A Framework for Incident Response in Industrial Control Systems

Roman Schlegel, Ana Hristova and Sebastian Obermeier
ABB Switzerland Ltd., Corporate Research, Baden-Dättwil, Switzerland

Keywords: Industrial Control System Security, Forensics, Incident Response.

Abstract: Industrial control systems are used to control and supervise plants and critical infrastructures. They are crucial for operation of many industries and even society at large. However, despite efforts to secure such systems, there are frequent reports of incidents that lead to problems because of human error (e.g., installing unauthorized software on a mission-critical machine) or even cyber attacks. While such incidents should be prevented in the first place, it is not feasible to achieve 100% security; therefore, operators should be prepared to deal with incidents promptly and efficiently if they occur. In this paper, we present a general methodology and framework for investigating incidents in industrial control systems. The methodology is supported by a tool to automate an investigation, especially to efficiently determine the state of files on a device after an incident. This enables faster recovery from incidents by being able to identify suspicious files and focus on the files that have been modified compared to the initially installed files, or a previously taken baseline. An evaluation confirms the applicability of the methodology for an embedded industrial controller and for an industrial control system.

1 INTRODUCTION

Industrial automation and control systems (IACS) are used to monitor and control the behavior of physical processes, for example in chemical plants, electricity generation, and distribution or water management. The first networked IACS were running within isolated networks and did not include any specific cyber security mechanisms. However, nowadays more and more SCADA systems are communicating using public IP networks (Rao Kalapatapu, 2004), which introduces security threats that the systems are not prepared for. As a result, vendors, regulators, and asset owners have started to address this problem by means of security mechanisms, processes, standards, and regulation (Brandle and Naedele, 2008).

In order to increase the overall cyber security level, it is important to detect potential cyber security incidents at an early stage. Traditional digital forensics is aimed at an offline analysis to find court-proof evidence of criminal activities. However, IACS follow a different prioritization regarding the relevance of security objectives, cf. (Dzung et al., 2005; Naedele, 2007). For instance, availability, authenticity, and integrity are paramount for an IACS, while confidentiality is usually less important. There are also additional requirements for live forensics (Ahmed et al., 2012).

1.1 Problem Statement

Cyber security incidents can often go unnoticed for a significant period of time, and when they are discovered it is difficult to evaluate the extent and the severity of an incident. Even though a security tool might generate an event or alarm, the information regarding the event is often not specific enough or is unable to provide any concrete information on the consequences of the event. As an example, an antivirus product might generate an event indicating that a virus has been detected on a machine. It will normally also give the name of the virus (e.g., “Exploit Exp/JAVA.Niabil.gen”), and in which file the virus was found. However, it cannot typically tell the operator whether there are any other consequences, such as which files the virus modified on the system. However, if this happens on a machine with a SCADA application, the operator needs to know whether any other files have been affected by the virus, as this could have an impact on the control system and the process. However, even if a virus has been detected and was removed from the system, there is some uncertainty whether all affected files have been found and removed. As industrial control systems are more deterministic than regular desktop computers and undergo fewer changes in software, this allows for new approaches to detecting incidents or anomalies (Hadeli et al., 2009).

We consider three different scenarios that a forensic investigation tool for IACS should support to aid in incident response. These are:

System Inventory: determine the list of software packages installed on a device.

Baselining: compare a machine to a snapshot of itself in an earlier state, highlighting the changes between the snapshots.

Installation Verification: verify the installation of a software package, identifying genuine files of the software package.

All three scenarios can be used in an incident response to rapidly get an overview of a machine or device and this enables to quickly focus on the root cause of an incident by removing files from the investigation that are verified to be genuine or part of a baseline.

1.2 Contributions

In this paper, we present an approach that leverages the deterministic character of industrial machines by fingerprinting all files on a system and comparing them to a reference set. The main challenge is identifying changes that occur during the normal operation of a system, i.e., minimizing false positives, for example in log files, database files and configuration files. Our approach aims at making the fingerprinting as reliable as possible using different techniques such as regular hash comparison and fuzzy hash comparison, in order to minimize the work remaining to investigate a system after an incident or during routine checks. The contributions of the paper are the following:

- it identifies the methodological differences between analyzing industrial control systems and traditional office IT systems, and develops a specific forensic methodology targeted to industrial control systems;
- it presents a framework architecture comprising an analysis tool that facilitates the forensic analysis of industrial control systems in the different scenarios;
- it evaluates the methodology and the framework on a real embedded controller and on a real industrial control system.

In Section 2, we present related work while in Section 3 we introduce a methodology for a forensic analysis of industrial control system devices by highlighting the differences to traditional forensic analysis of typical machines used in IT systems. Section 4 introduces the framework developed to support a forensic analysis of industrial control systems, followed by the evaluation of the concept in Section 5. We discuss future work in Section 6 and conclude in Section 7.

2 RELATED WORK

Mechanisms and approaches for incident response have been extensively studied, especially over the past decade. As the security of IACS came into the spotlight with the appearance of Stuxnet (Langner, 2011), the interest in detecting and responding to incidents in the IACS domain increased as well. However, the scope of the literature that exists at present is very specific to a certain category of embedded devices and does not address a general incident response and forensics methodology taking into account the operational challenges and considerations for IACS. Of the few papers that address forensics for embedded systems, the scope is typically limited to cell phones, navigation and personal entertainment devices or performing forensic analysis at a network level.

A unified forensics methodology is described in (Shaw and Atkins, 2010), in the context of embedded devices such as data recorders in cars, mobile phones, navigation devices, etc. The authors divide the methodology into three phases, a preparation phase, hardware phase and software phase and they also present a comparative analysis of several related methodologies against the general forensic analysis methodology as described in (US DoJ, 2007). However, they do not cover the particularities of embedded devices in IACS nor give pointers of how they could be analyzed. Furthermore, the use of Snort IDS and use of honeypots for aiding the process of network forensics in a SCADA system is described in (Valli, 2009). An architecture that supports the forensic analysis of SCADA systems and networks is described in (Kilpatrick et al., 2008), where forensic agents are deployed at strategic locations to forward relevant portions of network packets to a central location for storage and analysis.

An interesting approach for forensic analysis using whitelisting is described in (Chawathe, 2009). The paper describes the suitability of signature-based methods in forensic analysis using MD5, SHA1 etc., to classify and prioritize files. The paper also recognizes the need for detecting near matches for effective analysis and describes potential methods to detect near or approximate matches. Moreover, an overview of the use of hashing in digital forensics is presented in (Roussev, 2009), introducing fingerprinting based on random polynomials and similarity hashing based on fuzzy hashes. The authors also describe optimizations such as Bloom filters for hashing large amounts of data and highlight how different forms of hashing (traditional and fuzzy) can be a valuable tool when doing forensic analysis, however, without considering how to obtain data or manage it.

Our methodology and framework share a similar purpose as some of the outlined works above. However, the goal of our framework is neither to address incident response and forensics at a network level, nor to perform optimizations in the hashing algorithms, nor to cover only general purpose computers or a vast range of general purpose embedded devices. Instead we give insight into a unified solution for performing incident response and in-house preliminary forensic analysis at a system or product level for IACS, both, in real time and/or offline. To the best of our knowledge our framework is the first to directly address this challenge for IACS.

3 METHODOLOGY

In this section, we will give an overview of traditional forensics methodology and introduce a general methodology for forensic analysis of IACS.

3.1 Traditional Computer Forensics Methodology

The traditional forensics methodology is typically divided into three phases. These are first the acquisition of evidence, followed by the analysis of the acquired evidence and finally the synthesis of the findings into a report. The acquisition phase consists of turning a device off and copying data from the non-volatile memory (e.g., hard disks) for analysis, as the analysis is never done on the original media. Furthermore, the copying has to be done without modifying the original data, and it should be a verifiable, identical bit-by-bit copy. If volatile data is also needed (e.g., the contents of the RAM), the extraction of that data should be done before powering off a device. The analysis phase depends very much on the context of the investigation and can include listing of all files, recovering log files or deleted files, looking for hidden or encrypted files, eliminating files that are known to be benign, etc. Finally, a report summarizing the findings of the analysis and describing the steps taken to analyze the data is prepared. The same process is applicable to embedded devices with two additional considerations:

1. Most embedded devices do not contain hard disks, but use flash memory instead. In this case a hardware device (a so-called *write blocker*) inserted between the flash card and the host computer can be used to prevent inadvertently modifying the contents of the flash card.
2. Not all information retrievable from a computer system can be mapped to adequate data artifacts in an embedded device. The OS of an embedded device has typically been stripped down and contains less functionality than regular operating systems.

3.2 General Forensics Methodology for IACS

In the IACS domain, powering down a server or an embedded device might impact the control of a process, as continuous operation is typically required. Therefore, the traditional forensic approach should be applied with care, and should only be used when it does not jeopardize the process that is controlled or monitored by the system.

The data that can be extracted from embedded devices also directly depends on the device ecosystem, i.e., the tools that are already available for management, maintenance and configuration of a device. There is no single universal approach that can be used to extract relevant data artifacts from all different types of embedded devices. To be able to extract the most information from different embedded devices, it is necessary to analyze each type and create type-specific guidelines for the extraction of relevant data artifacts that can be used for incident response and forensic analysis.

Evidence Acquisition: There are different approaches for extracting data from an embedded device that can help in an incident response and/or live forensic investigation. For offline data acquisition, imaging tools can be used to obtain a disk image. If the device cannot be shutdown to image disks, different approaches need to be considered, such as a dedicated incident response and forensics agent running on an embedded device, where the agent can be used to acquire data from the device when required at runtime. The agent can be made to retrieve certain files or hashes of all files, it can check files for existence of alternate data streams (ADS), etc. Another approach would be to use the engineering and maintenance tools used by operators and technical experts for configuring a device, debugging and troubleshooting an installation, as well as performing diagnostics. It is also possible to run data collecting servers on embedded devices. For example, an FTPS (secure FTP) server can be installed on an embedded device to facilitate copying files between the device and a

PC connected through the network. If a web server is running on the device, the device can be accessed via HTTPS (secure HTTP) to download files and retrieve event information. If access to the embedded device is allowed over SSH, then this can be another method to acquire data from the device, e.g., through shell commands. In addition, Breeuwsma (Breeuwsma, 2006) describes a method that can be used to extract data from an embedded device using a JTAG port, a port that is normally used for testing integrated circuit boards. Using this method the chance that data is altered through the extraction process is minimized as the memory chip can be addressed and read directly. However, in production systems JTAG ports are typically neither enabled nor necessarily accessible.

Data Artefacts: During the acquisition phase the investigator should decide on the data artefacts that are relevant for further forensics analysis and incident response. However, when retrieving data from an embedded device, the order of volatility should be taken into account as some data has a very short lifespan. More volatile data, i.e., with a shorter lifespan, should be extracted first, to prevent data being lost before the extraction can be completed. Non-volatile data such as log files, configuration files, dump files, temporary files, authentication information, alternate data streams can be read after an incident, as they typically persist when a device is shutdown or reset. Volatile data like operating system time, logged-on users, network status, network information, network connections, process information, process-to-port mappings, process memory, service/daemon/driver information, open files, mapped drivers and shares will be lost immediately when power is interrupted and should therefore be extracted first.

Analysis: The exact analysis done in the analysis phase depends on the context and the goal of the investigation. If a forensic investigation is for example trying to find evidence of an information leak, an investigator would analyze, e.g., local mail data, among other things. In our methodology, where the goal is to verify the system integrity after an incident, the analysis is performed based on a comparison of file hashes with a database of known files. This database contains hash information of known files and other meta-data such as matching product information, version number, manufacturer etc. Depending on the context of the analysis, the comparison can be done against the entire database or over a subset of files belonging to specific products. The analysis is done in two steps:

Regular Hash Comparison. A regular hash of the file such as SHA1, SHA-256, etc. is compared to the list of hashes in the database. If the hash is found, the file has been identified and the information associated in the database is listed. If the hash of the file is not found in the database, a second comparison step is performed using fuzzy hashes.

Fuzzy Hash Comparison. If the comparison of a regular hash does not yield any results for a field, a fuzzy hash comparison can be made. Fuzzy hashes (Kornblum, 2006) match inputs that have homologies, i.e., it can give information about the similarity of a file to another file. If a regular hash (which requires a file to be exactly the same) does not match, the fuzzy hash can reveal files that are at least similar to the file in question.

This two-step approach ensures that as many files as possible can be identified, or if a file cannot be identified, that it can at least be detected to be similar to a known file. Using fuzzy hashes requires the investigator to define a threshold above which a file is classified as similar. For text files (e.g., log files) a lower threshold can be chosen, e.g., 75 (out of 0-100), as there changes are expected, while other file types (e.g., executables) should require more similarity, e.g., 90.

Report: The last step of a forensic investigation is to create a report with the findings.

4 FRAMEWORK

Figure 1 illustrates the general architecture of the incident response framework we developed to analyze incidents in an IACS infrastructure.

The framework is able to accept hashes from various sources, which are fed into the *Industrial Forensics Analysis Tool* (IFAT):

Direct Hashing. The IFAT can hash all files in a given directory, or from a mounted disk image of a device.

Hash Export. The IFAT can be run from a USB stick on a device to hash all files on the device and export the results into a data file. This data file can then be imported and analyzed with a central installation of the IFAT.

Remote/Online Hashing. The framework can also make use of an extended version of GRR Rapid Response (Cohen et al., 2011; Moser and Cohen, 2013), a forensic framework developed by Google, to hash files on a device remotely and online. Through an agent, a device can be instructed to hash certain directories and send back the list of hashes.

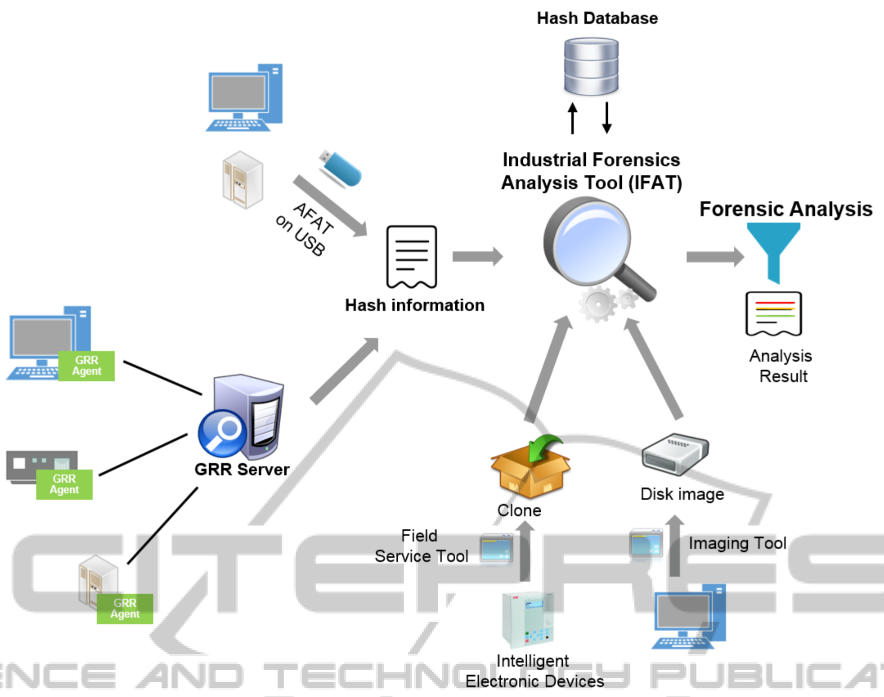

Figure 1: Architecture of the Incident Response Framework.

Once the list of hashes has been acquired, the framework makes use of a comprehensive hash database to identify and determine the provenance of individual files and generates a report. This hash database contains hash information of general all-purpose software, as well as hashes from software specific to the IACS domain and it can also contain baseline versions of complete devices. During the analysis the investigator can then determine whether the comparison should be made against the complete database, only against certain software products or against a specific baseline.

4.1 Industrial Forensics Analysis Tool (IFAT)

Besides regular hash functions like SHA1, MD5, etc. the IFAT also uses fuzzy hashes (Kornblum, 2006), to find similarity between files and to indicate whether a particular file is similar to a known file, or even to an earlier snapshot of the file itself (for example in a baselining scenario). The IFAT can also detect and hash the contents of Alternate Data Streams (ADS) (Marlin, 2013) attached to regular files.

In order to identify a particular file, the IFAT connects to a database that contains hashes of many different software products. The database is a combination of publicly available file hashes, such as the NIST NSRL database (National Institute of Standards and Technology (NIST), 2009), but also contains hashes from proprietary IACS software. For each file, there are the following three possible analysis outcomes:

Exact Match: The file exactly matches a file in the database, indicated by an exact match of their respective hashes.[1]

Partial Match: A file does not exactly match any file in the database, but a fuzzy hash indicates similarity to a file in the database (e.g., a similarity score of 0.9 on a scale from 0 to 1). Whether a partial match has been found also depends on the similarity threshold chosen by the investigator.

No Match: The file did not exactly match any file in the database, nor had a similarity score lower than the threshold chosen by the investigator.

Once an analysis has been completed and all file hashes from a target device have been compared to the hashes in the database, the results can be examined using a virtual file system view.

The IFAT also provides additional functionality that can be used to manage the database of hashes, such as updating the NIST NSRL database (which is itself being updated regularly), adding hashes of new software products or device baselines and generally managing the hashes in the database.

[1]There is a negligible, if non-zero, probability that two different files have the same hash. However, if hashes such as SHA1 or SHA-256 or stronger are used, this probability is so small as to be irrelevant for all practical purposes.

5 EVALUATION

The use of the Industrial Forensics Analysis Tool (IFAT) and the framework has been evaluated in the following scenarios:

- Regular, fresh install of Windows XP (system inventory scenario). Windows XP has been taken as an example operating system as it is still widely used on workstations in IACS.
- Regular, fresh install of Windows XP baselined and then infected with a virus (baselining scenario).
- Embedded industrial automation controller (baselining scenario).
- Industrial control system software package (installation verification scenario).

Windows XP Installation. In this experiment we created a fresh, regular install of Windows XP, hashed all files on the drive and compared it with the hash database. The comparison was able to identify a majority of files (approximately 78%), as shown in Table 1, although fuzzy hashing only improved this by identifying a further 23 files (1.6% of remaining unknown files after SHA1 hashing).

Table 1: File match rate for regular Windows XP installation using NIST NSRL database.

Windows XP	# Files	
Total # of Files	6735	
SHA1	5282	78.4% (of all files)
Fuzzy Hashing	23	1.6% (of files unknown after SHA1 hashing)
SHA1 & Fuzzy	5305	78.8% (of all files)
Unknown Files	1430	21.2% (of all files)

This shows that the publicly available hashes as part of the NIST NSRL database, if used without adding additional hashes to the database, are not sufficient for eliminating a multitude of files. Therefore, in the next scenario, we baselined the new system to achieve better accuracy.

Windows XP Installation with Virus. In this scenario, we used a plain Windows XP installation and baselined it, i.e., we hashed all files and inserted all hashes as a baseline into the IFAT hash database. We then infected the system with a virus that is hidden in a script file for a popular IRC chat client. The virus itself consists of 14 files, including the chat client itself (which is started through autorun in a ”quiet mode” configuration).

Table 2 shows the results of examining the baselined Windows XP machine after the virus infection. Compared to the baseline taken before the virus installation, the tool can identify 98.6% of all files through their SHA1 hashes. Of the remaining 92 files, fuzzy hashing can remove a further 28% (26 files). The remaining 66 files cannot be determined and would yield the files that would have to be checked in more detail. However, of those 66 files, more files could be removed by using additional classification. For example, 17 of these 66 files are Windows prefetch (PRF) files, which could be verified to be genuine (through a semantic analysis, verifying that the files conform to the PRF format), reducing the number of unknown files to 49. The remaining 49 files could be reduced further, as there are for example registry hives among these files, or Windows desktop.ini files that could be verified to be genuine, etc. In this case the virus actually consists of 14 files, meaning that there are 52 files that changed with normal usage of Windows XP, but many of them can be removed through further classification as explained above.

This shows that baselining can very effectively reduce the number of files that need to be examined further after an incident, reducing the total number of files by 99% after the classification with SHA1 and fuzzy hashing in this experiment.

Table 2: File match rate for baselined Windows XP installation infected by a virus.

WinXP (Virus)	# of Files	
Total # of Files	6785	
SHA1	6693	98.6% (of all files)
Fuzzy Hashing	26	28% (of files unknown after SHA1 hashing)
SHA1 & Fuzzy	6719	99.0% (of all files)
Unknown Files	66	1% (of all files)

Baselined Industrial Controller. One scenario of our framework is to compare a controller of an industrial control system with a baseline that was stored earlier. This is similar to the use-case of baselining regular machines, but for the industrial controllers, this can be done regularly (e.g., once every week) and automatically.

We performed some limited tests using cloned drive contents of actual controllers. We manually added two new, unrelated files and changed parts of an existing file, and could verify that these changes to the controller were detected correctly and accurately by the IFAT.

Industrial Automation and Control System Software. To test the scenario of efficiently verifying the installation of a software package, we also tested the

IFAT on a large Industrial Automation and Control System (IACS) software product. We first imported the hashes of the files on the installation media of the software package into the database (extracting installer data as far as feasible). We then used the IFAT to hash a real, existing installation of the software, and compared the installation directory of the software on the machine with the data contained in the database that was derived from the installation media.

The results (see Table 3) show that by adding the installation media of a software package, the tool is able to classify more than 90% of the files of the installation as belonging to that product. In this case, adding the installation media to the database was done manually, but if the process can be automated, including recursive unpackaging of installer data, then the match rate could be improved even further.

Table 3: File match rate for an Industrial Automation and Control System software installed on a machine.

IACS Software	# of Files	
Total # of Files	8995	
SHA1	8045	89.4% (of all files)
Fuzzy Hashing	99	10.4% (of files unknown after SHA1 hashing)
SHA1 & Fuzzy	8144	90.5% (of all files)
Unknown Files	851	9.5% (of all files)

Other Performance Measures. We tested the IFAT together with a database containing information about approximately 110 million files. This includes both the contents of the NIST NSRL database and custom hashes of software that we inserted. Also, the NIST database only contains SHA1 hashes, while for the files that we added we also included fuzzy hashes. The database size was approximately 24GB (including both data and indexes), and the (virtualized) server had 5.5GB of RAM available and was assigned 3 Intel Xeon CPUs at 1.86GHz. With these specifications the server was able to perform about 360 queries per second for SHA1 hashes (using only one CPU). One direction for improving the performance would be by adding more RAM to the server, as in our tests the performance bottleneck was the hard disk IO. Performing fuzzy hash queries was significantly slower by two orders of magnitudes, because comparing fuzzy hashes requires computing an edit distance between a queried fuzzy hash and all fuzzy hashes stored in the database, and this operation could not be improved through the use of indexes.

Conclusion. The evaluation of the IFAT prototype has shown that it can handle the scenarios that we outlined earlier, such as system inventory of a machine to baselining and verifying installations of specific software packages.

Limitation. One limitation is that there exists kernel-level malware that can falsify the analysis of files by returning "clean" data or by hiding malicious files when listing directories. This is a problem for every live analysis tool (e.g., using the IFAT directly on a machine, or hashing files through the GRR agent), but can be solved for example through offline analysis (i.e., analyzing an image extracted from the data storage, e.g., hard disk, flash drive, etc.). Another solution to this problem would be to run the machine in question in a virtualized environment, and have the hypervisor directly access the underlying data storage, without going through the kernel of the virtualized machine itself. Another limitation is that even with a recognition rate of 90%, the number of files still unknown after an analysis can still number in the thousands, which would then still need to be further analyzed. This further illustrates that the quality of the hash database is crucial.

6 FUTURE WORK

Currently, the hash database used in our approach contains hashes of legitimate IACS software and general purpose software. One direction for future work would be to study the effectiveness of a heuristic analysis of unmatched files for more accurate results. This could potentially be of help when dealing with a large number of unmatched files and could be used for preliminary prioritization. Another direction is to extend the Incident Response Framework to be able to perform online analysis of other operating systems, particularly those used in the IACS domain such as real-time operating systems. Finally, with the increase in discovered vulnerabilities and the growth of digital threats in the IACS domain, incident response will become even more important in the future, requiring ever more elaborate methods to produce precise results.

7 CONCLUSION

We have presented a comprehensive methodology for forensic analysis of IACS that outlines the steps necessary to be performed and gives an overview of possible ways for extracting file system data from embedded devices. However, our analysis has shown that because of the diverse nature of such devices, each device type would need to be studied individually and the best methods available for data extraction determined for each device. This is because embedded devices differ in their capabilities, their architecture, the supported operating systems, available interfaces and organization of the file system, which requires different extraction methods for each particular type of device.

Furthermore, we have presented a framework that allows for an initial analysis of the non-volatile storage of IACS systems and devices as well as of general-purpose computers. This analysis is foreseen to be done in response to an incident, as an in-house preliminary forensic analysis or as part of a periodic routine analysis. The framework supports a variety of operating systems and has been shown to be suitable for examining entire file systems, specific directories or single files. Altogether, the framework covers well the use-cases outlined in the introduction of this paper.

In addition, we have also performed an evaluation, demonstrating the performance of the framework in different scenarios. The recognition rate of matched files, as expected, is directly correlated with the comprehensiveness and completeness of the hash database. A more complete database that includes hashes of as many software products possible will result in more accurate results. However, for readily available databases such as the NIST NSRL database, there are potentially still a large amount of "unknown" files that need to be further investigated after running our analysis tool. The evaluation also showed that a fuzzy hash comparison can improve the recognition rate, although not substantially for every scenario. The performance of the hash comparison also directly depends on the performance of the server where the database is stored and the resources allocated to the database, and we have shown that reasonable performance can be achieved using moderately powerful hardware.

REFERENCES

Ahmed, I., Obermeier, S., Naedele, M., and Richard, G. G. (2012). Scada systems: Challenges for forensic investigators. *Computer*, 45(12):44–51.

Brandle, M. and Naedele, M. (2008). Security for process control systems: An overview. *IEEE Security & Privacy*, 6(6):24–29.

Breeuwsma, I. M. (2006). Forensic imaging of embedded systems using jtag (boundary-scan). *Digital Investigation*, 3(1):32 – 42.

Chawathe, S. (2009). Effective whitelisting for filesystem forensics. In *Intelligence and Security Informatics, 2009. ISI '09. IEEE International Conference on*, pages 131–136.

Cohen, M., Bilby, D., and Caronni, G. (2011). Distributed forensics and incident response in the enterprise. *Digital Investigation*, 8, Supplement(0):101 – 110. The Proceedings of the 11th Annual Digital Forensic Research Workshop (DRFWS '11).

Dzung, D., Naedele, M., von Hoff, T., and Crevatin, M. (2005). Security for industrial communication systems. *Proceedings of the IEEE*, 93(6):1152–1177.

Hadeli, H., Schierholz, R., Braendle, M., and Tuduce, C. (2009). Leveraging determinism in industrial control systems for advanced anomaly detection and reliable security configuration. In *Proceedings of the 14th IEEE International Conference on Emerging Technologies & Factory Automation*, ETFA'09, pages 1189–1196, Piscataway, NJ, USA. IEEE Press.

Kilpatrick, T., Gonzalez, J., Chandia, R., Papa, M., and Shenoi, S. (2008). Forensic analysis of scada systems and networks. *Int. J. Secur. Netw.*, 3(2):95–102.

Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3, Supplement(0):91 – 97. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

Langner, R. (2011). Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51.

Marlin, J. (2013). Alternate Data Streams in NTFS. Online: http://blogs.technet.com/b/askcore/archive/2013/03/24/alternate-data-streams-in-ntfs.aspx.

Moser, A. and Cohen, M. I. (2013). Hunting in the enterprise: Forensic triage and incident response. *Digital Investigation*, 10(2):89 – 98. Triage in Digital Forensics.

Naedele, M. (2007). Addressing IT security for critical control systems. In *HICSS*, page 115.

National Institute of Standards and Technology (NIST) (2009). National Software Reference Library.

Rao Kalapatapu (2004). SCADA Protocols and Communication Trends. ISA EXPO.

Roussev, V. (2009). Hashing and data fingerprinting in digital forensics. *Security Privacy, IEEE*, 7(2):49–55.

Shaw, R. and Atkins, A. (2010). Unified forensic methodology for the analysis of embedded systems. *Proceedings of 4th International Conference on Advanced Computing & Communication Technologies*.

US DoJ (2007). Digital Forensic Analysis Methodology. Online:http://www.justice.gov/criminal/ cybercrime/docs/forensics_chart.pdf. Cybercrime Lab in the Computer Crime and Intellectual Section.

Valli, C. (2009). SCADA Forensics with Snort IDS. In *Proceedings of WORLDCOMP, Security and Management*, pages 618–621, Las Vegas.