SEAKER: A Tool for Fast Digital Forensic Triage

Eric Gentry[1], Ryan McIntyre[1], Michael Soltys[1(✉)], and Frank Lyu[2]

[1] California Statue University Channel Islands, Camarillo, CA 93012, USA
michael.soltys@csuci.edu
https://compsci.csuci.edu/

[2] SoCal High Technology Task Force (SCHTTF), Camarillo, USA

Abstract. Faced with a preponderance of high capacity digital media devices, forensic investigators must be able to review them quickly, and establish which devices merit further attention. This early stage of an investigation is called *triage* and it is a chief part of *evidence assessment*; see [1, Chap. 2]. In this paper we present a digital forensic device, which we named SEAKER (Storage Evaluator and Knowledge Extraction Reader), which enables forensic investigators to perform triage on many digital devices very quickly. Instead of imaging the drives, which takes hours, SEAKER does a search for files with names that conform to pre-established patterns. The search is done by mounting the devices in read-only mode (to preserve evidence) and listing the contents of the device. Unlike imaging, this approach takes minutes rather than hours. Also, SEAKER's hardware consists principally of a Raspberry Pi (RP) and so it is very inexpensive—this is crucial in this era of budgetary constraints; see [2]. Once SEAKER has identified media devices of interest, those can be confiscated for further investigation in a lab. But devices that do not have hits can be left at the scene. This has two principal benefits: forensic examiners can concentrate on those devices that are promising in terms of evidence for the given investigation, and devices without hits are not confiscated from legitimate users.

Keywords: Triage · Digital evidence assessment · Automation · Raspberry Pi · Storage forensics · Digital evidence and the law · Digital evidence preservation

1 Introduction

In this paper we are going to present a digital forensic tool (SEAKER) that was designed and implemented as a result of a collaboration between law enforcement, specifically Southern California High Technology Task Force—Ventura DA, and academia, specifically California State University Channel Islands Department of Computer Science. In the words of [2]:

K. Arai and R. Bhatia (Eds.): FICC 2019, LNNS 70, pp. 1227–1243, 2020.
https://doi.org/10.1007/978-3-030-12385-7_87

> Law Enforcement investigators and forensic analysts have extensive knowledge of what is required to progress an investigation and secure a conviction. Academia is able to provide scientific support, recognized testing procedures and computer science specialists.

The collaboration that we describe in this paper has been especially fruitful, as we designed a device that is able to significantly speed up *Digital Forensic Triage (DFT)*.

We named our device SEAKER, which stands for *Storage Evaluator and Knowledge Extraction Reader*, and its intended application is to quickly scan storage drives in read-only mode, forwarding hits to handhelds of investigators, according to predefined regular expression searches.

SEAKER performs what [3] calls "enhanced preview" for Law Enforcement digital forensic investigations. Also, quoting from [2]:

> Due to budgetary constraints and the high level of training required, digital forensic analysts are in short supply in police forces the world over. This inevitably leads to a prolonged time taken between an investigator sending the digital evidence for analysis and receiving the analytical report back. In an attempt to expedite this procedure, various process models have been created to place the forensic analyst in the field conducting a triage of the digital evidence.

SEAKER is an example of such a process model, designed to help with the triage of digital evidence.

2 Digital Triage

Hitchcock et al. [2] covers the initial phase of an investigation, from issuing a search warrant to acquisition of evidence. As there are more Detectives and Investigators than Computer Forensic Examiners (CFEs), CFEs will always have a long queue of cases. A solution might be to create a tiered system for forensic examiners, consisting of examiners who attend search warrants and use approved tools to conduct triage, and in-lab examiners who will analyze the results of the triage. In this model SEAKER's role will be that of a well understood and trusted tool to be used by the triage personnel. SEAKER would fill this role as it is:

1. *Simple to use*: in the fluid and chaotic environment of a search warrant, the examiners operate a simple triage system.
2. *Tested*: can be operated with confidence at the scene with assurance that evidence thus obtained will stand in court.
3. *Fast*: many devices can be examined, and actionable intelligence can be provided to the investigators at the scene.

SEAKER also acts as a distributed system, as many media devices can be connected to the Raspberry Pi (RP) simultaneously, and all those devices can be examined by many investigators working concurrently. For each connected device, SEAKER creates a unique file containing the complete listing of files on that device. This file can be then searched by several investigators simultaneously, each for a different (if needed) set of patterns.

3 SEAKER

3.1 SEAKER Functionality

It was a conscious design decision to make SEAKER very easy to use. After collecting all the media devices at the scene, the investigator will triage them with SEAKER. Here are the steps:

1. Connect the RP to the power, and after waiting for about a minute to let it start, connect to the RP's hotspot (WiFi network). Depending on the setup, the WiFi's SSID will be "SEAKER01" or "SEAKER02" etc. (if there were to be more than one SEAKER at the scene). See Figure 1. Note that the hotspot is password protected; again, the password may be configured.
2. At the same time the investigator may connect all the media devices to the RP. This may be done concurrently with the previous step.
3. Once connected to the hotspot, the investigator will open any web browser on their handheld, and direct it to go to http://seaker01.local . We decided to allow access through a web browser as this is the most universal way to connect; any device (iPhone, iPad, Android, laptop, etc.) can connect to a hotspot and open a browser. Once the browser establishes the connection, the user will see Fig. 2. Note that the keywords (or regular expression patterns) present in the "Type in Search Terms:" can be pre-loaded before arriving at the scene, or changed/updated at the scene.

   The regular expression can be given using the syntax of the `grep` utility. For example, if we want to find occurrences of either 'two' or 'too', we use `t[wo]o`; if we want to find every word that start with capital letters, we use `[A-Z]`; if we want to find words where number 9 is the last character of the line, we use 9. There are a vast number of possibilities; we can also replace `grep` with `egrep` that has an even richer syntax. Check the `grep` and `egrep` man pages for all the details.

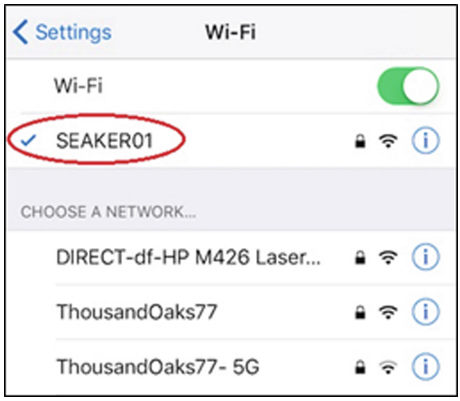

Fig. 1. Investigator's handheld view: connecting to the RP's WiFI (hotspot)

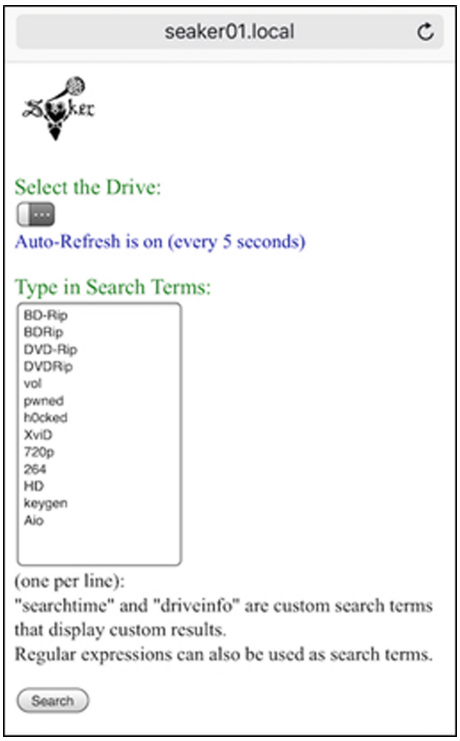

Fig. 2. Investigator's handheld view: using a browser, connect to http://seaker01.local

4. Once any storage devices that are found at a search warrant scene are connected to the RP, the investigator will typically wait for a few minutes (we have seen times up to 10 min for 1 Tb disks with millions of files) for the file list to be built. The search will then be carried out very quickly: essentially, `grep` browses the file list, line by line, outputting those lines that conform to at least one pattern specified in the "Type in Search Terms:" window. Once this finishes, the investigator will have the results presented as in Fig. 3.

The filenames themselves can be incriminating evidence, such as in Child Pornography (CP) cases, where the material has a commonly used naming convention, e.g., "lolita" which can be found with the grep pattern `.*lolita.*` ('`.*`' means the following: '`.`' (period) matches any single character of any value, except a newline, and '`*`' (asterisk) matches zero or more of the preceding character or expression) or simply `lolita`. This can be used by the investigators to question the suspects. The questioning usually takes place at the same time as the forensic examiners triage the evidence, and one of the requirements of SEAKER was to be fast so that investigators can start getting intelligence quickly from the initial processing of the scene. This can be seen in the User Process Flow shown in Figure 4.

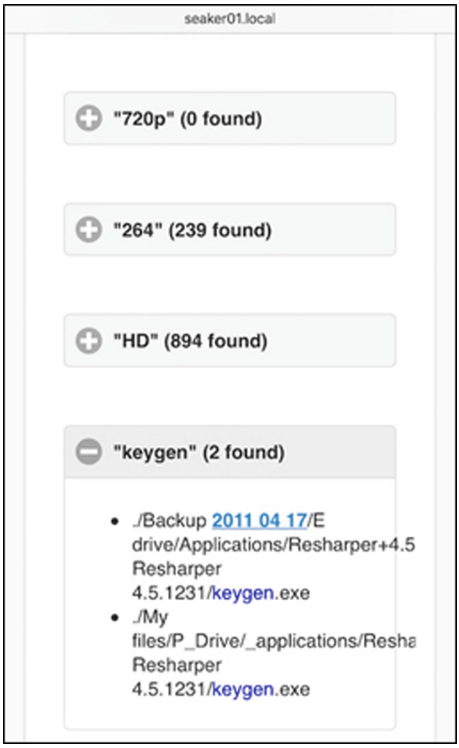

Fig. 3. Investigator's handheld view: the results of the search of a particular device

3.2 SEAKER Design

SEAKER has been custom designed to meet a particular need of forensic examiners; quoting [2]:

> One of the benefits of a custom built tool is that changes are able to be made from a grass roots level.

SEAKER consists of standard, inexpensive and easy to obtain components. The heart of the device is a Raspberry Pi (RP) together with ancillary components such as SATA, USB and power cables. All these components can be purchased for under $150. The RP has to be set up for usage; this is accomplished with a single long bash script that installs all the necessary software components.

The design principles, indeed the requirements of the solution were the following:

1. *Fast*: digital forensic examiners already have at their disposal methods for reviewing hard disks (hds), thumb-drives and other devices in a way that prevents tampering with evidence. The problem is the proliferation of hds

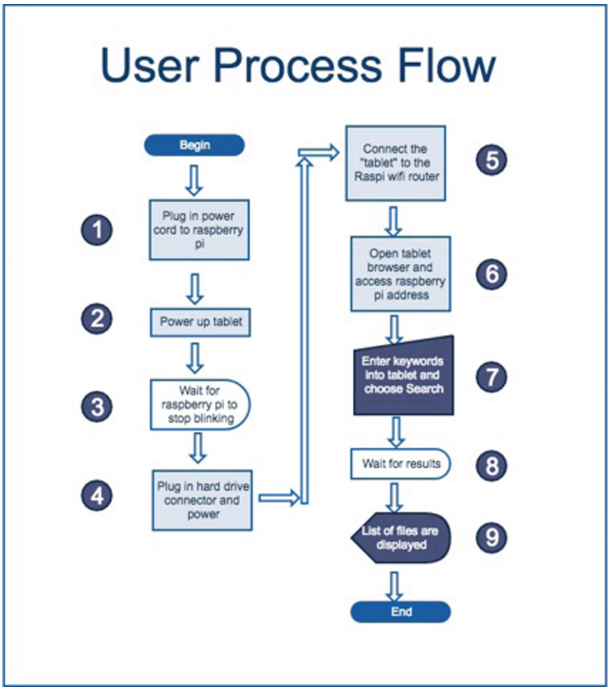

Fig. 4. The functionality of SEAKER from the user perspective

with terabytes of data. Imaging such hds takes hours, and when there are many hds this is infeasible. SEAKER provides a way to triage devices quickly at the scene, and aids in selecting devices that warrant in-depth examination.

2. *Robust*: the device will be used in the field, in chaotic settings, where the examiners cannot be spending time tinkering with the device, and following complicated instructions; it has to be simple and reliable.
3. *Inexpensive*: given the budget cuts facing the public sector all over the world, the aim was to create a working prototype with off-the-shelf components that can be purchased under $200 per unit.
4. *Compact*: the system must be brought to the scene, so portability is key. The small nature of the RP and accompanying plugs and cables makes SEAKER ideal for transport.
5. *Easy to use*: once configured in the lab, the system is ready to work at the scene. In fact, it can be deployed by investigators with little knowledge of IT.

The process flow of SEAKER (Fig. 5) is very simple. After powering the RP, two processes happen simultaneously. Immediately when the drive(s) are connected to the RP, the file names and their paths are copied to a text file. Meanwhile, the user must connect to the RP via WiFi (usually on a handheld device, but any device with a browser works). So in fact, the RP becomes a hot spot. The user must then open the SEAKER web page. As shown in Fig. 5,

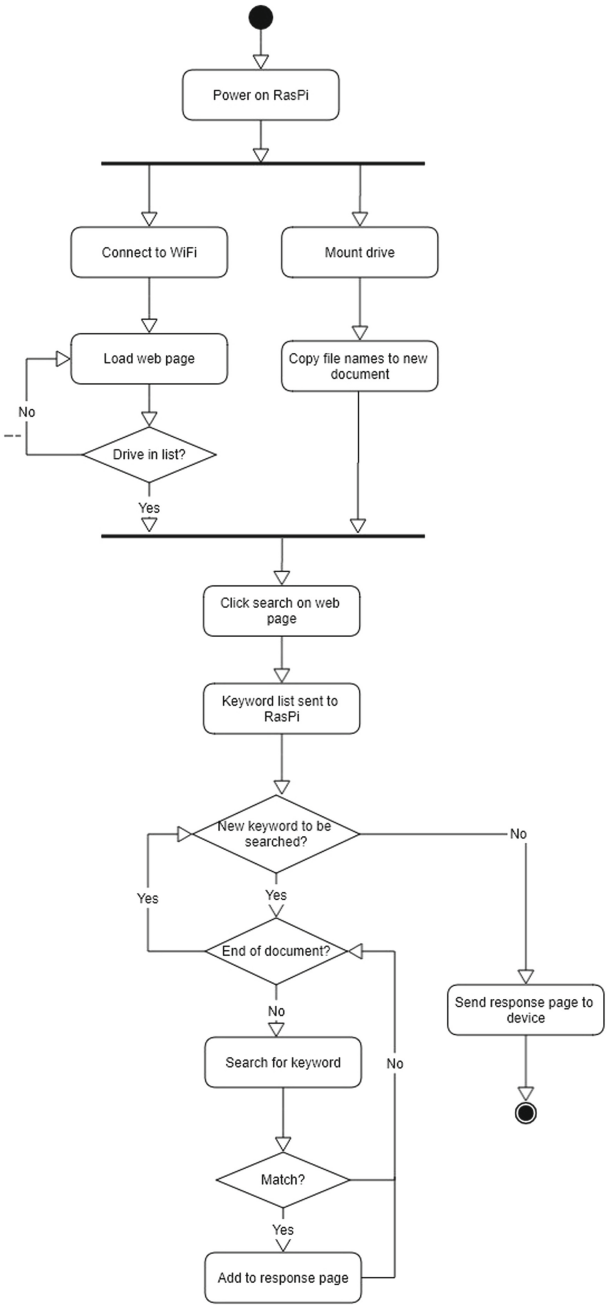

Fig. 5. SEAKER flowchart. Note the left-most "No" arrow indicated with two dashes: the page refreshes every 3 s, and the first drive connected is selected automatically; other drives must be selected manually

the web page will refresh every 5 s, checking each time if new drives have been connected (and their file system recognized) by the RP.

All selected drives will then go through the search process as shown in the lower section of Fig. 5. Each drive will be processed sequentially. For example, the list of files for the drive will be scanned for any matches to the first keyword in the list. The search is done using the UNIX `grep` utility. All files that are found to match that keyword will be added to a text file. Once the whole list of files has been searched for that keyword, the process will begin with the next keyword in the list. This process will continue until there are no more keywords to be searched. If multiple drives have been selected to be searched, this process will continue for each drive. Finally, the text file is used to create the response page and is sent back to the user's mobile device.

While the listing of all the files on the drive can be accomplished with standard UNIX utilities such as `find` or `ls -R`, we opted for a simple `C` program that does so faster as it does not implement all the additional functionality of those standard utilities. Our `C` code is contained in `collect_files.c` and it implements a recursive depth-first search of a given directory tree structure (outputting only file names):

```c
#include <unistd.h>
#include <sys/types.h>
#include <dirent.h>
#include <stdio.h>

void listdir(const char *name) {
  DIR *dir;
  struct dirent *entry;
  if (!(dir = opendir(name))) {
    return; }
  while ((entry = readdir(dir)) != NULL) {
    if (entry->d_type == DT_DIR) {
      char path[1024];
      if (strcmp(entry->d_name, ".") == 0 ||
          strcmp(entry->d_name, "..") == 0) {
        continue; }
      snprintf(path, sizeof(path), "%s/%s", name,
               entry->d_name);
      listdir(path); }
    else {
      printf("%s/%s\n", name, entry->d_name); } }
  closedir(dir); }

int main(void) {
  listdir(".");
  return 0; }
```

4 SEAKER Technical Specifications

4.1 Hardware Requirements

The setup of a SEAKER requires the following hardware:

1. *Raspberry Pi (RP) 3 Model B*: Inexpensive, compact computer hardware that runs a version of Linux. It comes with built-in WIFI, 4 USB ports, and other typical computer connections. This also doubles as the WIFI router for connecting the handheld phone or tablet. External power must be supplied to the RP.
2. *MicroSD card*: This is the hard drive for the RP that will hold the operating system, web server, WIFI router software, and the searched contents of the suspect hard drives.
3. *USB hard drive adapter*: Plugs into the RP's USB port and enables hard drives to be read. External power must be supplied to the USB hard drive adapter.
4. *Handheld phone or tablet with WIFI*: This could be anything from an Apple iPad to a smart phone to a laptop computer. Several devices are able to connect at the same time to the RP via the WIFI hot spot capability.
5. *Ethernet internet connection*: Required for initial setup only. The connection is necessary to download the initial prep script, obtain software updates during the setup phase, and allow for the wireless NIC to be setup for WIFI.

4.2 RP Preparation Script

SEAKER is designed for portability and ease of use, given the chaos and complexity of a crime scene digital forensic triage analysis. The initial setup and configuration is also designed with similar simplicity in mind, but may require a person with IT knowledge.

The prep script is a simple `bash` script written for the September 2017 release of Raspbian Jessie Lite operating system, but is very likely to continue to work with newer versions. The first step for setting up the initial state of the RP is to load the standard "Lite" version of this image onto the microSD card.

The following steps are to customize and finish setting up the RP. Editing the top few lines of the script will enable a customized setup for the particular instance of SEAKER. Here are the editable lines:

```
PI_PASSWORD="raspberry"
WIFI_NAME="SEAKER01"
WIFI_PASSWORD="raspberry"
WIFI_ROUTER_IP="192.168.101.1"
WIFI_ROUTER_DHCP_RANGE="192.168.101.50 192.168.101.100"
```

After editing the script, it must be copied to the RP and run. Using the *pi* account and home path is best for this. The script will automatically reboot when it is finished and the RP is then setup for use as a SEAKER. The script is also designed to clear itself out and clear out the history to ensure that the passwords are not able to be recovered from the script file or the history.

Once the setup is complete and the RP is powered on, a new WIFI access point will be available using the broadcast name that was set in the *WIFI_NAME* setting from the script. Using the handheld phone or tablet to connect to this WIFI access point enables a web-page to be called up using the same web address as the name of the access point. This web-page is the main mechanism for searching for specific information from the connected suspect hard drives.

4.3 Mount Rules

When a drive is connected, SEAKER automatically detects and mounts it, then calls the collect and search procedures. This is done with a `.rules` file in the

```
/lib/udev/rules.d/
```

directory; these files can be used to automatically perform actions when certain events occur (in this case, mounting and listing the contents of storage devices upon connection). In SEAKER's mounting rules, below, each line begins with conditions, characterized by "`==`" and "`!=`." If all conditions are met, then the actions described in the rest of the line is taken. Indented lines are actually continuations.

```
KERNEL!="sd[a-z][0-9]",
  GOTO="auto_mount_usb_storage_by_label_end"

# Import FS infos
IMPORT{program}="/sbin/blkid -o udev -p %N"

# Specify a label
ENV{dir_name}="usbhd-%k"

# Global mount options
ACTION=="add", ENV{mount_options}="ro"

# Filesystem-specific mount options
ACTION=="add", ENV{ID_FS_TYPE}=="ntfs|exfat",
  ENV{mount_options}="$env{mount_options},
   user_id=1000,group_id=1000,ntfs-3g"
ACTION=="add", ENV{ID_FS_TYPE}!="ntfs|exfat",
  ENV{mount_options}="$env{mount_options},
   uid=1000,gid=1000"
ACTION=="add", ENV{ID_FS_TYPE}=="ext3|ext4|ntfs",
  ENV{mount_options}="$env{mount_options},noload"

# Mount
ACTION=="add", ENV{ID_FS_TYPE}!="ntfs|exfat",
  RUN+="/bin/mkdir -p /mnt/%E{dir_name}",
  RUN+="/bin/mount -o $env{mount_options}
  /dev/%k /mnt/%E{dir_name}",
  RUN+="/home/pi/seaker_collect.sh %E{dir_name}
  | at now"
ACTION=="add", ENV{ID_FS_TYPE}=="ntfs|exfat",
  RUN+="/bin/mkdir -p /mnt/%E{dir_name}",
  RUN+="/home/pi/mount_ntfs.sh %k %E{dir_name}"

# Clean up after removal
ACTION=="remove", ENV{dir_name}!="",
  RUN+="/bin/umount -l /mnt/%E{dir_name}",
  RUN+="/bin/rmdir /mnt/%E{dir_name}"

# Exit
LABEL="auto_mount_usb_storage_by_label_end"
```

Note that `NTFS` and `EXFAT` are mounted using different rules. Some file systems have issues (well known among Linux users) with being mounted by custom rules, and require said rules to mount them indirectly through an external script. Details like this provide the primary barrier to filesystem support.

It is important that drive contents are not changed when SEAKER searches them; the drive contents may be evidence in an investigation, after all. SEAKER utilizes two mounting options to avoid tampering: `read only` and `noload`. `read only` is straightforward: SEAKER can only read the contents of the drive—that is, it cannot write or execute. `noload` is applicable to journaling filesystems. A journal serves as a change-log for the drive. If it is accessed, it will change, which will result in a change in the drive's hash. Though such a change would

not affect the stored contents, the change in hash value could cast doubt on the integrity of the evidence. `noload` prevents journaling from occurring, allowing for access without any change to the hash and thereby making it possible to quickly demonstrate that SEAKER leaves the drive contents unchanged.

5 Future Work

SEAKER is currently a prototype; there are many improvements to be made, and we will discuss some of them in this section. These improvements may be implemented by future students, or by digital forensics professionals. We encourage anyone who implements them to share their work.[1]

First, SEAKER supports a select few filesystems. Namely, `NTFS` and `FAT` (`exfat`, `FAT32`, `FAT16`, ...). There are likely bugs to be worked out in the supported filesystems, and there is certainly work to be done in expanding the list of supported systems.

If an unsupported filesystem is in use, it may simply fail to mount, and not show up on the SEAKER site at all. Similarly, drives from which files are currently being collected do not appear; the site displays them only when files have been collected. Here there is opportunity for improvement; as opposed to displaying drives for which collection is complete, SEAKER could display all drives, and a status next to each. This status only requires three states: failed search (for unsupported systems), collection in progress, and collection complete.

It can be difficult to match a hard drive to its corresponding search results. Partitions are uniquely identified by a UUID, and some properties (capacity, for example) are displayed with the search results, but these properties do not provide a perfect way to determine which physical device corresponds to which mounted partition or device. Storage devices generally have a serial number of sorts, but this serial number is not visible to SEAKER. This is a problem which requires some creativity to solve well. SEAKER could take a picture when a drive is plugged in, and associate that picture with the search results, for instance, but this solution requires that investigators position each storage device in front of a camera; this approach requires a camera, and moreover it is tedious and error-prone.

When a search finds a hit (i.e., a matched expression or file extension), investigators may want to view the corresponding file. Currently, this would require them to manually find and open the file. Speed and ease of use are priorities, so it would be best if investigators could select a file in the search results and have SEAKER fetch a copy of it for them. This function inevitably requires that the storage device being queried is still connected—assuming that this condition is met, copying and viewing a file should not be too complex.

Similarly, it would be useful if investigators could view thumbnails of images and videos in the search results. One example of the motivation here is child

[1] The main `bash` script for SEAKER is available on GitHub at https://github.com/michaelsoltys/seaker.

pornography cases; incriminating images may have innocuous names, but thumbnails would indicated the true content.

This leads to another issue: as incriminating files may be named innocuously, investigators will often want to search simply for all images, videos, etc. SEAKER could minimize the work necessary by allowing for preset groups of search terms, which can be created and edited by administrators. For example, an admin could create an "images" group which causes SEAKER to include jpg, pdf, png...

We are very interested in a "Data Carving" option. *Data carving* is the identification and extraction of files from unallocated clusters using file signatures. A file signature, also commonly referred to as a magic number, is a constant numerical or text value used to identify a file format. The object of carving is to identify and extract (carve) the file based on this signature information alone. We are interested in hidden files (which are sometimes easy to locate, as for example in UNIX with '`ls -a`' command) and deleted files, which is more tricky as the files can partially overwritten. A partially overwritten file may still constitute valuable evidence: for example, a portion of an image can be taken as solid evidence that the entire image was on the disk at some point. How can one establish whether a portion of an image comes from a particular image? It seems that the only way to accomplish that is by visual inspection, and having an investigator recognize the original image. In order to automate this process one could attempt one of two things: build a massive database of frequently circulating (say, CP) images, and hashing different formats of these images (.pdf, .jpg, .giff, .tiff, etc.), as well as different resolutions, and chunks of standard sizes (say, 64Kb). This still seems like a shot in the dark. The second approach is to define something akin to *fuzzy hashes*, the type of hashes that are used to recognize variants of the same malware. This new type of fuzzy hashing would be invariant under different formats, or standard resolutions, and chunks of an image could be identified by close proximity to the original hash. Hits would be still confirmed visually to avoid false positives; a bigger issue would be false negatives.

Finally, documentation is important in any investigation. When triage reveals media which motivates investigators to confiscate the corresponding storage device, they should document this motivation. As such, it would aid investigators if SEAKER could generate a search report for a selected drive from the search results screen. This report could be downloaded to the investigator's device or saved on the SEAKER unit for later access by an administrator. It should contain the search results along with some circumstantial information, such as the date, the name(s) of investigator(s) requesting the report, and their reason for confiscating the device.

6 Development Tools

An important component of this project was the learning experience for the students. While we all worked toward a working prototype for the digital forensics lab, for the majority of the students this was the first time working on a

project for which the outcome would be more than the satisfaction of a course requirement. To work in a large team (18 students) different talents, abilities, personalities and work habits had to come together in order to produce a working device. The students used a large set of tools that enabled the development of SEAKER. Here is a list of the principal tools, with short descriptions of their features. It should be mentioned that most these tools are Open Source and free to use.[2]

1. *BASH*: shell scripting was at the heart of the project. While some creative work went into the hardware arrangements (discussed in Sect. 4.1), most of the work consisted in developing a long BASH script. The goal of the script was to set up a RP as SEAKER (the script, called `prep.sh` is discussed in Sect. 4.2). Learning to code in BASH was a big part of the project for many of the students.
2. *Slack*: this is a fantastic collaboration tool that was introduced to the team by the third author of this paper. It allows for an easy and convenient exchange of ideas and brainstorming while developing a product. One of the best features is the ability to divide the conversation into different channels; there were channels for discussion of hardware, software development, testing and documentation.
3. *Dropbox Paper*: this became a de facto Wiki for maintaining the project documentation. It is simple to use; it requires little beyond familiarity with a standard Markdown language. It facilities a distributed documentation development effort. Some participants had "edit rights" while all participants could read and post comments. Eventually, two documents emerged: a set up installation guide, and usage documentation.
4. *GitHub*: a fundamental tool, known to all software developers. About one third of the students requested GitHub collaborative access to help in the development of `prep.sh`. While there are many tools for collaborative software development, it is hard to find something better than GitHub.[3] Anyone can preview the history of the development, and read the annotations.
5. *AWS*: we used an Amazon Web Services (AWS) S3 bucket to have a place with beta versions of the software. The `prep.sh` script, in its most recent version, as well as the most recent version of the documentation, are maintained in the bucket. We learned to mount the bucket onto an AWS EC2 instance, so we can update the staging folder with the latest version with `git`.
6. *C*: while relatively little has been done with the C programming language, a core functionality of the project has been developed in C. (See

[2] Technically, free for the students. Some services required nominal payments; for example, the fourth author has a GitHub subscription which allows for development with private repositories—anyone can open a GitHub account, but a free plan only allows public repositories. Similarly, the fourth author has an AWS subscription; we used Amazon Web Services (AWS) S3 buckets to have a staging repository for ready to use software, our beta versions.

[3] We used GitHub to collaborate on this paper—which allows us to work together while hardly meeting in person.

`collect_files.c` described on page 1234.) A good reminder that when one wants system performance one has to work with C (the `find` and `ls -R` functions were too slow, due to all their features, to list the entire disk).
7. *grep*: this tool is applied in the final stage of the capture, when all the files downloaded by `collect_files.c` are examined for the pre-assigned patterns.
8. *Raspbian Linux*: a free OS, based on Debian Linux, and optimized for running on Raspberry Pi (RP) hardware. We used the "Lite" version, as we just needed the basic functionality, without the 35K+ packages that are present in the full version. It is indeed a revolution in controllers technology to be able to have a hardware controller furnished with a complete Linux OS. For more on the hardware, see Sect. 4.1.

It is important to remember that the principal contribution of this project rests in the performance of the device. Obviously digital forensics had methods to examine data in a sound manner; the augmentation offered by this device is the speed at which triage can be performed in the field. The advantage of our system is so obvious (minutes to list and search all the file names, rather than the hours it takes to image a disk) that we did not need to justify the benefits of our solution. Still, as the device will be used in the field by practitioners, we plan to collect data to keep track of the performance as bigger disks and new filesystems are being added.

7 Conclusion

The SEAKER project was a successful collaboration between two different institutions in the public sector: law enforcement and academia. The former has many interesting problems to offer, but as they are overwhelmed with cases they typically do not have the man power to do research and development. The latter is happy to do R&D, as it enhances the educational experience of the students to be learning in the context of applications to real life problems. It is a fortuitous and symbiotic relationship, and we plan to embark on other such projects in the future.

SEAKER is also the testament to the fact that supremely useful devices, meeting the needs of practitioners, can be constructed from relatively simple components; what is required is expertise and enthusiasm, which in the best cases academia possesses in ample measure. RPs are a revolution in embedded controllers, and we are just scratching the surface of their applicability. They are inexpensive, but wield the power of the Linux OS.

For the students, the experience was invaluable. Perhaps the most important aspect was non-technical: how to work well in a large team. There were eighteen students in the group; a composition of different backgrounds, talents and strengths. We divided the task into five different but interconnected teams: Task #1 was connecting the external devices to the RP; Task #2 was searching the contents of the devices; Task #3 was responsible for sending the query and retrieving the results of the search to the handheld; Task #4 was responsible for

the documentation of the project (both a user set up and guide, as well as the technical documentation of the solution); Task #5 was responsible for testing.

Digital forensics and academia would both benefit greatly from increased collaboration; students can offer relatively inexpensive development in exchange for real-world experience and the opportunity to create something which will be used. As a side effect more students would consider digital forensics as a career, resulting in some level of alleviation of the problems mentioned in the second quote in the introduction [2]. Everyone wins.

Acknowledgements. This work arose from a fruitful collaboration between SoCal HTTF (Southern California High Technology Task Force, Ventura County) and CSUCI (California State University at Channel Islands). We are very grateful for the opportunity to work on such an interesting and eminently applicable problem. We are especially grateful to Senior Investigator Adam Wittkins who facilitated this collaboration. The SEAKER development work was undertaken as a final project for a graduate course in Cybersecurity at CSUCI (COMP524: "Cybersecurity"). The first and third authors were students in this course, and they emerged as leaders of the project, but we are very grateful for the contribution of the rest of the class (in alphabetical order): Geetanjali Agarwal, Nick Avina, Jesus Bamford, Jack Bension, Apurva Gopal Bharaswadkar, Amanda Campbell, Christopher Devlin, Nicholas Dolan-Stern, Manjunath Narendra Hampole, Mei Chun Lo, Christopher Long, Clifton Porter, Deepa Suryawanshi, Mason U'Ren and Zhe Zhang (see http://soltys.cs.csuci.edu/blog/?p=2713).

A Instructions for Setting Up SEAKER

This section contains step by step instructions to build a SEAKER:

1. Prepare the MicroSD card
   (a) Download latest version of Raspbian Lite Image to a local computer (https://goo.gl/eNvdMu)
   (b) Download Etcher software for writing the image to the MicroSD card (https://goo.gl/f6LHBU)
   (c) Download PuTTY if using a Windows based local computer (https://goo.gl/Tvifot)
   (d) Write the image to the MicroSD card (at least 8GB) using Etcher (https://goo.gl/FTvTVx)
   (e) Before removing the MicroSD card from the computer, add a file named '`ssh`' (no quotes, no extension, no contents) to the root of the MicroSD card (https://goo.gl/tTs2vd).
2. Plug in and boot the Raspberry Pi (RP)
   (a) Connect the RP to your network using the Ethernet port (Do not connect using WiFi)
   (b) Plug in power to the RP and wait 10–20 s for the Raspbian Lite operating system to boot.

3. Find the RP's IP address and connect to it
   (a) Find and make a note of the IP Address and substitute it in the rest of setup when RASPBERRYPI_IP is used; this can be done by tools like "Advanced IP Scanner" or by accessing your router administration page
   (b) Use ssh (or PuTTY for Windows) to start a secure shell for example: `ssh -l pi RASPBERRYPI_IP`
   (c) When logging in, the default login is username: 'pi', password: 'raspberry'.
4. Get the prep script and run it
   (a) At the RP prompt, download the prep.sh script: `wget -O prep.sh https://goo.gl/5RU1Yv`
   (b) Modify the first few lines to prevent collisions with other SEAKERs:
   `PI_PASSWORD` (line 18) - Sets the RP's password
   `WIFI_NAME` (line 19) - Sets the WiFi access point name
   `WIFI_PASSWORD` (line 20) - Sets the WiFi WPA2 password
   `WIFI_ROUTER_IP` (line 21) - Sets the WiFi access point IP address (must always end in .1)
   `WIFI_ROUTER_DHCP_RANGE` (line 22) - Sets the DHCP address range (must have the same prefix)
   (c) Set the permissions of `prep.sh` to 744: `chmod 744 /prep.sh`
   (d) Run the prep script: `./prep.sh`
   (e) The script will automatically reboot when finished.
5. Verify that SEAKER is working
   (a) After the reboot, use a separate WiFi enabled handheld phone or tablet (look for a new WIFI access point named using the `WIFI_NAME` setting in the prep.h script)
   (b) Type in the WiFi password (from the `WIFI_PASSWORD` setting)
   (c) Use a web browser from the handheld phone or tablet and type in the WIFI_NAME or new SEAKER IP address after "http://"; for example: `http://SEAKER03.local`.

References

1. Hart, S.V.: Forensic Examination of Digital Evidence: A Guide for Law Enforcement. U.S. Department of Justice (2004)
2. Hitchcock, B., Le-Khac, N., Scanlon, M.: Tiered forensic methodology model for Digital Field Triage by non-digital evidence specialists. Digit. Investig. **16**, S75–S85 (2016)
3. James, J.I.: A survey of digital forensic investigator decision process and measurement of decision based on enhanced preview. Digit. Investig. **10**, 148–157 (2013)