, Ameer Rosic, Oct 1, 2016.
4. Fitzgerald, B. (2017, 12 06). Software Piracy: Study Claims 57 Percent of The World Pirates Software. Last retrieved from Huffington Post, May 2018.
5. Getting Started with Reverse Engineering, 2015.
6. Gibert-Knight, A. techsoup. “Making Sense of Software Licensing”, 2012.
7. Microsoft, adobe lose \$13.5bn to piracy, 2011.
8. Nagpal, S. What will happen if Adobe found that I have used piracy software before, 2017.
9. OllyDbg. (n.d.). Retrieved from OllyDbg: <http://www.ollydbg.de/>.
10. Quora, How do People Crack Computer Programs and Games, 2017.
11. Rouse, M. (n.d.). Authentication -Search Security, 2017.
12. Stack Overflow: .dll to .exe relation, Jon Cage, May 10, 2011.
13. Ali Rantakar Validating In-App Purchases in your iOS App, February 10, 2015.
14. K. Kaur and X. Xiaojiang Du and K. Nygard, “Enhanced routing in Heterogeneous Sensor Networks”, IEEE Computation World'09, pp. 569-574, Athens, Greece, Nov. 15-20, 2009.

15. Lauren Evanoff, Nicole Hatch, Gagneja K.K., "Home Network Security: Beginner vs Advanced", ICWN, Las Vegas, USA, 2015; July 27-30.
16. Gagneja K.K. and Nygard K., "Heuristic Clustering with Secured Routing in Heterogeneous Sensor Networks", IEEE SECON, New Orleans, USA, 2013; 51-58.
17. Gagneja K.K., "Knowing the Ransomware and Building Defense Against it - Specific to HealthCare Institutes", IEEE MobiSecServ, Miami, USA, 2017; 1-5: 11-12.
18. Gagneja K.K., "Secure Communication Scheme for Wireless Sensor Networks to maintain Anonymity", IEEE ICNC, Anaheim, California, USA, 2015; 1142-1147.
19. Gagneja K.K., "Pairwise Post Deployment Key Management Scheme for Heterogeneous Sensor Networks", 13th IEEE WoWMoM, San Francisco, California, USA, 2012; 1-2.
20. Gagneja K.K., "Global Perspective of Security Breaches in Facebook", FECS, Las Vegas, USA, 2014; 21-24.
21. Gagneja K.K., "Pairwise Key Distribution Scheme for Two-Tier Sensor Networks", IEEE ICNC, Honolulu, Hawaii, USA, 2014; 1081-1086.
22. Gagneja K., Nygard K., "Energy Efficient Approach with Integrated Key Management Scheme for Wireless Sensor Networks", ACM MOBIHOC, Bangalore, India, 2013; 13-18.
23. Gagneja K.K., Nygard K., "A QoS based Heuristics for Clustering in Two-Tier Sensor Networks", IEEE FedCSIS, Wroclaw, Poland, 2012; 779-784.
24. K. K. Gagneja, K. E. Nygard and N. Singh, "Tabu-Voronoi Clustering Heuristics with Key Management Scheme for Heterogeneous Sensor Networks", IEEE ICUFN, Phuket, Thailand, 2012; 46-51.
25. Gagneja K.K., Nygard K., "Key Management Scheme for Routing in Clustered Heterogeneous Sensor Networks", IEEE NTMS, Security Track, Istanbul, Turkey, 2012; 1-5.
26. Runia Max, Gagneja K.K., "Raspberry Pi Webserver", ESA, Las Vegas, USA, 2015; 27-30.
27. S. Gagneja and K. K. Gagneja, "Incident Response through Behavioral Science: An Industrial Approach," International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, 2015; 36-41.
28. Tirado E., Turpin B., Beltz C., Roshon P., Judge R., Gagneja K., "A New Distributed Brute-Force Password Cracking Technique", Future Network Systems and Security, FNSS Communications in Computer and Information Science, 2018; 878: 117-127.

29. Caleb Riggs, Tanner Douglas and Kanwal Gagneja, "Image Mapping through Metadata," Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC), Shanghai, China, 2018; 1-8.
30. Keely Hill, Gagneja K.K., "Concept network design for a young Mars science station and Trans-planetary communication", IEEE MobiSecServ, Miami, FL, USA, 2018; 24-25.
31. Javier Campos, Slater Colteryahn, Gagneja Kanwal, "IPv6 transmission over BLE Using Raspberry PI 3", International Conference on Computing, Networking and Communications, Wireless Networks (ICNC'18 WN), March, 2018; 200-204.
32. Gagneja K., Jaimes L.G., "Computational Security and the Economics of Password Hacking", Future Network Systems and Security. FNSS. Communications in Computer and Information Science, 2017; 759: 30-40.
33. Gagneja K.K. Ranganathan P., Boughosn S., Loree P. and Nygard K., "Limiting Transmit Power of Antennas in Heterogeneous Sensor Networks", IEEE EIT, IUPUI Indianapolis, IN, USA, 2012; 1-4.
34. Keely Hill, Kanwalinderjit Kaur Gagneja, Navninderjit Singh, "LoRa PHY Range Tests and Software Decoding - Physical Layer Security", 6th IEEE International Conference on Signal Processing and Integrated Networks (SPIN 2019), 2019; 7 - 8.
35. Alexandro Riuz, Carloas Machdo, Kanwal Gagneja, Navninderjit Singh, "Messaging App uses IRC Servers and any Available Channel", 6th IEEE International Conference on Signal Processing and Integrated Networks (SPIN 2019), 2019; 7 - 8.
36. C. Riggs, J. Patel and K. Gagneja, "IoT Device Discovery for Incidence Response," Fifth Conference on Mobile and Secure Services (MobiSecServ), Miami Beach, FL, USA, 2019; 1-8.
37. S. Godwin, B. Glendenning and K. Gagneja, "Future Security of Smart Speaker and IoT Smart Home Devices," Fifth Conference on Mobile and Secure Services (MobiSecServ), Miami Beach, FL, USA, 2019; 1-6.
38. Sasko A., Hillsgrove T., Gagneja K., Katugampola U. System Usage Profiling Metrics for Notifications on Abnormal User Behavior. In: Doss R., Piramuthu S., Zhou W. (eds) Future Network Systems and Security. FNSS. Communications in Computer and Information Science, 2019; 1113.