CMPE-202 - Team 1 :Class Project

Authors

Sanjeevi Vijaykumar, Chaitanya Lala, Venkata Satya Phani Pavan Chitturi kumar, Partha Ghosh

# 

[**Introduction**](#_yr0v8wr4y4j0) **2**

[**System Architecture**](#_4f2ozjsppsjp) **3**

[**Project Definition**](#_rpoevkpif9sl) **3**

[Assumptions](#_3e5zskdzb0gk) 3

[**Security Baseline**](#_vlnhbjjqcbqo) **3**

[Research & Methodology](#_gdrye5xz6n5g) 4

[Which Stage Of Project this was Done ?](#_qjs9dx5hd51a) 4

[Who were Involved](#_k0lw8rylc33e) 5

[**Threat Modelling**](#_3fhi1ghkejp8) **5**

[Research And Methodology](#_jfeiuvwil0zj) 5

[**Source Code Static Analysis**](#_eq6vnen4zcym) **6**

[Requirements](#_jbbygyaohusm) 6

[Research](#_hv5hn3y3883y) 7

[Methodology](#_yrfzfu2h8zbi) 7

[Software Design](#_xkqmbu492l0g) 8

[The design is modular. New functionalities could be developed as the project progresses.](#_m14n74snqirf) 8

[Compilation](#_4xrpoqf7d16y) 8

[Static Analysis Results](#_2m25m25c2n5w) 8

[Sample Static Analysis output](#_ic4oh93gjdgx) 9

[Code Coverage Results](#_7nwc0sijyb2e) 9

[Code Review](#_295cqeee0f41) 9

[Source Code](#_yth0yws7yi93) 10

[**Fuzzing**](#_8wc0c24k9k3d) **10**

[Research](#_xri1kib7mss0) 10

[Methodology](#_28rw3232u1kg) 10

[Which Stage Of Project this was Done ?](#_wfu3hm94cu9x) 13

[Fuzzing Analysis](#_60v2edwnnlip) 13

[All the artifacts ar](#_3xzbqtrl2aza) 13

[Who were Involved](#_yek17ngdk677) 13

[**Third Party Software Tracking**](#_8p9vv8kbuz3k) **13**

[Methodology](#_z431kfoge47h) 14

[Which Stage Of Project this was Done ?](#_qaox2x45wkmo) 14

[Who were Involved](#_qlmzkti5400y) 14

[**Answers to The Question in the Problem**](#_924fuuu3ad6z) **14**

[1. Custom Sensor/Controller Protocol](#_h3ewjdsbmg) 14

[2. SDL Activities](#_dg38i2gusyx9) 16

[3. SDL Metrics](#_hyqxbnl634g7) 17

[Static Analysis Metrics](#_f8mdywokiqsx) 18

[4. Changes to the Solution](#_1fdraa2210bl) 18

[A. Softcorp decides to add more fields in the UI page of the configuration servlet. Which SDL activities would this trigger?](#_s6iqpto1e0u9) 19

[B. Softcorp now plans to add a C++ process to its appliance. The process would use the sensors data in the Oracle database and send SNMP traps if some concerning trend is identified in the data set. Which SDL activities would be triggered in this case?](#_n1a27tofcerc) 19

[**Conclusion**](#_ayep7sgf0v5a) **19**

[**Addendum**](#_f6al1xo3750g) **19**

[Artifacts List](#_pjba77j7w9t3) 20

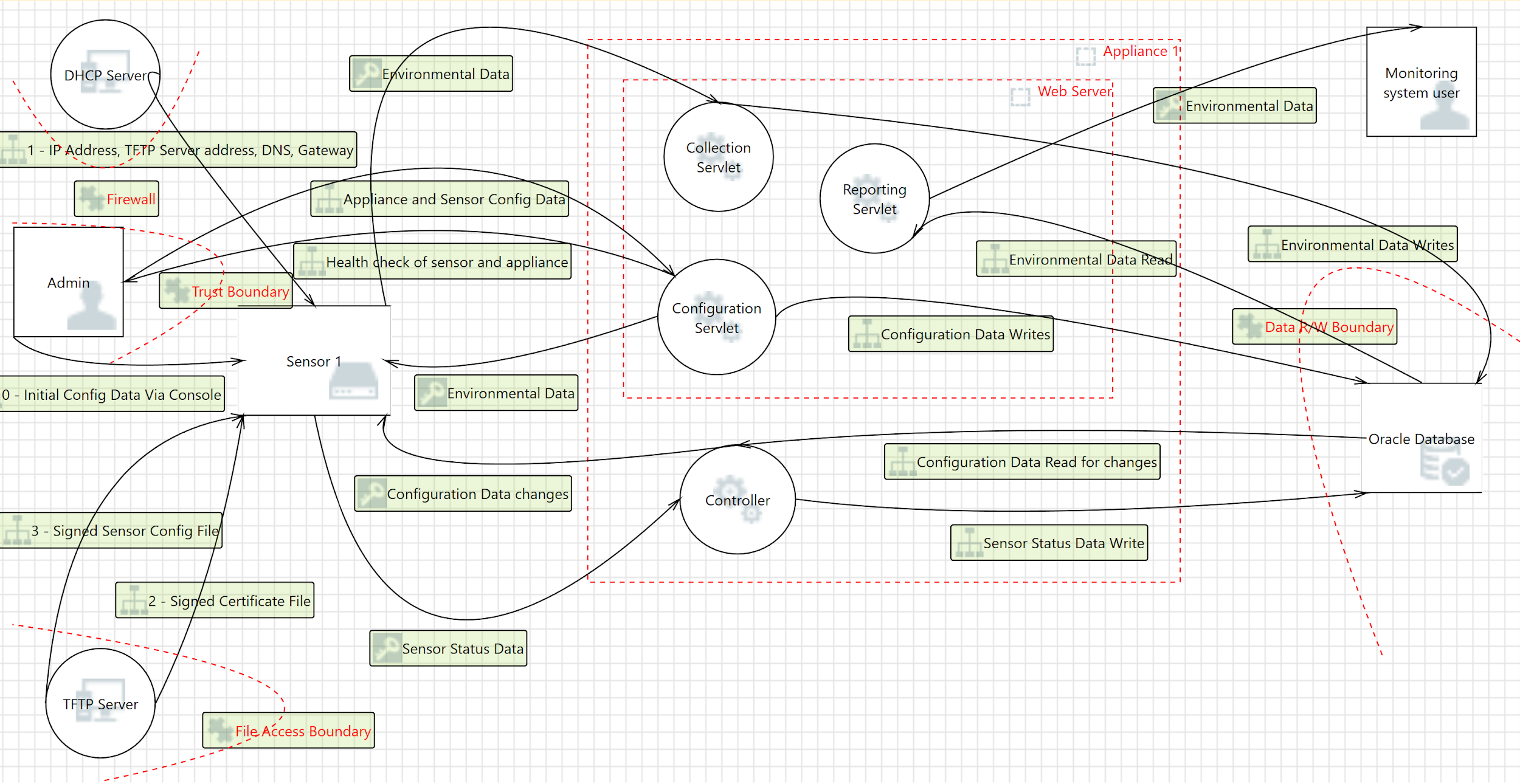
[References](#_d2yafgep4aql) 21

# 

# Introduction

[Sanjeevi]

# System Architecture



[sanjeevi] itemize the entities

# Project Definition

The Project Definition is detailed in the CMPE-202 Class Project Document. The Document is part of the [artifacts list](#_pjba77j7w9t3).

## Assumptions

* The TFTP server is component which has no security protections in place and hence it is assumed to be in a secure private network. In-fact, we assume that the sensor has a private subnet over a distinct network interface which cannot communicate with the controller side of the network ( due to internal firewalling ).
* The sensor’s file system is not-accessible via console and other users without having a local super user privileges otherwise the SCF cert file and settings can be modified without notice.

# Security Baseline

HIgh level Definition of the CSM Control Domain : [pavan]TBD

Definitions of Physical, Network, Compute : [pavan]TBD

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| NO | Component | Physical | Network | Compute | Storage | App | Data |
| 1 | Sensor | **X** | **X** | **X** | **X** | **X** | **X** |
| 2 | TFTP server | **X** | **X** | **X** | **X** | **X** | **X** |
| 3 | DHCP Server | **X** | **X** | **X** | **X** | **X** | **X** |
| 4 | Appliance | **X** | **X** | **X** | **X** |  | **X** |
| 5 | Controller |  | **X** | **X** | **X** | **X** | **X** |
| 6 | Web Server |  | **X** | **X** | **X** | **X** | **X** |
| 7 | C++ Process for Analytics |  | **X** | **X** | **X** | **X** | **X** |
| 8 | SNMP Trap Receiver | **X** | **X** | **X** | **X** | **X** | **X** |
| 9 | Oracle Database | **X** | **X** | **X** | **X** | **X** | **X** |
| 10 | Admin Console | **X** | **X** | **X** |  | **X** | **X** |
| 11 | Monitoring System User | **X** | **X** | **X** |  | **X** | **X** |

### Research & Methodology

The purpose of the security baseline is to create a security baseline which the product will adhere to. We need to perform Gap analysis based on the security requirement of the project.

### We evaluated “*NIST Baseline tailor*” and *Cloud Control matrix* from Cloud Security Alliance. We used *Cloud Control Matrix* in this project.

The Baseline matrix is part of Software Artifacts submitted along with project report.

### Which Stage Of Project this was Done ?

This was done in the planning stage of the project.

### Who were Involved

We played the role of Product Management and System Architect to come up with the security Baseline.

# Threat Modelling

### Research And Methodology

* **Summary - Think Ahead what can go wrong, Weigh the Risks and Act accordingly**
* Approach - Design Centric / Software Centric
* Tools used : Microsoft Threat Modeling Tool
* Output of threat modelling - Identify security requirements and Reduce attack surface
* Purpose of threat Modelling
  + Define Trust boundaries
  + Highlight Areas of Specific security focus
    - E.g password missing, access control enablement
  + Provide Endpoint security - IOT Devices
* Threat Classification via **STRIDE** model
* Other Methods for threat Model - Kill chains, Attack Trees, OCTAVE etc
* **Secure Design Principles**
  + Defense in depth - perimeter security, Layered Security (HTTPS and Certificates)
  + Least Common Mechanism - Three Java servlets and C++ controller
  + Least Privilege - Read only access to Reporting Servlet

STRIDE-based threat modeling methodology

* Decompose System into Components
* Plot DFD for System Components
* Analyze Threats in the DFD
* Identify Vulnerabilities
* Plan Mitigation Strategies

STRIDE Threats provide opposite properties

**THREAT DESIRABLE PROPERTY**

Spoofing -> Authentication

Tampering -> Integrity

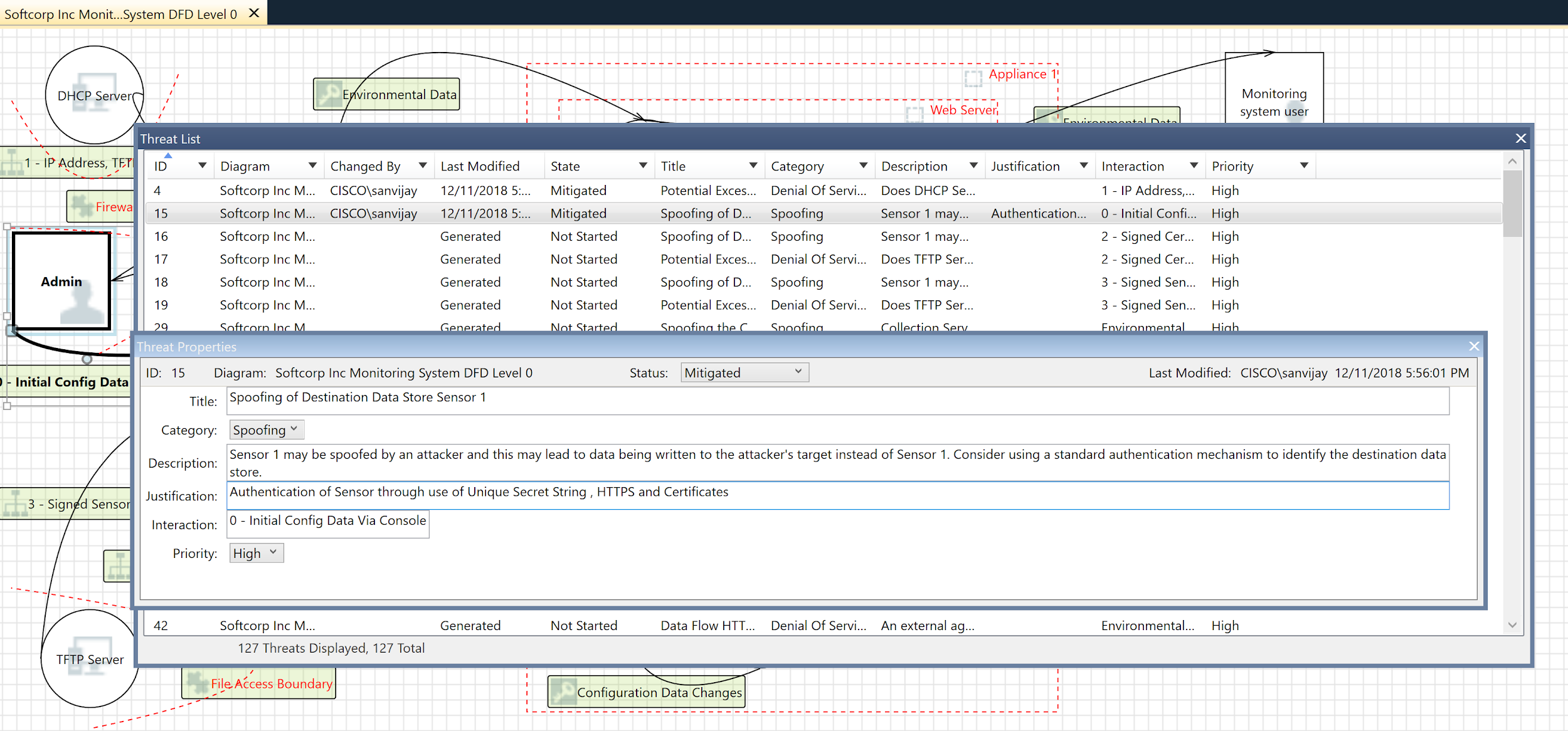
Repudiation -> Non-Repudiation

Information -> Disclosure Confidentiality

Denial of Service -> Availability

Elevation of Privilege -> Authorization

Threats and Mitigation for Softcorp Inc Monitoring system



Threat Modelling Report -> <https://drive.google.com/open?id=1YY14ILFXAzbUYWhMorG2nOR0QaKwpJ4I>

# Source Code Static Analysis

### Requirements

The Source code analysis is part of the SDL. Here are the requirement for the Parser.

* Parsing program
  + Parse an SCF file
  + Parse all the certificates
  + Print the CN Names

The Goal was to use the parser to demonstrate Secure Development lifecycle activities. That included

* Code Repository
  + Ability to phase code development based on Milestone.
* Compiler
* Static Analysis
  + Choose a tool for Static analysis
  + Develop Metrics for Static Analysis
  + Perform Static Analysis
  + Fix the issues found in static Analysis
  + Perform Code Review
  + Perform Code Coverage
* Dynamic Analysis

### Research

The choice of the programming language is c. This determined the choice of the tools. The other consideration was to use free software which are of “excellent reputation”.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Item | Choices or tools | What did we use | Notes |
| 1 | Code Repository | svn or git or Clearcase or cvs | **git** | * Multiple free online repository as long as keep the code as open source. * GIt is the defacto code repository for most s/w organisation |
| 2 | Compiler | gcc or clang (front end for LLVM for c) | **clang** | * Clang is used as the llvm tools could be leveraged for static analysis, code coverage |
| 3 | Static Analysis | Scan-build or coverity | **scan-build** | * Free , open source, can create user defined recipe for static analysis |
| 4 | Code Coverage | Coverity, llvm-coverage | **llvm-coverage** | * Free , open source, fits well with other tools for the SDL |
| 5 | Code Review | Crucible, git issues | **Git issues** | * Used the online tool for code review and filing issues. Integrated with git |
| 6 | Dynamic Analysis | Valgrind | **valgrind** |  |

### Methodology

The activity started with

1. Designing the parser
2. Phased development of the parser
3. Started to perform static analysis before every commit
4. After code completion,
   1. code coverage
   2. code review
5. Once the code was complete dynamic analysis was done

#### Software Design



### The design is modular. New functionalities could be developed as the project progresses.

#### Compilation

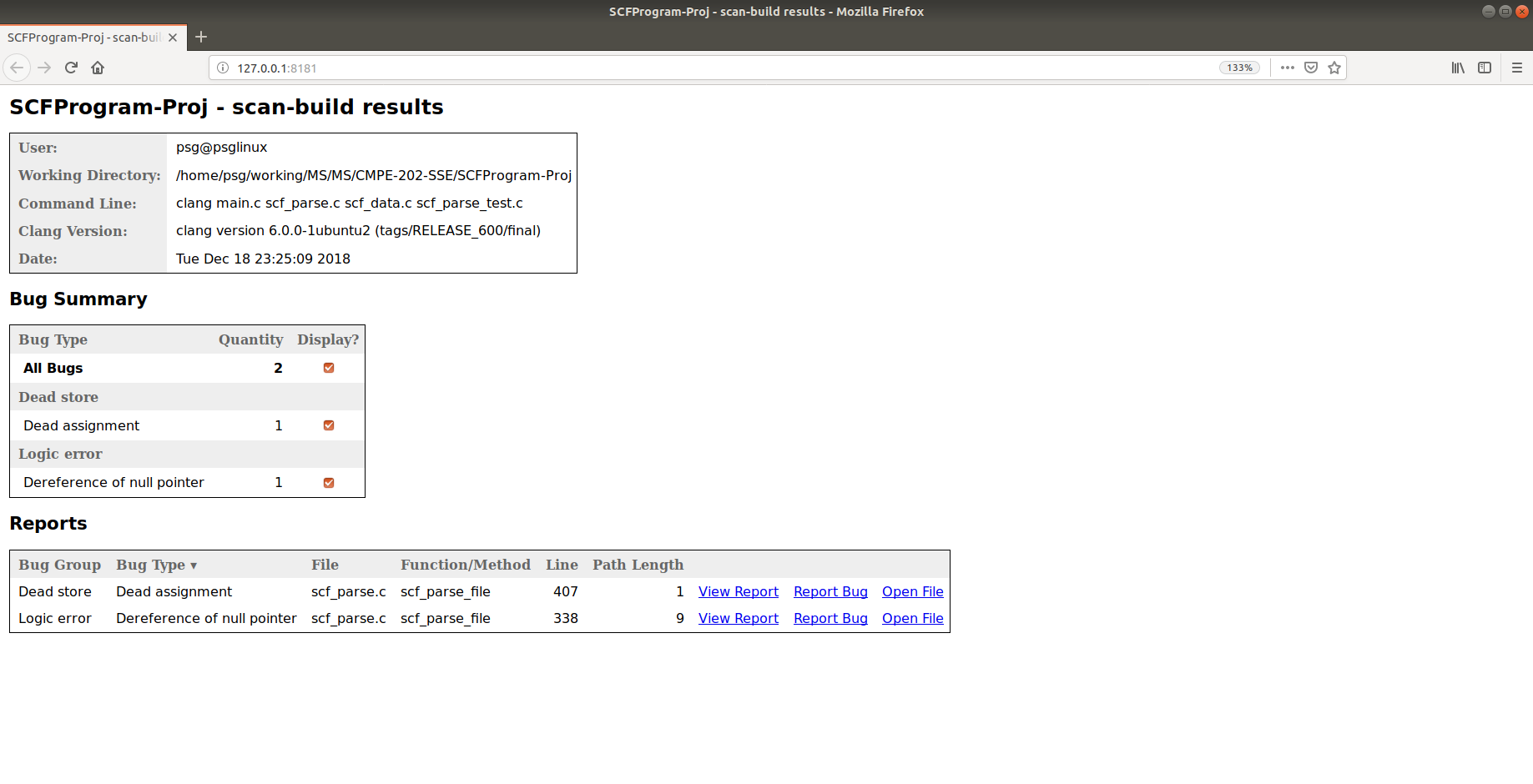
A Makefile was written which would automate the process of build, static analysis and code coverage.

#### Static Analysis Results

Static Analysis was done during different stages of development. The logs of static analysis are provided as artifacts. Here is a summary of the static analysis results.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Build** | **Date** | **Total LInes of Code** | **Total Number of Defects** | **Nos of Defects per 1000 lOC** | **showstopper** | **severe** | **moderate** | **low** |
| 1 | stage-1 | 12/05/2018 | 716 | 6 | 8.379888268 | 0 | 2 |  | 4 |
| 2 | stage-2 | 12/06/2018 | 805 | 8 | 9.937888199 | 0 | 3 | 1 | 4 |
| 3 | stage-3 | 12/07/2018 | 1033 | 6 | 5.808325266 | 0 | 2 |  | 4 |
| 4 | stage-4 | 12/07/2018 | 1034 | 2 | 1.934235977 | 0 | 1 |  | 1 |

#### Sample Static Analysis output



The software artifacts contains the static analysis results done in different phases.

#### Code Coverage Results

Filename Regions Missed Regions Cover Functions Missed Functions Executed Lines Missed Lines Cover

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

main.c 22 13 40.91% 1 0 100.00% 77 46 40.26%

scf\_data.c 65 21 67.69% 12 5 58.33% 133 44 66.92%

scf\_parse.c 151 36 76.16% 12 2 83.33% 251 29 88.45%

scf\_parse\_test.c 9 9 0.00% 3 3 0.00% 18 18 0.00%

Files which contain no functions:

scf\_parse.h 0 0 - 0 0 - 0 0 -

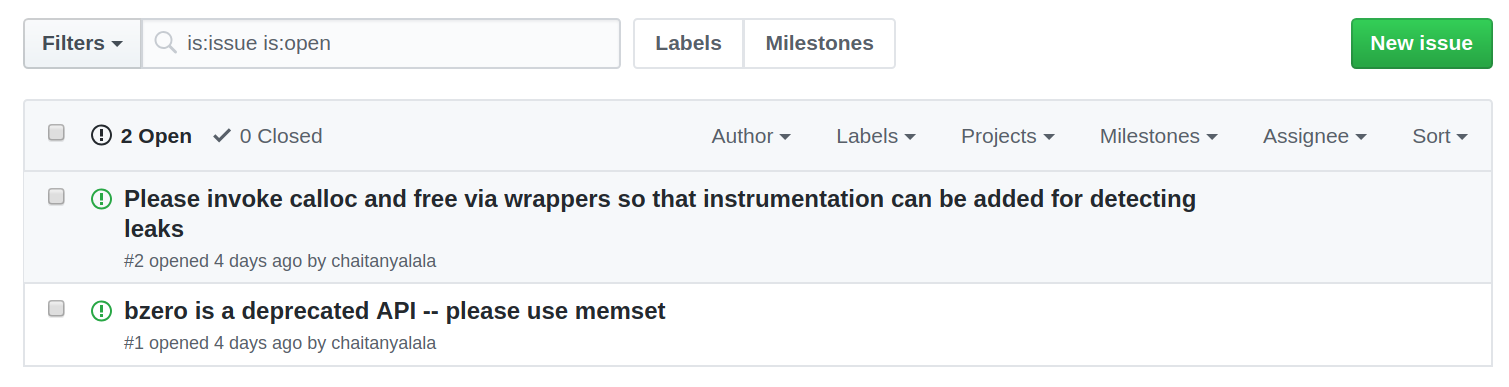
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

TOTAL 247 79 68.02% 28 10 64.29% 479 137 71.40%

The code coverage report also points to the code that is not being used in detail. The output is present in the Static Analysis artifacts.

#### Code Review

The code review comments could be found in <https://github.com/psglinux/MS/issues>



Dynamic Analysis

Dynamic analysis was performed in the code to observe run time behavior of the application. The results could be found in <https://github.com/psglinux/MS/issues/3>

#### Source Code

The source code location is <https://github.com/psglinux/MS/tree/master/CMPE-202-SSE/SCFProgram-Proj>

# Fuzzing

*Fuzz testing* or *Fuzzing* is a Black Box software testing technique

Finding implementation bugs using malformed/semi-malformed data injection

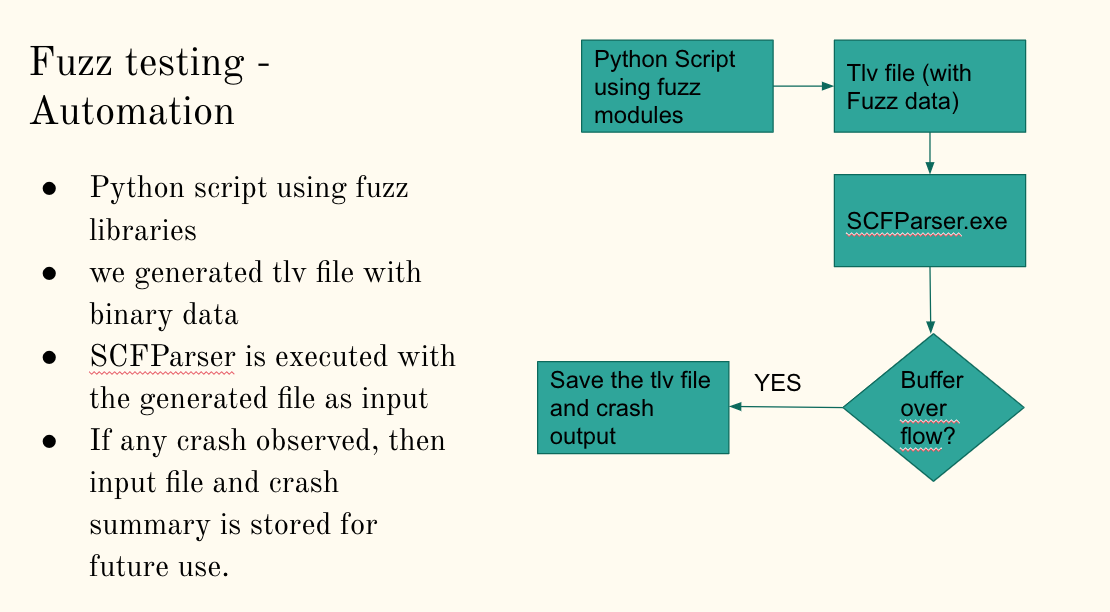
### Research

Discussed on which programming language to chose to automate fuzzing. Decided to use python language. Investigated various fuzz libraries in python.

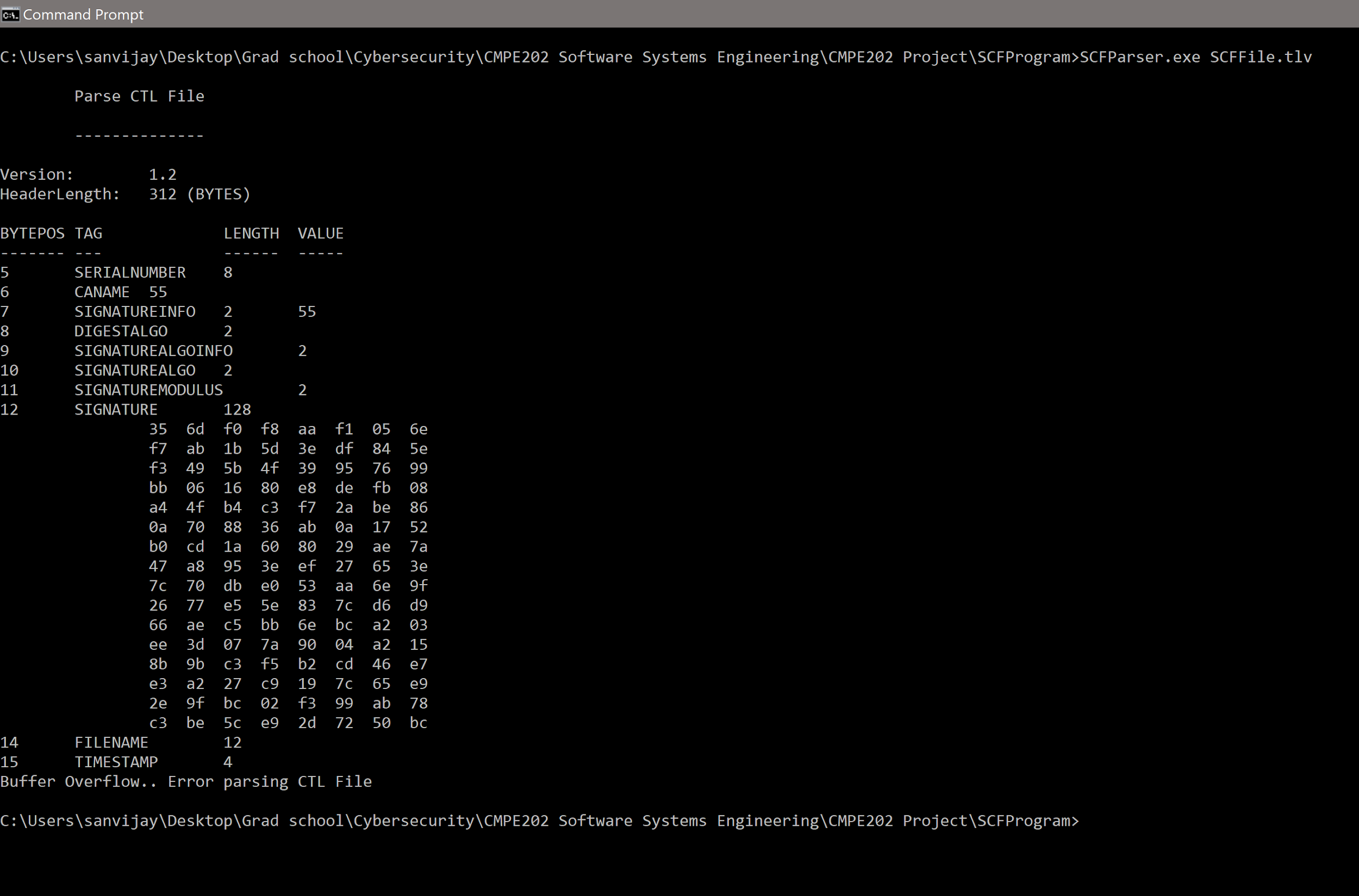
### Methodology

The methodology used is

1. Manual
2. Automated



In manual fuzzing mode , we edited tlv file using hex editor and added random strings. This way we can target on what fields to edit and randomize the input.

Screen shot of the buffer overflow of SCFParser.exe is given below.

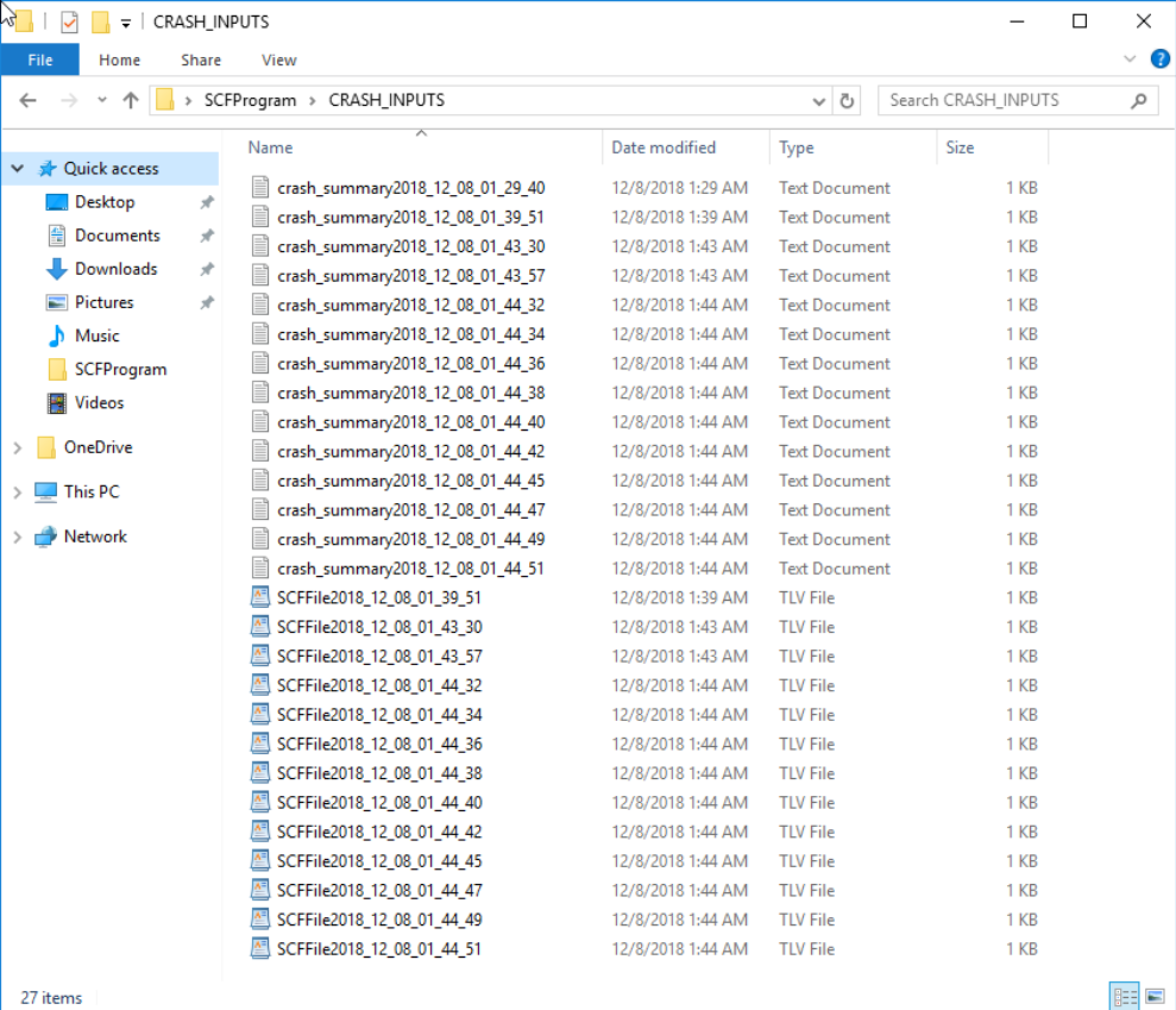
### 

### Which Stage Of Project this was Done ?

Fuzzing is normally done once the code was completed. But since for this project Application was already provided, we did it independently.

### Fuzzing Analysis results:

Crash results and test inputs were saved in crash results.



### Who were Involved

We played the role of Test Engineers for the fuzzing part of the project.

# Third Party Software Tracking

There are different components in the projects which uses 3rd party tools and libraries. The primary security consideration for the 3rd party software are

* Selection
* Acquisition
* Hardening
* Inventory
* Security patching

### Methodology

Initially we used an excel sheet to create the 3rd party dependency matrix. However tracking patches , latest vulnerabilities becomes immensely complex using a excel sheet. So we used **Dependency Track** (A software composition analysis platform) to perform the third party software tracking. This Tool is helpful in automated discovery of vulnerabilities, keep track of new software version and develop a good patching strategy.

### Which Stage Of Project this was Done ?

Third party software tracking was is done during the planning phase of the project, during sustenance and if new vulnerabilities are discovered.

### Who were Involved

We played the role of Technical Lead for the 3rd Party Software tracking.

The 3rd party software artifacts are delivered in the 3rd part folder with the other s/w artifacts.

# Answers to The Question in the Problem

## 1. Custom Sensor/Controller Protocol

You are tasked with designing the custom sensor/controller protocol. Explain in at least a

half page the steps you would take to design a secure protocol.

The Sensor(s) communicates with the controller over a permanent TLS/TCP connection so basic encryption is ensured. But the sensor assumes that it is talking to the actual controller without actually authenticating the controller.

One basic addition to the protocol should be to authenticate the controller via a challenge-response model where the sensor, before actually sending a registration request, sends a encrypted challenge message. The challenge should be encrypted using a key generated from the shared string using a cryptographically secure hash function (for example, SHA256 should be run on the shared string and the output should be used as secret key). In addition to the challenge a sensor ID in clear-text should be sent. The controller on reception of the challenge should look at the sensor ID & using that find the shared secret string from it’s database. It should then perform the required transformation using the same cryptographically secure hash function to generate the shared key, which would then used to decrypt the challenge. The controller then responds to the sensor with the response message in clear-text. This would authenticate the controller with the sensor.

An additional aspect of the protocol is the integrity of the message. The sender and receiver both could attach a HMAC digest (using the previously generated secret key) for each message send it along with the message.



Notes: This protocol of using a challenge response model based on the secret shared string is immune to unauthorized changes done to the default configuration file sitting on the TFTP server. This is because, the secret shared string is keyed in via the console by the user. Hence, unless an attacker modifies both the TFTP server file \_and\_ the shared string entered via the sensor console, the challenge response authentication with a rouge controller server will fail.

A second mechanism to protect the sensors is akin to Multi-factor-Authentication (MFA). If the TFTP file can be modified by an authorized user, then it makes sense that the user who is configuring the sensors, keys in the expected checksum of the config file which is to be downloaded. The sensor will discard that file if the checksums do NOT match.

How to update SCF file without manual intervention ?

As one of the requirements of the system is to use TFTP server, which cannot use secure communication, the next best approach is to secure the configuration artifacts. Assume that the admin wants to add a new service on the controller, then the controller should send a message to the sensor to use a new service with all parameters needed for the sensor to \_trust\_ the new service/controller. This way a full chain of trust is maintained without needed any manual changes.

## 2. SDL Activities

Can you think of any other SDL activities that would be needed for this project beside the

ones covered in this project? Explain why you think they are needed.

For this project we have performed

* Security Baseline
* Threat Modeling
  + Once at the beginning of the project
  + Once for again for problem 4 when a new process was required
* Design & Development
* Static analysis
* Dynamic Analysis
* Code review
* Fuzzing
* Third Party Software Tracking

The following activities could also be done for this project

* SDL Discovery
  + Estimate the actual cost of the project
  + FInd out the security Requirements
* Security Training and Awareness
* Security Design review
* Vulnerability Scanning
* Security Testing
* Penetration testing
* Fault injection
* Gap Analysis

## 3. SDL Metrics

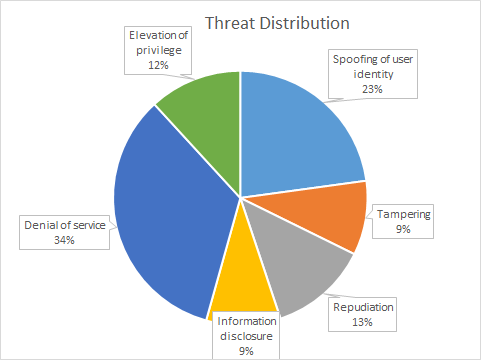
Meaningful security metrics allow organization to determine the effectiveness of its

security controls. List the SDL metrics that you would capture in this project and explain

how they would help bring maturity to your SDL process.

[Pavan + Sanjeevi]

**Threat Modelling Metrics**





[Threat Modelling Metrics](https://drive.google.com/open?id=1KfAkP9WvSbckuXWZtDGZDInM7iY4hIce)

#### Static Analysis Metrics

The following software metrics could be used for Static Analysis apart from the ones mentioned in the static analysis metrics of nos of defects per kloc (kilo lines of code).

Other Metrics that are important

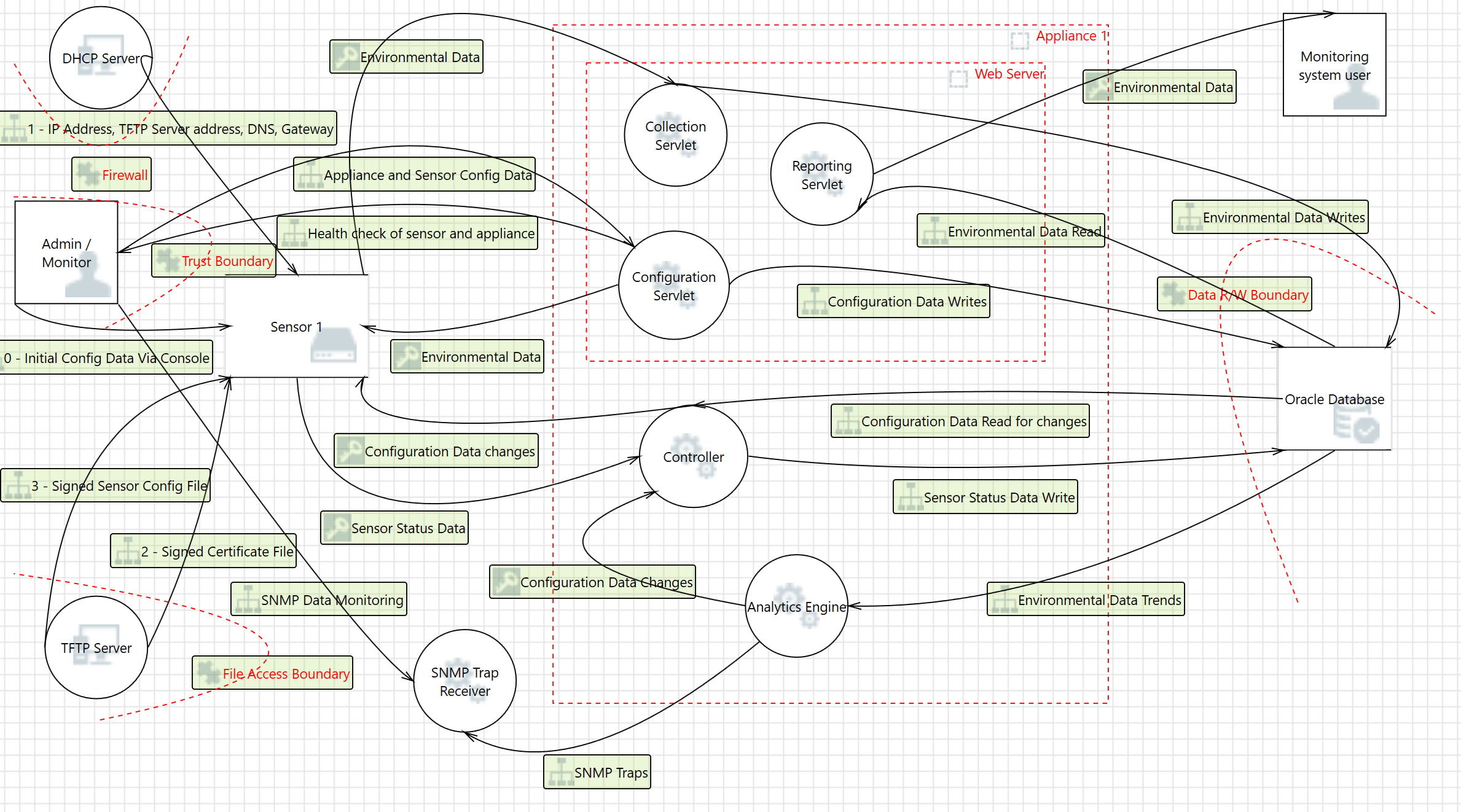
* Time taken to fix the warnings
* Classification of the warnings, like security issues, LIbrary issues
* Code review comments and categorizations
  + coding guideline violations
  + Denial of service errors
  + Logical Error
  + input validations
* Nos of defects found in the dynamic analysis
* Nos of defects found during reviews

## 4. Changes to the Solution

### A. Softcorp decides to add more fields in the UI page of the configuration servlet. Which SDL activities would this trigger?

### B. Softcorp now plans to add a C++ process to its appliance. The process would use the sensors data in the Oracle database and send SNMP traps if some concerning trend is identified in the data set. Which SDL activities would be triggered in this case?

**Threat Modelling changes after incorporating the new requirements from Softcorp**



# Conclusion

This project help us learn system Software engineering from a security perspective. Our learning for the activity has been summarized below.

* How to Analyse Distributed System Architecture, from a Security perspective
* How to map system requirement to threat model, like STRIDE
* Break a system using Fuzzing.
* Static Analysis should be fundamental component of every development cycle.
* Practical usage of 4 pillars of Security (Quality S/W, Secure Design Principle, Knowledge Security Technology and Secure Development Lifecycle)

# Addendum

### Artifacts List

|  |  |  |  |
| --- | --- | --- | --- |
| NO | SDL Activity | Artifact File Name | Description |
| 1 | Threat Modelling | Softcorp Inc Monitoring System TM.tm7 | Level 0 Threat Model for Soft Corm Monitoring System |
| 2 | Threat Modelling | After changes Softcorp Inc Monitoring System TM.tm7 | Level 0 Threat Model with the changes needed for Question 4a,4b |
| 3 | Threat Modelling Report | latest TM report.pdf | Threat Modelling Report |
| 4 | Fuzzing | SCFProgram-Fuzzing-code.zip | Present in the zip files |
| 5 | Static Analysis Source Code | Source Code <https://github.com/psglinux/MS/tree/master/CMPE-202-SSE/SCFProgram-Proj> | The source code is present in the github. README file has the instructions |
| 6 | Static Analysis Artifacts | <https://github.com/psglinux/MS/blob/master/CMPE-202-SSE/SCFProgram-Proj/StaticAnalysisArtifacts.tgz> | This is a Tar file containing Static. This contains the static analysis done during the course of development. To create a current static analysis run ‘**make sa**’ on the code |
| 7 | Code Coverage | <https://github.com/psglinux/MS/blob/master/CMPE-202-SSE/SCFProgram-Proj/scf-code-coverage.report>  <https://github.com/psglinux/MS/blob/master/CMPE-202-SSE/SCFProgram-Proj/scf-code-coverage.view> | Code coverage reports are present in the github. Code coverage can be generated running ‘**make coverage**’ on the code. |
| 8 | 3rd Party Software Tracking | Manual 3rd Party Tracking 3rd-Party Software Tracking  ThirdPartyTrackingScreenShot.png | Used the tool **DependencyTrack** to monitor 3rd party software trackingP |
| 9 | Dynamic Analysis | <https://github.com/psglinux/MS/issues/3> | Dynamic Analysis was performed and the artifact was attached in the Static SOurce Code Analysis |
| 10 | Security Baseline | CSA\_CCM\_v.3.0.1-09-01-2017\_FINAL.xlsx |  |

### References

* Source Code Analysis:
  + Compiler (https://clang.llvm.org/docs/UsersManual.html#introduction)
  + Static analysis (https://clang-analyzer.llvm.org/)
  + Code coverage (<https://clang.llvm.org/docs/SourceBasedCodeCoverage.html>)
  + Glibc (https://www.gnu.org/software/libc/)
* Fuzzing:
  + <https://www.owasp.org/index.php/Fuzzing>
  + https://github.com/jmcph4/fuzzbang
* Threat Modelling
  + https://www.owasp.org/index.php/Category:Threat\_Modeling
  + Threat Modeling : Designing for Security - Adam Shostack
  + https://en.wikipedia.org/wiki/STRIDE\_(security)
  + <https://www.us-cert.gov/bsi/sdlc/design>
* Dependency Tracking
  + <https://dependencytrack.org/>
* Security Base line
  + <https://cloudsecurityalliance.org/working-groups/cloud-controls-matrix/#_overview>