UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Forensic Box for Quick Network-Based Security Assessments

João Bernardo Ferreira Sequeiros

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Ricardo Morais Inácio

Covilhã, outubro de 2016

Acknowledgements

Approaching the end of this chapter of my academic life, I would like to express my gratitude to some very special people that made it all possible.

I start with my parents for all their unconditional support, love and care. For always being there when needed, for every word of encouragement, and for all the insight and knowledge they impart on me on a daily basis.

No less important, I thank my supervisor, Professor Doutor Pedro Inácio, for all the support given, and for giving me the opportunity to work with him, share ideas and to learn under his tutelage.

I am also particularly grateful to some special colleagues from the lab and from next door, namely to Acácio Correia, Manuel Meruje, João Neves, Musa Samaila and Pedro Tavares. A special thank you goes to my *borrowed* little sister, Patrícia.

Last but not least, to some very special friends. To João Pais, Nuno Carapito, José Ribeiro, Mafalda Baptista, Miguel Fernandes and Guilherme Fernandes (sorry if I forgot someone!). Their support has been paramount and duly appreciated over these last few years.

Resumo

As avaliações de segurança de uma rede (e dos seus dispositivos) são vistas como tarefas importantes, mas pesadas e que consomem bastante tempo, devido à utilização de diferentes ferramentas manuais. Normalmente, estas ferramentas são bastante especializadas e exigem conhecimento prévio e habituação, e muitas vezes a necessidade de criar um ambiente de teste. No entanto, em muitos casos, seria útil obter uma auditoria rápida e de forma mais direta, ainda que pouco profunda. Nesses moldes, poderia servir como passo inicial para uma avaliação mais detalhada, complementar outra auditoria, ou ainda ajudar a prevenir fugas de dados e falhas de sistemas devido a problemas comuns de configuração, gestão ou implementação dos sistemas.

Esta dissertação descreve o trabalho efetuado com o objetivo de desenhar e desenvolver um sistema portátil para avaliações de segurança de uma rede de forma rápida, e também a investigação efetuada com vista à automação de várias tarefas (e ferramentas associadas) que compõem o processo de auditoria. Uma concretização do sistema foi criada utilizando um Raspberry Pi 2, várias ferramentas conhecidas e de código aberto, cujas funcionalidades variam entre descoberta da rede, identificação de sistema operativo, descoberta de vulnerabilidades a captura de tráfego na rede, e *scripts* e programas personalizados que interligam as várias partes que compõem o sistema. As ferramentas são integradas de forma transparente no sistema, que permite ser lançado em ambientes cablados ou *wireless*, onde o dispositivo executa uma análise meticulosa e maioritariamente automatizada. O dispositivo é praticamente *plug and play* e produz um relatório estruturado no final da avaliação. Várias funções simples, tais como analisar novamente a rede ou efetuar ataques de envenenamento da *cache Address Resolution Protocol* (ARP) na rede estão disponíveis através de um pequeno ecrã LCD montado no topo do dispositivo. Este oferece ainda uma interface *web*, também desenvolvida no contexto do trabalho, para configuração mais específica das várias ferramentas e para obter acesso ao relatório da avaliação. Outros *outputs* mais específicos, como ficheiros com tráfego capturado, estão disponíveis a partir desta interface.

O sistema foi utilizado em redes controladas e reais, de forma a verificar a qualidade das suas avaliações. Os resultados obtidos foram comparados com aqueles obtidos através de auditoria manual efetuada às mesmas redes. Os resultados obtidos mostraram que o dispositivo deteta a maioria dos problemas que um auditor detetou manualmente, mas mostrou algumas falhas na deteção de algumas vulnerabilidades específicas, maioritariamente injeções *Structured Query Language* (SQL).

A imagem do Sistema Operativo com as ferramentas pré-configuradas, *scripts* de automação e programas está disponível para download de [Ber16b]. Esta imagem corresponde a um dos principais resultados deste trabalho.

Palavras-chave

Análise Forense, Auditoria de Segurança, Automação de Ferramentas e Processos, Avaliação de Segurança da Rede, Deteção de Vulnerabilidades, Segurança de Sistemas, Testes de Penetração

Resumo alargado

Este resumo alargado tem como objetivos apresentar, na Língua Portuguesa, o conteúdo desta dissertação de uma forma um pouco mais detalhada que a secção anterior. A maior parte desta dissertação está escrita na Língua Inglesa.

Introdução

O primeiro capítulo tem como objetivo enquadrar o trabalho descrito ao longo desta dissertação, introduzindo o tema geral, e apresentando a motivação e enquadramento em que este se insere. É também neste capítulo que é apresentado o problema que este trabalho se propõe a resolver, e os seus objetivos gerais. Neste capítulo é ainda apresentada a abordagem tomada para resolver o problema proposto, e as principais contribuições do trabalho.

Enquadramento, Descrição do Problema e Objetivos

Hoje em dia, a maioria das organizações são suportadas for infraestruturas de rede tecnologicamente complexas, que se apresentam como potenciais pontos críticos de falha. Devido a estas falhas, todos os anos, dados de instituições e de clientes estão expostos a ataques. Consequentemente, as auditorias de segurança a uma rede são uma prática comum e uma ferramenta para garantir um maior nível de segurança, e manter dados e serviços melhor protegidos de ameaças, quer internas quer externas. Estas auditorias, no entanto, são de complexidade elevada, visto requererem a utilização de diferentes ferramentas, e conhecimentos avançados e técnicos a nível da sua utilização, assim como de guias e metodologias a seguir de maneira a efetuá-las da melhor forma possível.

O objetivo deste trabalho passa então pelo desenvolvimento de um dispositivo compacto e de baixo custo que permita efetuar auditorias de segurança a uma rede de forma rápida. Isto inclui desenvolvimento e implementação das regras de negócio que permitem ao dispositivo efetuar diferentes tarefas de uma forma automática, transparente e simples para o utilizador. O sistema deve detetar não só problemas no funcionamento da rede, mas também auditar o estado dos sistemas e redes identificados. Deve ser ainda possível ligar o dispositivo rapidamente tanto numa rede cablada como sem fios. Para a maior parte das funcionalidades, o utilizador deve apenas necessitar de ligar o dispositivo a um cabo *Ethernet* ou colocá-lo no alcance de uma rede sem fios, e ligar o dispositivo. Como principais funcionalidades, o dispositivo deve ser capaz de capturar e analisar tráfego, descobrir sistemas numa rede, detetar vulnerabilidades e reportar os resultados.

Principais Contribuições

Esta secção apresenta as principais contribuições científicas resultantes do trabalho desenvolvido e apresentado aqui. As principais contribuições podem ser então descritas da seguinte forma:

- A primeira contribuição deste trabalho é o estudo efetuado acerca de análise e auditoria de segurança numa rede, dos diferentes trabalhos científicos existentes na área e principais metodologias e guias a seguir. Este estudo compreende diferentes ferramentas existentes e comummente utilizadas em auditorias, assim como os procedimentos seguidos durante uma auditoria, e as melhores práticas na sua realização. Foram também analisadas várias propostas de *frameworks* ou infraestruturas integradas de análise de segurança;
- A segunda contribuição é a definição das especificações de *hardware* e requisitos de funcionamento do sistema, assim como as ferramentas selecionadas e automatizadas e toda a lógica desenvolvida de forma a permitir que o dispositivo efetue auditorias de segurança em rede de forma automática. O desenvolvimento e definição do sistema materializou-se numa comunicação, intitulada *Forensic Box for Quick Network-Based Security Assessments*, publicada nos procedimentos da INForum 2016, que se realizou em Lisboa, Portugal, entre os dias 8 e 9 de Setembro de 2016;
- Outra contribuição do trabalho é o sistema em si, que pode ser utilizado por qualquer administrador de sistemas ou redes de forma a efetuar auditorias rápidas à segurança da sua rede. Este dispositivo foi apresentado e o seu funcionamento demonstrado na Techdays Aveiro, um fórum de tecnologia que decorreu entre os dias 15 a 17 de Setembro de 2016 em Aveiro, Portugal.

Trabalho Relacionado e Tecnologia

O capítulo 2 explora as ferramentas de auditoria existentes, juntamente com metodologias e guias gerais de como proceder na realização de uma auditoria. É também apresentada uma análise de vários trabalhos científicos de sistemas de análise e auditoria de segurança de uma rede.

Destacam-se alguns trabalhos acerca de análise de segurança em redes, tanto a nível da criação de *frameworks* que permitem analisar o estado de segurança numa rede, como a CNSSA [XJYZ11], ou a descrita por Hallberg *et. al.* [HHP05], como a nível de ferramentas a utilizar em análises de segurança [CA06], ou políticas de segurança a tomar e os passos que as constituem, incluindo análise e testes [Cis03].

A análise de várias ferramentas apresenta alguns dos softwares mais comummente utilizados nesta área, incluindo *suites* completas e software profissional. São descritos softwares populares como o Nessus ou o Snort, de forma a dar um melhor entendimento em relação às opções existentes e as suas vantagens e desvantagens.

Este capítulo também apresenta as linhas gerais que definem uma auditoria de segurança, sendo que a definição apresentada é o fio condutor da especificação e desenvolvimento do dispositivo que este trabalho descreve. Esta definição é maioritariamente baseada no guia proposto pelo *National Institute for Standards and Technology* (NIST). Mencionam-se as principais recomendações de uma auditoria de segurança e as algumas técnicas e tipos de testes efetuados.

Especificações de *Hardware* e *Blueprinting*

O capítulo 3 apresenta as especificações de hardware definidas para o dispositivo a ser utilizado, assim como a definição das principais funcionalidades a serem disponibilizadas, interações entre as diferentes partes e módulos que o dispositivo deve integrar.

Tendo em conta as definições abordadas no estudo efetuado sobre trabalhos relacionados e aquilo que compõe uma auditoria de segurança na sua essência, bem como o que é proposto como objetivo final deste trabalho, podem-se definir diferentes requisitos de forma a selecionar-se o *hardware* a ser utilizado no desenvolvimento do dispositivo, nomeadamente: necessidade de portabilidade; eficiência energética; alimentação através de uma bateria; ligação à rede tanto cablada como sem fios; compatibilidade com ferramentas de segurança conhecidas e, devido a isto, de correr um sistema operativo de tempo real, como um baseado em UNIX; possibilidade de o utilizador interagir fisicamente com o dispositivo e de receber informação básica direta a partir do mesmo; capacidade de alojar uma interface *web*; e ainda ter um custo reduzido. Destas especificações, foi escolhido um Raspberry Pi 2, que à data do início do trabalho era a versão mais recente e poderosa do Raspberry Pi. Para complementar os restantes requisitos, foi adicionado um módulo *wi-fi* e um módulo com botões físicos e um pequeno ecrã montados no topo do dispositivo.

A parte seguinte passou pela definição das funcionalidades de auditoria que o dispositivo deve apresentar, a saber: ligação à rede; descoberta de máquinas na rede; descoberta de portas; serviços e sistema operativo das máquinas; descoberta de vulnerabilidades nas máquinas; e descoberta de vulnerabilidades nos serviços. A adicionar à parte da auditoria, o dispositivo deve também permitir captura de tráfego, disponibilizar um relatório final com a informação recolhida, dar acesso a alguns parâmetros de configuração e *input* direto pelos botões físicos. Foram assim projetados vários módulos: análise da rede; análise de vulnerabilidades de sistema; análise de vulnerabilidades em serviços; recolha de dados e criação de relatório; interface *web*; interface de *hardware*; captura de tráfego; e ferramentas *wireless*.

Automatização e Interfaces

Os capítulos 4 e 5 apresentam o principal desenvolvimento realizado no decorrer do trabalho, incluindo a seleção, automatização e desenvolvimento de ferramentas, assim como as interfaces para comunicação com o utilizador e a geração do relatório final.

A partir da definição de funcionalidades e dos módulos projetados, várias ferramentas foram selecionadas para integrar o dispositivo, e posteriormente automatizadas de forma a permitir que a auditoria decorra de forma automática, com um mínimo de intervenção por parte do utilizador. Foram selecionadas para integrar o dispositivo as seguintes ferramentas: Nmap, OWASP ZAP, SSLyze, Hydra, Ettercap, SQLmap e Aircrack-ng. Foi ainda desenvolvido um agregador de tráfego recorrendo à biblioteca `Pcap`.

Estas ferramentas foram então automatizadas e configuradas no sistema com parâmetros pré-definidos de forma a tentar garantir a maior abrangência possível. Algumas fornecem *Application Programming Interfaces* (APIs), e o processo de chamada às suas funções é o processo de automatização escolhido, enquanto outras apenas funcionam através de linha de comandos. Nestes casos, foram testados e escolhidos os melhores comandos e criados *scripts* de forma a efetuar a chamada destas ferramentas, tendo em conta os dados adquiridos ao longo do processo de auditoria, de forma a otimizar a utilização das mesmas.

Em termos de interfaces, foram implementadas duas, uma de hardware, através do módulo RGB1602, e que dá acesso a algumas funcionalidades mais simples e diretas, como permitir efetuar nova análise à rede a que o dispositivo está ligado, ou iniciar captura de tráfego, permitindo também obter informações simples através do ecrã que o módulo possui. A interface *web*, por sua vez, dá acesso às configurações mais avançadas do dispositivo, como a definição de alguns parâmetros das ferramentas que integram a auditoria, visualização do relatório da auditoria e ainda a possibilidade de definir que ferramentas se pretendem excluir do processo, com vista a permitir agilizar o processo quando apenas se pretende analisar áreas ou vulnerabilidades específicas. Também é possível, através desta interface, lançar uma auditoria a um sistema remoto, desde que o dispositivo tenha ligação a este através da Internet.

Testes

O sexto capítulo descreve os testes efetuados para validar a implementação efetuada, assim como alterações e melhoramentos adicionados, e ainda os resultados de um estudo comparativo do dispositivo num ambiente real com um auditor humano.

Vários testes foram então preparados, baseados sobretudo na deteção de vulnerabilidades conhecidas em sistemas de teste. Foram utilizados uma rede doméstica previamente conhecida, as aplicações *web* BodgeIt, ZAP-WAVE e Awstats, um servidor com um certificado com erros propositados e serviços *File Transfer Protocol* (FTP), *Secure Shell* (SSH) e MySQL com palavras-passe fracas.

Dos testes efetuados, o dispositivo funcionou, no geral, como esperado, não tendo, no entanto, sido capaz de detetar injeções de *Structured Query Language* (SQL). Durante o seu funcionamento, detetou todos os dispositivos da rede doméstica, assim como as portas abertas e serviços que corriam neles, e os seus sistemas operativos. Foi ainda capaz de detetar as restantes vulnerabilidades nas aplicações *web*, incluindo *Cross-Site Scripting* (XSS), redirecionamento para endereços externos e inclusão remota de ficheiros, e foi capaz de descobrir as credenciais de autenticação dos serviços referidos.

Para validar a utilização do dispositivo num ambiente real, este foi utilizado numa auditoria realizada na Universidade da Beira Interior, promovida pela equipa UBI-CSIRT, uma equipa de análise de segurança localizada no departamento de informática da instituição. Após comparativo com a auditoria realizada por um dos peritos da equipa, e a realizada de forma automática pelo dispositivo, verificou-se um cenário semelhante ao obtido nos testes controlados: o dispositivo foi capaz de identificar os mesmos sistemas e vulnerabilidades que o auditor, tendo falhado na deteção de injeções SQL.

Conclusões

O capítulo 7 enumera as principais conclusões a serem retiradas deste trabalho e apresenta possível trabalho futuro a realizar.

O processo de auditar uma rede em termos da sua segurança é sempre um processo moroso que exige capacidades técnicas e conhecimentos aprofundados da área. Através de um dispositivo que permite obter informações acerca de potenciais vulnerabilidades na rede, sem que para isso os administradores de sistemas ou de redes tenham de possuir os conhecimentos supra referidos, há uma mais-valia em termos da segurança da rede, sem a necessidade de despender muitos recursos para efetuar análises regulares.

Algumas dificuldades notadas durante este projeto advieram da utilização exclusiva de código *open-source*, por vezes não documentado, ou aplicações não preparadas para correr em dispositivos com arquitetura *Advanced RISC Machine* (ARM), na qual o Raspberry Pi é baseado. No entanto, o protótipo foi criado com sucesso. Observando os testes efetuados, pode-se concluir que os objetivos principais foram atingidos, tendo-se obtido um rácio de deteção globalmente positivo, e completa automação nas tarefas principais.

Como trabalho futuro foram apontados vários melhoramentos que podem ser incorporados no protótipo. Entre outros, os melhoramentos mencionados incluem a integração de novas ferramentas, tanto como alternativas às existentes como para adicionar novas funcionalidades, como por exemplo testes de penetração, melhorar a automatização a nível de deteção de injeções

SQL, onde se verificou a maior falha do dispositivo, ou ainda melhorar a interface de hardware de forma a permitir a ligação a uma rede sem fios sem a necessidade de aceder ao dispositivo sem necessitar de ter acesso a este através de linha de comandos.

Abstract

Network security assessments are seen as important, yet cumbersome and time consuming tasks, mostly due to the use of different and manually operated tools. These are often very specialized tools that need to be mastered and combined, besides requiring sometimes that a testing environment is set up. Nonetheless, in many cases, it would be useful to obtain an audit in a swiftly and on-demand manner, even if with less detail. In such cases, these audits could be used as an initial step for a more detailed evaluation of the network security, as a complement to other audits, or aid in preventing major data leaks and system failures due to common configuration, management or implementation issues.

This dissertation describes the work towards the design and development of a portable system for quick network security assessments and the research on the automation of many tasks (and associated tools) composing that process. An embodiment of such system was built using a Raspberry Pi 2, several well known open source tools, whose functions vary from network discovery, service identification, Operating System (OS) fingerprinting, network sniffing and vulnerability discovery, and custom scripts and programs for connecting all the different parts that comprise the system. The tools are integrated in a seamless manner with the system, to allow deployment in wired or wireless network environments, where the device carries out a mostly automated and thorough analysis. The device is near plug-and-play and produces a structured report at the end of the assessment. Several simple functions, such as re-scanning the network or doing Address Resolution Protocol (ARP) poisoning on the network are readily available through a small LCD display mounted on top of the device. It offers a web based interface for finer configuration of the several tools and viewing the report, aso developed within the scope of this work. Other specific outputs, such as PCAP files with collected traffic, are available for further analysis.

The system was operated in controlled and real networks, so as to verify the quality of its assessments. The obtained results were compared with the results obtained through manually auditing the same networks. The achieved results showed that the device was able to detect many of the issues that the human auditor detected, but showed some shortcomings in terms of some specific vulnerabilities, mainly Structured Query Language (SQL) injections.

The image of the OS with the pre-configured tools, automation scripts and programs is available for download from [Ber16b]. It comprises one of the main outputs of this work.

Keywords

Forensic Analysis, Network Security Assessment, Pentesting, Process and Tools Automation, Security Audit, System Security, Vulnerability Detection

Contents

List of Figures

List of Tables

Listings

Acronyms

API	Application Programming Interface
ARM	Advances RISC Machine
ARP	Address Resolution Protocol
CPU	Central Processing Unit
CSRF	Cross Site Request Forgery
CVE	Common Vulnerabilities and Exposures
DBMS	DataBase Management System
eMMC	embedded MultiMedia Controller
FTP	File Transfer Protocol
GPIO	Global Pin Input Output
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IBM	International Business Machines
IP	Internet Protocol
IV	Initialization Vector
LCD	Liquid Crystal Display
MAC	Media Access Control
NIST	National Institute of Standards and Technology
ONT	Optical Network Terminal
OS	Operating System
OWASP	Open Web Application Security Project
PCB	Printed Circuit Board
PHP	PHP: Hypertext Preprocessor
PwC	PricewaterhouseCoopers

RAM	Random Access Memory
SHA1	Secure Hash Algorithm 1
SQL	Structured Query Language
SSH	Secure SHell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UBI	University of Beira Interior
UBI-CSIRT	University of Beira Interior Computer Security Incident Response Team
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VM	Virtual Machine
VoIP	Voice over Internet Protocol
WEP	Wired Equivalent Privacy
XML	eXtensible File Format
XSS	Cross Site Scripting
ZAP	Zed Attack Proxy

Chapter 1

Introduction

This document describes the work performed in the scope of a project for the attainment of a master's degree in Computer Science and Engineering at the University of Beira Interior (UBI). This dissertation addresses the subject of network security assessment, and how its automation may simplify auditing processes on a network. The following section presents the motivation and scope of the work. Section 1.2 discusses the adopted approach. The penultimate section presents the main contributions of this work, and the last section describes the structure of this dissertation.

1.1 Motivation and Scope

Most organizations nowadays are supported by complex technological and network infrastructures, which represent a critical point of failure for most of them. Every year, costumer and institutional data are exposed due to attacks on these infrastructures. Included in the top 10 list of most common vulnerabilities by Open Web Application Security Project (OWASP) [OWA16a] are injections, security misconfiguration and the use of components with known vulnerabilities. Most of which could be prevented if systems and their services were up-to-date, patched and properly configured.

To provide a better perspective on the extent of the issue it can be mentioned that, in 2015, 90% of large organizations in the UK had a security breach, as reported by PricewaterhouseCoopers (PwC) [PwC15]. Also, and according to International Business Machines (IBM) [Pon15], the cost of these data breaches is increasing each year. The single largest data breach in 2015 saw data of 80 million users exposed. Hacking continues to be the main threat responsible for security breaches in 2016 [Ver16]. Many of these hacks are due to successful exploitation of Common Vulnerabilities and Exposures (CVE). 85% of the exploited vulnerabilities are from the top 10 vulnerabilities and the remaining 15% concern over 900 different vulnerabilities.

As such, network security audits have become common practices in the process of guaranteeing the security of computer systems and networks, and keep data and services protected from threats, both external and internal. The security assessment phase is one of the most time consuming tasks in the audit process, due to being typically based on the use of a varied set of tools, each requiring knowledge of its inner workings, with several different configuration parameters. For these reasons, the existence of a device capable of automating the process, even if only partially, constitutes a valuable addition to an audit team.

The scope of this dissertation is limited by the areas of networking and computer security, as it aims to explore connectivity to perform the assessment of a network and of its systems in terms of security. Under the 2012 version of the ACM Computing Classification System (CCS), the topics that best describe this dissertation can be defined as follows:

- **Security and Privacy~Network Security;**

- **Security and Privacy~Software and application security;**

- *Security and Privacy~Systems Security;*

- Networks~Network services.

1.2 Problem Statement and Objectives

The main problem addressed in this work is the complexity inherent to network security assessments and the time they consume, which can easily be associated with monetary costs for the infrastructure or services owner. Due to the rapid advancement of technology, proliferation of malware and discovery of breaches, exploits and bugs every day, it becomes a taxing work to keep up and monitor a network in an efficient way. Additionally, in many cases, direct access to systems or social engineering is not permitted during an audit, which means that the auditor may not have access to target systems, which are previously configured and connected to the network, or be able to plug in any device into a machine. A device like the one proposed in the scope of this work may prove itself to be an invaluable resource, as the only way to gain access to the network is by connecting said device into the network.

The main objective is then to design and construct a compact and inexpensive device for quickly performing network based security assessments. This includes the required programming logic that will enable the box to perform several tasks in a fully automated, near-transparent and user-friendly manner. The system should not only detect functioning problems, but also assess the security state of previously identified systems and networks. The system should be able to be quickly deployed in a wired or wireless network, or deploy-and-play. For most of the functionality, the user should only need to connect the cables or place the device in range of a wireless network, and turn the system on. As the main functions, the system should be able to sniff and analyze network traffic, perform network discovery, assess vulnerabilities and do penetration testing, and finally produce a report.

1.3 Adopted Approach for Solving the Problem

The chosen approach to solve the aforementioned problem began with the study of what generally comprises a security audit, the components tested and the utilized techniques. Such information was needed to identify and test different tools before integration and to understand how they were executed. The identification of the functionalities that should be made available to the user, both automated and manual, also needed to be performed. This was done in a third phase of the project. Next, it was necessary to choose the most appropriate tools to be integrated. As the system should be able to present a final report with minimal input from the user, the next step was to find a way to automate and cascade the execution of tools and organize the collection and aggregation of relevant output data. To complete the system, a graphical interface for ease of use and transparent communication with the user needed to be developed. After the development of an initial version of a prototype, several tests needed to be conducted, both in a controlled environment and in-the-wild, deploying the device in a real network and comparing results with a manual audit to validate its utility.

1.4 Main Contributions

The main contributions achieved from the research and development of the idea herein presented can be enumerated as follows:

1. The design, assembling and testing of a small computer system for network based security assessments, along with the disclosure of its plans;
2. The study and structuring of a basic network security assessment, and the development and setting up of the means required to automate it. This included the configuration of the operating system, adaptation of tools and development of scripts, all disclosed as open-source;
3. The development and disclosure of specialized lightweight tools, required for the correct functioning of the forensic box, taking its limited computational resources into account. The traffic sniffer is an example of such a tool.

The work developed in this dissertation was presented in the form of a communication entitled *Forensic Box for Quick Network-Based Security Assessments*, published in the proceedings of INForum 2016 conference, held in Lisbon, Portugal, between the 8^{th} and 9^{th} of September, 2016. It was also presented in *Techdays*, a technology forum held from the 15^{th} to the 17^{th} of September in Aveiro, Portugal. The device developed during the course of the master's project was used in an audit to several machines of UBI in the scope of activities promoted by the University of Beira Interior Computer Security Incident Response Team (UBI-CSIRT) team, a security audit team based on the computer science and informatics services department of the institution.

1.5 Dissertation Organization

The dissertation is organized in 7 chapters and 2 appendices, which can be briefly presented as follows:

- Chapter 1 — `Introduction` — presents the motivation and scope of this project, the problem that it attempts to solve, the main objectives, the chosen approach for solving the presented problem, the main contributions of this work and the organization of the document;
- Chapter 2 — `Related Work and Technology` — the analysis on similar works and different existing tools, along with the existing main technologies related to the work herein presented or used within its scope;
- Chapter 3 — `Hardware Specifications and Blueprinting` — presents the chosen hardware for the device, the reasoning behind the choices and the blueprint of the system components and their interactions;
- Chapter 4 — `Forensic Tools Automation` — discusses the tools chosen to integrate the device, the choices made and the reasons for such choices, and the steps taken towards integration, automation and correct functioning of said tools;
- Chapter 5 — `Reporting and Interface` — presents the development and inner workings of the functionality for creating the report of the security assessments, including their structuring, as well as those of the interface, and its functionalities;
- Chapter 6 — `Testing, Fine-Tuning and Security Audits` — specifies and discusses the tests done to validate the results of this work, the fine-tuning done to improve the given results, and examples and comparisons with audits done in both controlled and in-the-wild environments, to validate the utility of the device;
- Chapter 7 — `Conclusions and Future Work` — discusses the main achievements and conclusions and contains a list of possible improvements that can be made in future revisions of this work;
- Appendix A — `Code Excerpts` — presents some of the larger code excerpts as a complement to the explanation of the automation process presented in chapter 4;
- Appendix B — `Full Report` — contains a full report of a performed audit, for better insight on report structure and detection parameters.

Chapter 2

Related Work and Technology

2.1 Introduction

This chapter describes several works related to the subject at hands and technologies that can be considered alternatives or complements to the system present herein. Performing security audits on a regular basis is paramount to keep data and services protected from threats, both internal and external [Mid02, Tan14]. Audits are typically based on the use of a varied set of tools, each requiring knowledge of its inner workings, with several different configuration parameters. While there are several guidelines and methodologies that can be followed during security assessments, there is still not a de facto standard for such complex tasks. Section 2.2 presents works related to auditing processes, automation suites, methodologies and guidelines.

Some of the tools used in penetration testing make the task of a network audit cumbersome. They also require advanced technical expertise and consume a considerable amount of time, due to their many configuration parameters and different functioning modes. Section 2.3 is devoted to the description of some of the most popular tools used in network or system security audits, and since the main purpose of this work is to provide a device capable of performing automated security audits, a better insight on what comprises an audit is provided in section 2.4.

2.2 Related Work

In this section, several works towards automation, regulation and standardization of security assessments and audits are described, together with a brief discussion on how they influenced this work.

2.2.1 CNSSA

Rongrong Xi *et. al.*, present what they call CNSSA (Comprehensive Network Security Situation Awareness) [XJYZ11]. This tool gives a quantitative assessment on the state of the security of a network by analyzing information from different sources, from data flowing between hosts on the network to information on threats, vulnerabilities and alerts. It is based on three main modules: (i) an information collector module, which gathers data from the network and stores it in a database; (ii) a situation awareness module, that takes the collected data as input and generates assessments on threats, vulnerabilities, stability and situation, where each of these assessments outputs a score that can measure from 0 to 10, where higher values indicate an insecure network; and (iii), a situation visualization module, which provides a user interface that gives different views on the security status of the network.

2.2.2 A Framework for System Security Assessment

Hallberg *et al.* describe a framework [HHP05] that attempts to include the system structure in the assessments. The objectives are to help categorize existent methods for performing assessments and aiding in developing new methods. The framework is based on the hypothesis that, by knowing the security values of all security-relevant system entities and knowing all security-relevant relations between those entities, one can decide the security values for the entire system. The framework defines a workflow involving two main tasks: (i) model the system in terms of its entities and their relations and (ii), use this model to assess the security of the system. System modeling comprises the description of the system under analysis, its predefined entities and their relations. The modeling of entities and relations is one of the most critical steps in terms of information gathering of the framework, since it will be later used to assess the security status. Using entities that have been previously assessed may simplify the assessment process, just as using standardized relations. Adjusting parameters to attempt to bring these standard relations and entities closer to the real characteristics of the entities and their relations can also improve their modeling. The final steps comprise the assessment of the entities and their relations, and interpretation of the gathered information. The presentation of the results is the final step of the framework. The framework does not necessary specify the method for calculating the level of security of a system. Different methods can be used for such purpose.

2.2.3 Web Application Security Assessment Tools

Mark Curphey and Rudolph Araujo approach the security assessment on web applications and websites topic on their work [CA06]. Assessing the security of these applications is becoming the focus of most companies, mostly due to the fact that web applications are becoming the main entry and exit point for data everywhere. According to them, the starting point for assessing the security of web applications is threat modeling. Modeling immediately provides an overview on the architecture of the system. It then becomes important to identify the two major types of vulnerabilities, implementation bugs and design flaws, which allows for the definition of an analysis framework. Different vulnerability types can be defined in the framework, such as configuration mismanagement, authentication issues or user session management. This facet of the framework shows once again that there is no one size fits all for security audits.

The tools utilized in the framework can then be divided into several categories. Source-code analyzers search through the source-code of an application looking for specific strings or patterns that can represent security issues. There are static and dynamic analyzers and their outputs may comprise invaluable resources during the development process, where correcting security flaws is simpler and has reduced costs. Code analyzers are typically better when applied to languages such as C and C++, and less efficient when applied in Java or .NET applications. This is mainly due to the maturity and structure of the traditional languages. Web application scanners, also known as black-box scanners, use browser-based exploration techniques, looking for Uniform Resource Locators (URLs) and executing predefined tests on each page found with the previous procedure. They are not suitable for testing during the production stage, as they require a minimally functional application to be used, and the detected issues are often not enough to determine where the application code is problematic, leaving developers to explore and find the issues. Other tools used in the context of security assessments of web applications include:

- database scanners, acting as Structured Query Language (SQL) clients and doing various queries to analyze the security configuration of a database;
- binary analysis tools, used to test a number of different inputs and identify unexpected behaviors or crashes in C and C++ applications;
- runtime analysis tools, which work as profilers and can log function calls and parameter values. They are mainly used during the development phase and for code reviewing;
- configuration analysis tools, mostly used to inspect configuration files, host settings or server configurations, or proxies that intercept web traffic and allow packet manipulation. These tools are typically used to evaluate the effectiveness of implemented security measures at both the client and the server side.

The authors conclude by stating that it is not possible to pinpoint a specific tool for every job and that tools should also be tested for their effectiveness, since different tools of the same class can perform differently when confronted with different types of vulnerabilities.

2.2.4 Security Wheel - Cisco

Cisco uses a security wheel [Cis03] when referring to network security to emphasize that it is a continuous and cyclical process, encouraging regular testing and updating of security measures as a way of protecting a network. The entry point of the wheel is the definition of a security policy, that should identify the main security objectives of the organization/network, document the resources that should be protected, identify the infrastructure and also the critical resources of the network. The main four steps of the wheel are called secure, monitor, test and improve. The first step – secure – refers to the need of establishing and implementing security measures, from user authentication and firewall configuration to vulnerability patching. The second step – monitoring – involves detection of security violations, mainly through the analysis of event logs. The third step – test – engages in testing the policies and measures implemented in the first step, through system and network auditing, resorting to different tools. The last phase – improvement – picks on the data collected on the previous two phases and extracts the required improvements that are needed to solve the issues that arose. The new improvements will be applied in a new phase one, and the cycle proceeds to the initial stage again.

2.2.5 How to Build Your Own Penetration Testing Drop Box

Beau Bullock [Bea16] created a penetration testing and auditing *drop box*, based on a device similar to a Raspberry Pi. The defined requirements were that it should be a portable, inexpensive device, easily hidden, that could connect using wired and wireless means, run a full Operating System (OS) and be fast enough to be used in a real environment. The author tested three devices, a Raspberry Pi 3, a Beaglebone Black and a Hardkernel ODROID-C2. He chose the ODROID-C2 as the final platform for the device, due to its superior performance and possibility of installing embedded MultiMedia Controller (eMMC) storage.

The device was added a wireless adapter, the eMMC memory chip and a case with active cooling to complement its base configuration. In terms of OS, Kali Linux was used for its ease of use, fast installation and Advances RISC Machine (ARM) support.

To assess the performance and utility of the devices, the author made several different tests, from the time they took from booting up to opening a Metasploit console, to password cracking, port scanning, and read and write operations speed on their storage. From the tests, the ODROID-C2 pulled consistently ahead in terms of its performance, with the exception of the Nmap test, where all the devices had similar performance, mainly due to the test being more dependent on the network connection than on the processing power of the device.

The author used the device in a Red Team exercise (an exercise where a team of security experts assess the security of an organization, usually without knowledge to clients and staff, nor previous knowledge from the team on the target network), where the device was left connected for three days and remotely accessed, through a Secure SHell (SSH) tunnel, to perform penetration tests directly from the inside of the network. The author supplies a detailed description on the installation process and configuration of the device for simple replication.

It should be mentioned that the *drop box* does not perform automated audits. Its purposes slightly different from the ones of this project, since the idea is to provide a box that can be eventually connected to a network and hidden to perform pentesting from a remote location.

2.3 Technology

This section presents several existing tools and security software available and commonly used for auditing the security of systems and/or networks. Some of these tools were considered to be integrated in the prototype of the device developed in the scope of this work. The tools automated and integrated in the device will be described in the next chapter.

2.3.1 Lynis

Lynis [cis16b] is a powerful security auditing tool for the cases where one has direct access to the system and administrative rights to run software on it. It is capable of revealing more vulnerabilities than the common security scanners can do remotely. It runs on a wide variety of UNIX-based distributions, and is an open-source software.

The tool is an opportunistic scanner, which means that it only tests what it finds, for efficiency purposes. It starts with the detection of the operating system, and proceeds with the identification of tools, services and utilities. It then runs tests based on installed plugins for each category discovered in the beginning of the process. It finishes by presenting a report of its findings.

In terms of end results, Lynis has some similarities with what is proposed in this work (e.g., it is semi-automated and runs plugins based on previous findings). The main difference is that Lynis can only run locally, and must be installed on the machine one wants to test. This is its biggest limitation, alongside with lack of support for non UNIX-based systems, which renders the tool useless when wanting to audit a Windows system, for example.

2.3.2 Metasploit

Metasploit [Rap16b] is a powerful framework that handles hundreds of different payloads capable of compromising a system. Featuring a database with over 1300 exploits, it is one of the most well known penetration testing suites existing today. It is offered both in a paid and a free version, where the paid version allows for integration with other tools from Rapid7 (responsible for its maintenance) for increased productivity, better reporting and extended functioning.

Metasploit can be further improved with the addition of a Graphical User Interface (GUI) open-source tool, like Armitage [Str16], which improves usability and gives access to some useful features, chief among them being Hail Mary. Hail Mary is a brute force method of attempting any potential exploit in every host, given the operating system and the services available. While not a subtle method (as brute force never is), it can achieve positive results without the need to specifically search and individually test different exploits.

Another available suite to complement Metasploit is Cobalt Strike [Rap16a], a paid tool which allows for threat emulation and security assessment, creating an environment that can be deployed on a real network to test security and incident response. It gives access to different types of testing, from phishing to payload injection.

2.3.3 Nessus

Nessus [Ten16] is a proprietary vulnerability scanner developed by Tenable Network Security. Available for free as a trial for personal use, it is comprised of two main components: (i) a daemon, for scanning, and (ii) a client, to control the scans. When executed, it begins with a port scan, and attempts several exploits on the open ports. Other functionalities include password auditing through dictionary attacks and brute force attacks.

Nessus can be deployed in physical, virtual and even cloud environments, adding to its versatility in terms of usage scenarios. As a full vulnerability assessment solution, Nessus is not only able to find potential vulnerabilities and misconfigurations, as also to detect malware, viruses and even communications with botnets. The plugin database is updated on a regular basis, so as to incorporate new vulnerabilities and threat signatures. Reports can be generated following standards compliant with the specific area the company is integrated.

2.3.4 OpenVAS

OpenVAS [Gre16] is an open-source fork off of Nessus, after the software became proprietary in 2005. It is, much like Nessus, a framework that offers several services and tools for comprehensive vulnerability scanning and vulnerability management. It has over 47000 network vulnerability tests in its database, which are executed by the scanner provided with the framework. The tests offered by OpenVAS are served through a public feed, updated weekly. The tool allows for the change of the feed for a commercially licensed one.

2.3.5 Snort

Snort [Cis16a] is an open-source network intrusion prevention and detection system developed by Cisco. It performs real-time traffic analysis and packet logging, as well as packet *sniffing* (displays or logs all captured packets). It allows for the specification of different rules that can then be matched with the analyzed packets, allowing detection of OS fingerprinting attempts, port scans and other probes and attacks.

Snort has its own language for creating rules. There is also a group – Talos – composed by several security experts, that provides a suite of rules that are constantly updated for improving detection in hacking activities, intrusion attempts, malware and vulnerabilities. This suite of rules is available for a subscription fee.

2.3.6 dsniff

Dsniff [Dug00] is a group of network analysis, auditing and penetration testing tools, that monitor a network for relevant data, such as passwords, e-mails or files. Several of its tools facilitate the interception of such traffic, while others allow for the realization of man-in-the-middle attacks. It mainly works as a network sniffer, but can also be used to disrupt the normal functioning of the network, and even cause traffic from other hosts on the network to be visible. Dsniff is available on some UNIX-based platforms, mainly OpenBSD, Redhat and Solaris and is available for free, though as a closed-source software.

2.3.7 Xplico

Xplico [Cos16] is a network forensics analysis tool whose purpose is to reconstruct data extracted from captured network packets. It analyzes not only the protocol of a packet, but the application data itself, namely HyperText Transfer Protocol (HTTP), Voice over Internet Protocol (VoIP) calls, File Transfer Protocol (FTP) files, etc. Data can be dumped into a database or eXtensible File Format (XML) file, and decoding can be defined on a pert protocol basis, allowing for decoding only the protocols the user desires. The tool is able to decode large amounts of data at a time, claiming inclusively that several terabytes of data supplied from different sources can be decoded, due to its multithreaded implementation. Xplico is available as a free and open-source software.

2.4 Network Based Security Audits

A security audit on a network comprises different steps to achieve the final goal of assessing the status of the security of the elements (e.g., servers, hosts and routers) on that network. There are many different standards and guidelines to guide the process of an audit, many of them specific to certain areas of business (health, energy, among others)- There are also general guidelines published by different organisms whose main objective is to create a basis on what a network security audit should cover, its principles and objectives. One of the most concrete and respected documents on this subject is the *Guideline on Network Security Testing* [WTS03] by the National Institute of Standards and Technology (NIST), which includes the following recommendations concerning security audits:

- make network security testing a routine and integral part of the system and network operations and administration;
- test the most important systems first;
- use caution when testing;
- integrate security testing into the risk management process;
- ensure that system and network administrators are trained and capable;
- ensure that systems are kept up-to-date with patches;
- look at the big picture;
- understand the capabilities and limitations of vulnerability testing.

From these points, one can see that the auditing process should be done on a regular basis, if possible in a non-intrusive way, and giving priority to critical systems. The need for trained and capable administrators is an issue that the work presented herein attempts to minimize through the construction of a plug-and-play device for such task. Nonetheless, a deeper understanding on cyber-security should be a prerequisite for people performing security audits. In this case, the capacity of placing oneself in the mindset of an attacker is specially useful.

Additionally to the recommendations, different techniques and areas to take into account when performing tests are also discussed in the aforementioned document. The following list summarizes some of them:

- network scanning, which involves the use of Internet Protocol (IP) sweep and port scanning to identify hosts on a network and the services they provide;
- vulnerability scanning, involving the attempt to find vulnerabilities on the detected services on the hosts, following the data gathered from the previous step;
- password cracking, which attempts to identify weak passwords used in different services or systems, using either a dictionary or simply brute forcing by generating thousands of attempts to find a valid login;
- log review, comprising the analysis of logs from different components of the network, from firewalls to intrusion detection systems, in an attempt to find deviations from the normal functioning of these systems. This is usually not seen as testing but serves as a way to detect anomalies or check the correctness of functioning of defined policies;
- integrity checkers, involving the creation and comparison of checksums for every file on each computer on a network, and joining these checksums in a database, to prevent tampering of files;
- virus detection, both the type installed on hosts (commonly known as antivirus software) and those installed on the network infrastructure;
- war dialing, which consists in dialing a wide spectrum of phone numbers in an attempt to find a modem that may provide access to a network;
- war driving, the act of moving around collecting wireless networks;
- penetration testing, which attempts to circumvent security features on a system knowing the design of the system and its implementation, trying to gain access and, if possible, scale privileges to gain control over a host or even the entire network.

Log reviewing, integrity checking, virus detection and war dialing are tests that are not well suited for the work at hands, given the application scenarios and objectives of the forensic box. Some of these tests are host based, while others are clearly more suitable for offline execution. From the remaining, a system can be designed to fit the different techniques and attempt to automate them.

When defining the frequency of an audit, one should take into consideration the importance of the system and of the data it stores within the surrounding ecosystem. Critical systems should be audited with greater frequency, although some tests should be conducted more sparingly due to their offensive nature.

2.5 Conclusions

This chapter laid ground on different methodologies, guidelines and tools related with the subject of security assessments. One of the main conclusions is that there is no one size fits all solution for security audits. The ever changing and increasingly large landscape contribute for this problem. Fortunately, there seems to be plenty of tools, under active development, to aid in the task of analyzing the security of systems and networks. Some of those tools are open-source and community driven, regardless of the fact of being extremely powerful and effective.

Related works mention that modeling may comprise an interesting resource for an initial phase of an assessment. Network topology discovery via IP sweeps and port scanning can be seen as a means to construct part of the model automatically. The intermediate phase of audits can then be performed using different tools, depending on the objectives and, mostly, on the information that was previously obtained. Efficiency is dependent on how well the tasks can be branched out. The final phase is typically the one concerning the presentation of data, i.e., reporting. It can be said that the device developed in the scope of this work respects this general flow.

According to NIST and to Cisco, security testing should be a routine. The former also states that specialized personnel should be in charge of security, and that critical systems should be analyzed first. As it will become clearer in the following chapters, the development of the forensic box was partially influenced by some of the guidelines and recommendations discussed herein, specially its architecture and the sequencing of the several tools it integrates.

Chapter 3

Hardware Specifications and Blueprinting

3.1 Introduction

This chapter presents the specifications of the chosen hardware and the reasoning behind it, as well as the process of blueprinting the system structure and software components.

Taking the technologies and ideas in works presented in the previous chapter and the definition of a network based security audit, the requirements of the system to be developed were thus defined. Section 3.2 presents the hardware requirements and the reasoning behind them, of what specific needs the device has, as well as the different possibilities, choices made and the argument for them. The following section, 3.3, defines the blueprint of the system and its main components, their purpose, functionalities, and how they communicate with each other and the user, in terms of software. From these components, or modules, specifications, the software features and implementation can then be defined.

3.2 Hardware Specifications

When considering the appropriate type of device to be chosen, it was necessary to make an uplift of the requirements that the device had to meet. The requirements were mostly derived from the application scenario and objectives of the device and network based security audits, described in section 2.3. The following requirements were identified during this phase:

- Portable - the system should be able to be easily transported and deployed in any place;
- Energy Efficient - it should have low power consumption, much lower than a standard desktop or even laptop computer;
- Be able to run from a battery - this ties in with the need of being energy efficient, as a battery pack (such as a powerbank) should be able to supply the device;
- Wired and wireless connection to networks - the system should have both an ethernet port and wireless support for the most common standards;
- Capable of running a real time OS - as the system should have some fairly advanced capabilities, a real time OS (such as a UNIX based one) is needed;

- Compatible with security tools - security tools comprise the bulk of the system, and as such, it should be compatible with these;
- Physical interaction with the user - some simple commands should be able to be issued by the user without the need to access the device through its web interface, resorting to some form of hardware interaction, both to issue commands and receive basic information;
- Hosting a web service - for more complex customization and to supply the user with the findings of the device, a web interface must be made available. This interface will allow to configure parameters in several tools, access to reporting, and allow the user to target, for example, a machine with a public IP address outside of the network the device is in;
- Inexpensive - the device should have a low cost so it can be easily acquired and assembled by any professional.

The best option was to choose a single board computer: small form factor devices, with all needed hardware and features integrated in a single Printed Circuit Board (PCB). Looking at the remaining requirements, some of these devices would fit the specified requirements. Due to size and cost restrictions, the choice would have to fall upon a low-power device. However, Arduino and other similar devices, such as the Beaglebone A6 or the Texas Instruments MSP430 LaunchPad, are too under powered in terms of processing capabilities and cannot run a real time OS, rendering them inappropriate for the system. This reasoning narrows the list down to devices such as the Raspberry Pi or the Beaglebone Black as the best-suited alternatives. These devices meet the majority of the requirements, with the exceptions of wireless connection and providing a physical interface for interaction. However, both can be expanded very easily to support both features.

The main major difference between the two aforementioned devices, the Raspberry Pi 2 and the Beaglebone Black, concerns performance: the Raspberry Pi 2 (and upwards) has twice the amount of RAM and a much more powerful quad-core processor. Considering that the device will be used mainly to run different demanding software applications, the performance advantage benefits the Raspberry Pi.

The Raspberry Pi, being the most popular device, also has a much greater community behind it, and better support and accessories. The main advantages of the Beaglebone are the I/O in terms of pinout, the slightly lower price and the embedded storage. However, and considering the lesser role the hardware output has in the project, and the need for more storage space than what the device has, rendering the final cost similar, they do not comprise real comparative advantages for the device.

The choice fell then on the Raspberry Pi 2. The main reasoning behind the choice was the low cost, advantage in performance, the existence of several specific Linux distributions, as well as the community behind the device, both in terms of support and software.

To cover all requirements and features, additional modules were required. A wi-fi USB module and a small PCB with a LCD display and hardware buttons, for physical interaction, were added. The PCB is a RGB1602 module from PI52 [PI515], which integrates a MCP 23017 Global Pin Input Output (GPIO) expansion chip, for which several libraries are available for the Raspberry Pi. The wi-fi adapter is an Edimax EW-7811Un USB adapter,with support for the 802.11b/g/n standards and plug-and-play compatibility with the Raspberry Pi [Edi16]. The final assembled device has a compact design, as can be seen in figure 3.1.

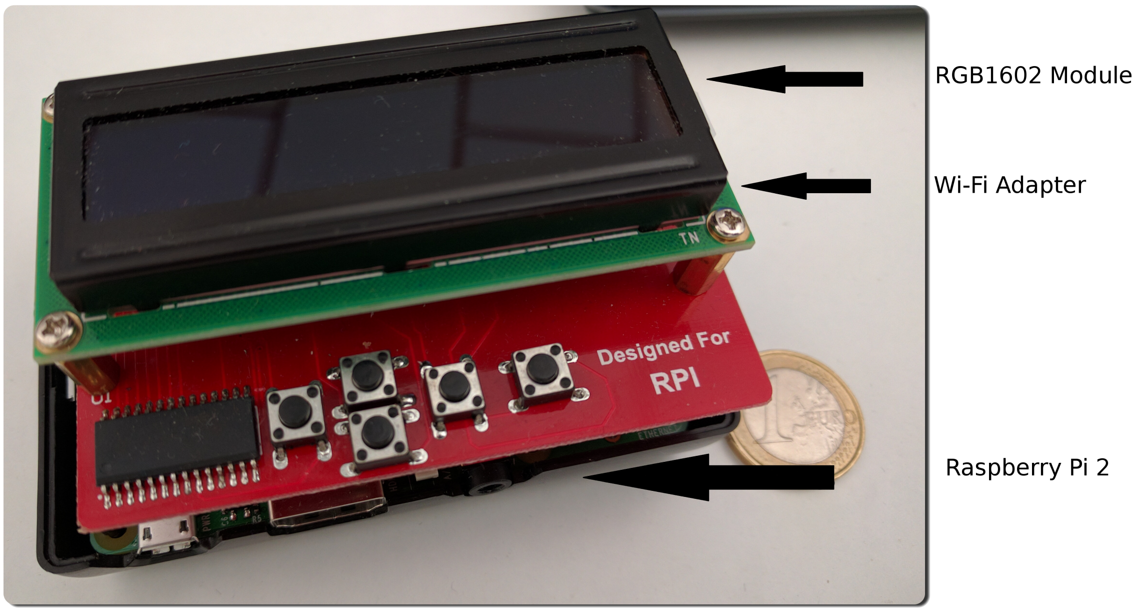


Figure 3.1: The assembled device based on the Raspberry Pi 2.

3.3 Blueprinting

To achieve its objectives, the proposed system needs several modules, each with a specific function, that interconnect and send or receive data from one another. From the analysis of what comprises a security audit, and establishing what data must be gathered to generate a concise, yet informative report, it is possible to identify components that the system needs and the interactions needed between them, and with the environment and the user.

To identify the required components, a definition of what the system will offer must be proposed. Looking at the existing methodologies and guidelines, the path for an audit can be established in the following steps:

- connect to the network - if the device is plugged in into a network, it should be able to acquire an IP address and connect to said network;

- discover hosts in the network - after being connected, the system should be able to discover all hosts on the network;
- discover open ports, services running, and the OS of the hosts - the system should scan the detected hosts for this data;
- discover vulnerabilities on the hosts - from the system data of the detected hosts, a list of vulnerabilities can be compiled;
- discover vulnerabilities in the services running on the hosts - in a similar fashion, a list of vulnerabilities can be created through scanning and probing the different services running on the network hosts.

Adding to the auditing part, the system should also be able to capture traffic (and be able to do so in a switched network), report on its findings, allow for some manual configuration of specific tools (both these functions available through a web interface), and allow for input through its hardware buttons.

With these prerequisites in mind, the following modules can be projected:

- network scan;
- system vulnerability scan;
- service vulnerability scan;
- data parsing and report creation;
- web interface;
- hardware interface;
- traffic capture;
- wireless tools.

Beginning with the network scan, this module should be able to be either manually or automatically launched (when detecting a wired or wireless connection), discover the existing hosts on the network, their ports and services, and the OS of each host. The data gathered in this module is of critical importance for the system, since it is the input that will be fed to the majority of the remaining modules.

The system vulnerability scan should encompass the discovery of possible vulnerabilities in hosts in terms of their OS or running programs. The information generated in this module should then be sent to the data parsing module for integration in the final report.

The service vulnerability module, with the input of the data acquired on the services running on the hosts, should check for vulnerabilities in these services. It should be noted that this module encompasses, among others, web scanning and brute force attacks. The results are also sent to the data parsing module.

The data parsing module will be responsible for receiving the output of all the tools involved in the auditing process, parsing said outputs and compiling them into a report that can then be presented to the user with the results of the automated audit.

The web interface will allow for the user to fine tune some parameters on the integrated tools, as well as direct the audit to a specific network or host, view the report, and use some of the tools manually.

The hardware interface module will allow the user to interact with the device through its top mounted display and buttons, and issue some simple commands, such as starting network traffic capture. Some information on the status of an ongoing audit, for example, can also be displayed.

The traffic capture interface makes use of packet capture, that will then be stored in a file the user can consult and extract to an external device for detailed analysis, or for cataloging the main type of traffic circulating on the network. This module will also support some man-in-the-middle attacks to redirect traffic on a switched network, so that all network traffic can be captured (more details about this feature in chapter 4).

The wireless tools module will allow for the attempt on cracking Wired Equivalent Privacy (WEP) keys when a WEP protected network is in range.

All interactions between the aforementioned modules are represented in figure 3.2. Dashed arrows denote data flows, while normal arrows denote issuance of commands. For example, the web interface can issue commands for the several modules, and the network discovery module provides data to many other modules.

3.4 Conclusions

This chapter laid the foundation for both hardware and software comprising the device. Having defined the hardware components and the software interactions and structure of each module in this chapter, it is possible to define OS, types of tools needed to implement and automate, as well as interaction logic and interfacing. The next chapter presents the forensic tools implemented in the system and the steps towards their automation. The chapter after the next is also influenced by several decisions taken in this chapter.

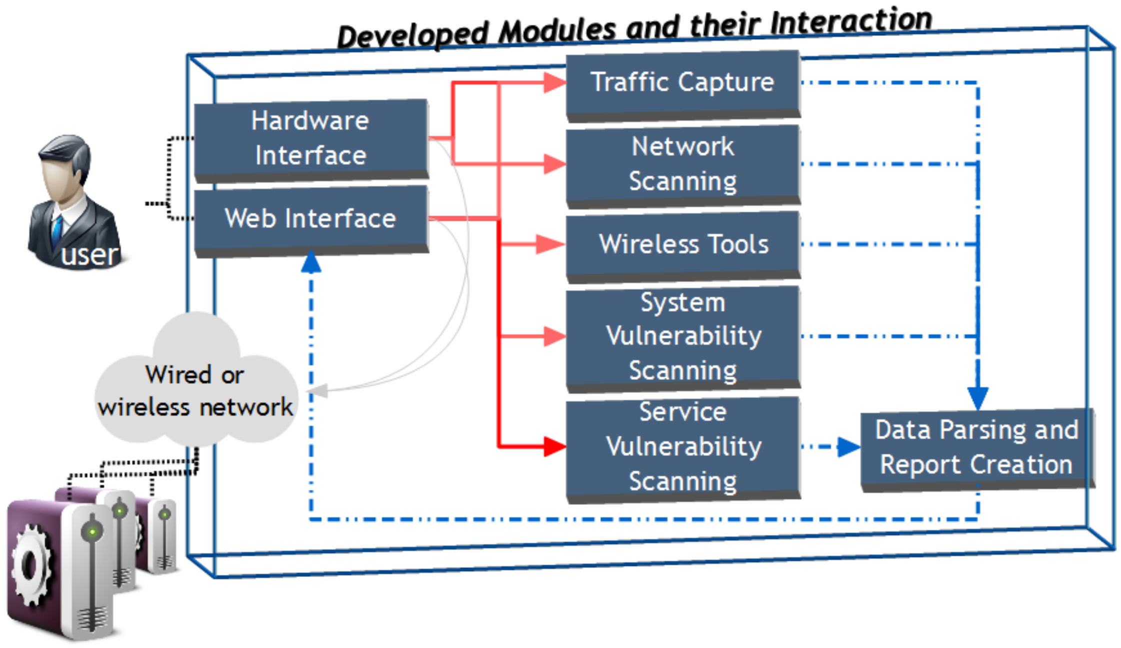

Developed Modules and their Interaction
user
Hardware Interface
Web Interface
Wired or wireless network
Traffic Capture
Network Scanning
Wireless Tools
System Vulnerability Scanning
Service Vulnerability Scanning
Data Parsing and Report Creation

Figure 3.2: Simplified blueprint of the system components and their interactions.

Chapter 4

Forensic Tools Automation

4.1 Introduction

This chapter describes the tools integrated in several system modules and the process towards their automation.

Starting from the modules identified, presented and defined in chapter 3, a study of the existing tools for the appropriate tasks was made and different tools were selected. The basis for this selection is given in this chapter, with the reasoning for the choices duly presented from several angles, from the effectiveness of the tool, the type of output given to the existence of an Application Programming Interface (API) for controlling the tool. Section 4.2 presents the different tools integrated in the system along with an explanation of its objectives, its main role in the system and the module it integrates, as well as its main options and functionalities. Section 4.3 introduces and provides details on the steps towards the automation of the tools referred in the previous section. Along with the steps taken, some examples are also shown to illustrate the approach.

4.2 Forensic Tools

The different tools used in the system developed in the scope of this project are presented herein. Each tool is accompanied by a description of its functionalities, its role in the system and why the choice befell on that particular software.

4.2.1 Nmap

As the first part of an audit is the detection of hosts on the network, a network scanner (or mapper) was the first type of tool to be taken into consideration. While several options exist, such as the Network Scanner from SoftPerfect [Sof16] or the Advanced IP Scanner by Famatech [Fam16], most existing tools are either exclusive to Windows OSs or only give access through a GUI, neither of which proves suited for the intended use. The choice fell upon Nmap [Gor16b] for the prospect of network scanning.

As its name implies, Nmap discovers hosts on a given network as its core functionality. However, Nmap can double as a potential vulnerability scanner. Among its many functionalities, the service scanning and OS fingerprinting features are the most interesting ones for the system.

The first one, service scanning, provides means to perform the scanning of services running on any open port on a host. Nmap is able to discover these services through probing the port and analyzing the received answer. Most of the time, it is even able to discover the version of the service. This can prove invaluable as it allows the system to report the version of the service so that the user can check if it is up-to-date or not, or what known vulnerabilities the current version has. The second functionality, OS fingerprinting, ties in with the first one advantages as well. By being able to detect the OS a host is running, it is also possible to, through the version of the OS, discover potential system vulnerabilities.

On a more practical side, one of the major advantages of Nmap is the possibility of exporting its result into a XML file. This proves advantageous due to the ease with which such a file can be read and parsed, and its data registered on a structure that can then be directly accessed. As this part of the audit is the one whose data branches out towards the other parts, the convenience of this functionality is highly valued.

4.2.2 OWASP ZAP

Usually, most systems accessible from the outside of a network (public servers, for example) provide some type of service or services that anyone can access. These services typically span from simple web pages to advanced web applications. They are entry points in attack attempts and their vulnerabilities are critical, as emphasized by the OWASP Top 10 project.

There are several tools for scanning web services for potential vulnerabilities. These tools are usually known as web application vulnerability scanners or, for short, web scanners. Some examples are the Burp Suite [Por16], Acunetix WVS [Acu16], OpenVAS [Gre16] and Nexpose [Rap16c]. Most of these tools are closed-source and expensive, though they incorporate other functionalities, such as network scanning. As the system developed is based solely on open-source software, the chosen vulnerability scanner is the Zed Attack Proxy (ZAP) [OWA16b] by OWASP. It encompasses vulnerability discovery in web applications, and has a set of features that benefit its integration in the system. Among these features, the existence of an API written in different languages, such as Java or Python, and the XML output of the scan results are the most advantageous ones.

The features of ZAP are structured into two main parts. The first one is the spider. The spider finds all the URLs existent on a given site. Internally, ZAP has two spiders; the original one, mostly for HyperText Markup Language (HTML), and an AJAX spider for Javascript and XML-rich sites. The second part of ZAP is the scanner, which works in both a passive and an active way. The passive scan analyzes the responses of the web application in an attempt to find potential issues. The active scan, as the name implies, actively attacks the web application to find vulnerabilities.

4.2.3 SSLyze

Transport Layer Security (TLS) provides secure communications over a network, and it is nowadays used on most websites and web applications. It can guarantee that the connection between a client and a server is private, as all data is encrypted with symmetric cryptography, and it can enable the authentication of the identity of both parties involved in the connection through the use of public key cryptography, as well as ensure the integrity of each message resorting to message authentication codes.

SSlyze [Alb16] is an Secure Sockets Layer (SSL)/ TLS analysis tool, written in Python, that checks the configuration of SSL on a given host, in an attempt to find misconfigurations and problems with SSL. It checks certificates, cipher suites, insecure renegotiation, and commonly known SSL vulnerabilities such as Heartbleed, among others.

This tool is available for Linux systems and interacts with the user through the command line. It supports different configuration parameters, allowing to choose what needs to be tested. Another advantage of this tool, just like the ones already presented in this section, is the possibility to output the data into an XML file.

4.2.4 Hydra

Many servers offer different services, such as FTP, that require simple authentication through a username and password. Many times these services do not offer protection against brute force attacks. Brute forcing, in this situation, is attempting several different combinations of usernames and passwords so as to discover a valid combination. While this method is not refined nor discrete, many times it proves successful due to the usually simple credentials utilized by the majority of users.

Among the plethora of password cracking tools that exist, Hydra, John the Ripper and Medusa are the most popular ones. All of these are open-source tools that accept different parameters, can use passwords and usernames saved on dictionaries, or deliver a pure brute force attack where they attempt all possible character combinations up to a definable length. All have support for several different protocols, from FTP, HTTP or Hypertext Transfer Protocol Secure (HTTPS) to SSH, and are able to run several concurrent threads to enable a higher number of attempts per second. From these three, Hydra was chosen to integrate the system due to its better protocol support and superior performance [Hac16].

4.2.5 Ettercap

Man-in-the-middle attacks are a common practice when attempting to listen to encrypted communications, usually with the intent of acquiring confidential data. However, and in the scope of this work, a particular type of attack is a valuable addition. Address Resolution Protocol (ARP) poisoning consists in spoofing the ARP cache of the network hosts. In this case, it is used to route all network traffic through the device. This is needed in a switched network when the user wishes to capture network traffic.

Ettercap [Alb15] is one of the most well known man-in-the-middle attack suites, and is implemented in the system with the purpose described in the previous paragraph. Different from the other tools presented above, it does not incorporate the auditing part of the system, but integrates the traffic capture module described in 3.3. Along with the traffic sniffer, kit composes the core of this module.

4.2.6 SQLmap

While webscanners are able to detect most vulnerabilities in webservices, SQL injections can be more difficult to detect, and even harder to explore. Due to the relevance that this type of vulnerabilities have, the system should integrate a specific tool to detect them.

SQLmap [Ber16a] is a SQL injection scanner, capable of finding potential vulnerable fields in webpages and exploit them. SQLmap can, beyond discovering potential injections, perform the injections and dump databases. It has support for most of the common DBMS, from MySQL to Microsoft SQL Server.

4.2.7 Traffic Sniffer

A network or system administrator may have the need to check the type of traffic flowing through a network. Many times, specific traffic flows can be a sign of an attack or of an abnormal activity. As such, being able to capture packets for later analysis, or have a real-time classification of the type of traffic can prove to be a boon for the administrator.

Resorting to the `Libpcap` [Tim15] library, a lightweight traffic sniffer was implemented. It allows for swift packet classification or capture and subsequent storage in a file for a more thorough offline analysis (*i. e.*, through wireshark or other similar software). The tool was specifically constructed within the scope of this work to meet the conditions of the less powerful system in which it was going to run, and it was released as an open open-source project in [Ber16c].

4.2.8 Aircrack-ng

Nowadays, wireless networks are as ubiquitous as wired networks, perhaps even more. Mostly because of this, many tools capable of capturing packets or crack weak security wireless protocols have been developed. Aircrack-ng [Air16a] is one of those tools. It aggregates several utilities regarding wireless networks, and works with a wide variety of wireless adapters. Through Aircrack-ng, it is possible to capture traffic, or attempt to crack a wireless network protected with a vulnerable protocol, such as WEP.

4.3 Automation

After identifying the most suitable tools, the next step consisted of their integration, automation and orchestration to perform an audit. Several steps were taken towards automation, beginning with the definition of the parameters for each tool. This was made while striving to encompass the largest amount of vulnerabilities possible, while keeping the execution time within reasonable operating intervals. The second step was the definition of the order in which tools were to be invoked and their input/output data. This was particularly relevant in the context of how the parsing of the network mapping was to be performed. The last step was the automation in itself, either through API calls or by executing commands directly. The automation and orchestration scripts were all developed within the scope of this project.

4.3.1 Nmap

As the first step of the audit is the detection of network hosts, Nmap is the first automated tool in the system to be run. As the tool also performs OS fingerprinting, to attempt to identify the OS running on the host, and service detection, it fulfills the service version detection part as well.

In terms of parameters, `-sV` and `-O` add the service detection and OS fingerprinting functionalities, respectively. Due to potential time constraints, the `-T3` option was also added to avoid the mapping to take a significant amount of time. From performed tests, utilizing this option in detriment of a more thorough scan had no influence on the performed detection.

For the automation part, as the scanning should be automatically started on a network connection, a small bash script was created that calls Nmap on detecting a network connection. This is done both when a network cable is plugged in or the device connects to a wireless network. The output is saved into an XML file. From this file, and with recourse to the `LibXML2` library [Dan16], the file is parsed into a structure where all relevant host data is kept. A small excerpt of the script that is called on a network connection is presented in 4.1.

```
...
cidr=$(ipcalc −b $ip $nm | grep Network | cut −d":" −f2)
if [ "$IF" == "eth0" ]
then
    case "$2" in
        up)
        logger −s "NM Script up triggered"
        sudo nmap −T5 −oX /var/www/html/nmap−output.xml −O −sV $cidr
        ...
```

Listing 4.1: Excerpt of the script that invokes nmap.

4.3.2 OWASP ZAP

From the data acquired in the network mapping part, ZAP is called to scan the hosts where web services are detected. Since ZAP is a webscanner, it will detect different types of vulnerabilities, ranging from Cross Site Scripting (XSS) injections to Cross Site Request Forgery (CSRF). The tool usually presents a GUI for easy interaction and usage. Nonetheless, in this particular case, the integration with the remaining components was made through its Python API, being called individually for each scanned host. A new session is created for each host, executing the spider functionality first, only then to be followed by the scan. Potential higher risk alerts (low risk alerts are ignored, since they are normally considered non-threatening, and would only make the report harder to read) are then exported into an XML file for later parsing.

In terms of parameters, ZAP is run in a straightforward fashion and mostly using its default parameters. The spider was nonetheless setup not to run for the full length scanning, due to time related concerns. This choice is the result of several (unit) tests during integration, so as to be certain that it would have minimal impact in terms of detection. The spider is setup to run until the 35% mark only. The scanner is allowed to run its full length.

An example of the use of the API to start ZAP through a `Python` script is presented in Listing 4.2.

```
subprocess.Popen(['/root/Documents/AuditControllers/ZAP_2.5.0/zap.sh','-daemon', '-port','8090'],stdout=open(os.devnull,'w'))
print 'Waiting for ZAP to load, 20 seconds ...'
time.sleep(20)

# Specify the URL to start the attack
TARGET = "http://"+sys.argv[1]+"/"

print "Attacking %s with ZAP" % TARGET

zap = ZAPv2(proxies={'http': 'http://127.0.0.1:8090', 'https': 'http://127.0.0.1:8090'})
zap.core.new_session(apikey = 'mbml8683889jr88up9s872p9b2')
zap.urlopen(TARGET)
```

Listing 4.2: Excerpt of the Python script that invokes ZAP.

4.3.3 SSLyze

If one of the detected services is an HTTP server, and if it supports HTTPS, SSLyze is automatically invoked to audit the SSL security of the server. SSLyze is initialized with the `-regular` parameter, so as to assess the most common SSL issues. The output is stored into an XML file, and the parsing processes the data to present potential issues, namely if the certificate issuing and expiration dates are still valid, if the host name coincides with the one of the system and the one on the certificate. It also reports on other SSL specific misconfiguration problems and commonly known vulnerabilities, such as cipher suite downgrading and Heartbleed.

An example of the programming logic that invokes SSLyze is presented in listing A.1, included in the appendix. The snippet of code only shows the call for the script that runs SSLyze after verifying the host is running HTTPS. SSLyze itself runs through a `Python` script.

4.3.4 Hydra

Due to its nature, Hydra is only called when certain services are detected on a host (e.g., protocols requiring authentication). The system is shipped with a dictionary to be used on the attacks, though the user can supply its own dictionaries via the web interface also.

The automation script developed for this tool performs a verification on the detected services, as emphasized by the code snippet in listing 4.3. The services that spawn this tool are FTP, MySQL and SSH. When one of these services is found in the output of Nmap, Hydra is called using the provided dictionaries or the default one provided with the system. This function of the system is one of the most time consuming, due to the way a brute force or dictionary attack works. This is also why the number of services is only 3. They are some of the most commonly vulnerable [Dan13] [Den13] [Den16] and if successfully exploited, they can provide access to sensible information (or to the entire system, in the case of SSH). A small example of how the call of Hydra is made is shown in 4.3.

```
 ...
   if(strcmp(hosts[i].ports[j].serviceName, "ftp") == 0)
   {
  char *hydraCommand = concat("hydra -L users.txt -P passwords.txt ftp://",hosts[i
     ].address);
  ...
   }
   else if (strcmp(hosts[i].ports[j].serviceName, "ssh") == 0)
   {
  ...
   }
   else if (strcmp(hosts[i].ports[j].serviceName, "mysql") == 0)
    ...
```

Listing 4.3: Small excerpt of the script invoking Hydra on available hosts.

4.3.5 Ettercap

Ettercap is not part of the main auditing done by the system. As a Man-in-the-Middle attack suite, it is used mainly for the ARP poisoning functionality, with the purpose of capturing traffic in a switched network. In this case, the call is not done on an automated way, but manually triggered by the user. Only the parameters themselves are predefined. By default, Ettercap is called with the `-Tq -w dump -M arp:remote` //// parameters, where `-Tq` calls the tool in text-only mode and using quiet mode (it does not print packet contents), `-w dump` writes the captured traffic file into a file called `dump`, `-M arp:remote` // designates ARP poisoning as the attack, where `remote` enables sniffing of remote traffic the hosts make through the gateway, and the //// defines the poisoning for all hosts on the network.

4.3.6 SQLmap

SQLmap is perhaps the most focused tool integrated in the forensic box. While many other tools, namely ZAP, are able to detect SQL injection related problems (and even exploring them), this tool is focused only on the exploration of this type of vulnerability. Its main advantages are that it is capable of discovering potential injection-vulnerable fields and then launch different exploits in an attempt of finding a more precise security breach.

The main issue with the automation of SQLmap is that it requires the URL of the page with the HTML fields. It is not capable of automatically detect all URLs that a webpage may have. As such this detection is performed with the spider of ZAP and an additional file is kept with all URLs of a specific host, which is then fed to SQLmap. When service detection identifies the type of DataBase Management System (DBMS) existent on the system, it is passed on to SQLmap also, so that injections specific to the other DBMSs can be bypassed, and thus reduce the auditing time.

The tool is run with the option to detect fields and immediately attempt the exploit. An excerpt of code triggering SQLmap is shown in listing Listing 4.4. As can be seen, the tool is initialized with the `--dbms` option, so as to set up the target DBMSs. The port is also provided and the output is redirected to the file named `sqlmap-output-HOST_ADDRESS.txt`, where `HOST_ADDRESS` is a placeholder for the IP address of the target host.

```
...
if(strcmp(hosts[i].ports[j].serviceName, "http") == 0)
{
  for(k = 0; k < hosts[i].nPorts; k++)
  {
    if((strcmp(hosts[i].ports[k].serviceName, "mysql") == 0) || (strcmp(hosts[i].
        ports[k].serviceName, "postgresql") == 0) ||
...
```

```
    {
      db = 1;
      char *sqlCommand = concat(concat("sqlmap -m ", concat(hosts[i].address,"-urls.
          txt")), concat(" --dbms=",strcmp(hosts[i].ports[k].serviceName), concat(
          concat(concat(" --smart --forms --batch -o > sqlmap-output-", hosts[i].
          adress),".txt "), "2>&1")));
      system(sqlCommand);
...
```

Listing 4.4: Small excerpt of a script that starts and directs SQLMap towards available hosts.

4.3.7 Aircrack-ng

The module that integrates Aircrack-ng is used in WEP cracking attempts. This module is only triggered when a network with such protocol is in the vicinity. The detection of the wireless protocol per se is not performed by Aircrack-ng, but by the `iwlist` tool, which is invoked on a regular basis, until a WEP protected network is found. When such network is found, the wireless network interface is placed in monitor mode, so that the card can listen to any packet transmitted wirelessly. The tool then proceeds with the injection of packets towards the access point. This is done so that a large number of Initialization Vectors (IVs) is generated, which are later on captured. The device then attempts to make a fake authentication with the access point, with the objective of leading the access point into the acceptance of frames with the Media Access Control (MAC) address of the device. The last two steps are the capturing of ARP packets and their re-injection in the network, so that more IVs are generated, consequently leading to obtaining the WEP key of the network. A list of the commands and steps taken comprising the described procedure is shown in listing Listing 4.5, where `ESSID` and `myMAC` are placeholders for the real values. The options used are typical for the attack under analysis and their description can be found, e.g., in [Air16b].

```

iwlist wlan0 scan | grep 'ESSID\|IE: W\|Address\|Frequency'
airmon-ng start wlan0 9
aireplay-ng -9 -e ESSID -a Address ath0
airodump-ng -c channel --bssid Address -w output-ESSID ath0
aireplay-ng -1 0 -e ESSID -a Address -h myMAC ath0
aireplay-ng -3 -b Address -h myMAC ath0
aircrack-ng -b 00:14:6C:7E:40:80 output*.cap
```

Listing 4.5: Steps taken to obtain a WEP key.

4.4 Conclusions

This chapter presented several tools that were integrated in the device prototyped in the scope of this work and also the steps taken towards their automation. Each tool follows similar process in terms of automation, though their specificity had to be dealt with individually. While some of the tools provide APIs, simplifying the process of integration and automation, or allow outputting in a well-behaved XML file, others must be invoked directly from the command line, and their outputs saved in a file for dedicated processing.

A report can be generated to be presented to the user compiling the data generated with the automated processes of these tools, providing the appropriate interfaces. Some of the tools presented in this chapter can also be manually operated through these interfaces. Both these functionalities are presented in the following chapter, along with the interfaces design and report structuring. Though several (unit) tests were performed along the integration and automation phase, the system as a whole needs to be validated in a real scenario. This will be the subject of chapter 6.

Chapter 5

Interface and Reporting

5.1 Introduction

This chapter presents the interfaces available for the interaction between the user and the prototyped device, as well as the functioning of the reporting process for conveying the audit results. The previous discussion emphasizes that the user will need simple means to access and control the system, and all the different functionalities presented in chapter 4, and that the data coming from all pentesting tools needs to be presented in a very clear manner. These requirements led to the creation of two different interfaces to the system: (i) a web-based interface, where the user can both consult results from an audit, or launch and configure the majority of the system tools; and (ii), a hardware interface, where the user can issue simple commands and receive basic feedback on the status of an operation. Both interfaces and their implementation are presented in section 5.2. Section 5.3 defines the system reporting, providing details on how the data is parsed and presented to the user, and the overall structure of a report.

5.2 Interfaces

The software and hardware interfaces allow the user to interact with the device and receive both feedback from the execution of the tools and output data from audits. They have mostly distinct functions, with only a few overlapping, and they both are implemented with user-friendlyness in mind. Though access through SSH is possible also, it is not practical nor user-friendly, and no additional programming logic was added to the panoply of already available commands and scripts for this particular protocol. The hardware interface is presented in subsection 5.2.1 and the web interface in subsection 5.2.2.

5.2.1 Hardware Interface

The hardware interface is based on the RGB1602 module, briefly described in chapter 3. This module connects to the Raspberry Pi through its GPIO pins and offers a small Liquid Crystal Display (LCD) display with two lines of effective output and five hardware buttons, four of them positioned in a D-pad shape, which is optimal for menu navigation. The LCD is used to output simple text messages, which typically rotate from the right to the left.

The module integrates the MCP23017 expansion chip. To enable communication with the module and the chip, a library called WiringPi52 [Gor16a], with support for the components, was used.

It is possible to directly access the LCD to display messages and capture button presses through the several methods provided by the library.

The programming logic behind the hardware interface makes it possible to display status messages and access to a simple menu for issuing direct commands. When an audit is in progress, the LCD displays the message `Audit in Progress`. Once it is finished, it displays `Audit Finished`. An example of how text output to the display is done can be seen in Listing 5.1.

```
wiringPiSetup();
    mcp23017Setup (100, 0x20);
    printf ("Raspberry Pi\n");
    for(i=0;i<16;i++)
        pinMode(100+i,OUTPUT);
    digitalWrite(101,0);
    display=lcdInit(2,16,4,100,102,103,104,105,106,0,0,0,0);
    lcdHome(display);
    lcdClear(display);
    lcdPosition(display,0,0);
    lcdPuts(display,"Audit in Progress");
    pinMode(0, OUTPUT);
    pinMode(2, OUTPUT);
    pinMode(3, OUTPUT);
```

Listing 5.1: Output to LCD display example.

When the system is capturing traffic, the LCD display is also periodically updated to show the main type of traffic being captured (e.g., Transmission Control Protocol (TCP), User Datagram Protocol (UDP), etc.). This feature was added because it might come in handy during audits to a single computer, in which case it will be possible to detect, for example, if a virus is sending HTTP requests when all applications are shutdown.

In terms of operations directly accessible through the module, the user can: (i) launch a new, full audit on the connected network (for cases when new hosts were connected and the user wishes to scan again); (ii) begin packet capturing (to a file or only to check the main type of traffic passing in the network); or (iii), launch an ARP poisoning attack. The selection is done through the hardware buttons, following the scheme shown in figure 5.1.

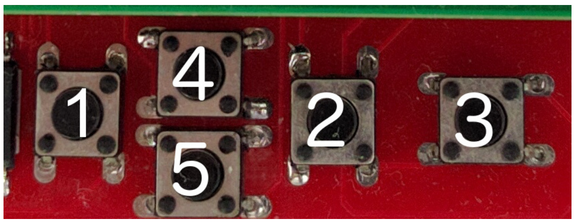


Figure 5.1: The buttons module placed on top of the Raspberry Pi. Button 1 and 2 allow for navigation when choosing different functionalities, while buttons 4 and 5 cycle through options for a specific functionality. Button 3 works as a selection button.

The access to reading button pressing values is also provided by the WiringPi library. An example on how a button is read is presented in Listing 5.2.

```
int pressedButton()
{
    int i;
    while(1)
        for(i = 108; i < 113; i++)
            if(digitalRead(i)) return i;
}
```

Listing 5.2: Function that captures the pressing of a button.

A small example on how the option selection is processed is given in Listing 5.3. As can be seen in the excerpt of code, `ettercap` is being used as the tool for poisoning the network.

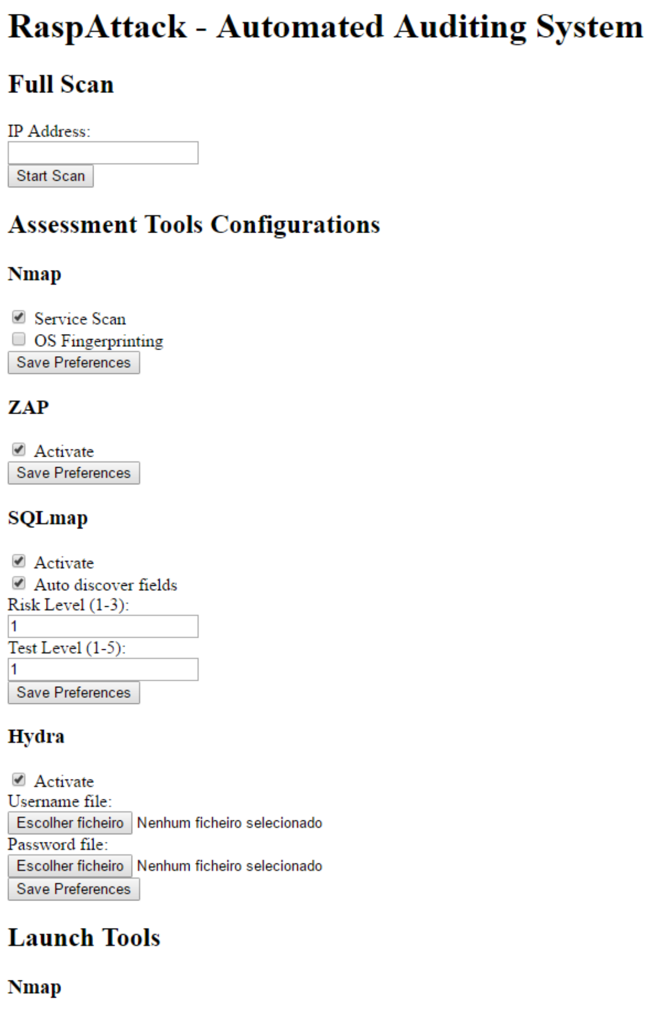
```
        int button = pressedButton();
        switch (button)
        {
            case 108:
                lcdPosition(display,0,0);
                lcdPuts(display,"Network will be poisoned.");
                lcdPosition(display,0,1);
                lcdPuts(display,"Are you sure? Other button to cancel");
                while(1){
                    button = pressedButton();
                    switch (button) {
                      case 108:
                            system("ettercap -Tq -w dump -M arp:remote //");
                            while(1)
    ...
```

Listing 5.3: Excerpt of menu structuring on the module.

5.2.2 Web Interface

The web interface gives access to the majority of features and results. From this interface, the user can define parameters, launch audits, launch specific tools on an assortment of targets, consult the complete report and have access to the different output of the tools.

The interface is implemented in HTML and PHP: Hypertext Preprocessor (PHP), where the user preference submissions are stored in a file, read by the auditing scripts. From the user preferences, different parameters can be chosen, and tools can be excluded from the automated process. The user can also launch some of the tools directly from the interface, with their output being directed to the report page. Also provided are a link for the report, and the option to download captured traffic in the shape of a pcap file. A screenshot demonstrating a small part of the interface is depicted in Figure 5.2.


RaspAttack - Automated Auditing System

Full Scan

IP Address:

Start Scan

Assessment Tools Configurations

Nmap

☑ Service Scan
☐ OS Fingerprinting

Save Preferences

ZAP

☑ Activate

Save Preferences

SQLmap

☑ Activate
☑ Auto discover fields

Risk Level (1-3): 1

Test Level (1-5): 1

Save Preferences

Hydra

☑ Activate

Username file: Escolher ficheiro Nenhum ficheiro selecionado

Password file: Escolher ficheiro Nenhum ficheiro selecionado

Save Preferences

Launch Tools

Nmap

Figure 5.2: Screenshot of part of the web interface.

5.3 Reporting

When all data is gathered and an audit is finished, the user can consult a digital document where all the information is presented in a structured and simple to read manner. Most tools output data either in XML format or directly to the command line. Within the system, the tools outputting to the command line have their `stdout` redirected into text files, so that the data is always stored and identified. The files are named according to the targets of the audit, meaning they all use the IP address of each host as an identifier.

For building up the final report, the data is parsed from the text files and XML outpus and rearranged in a concise manner. For parsing the XML files, the `LibXML2` [Dan16] is used. An excerpt of the code for parsing of a ZAP output is given in Listing A.2, included in the appendix to keep this explanation shorter. The aforementioned listing only presents the parsing of the nodes in the XML file. The complete treatment of the parsed data is handed out to a different function, where each node is processed. A better insight on how the node processing is performed, in this case for the output of SSLyze, is shown in Listing A.3.

The gathered data is compiled into structures, part of which can be seen in Listing 5.4. These structures are then compiled into a final `txt` file to be presented to the user. This structure, defined in `C` programming language as many of the remaining scripts and logic, contains data concerning each host and all the relevant information found by each tool. The structures is used not only to store data for the report, but also to feed data into the different tools during the auditing process.

```c
typedef struct{
  char *name; char *risk; char *attack; char *evidence; char *description;
}hostAlert;

typedef struct{
  int pNumber; char *protocol; char *service; char *serviceName; char *
      serviceVersion;
}port;
```

Listing 5.4: Part of the data structure that encompasses information of the hosts.

Figure 5.3 exemplifies the way a report is structured for each host found and probed by the device. The report encompasses the main information on the encountered hosts, and details on the services running, their versions and potential vulnerabilities. Note that this report does not include results of other tools that the device integrate. For example captured network traffic is located on a file that a user can download, and it is not available in the report.

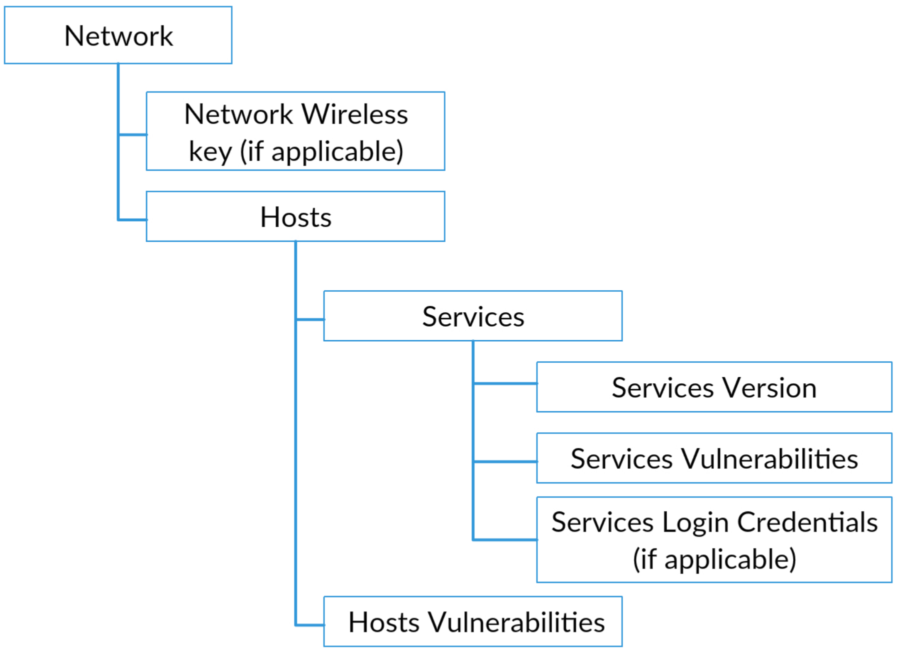


Figure 5.3: Scheme detailing the structure of a report.

The report starts by showing information regarding the wired or wireless network, such as the network name (IP address and network mask) and wireless key, if applicable. For each host, it will then present the IP address, OS, services and vulnerabilities. Vulnerabilities are shown at the host and at the services level. A full report of the system can be found in the appendix, namely in Listing B.1.

5.4 Conclusions

This chapter elaborated further on the approaches taken to enable the user to communicate with the forensic box, and how the different data is gathered and treated before being made available. Since the interaction with the device was to be non-intrusive and simple, this phase comprised one of the most important ones for the project at hands. Two different interfaces were developed: an HTTP based and an hardware interface. From the interfaces, the user can interact with the device on different levels, namely fine-tunning and launching tools and audits, or obtain feedback on the status of some operations. From the reporting, it is possible to have a concise and comprehensive view on the state of the audited system(s).

The following chapter will describe some tests performed on the system to assess its efficiency, as well as adjustments made to improve its functioning. A comparison with a manual audit done on real servers.

Chapter 6

Testing, Fine-Tunning and Security Audits

6.1 Introduction

This chapter is focused on the fine-tuning of the system and on tests made to evaluate its performance. Fine-tunning was necessary at a later stage of the project to make sure that the device was not taking too much time while performing its main tasks. A comparison of the outputs of the device with the ones obtained during a human assisted audit is also included herein. This comparison was performed to evaluate, to a certain extent, if the detection of the main vulnerabilities was on par with the ones detected during human analysis.

After completing the implementation and initial configuration of the device, different tests were performed to assess its correctness in terms of functioning. The results of these tests are presented in section 6.2. During these tests, several tweaks and changes to the parameters of the tools had to be made in an attempt to improve detection rates in some key areas. These changes and their results are explored in greater depth in section 6.3. Section 6.4 compares the results of real life audits performed with the human assistance and with the device on several servers of the university (whose addresses or configurations were not disclosed, due to security reasons). Loosely speaking, these results can be understood as a benchmark for the success rate of the device when compared with an actual expert directly utilizing different tools.

6.2 Testing

To assess if the main functionalities of the device were working correctly, namely the detection of vulnerabilities and reporting, different existing servers were used. Most of these servers are purposely configured for penetration testing. Local machines were also used with that objective. The tables contained in this section show the different vulnerabilities and issues exhibited by different servers, and the capability of the device to detect (or failure to detect) those vulnerabilities or issues.

The first test performed on the system concerns the mapping of the network, and the detection of open ports and available services. This test was performed on a local network, consisting of the Raspberry Pi itself, running Kali Linux, a laptop computer, running Windows 10, a set-top box, a router and an Optical Network Terminal (ONT) from an Internet service provider. Table 6.1 summarizes the results concerning the identification of the devices and the enumeration of their ports and services. The hosts identified in the table are the Raspberry Pi (in the first row), the

Table 6.1: Results concerning the network hosts identification, and detection of open ports and associated services.

Host	Ports	Services	OS
.71	22, 80	ssh, http	Linux
.77	8080, 8086	http-proxy, d-s-n	Linux
.84	22, 443, 902, 912, 2869, 5357	ssh, https, iss-realsecure, apex-mesh, icslap, wsdapi	Windows
.253	22, 80, 139, 443, 515	ssh, http, netbios-ssn, https, printer	Linux
.254	21, 23. 53, 80, 443, 1723, 8000	ftp, telnet, domain, http, https, pptp, http-alt	Router

The order of the services running in each port correlates with the port order presented on the table directly.

set-top box (in the second row), the laptop computer (third row), the ONT (fourth row) and lastly the router (fifth row). As this is a known and fully controlled network, verification that the device detected all existing hosts and their open ports and services can be done easily. The OSs are also correctly detected (for example, the router detected as host `.254` is indeed a Thomson with the model TG799vn). The results were obtained by plugging the Raspberry Pi into the switch that is directly connected to the router, and letting it perform the detection autonomously.

The next step consisted in testing the detection of potential service vulnerabilities. In this case, web services containing different vulnerabilities were setup on a local machine to which the device was then connected to. The BodgeIt [Sim16], ZAP-WAVE [OWA15] and Awstats [Lau16] web applications were used as targets for these tests. The Awstats web application is a real application, whose older versions are known to suffer from different vulnerabilities. These web applications ran on a Virtual Machine (VM), using the VMWare virtualization software, with Elementary OS as the OS. The machine was configured with 4GB of Random Access Memory (RAM) and two dedicated processing cores (the used Central Processing Unit (CPU) was an Intel Core i7 4720HQ). All three applications have XSS vulnerabilities, while both BodgeIt and ZAP-WAVE have SQL injections, and ZAP-WAVE also includes URL redirection and remote file inclusions. Table 6.2 shows the detection performed by the device on the vulnerable web service.

Table 6.2: Results of the audit to vulnerable web applications.

Vulnerability	BodgeIt	ZAP-WAVE	Awstats
XSS	Y/D	Y/D	Y/D
External URL Redirect	N/ND	Y/D	N/ND
SQL Injection	Y/S	Y/S	N/ND
Remote File Inclusion	N/ND	Y/D	N/ND

Y - The vulnerability is present on the service; N - The vulnerability is not present on the service; D - The vulnerability was detected by the device; ND - The vulnerability was not detected by the device; S - Vulnerability suspected by the device but not discovered.

As can be observed, the main web vulnerabilities were detected. The SQL injection vulnerabilities were the exception in this case. During an initial phase of the scan, a potential injection vulnerability was discovered (hence suspected), but further testing was not able to discover a functional injection. Nonetheless, these results would suffice to trigger further investigation.

The tests that followed focused on testing SSL and related vulnerabilities detection. To achieve that purpose, a VM with several known SSL issues was used. This is the same VM that was used in the previous test, using Elementary OS. It was known *a priori* that the certificate was from an invalid authority and that it an older signature scheme was being used. Table 6.3 summarizes the results of these tests.

Table 6.3: Results of the audit targeting SSL issues.

Issue	**Detected**
Downgrade Attack	✓
Hostname invalid	✓
Weak Signature	✓
Self Signed	✓
Certificate Expired	✓

As can be seen in the table, the vulnerable server had a self-signed, expired certificate, an invalid host name (`www.example.com`), was susceptible to a downgrade attack (so the client can attempt to negotiate using an older, vulnerable SSL protocol version) and was using Secure Hash Algorithm 1 (SHA1) as the hash function for the signature. SHA1 is considered insecure nowadays and will stop being accepted by most browsers in the near future.

The last test performed in a controlled environment was that of the cracking capabilities of the device. In order to do that, a virtual machine with Elementary OS was setup with the SSH, FTP and MySQL services. These services were configured with passwords contained in dictionaries from dumps. The experiment assessed if the device was (i) automatically attempting to crack the logins of the detected service, (ii) if the attempt was successful or not and (iii), it measured the time the process took. Part of the results of this test can be found on Table 6.4.

Table 6.4: Results concerning the password cracking feature.

Service	**Password**	**Cracked**
FTP	123	✓
SSH	toor	✓
MySQL	password	✓

Since one of the objectives was to test if the device was starting the cracking attempt automatically, the passwords were chosen from the dictionary therein contained. This way, it was certain that any failure would not be due to that fact. During the audit, all passwords were successfully discovered, though the time spent on this part of the auditing process was significant. In a real world situation, it would certainly take more time than all the remaining parts together. Actually, based on additional experiments and given the hardware involved, it can be said that the procedure takes more than one hour when the password is located halfway on a 10000 password list file. This poses an issue in terms of the efficiency of the device and this issue has no easy solution, since the limitations of using a small, portable device such as the Raspberry Pi cannot be overcome.

6.3 Fine-Tunning

After performing different tests, some parameters and scripts were changed to improve the detection capabilities and reduce the time the device takes to perform the audit.

From the tests, it was possible to observe that SQL injection detection had a low success rate. At the time of the first SQL injection tests, SQLmap was simply being called for a given host, with the DBMS discriminated when the service was previously detected. The change done for this tool was already explained in chapter 4, where all URLs captured by ZAP are saved and then used with SQLmap. This improved detection, but did not solve all the issues. When a website utilizes a POST instead of a GET, SQLmap requires that the data on the forms is given as input. As such, in its present state, SQLmap is unable to detect SQL injections in fields that use POST.

Another issue that arose was that the PHP version is not detected, most of the times, when scanning for services. The auditor that performed the audits presented in the following section always issued a warning when PHP was out-of-date, which means that this was an important detail that needed to be addressed. It was solved by using `curl` to get an answer from the server where, most of the time, the PHP version is shown in a line similar to `X-Powered-By: PHP/4.3.2`. This line is then captured and included in the report.

It was noticed that the time to perform an audit increases considerably if a web scan is performed. On the other hand, if the device performs an attempt at login cracking, the time increases exponentially, and the audit takes several hours. While the problem mentioned in last could not be solved, as password cracking is a brute-force attack and the only way to increase its speed was to increase the power of the hardware, which goes against the design philosophy of the device, the web scanning time was reduced by limiting the depth of the spider. By limiting it to 50 or even 30% of the progress before beginning the scanning activity, the time of the audit was reduced considerably (from over 10 minutes per host where the web scan is executed to under 2 minutes) without affecting the detection performance of the device. From the executed tests, and after applying the aforementioned tweaks and adjustments, the attained results were the same as before the modifications.

6.4 Security Audits

UBI hosts a team of students and collaborators whose main objective is to deal with security aspects affecting the information technology infrastructure of the institution. This group performs security audits on predefined production systems on demand and on a regular basis. These audits are scheduled and performed over the real world systems. A senior member (the auditor) of the team performs the audits, eventually helped by a junior member. In order to test the usefulness of the developed device, it was used in one of the major audits performed in 2016, where several servers were tested. This section presents a comparison of the results obtained via both types of audit (human assisted and automated with the device) and a brief conclusion on the actual efficacy of the device on an uncontrolled environment is drawn.

Table 6.5 identifies the different servers and the vulnerabilities that were detected for each one of them both manually and automatically. Since these servers are actually publicly available, they are herein referred to by numbers, i.e., symbolically. As can be concluded from the analysis of the the table, most of the issues detected by the auditor were also detected by the device. The main exceptions are SQL injections and directory traversal (having a list with all directory paths of folders and files that comprise the website).

Table 6.5: Excerpt of audit results and comparison with the manual audit.

Host	Out-of-date Services				XSS	SQLi	Brute-Force	SSL issues	Other
	Apache	PHP	FTP	SSH					
1	Y/D	Y/D	N/ND	Y/D	Y/D	S/ND	N/ND	N/ND	Directory Traversal/ Y/ND
2	Y/D	Y/D	N/ND	N/ND	N/ND	N/ND	Y/D	Y/D	Dangerous Ports Open Y/D
3	Y/D	Y/D	N/ND	N/ND	N/ND	Y/ND	Y/D	Y/D	Dangerous Ports Open Y/D
4	Y/D	Y/D	N/ND	N/ND	N/ND	N/ND	N/ND	Y/D	N/ND
5	Y/D	Y/D	N/ND	N/ND	N/ND	Y/ND	N/ND	Y/D	N/ND

Symbols: Y - Detected by the auditor; S - Suspected by the auditor; N - Not detected by the auditor; D - Detected by the device; ND - Not detected by the device.

The following two listings contain excerpts of the reports produced by the device (Listing 6.1) and by the auditor (Listing 6.2). The first listing shows that the device correctly detected the open ports and the respective bound services, along with their versions. It also found the XSS vulnerability that the auditor also pointed out in e) of Listing 6.2. The automated system proved its usefulness after pointing out an out-of-date version of SSH, which the auditor failed to see. On the other hand, the auditor raised a suspicion on a potential SQL injection on Host 1, but he was not able to discover if it could be exploited or not during the audit. The device, being an automated machine, does not make assumption and only reports on the results it finds. This is an obvious advantage of humans on these tests, as they can flag a suspicion for further testing on a later date, or leave the verification for the system administrator, while the device simply does not report it. The main conclusion drawn from this experiment is that the automated audit should be complementary to the human assisted assessment, but with the great advantage of never forgetting tiresome tasks such as verifying obsolete versions for the services. Interestingly, the device was able to perform all tasks within the time frame that the expert took to perform the audit.

```
IP Address: xxx.xxx.xx.xxx
OS: MontaVista embedded Linux 2.4.17
——Open Ports———
Port Number: 22
Protocol: tcp
Service: OpenSSH
Service Name: ssh
Service Version: 3.6.1p2
——————//——————
Port Number: 80
Protocol: tcp
Service: Apache httpd
Service Name: http
Service Version: 2.0.46
——————//——————
Port Number: 1720
Protocol: tcp
Service: table
Service Name: h323q931
Service Version: (null)
——————//——————
PHP Version: PHP/4.3.2
—————//—————
———————————Alerts————————————
Risk: High
Name: Cross Site Scripting (Reflected)
Attack: lt;/scriptgt;lt;scriptgt;alert(1);lt;/scriptgt;lt;scriptgt;
Evidence: lt;/scriptgt;lt;scriptgt;alert(1);lt;/scriptgt;lt;scriptgt;
...
```

Listing 6.1: Part of the report produced by the device developed in the scope of this project during the audit on Host 1.

```
xxxxxxxx.xxx.pt (xxx.xxx.xx.xxx)
          a] Apache out—of—date version (2.0.46).
             Fix: Upgrade Apache for the latest stable version.
```

```
        b] PHP out—of—date version (4.3.2).
           Fix: Upgrade Apache for the latest stable version.

        c] System is vulnerable to the directory traversal.
           Target URL: [http://arqueotex.ubi.pt/db/funcoes/js].

        d] System appears vulnerable to SQLi (high possibility).
           Target: [http://arqueotex.ubi.pt/main.php?sortpesquisa=1].

        e] XSS Injection.
           Result: The system is vulnerable.

        Target1: http://arqueotex.ubi.pt/main.php?sortpesquisa={%injection_here%}
           Options: ['"><script>alert(document.cookie)</script'>]
...
```

Listing 6.2: Part of the report produced by the auditor during the audit on Host 1.

6.5 Conclusions

The experiments reported in this chapter clearly demonstrate the usefulness of the approach and device developed along this project. The overall performance was very satisfactory, achieving detection levels that are close to the ones of a human auditor and producing the results in the time frame of the audit. Actually, since the device is fully automated, lengthier audits may be left executing during less busy periods, though human monitoring is advised.

An automated approach for such complex scenario is still inferior to a human analysis, mostly because there are many possible paths for an audit after some point. As such, the report of the device may be seen as both complementary to and as an initial step towards a more detailed audit.

There is still room for improvement, as shown in section 6.3, where some tweaks and adjustments to make the device perform more efficiently were discussed. Nonetheless, the main objective of this master's project was achieved with a fully functional prototype. Improvements and main conclusions are discussed in the next chapter.

Chapter 7

Conclusions and Future Work

This final chapter presents the main conclusions of the work described in this dissertation, in section 7.1, and points out potential lines of research and development that may be used to improve this work in the future in section 7.2.

7.1 Main Conclusions

The objective of the work presented in this dissertation was to research and devise a system capable of detecting security issues on a network in an automated way. Chapter 2 presented some existing tools and works whose purpose was as similar to the one presented herein. Nonetheless, analysis of the literature and of these works revealed that most existing softwares and tools are either paid or not fully autonomous, emphasizing the gap that partially motivates this work. The aforementioned chapter also shed some light on network based security audits, establishing a basis on what a system for security audits should be capable of, and allowing for a better definition of its design and functioning.

The flow of work evolved to the definition of the hardware and software requirements, and to the structuring of the several tools that would allow the system to operate autonomously. The prototyping phase comprised the selection and configuration of the OS, installing of the security related tools and their parameterization, and the development of scripts and programs that glue all of them together. Some difficulties were felt during this phase, which were mostly due to the using only open source programs and code, which led to situations where documentation was scarce or non existent, and to some incompatibilities related with the usage of an ARM based device (the Raspberry Pi). However, it was possible to integrate all the tools selected after the prototyping phase, as shown in chapter 4. The OS image was made available in [Ber16b], so that anyone can immediately use the outputs of this work.

The interfaces provided and presented in chapter 5 were created so that they enable an inexperienced user to control the system and obtain feedback, without requiring an extensive knowledge on any of the tools tasked with the auditing nor on networking and security. The objective of building a plug-and-play device is partially achieved by these interfaces.

The proposed objectives for this work were achieved, though there is still room for additions and improvements. Network audits are very complex tasks and the time frame of the master's project is limited. By implementing several well known and community-verified open source tools, and connecting them to perform an automated work, it was possible to obtain satisfying results both in a closed, controlled environment and on a real network, as discussed in chapter 6. From the results obtained, it can be concluded that, for now, the major issues with the device automation are the time spent and the detection of SQL injections. The latter is mostly due to the usage of POST (instead of GET) in forms of many websites, where the injections are precisely to occur. Automation of the tools to perform SQL injection is easier when GET is used, since the injection is simply introduced in the URL and no additional, very granular, information regarding POST forms needs to be passed to, e.g., SQLmap. The current version of the system can only successfully perform automatic injection exploration when forms use the GET method. The other type of exploit requires human intervention. Many times, an experienced auditor will also exploit some of the vulnerabilities as a proof-of-concept, which the device does not do. In light of the tests performed, both in a controlled environment, where the true state of a machine/network was known, and in a real life audit, where the status of the servers was analyzed without prior knowledge of security state they were in, it can be concluded that the device and approach may comprise an added value for any institution or company.

An effort was made to fine-tune some parameters of the tools in order to favor the performance of the device, both in terms of detection and time consumed to perform an audit. These adjustments were discussed in section 6.3. Nonetheless, depending on the number of hosts on a network and the need to use some of the more time consuming tools, such as password cracking, the device may take up to several hours to perform an audit. This may not comprise a problem since the device can be left running autonomously up to the point of producing the report.

Even though the device is able connect to a cabled network automatically, wireless connection is not as straightforward, as it usually requires the user to input a key or password for the connection to be allowed. The current version of the system requires either connecting to the device through a cable, and accessing the device via SSH, or plugging in a monitor and keyboard, and use the graphical interface of the OS to perform the connection. This is not very practical when in need to quickly connect the device or when a computer is not accessible to configure the wireless card.

The report that the device output after an audit was also structured in the scope of this work. It presents the data from the different tools in an integrated manner through the web interface. In terms of out-of-date service versions, the device will only report the version of a service, and will not indicate if that version is the most recent one or not.

7.2 Future Work

Several functionalities and tools can be added to the system to improve its utility and value. On a purely functional side, an easier wireless configuration through the hardware interface, allowing the user to choose a nearby network and provide the password, requiring only physical access to the device and forgoing the need to connect via SSH, would comprise a significant improvement for the plug-and-play character of the device. Having more detailed information through the LCD screen, such as an estimate on the time to completion, would also comprise a valuable addition regarding the feedback to the user.

The integration of other tools, so as to actually allow the user to choose between a set of different tools for each task, increasing granularity in terms of the options on the automated audits, and giving an auditor the possibility to choose between the tools they are most accustomed with, is another possible line of future work. Adding more options for the user to choose from for each tool, including configuration parameters, is something that can be added with some work on the web interface, potentially as a side project.

The current version of the system does not integrate a framework such as Nessus or Metasploit. Configuring and executing such frameworks in an ARM system may comprise a challenge requiring extra effort, but it should be worth it. The possibility to actively exploit the audited systems may be beneficial in some cases, e.g., to assess to which extent a vulnerability can be exploited and for proof-of-concept.

Configuring the device to scan a network it is attached to on a set time interval and report any changes it finds between audits is an advanced functionality that may prove useful in the long run. With this functionality, the device can be used as a monitoring tool on a dynamic network, detecting changes in terms of hosts, services and vulnerabilities. It would be specially useful in cases where machines are being added or removed from the network on a regular basis.

Since it was out of scope, the updating of the tools and operating system of the device was not dealt with in this work. Nonetheless, assessing how the system can be updated without jeopardizing its automation and functioning constitutes and interesting (and potentially very challenging) line of future work. The solution may be to maintain a Linux based distribution as a long term project.

Bibliography

[Acu16] Acunetix. Web application security with acunetix web vulnerability scanner [online]. 2016. Available from: http://www.acunetix.com/vulnerability-scanner/ [cited 04 October 2016]. 22

[Air16a] Aircrack-ng. Aircrack-ng [online]. 2016. Available from: https://www.aircrack-ng.org/ [cited 21 September 2016]. 24

[Air16b] Aircrack-ng. simple_wep_crack [Aircrack-ng] [online]. 2016. Available from: https://www.aircrack-ng.org/doku.php?id=simple_wep_crack [cited 21 September 2016]. 29

[Alb15] Alberto Ornaghi, Marco Valleri. Ettercap [online]. 2015. Available from: https://ettercap.github.io/ettercap/ [cited 14 July 2016]. 24

[Alb16] Alban Diquet. isecpartners/sslyze: Fast and full-featured ssl scanner [online]. 2016. Available from: https://github.com/iSECPartners/sslyze [cited 23 September 2016]. 23

[Bea16] Beau Bullock. Black hills information security [online]. 2016. Available from: http://www.blackhillsinfosec.com/?p=5156 [cited 04 October 2016]. 8

[Ber16a] Bernardo Damele. sqlmap: automatic sql injection and database takeover tool [online]. 2016. Available from: http://sqlmap.org/ [cited 25 September 2016]. 24

[Ber16b] Bernardo Sequeiros. Automatedauditingsystem download | sourceforge [online]. 2016. Available from: https://sourceforge.net/projects/automatedauditingsystem/ [cited 07 October 2016]. v, xiii, 45

[Ber16c] Bernardo Sequeiros. Pcappacketsniffer - small program that captures packets and prints header information. [online]. 2016. Available from: https://github.com/Inthen/PcapPacketSniffer [cited 01 October 2016]. 24

[CA06] Mark Curphey and Rudolph Arawo. Web application security assessment tools. *IEEE Security & Privacy*, 4(4):32–41, 2006. viii, 6

[Cis03] Cisco Systems, Inc. Cisco networking academy program [online]. 2003. Available from: http://www.cisco.com/web/learning/netacad/demos/FNSDemo1_1/ch1/1_3_1/index.html [cited 25 September 2016]. viii, 7

[Cis16a] Cisco. Snort - network intrusion detection & prevention system [online]. 2016. Available from: https://www.snort.org/ [cited 25 September 2016]. 10

[cis16b] cisofy. Lynis - security auditing tool for unix/linux systems [online]. 2016. Available from: https://cisofy.com/lynis/ [cited 20 June 2016]. 9

[Cos16] Costa, Gianluca and Franceschi, Andrea de. Xplico - open source network forensic analysis tool (nfat) [online]. 2016. Available from: http://www.xplico.org/ [cited 02 October 2016]. 11

[Dan13] Daniel Cid. Ssh brute force - the 10 year old attack that still persists [online]. 2013. Available from: https://blog.sucuri.net/2013/07/ssh-brute-force-the-10-year-old-attack-that-still-persists.html [cited 25 July 2016]. 27

[Dan16] Daniel Veillard. The xml c parser and toolkit of gnome [online]. 2016. Available from: http://www.xmlsoft.org/ [cited 01 October 2016]. 25, 35

[Den13] Denis Sinegubko. Ftp brute force attacks? [online]. 2013. Available from: http://blog.unmaskparasites.com/2013/06/26/ftp-brute-force-attacks/ [cited 25 July 2016]. 27

[Den16] Denis Sinegubko. Atlas attack report - global mysql brute-force login attempts [online]. 2016. Available from: https://atlas.arbor.net/attacks/2001689 [cited 04 Octobers 2016]. 27

[Dug00] Dug Song. dsniff [online]. 2000. Available from: https://www.monkey.org/~dugsong/dsniff/ [cited 02 October 2016]. 10

[Edi16] Edimax. Edimax - wireless adapters - n150 - n150 wi-fi nano usb adapter, ideal for raspberry pi [online]. 2016. Available from: http://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/global/wireless_adapters_n150/ew-7811un [cited 15 August 2016]. 17

[Fam16] Famatech. Advanced ip scanner - download free network scanner. [online]. 2016. Available from: http://www.advanced-ip-scanner.com/ [cited 01 October 2016]. 21

[Gor16a] Gordon Henderson. Wiring pi [online]. 2016. Available from: http://wiringpi.com/ [cited 12 July 2016]. 31

[Gor16b] Gordon Lyon. Nmap: The network mapper [online]. 2016. Available from: https://nmap.org/ [cited 20 September 2016]. 21

[Gre16] Greenbone Networks GmbH. Openvas - openvas - open vulnerability assessment system [online]. 2016. Available from: http://www.openvas.org/index.html [cited 07 August 2016]. 10, 22

[Hac16] Hacker Target Pty Ltd. Brute forcing passwords with ncrack, hydra and medusa [online]. 2016. Available from: https://hackertarget.com/brute-forcing-passwords-with-ncrack-hydra-and-medusa/ [cited 01 October 2016]. 23

[HHP05] Jonas Hallberg, Amund Hunstad, and Mikael Peterson. A framework for system security assessment. In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, pages 224–231. IEEE, 2005. viii, 6

[Lau16] Laurent Destailleur. Awstats - free logfile analyzer for advanced statistics (gnu gpl). [online]. 2016. Available from: http://www.awstats.org/ [cited 04 October 2016]. 38

[Mid02] Paul Midian. Perspectives on penetration testing?black box vs. white box. *Network Security*, 2002(11):10–12, 2002. 5

[OWA15] OWASP. Owasp broken web applications project - owasp [online]. 2015. Available from: https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project [cited 04 October 2016]. 38

[OWA16a] OWASP. Owasp: Owasp top ten project [online]. 2016. Available from: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project [cited 18 June 2016]. 1

[OWA16b] OWASP. Owasp zed attack proxy project [online]. 2016. Available from: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project [cited 12 September 2016]. 22

[PI515] PI52. Rgb 1602(english) - 52 pi wiki [online]. 2015. Available from: http://wiki.52pi.com/index.php/RGB_1602(English) [cited 16 August 2016]. 17

[Pon15] Ponemon Institute LLC. 2016 ponemon cost of data breach study [online]. 2015. Available from: http://www-03.ibm.com/security/data-breach/ [cited 17 June 2016]. 1

[Por16] PortSwigger Ltd. Burp suite [online]. 2016. Available from: https://portswigger.net/burp/ [cited 04 October 2016]. 22

[PwC15] PwC. 2015 information security breaches survey [online]. 2015. Available from: http://www.pwc.co.uk/services/audit-assurance/insights/2015-information-security-breaches-survey.html [cited 17 June 2016]. 1

[Rap16a] Raphael Mudge. Adversary simulations and red team operations software - cobalt strike [online]. 2016. Available from: https://www.cobaltstrike.com/ [cited 17 June 2016]. 9

[Rap16b] Rapid7. Penetration testing software | metasploit [online]. 2016. Available from: https://www.metasploit.com/ [cited 17 June 2016]. 9

[Rap16c] Rapid7. Vulnerability management software, top rated tool | rapid7 [online]. 2016. Available from: `https://www.rapid7.com/products/nexpose/` [cited 04 October 2016]. 22

[Sim16] Simon Bennetts. psiinon/bodgeit: The bodgeit store is a vulnerable web application which is currently aimed at people who are new to pen testing. [online]. 2016. Available from: `https://github.com/psiinon/bodgeit` [cited 04 October 2016]. 38

[Sof16] SoftPerfect. Softperfect network scanner - fast and free network scanner. [online]. 2016. Available from: `https://www.softperfect.com/products/networkscanner/` [cited 01 October 2016]. 21

[Str16] Strategic Cyber LLC. Armitage - cyber attack management for metasploit [online]. 2016. Available from: `http://www.fastandeasyhacking.com/` [cited 17 June 2016]. 9

[Tan14] Andrew Tang. A guide to penetration testing. *Network Security*, 2014(8):8–11, 2014. 5

[Ten16] Tenable Network Security. Nessus vulnerability scanner | tenable network security [online]. 2016. Available from: `https://www.tenable.com/products/nessus-vulnerability-scanner` [cited 08 August 2016]. 9

[Tim15] Tim Carstens. Tcpdump/libpcap public repository [online]. 2015. Available from: `http://www.tcpdump.org/` [cited 25 May 2016]. 24

[Ver16] Verizon Enterprise Solutions. Verizon data breach investigations report [online]. 2016. Available from: `http://www.verizonenterprise.com/verizon-insights-lab/dbir/` [cited 16 June 2016]. 1

[WTS03] John Wack, Miles Tracy, and Murugiah Souppaya. Guideline on network security testing. *Nist special publication*, 800:42, 2003. 11

[XJYZ11] Rongrong Xi, Shuyuan Jin, Xiaochun Yun, and Yongzheng Zhang. Cnssa: a comprehensive network security situation awareness system. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 482–487. IEEE, 2011. viii, 5

Appendix A

Code Excerpts

This appendix presents some of the longer code excerpts mentioned in the body of the dissertation. They were included so as to provide more details on the implementation presented in chapter 4, mainly on how tools are invoked and on how the parsing of their output is performed.

```c
void launchSSLyze(host *hosts)
{
    int i, j;

    for(i = 0; i < nHosts; i++)
    {
        for(j = 0; j < hosts[i].nPorts; j++)
        {
            if((strcmp(hosts[i].ports[j].serviceName, "http") == 0) && (hosts[i].ports[j].pNumber == 443))
            {
                char *sslyzeCommand = concat("python sslyze_cli.py --regular ", hosts[i].name);

                char *sslyzeOutput;

                if(hosts[i].name != NULL)
                    sslyzeOutput = concat(hosts[i].name, "-output.xml");
                else
                    sslyzeOutput = concat(hosts[i].address, "-output.xml");

                sslyzeCommand = concat(sslyzeCommand, " --xml_out=");
                sslyzeCommand = concat(sslyzeCommand, sslyzeOutput);
                system(sslyzeCommand);

                streamSSLFile(sslyzeOutput, hosts, i);
            }
        }
    }
}
```

Listing A.1: Invocation of the SSLyze Python script from the service scanning module.

```
static void streamAlertFile(const char *filename, host *hosts)
{
    xmlTextReaderPtr reader;
    int ret;
    reader = xmlReaderForFile(filename, NULL, 0);

    if (reader != NULL)
    {
        ret = xmlTextReaderRead(reader);
        printf("%s\n", xmlTextReaderConstName(reader));
        while (ret == 1) {
            if((ret == 1) && (strcmp(xmlTextReaderConstName(reader), "alert") == 0))
            {
                hosts[nHosts-1].nAlerts++;
                hosts[nHosts-1].alerts = realloc(hosts[nHosts-1].alerts, hosts[
    nHosts-1].nAlerts*sizeof(hostAlert));
            }
            processAlertNode(reader, hosts);
            ret = xmlTextReaderRead(reader);
        }

        xmlFreeTextReader(reader);
        if (ret != 0)
            fprintf(stderr, "%s : failed to parse\n", filename);
    }
    else
        fprintf(stderr, "Unable to open %s\n", filename);
}
```

Listing A.2: Excerpt of the code that parses the XML output from ZAP.

```
hosts[i].sslIssues.SSLEnabled = 1;

    if((strcmp(name, "certificateChain") == 0) ||
        (strcmp(name, "hostnameValidation") == 0) ||
        (strcmp(name, "compressionMethod") == 0) ||
        (strcmp(name, "tlsFallbackScsv") == 0) ||
        (strcmp(name, "openSslHeartbleed") == 0) ||
        (strcmp(name, "openSslCcsInjection") == 0) ||
        (strcmp(name, "sessionRenegotiation") == 0) ||
        (strcmp(name, "notAfter") == 0))
    {
        if(strcmp(name, "certificateChain"))
        {
            if(strcmp(xmlTextReaderGetAttributeNo(reader, 0), "True") == 0)
                hosts[i].sslIssues.weakSignature = 1;
            else
                hosts[i].sslIssues.weakSignature = 0;

            if(strcmp(xmlTextReaderGetAttributeNo(reader, 1), "True") == 0)
                hosts[i].sslIssues.chainOrderOK = 1;
```

Listing A.3: Example of the processing applied to an SSLyze output file.

Appendix B

Full Report

This appendix contains a full report from an audit performed by the device. Its purpose is to provide a clear perspective on the information gathered by the device and how it is displayed and structured. It complements the discussion in chapter 5. Note that the IP addresses in the report were obscured since this audit was performed on a network that contains publicly accessible machines.

```
————————//REPORT//————————
Network: xxx.xxx.xx.x/24
——————————————//——————————————

Ip Address: xxx.xxx.xx.xx
MAC Adress: (null)
Vendor: (null)
OS: (null)
———Open Ports————
Port Number: 80
Protocol: tcp
Service: Apache httpd
Service Name: http
Service Version: 2.2.3
————————//—————————
Port Number: 443
Protocol: tcp
Service: Apache httpd
Service Name: http
Service Version: 2.2.3
————————//—————————
Port Number: 1720
Protocol: tcp
Service: table
Service Name: H.323/Q.931
Service Version: (null)
————————//—————————
```

```
————————Alerts————————
————//————
————————Password Cracking————————
————//————
————————SSL————————
SSLEnabled: 1
Certificate expired: 0
Hostname Correct: 1
Weak Signature: 0
Chain OK: 1
Vulnerable to compression: 0
Vulnerable to downgrade attack: 0
Vulnerable to Heartbleed: 0
Vulnerable to CSSInjection: 0
Vulnerable to renegotiation: 0
————————//————————

Ip Address: xxx.xxx.xx.xxx
MAC Adress: (null)
Vendor: (null)
OS: (null)
——Open Ports——
Port Number: 80
Protocol: tcp
Service: Apache httpd
Service Name: http
Service Version: 2.4.7
————//————
Port Number: 631
Protocol: tcp
Service: CUPS
Service Name: ipp
Service Version: 1.7
————//————
Port Number: 3306
Protocol: tcp
Service: MySQL
Service Name: mysql
Service Version: 5.5.50—0ubuntu0.14.04.1
```

```
————————//—————————
—————————————Alerts—————————————
Risk: High
Name: Cross Site Scripting (Reflected)
Attack: lt;/h2gt;lt;scriptgt;alert(1);lt;/scriptgt;lt;h2gt;
Evidence: lt;/h2gt;lt;scriptgt;alert(1);lt;/scriptgt;lt;h2gt;
Description: Cross—site Scripting (XSS) is an attack technique that involves echoing
     attacker—supplied code into a user's browser instance. A browser instance can
    be a standard web browser client, or a browser object embedded in a software
    product such as the browser within WinAmp, an RSS reader, or an email client.
    The code itself is usually written in HTML/JavaScript, but may also extend to
    VBScript, ActiveX, Java, Flash, or any other browser—supported technology.
When an attacker gets a user's browser to execute his/her code, the code will run
    within the security context (or zone) of the hosting web site. With this level
    of privilege, the code has the ability to read, modify and transmit any
    sensitive data accessible by the browser. A Cross—site Scripted user could have
    his/her account hijacked (cookie theft), their browser redirected to another
    location, or possibly shown fraudulent content delivered by the web site they
    are visiting. Cross—site Scripting attacks essentially compromise the trust
    relationship between a user and the web site. Applications utilizing browser
    object instances which load content from the file system may execute code under
    the local machine zone allowing for system compromise.

There are three types of Cross—site Scripting attacks: non—persistent, persistent
    and DOM—based.
Non—persistent attacks and DOM—based attacks require a user to either visit a
    specially crafted link laced with malicious code, or visit a malicious web page
    containing a web form, which when posted to the vulnerable site, will mount the
    attack. Using a malicious form will oftentimes take place when the vulnerable
    resource only accepts HTTP POST requests. In such a case, the form can be
    submitted automatically, without the victim's knowledge (e.g. by using
    JavaScript). Upon clicking on the malicious link or submitting the malicious
    form, the XSS payload will get echoed back and will get interpreted by the user'
    s browser and execute. Another technique to send almost arbitrary requests (GET
    and POST) is by using an embedded client, such as Adobe Flash.
Persistent attacks occur when the malicious code is submitted to a web site where it
    's stored for a period of time. Examples of an attacker's favorite targets often
     include message board posts, web mail messages, and web chat software. The
    unsuspecting user is not required to interact with any additional site/link (e.g
    . an attacker site or a malicious link sent via email), just simply view the web
     page containing the code.
————//————
——————Password Cracking——————
[3306][mysql] host: xxx.xx.xx.xx   login: root password: toor
————//————
——————SSL——————
SSLEnabled: 0
Certificate expired: 0
Hostname Correct: 0
Weak Signature: 0
Chain OK: 0
Vulnerable to compression: 0
Vulnerable to downgrade attack: 0
Vulnerable to Heartbleed: 0
Vulnerable to CSSInjection: 0
Vulnerable to renegotiation: 0
——————//——————

Ip Address: xxx.xxx.xx.xxx
MAC Adress: (null)
Vendor: (null)
OS: Linux 2.6.17 — 2.6.36
——Open Ports——
Port Number: 22
Protocol: tcp
Service: OpenSSH
Service Name: ssh
Service Version: 5.3p1 Debian 3ubuntu4
————//————
Port Number: 80
Protocol: tcp
Service: Apache httpd
Service Name: http
Service Version: 2.2.3
————//————
——————Alerts——————
————//————
——————Password Cracking——————
```

```
————————//—————————
———————————————SSL——————————————
SSLEnabled: 0
———————————————//——————————————

Ip Address: xxx.xxx.xx.xxx
MAC Adress: (null)
Vendor: (null)
OS: (null)
———Open Ports————
Port Number: 53
Protocol: tcp
Service: probed
Service Name: tcpwrapped
Service Version: (null)
————————//—————————
Port Number: 80
Protocol: tcp
Service: Apache httpd
Service Name: http
Service Version: 2.2.9
————————//—————————
Port Number: 139
Protocol: tcp
Service: table
Service Name: netbios—ssn
Service Version: (null)
————————//—————————
Port Number: 445
Protocol: tcp
Service: table
Service Name: microsoft—ds
Service Version: (null)
————————//—————————
Port Number: 1001
Protocol: tcp
Service: OpenSSH
Service Name: ssh
Service Version: 5.1
```

```
————//————
Port Number: 1720
Protocol: tcp
Service: table
Service Name: H.323/Q.931
Service Version: (null)
————//————
——————Alerts——————
————//————
——————Password Cracking——————
————//————
——————SSL——————
SSLEnabled: 1
Certificate expired: 0
Hostname Correct: 1
Weak Signature: 1
Chain OK: 1
Vulnerable to compression: 1
 Vulnerable to downgrade attack: 0
 Vulnerable to Heartbleed: 0
 Vulnerable to CSSInjection: 0
Vulnerable to renegotiation: 0
——————//——————
```

Listing B.1: A full report output of the device.