Automated PCB Reverse Engineering

Stephan Kleber
stephan.kleber@uni-ulm.de

Henrik Ferdinand Nölscher
henrik.noelscher@uni-ulm.de

Frank Kargl
frank.kargl@uni-ulm.de

Institute of Distributed Systems, Ulm University

Abstract

The very first step for analyzing the security of an embedded device, without prior knowledge of the device's construction, is analyzing the printed circuit board (PCB) of the device, in order to understand its electrical implementation. This analysis is called PCB reverse engineering and its results are a list of components, technical documentation related to those components, and a schematic reconstruction, that illustrates basic connections between the PCB's components.

Motivated by the lack of inexpensive methods for efficiently performing PCB reverse engineering, we propose a novel framework that formalizes and automates most of the tasks required for PCB reverse engineering. The framework is capable of automatically detecting components using machine vision, gathering technical documentation from the internet and analyzing technical documents to extract security-relevant information. We implement the concept and evaluate the gain of efficiency and analysis coverage. Our results show that automating almost all steps of PCB reverse engineering is possible. Furthermore, we highlight novel use cases that are enabled by our approach.

1 Introduction

More and more embedded devices pervade business processes and personal activities. From a security perspective this leads to an increased attack surface and higher risk by the common utilization of devices. Securing software can only reduce part of this risk, since in particular embedded devices' security properties are dependent on the underlying hardware platform. Therefore, an integral part of a thorough penetration test, to assess the security of a device, is to analyze the hardware implementation, as demonstrated by Grand [4] and Oswald et al. [14]. The common basis of all devices is their hardware implementation as printed circuit boards (PCBs) which contain the physical circuitry to connect all components, such as microchips. To assess the attack surface and the security risk induced by the hardware design of a device, the analysis of the employed PCB is necessary. To this end, a penetration test is a structured process of searching, confirming, and reporting vulnerabilities of a specific test specimen. Without having access to proprietary specifications about the device under test, a penetration test is called black-box test. This is a common and important kind of penetration testing, since it most closely emulates the actions of a real attacker. Particularly for embedded devices, black-box tests are common since most devices use proprietary PCB-designs and there is almost no re-use of designs across multiple PCBs of different products and manufacturers. On the other hand, lacking the specification makes PCBs hard to analyze.

To be able to perform a black-box penetration test of the PCB of an embedded device without further documentation available, the PCB has to be reverse engineered. The PCB reverse engineering task has been investigated for goals like refurbishing or complete reconstruction. However, existing techniques for these goals cannot be applied for penetration tests, since they are too time-consuming, require specialized equipment, such as X-ray units, and often destroy the device under test in the process. Moreover, there is no formalized process to conduct the analysis. Thus, the process is unstructured and, from experience, we know that it is tedious and error-prone, because it involves repetitive manual work. There are no tools supporting the process to get a high-level understanding of each component's role, which is required for the subsequent penetration test.

In this paper we present a workflow and automated method for PCB reverse engineering in preparation of a black-box penetration test. The goal is to make analyzing devices easier, faster, and to improve the coverage of the analysis. Since penetration tests are conducted within a limited amount of time and with access to few testing devices only, the process needs to avoid destruction of devices so they can be used for actual attacks, later on. In essence, this kind of hardware reverse engineering considers the components placed on the PCB and their interconnections. Therefore, the main task our approach addresses is to identify components, as well as finding and analyzing their documentation. Moreover, we show how manual probing for connections between components on the PCB can be augmented by our approach. For component identification, document analysis, and connection probing we present a software architecture and show its feasibility by a proof-of-concept implementation. We evaluated multiple methods for the component identification and found that image segmentation is the best alternative for the task.

While PCB reverse engineering guides further steps during penetration testing, we do not address vulnerabilities of PCBs themselves. Moreover, our method is not fully automated, but still requires human intervention. Manual steps can not be fully avoided due to the fact that we deal with hardware which needs to be handled in the first place. However, we show how to reduce the amount of manual work to the minimum.

The structure of the paper is as follows: First, we discuss existing approaches of PCB reverse engineering in Section 2. We derive a common workflow and requirements of its single tasks in Section 3 to provide a basis for the automation of this workflow in Section 4.1. Addressing this workflow, we present the design of a software architecture in Section 4.2 and its proof-of-concept implementation in Section 4.3. Our evaluation results of methods for component identification are discussed in Section 4.4. Finally, we deduce leverage points for future work in Section 5 before concluding the paper in Section 6.

2 Related Work

Previous work on automated PCB reverse engineering can be categorized into destructive and non-destructive approaches. While destructive methods will destroy the device during the test procedure, a non-destructive process retains full functionality of the device under test after the analysis. Grand published a survey [6] of all different sorts of methods for PCB deconstruction. The goal of these methods is to fully reverse-engineer a device, by extracting images of all PCB layers. A PCB's complete reconstruction, in this case, contains all details of physical connections (traces). It highlights that most deconstruction techniques require destroying the PCB and a significant amount of either time or cost. We seek for a compromise between the detailed but expensive approaches and the cheap but tedious manual approach of performing PCB reverse engineering.

Longbotham et al. [9] describe how a PCB can be reconstructed using X-ray imaging techniques. They regard multi-layered PCBs with connection traces stacked within the board. Therefore, by this method, all layers can be inspected and traces can be reconstructed clearly. This approach generates a very detailed copy of all layers of a PCB, however, it requires removing all the components and access to an X-ray machine. These preconditions are rarely met in a black-box penetration test.

Johnson [8] shows how traces on a PCB can be analyzed using machine vision. He presents a program that reconstructs traces from the image of a raw PCB. A raw PCB is a circuit board that is stripped off components and solder mask. In order to use this approach on arbitrary PCBs, the analyst needs to unsolder the components and scrape off the solder mask. This makes it a tiresome process and destroys the device, thereby invalidating the approach for our scenario. In 2015, Carne presented the tool “PCBRE” at the conference ReCon [2]. This tool allows to reconstruct a whole PCB using images of all layers. These images, again, have to be taken using X-Rays or other means of analyzing the whole PCB. However, both Johnson and Carne demonstrate that an automated analysis of PCB traces is possible.

The book “Hacking the Xbox” by Huang [7] outlines basic steps to classify and identify components of a PCB. From our experience, we can conclude that manually repeating the required steps, such as reading the part identification number and manually searching for documentation online is inefficient for complex systems. While the author explains the most basics steps for reverse engineering an embedded system, the book lacks a workflow description for a professional analysis.

Grand [5] defines a set of attacks on and possible defenses for embedded systems. With our approach, we improve the testability and at the same time provide a more realistic estimation about an attacker's obstacles when reverse engineering a system to clone it or to find vulnerabilities on the physical or the firmware level.

As already indicated, all presented methods work destructively, rendering the device unusable after the analysis. A penetration tester, however, may want to avoid destroying the device because of practical, budget, and time constraints. Previous work highlights the importance of PCB reverse engineering and suggests that machine vision can be applied for analyzing PCBs. Yet, none of these approaches addresses all required tasks, such as identification of components and gathering of technical information. Furthermore, the focus of existing methods lies in analyzing the whole PCB, including all layers of a multi-layered board. Extracting the detailed physical layout allows sophisticated tasks such as cloning devices completely. Nonetheless, it is not required in preparation of a penetration test.

3 Workflow of Manual PCB Reverse Engineering

Because advanced methods of PCB analysis are costly and time-consuming, in practice, analysts perform manual steps to learn how a specific embedded system is built. To the best of our knowledge, here we present the first explicit compilation of the manual workflow to reverse-engineer a PCB. The steps of this workflow are:

1. Identification of components by visual inspection (finding and reading the part number)
2. Gathering of documents related to these components (e. g., datasheets and reference manuals)
3. Extracting information from these documents (e. g., pinout diagrams and pin-signal mappings)
4. Probing important signals, such as buses and debugging interfaces, by using a multimeter (e. g., to find out connectors, test points, etc.)

The results of the workflow are [11]:

- A part list (bill of materials)
- Technical documentation for all components
- A schematic reconstruction (e. g., a graph displaying connections between critical components)

Critical components are components that store, generate, or process data. Passive components, components like voltage regulators, or simple bus drivers are usually not investigated further, as their purpose is intuitive.

The obtained results serve as basis for all following activities: they represent what is known about the system's architecture. As of now, no solution exists that either aids the analyst while performing these steps or helps extracting and saving the information gained from them. This leads to confusion and mistakes, such as overlooked signals and wrong assumptions, based on experience gained solely from other projects or incomplete documentation. Furthermore, information presented in technical documents is only represented for the purpose of engineering or sales. Thus, information that is irrelevant for the security assessment is displayed in the same way that critical facts are displayed, making it harder to quickly analyze a document. Only a fraction of a document's information is needed to understand how the device works.

4 Automated Analysis

An attacker usually does not care about the physical details of traces on the PCB. It is more important to quickly gain a rough understanding of the whole system, meaning, to be able to map specific functions, such as "debugging interface" or "main microcontroller" to components. A PCB's components are soldered on either side of the board, allowing the analyst to probe connections between components using a multimeter on exposed pins or connectors. The only exception are microchips with *ball-grid-array* packages (BGA), with pads underneath the plastic or ceramic package. Still, these microchips can be removed selectively, exposing the pads if needed. An analysis of both sides of the PCB, including the connections, is sufficient to get a sound understanding of the device's inner-workings. Our proposed method aims to automatically gather as much information as possible, without destroying the PCB, so the analyst can quickly decide which parts of the system are most vulnerable.

In this section, we propose a novel, non-destructive and automated approach to quickly analyze a PCB's components and thus, the attack surface of an embedded system in context of a black-box penetration test. First, we derive an automated workflow from the manual steps an analyst has to perform identified in Section 3. We introduce methods for automating each of the steps of the workflow. Finally, we evaluate this concept by implementing and testing an object detection system that identifies components and gathers information about them.

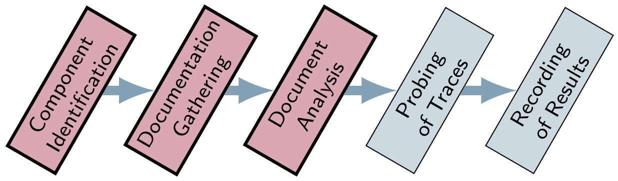


Figure 1: PCB reverse engineering workflow with automated parts highlighted in red.

4.1 Automated Workflow

Our proposed method processes high-resolution images of both sides of the PCB. In particular, it automates the necessary initials steps of identification of components and the gathering and analysis of information related to those components. An overview of these automated steps is illustrated in Figure 1 and explained in the following.

High-resolution cameras are inexpensive and pictures are usually taken for documentation purposes as part of the project, which makes our approach easy and cheap. The images are first segmented by detecting common visual properties of microchips. The segmentation yields areas of the chip packages. These areas are then handed over to an Optical Character Recognition (OCR) library in order to extract the part number that is printed on the package of an integrated circuit (IC). The OCR re-

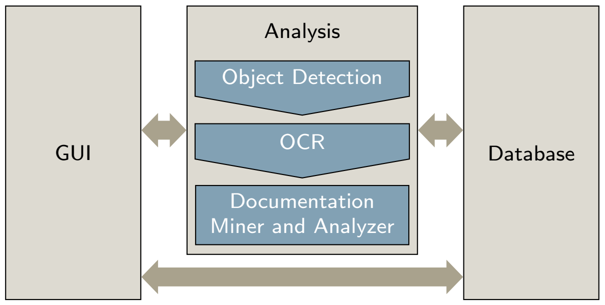


Figure 2: Components of our software architecture.

sults are matched against a database of known part numbers and manufacturer naming patterns in order to reduce false positives. If a valid part-number has been determined, the Internet is searched for related technical documents. The latter is achieved by using conventional search engines, as well as specialized search engines, that only index datasheets. Matching documents are downloaded and basic data is extracted from them, such as the feature description, pinout diagrams and pin-signal tables. Because technical documents usually come in PDF format, it is helpful to convert them into XML in order to easily parse the document's structure and relevant pages. Based on matching keywords and the number of pages, the software filters out unrelated documents such as marketing brochures. All results are persisted in a well-organized file system structure or database and presented in a graphical user interface, so the analyst can access all information quickly.

After every critical component has been identified and technical documents have been analyzed, the analyst probes physical connections between components on the PCB using a multimeter and enters them into the software. For example, a serial connection between two components can be added to the knowledge about the PCB by selecting both components and denoting their connection using the graphical user interface. The connections are stored in the database and can be changed or annotated at any point in time. This is especially useful for generating a visual report of all findings and communicating the attackers point of view to other parties involved, such as customers or engineering teams.

4.2 Software Architecture

We propose a software architecture that supports the kind of automation of the PCB reverse engineering workflow presented in the previous section. As illustrated in Figure 2, this software consists of three parts: a graphical user interface, an analysis backend, and a database.

All components of the graphical user interface are arranged around the PCB picture as shown in Figure 3. By this composition, an analyst can inspect and add information about the system. The navigable image display allows the analyst to spot details on the PCB's surface easily without having to use a magnifier on the physical PCB. It also allows adding miscellaneous remarks onto any location of the picture to highlight and annotate interesting features. By the surrounding control elements, the analyst can easily add components that were not automatically detected and change information, such as the part number of component. Components also support to be labeled with discovered vulnerabilities, enabling to be recalled in other projects based on the part number of components. This user interface layout makes interaction intuitive for beginners and experienced analysts alike since, in practice, analysis results are often communicated by annotating PCB images.

The analysis backend executes a series of modules. The first module identifies bounding boxes of components, whereas the second module performs OCR on those bounding boxes to recognize the part number. For each identified component, the part number is used to query search engines for its documentation. Finally, any retrieved documentation is processed, yielding summaries, pinout, and pin-signal tables.

The resulting information is persisted into a database, so it can be modified and shared in a structured way. Single pieces, such as a PDF-document, but also results like pinout tables, are called *artifacts* and each artifact is stored as an object in the database. Because artifacts have strong relations to each other, they are organized in trees. The top-level element is the component, while sub-elements can be documents and results. This is reflected in the user interface as well, so unnecessary information can be hidden at any time. Because the database is a separate part of the application, multiple programs can query it independently.

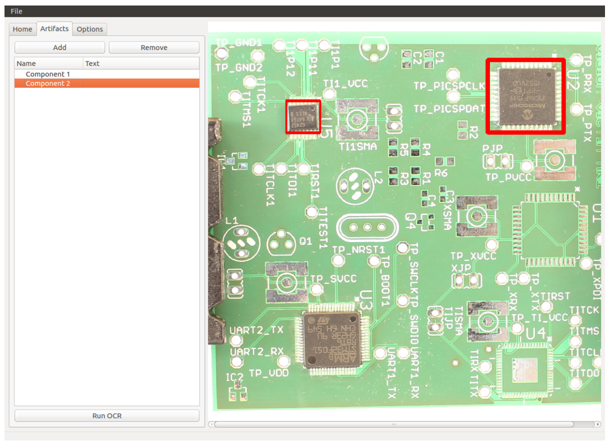


Figure 3: User interface showing control elements beside the PCB image containing identified components

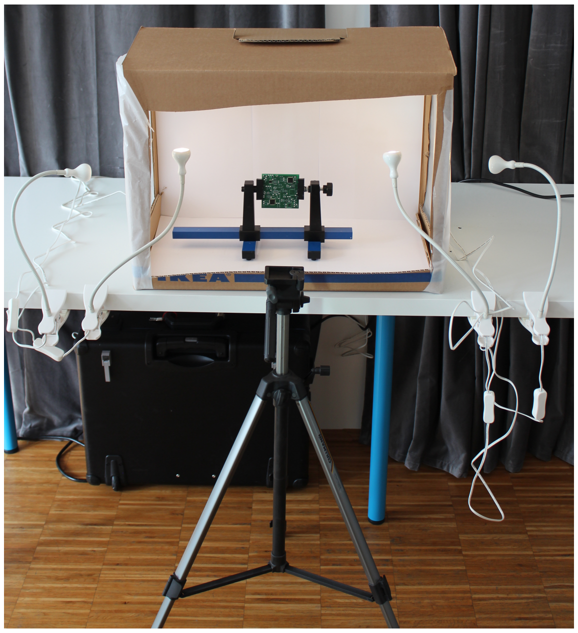


Figure 4: Stage with lamps and a tripod for the camera.

4.3 Proof-of-Concept Implementation

We implemented the software architecture together with a hardware testbed as a proof of concept to test our approach. The software implementation consists of the modules required for automated analysis and the testbed contains a camera setup to take photos of PCBs.

With image recognition that is robust enough, any high-resolution camera can be used for taking pictures of the PCB. Our test setup uses a Canon EOS 600D DSLR with a standard 18 to 55 mm lens. In order to take pictures that are consistent between different PCBs and independent of natural daylight, a small stage, depicted in Figure 4, was set up. The stage is composed of matte, white paper and four adjustable LED lamps, to produce a continuous, matte ambient light. The PCB is mounted on a stand inside the stage.

The software implements a graphical user interface and algorithms for component identification in Python [17] and OpenCV [13]. The graphical interface is using the Qt library [18]. All analysis steps are executed in the background directly after the image has been imported into the software. We assume that the image is cropped or a close-up, so that only the PCB is depicted. However, the background does not interfere with the analysis as long as it is a bright, solid color.

Since we used a state-of-the-art OCR engine as building block for the second step, we will focus our description of the proof-of-concept on object detection and document analysis in particular. We revisit the employed OCR during our evaluation in Section 4.3.3.

4.3.1 Object Detection

Object detection consists of pre-processing, filtering, and the creation of bounding boxes for detected components. The intermediate results of each image processing stage are depicted in Figure 5. Identified regions are marked in purple, while blue rectangles indicate the final component bounding boxes. An image segmentation separates different regions of the image to determine the components' bounding boxes by using pixel information. For example, continuous areas can be used to divide the image into interesting foreground pixels and uninteresting background pixels. Because chip packages create dark, rectangular and continuous areas, those areas can be classified by their size and color using image segmentation.

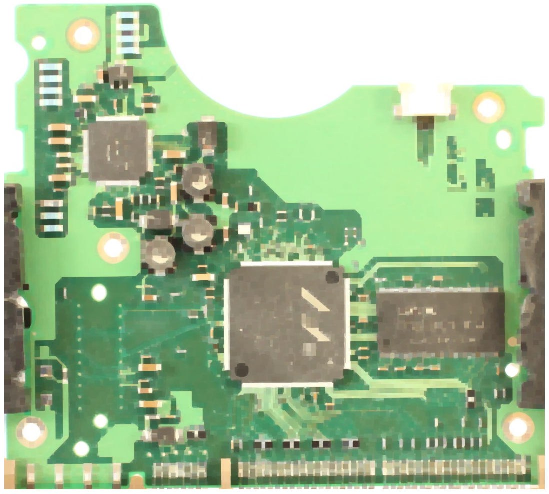
First, the image is *pre-processed* using morphological operations. In particular, erosion is used to flatten and expand continuous areas of color and for removing noise [3]. Morphological operations modify images by considering the relative ordering of pixels within a structuring element, also called *kernel*. A kernel usually is a small matrix, 3×3 in our case, that consists of ones and zeros. The ones in the kernel define a pixel-neighborhood and all pixels in that neighborhood are set to the minimum (in case of erosion) or the maximum (in case of dilation) of the neighboring pixels [19]. In our case, the kernel is positioned and applied on all possible locations of the image.

After that, *thresholding* is applied, in order to remove areas of the image that do not show the visual characteristics of chip packages. Thresholding is performed by computing a function for each pixel. Let $src(x,y)$ be the original pixel value at position (x,y) on a hue-saturation-value (HSV)-encoded image for which we define a thresholding function $dist(x,y)$ by

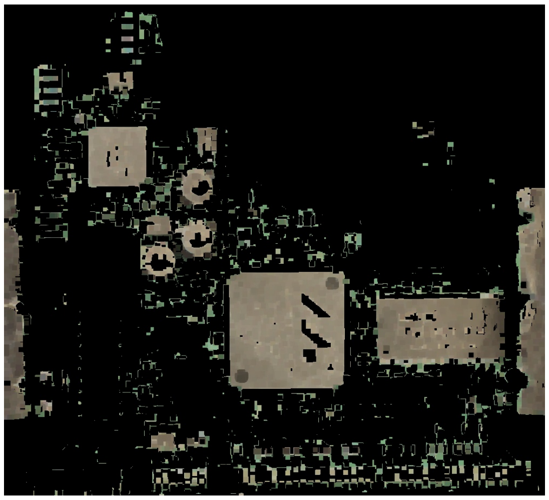
$$dist(x,y) = \begin{cases} 0 & \text{if } H(src(x,y)) > max_h, \\ & \text{or } S(src(x,y)) > max_s, \\ & \text{or } V(src(x,y)) > max_v, \\ src(x,y) & \text{else.} \end{cases}$$

with max_h, max_s, max_v being the threshold values. This function sets all pixels of greater hue (H), saturation (S), or value (V) than max_h, max_s, max_v to zero, effectively removing their information from the image. We use this to retain only the pixels of chip packages, which can be identified by their characteristic range within the HSV-spectrum. While this step is not able to remove every unrelated pixel, it performs well in leaving the outlines of the packages intact and distinguishable.

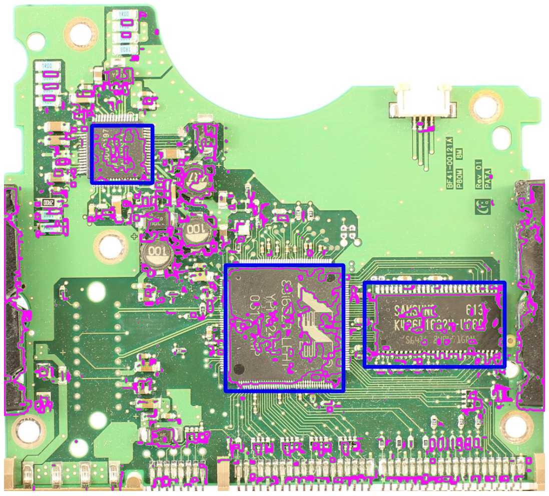
After that, the *watershed algorithm* is used in order to determine the continuous areas of the chip packages. The watershed algorithm for image segmentation simulates rising water that accumulates in minima (valleys)



(a) PCB after pre-processing.



(b) PCB after HSV-filtering.



(c) Final segmentation.

Figure 5: Intermediate results of image processing stages

and is held back by maxima (hills) [20]. A set of bounding points defines the established regions which we then approximate into rectangles. A greedy approximation calculates these rectangles: Let P be one of the sets of bounding points, that the segmentation algorithm returns. Let (p_x, p_y) be the coordinates for each point $p \in P$, then $\forall p \in P$ calculate:

$$x_{min} = min(p_x) \text{ and } y_{min} = min(p_y) \tag{1}$$

$$x_{max} = max(p_x) \text{ and } y_{max} = max(p_y) \tag{2}$$

Using the minimum and maximum coordinates, calculate

$$r_1 = \begin{cases} (x_{min}, y_{min}) & \text{if } x_{min} \cdot y_{max} > x_{max} \cdot y_{min} \\ (x_{max}, y_{max}) & \text{else} \end{cases} \tag{3}$$

and

$$r_2 = \begin{cases} (x_{max}, y_{max}) & \text{if } x_{min} \cdot y_{max} > x_{max} \cdot y_{min} \\ (x_{min}, y_{min}) & \text{else.} \end{cases} \tag{4}$$

Then, (r_1, r_2) is a rectangle and describes a correct bounding box around a component if

$$\begin{aligned} x_{max} - x_{min} &\in [MIN_X, MAX_X] \\ y_{max} - y_{min} &\in [MIN_Y, MAX_Y] \end{aligned} \tag{5}$$

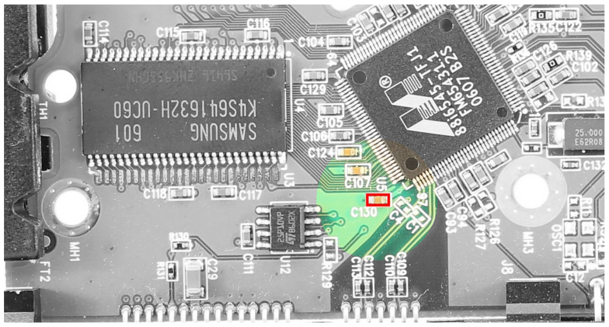
with MIN_X, MAX_x, MIN_y, MAX_y defining minimum and maximum lengths of both sides of the rectangle. This is done to prevent false positives of rectangles being too small or too big to be a component. For our purposes, those values were chosen to be a fraction of the respective PCB's dimension in pixels. This rule calculates the largest possible rectangle around a determined region. Note that this is not the most terse circumscription if components are rotated by factors other than 90 degrees. This can easily be improved by state-of-the-art functions to approximate a rectangle.

The resulting bounding boxes are forwarded to the OCR module to determine the part numbers.

An alternative, semi-automated way of identifying components we implemented is *brute-force template matching*. First, a known component is manually marked on a first PCB or returned by the image segmentation. After that, the marked region is used as a template to detect other instances of the same component on a second PCB. Figure 6 shows the marked template and the resulting detection of multiple instances of the component on a different PCB using brute-force template matching. Brute-force matching compares a pixel window against a template by calculating a correlation function that indicates the similarity of both regions [1]. This technique reduces the number of required user interactions to mark all components on a PCB even if a fully automated object recognition is not successful. Moreover, in order to increase the number of detected components, an automated brute-force matching step after the object segmentation uses the detected components as templates.

4.3.2 Optical Character Recognition

OCR applied within the detected component bounding boxes yields the part numbers of the respective ICs. Part numbers are usually printed on top of the black chip package in a simple, white font. As input, the OCR service gets a pre-processed image of the microchip's bounding box, and in turn, it outputs an array of detected words and their positions. In our case, the service performing the actual character recognition is Microsoft's Cloud Vision API [12]. For pre-processing, the image is inverted so that the black package becomes a white background with black characters printed on it. This is necessary because OCR programs usually expect high contrast between characters and background, such as in scanned documents. Resulting words are concatenated into a single sentence by adding a white-space character between every word. Because manufacturers also print batch numbers and other identifiers on the chip, only a fraction of the words form the part number. In order to extract the part number, all possible sentences are built by calculating the Cartesian product of the words. Then, the sentences are matched against common manufacturer patterns, which have been obtained from electronics distributors' catalogs. If there is a matching pattern, the sentence is immediately marked as a correct part number.


(a) Template (red bounds) marked for cross-PCB detection.

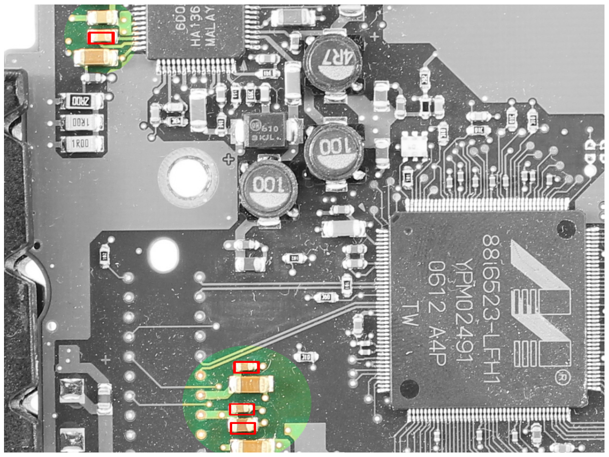

(b) Matched components (red bounds) on another PCB

Figure 6: Brute-force template matching

4.3.3 Document Analysis

The part numbers of ICs determined by OCR are used to obtain the corresponding technical documents for analysis by search engines. To extract relevant component data about these ICs from found PDF-documents, multiple open-source tools are used. First, the document is converted to both plain-text and XML using *pdfminer* [15]. In order to filter out false positives, like marketing brochures, the generated XML allows categorization of documents by their structure because it contains a structured and parsable table of contents, including section headings and references. If, for example, the document does not contain a section called "Feature Description" or the word "Datasheet", it is likely that it is not a datasheet. For this, simple regular expressions or a linear classifier can be used.

Parsing pinout diagrams is challenging because they are embedded as vector graphics or binary images inside the documents. Thus, the software will determine the location of these diagrams and generate only images of the page using the tool *pdftoppm* [16]. This way, the diagrams can still be extracted and saved, so the analyst does not have to search through the document in a dedicated PDF-viewer.

Pin-signal tables map signals to physical pins, for example `UART1_TX` to PIN31. The analyst requires this information to probe traces on the PCBs for locating s or connections between components. In order to extract pin-signal tables from the documents, *Tabula* [21] was used to generate CSV-documents that in turn are parsed by our software to be presented to the analyst.

4.4 Evaluation

Besides the fully-automated color-based *object segmentation* process and the brute-force *template-based matching* used in our proof-of-concept, we implemented and evaluated alternative approaches for detecting the areas of microchips inside a PCB. In particular, the alternatives we discuss in this section are grayscale-based object segmentation and feature-based matching. Moreover, the results of the integration of *OCR* and implementation of *document analysis* for the subsequent workflow-steps are discussed. All approaches have been evaluated using the proposed stage-camera-setup and a total of eight PCBs, taken from real-world assessments and other hardware development projects. At the end of this section, we round up the individual results to provide an assessment of the overall outcome of PCB reverse engineering performed according to our workflow and method.

4.4.1 Object Detection

For **image segmentation** we used a color-based segmentation in our proof-of-concept. We compared this to grayscale-based segmentation approaches, which proved usable for most PCBs. However, using color-information greatly increased robustness on PCBs that have big copper-areas in light silkscreens which generate dark areas. Using a manually determined set of parameters (maximum and minimum area of segments, HSV thresholds) color-based image segmentation correctly detected more than 65 percent of components of the PCBs. These parameters depend on the camera and light conditions of the photography setup and are determined by iterating them in the graphical user interface to minimize false-positives on an arbitrary PCB. After this one-time configuration effort, all parameters can be transferred to future projects without re-evaluation. On average, four false-positives were detected per PCB, with each PCB having an average of twelve components. False-positives were caused by the clamps of the PCB-mount but also by connectors on the PCBs in our test-cases.

For detecting components using patterns from already classified components, our proof-of-concept uses the **template-based matching** approach. While highly accurate, it proved less feasible to use across multiple PCBs than the presented alternatives because details, like variations in package size, dust, or differences in ambient light, already decrease the detection-rate of new components significantly. On the other hand, when used for detecting multiple instances of a component on the same PCB, the template-based approached reduces the amount of manual interaction by over 82 percent on our testing set of PCBs. As a baseline, the amount of manual steps necessary to select all components of one type has been used. For example, a board with 16 identical components requires 16 click-and-drag interactions, without brute-force matching. With brute-force matching, only three interactions are necessary. Small components, like voltage regulators, transistors, or bus-drivers, may occur multiple times, and since manufacturers usually stick to one specific component type for one function, the type often is identical. Furthermore, this approach yields no falsely classified components as long as the template contains the whole area of the IC. Because of this, we decided on implementing the template-based approach using brute-force matching in our proof-of-concept.

An alternative to template-based matching is **feature-based matching**. We evaluated the SIFT algorithm [10] for feature-based matching of components in PCB-images. SIFT uses different levels of blurring to determine so-called keypoints, which are stored in a descriptor. The assumption is that if a visual feature is distinct across multiple levels of blurring, it is a good keypoint. Descriptors can be compared to other images' descriptors, for example by calculating the distance between sets of keypoints and determining the sets with low distance, indicating an occurrence of the template. In order to evaluate SIFT, descriptors of manually selected component bounding boxes were generated and matched to descriptors of other PCB images. This feature-based approach proved infeasible because of the small sizes of components in the images, resulting in few key-points being detected. In contrast, the presented image segmentation approaches proved most robust and accurate for identifying components.

4.4.2 Optical Character Recognition

For extracting the part numbers of detected components different OCR-frameworks have been evaluated. Initially, Tesseract [22] was chosen as an OCR program, but was quickly replaced by Microsoft's Cloud Vision API [12] because the latter proved to be more reliable in detecting the part numbers correctly. 15 out of 18 part numbers were recognized correctly by Cloud Vision. In contrast, Tesseract failed to detect any part numbers. The main reasons for Tesseract's failure were manufacturer logos obstructing the character classification and also the lack of text alignment, that is usually found in a scanned document.

4.4.3 Document Analysis

The document analysis performance was evaluated using a set of eleven datasheets that correspond to components identified in the object detection evaluation. It was checked whether the analysis successfully extracts memory maps, pinout diagrams, and pin-signal tables, if applicable. 10 out of 12 datasheet documents could be found automatically. Out of these, all available memory maps, 8 out of 10 pinout diagrams, and 5 out of 8 pin-signal tables could be extracted. All of the invalid results in either memory maps or pinout diagrams originate from formatting issues in the PDF-document: Section-headings appearing one page before the actual content prevented the correct recognition. Pin-signal tables sometimes were garbled because of line-breaks and missing column delimiters.

Component	Identified	Part-No.	Document	Memory Map	Pinout	Pin-Signal Tbl.
MN4164P-15A*	yes	yes	yes	NA	yes	NA
TMM41256P*	yes	yes	yes	NA	no	NA
HM50464P	yes	no	-	-	-	-
L9959t	yes	yes	yes	NA	yes	yes
K4S641632H	yes	no	-	-	-	-
88i6545	yes	yes	no	-	-	-
L6283	yes	yes	yes	NA	yes	no
PC74HC257P*	yes	yes	NA	-	-	-
PC74HC10P*	yes	yes	NA	-	-	-
PC74HC86P*	yes	yes	NA	-	-	-
T74LS139B1*	yes	yes	no	-	-	-
SI8608	yes	no	-	-	-	-
SN74LS245N	yes	yes	yes	NA	yes	yes
UM82C11*	yes	yes	yes	NA	no	no
PAL16L8ACN*	yes	yes	yes	NA	yes	no
STM32F051	yes	yes	yes	yes	yes	yes
PIC16F1519	yes	yes	yes	yes	yes	yes
G2452	yes	yes	yes	yes	yes	yes

Table 1: Intermediate results of all automated steps, applied on each unique PCB component.

4.4.4 Result Overview

Finally, all analysis modules were combined in order to test overall performance and relevancy. Table 1 depicts the results of this test, for brevity including only components that were correctly identified. Therefore, nine undetected components were excluded. In the table, *NA* denotes that the specific result was not available. Because the memory map, pinout, and pin-signal table analyses depend on the respective document, the dashes represent their absence because of a missing or misclassified document. Components marked with a star (*) have not been manufactured for at least five years and their datasheets are only obtainable in form of scanned documents which can not be parsed in the same way as all other documents. In particular, they only contain images and are lacking a document structure and the plain-text.

Table 1 shows that for modern components, such as the STM32F051, all the analysis results are produced correctly. Except for three instances, the part numbers of all components were correctly detected, which proves that modern OCR algorithms are capable of performing this task. Given the part number, almost all documents have been found, which proves that the approach of using a search engine is valid. Apart from these findings, all other expected results to be found in Table 1 have been achieved correctly with only minor exceptions. Pin-signal table interpretation is subject to further research, because pin-signal tables can be quite complex. Most often, manufacturers include different packages or even product-lines and configurations in their pin-signal tables, which complicates automated parsing.

All image processing modules perform fast on a single image. On an Intel i7-6820HQ CPU, the segmentation takes less than a second on a 12 megapixel image. Brute-Force matching performance scales linearly with the number of templates and with the resolution of the PCB's image. With eight templates, brute-force matching completes within 200 milliseconds on a 12 megapixel image. However, brute-force matching thousands of templates on a high-resolution image would require significantly more processing power. On the other hand, the image segmentation's execution time is nearly constant, regardless of how many components are to be detected. This makes brute-force matching less interesting for mass-scale component detection.

Overall, our results show that the software performs well and aids in the process of reverse engineering by providing an intuitive interface and automated conducting of the targeted initial tasks. Once each component has been identified and available information about them has been gathered the single remaining task of PCB reverse engineering is manual probing of the connections between components.

5 Future Work

The proposed framework allows for further automation directly enhancing our approach. For example, statistical analysis could be done on common components and patterns could be extracted from the arrangement of components. This may guide the decision on which common components or sub-systems further targeted research would improve results. An example for this would be to conduct a broad analysis of third-party PCB images from the Internet to determine common patterns, particularly from reference designs, among manufacturers and product types.

As already indicated, an analysis of pin-signal tables and pinout diagrams during document analysis could yield valuable information. For example, knowing the physical location of a signal, the connections can automatically be mapped to physical pins and communication protocols. Detection of single pins on packages should be trivial using one of the proposed methods for object-detection, assuming that the component's bounding box has already been defined. This applies only to packages where the pins are clearly visible. For example, packages such as Ball Grid Array (BGA) require removal of the component to reveal the pins.

However, actually mapping the pin to a signal is not trivial as the information is embedded in images or vector graphics inside the datasheets. One approach would be, instead of PDF-documents, to analyze commonly available CAD libraries that provide pin mappings which may be easier to parse. These CAD libraries are used and created by engineers when designing a PCB. They contain information about components, such as pin-signal mappings and mechanical specifications.

In conjunction with PCB deconstruction techniques, our method could be used to reproduce a perfect representation of the device under test. This would allow reconstruction and extraction of circuits that, in combination with rapid prototyping techniques, could be used to build test-beds for particular applications. For example, the PCB layout could be extracted and modified so that buses are directly exposed, enabling man-in-the-middle attacks without requiring to reverse-engineer and design the test-PCB by hand.

In order to make interaction with the database even more intuitive, augmented reality applications can be realized using the proposed framework. Component names and annotations could be directly projected into the field-of-view of an analyst. This would make all the information available even in situations where a location-change is needed, for example, when soldering wires onto the PCB.

6 Conclusion

Within this paper to the best of our knowledge, we derived the first explicit workflow description for non-destructive PCB reverse engineering from the current state of the art. The proposed process comprises of component identification, gathering documentation, extracting relevant information from the documentation, and finally physical probing of signals on the board. We derived a software architecture supporting this workflow and simplifying the task of PCB reverse engineering by automation. Our automated approach allows the analyst to perform the initial three out of four steps sequentially on a computer. This clear separation of automated and manual tasks eliminates context switches between looking up information and physical probing. We implemented this software architecture into a framework as a proof-of-concept, showing that the desired automation is possible and performs well. Using image segmentation, we achieved a component detection rate of over 65 percent. We reduced the number of required click-and-drag operations for identifying multiple instances of components, that have not been automatically detected by the software, by 85 percent. Furthermore, we successfully demonstrated how to abstract away irrelevant information from technical documents with high accuracy.

Once the proposed automatic analysis completes, critical components can be identified to guide the subsequent test activities. Thus, critical traces or signals can targetedly be probed in one pass. Furthermore, the method aids the analyst in extracting and storing gathered information in a structured way to prevent confusion and mistakes. The software keeps track of all results in a database and thus, eases sharing results among different analysts, teams or customers. Assuming the more complex methods of PCB deconstruction are infeasible, this approach defines the most structured and cost-efficient way of analyzing a PCB.

PCB reverse engineering is an essential step during information gathering at the beginning of black-box penetration tests and vulnerability testing of hardware components. Our approach for PCB reverse engineering allows for dependable and comparable results that can optimally be utilized as prerequisite of penetration tests. Moreover our approach allows for a new set of possibilities for hardware security testing and engineering. Now it is possible to consider the underlying hardware platform in quicker and more focused test assignments where in the past the effort would not have been taken. It may become possible to continuously annotate a PCB on-the-fly using a smartphone (augmented reality). This can be used to highlight important components, interfaces, or pins. On the other hand, larger PCBs become easier to analyze because the software automates critical steps and allows inspecting the PCB without requiring physical handling or access to the PCB. These new applications and use cases are enabled by our method due to the improved analysis coverage, reduced effort, and structured storage of data.

Acknowledgements

We greatly appreciate the advice and resources for this project provided by Code White GmbH, Ulm.

References

[1] M. J. Atallah. “Faster Image Template Matching in the Sum of the Absolute Value of Differences Measure”. In: *IEEE Transactions on Image Processing* 10.4 (Apr. 2001).

[2] David Carne. *PCBRE*. `https://github.com/davidcarne/pcbre`. 2015.

[3] Edward R Dougherty, Roberto A Lotufo, and The International Society for Optical Engineering SPIE. *Hands-on Morphological Image Processing*. Vol. 71. SPIE Optical Engineering Press Washington, 2003.

[4] Joe Grand. “Can You Really Trust Hardware? Exploring Security Problems in Hardware Devices”. Black Hat Europe. 2005.

[5] Joe Grand. “Practical Secure Hardware Design for Embedded Systems”. In: *Proceedings of the 2004 Embedded Systems Conference*. Vol. 23. CMP Media, 2004.

[6] Joe Grand. “Printed Circuit Board Deconstruction Techniques”. In: *8th USENIX Workshop on Offensive Technologies (WOOT 14)*. USENIX Association, Aug. 2014.

[7] Andrew Huang. *Hacking the Xbox: An Introduction to Reverse Engineering*. No Starch Press, Inc., 2002.

[8] Ben Johnson. *Reverse Engineering of Printed Circuit Boards*. Tech. rep. Stanford - Digital Image Processing, 2014.

[9] H. G. Longbotham, Ping Yan, H. N. Kothari, and Jun Zhou. “Nondestructive Reverse Engineering of Trace Maps in Multilayered PCBs”. In: *AUTOTESTCON ’95. Systems Readiness: Test Technology for the 21st Century. Conference Record*. 1995.

[10] David G Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60.2 (2004).

[11] I. McLoughlin. “Secure Embedded Systems: The Threat of Reverse Engineering”. In: *14th IEEE International Conference on Parallel and Distributed Systems.* Dec. 2008.

[12] *Microsoft Computer Vision API*. `https://www.microsoft.com/cognitive-services/en-us/computer-vision-api`.

[13] *OpenCV Computer Vision*. `http://opencv.org/`.

[14] David Oswald, Daehyun Strobel, Falk Schellenberg, Timo Kasper, and Christof Paar. “When Reverse-Engineering Meets Side-Channel Analysis – Digital Lockpicking in Practice”. In: *20th International Conference on Selected Areas in Cryptography*. Springer, 2013.

[15] *PDFminer*. `https://euske.github.io/pdfminer/`.

[16] *PDFtoPPM*. `https://poppler.freedesktop.org/`.

[17] *Python*. `https://www.python.org/`.

[18] *Qt Development Frameworks*. `http://opencv.org/`.

[19] Jean Serra. *Image Analysis and Mathematical Morphology*. Orlando, FL, USA: Academic Press, Inc., 1983.

[20] Satoshi Suzuki and Keiichi Abe. “Topological Structural Analysis of Digitized Binary Images by Border Following”. In: *Computer Vision, Graphics, and Image Processing* 30.1 (1985).

[21] *Tabula-Java*. `https://github.com/tabulapdf/tabula-java`.

[22] *Tesseract Open Source OCR Engine*. `https://github.com/tesseract-ocr/tesseract`.

All URLs last accessed on May, 27 2017.