4/16/2023

OT Device Software Asset Extractor Binder

Sponsored By: FoxGuard Solutions

Project Name: OT Device Software Asset Extractor

Members: Jack Benning, Keaton Boodlal, Simple Gomez, Anthony Lee, Sam Stewart, Henry Trochlil

Customer: Colin Grant w/ FoxGuard Solutions

SME/Mentor: Dr. Joe Adams

Table of Contents

Project Background

FoxGuard Solutions is a Christiansburg, VA based cyber security company focused on protecting our nation's critical infrastructure such as power generation, transportation, and oil/gas. Many of these infrastructure devices, or Operational Technology (OT) devices have a large amount of software installed in order to function properly. This software can introduce security vulnerabilities to the system, and it is important to perform security patches when relevant in order to prevent these vulnerabilities from being exploited. Due to the nature of so many applications for OT devices, the consequence of these exploits can be catastrophic. Per CISA's Cybersecurity Performance Goals, the most beneficial, yet costly method of building a cybersecurity program is to inventory all software assets to determine what vulnerabilities and patches are relevant on a system. The current methodology of maintaining a list of software assets involves scanning each device, which impacts the very operation of the device that the scan is intended to protect. Additionally, the current approach needs to be performed entirely onsite.

This project evaluates the feasibility of extracting software assets from an offline backup file containing the OT device's environment. This is done to limit the impact on the efficiency/functionality of maintaining these software asset lists.

Objectives/Requirements/Constraints

Develop a Windows executable that takes in an OT device's backup file, and outputs a JSON formatted list of installed software. The host machine must be physically disconnected from the OT device, and air-gapped. The extractor must support Acronis, VEEAM, and

Windows Native Backups. The extractor also needs to have support for Linux & Windows OT backups.

Engineering Standards:

The first major standard that the project adheres to is ISO/IEC/IEEE 12207. This standard contains processes, activities, and tasks that apply during any period of development of a software product.

The second major standard that the project adheres to is ISO/IEC/IEEE 15939. This standard relates to identifying a process for defining, measuring, and communicating the performance of a system for the purpose of continually improving its performance throughout its development.

The third major standard that the project adheres to is NIST 800-171 r2. This standard provides security requirements for protecting confidentiality of non-federally controlled unclassified information. This standard applies particularly to how FoxGuard utilizes the Asset Extractor project, in terms of their treatment of customer information.

Detailed Design/Specification

<u>Overall Research and Approach:</u>

Our approach to this problem was to write our program using the C++ programming language. This is because all members of the design team have experience writing programs in C++ as the ECE curriculum at Virginia Tech uses this language. In addition, we have chosen our IDE to be Microsoft Visual Studio 2019/2022, since it is able to generate a Windows Executable file of our C++ program which is a customer requirement.

In order to gain a full understanding of how computer images work, the team first did some research on the internal workings and architecture of a given computer image. Through this research, we were able to learn that there are several layers to a computer image. A figure describing the architecture and layers can be seen in Figure 1 below.

Figure 1: Computer Image Hierarchy.

This was extremely helpful as it allowed us to understand the way we should iterate through the computer image to locate the desired files we needed. The next step in our research was to look into the different image formats that are out there. We discovered that there are many different types of formats used for storing computer images so we had to consult with our customer to determine which image formats should be supported. After discussing this with the

customer, we focused on three different computer image formats: VHD/VHDX, Acronis, and Veeam. We then researched existing C++ libraries that provided us with the file forensic capabilities we desired. This is when we came across The Sleuth Kit (TSK). The Sleuth Kit is a collection of command line tools as well as a library written in C (and compatible with C++) that provides various file forensic capabilities that we need in order to extract the necessary information from the computer image backups. We were fortunate enough that the developers of the program provided their source code and API documentation so that developers (like ourselves) can use their library inside of their own programs. In addition, the team discovered that TSK only supported the VHD/VHDX and VMDK format. So to solve this issue, we wrote a PowerShell script that uses proprietary tools provided by Acronis and Veeam to convert the Acronis and Veeam backups into the VHD/VHDX format that is then fed into our program. After we were satisfied with the amount of research that was conducted, we moved onto the development phase of the project.

For our development environment, we used Visual Studio 2019/2022 as it provides support for the C++ language. The first step of the development phase was to download and build TSK using the documentation provided so that we can connect it with our own Visual Studio Project. Our coding project is called VHD_Extractor_Module and our development code is located in VHD_Extractor.cpp. To connect TSK with our project, we added the necessary dependencies and its paths to the project properties. More specifically, we included the path to the library's header files, .lib, and .dll files in the compiler and linker settings of the Visual Studio project properties. Then we verified that our program was able to interact with the library by using some basic methods from TSK.

Once we were able to verify that our program is able to interact with TSK, we first started by programming the command line parameters for the program's executable. We have two command line parameters, the first parameter is the name of the computer image file to be analyzed. The second parameter is the Operating System type flag (-W for Windows and -L for Linux) to specify to our program which method should be called for analysis. Throughout our development, we maintain code modularity and allow for a clear design by having each image layer (Figure 1) represent a method in our program. In order words, we have a method that is responsible for handling each layer in the computer image hierarchy.

Windows Development:

The first method we created in our program is called 'VHDX_ExtractorWindows'. This method is responsible for verifying and opening the provided computer image file. This is done by creating a 'TskImgInfo' object using the Sleuth Kit library. Using TskImgInfo's open method, we assign the provided file to the object. If the library is unable to locate the image, the TskImgInfo object will be assigned to null. We check to see if it is null and return a error if it is. If it is not an error, we feed the object as a function parameter into a method called 'processVolumeWindows', which is responsible for analyzing the volume system.

Inside of the volume system, there are many partitions that need to be looked at. We use TSK's 'TskVsInfo' object to open up the volume system. This allows us to determine the number of partitions there are in the volume system. We then iterate through each partition and determine if the partition is unallocated or if there exists a file system (allocated). Unallocated partitions are ignored since they don't contain any data. If the partition does contain a file system, we feed the found partition to the method called 'findWindowsBinary' which is responsible for analyzing the file system in that partition.

The file system represents the file directory architecture that the computer image has. In a Windows Environment we are interested in two files: a DAT file called ‘NTUSER.DAT’ that is generated for each user, and a file called ‘SOFTWARE’ where there is only one of its kind for the entire image. The location of these files are located at ‘C:\Users\[user]\NTUSER.DAT’ (where [user] represents a username) and ‘C:\Windows\System32\config\SOFTWARE’ respectively.

The DAT file generated for each user contains registry keys for the settings and software that is installed for a specific user account, while the ‘SOFTWARE’ file contains registry keys for software that is installed system wide on the computer.. We use both of these files to extract a complete list of software installed on the computer image. This is done by using the TSK file system functionality to traverse the file system to the locations of all possible ‘NTUSER.DAT’ files for each user and to the location of the ‘SOFTWARE’ file. We then create local copies of these files by using a memory buffer to read the contents of the files 2048 bytes at a time and then write those buffer contents into a separate file that we generate. We repeat this process until the entire file is read.

Using the locally generated copies of the ‘NTUSER.DAT’ and ‘SOFTWARE’ files, we can now read the registry contents by using the library LibREGF [5] to extract the desired information to formulate the list in a JSON output. More specifically, LibREGF goes through each file which represents a registry hive. Within those hives, they contain keys and each key has subkeys. The subkeys contain the programs that are installed which is the information we desire. Using LibREGF we iterate through each key and subkey to extract the values we need and then use those values to generate the nicely formatted JSON output using a C++ JSON library in a file called ‘software.json’. This can then be used by our sponsor FoxGuard to analyze which

software components are outdated and require an update and perform the necessary security patching.

Linux Development:

When the Linux Extractor is called based on the input flags, the first thing it does is it finds the distribution of the image. Initially calling on findLinuxBinary, which sets up all of the needed things for the following function calls. It first calls on the ExtractLinuxType function call, which recursively searches the file system for the “os-release” file. Once this is found, it calls upon the printFile function to write the file to disk. Once it is written to disk, the parseForDistro function is called: this function parses the file for the distribution (Ubuntu, Fedora or CentOS) and the version. These are then added to the json output and the two of these are returned in a vector. The ExtractLinuxType then takes the distribution from the returned vector and returns it. Once the distribution is returned to the findLinuxBinary function, it uses the distribution to decide the next function call. If it is Ubuntu, it calls on extractDirectoryFilesLinuxUbuntu, likewise for Fedora and CentOS with the function calls being extractDirectoryFilesLinuxFedora, extractDirectoryFilesLinuxCentOS.

Ubuntu: Ubuntu stores all of its package information in a text file. Once extractDirectoryFilesLinuxUbuntu is called, it looks for the file named “status” which is located in the “/var/lib/dpkg/” directory. It works by parsing every entry in the directory until it finds each of the required names recursively calling on extractDirectoryFilesLinuxUbuntu when needed. Once the status file is found, it prints the file using processSoftware. Once the file has been printed, using txtToJson, it then parses the status file for the name, version and vendor.

Once it has found all 3 of these components, it puts these into a json object and adds it to the json file that was originally made when it found the distribution.

Fedora: The method for Fedora is slightly different than the method for Ubuntu. Once extractDirectoryFilesLinuxFedora is called, it looks for the SQLite database called “history” which is stored at the path “/var/lib/dnf/”. Similarly to Ubuntu, it parses each directory for the name of the subsequent directory, going into that directory recursively until it finds the “history.sqlite” file. Once this file is found, it is printed to disk using printFile. After this is done, getDatasqliteFedora is called. This function uses the sqlite3 library to parse the database columns with the information that we need, which from Fedora, is name and version number. The information for the Vendor is compressed in such a way that we have not been able to access it as of now. It then adds the name and version number to the json file that was originally created when the distribution was found.

CentOS: The method for CentOS is almost identical to the method for Fedora. Once extractDirectoryFilesLinuxCentOS is called, it looks for the SQLite database that is named approximately “history.sqlite”, it is “history-time.sqlite”. It is in the “/var/lib/yum/history” directory. Using the same method as the previous two, recursively looking through the file system, it finds the history.sqlite file. The file is printed to disk using printFile. Using getDatasqliteCentOS, which uses the sqlite3 library, it takes 3 columns which contain the name, version number and vendor, and uses those columns to add to the json file that was originally created when the distribution was found.

SQLite Databases: To read the sqlite database we used the sqlite3 library using the command ‘sqlite3_open’ to specify the file. Similarly to querying a database, the library allows us to prepare a query by using the command ‘sqlite3_prepare_v2(db, query, -1, &stmt, NULL)’ where query is the request to access the data in the file. Before preparing the query we have to instantiate a statement object (stmt) which represents a single SQL statement that has been compiled into binary form and is ready to be evaluated in the ‘sqlite3_prepare_v2’ method.

LVM: To add support for Linux images that use the Logical Volume Manager (LVM), we had to add code into the existing method ‘processVolumeLinux’ method. More specifically, if the volume system could not be opened, we had to check if the memory address that we were reading was actually classified as a pool of volumes. To do this we used the method ‘tsk_pool_open_img_sing’ to feed in the image file and provide the memory address of where it is located and created a TSK_POOL_INFO object type that represents the pool of volumes. If the pool does not exist, the object type will be assigned to NULL which we use for error handling. If the pool does exist, we use the method ‘get_img_info’ to retrieve an image type that represents the pool layer. This image is then fed into the normal Linux extracting implementation to retrieve the desired data.

JSON Implementation: For json implementation, we used a library called nlohmann/json, which adds a new class called json. To make a json output, we add multiple json objects to one and then use the dump method to put the json into a file.

Test Cases:

The test cases we wrote were based on the extractor software using .vmdk and .vhd/.vhdx images. There was some tests involving our script for VEEAM to make sure it called what it needed to in order to extract the backup into the appropriate file, but VEEAM/Acronis to

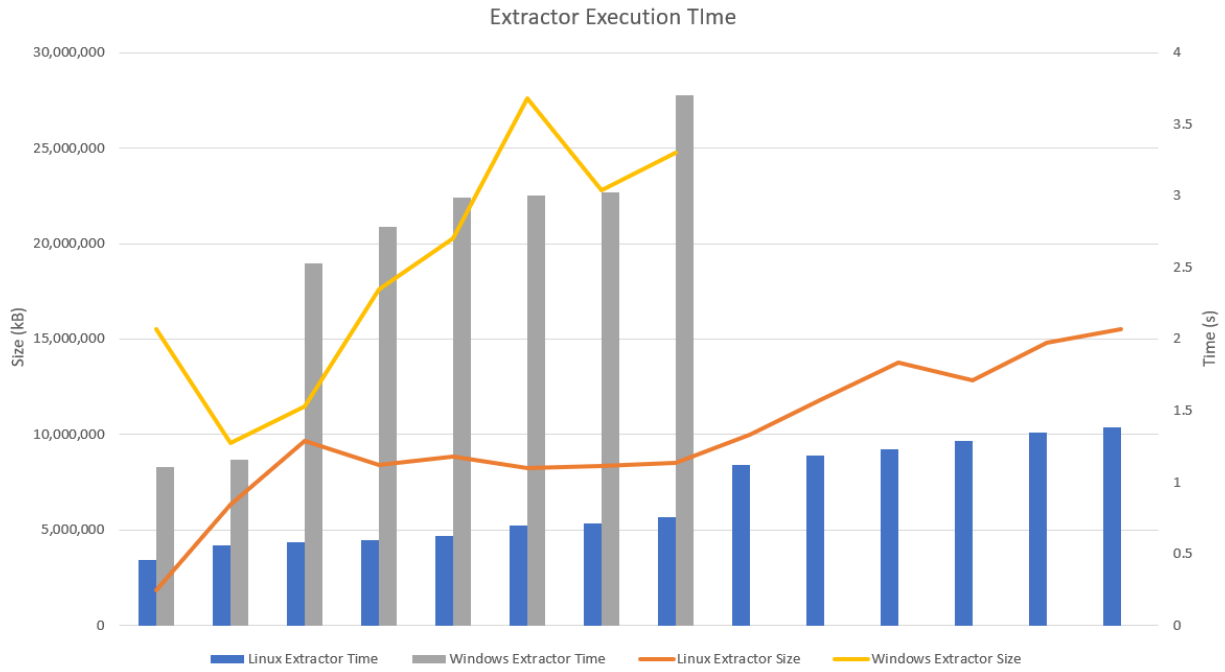
.vmdk/.vhd file conversion was not focused on as we trust that their software can extract their own files as advertised. For the test cases of each tested operating system, which were Windows, Ubuntu, CentOS and Fedora, periodic backups were made as more software was installed on the virtual machines. At the first installation of the operating system, and after each group of new software was installed, the backup was copied to a separate folder and the size of the backup was recorded. Once the backups were made, simple python scripts that recorded the runtime of the respective extractors over the course of 5 runs, returning the average, were used. These gave us the runtime of the extractors when compared to our size, which we took as our results. From there, we compared the list of the extracted software to the list of what was installed, marking those that were found with the correct version and vendor or link as ‘found’, and those not found as ‘not found’. This gave us our accuracy rating, and as an effort was made to use several different installation methods this is an accurate representation of the accuracy even with varying installation methods.

Project Status / Results Analysis:

As of right now, our project has satisfied the majority of the customer’s requirements. More specifically, our software extractor program is able to support the Windows operating system completely. On the Linux side, we have full support for Ubuntu and and CentOS Linux Distributions but only have partial support for Fedora Linux Distributions. This is because as of right now our program is not able to retrieve the vendor information since Fedora stores them in a unique location that requires more development time that we do not have. But we will be providing detailed documentation on the research we conducted and our findings so the project can be continued. We were also able to add XFS and LVM support for the majority of Linux image types. Some images are currently not supported because we have discovered that some

images place the necessary files we need in different locations. Further research in this matter will need to be done.

The results across the board were very satisfying, with 100% accuracy rating on Windows, Ubuntu and CentOS machines and an extremely high accuracy for Fedora, with the location of the missing software in the operating system having been found and recorded for FoxGuard's continuation with the project. The runtime never had an average above ~3.7 seconds, and while the runtime is dependent on the computer and the computer's load at the time, in the worst case scenarios it would still have a very low runtime. Also, while there is a correlation between the size of the backup and the length of the runtime, as was expected, we do not believe it to be linear but instead that it would plateau after a few more increases in size and therefore even with extremely large backups the time would still be minimal. The graph for the runtime displaying the data referenced above can be seen below.



Key Accomplishments:

As per the objectives of the project, we accomplished supporting three backup file types: Acronis, Veeam, and Windows Native. Additionally, we developed modules to extract software into a .JSON list format with their name, version number, and vendor for both windows and three flavors of Linux: Ubuntu, CentOS, and Fedora from a given backup file. Note currently Fedora is not able to provide software vendors. Our tool is packed into an executable and runs from the command line, it's efficient and has a 100% accuracy at detecting all installed software on Windows, Ubuntu, and CentOS.

Customer Satisfaction:

During our meetings, Colin Grant, the representative who oversaw our project for FoxGuard Solutions, gives us great feedback on our progress, and is sometimes surprised by what we deliver. Overall, our customer is very pleased with our work, as quoted from Colin Grant “Fox guard is excited at the potential that this team has demonstrated with their proof-of-concept implementation. We intend to transform their work into a meaningful product and are already introducing the concept to some major customers as we expand our support of their systems,” showing great enthusiasm to use the tool we created. Additionally, praising the team’s diligent work and problem-solving skills, “The team demonstrated the ability to learn and adapt throughout the project. They faced unexpected challenges while developing the software. These challenges were often overcome by clever solutions to tricky problems, or the team recognized the impracticality or fragility of solving the challenge and redesigned around the challenge to avoid it.”

Deliverable Status:

Our project is completely software based and therefore does not require any hardware deliverables. To ensure that our customer has everything they need to set up the coding environment and run our software extractor, we will be providing the following deliverables:

- Documented Coding Package:
    - Includes all necessary libraries
    - Commented implementation code
    - Any modifications made to original library code (Adding XFS support)
- Instructions on how to set up Visual Studio Project on machine
- Instructions on how to use the program
- Final Project Binder (this document)
- Program's ready to run executable file and related Dynamically Linked Libraries (.dll files)
- Computer Images used to test.
- Testing Methodology and Data

In addition to these deliverables, we will also be providing detailed documentation/research of where to locate various files needed to extract software vendor information in Fedora Linux distributions as we did not have time to complete that functionality of our software extractor. All of these deliverables will ensure the customer is able to achieve the following:

- Use our program to extract a list of software containing the software's name, version, and vendor from a computer image
- Understand the internal workings of our program
- Allow the customer to use our code for further project exploration and development.

Challenges:

During the development process of the project the team encountered many different challenges. The first and largest challenge was that of reverse engineering the proprietary backup file types of both Acronis and Veeam. In our original design of the product we intended to have our program use the backup files as direct inputs. This meant adapting The Sleuth Kit (TSK) to be compatible with Acronis and Veeam backup files, which would require a very in-depth understanding of the structure of these files. We quickly realized that due to the number of variables that affect these backup files and the minimal experience in file type reverse engineering on the team (amongst other challenges), that given our schedule constraints, we would not be able to implement this functionality as we had planned. This led us to utilize the proprietary extractor tools that we implemented in our final product.

Another very consistent challenge the team faced was inconsistencies between operating systems, especially for Linux. For the most part, all the versions of Windows that we ensured compatibility with stored their information in the same way. The only difference was in the way they stored a few specific details about the OS version, but this was easy to navigate around, and the software extraction methodology was identical across all versions. Perhaps most importantly, all of the methods Windows uses for filesystem storage (basic partitioning, NTFS, exFat, etc.) are all natively compatible with TSK, and thus didn't require any extra work. The same unfortunately could not be said for Linux. Because Linux is a much larger family of operating systems, with a much greater variety of technologies in use, we had to address more individual exceptions. Ubuntu didn't pose any large challenges, as its filesystem and package manager file format didn't require any external libraries. Fedora and CentOS, however, both required external libraries and additional code for full compatibility. Both operating systems use package

managers that store information in SQL databases, which require an external library to parse the necessary information out of. In addition, CentOS uses XFS and LVM by default, neither of which are natively implemented in the Windows source code of TSK. With every new library or piece of functionality we needed to add, there were inevitably challenges with writing the code implementation and/or issues with compiling the final executable. However, in the end, we were able to overcome all of the challenges we encountered to create a successful final product.

Schedule:

Our team has decided to use Microsoft Projects to create a Gantt chart to provide our customer with the most up to date information about our project progress and deadlines. The Gantt Chart is updated every week when we email our customer and mentor our weekly progress report. These reports provide a more detailed version of what was accomplished and our schedule moving forward. Here is the link to our Gantt Chart: Link

Cost Status:

The only expense we had while working on this project was the purchase of 4 Acronis Licenses to help with the process of Reverse Engineering. Through the use of these licenses, we found the method that we are currently using, which is the backup converter. The four licenses totaled to $85.

Takeaways:

Overall, the team is happy with the work we have done across both semesters and is proud that we could provide a valuable product to our customer. Looking forward, the team has learned valuable lessons about software development as well as the engineering project management

process that we hope to put into practice once we join the workforce. For our customer, FoxGuard, this project proved that an automated process for compiling the list of installed software on OT devices is feasible and would be greatly beneficial once put into use. Despite the somewhat limited scope of compatibility due to our development timeline, it wouldn't be unreasonable for FoxGuard to extend our product into one that has a much wider range of use cases, further increasing their process efficiency.

Acknowledgements:

The team would like to acknowledge and thank the following individuals for their contributions to the team's success:

- **Dr. Joe Adams -** *Mentor and Subject Matter Expert to the team*. We would like to thank Dr. Adams for giving feedback, answering our many questions, and helping guide us to success throughout the course of both semesters of this project.
- **Colin Grant -** *FoxGuard Representative.* We would like to thank Colin Grant for the feedback, insight, and industry knowledge he gave the team throughout the project.
- **Brian Carrier -** *The Sleuth Kit Developer.* We would like to thank Brian Carrier for his work on The Sleuth Kit library, which was integral to our design and development of our final product.
- **Joachim Metz -** *Libyal Developer.* We would like to thank Joachim Metz for his work on a multitude of libraries (libregf, libvslvm, libvhdi, etc.) which our team implemented in our project.