# System for integration of tools for full content verification on multiple sensors

Tommy Steensnæs


Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2010

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

# System for integration of tools for full content verification on multiple sensors

Tommy Steensnæs

30th June 2010

# Abstract

Cyberspace is the newest addition to the domains for warfare. As in the other domains both offensive and defensive operations are relevant. Defensive operations in cyberspace can draw on experience from traditional information security, but might use other procedures and use tools in a different way. This thesis shows how existing command line tools for packet captures can be used more effectively. The research provides a definition of a scalable effective system for packet capture for use in Computer Network Defense (CND) verification and analysis. A prototype of the system is presented and evaluated in terms of efficiency and effectiveness.

# Sammendrag (Abstract in Norwegian)

Cyberspace er det nyeste domenet for krigføring. Som i de andre domenene er det behov for både offensive og defensive operasjoner. I defensive operasjoner i cyberpace kan en bruke erfaringer fra tradisjonell informasjonssikkerhet, men det kan være behov for å forandre prosesser og prosedyrer, ogbruker verktøyene på en ny måte. Denne masteroppgaven viser hvordan eksisterende kommandolinjebaserte verktøy kan brukes mer effektivt. Forskningen viser et effektivt skalerbart system til bruk i verifikasjon og analyse knyttet til Computer Network Defense (forsvar av datanettverk). En prototype utvikles og evalueres med tanke på effektivitet.

# Contents

# List of Figures

# List of Tables

# Preface

The author of this thesis is employed by the Norwegian Defense Security Service(NODSS) and works as an analyst at the Critical Infrastructure Protection Center(CIPC) at Jørstadmoen outside Lillehammer. The main task of this center is to conduct Computer Network Defense(CND) for the Norwegian Defense. This is a relatively new military discipline and over the last few years we have experimented with several different tools to help effectively defend our computer networks. The amount of traffic and complexity of the networks is very high and being able to use the proper tools is imperative to solve this task. The work done in this thesis is based on the experiences from the last few years and is one small step on the way to create an effective toolbox specially adapted to our operators, environment and spesific tasks.

## Acknowledgements

This thesis would not have come together without help for which I am very grateful. First of a I would like to thank my supervisor Prof. Slobodan Petrović for guiding me through all phases of this thesis and keeping me on track. I have had periods of doubt. I would also like to thank my good friend and fellow student Kjell Tore Fossbakk for always being open for discussions and patiently supporting me in my programming. I am ever grateful to my boss and all my coworkers at the Critical Infrastructure Protection Center for their contributions to my research and for giving me the time and opportunity to write this thesis. Lastly, I want to send a big thanks to my girlfriend Guro, whom have made my life bearable the period of writing this thesis. Her efforts in keeping our home livable and taking care of just about everything in my absence have been invaluable.

# 1 Introduction

## 1.1 Topic covered

The cyber domain has over the last years become increasingly relevant for warfare. This domain has proven especially fitted for asynchronous warfare and has on several occasions been used in political conflicts. Therefore most modern countries have started to develop capacities in this area of warfare. As in all areas of warfare there is a need to be able to run both offensive and defensive operations. As in other countries around the world this is one of the military disciplines described in the Norwegian defense doctrine [1]. Although part of the doctrine there are still several challenges to be met both in processes, procedures and tools on both the tactical and the strategic level. This thesis has focused mainly on tools and presents a system for packet capture for use in Computer Network Defense(CND).

## 1.2 Keywords

Information security, network security monitoring, intrusion detection, information warfare, Computer Network Defense

## 1.3 Problem description

There are no commercial tools especially designed for CND. Therefore different combinations of commercial software, open-source software and in-house developed software are being used. This constellation derives problems with scalability and manageability. One of the common challenges is that of data collection when verifying an indicator.

### 1.3.1 Example

To describe the problem and show the need for this thesis an example is in place. The presented example shows a typical event and the work flow towards verification and handling of a possible incident. First one of the deployed sensors shows an indication of malicious activity on the network. Let the activity for this example be a connection to a website which is a known distributor of malware. In this case the hosted malware is an IRC bot. This will obviously trigger a need to verify if the computer connecting to this website got infected with the malware. Because we know the bot we will also know where this bot connects to get updates or commands. Therefore to verify we can capture traffic from the computers we suspect are infected towards the botnet controller. If we get traffic we can easily determine if the computer is infected and advise on how to handle it. This example is trivial, but illustrates the principle.

Let us say we needed to find out if more computers were infected with the same bot. We could set up the capture to look at all traffic towards the controller. This would catch other possibly infected computers and not only the computer we have suspicions towards. As long as all traffic passes by the same sensor that should be just as easy. If there are several access-ways to the Internet and there are several deployed sensors the analysts will have to do the same process repeated for all sensors. When this scenario takes place several times a day a lot of time is wasted administrating packet captures.

The results of this thesis presents a solution to how analysts with simple means can make this process more effective. This thesis is focused towards packet capture, but the methodology will easily relate to other command-line tools as well.

The above example shows how verification by capturing traffic can be relevant and highlights the problem when the operations need to be repeated for several sensors. This process gives a linear relationship between the operations needed to start a capture and the number of sensors. Further on, the analyst will need to access the captured data separately on each of the sensors additionally lengthening the process. This thesis shows how the operations needed from the detection of an indicator to the point where useful data can be presented to the analyst can be substantially decreased in a multi-sensor environment.

The situation today can be described as shown in Equation 1.1

$$O = o * s \tag{1.1}$$

where:

**O** is total number of operations

**o** is number of operations per sensor (needed to start packet capture)

**s** is total number of sensors

The goal of this project is to provide a system that can be described by the following equation (1.2)

$$O = o_1 * s + K_o \tag{1.2}$$

where:

$o_1$ is number of operations exclusive to each sensor

$K_o$ is number of operations needed to start capture (constant)

and

$$o >> o_1$$

## 1.4 Computer Network Defense

CND is not a concept that is uniformly defined among all users of the term. Other terms such as Information Security(InfoSec), Information Assurance(IA) Network Security Monitoring (NSM)[1] usually contain some of the same elements. Some literature even describes one term as a subset of the other. Even though the range of definitions is wide, most of them include some variation of the act of securing information and computer networks. The US Department of Defense defines CND as follows:

> "[CND]Describes the actions taken, within the Department of Defense (DoD), to protect, monitor, analyze, detect, and respond to unauthorized activity within DoD information systems and computer networks. CND protection activity employs information assurance principals and includes deliberate actions taken to modify an assurance configuration or condition in response to a CND alert or threat information."

A similar definition can be found in the "*Dictionary of Military and Associated Terms. US Department of Defence 2005.*":

[1] NSM is described thoroughly in [2]

> "Defensive measures to protect and defend information, computers, and networks from disruption, denial, degradation, or destruction. Also called CND."

[3] goes further in describing Computer Network Operations.

In Norway the concept of Computer Network Defense is pretty new and not well described in official documents, but the Norwegian Defense Operational Doctrine [1] (in Norwegian) describes CND as:

> "Measures to actively protect information by monitoring, analyzing and realizing countermeasures to attacks on ones own information systems"

As we see from the definitions presented above the scope of CND is very wide. In this thesis the definition from [1] as shown above will be used. CND as a whole is a complex task and includes several different subprocesses. To be effective the CND professionals need good tools to support the different stages of CND. The system described in this thesis applies to the borderline between detection and analysis where the operator detects some anomaly and is in need of more data for verification and analysis. In Figure 1 a rough outline of the steps in the CND process is shown. It all starts with the Commanders Critical Information Requirements(CCIR). These are the categories of incidents that are of interest. Based on those requirements one starts collection planning. This phase includes putting sensors in strategic spots in such a way that indicators of incidents of interest can be collected. When all is in place the continuous phase of monitoring and collecting indicators starts. The indicators of interest are analyzed. If needed additional data collection are initiated. If an actual security incident is verified the collected indicators are escalated to incidents and operations planning and incident handling is started. As stated earlier, the system presented in this thesis is used for collection of additional data for verification and analysis. Based on the example from Chapter 1 (Section 1.3.1) Figure 2 shows this part of the process in more detail.

## 1.5 Justification, motivation and benefits

As mentioned, the cyber domain is becoming increasingly important. As a consequence several nation states have started developing capacities in order to conduct warfare in this new domain [4]. The arise of such capacities and the will to actively use them show the necessity of being able to defend the national(NII) and defense information infrastructure(DII).

The information systems in the DII are changing at a very slow pace. Both architecture and even software can at any time be considered out of date. Complicating this further is the fact that the people defending the networks do not necessarily have access to the schematics and the inner workings of the network. There are many reasons for this and this situation is probably not going to change considerably for a long time. Therefore the security analysts can never hope for optimal conditions. To prepare for defensive operations in cyberspace we do what would be done in the physical world. We analyze the battlefield and make the best of the terrain at our disposal. This makes it absolutely essential for the analysts or the "cyber warriors" to have flexible, scalable tools that can be easily implemented in any network. Big, heavy enterprise Security Incident Managers(SIMs) and the like do not work as advertised without being designed in to the information infrastructure and are therefore often not the right tools. Anyway, SIMs will give added benefit if already present in the network to defend.

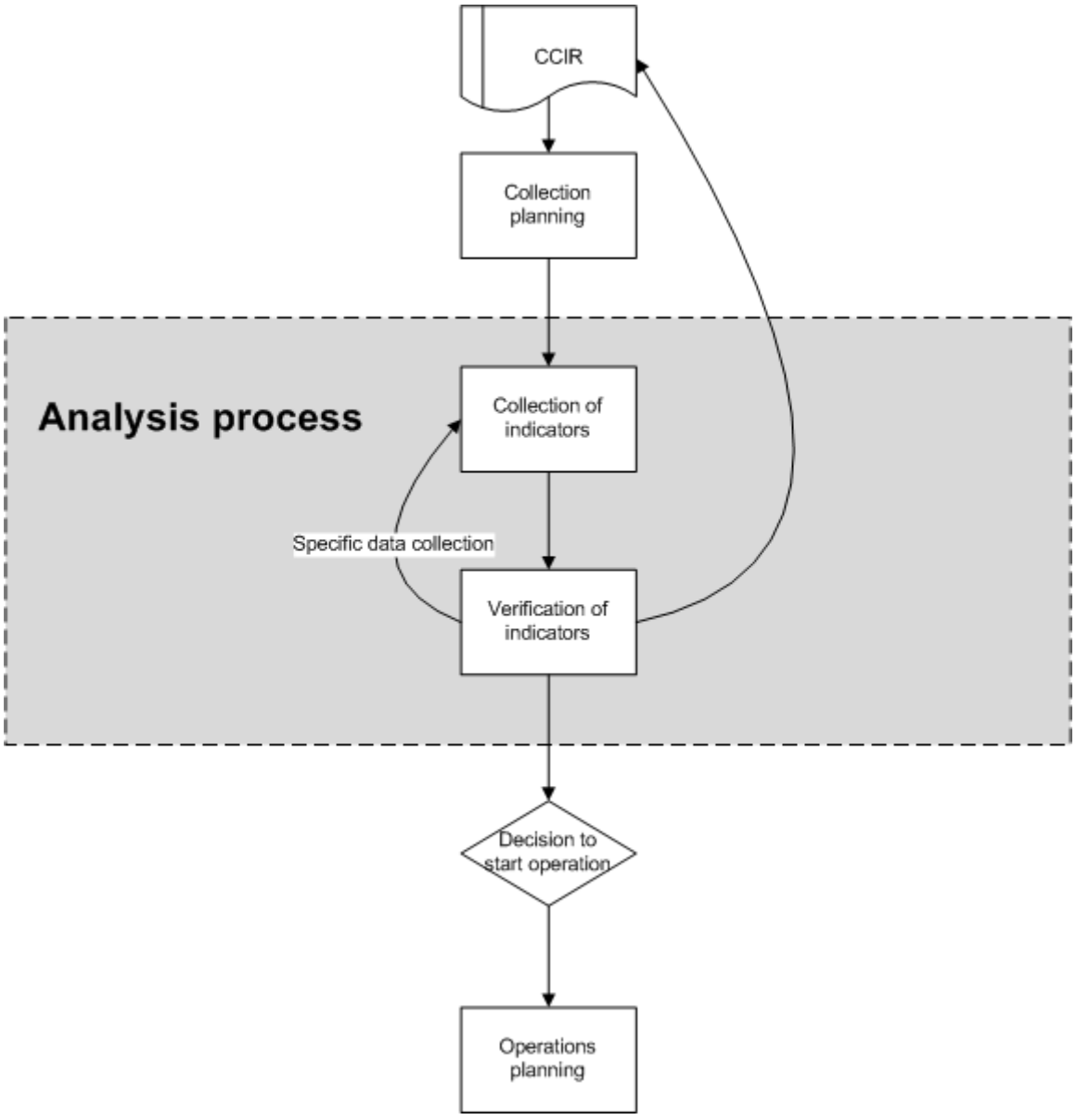


Figure 1: Schematic of the CND process

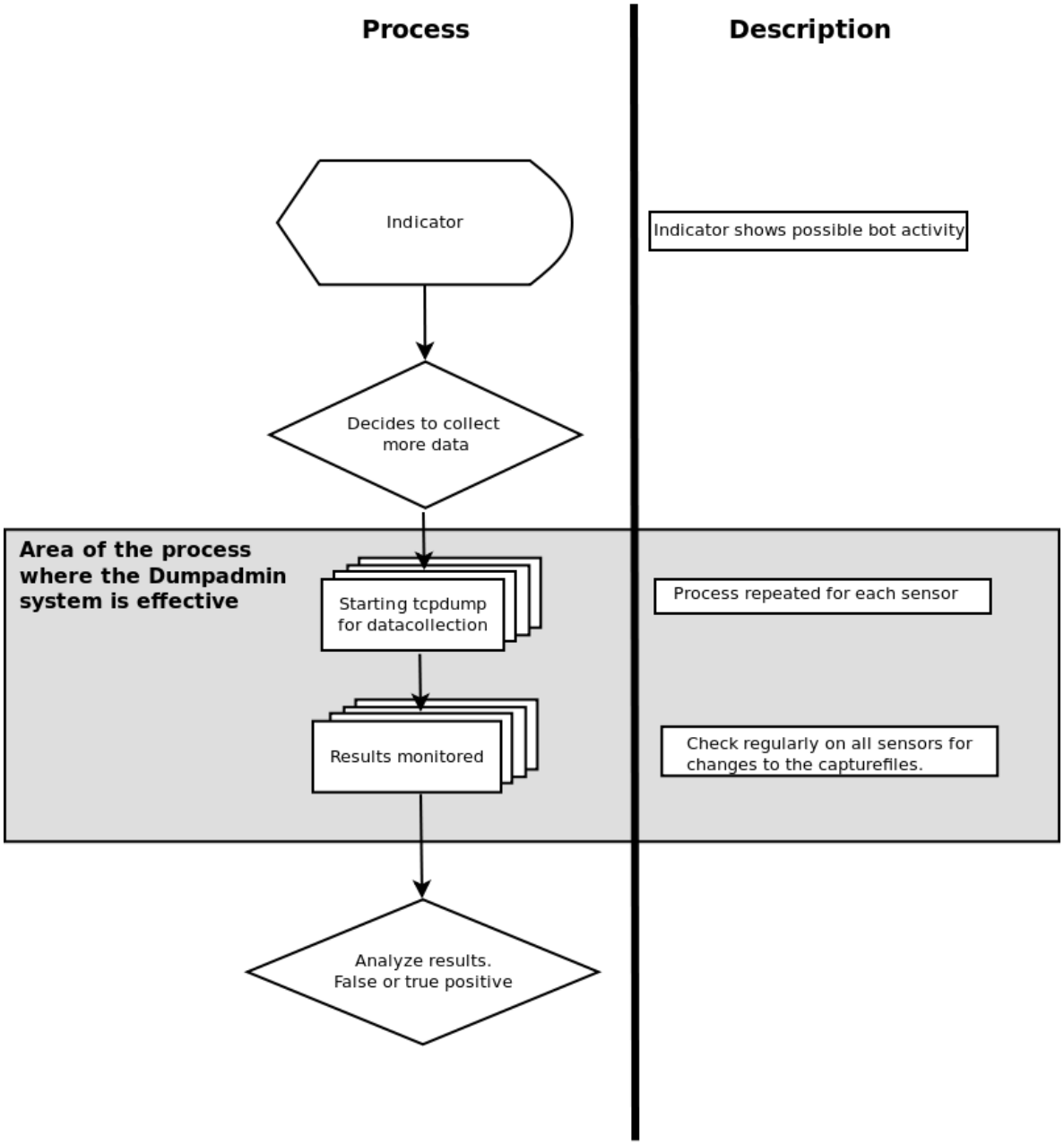


Figure 2: Verification process

The current state of the art is represented mainly by commercial products. These products are expensive and to be attractive on the market they have to appeal to the people with the means to buy them. In most cases that is not technicians or security specialists, but upper management staff. The most popular products today focus on compliance and automation and make promises to be the "complete solution". For analysts they are often too complex to administer, not flexible enough and difficult to optimize for the relevant infrastructure. Further on in a military context the goal of defending networks can have more nuances than stopping the attack as fast as possible. To obtain the necessary flexibility and the ability to operate in any infrastructure presented, analysts often end up making their own toolbox compiled of a collection of open-source tools combined with in-house developed scripts. Inquiries to several highly recognized actors in the security business both at the national and the international level have yielded the same answers concerning CND tools.

The SANS methodology for incident handling which is widely adopted as best practice follows the steps of preparation, identification, containment, eradication, recovery and lessons learned [5, 6, 7]. This process does not take in the possibility of military defensive operations. As mentioned above, in a military context there are often other priorities than stopping the attack and recovering the system. To support future operations it is always important to gain as much knowledge about an adversary as possible. One way to do this is to watch the adversaries' movement and collect information. In [8] Pham defines the differences between events and incidents. He points out that who, what, where, when are the questions important to answer when verifying an event. He goes on pointing out that the most important thing to answer is whether the event is contrary to the security policy. In information warfare there is a question that is at least as important as those mentioned. The question is why? What were the objectives of the adversary? Answers to this question will be very important input to our own planning process. Although there exists a security policy for military networks, policy violations will be secondary to keeping a relative information superiority compared to the adversary.

As mentioned, there is a need to have tools that will have effect in all kinds of networks. Some commercial businesses have the privilege of being able to build their corporate networks with security in mind and making them optimal for an active defense. This privilege is not the case for most military security analysts. Primarily the networks are built for functionality and the security is added when needed. Therefore a military security analyst has to make use of the infrastructure available. This, as pointed out above, shows the need for simple and scalable tools that can be incorporated with minimal effort and be effective in any infrastructure. Experience shows that tools an analyst always will need include simple command line tools like tcpdump, netflow, ngrep and the likes. These tools will be effective in any environment. What they lack is enterprise support making them scalable for large distributed deployment. This thesis shows how tcpdump can effectively be used for verification in a multi-sensor infrastructure.

## 1.6 Research questions

Based on the the previous discussion, in this section we define our research questions. As stated, we will show that tcpdump can be used more effectively. In order to prove increased effectiveness we define the following research questions:

- To what extent is it possible to integrate the use of packet capture tools in a way

that minimizes the added effort of starting a packet capture when expanding the number of sensors?

- Can the data from the captures be organized in a way that makes them accessible for the analyst?
- If so, can this integration also be used to document the process in such a way that it reduces the need for administrative documentation?

## 1.7 Claimed contributions

As a result of the research in this thesis, the following contributions will be made.

- Definition of a scalable effective system for packet capture in CND analysis.
- Definition of a method for integrating unscalable open source command line based security tools for efficient CND incident analysis.
- Implementation of a prototype for the methods mentioned above.

# 2 Related work

## 2.1 Related work

Although information security has been well established as a profession, active defense of computer networks and especially cyber warfare are very new professions. Both tools and procedures are focused towards compliance and "optimized" environments [9]. To actively defend an information system in a cyber warfare perspective the analysts need to focus their efforts slightly different. The main goal of Computer Network Defense is not always to stop an intrusion or an attack as fast as possible. There are several traditional defensive military operations that can be conducted in cyber space. How one handles attacks in cyberspace depends on the overall operations plan and must support the commander's need for information superiority. Information superiority is the single most important prerequisite for success in asymmetric warfare [10]. One of the key requirements is flexible tools for information collection. The field of defensive computer network operations is young and research is limited. The relevant literature that is found is not directly related to CND, but more focused on tools for security professionals.

### 2.1.1 Commercial tools

Examples of tools developed for security monitoring are Symantec Security Information Manager[11], Netforensics SIM One[12] and CiscoWorks SIMS[13]. These tools are so called security information managers. They aim to collect and correlate security related information and present events to the analyst. Common for these tools is that they aim to be "the" complete tool for the security organization in any business. These tools focus on giving high-level views to the user/analyst and are designed specifically to prove compliance with different laws, regulations and standards more easily [11, 13, 12, 14, 15].

In [9] Bouchard discusses requirements for log collection and security monitoring solutions. He argues that "every IT department needs log management". The paper describes which requirements one should consider when choosing a log management solution. Bouchard goes on to point out that there is a need for more than a passive log collection system and that live event monitoring and handling is a necessity as well. The focus of the paper is directed towards enterprise solutions for log management, but some of the criteria are relevant for other solutions as well.

Similarly, [15] points out pitfalls to consider when choosing a SIM. Both these papers are written for Netforensics and the recommendations must be considered favorable to products from Netforensics. However, most of the considerations are relevant for security thinking and should be taken into the process of building an environment for security analysts.

### 2.1.2 Open-source tools

One effort to make a collection of tools for detection and analysis is NSMNow[16], which is mostly based on sguil [17]. Although a good effort, this tool is not yet flexible enough to be the solution in CND. sguil has the possibility to look up netflow directly from a

Snort[18] alarm, but not the ability to start packet capture for that specific address. Further on, the project is now on hold according to the web-page when it comes to functionality and will only be updated with bug-fixes.

### 2.1.3 Related work conclusion

Since the military discipline of CND and even computer security are young fields of science, the prior research in this specific area is very limited. The examples above represent the current state of the art, but are only peripherally relevant to the problem at hand.

# 3 Choice of methods

This thesis was a qualitative study based on existing tools and procedures. We present a system for using tcpdump more effectively in an environment with several sensors. This system is based on experience from CND work in Norwegian Defense Security Service (NDSS) and in dialog with operators. A prototype of the system is tested in a live environment in NDSS. Results are based on both a survey and a calculation of efficiency as described below. The work done in this thesis is a good basis for further development of CND tools and procedures.

The goal of making this sort of interface is to make the CND process more effective. However mainly measuring the time or number of operations used to complete the tasks will not necessarily give the correct view of overall effectiveness. For some reason organizations buy expensive analyst tools to provide good working conditions for the analysts, but the analysts in our experience keep returning to the same non-optimized command line tools. The reason for this may lay in usability. Therefore the success of this project will be measured as a combination of increased efficiency and acceptable usability. Acceptable usability in this case will be based on interpreting user feedback and deducing whether the system will be the preferred tool over the existing solutions.

## 3.1 Efficiency

To measure how efficient the new system is compared to the existing manual process simple keystroke-level modeling was used. In [19] it is described how to use Keystroke-Level Modeling (KLM) to estimate execution times. In this thesis there is no need to estimate execution in time. The number of physical operators will provide a sufficient basis for comparing the new system with the manual method currently in use. Therefore the number of keystrokes needed to complete the task was counted and compared.

## 3.2 Effectiveness

In order to capture effectiveness differences based on the operators' experience of the new system a field test was conducted. The system was installed in a live environment and operators were given an opportunity to use the system in their daily routine. To increase usage in a limited experiment period some tasks were created for the operators. The tasks represent problems similar to real incidents. After the experiment period was concluded the analysts were set to answer a questionnaire. The questions were designed to reflect how easy the system is to learn, the user experienced efficiency and the overall value of such a system.

# 4 System for integrating non-scalable command line tools for verification on multiple sensors

This thesis provides a more effective way of using tcpdump in an environment with multiple sensors. Tcpdump is an open-source command line tool used to capture network traffic in real time. This tool is not in its own designed for use in an enterprise environment and therefore does not scale well for multiple sensors. When detecting and handling security incidents the task of verifying indicators and seeking information on who conducts the attacks and why they are doing so is a very important part of the process. What an analyst will always need is more information. The required information is often sought by using tcpdump, or other similar tools, to provide full-content packet captures of the relevant communication. When sensors are distributed in a large network the task of starting packet captures with the correct filtering on all sensors can be tedious and error prone. To make this process easier, this thesis proposes a way of abstracting the interaction with tcpdump on each sensor providing the operator with one common interface to all sensors.

## 4.1 The system

The main idea of this thesis is to provide a Web based interface for the operators to input the correct command and filter. This interface also provides a list of available sensors and gives the operators an option to select which sensor should activate the capture command. Every command sent to the sensors is stored in a database making it possible to provide history for all captures. The benefit of this feature will be discussed later. Further on, the interface shows all ongoing captures and their status. When a capture filter receives data and the capture file is updated the status immediately changes in the interface. In this way analysts can immediately see activity and act upon the incidents. Figure 4 shows an example of how a typical network is set up. The operators use clients placed in a local network. The Dumpadmin system runs on a central master server. This server can be placed locally on the same network as the clients on a separate network. Practical and security considerations apply as for all networks and the network should be designed to best serve the needs of the organization. The main requirement is that the master server has to be able to connect to all the remote sensors and all clients must be able to connect to the master server. The different remote sensors do not have to be able to connect to each other.

### 4.1.1 Compatibility and environment requirements

The system proposed in this thesis communicates with all sensors in a given environment. To accomplished this it is assumed that there is one central place in the network environment where the system can be installed witch provides network connectivity to all sensors. The system can be installed on an existing server or on a dedicated server. Installation requirements is not taken into further consideration in this thesis apart from the design choices described for the prototype implementation. For the field test the existing network at NODSS/CIPC was used.

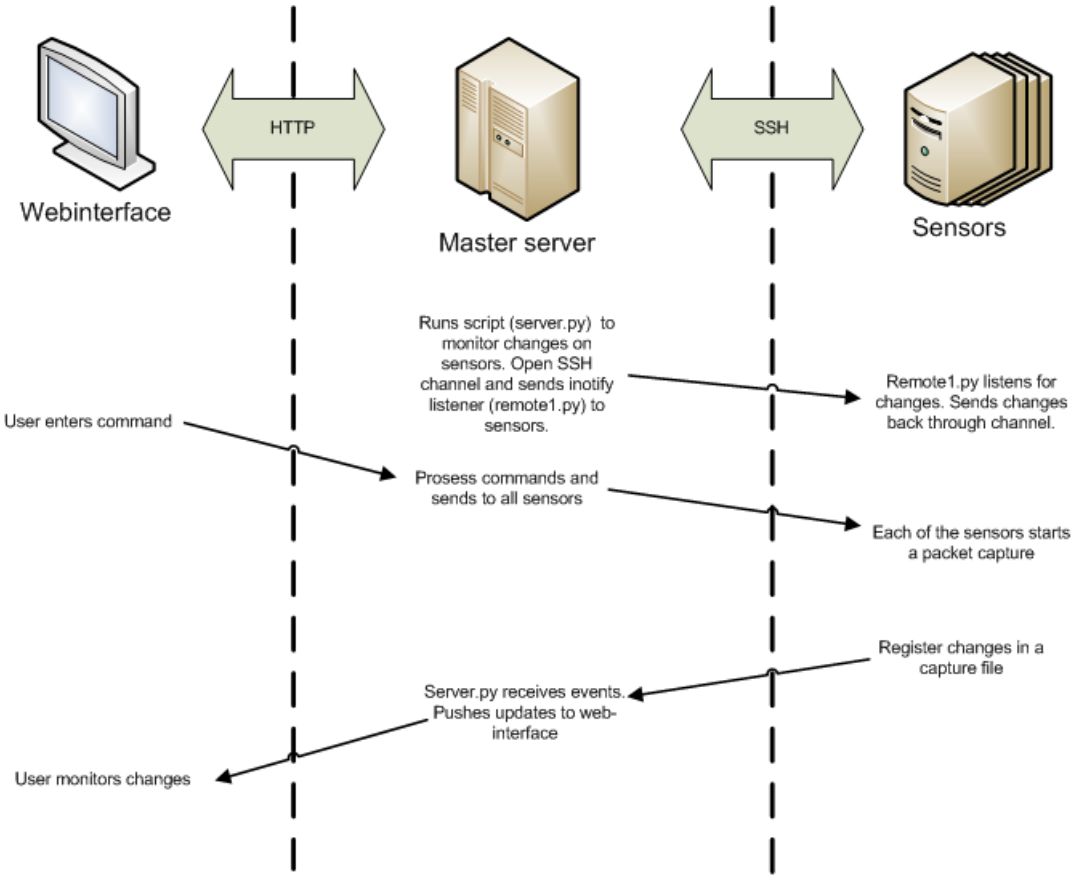


Figure 3: Workflow in Dumpadmin

### 4.1.2 Communicating with sensors

One of the key functions of the framework is to send commands to different remote machines. Sub-requirement for this function is to be securely transmitting. The universal way of communicating securely with a command line interface is SSH. Therefore separate SSH connections to multiple servers is a requirement. The second key requirement for the command interface is that an analyst should be able to send commands to the different remote machines without needing to handle the SSH connections himself. The interface should also be accessible for several analysts simultaneously. Because of this, the interface is chosen to be Web based. In this way, one can easily distribute the interface through existing servers and protocols.

Lastly, the interface must provide the analyst with information regarding updates to the captures, preferably in real-time. This is required for the operator to be able to react quickly to incidents.

Other beneficial properties of the interface are logging of all actions and history of all captures. This gives a higher degree of verifiability in regard to actions taken by the operators,[1] which governmental organizations are required to provide.

## 4.2 Design choices

To test the framework and prove the increased effectiveness, a prototype was developed. This section describes which choices have been made considering design and implementation of the system.

### 4.2.1 Platform

Tcpdump is a tool originally developed for POSIX platforms (Unix, Linux) and later ported to Windows systems under the name windump. Based on the fact that tcpdump is natively installed and supported in most Linux systems, Linux is the platform chosen for this thesis. Among the many different flavors of Linux Ubuntu was the operating system of choice. Ubuntu is easy to install and maintain, has very good support both officially and by a large user community and has good support for different types of hardware. Also, the author has been using Ubunutu for some time and is somewhat familiar with its inner workings.

### 4.2.2 Programming language

There are several programming languages that could serve the purpose for this project. However, the language of choice is python. This is based on compatibility with other systems at NODSS/CIPC and the author's prior knowledge of this language.

### 4.2.3 Web Framework

The Django [20, 21]framework is perfect for easily creating advanced web based projects with python. The developers of Django have taken great care to separate logic form presentation and very little python code is mingled in with the html. This makes it possible to make changes to either the apperance or the logic without having to change them both. Django uses what is called an MVC development pattern . The acronym MVC stands for Model, View and Controller. As explained in the Django Book ([20]) the model is the data access layer. This is where data gets accessed and validated. The view part is the

[1] In Norwegian this is called "notoritet" which means one should have logs detailing all actions in such a way that a controlling body can get a clear view of incidents and how they are handled. More on this requirement in Section 4.4

layer taking care of presenting data to the user. In Django this is accomplished through templates. Lastly, the controller is the layer that handles all the logic of the application. This is where decisions are made based on user input. In Django this happens in the views.

## 4.3 Architecture and implementation

The main goal of the system is to send commands to several servers in different locations through SSH. The increased number of operations due to increased number of sensors should be transparent to the operator/analyst. To manage this, the system is built up of the following components:

**Django based Web interface** Handles web interaction, user-authentication and database interaction

**Database** The prototype is based on Sqlite3[22], but any database could be used. Stores sensors, captures and commands, users and other information necessary.

**Orbited and Stomp** Server for pushing events to web.

**Paramiko** SSH conncetions.

**Pyinotify** Trap inode events to watch changes in capture files.

**Execnet** Run pyinotify remotely through SSH channel

The most important part of the framework is were commands are entered by the analyst and sent to the different sensors. This is done through a simple form created with a few lines of Django code and executed via Paramiko. Paramiko is a Pyhton library constructed to easily connect and send commands through SSH.

Figure 3 shows the work flow of the Dumpadmin system. First of all the necessary server needs to run on the master server. Obviously a Web server is needed to provide the interface for the user. The Web server must be compatible with python and Django. For the prototype, the development server bundled with Django was used. Since Web technologies for the time being do not natively support sockets and thereby two-way communication we needed some mean to push updates to the Web interface. For this we used the COMET server Orbited and the Stomp client library. Orbited runs as a standalone server and handles connections from the browser and interacts with the Django application. The last script that needs to be run is server.py (Appendix B)which was developed for this prototype. It handles the events from the sensors. This is performed by connecting to all the sensors via SSH using the execnet library and establishing a gateway. Through this gateway, a separate script is executed to simply listen for inode events in the folder specified for captures. The pyinotify library provides this functionality. When an inode event is captured it is sent back to the master server through the gateway and server.py sends it to a message channel on the orbited server. These connections are run in separate threads so events from different sensors will not block each other. Starting a capture is executed by entering the required information in the Web interface. The interface for starting captures is shown in Figure 5

The following input fields are shown:

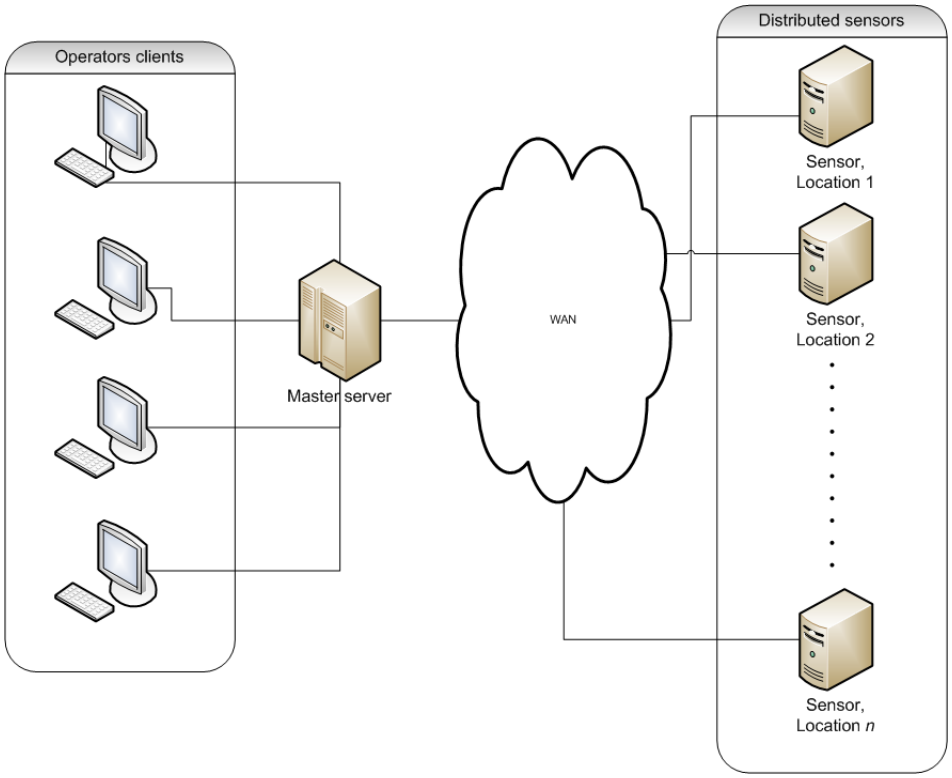


Figure 4: Outline of sensor network

- Command
- Description
- Authorized
- Sign
- Dfile
- Sensors

The 'Command' input field is the most important of all the input fields. It is intended for the tcpdump command followed by the necessary switches and a filter. The '-w' switch to give an output filename should not be used.The filename is given in the 'Dfile' field and the switch is handled by the system. The 'Description' field is used to give some background for this particular capture. For the example used earlier in this thesis (Section 1.3.1) a description could be something like:

> All traffic to known IRC bot controller. Shows possibly infected clients.

When the capture is a part of a case or ongoing operation a reference should also be present in the description.

To keep track of who is authorizing the use of full-content captures, the 'Authorized' field should be filled in. This information is required when inspected as described in Section 4.4. The 'Sign' field is intended for the initials of the operator executing the command. 'Dfile' is where the name of the output file should be entered. In the prototype the

Figure 5: The add capture interface of Dumpadmin

filename cannot include whitespace. The filename of the destination file is also used as the name of the identifier for the screen command used to run tcpdump. Finally, the 'Sensors' field lists available sensors. Multiple sensors can be selected by holding the control (<CTRL>) key when clicking on sensor names. When the user pushes the 'Run' button the form is sent to the Django application running in the Web server on the master server. In this application the following takes place. First the complete tcpdump command is built up. In order to make the output from the command accessible by logging on to the sensor manually. The tcpdump command is run inside a screen. The screen is identified with the same name as the filename for the capture file. The filename given from the user is prefixed with the current date and time to ensure uniqueness. When the complete command is ready for execution the Django application uses the Paramiko library to establish an SSH connection to each of the sensors in turn and executes the command. When the command is successfully executed the operation is stored in a database. Finally, a response is sent to the browser showing a list of the ongoing captures. When changes in any of the capture files occur an update is sent to this list via the Orbited server as described above.

Further, the web interface gives access to a couple of other important features. First, the 'Ongoing captures' page which is were all the ongoing captures is listed and their status updated in real-time. This is the view the operators would use most as part of their daily monitoring of the networks. The last feature worth mentioning is the 'Inactive captures'. This page produces a list of all previous captures that are now inactive. This list can be exported as PDF an is what would be shown to an inspection team.

## 4.4 Legal considerations

As information security professionals, we have to consider the legal aspects of our work and constantly monitor how relevant rules and regulations apply to our work. take care about legal aspects that need to be considered. The legal implications of monitoring computer networks for security incidents are different under the different national laws. The principles described here are based on Norwegian laws and regulations and are taken from the directives given for governmental and military practices. CND in the Norwegian defense are mainly regulated through the directive for the security in the Norwegian Defense, although new directives are under development. These directives points out who should perform security monitoring and incident handling as well as some examples of what is meant by these activities. Apart from that, there are at the moment few directions when it comes the the legal implications of CND. However, there are a few legal limitations that are especially important to consider on a daily basis in this line of work. First of all there are privacy concerns. Capturing and monitoring network data can give access to private information which must be handled accordingly. Privacy related issues are regulated through the law on handling private information (Personopplysningsloven, [23]). In addition, law on police authority for military personnel (Lov om militær politimyndighet, [24]) and law on defense secrets (Lov om forsvarshemmeligheter, [25]) authorize use of certain enforcement measures that has to be balanced between military operational concerns and national security on one side and privacy and legal protection of individuals on the other side. To assure that all public administrative bodies, such as the defense, all act within their authority, several controlling bodies regularly inspect the relevant government branches. In order to account for all past ac-

tivities when inspected it is imperative to be able to present detailed documentation on all activity. How, when and why packet captures are started are one of the issues that is vigorously controlled to ensure that the right judgments have been made concerning privacy issues.

# 5 Experiments and results

## 5.1 Keystroke-Level Modeling

In this thesis, the properties of the framework was measured in two different ways to show whether the goals have been reached. First, we performed a simple Keystroke-Level Modeling (KLM). This is basically counting the number of keys one has to press on the keyboard to start the desired capture. The capture filter will affect the number of keystrokes required and different filters might yield different results. Therefore, we shall give examples with different lengths of filters to show the relationship between the manual method and using the Dumpadmin system. The two methods can be compared using the following equations:

$$K_m = k * s \tag{5.1}$$

where:

$K_m$ is total number of keystrokes (manual method)

$k$ is the number of keystrokes per sensor

$s$ is total number of sensors

$$K_{da} = k_c + k_d + k_{auth} + k_{sign} + k_f + s \tag{5.2}$$

where:

$K_{da}$ is the total number of keystrokes (Dumpadmin)

$k_c$ is the number of keystrokes in tcpdump command (screen and -w <filename> not included)

$k_d$ is the number of keystrokes in description

$k_{auth}$ is the number of keystrokes in name of authorizing officer

$k_{sign}$ is the number of keystrokes in signature of operator starting the capture (initials only)

$k_f$ is the keystrokes in the filename (without extension)

$s$ is the number of sensors

When starting a packet capture from the command line, the command required is of the form:

```
screen -S '<name of screen>' tcpdump -<necessary switches> -w
  <filename> '<bpf filter>'
```

To calculate keystrokes for comparison of the two methods we used the following three tcpdump commands:

Listing 5.1: "Example tcpdump commands"

```
screen -S 'screenname1' tcpdump -nns 0 -w capturefile1.cap 'host 172.16.13.245'
screen -S 'screenname2' tcpdump -nns 0 -w capturefile2.cap 'host 172.16.13.245 and port 1337'
screen -S 'screenname3' tcpdump -nns 0 -w capturefile3.cap 'host 172.16.13.245 and not port 80 and not port 22 and not port 443 and not port 25 and not port 23'
```

Note that they are all preceded with a screen command. Since several operators should be able to work on the sensors and terminals have to be closed occasionally, all commands are run in screen to be able to reconnect from other terminals. In Dumpadmin, the screen commands are handled by the system itself and will not result in extra keystrokes.

Given Equations 5.1 and 5.2, we can calculated the total numbers of keystrokes for the two methods based on the above examples (Listing 5.1). In Table 1, the numbers of keystrokes for the different operations are given. In addition, there will be keystrokes for tabbing between input fields in the Dumpadmin interface. Note that execution times are not given and mental processes are left out of the model. These factors will not significantly affect the relationship in efficiency between the two methods.

| Example | k | $k_c$ | $k_d$ | $k_{auth}$ | $k_{sign}$ | $k_f$ |
|---|---|---|---|---|---|---|
| 1 | 78 | 35 | 50 | 8 | 3 | 8 |
| 2 | 92 | 49 | 50 | 8 | 3 | 8 |
| 3 | 159 | 116 | 50 | 8 | 3 | 8 |

Table 1: Keystrokes

As shown on the plots of the three example commands (Figure 6, 7 and 8) there was a significant improvement in efficiency in terms of keystrokes already with two sensors. When adding sensors the gain increases. The size (length) of the Berkeley Packet Filter(BPF) filter for the packet capture does not significantly change the relationship between the two methods. As shown in (Figure 9, 10 and 11), the average number of keystrokes per sensor is constant when using the manual method. When using the Dumpadmin system on the other hand the average amount of keystrokes will decrease as the number of sensors increases.

By means of keystrokes a considerable increase in efficiency was achieved. In addition there are several factors that was not measured in this model. First of all the complexity of using different shells for all the sensors will make it very difficult and time consuming for the operator to ensure that the commands are executed correctly on each sensor. This manual procedure also opens for a risk for human error. Further on there will be no central log keeping track of active and stopped captures. The legal implications were discussed in Section 4.4. Although Keystroke Level Modeling takes into account time used for cognitive or mental processes it would not be possible to reliably compute the time use for mental processes due to use of several different terminals. This would not be a linear function of the number of terminals and would require a study based on empiric data. Anyway this limitation does not affect the result of this experiment. The operator's experience, and therein the cognitive processes, was taken into consideration in the survey and interviews with the operators.

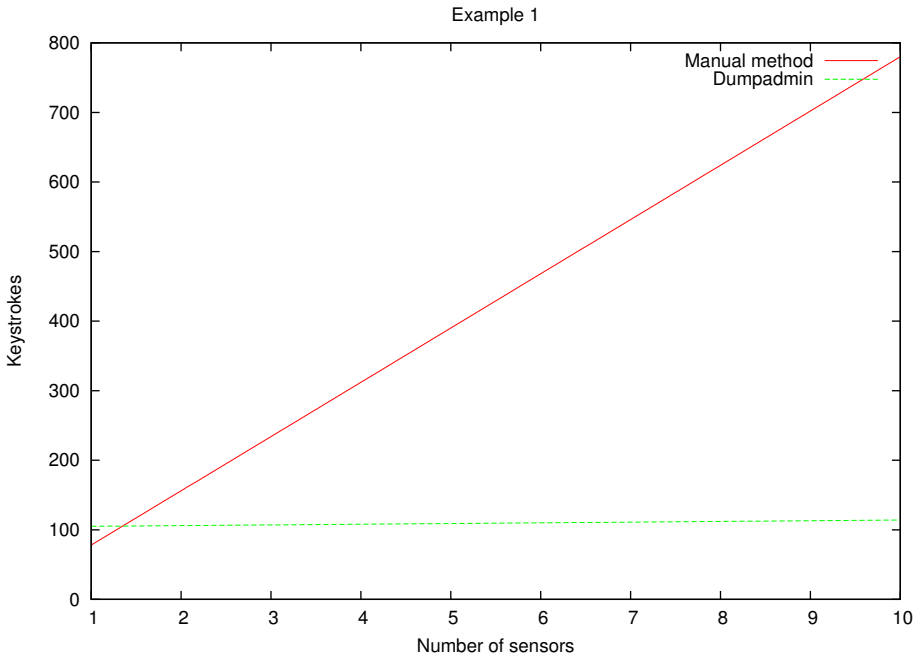


Figure 6: Keystrokes as function of the number of sensors (example 1)

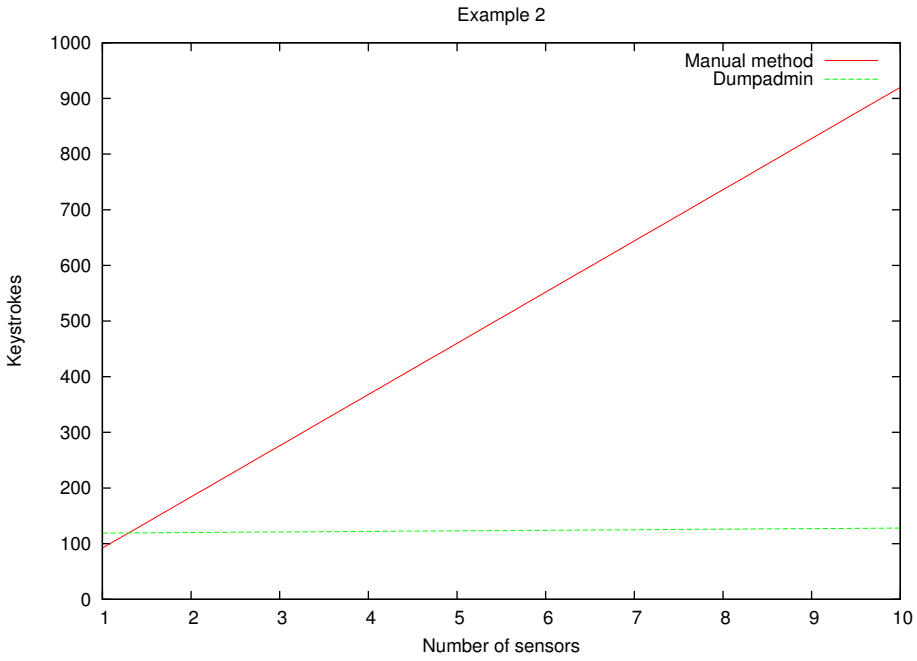


Figure 7: Keystrokes as function of the number of sensors (example 2)

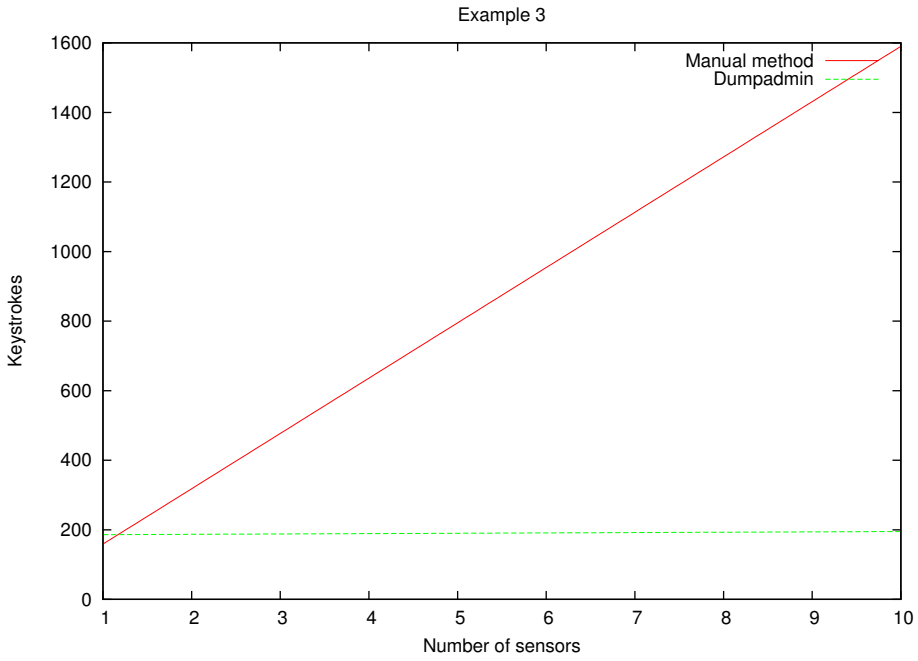


Figure 8: Keystrokes as function of the number of sensors (example 3)

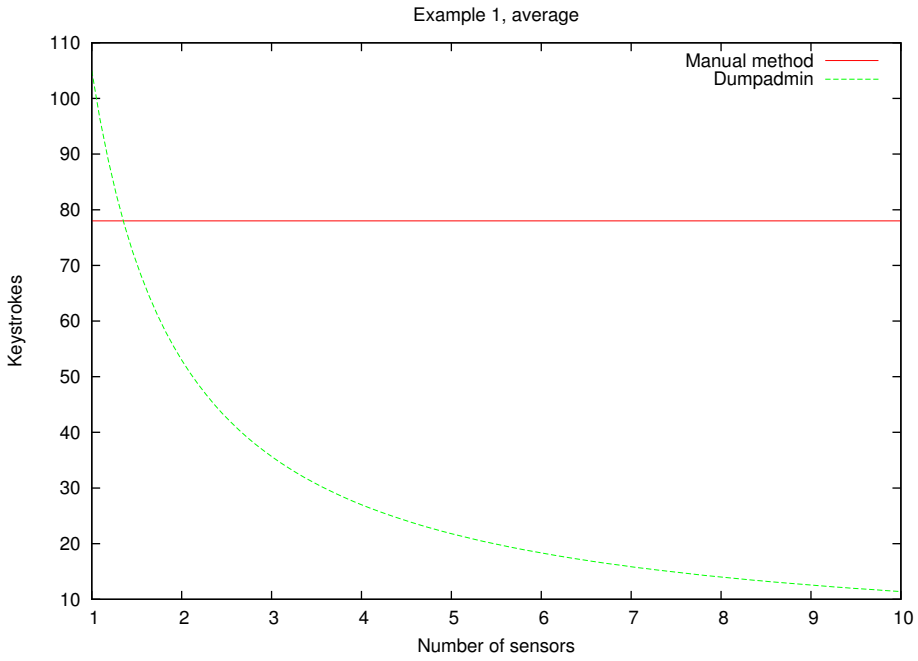


Figure 9: Average keystrokes per sensor (example 1)

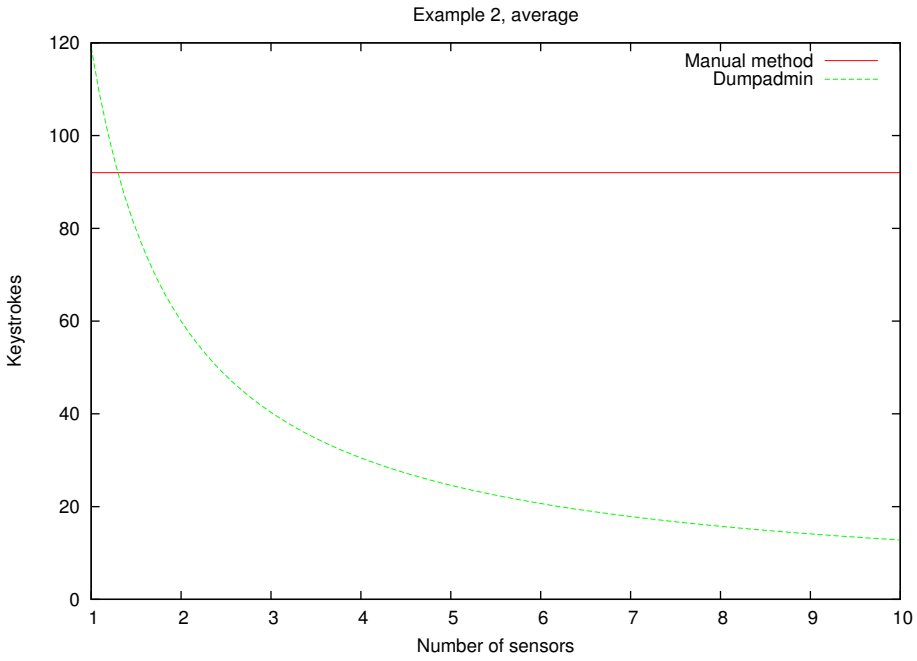


Figure 10: Average keystrokes per sensor (example 2)

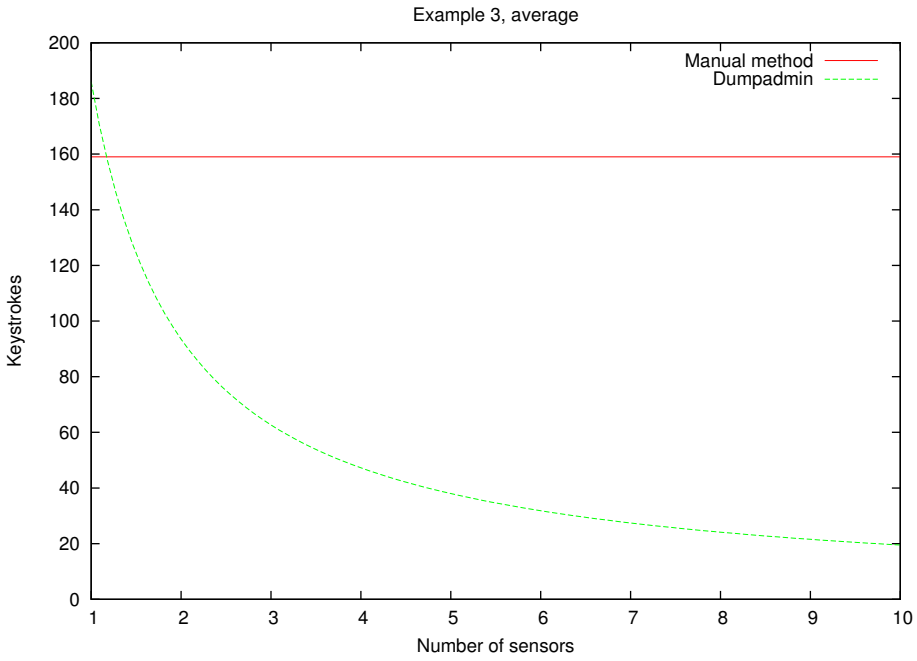


Figure 11: Average keystrokes per sensor (example 3)

In these examples calculations are shown only for up till 10 sensors. The patterns is linear so increasing the number of sensors will not affect the results. How many sensors the system could handle is not tested in this project. Increasing to a hundred or even several hundred should not cause problems because very little processing is required. What could be a limiting factor is the number of simultaneous network connections. Testing this is outside the scope of this project and such large numbers of sensors are not relevant at the time. However, such a test should be conducted for different types of hardware before putting the system into production.

## 5.2 Percieved effectiveness

In addition to KLM, we want to measure the experienced or perceived effectiveness of the system. That is, if the operators are more effective and feel more comfortable using this system compared to the manual way of starting packet capture. To accomplish this, a prototype was implemented and tested by a group of test subjects.

### 5.2.1 Test subjects

The test subjects were professional operators and analysts at the NODSS/CIPC. They are employed to monitor and analyze security incidents in the Norwegian Defense information systems. These operators are the main target group for this system and will provide valid results.

### 5.2.2 Testing

The testing was performed in a operational network at Norwegian Security Service Critical Infrastructure Protection Center. The system was installed parallel to other existing tools and the operators were provided with tasks that should be completed using the Dumpadmin system.

As stated in [26], the quality of a system relies on its ability to function in the real world. The test subjects have significant experience in detecting and analyzing security incidents and can test the system for real life performance. After completing the tasks they were given a questionnaire containing questions designed to evaluate usefulness, ease of use, ease of learning and satisfaction. The questionnaire was based on the paper by Arnold Lund in the October 2001 issue of Usability Interface ([27]). The questionnaire includes nine statements with a scaled answers from 1 through 7 where 1 is disagree and 7 is agree. Lastly, the test subjects were also given the opportunity to give additional comments on how to improve the systems usefulness. The complete questionnaire is shown in Appendix A. In addition the test subjects were observed occasionally during the test period and issues were discussed as they emerged.

The deployed system were merely a prototype so comments regarding bugs, system features and aesthetics is not considered in the effectiveness evaluation, but the most important comments were used as input for future work.

### 5.2.3 Results

Results from the questionnaires were gathered and mean, median and mode were calculated. Only eight operators answered the questionnaires. This a small sample, but still represents 67% of the operators that would normally use this system and it is therefore likely that the results are representative for the unit. Due to the nature of their shifts and personnel on leave a 100% sampling rate was not possible. The effect of the samplesize

on the results will be discussed further in the next section (Section 5.2.4).

| Question(statement) | Mean | Median | Mode |
|---|---|---|---|
| The system would help me be more effective | 4.4 | 5 | 5 |
| It would be useful in my daily job | 4.4 | 5 | 5 |
| Both occasional and regular users would benefit from it | 4.6 | 4.5 | 7 |
| I can use it successfully every time | 5 | 6 | 7 |
| The system is easy to use | 4.5 | 5 | 4 |
| The system was easy to learn | 5.3 | 6 | 7 |
| I easily remember how to use it | 5.5 | 7 | 7 |
| The system makes it easier to monitor packet captures | 4.9 | 5.5 | 5 |
| I prefer this system over the manual method for starting and monitoring packet captures | 5.1 | 6 | 6 |

Table 2: Results from questionnaires

Table 2 shows the results from the questionnaires filled out by the test subjects. The results show that the majority of the operators agree that the system will increase their effectiveness if used. They also prefer to use this system over the manual method.

Based on the comments from the test subjects there are a few factors to consider when interpreting these results. First of all, it is not possible to ensure a common basis for answering the questions. Some subjects gave a very low score for usefulness based on missing features and other have given scores based only on existing features. However, this affects the results in a negative direction meaning that it is still safe to conclude that a system based on this prototype would increase the overall effectiveness of the operators. Furthermore, the concept of the interface construction is acceptable for the operators and can be used successfully.

A few additions should be implemented in the system before put into production use. First of all there should be a possibility to use BPF filters from a filter file instead of entering the filter in the command field. Other features that would increase the usefulness of the system are the ability to edit ongoing captures, better error checking and recovery, as well as some way of inspecting the content of captures.

### 5.2.4 Reliability and validity

The sample size in this survey is to small to make the results quantifiable. That was not the purpose of this research either. As stated in the research questions the purpose of this research was to find whether or not it is possible to make the way packet captures are used in security incident verification more effective. Therefore a qualitative approach has been chosen. The sample size is mainly a convenience sample [28] in that all available operators has answered the questionnaire. However, there is a large amount of judgment in sampling implicated as well. The test subjects in this survey is a very homogeneous group in terms of tasks, experience and education. Therefore it can be stated with high confidence that the sample group is representative. Further on comments on the questionnaires and discussions with and observations of the operators suggests that we have reach data saturation with this sample.

One operator in particular has given answers that has affected the results negatively. As explained above the operators might have answered on slightly differing premises. This is due to evolvement of established procedures parallel to this study. In particular this is true for the most negative answers given from this one operator. The prototype has

not been sufficiently adjusted to the procedures per time, only to the procedures existing at the start of this research (January 2010). Despite this the results still suggests that the prototype shows promise in terms of increasing efficiency and effectiveness.

Due to the ever changing premises it can be difficult to reproduce the exact results of this research. Still a similar survey should yield the same conclusions. More effort could be put in to ensuring a common premise for all the test subjects, but observations of and comments from the subjects will likely reveal the same results.

# 6 Conclusion

This thesis has presented a system for administering full content packet captures for use in verification and analysis in a CND context. The main goal for this system is to increase effectiveness of the CND process. We now take a look at the research questions stated in Section 1.6 and conclude for each of them separately.

1. To what extent is it possible to integrate the use of packet capture tools in a way that minimizes the added effort of starting a packet capture when expanding the number of sensors?

This thesis has shown that it is possible to almost eliminate the added effort by adding extra sensors. This has been shown by KLM of the prototype implementation. By a usability survey, a set of test subjects have confirmed that a system like the presented prototype would make the verification process more effective. However, it was pointed out that some extra features would add to the benefits of the system. The most beneficial feature would be the ability to use predefined files for BPF filtering in the tcpdump command. There might be limitations to the number of sensors that can be administered, but not in the range of sensors that will be in use at CIPC in the foreseeable future.

2. Can the data from the captures be organized in a way that makes them accessible for the analyst?

The survey showed that the ability to effectively monitor the captures for changes was increased when using the Dumpadmin system. To further add to this increase, the possibility to do simple inspection of the capture files should be added. There is also a need to organize the way changes are presented in the 'Ongoing captures' view in order to accoubt for a greater number of sensors.

3. If so, can this integration also be used to document the process in such a way that it reduces the need for administrative documentation?

The proposed system stores all active and stopped captures in a database in such a way that it can later be documented what captures have been used for what purposes. As discussed in Section 4.4 this is an absolute requirement. When included directly in the system the need for extra logging and documentation is decreased which again makes for increased effectiveness for the operators.

# 7 Future work

This thesis has shown that it is possible to significantly increase effectiveness in CND by making interface to more easily use command line based tools in an environment of multiple sensors. Drawing on experience from this thesis, there are at least two paths of research that would benefit CND professionals. First off, the system proposed in this thesis could easily be expanded to include more tools and so be a more complete toolbox. To accomplish this several aspects needs to be studied. First of all, a broad survey of which tools to include should be conducted. A thorough usability study of the interfaces of existing SIMs and other commercial information security products could give valuable input to what works and what does not. Apart from giving operators the necessary tools, giving them an interface they can effectively use would be critical for the success of such work. The scope of these problems are wide and would probably be best solved in several delimited studies.

As a supplement to the proposed research a slightly different path is to build in more features to the system proposed in this thesis and conduct an empirical study of the usefulness of the system among several different user groups. Before starting a survey the possibility to load filters from files and some simple query options should be added. Possibly some way of editing the filter files through the interface would also be beneficial. The different filters should include comments describing the purpose of each filter. Lastly, some effort should be put in to error checking and correction making the system more robust. To get the best results from such a survey the prototype should be adjusted to fit the current work processes and should be tested and adjusted in cooperation with operators.

# Bibliography

[1] stabsskole, F. 2007. *Forsvarets Fellesoperative Doktrine*. Forsvarsstaben.

[2] Bejtlich, R. 2004. *The Tao of Network Security Monitoring - Beyond Intrusion Detection*. Addison-Wesley.

[3] Holdaway, E. J. 2001. Active computer network defense: An assessment.

[4] Hildreth, S. A. 2001. Cyberwarfare. *CRS Report for Congress*.

[5] Bilardo, C. 2002. Deterring cyber attacks.

[6] Pokladnik, M. 2007. An incident handling process for small and medium businesses. *SANS Information Security Reading Room*.

[7] Morreale, T. 2008. Incident handling for smes (small to medium enterprises). *SANS Information Security Reading Room*.

[8] Pham, C. 2001. From events to incidents.

[9] Bouchard, M. 2009. Demand more from your log management solution.

[10] Applegate, M. 2001. Preparing for asymmetry: As seen through the lens of joint vision 2020.

[11] Symantec. 2008. Symantec security information manager.

[12] Netforensics. 2009. Actionable security intelligence: Preparing for the next threat with a proactive strategy.

[13] Systems, C. 2005. Ciscoworks sims incident resolution management: Unifying the security team to eliminate threats.

[14] Netforensics. 2009. Closing the cyber-security gap in the bulk power system.

[15] Netforensics. 2008. 10 common pitfalls to avoid when evaluating security information management (sim) solutions.

[16] 2009. Nsmnow. http://www.securixlive.com/nsmnow/index.php.

[17] Sguil: The analyst console for network security monitoring. http://sguil.sourceforge.net/docs.html.

[18] The Snort Project. *Snort Users Manual*, 2.8.6 edition, April 2010.

[19] Kieras, D. 2001. Using the keystroke-level model to estimate execution times. 11.

[20] Holovaty, A. & Kaplan-Moss, J. The django book. Internet. Second edition. Only available online. First edition printed on apress.

[21] Django documentation. Internet(last visited 01.06.2010).

[22] About sqlite. Internet (last visited 01.06.2010).

[23] og politidepartementet, J. 2000. Lov om behandling av personopplysninger. Internet, lovdata.no. http://lovdata.no/all/hl-20000414-031.html.

[24] Forsvarsdepartementet. May 1988. Lov om politimyndighet i det militære forsvar. Internet, lovdata.no. http://lovdata.no/all/hl-19880520-033.html.

[25] Forsvarsdepartementet. August 1914. Lov om forsvarshemmeligheter. Internet, lovdata.no. http://lovdata.no/all/hl-19140818-003.html.

[26] Bevan, N. 1995. Measuring usability as quality of use. *Software Quality Journal*, 4, 115–150.

[27] Lund, A. M. 2001. measuring usability with the use questionnare. *Usability Interface, Society for Technical Communication Usability SIG Publication*, 8.

[28] Marshall, M. N. 1996. Sampling for qualitative research. *Family Practice*, 13.

# Appendix A Questionnaire

The next page shows the questionnaire used for gathering feedback from the test subjects.

# Dumpadmin usability questionnaire

| | Disagree | | | | | | Agree |
|---|---|---|---|---|---|---|---|
| The system would help me be more effective. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| It would be useful in my daily job. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Both occasional and regular users would benefit from it. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| I can use it successfully every time. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| The system is easy to use. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| The system was easy to learn. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| I easily remember how to use it. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| The system makes it easier to monitor packet captures. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| I prefer this system over the manual method for starting and monitoring packet captures. | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

What changes or improvements would make this system more useful?

_____

_____

_____

_____

# Appendix B server.py

```
import execnet, os,json,stomp,threading, sys
print sys.path
from batchutils import Daemonize
import remote1
from dumpadmin.models import Sensors

class connectThread(threading.Thread):
        """Threading class to support connections to several
           sensors simultaneously"""
        def __init__(self,host,remote_script,sensor_name):
                threading.Thread.__init__(self)
                self.host=host
                self.remote_script=remote_script
                self.sensor_name=sensor_name

        def run(self):
                self.exec_remote(self.host,self.remote_script,"tommy")

        #Callback function. Used in setcallback()
        def receiver(self,res):
                """Function used as callback for the
                   exec_remote. When events are received from
                   remote sensors they are sent to a orbited
                   channel."""
                res['sensor_name']=self.sensor_name
                print "POSTDATA:␣", res
                message = json.dumps(res)
                conn.send(message, destination='/messages')
                return None

        def exec_remote(self,host,remote,name):
                """This function connects to a remote host
                   throught SSH using the execnet library. The
                   remote host is given by 'host'. Commands
                   (script) to execute remotely is given by
                   'remote'."""

                #Opens a gateway to the remotehost
                gw = execnet.makegateway("ssh=root@%s" % host)
                   #In this case the root user is used. The
                   master server needs to be in the authorized
                   file on the remote sensor to connect
                   without a password. This setup should be
```

```
                changed to an unpriveleged user for security


            #Opens a channel and executes script remote
            ch = gw.remote_exec(remote)


            #Set callback function. Output from remote side
                gets sent to receiver(). All output from
                remote side will be processed immediately
            print "Setting callback"
            ch.setcallback(self.receiver)

            #Waits for remote script to close.
            ch.waitclose()

if __name__ == "__main__":
        #When started a stomp client is initiated and connects
            to the orbited server and channel.
        conn = stomp.Connection()
        conn.start()
        conn.connect()
        conn.subscribe(destination='/messages', ack='auto')


        sensors = Sensors.objects.all()
        for sensor in sensors:
                print sensor.ip
                if sensor.ip:
                        host = sensor.ip
                elif sensor.hostname:
                        host = sensor.hostname
                else:
                        raise Exception, "No hostname or IP
                            address"
                connectThread(host,remote1,sensor.name).start()
```

# Appendix C remote1.py

```
#Script that is sent to the remote sensors
import os,sys,json,re
import pyinotify
from datetime import datetime
sys.path.append('/home/tommy/Documents/Master/Thesis')
#from masterproject import settings
from django.conf import settings

if __name__ == '__channelexec__':

                wm = pyinotify.WatchManager()
                mask = pyinotify.IN_MODIFY

                class PTmp(pyinotify.ProcessEvent):
                        def process_IN_MODIFY(self, event):
                                """Function for handling
                                    modification events"""
                                size =
                                    os.path.getsize(os.path.join(event.path,
                                    event.name))

                                #Gather the details we need to
                                    send
                                #event_details = json.dumps({
                                    "modtime" :
                                    str(datetime.now()),
                                    "newsize" : size, "name" :
                                    event.name })
                                name = re.sub('\.cap$','',
                                    event.name)
                                event_details = { "modtime" :
                                    str(datetime.now()),
                                    "newsize" : size, "name" :
                                    name }

                                #Send event details for
                                    processing by the calling
                                    function
                                channel.send( event_details)

                                #Print for debugging. Comment
                                    out in production.
                                print event_details
```

```
wdd = wm.add_watch(settings.WATCH_DIR, mask,
    rec=True)

notifier = pyinotify.Notifier(wm, PTmp())



while True:
        try:
                notifier.process_events()
                if notifier.check_events():
                        notifier.read_events()
                        notifier.process_events()

        except KeyboardInterrupt:
                raise
                notifier.stop()
                break
```

# Appendix D views.py

```
# Create your views here.

#Imports
from contextlib import contextmanager
from django.contrib.auth.decorators import login_required
from django.contrib.auth import authenticate, login
from django.db import models
from django.forms import ModelForm
from django.forms.models import modelformset_factory
from django.http import HttpResponse,HttpResponseRedirect
from django.template import Template, Context
from django.template.loader import get_template
from django.shortcuts import render_to_response
from masterproject.dumpadmin.models import Captures, Sensors,
    Message
from django.conf import settings
from cgi import escape
import cStringIO as StringIO
import ho.pisa as pisa
import datetime, threading, execnet,paramiko,
    os,json,stomp,pyinotify, time,sys


#Start the Stomp server and subscribe. Necessary for realtime
    updates.
conn = stomp.Connection()
conn.start()
conn.connect()
conn.subscribe(destination='/messages', ack='auto')


class CaptureForm(ModelForm):
        class Meta:
                model = Captures
                #fields = ('command', 'description', 'sensors',
                    'authorized', 'sign', 'dfile', 'active')


def render_to_pdf(template_src, context_dict):
    template = get_template(template_src)
    context = Context(context_dict)
    html  = template.render(context)
```

```
    result = StringIO.StringIO()

    pdf =
        pisa.pisaDocument(StringIO.StringIO(html.encode("ISO-8859-1")),
        result)
    if not pdf.err:
        return HttpResponse(result.getvalue(),
            mimetype='application/pdf')
    return HttpResponse('We had some errors<pre>%s</pre>' %
        escape(html))


@login_required
def display_meta(request):
                """Displays metadata for the server"""
                values = request.META.items()
                values.sort()
                html = []
                for k, v in values:
                                html.append('<tr><td>%s</td><td>%s</td></tr>'
                                    % (k, v))
                return HttpResponse('<table>%s</table>' %
                    '\n'.join(html))

#This should be set up to use certificates and trust.
@contextmanager
def create_ssh(host, username='root'):
        """Setup a SSH connection. Take three arguments (host,
            username, password). Connects to host with username
            and password."""
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        try:
                print "Creating connection"
                #ssh.connect(host, username=username,
                    password=password)
                ssh.connect( host, port=22, username=username,
                    allow_agent=True, look_for_keys=True)
                print "connected"
                yield ssh
        finally:
                print "closing connection"
                ssh.close()
                print "closed"

@login_required
def dump(request):
        """This view provides a form to start a new packet
            capture. When POST data are sent the captures are
            started on the defined sensors. Redirects to list
            of ongoing captures"""
        CapturesFormset = modelformset_factory(Captures)
        form = CaptureForm()
```

```
        #print formset
        if ('run' in request.POST and
            request.POST['run']=='Run') and ('command' in
            request.POST and request.POST['command']):
                filename = 'cap%s%s' %
                    (time.strftime("%Y-%m-%d_%H-%M-%S",
                    time.localtime()),request.POST['dfile'])

                print request.POST
                req = request.POST.copy()
                req.__setitem__('dfile',filename)
                capture=CaptureForm(req)

                results={}
                results['out']=''
                results['err']=''


                for k in req.getlist('sensors'):
                        print "ID: %s" % k
                        #ip_adr=Sensors.objects.get(id='1')
                        sensor=Sensors.objects.get(id=k)

                        if sensor.ip:
                                host = sensor.ip
                        elif sensor.hostname:
                                host = sensor.hostname
                        else:
                                raise Exception, "No hostname
                                    or IP address"

                        with create_ssh(host=host) as ssh:

                                cmd = 'screen -dmS %s %s -i %s
                                    -Uw %s/%s.cap' %
                                    (req['dfile'],req['command'],sensor.interfa
                                #cmd = 'sudo screen -dmS %s
                                    sudo %s -Uw %s/%s.cap' %
                                    (request.POST['dfile'],request.POST['comman
                                #cmd = 'gnome-terminal -x %s -w
                                    /home/tommy/captures/%s.cap'
                                    %
                                    (request.POST['command'],request.POST['dfil
                                #cmd = 'sudo -n %s -w
                                    /home/tommy/captures/%s.cap
                                    &' %
                                    (request.POST['command'],request.POST['dfil
                                stdin, stdout, stderr =
                                    ssh.exec_command(cmd)
                                print cmd
                                out=stdout.read()
                                err=stderr.read()
```

```
                                print 'Output: %s, Error: %s' %
                                    (err, out)
                                #results['out'] += "%s
                                    output:\n\n%s\n\n" %
                                    (sensor.name,out)
                                #results['err'] += "%s
                                    error:\n\n%s\n\n" %
                                    (sensor.name,err)
                        #Her maa destinasjonsfil endres fra filnavn til
                            full path. Path tas fra settings
                        capture.save()
                        #CaptureForm(request.POST).save()
                        return HttpResponseRedirect('/ongoing/')
                return render_to_response('dump.html',{"formset":form})

@login_required
def show_ongoing(request):
        """This view lists the captures that are currently
            active."""
        data = Captures.objects.filter(active=True)
        return render_to_response('ongoing.html',{'data':data})


def login(request):
        """Use the Django included user authentication to log
            in"""
        username = request.POST['user']
        password = request.POST['pass']
        user = authenticate(username=username,
            password=password)
        if user is not None:
                        if user.is_active:
                                        login(request, user)
                                        return
                                            HttpResponseRedirect('/ongoing/')
                        else:
                                        return
                                            render_to_response('login.html',{"error":"
                                            is not enabled"})
        else:
                        return
                            render_to_response('login.html',{"error":"Username
                            or password is not valid"})

@login_required
def stop_capture(request):
                """Stop capture and move the capture file to
                    another directory"""
                print "Stopping capture"
                cap_id=request.POST['id']

                try:
                        cap=Captures.objects.get(id=cap_id)
```

```
                        for sensor in cap.sensors.all():
                                if sensor.ip:
                                        host = sensor.ip
                                elif sensor.hostname:
                                        host = sensor.hostname
                                else:
                                        raise Exception, "No␣
                                            hostname␣or␣IP␣
                                            address"
                                with create_ssh(host=host) as
                                    ssh:
                                        cmd = 'screen␣-X␣-S␣%s␣
                                            quit;mv␣%s/%s.cap␣
                                            %s' %
                                            (cap.dfile,settings.WATCH_DIR,cap.d
                                        print cmd
                                        stdin, stdout, stderr =
                                            ssh.exec_command(cmd)
                                        out = stdout.read()
                                        err = stderr.read()
                                print "Killing␣on␣sensor␣%s" %
                                    sensor.ip
                                print err
                                print out
                        print "Setting␣capture␣non-active"
                        cap.active=False
                        cap.save()
                except Captures.DoesNotExist:
                        print "That␣object␣is␣not␣in␣the␣
                            database"

                return HttpResponseRedirect('/ongoing/')

@login_required
def view_stopped(request):
        sensors = Sensors.objects.all()
        olddir_content = []
        for sensor in sensors:
                print sensor.ip
                if sensor.ip:
                        host = sensor.ip
                elif sensor.hostname:
                        host = sensor.hostname
                else:
                        raise Exception, "No␣hostname␣or␣IP␣
                            address"

                with create_ssh(host=host) as ssh:
                        cmd = 'ls␣-la␣%s' % (settings.OLD_DIR)
                        print cmd
                        stdin, stdout, stderr =
                            ssh.exec_command(cmd)
                        out=stdout.read()
```

```
                        olddir_content.append(
                            {"sensor":sensor.name, "data":out} )
        print olddir_content
        return
            render_to_response('stopped.html',{"data":olddir_content})

def show_inactive(request):
        """This view lists the captures that are currently
            inactive."""
        data = Captures.objects.filter(active=False)
        return render_to_response('inactive.html',{'data':data})

def topdf(request):
    #Retrieve data or whatever you need
    data = Captures.objects.filter(active=False)
    return render_to_pdf(
            'pdf.html',
            {
                'pagesize':'A4',
                'data': data,
            }
        )
```

# Appendix E models.py

```
from django.db import models


# Create your models here.

class Sensors(models.Model):
        name = models.CharField(max_length=30)
        location = models.CharField(max_length=50)
        ip = models.IPAddressField(null=True, blank=True)
        hostname = models.CharField(max_length=60,null=True,
            blank=True)
        interface = models.CharField(max_length=10,null=True,
            blank=True)

        def __unicode__(self):
                return self.name



class Commands(models.Model):
        time = models.DateTimeField(auto_now_add=True)
        command = models.TextField(editable=True)
        description = models.TextField(editable=True)
        sensors = models.ManyToManyField(Sensors, null=True,
            blank=True)

        class Meta:
                abstract = True

class Captures(Commands):
        authorized = models.CharField(max_length=15)
        sign = models.CharField(max_length=50)
        dfile = models.CharField(max_length=150)
        active = models.BooleanField(default=True)

        def __unicode__(self):
                return self.command

class Message(models.Model):
    user = models.CharField(max_length=100)
    body = models.TextField();
    time = models.DateTimeField();
```

```
    def __unitcode__(self):
        return user + "␣says:␣" + body
```