This degree project was performed with Inteno Broadband Technology
Inteno contact: Strhuan Blomquist

Configuration and device identification on network gateways

Konfiguration och enhetsidentifiering på nätverksgateways

SIMON KERS

Bachelor of Science in Engineering
15 ECTS Credits
Supervisor: Micael Lundvall
Examiner: Ibrahim Orhan
TRITA-STH 2013:22

KTH Royal Institute of Technology
School of Technology and Health
SE-136 40 Handen, Sweden
http://www.kth.se/sth

Abstract

To set up port forwarding rules on network gateways, certain technical skills are required from end-users. These assumptions in the gateway software stack, can lead to an increase in support calls to the network operators and resellers of customer premises equipment. The user interface itself is also an important part of the product and a complicated interface will contribute to a lessened user experience. Other risks with an overwhelming user interface include faulty configuration by the user, leaving the network vulnerable to potential attacks.

We present an enhancement of the current port forwarding settings, with an extensible library of presets. To help users with detecting available services, a wrapper for a network scanner was implemented, for detecting devices and services on the local network. These parts combined relieves end-users of looking up forwarding rules for ports and protocols to configure their gateway, basing their decisions on data collected by the network scanner or by using an applications name instead of looking up its ports.

Using the Nmap utility for identifying services on the network, could be considered harmful activity by network admins and intrusion detection systems. The preset library is extensible and generic enough to be included in the default software suite shipping with the network equipment. Working within the unified configuration system within OpenWrt, the preset design will add value and allow resellers to customize their services. This proposal could reduce support costs for the service operators and improve user experience in configuring network gateways.

Referat

Konfiguration och enhetsidentifiering på nätverksgateways

Vid vidarebefodring av portar i nätverksgateways, krävs ibland vissa tekniska förkunskaper av användaren. De höga kraven på kunskapsnivå kan bidra till ett ökat antal supportsamtal för återförsäljare och nätverksoperatörer. Användargränssnittet i sig är också en viktig del i produkten och ett komplicerat gränssnitt bidrar till försämrad användarupplevelse. Övriga problem med komplicerade användargränssnitt är risken för felaktig konfiguration, vilket kan utsätta nätverket för attacker.

En förändring av nuvarande *port forwarding*-inställningar presenteras, tillsammans med ett utbyggbart bibliotek med förinställda regler. För att hjälpa användare att identifiera enheter och sätta rätt inställningar, utvecklades en wrapper för en portskanner, vilken kan identifiera enheter och nättjänster i den lokala nätverket. Tillsammans underlättar dessa delar för slutanvändare, befriar dem från att referera till regler för portar och protokoll och möjliggör inställning enbart genom att använda portskanning eller välja namnet på tjänsten från en lista.

Användandet av verktyget Nmap för att identifiera nättjänster på nätverket kan möjligtvis betraktas som dataintrång av nätverksadministratörer och intrångdetekteringssystem. Databasen med förinställningar är utbyggbar, fungerar och passar in tillräckligt bra för att innefattas i standardmjukvaran. Via det centraliserade konfigurationssystemet i OpenWrt, kommer utformningen av systemet med förinställningar för port forwarding möjliggöra för komplementering av nättjänster och enheter från återförsäljare. Systemet kan minska supportkostnader för bredbandsleverantörer och bidra till en förbättrad användarupplevelse vid konfiguration av nätverksgateways.

Acknowledgements

A special thanks to Strhuan Blomquist of Inteno for teaching me how to walk like a hacker, instead of just talking like one. I would like to express my gratitude to Sukru Senli of Inteno, for helping me grok LuCI and perform Lua sorcery.

I'm grateful for the support of my supervisor Micael Lundvall, being an overall great guy and providing insightful comments on my work. Thanks to my examiner Ibrahim Orhan for giving great feedback, for making me focus on the task at hand and helping me with the disposition.

Finally I would like to thank everyone involved in free and open-source software for inspiring me to learn programming and teaching me best coding practices.

Contents

Chapter 1

Introduction

1.1 Background

There are simple ways in which to improve the user experience, developers of network gateway software often implement a set of presets of port forwarding rules for common applications. The interface presents the user with a list of application names and lets the user select an IP address, for which the forwarding rules should apply.

Working from the theory that a good user experience lies in the details, iterative improvements of the system is the preferred way to address it. This project contributes to the ongoing work of improving details of user experience, strengthening the product and saving costs for support. The project aspires to contribute to the continuous development of the OpenWrt ecosystem and aid users in configuring their gateways.

1.2 Goals

Simplifying configuration by abstracting common tasks for the end-user aims to relieve customers and technical support staff. Using automatic device identification and automating common tasks such as port forwarding, will result in savings in support along with a better user experience. This project will investigate the simplification of usability when configuring port forwarding and evaluate the changes. Many common support issues could be automated by the software running on the CPE and by effective communication with the end-user through the user interface, these improvements would also aid first-line support when guiding the customer over phone. In order to even begin configuring the device, the correct IP address to the CPE has to be typed in the web browsers address bar. A script which performs an automatic DNS translation from domain name to its LAN IP address will be developed. The project goals can be summarized as:

- Preset library of port forwarding rules
- Automatic detection of services on the local network devices
- Evaluate usability
- Internal DNS translation of the gateway IP address

1.3 Delimitations

One purpose of this degree project is to achieve a more user friendly operation by adding automatic service discovery on the local network within the OpenWrt system. A system which identifies the ports and services running on the network devices is developed, this works alongside the preset library for port forwarding, to reduce the amount and duration of support calls. These parts will work together, integrated into the web-based interface of OpenWrt. The internal DNS translation system will work independently from the rest of the project, it will not be dependant on either the preset library or the service detection.

It is assumed that operation of the device is user friendlier, when the end-user receive helpful hints and a more interactive workflow in the configuration process. The internal DNS translation for reaching the front-end of the CPE through an internal domain name, does not account for various browser implementations of the address bar, which could sometimes interpret a domain address as a search query. Evaluation of usability is meant to be qualitative and focuses on reason rather than test groups. The actual effect on support costs is out of the scope of this degree project.

1.4 Solutions

By building a library of presets for common port forwarding rules and developing a simple selection dialog, the end user can more efficiently set up port their firewall redirection rules and general configuration of their gateway. A limited range of settings and automatic portforwarding settings are presented to the user.

For the system of service identification, a wrapper around a port scanner is implemented, which performs a scan of the network nodes and returns a list of available services. This information is in turn used to match against the known presets and protocols, and offers the user a choice of applying the preset rules for the newly detected network device. The preset system is extensible, allowing retailers to add their own devices and services as preset definitions, each with their specific forwarding rules.

DNS translation of the gateway IP address, allows easy access through the web browser by simply entering a domain name in the address bar, which should be easier for users than remembering or figuring out its IP. This works by translating the gateways internal IP address to a domain name and keeping it up to date whenever the IP is changed.

These parts will be developed in an iterative process with support and suggestions from the company throughout the project.

Chapter 2

Current situation

Inteno Broadband Technology is a company that supplies *customer premises equipment* (CPE) for internet service providers. Their headquarters and *research and development center* is located in Stockholm, Sweden. Inteno Open Systems Platform, or *iopsys*, is a Linux-based open source platform running on their CPE. It is based on the OpenWrt distribution which targets embedded devices, specifically network gateways.[3]

The research and development department at Inteno works on improving the platform, adding value to the end users, the operators and the larger OpenWrt software ecosystem. By the nature of OpenWrt's free software licence[4], the code is publicly released and available for download from Intenos webpage.[2]

The resellers of Inteno gateways are mainly internet service providers, who then deploy the CPE among end users. Connectivity of the XBox 360 gaming console has been chosen as the reference device to do tests and verifications against. Based on previous information from technical support of service providers, one of the most commonly reported issues of end users, is setting up port forwarding for connecting their XBox 360 to the XBox Live network.

Chapter 3

Theoretical frame of reference

3.1 Design and layout

User interface and the graphical and sometimes physical design, is an important part of the brand and ease of use of products. Improving the graphical interface in electronic devices and its software, can cause irritations with the product and leave the user with a sense of hopelessness and self-blame.[12, p. 34] It is important to communicate a clear usage that is as unambiguous as possible to the user, this was decided to be the most important part of the layout. This was the main goal of the user experience, removing the possibility of misstep and communicating clearly, citing Krug's second law of usability[10, p. 41], "It doesn't matter how many times I have to click, as long as each click is a mindless, unambiguous choice". This lead to the idea of a port forwarding wizard.

3.2 Measuring improvements

For evaluating usability a more qualitative approach was chosen as opposed to testing on humans. The main reason for the more analytical approach is inexperience with putting together test groups and evaluating the results. This allowed for a more sequential approach in delivering the module and provided time to familiarize with the OpenWrt ecosystem before starting work on implementing the user interface. At the request of Inteno, the basic usability scenario chosen for evaluation was configuring port forwarding of the XBox 360.

Disadvantages of not using usability tests with actual people, includes developers missing out on valuable feedback from the people they are designing it for. The iterative process of producing a prototype, having users test is and improving the design based on feedback is lost. The main purpose of the usability improvements presented in this project, is presenting a more guiding and interactive approach, resulting in fewer mental steps.

We feel the method chosen works and is suitable for a project of this scope

3.3 Alternative using a locally run application

To test the newly applied configurations, web-based or locally run port scanners can be used, as opposed to a gateway-centric port forwarding solution presented by this report. They will scan the users external IP address for open ports and present which are open, however this does not guarantee that the packets are routed to the correct internal address.

Alternative solutions to simplifying port forwarding include using standalone applications which run on a PC, connected to the local network. These applications also use internal lists of port forwarding rules for common applications and devices, which is then applied for a specific IP address on the local network.[8]

Chapter 4

Research

End users of Inteno CPE have expressed concern about the relative difficulty of port forwarding and configuration of their network gateway. The default settings page for port forwarding is currently located under the *Firewall* tab in the OpenWrt front end, the forwarding procedure involves looking up ports for the specific device or unit, and entering these on the web page forms.

These set of rules sometimes involve several ports and protocols, increasing the possibility for misstep and faulty configuration by the end user. If we could reduce the complexities of this common task of address translation and present it in an way that are easily understandable, then customer satisfaction would increase. Such tasks could be well suited for automation by software, especially for applications and devices which require several port forwarding rules, automating some of these steps will save time and bring overall value to the user experience.

Currently the interaction with the web interface requires the user to enter the gateways IP address in the web browsers address bar. This potential barrier to access the web interface could be lowered by using address-to-name mapping, local to the internal network, much like how DNS functions on the global internet.

4.1 Current firewall tab

There are a few issues with the current firewall tab for novice users. It requires the user to acquire all the ports necessary for the network service and in the case of our reference unit, this increases the chance of missteps and faulty configuration. Another issue that has been identified are the tedious steps involved in applying the rules through the web interface. For the device or network service to function properly, the user has to add a forwarding rule for each required port. A screenshot of the port forwarding form in the web interface is shown in figure 4.1, in the case of our reference unit, the XBox 360, this requires four such steps. The mental steps and actions within the gray area of figure 4.2, will form a loop every time the service has several protocols and ports. The general idea is reducing as many of these harder choices or replacing them with simpler ones.

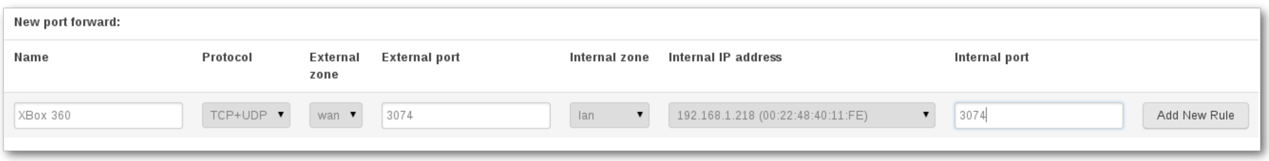

Figure 4.1. Current port forwarding dialog, the user is burdened with identifying and applying rules for every port of the service.

4.2 Guided firewall tab

In the projects improved port forwarding dialog, the user is confronted with fewer mental steps. In figure 4.2, we see a conceptual flowchart of the all the steps in the current user interaction when port forwarding, the two parts called “Look it up!” requires a switch of context which can delay the configuration of the port forwarding.

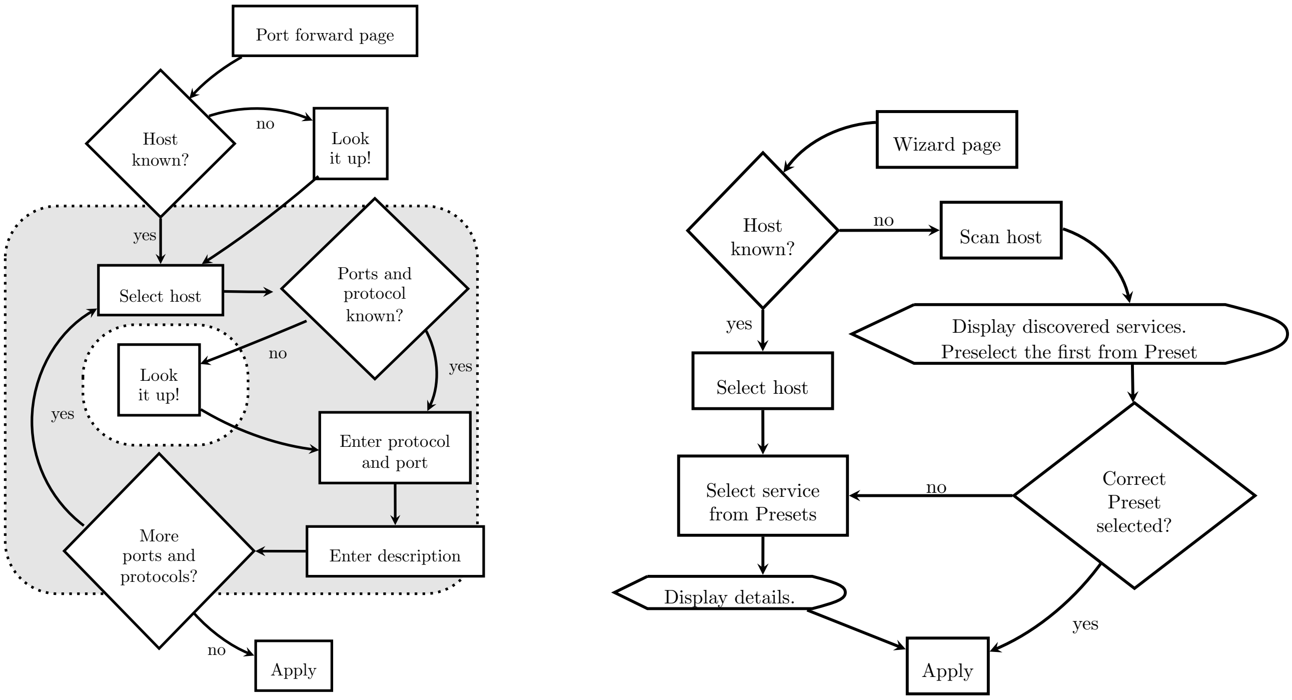

Figure 4.2. Usability flowchart for configuring port forwarding, illustrating differences in user interaction with the conventional port forwarding tab (left), to the one presented in this paper (right).

In addition to the “Look it up!” steps, in the example of the XBox 360 – which has four ports – the user have to perform seven additional actions for the three remaining ports:

- Select host
- Enter protocol and port
- Enter description

4.3 Programming languages

4.3.1 Lua

Lua is a programming language that is intended to be embeddable and extensible, it is implemented in C and enables the developer to employ different programming techniques with its multi-paradigm approach. The programming languages it is distributed with a permissive free software licence[5], while being open source allowing use within proprietary software. The language is dynamically typed, which means that the underlying variable type is determined at runtime and supports features such as memory management, closures and first-class functions.

Implementation-wise the language is small in size yet expressive, which makes it suitable for higher level programming in embedded systems such as network gateways. Since the web user interface of OpenWrt uses Lua, it was beneficial to implement as much of the application in the programming language and contained in a Lua Configuration Interface module. See section 4.4.4 for details on *Lua Configuration Interface*, or LuCI.

4.3.2 Almquist shell

Adhering to the POSIX standards, Almquist shell[1] on OpenWrt it is the default operating system command-line interface, scripting language and command processor. It has less features than *bash* and sometimes requires a strict and sometimes more verbose scripting style, without the permissiveness of bash idioms, such practices are referred to as *bashism* and are often incompatible with ash.

Shell scripting allows the developer to aggregate the power of a range of UNIX programs, using redirection to route the output of one program to the input of another, and with the shell prompt as an interactive development environment it allows for rapid prototyping. The permissiveness and features of bash, has brought along handy constructs such as *process substitution*, which is unavailable in ash. Instead the developer has to resort to manually handling named pipes, which creates a temporary buffer in which the processes can exchange data. An example redirecting the output from a command a named pipe, and processing the output of that command is available in appendix B.1.

4.3.3 JavaScript

JavaScript is a common interpreted programming language mostly used in client-side web development. For the LuCI web interface in OpenWrt, the view can be extended using JavaScript[2] to include custom graphical elements and interaction logic to the Model-View Controller framework. LuCI can perform much of the heavy lifting when it comes to interactive pages, but it supplies support routines for AJAX operations through their *xhr.js* library. AJAX calls are used extensively in the module, because of the difficulty of writing interaction logic sticking to the LuCI CBI models. For an example on how to perform an XMLHttpRequest, sending data from JavaScript to the Lua backend and receiving an answer, please refer to figure B.2.

4.4 Software suite

The newer Inteno devices ship with the OpenWrt distribution, which is a small GNU/Linux operating system. It provides the developer with the basic UNIX debugging tools and a POSIX compatible command-line interface shell. As common with free software, the OpenWrt exists in an ecosystem of applications and tools, in this section a few of these parts are discussed.

4.4.1 OpenWrt

OpenWrt is a free and open-source GNU/Linux distribution, targeting embedded devices, specifically wireless routers, but can run on almost any set of hardware. Cross-compilation is enabled by OpenWrt Buildroot, which compiles the C code using uClibc, a lightweight C library focusing on embedded Linux systems. It intends to be a meta distribution and offers developers a framework on which to base their firmware on.

OpenWrt is generally compiled and linked using gcc and binutils, with the help of Makefiles and patches for the various gcc versions and target platforms. Allowing end users as well as service operators and hardware manufacturers to compile the firmware. It offers the BusyBox set of barebones UNIX tools, enabling advanced users to fully interact with their Linux system and providing developers with a familiar platform for debugging and testing their product.[6]

Unified Configuration Interface, or UCI, is used in OpenWrt as a uniform format for commonly used configuration files. UCI has a Lua bindings as well as a command line interface, to read and modify

[1] Also called *sh*, *A shell* or *ash*
[2] Just like in ordinary web design

the configuration files. Rules for port forwarding are defined in the UCI compatible configuration file in $/etc/config/firewall$.

A port forwarding rule which forwards external HTTP traffic over port 80 to the internal IP 192.168.1.214, is shown in figure A.1 in appendix A. The line *config redirect* defines the start of a section, a section contains several values and a UCI configuration file can have several such sections.

4.4.2 OPKG

The package management system used in OpenWrt is Open PacKaGe Management, or *OPKG*. It is based off the discontinued *ipkg* and operates similar to APT and *dpkg* of Debian-based distributions. It targets GNU/Linux based operating systems for embedded devices and there are currently over 2000 OPKG packages available for OpenWrt.

The OpenWrt system and its packages are built using *GNU Autoconf*, which automates tasks associated with compiling larger software suites. This includes pulling in parts of the system from remote software repositories and automatically resolving dependencies on programs and libraries.

4.4.3 Inteno Open Platform System

For Customer Premises Equipment like wireless gateways, Inteno Open Platform System offers an open-source Linux distribution based on OpenWrt. It uses the OpenWrt build system including cross-compilation toolchain to ensure compatibility with the ecosystem and upstream.

Inteno maintains and hosts a remote repository, which contains a frozen release of OpenWrt and compatible packages and patches. Freezing an ever changing open source codebase means forking an existing version, submitting more conservative patches to the system and focusing on smaller changes. This leads to good compatibility with Inteno hardware and protection from breakage because of upstream[3] code changes.

4.4.4 Lua Configuration Interface

Lua Configuration Interface, or *LuCI*, is a suite of programs and libraries for extending OpenWrt using the Lua programming language and providing a web interface built with the Model-View Controller architecture. It originated in the OpenWrt project, but is now an independent project on its own. Developing LuCI pages that interact with the settings of the OpenWrt deployment is usually done with CBI models, which map the Lua module to OpenWrt and its configuration files.

LuCI relies on the *Model-View Controller* software architecture pattern and separates data and its visual representation. It is divided in three parts with the model representing the data and storing it in UCI configuration files, refer to figure A.2 for an example of a UCI file. The view provides a visual representation of the model. When sticking to CBI models, most of the code and design effort can be put into the Lua module, allowing LuCI to automatically handle GUI elements, form validation, and writing UCI files.

The controller part of the LuCI MVC framework is the dispatching tree. It associates input events with logic in the model, and contains a tree of dispatching actions which is most apparent in the display of the dropdown menus of the web site.

4.4.5 Dnsmasq

In order for the user to have access to the web interface by a domain name instead of an IP address, we use DNS. Dnsmasq is a lightweight DNS server, using the */etc/hosts* file to translate IP addresses on the local network to domain names.[1] The simple operation of Dnsmasq is suitable for serving address translations on smaller networks, such as LAN.

[3]Code released and maintained by the official project

Chapter 5

Implementation

The overall design of the system consists of two parts, the service identification and port forwarding presets. These parts are connected through the LuCI dispatcher[1], and the model. Communication between the parts of the port forwarding process is outlined in figure 5.1. The user initiates the identification procedure and the identification process starts. Nmap and its wrapper script is represented by the *package* : *detect* process in the digram, it receives a call from the dispatcher that originates in an AJAX call from the view. A scan is performed in the background which When the results from the identification are returned the list of presets is sorted, based on identification and the user can review their options.

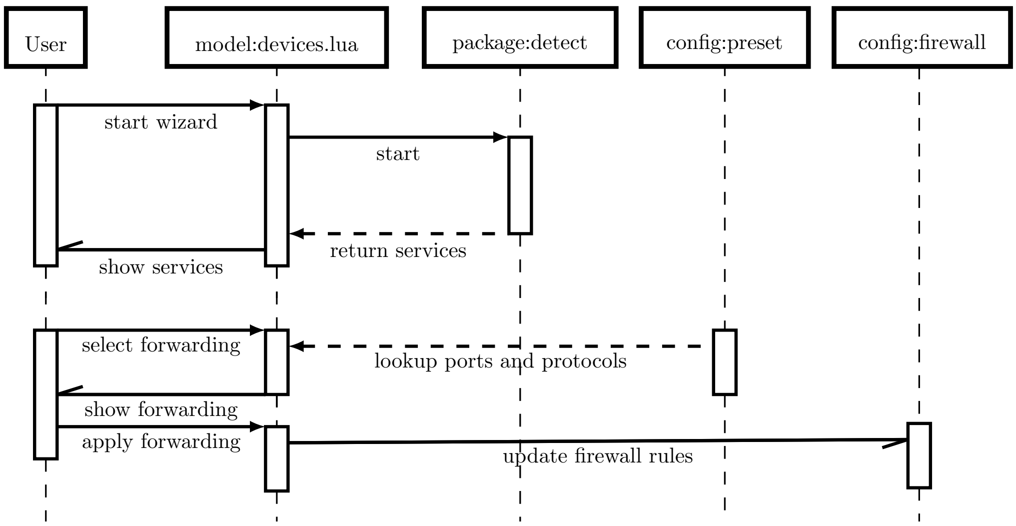

Figure 5.1. Sequential diagram of applying port forwarding rules, the User box represents the view or the user interaction part of the implementation.

By selecting the name of the service, the correct forwarding rules are loaded and presented to the user, who can then chose to apply them, after which they are written to the configuration files.

The port forwarding wizard works with the standard OpenWrt configuration files[2] and uses nmap as its backend for discovering and identifying services on the local network.

[1] Not shown in diagram

[2] OpenWrt can be configured using Unified Configuration System, or UCI. It also provides a command-line utility and provides an API for programming languages such as C and Lua

5.1 User interface

For a port forwarding interface page, the user is presented with detected nodes and their corresponding network services, as shown in figure 5.2. Listed presets are sorted by the output from the service identification process, presenting the user with the most likely services at the top of the list. To apply the port forwarding rule set, the user selects a node in the network and the service from the sorted list, then applies it.

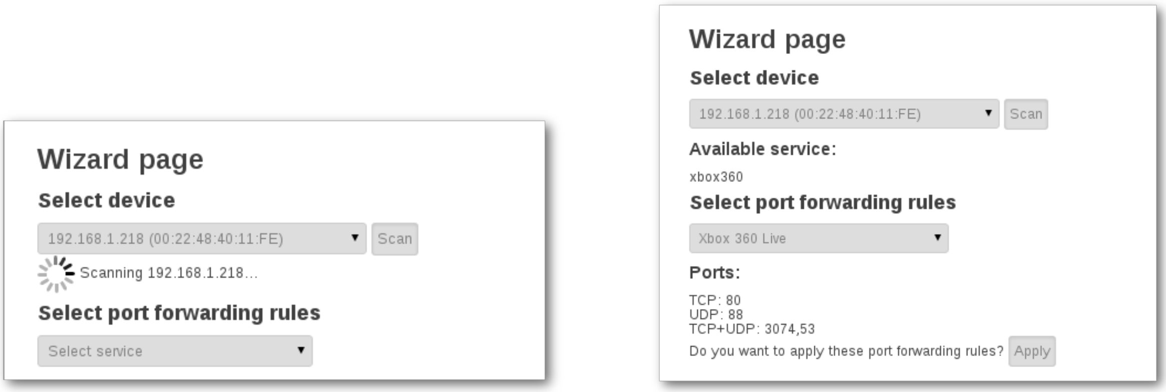

Figure 5.2. Screenshots of scanning the local network for available services.

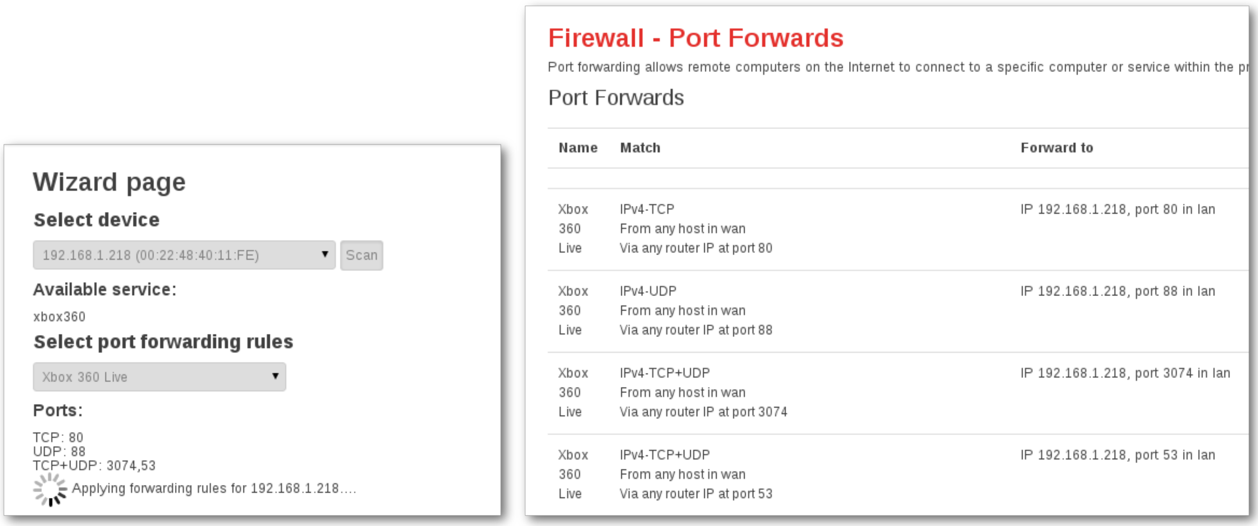

Figure 5.3. Screenshots of applying the forwarding rules and viewing the resulting redirection rules on the conventional firewall configuration page.

Instead of performing all the steps automatically, the user is required to interact and approve of the suggested changes. The service identification is there to help the users make choices, not deciding for them.

Visible in the sequential diagram of figure 5.1 are the two UCI configuration */etc/config/firewall* and */etc/config/preset*, located on the router filesystem. The preset file the source of the final redirection rule in the firewall configuration file. The service detection wrapper returns the newly discovered services, these can then be matched against a preset of forwarding rules.

5.2 Internal DNS

The implementation of the internal IP address translation is a small and simple improvement of the user experience. In order for the end user to reach the web interface they need to find out and enter the IP address of the gateway. DNS works by translating hard to remember IP addresses to memorable domain names, by using the lightweight DNS forwarder Dnsmasq we can make the web interface easier to access. Being a standard part of most operating systems, the hosts file keeps a localized record of address translations. On most Unix-like operating systems the hosts file is assessable from */etc/hosts*, and consist of a text file that list IP addresses and their corresponding domain name.

This small usability improvement consists of a shell script that keeps the routers current IP up to date with the domain name *login.lan*. Since any DNS request passes through the router, Dnsmasq can translate the domain name to its own IP address. The source code of this implementation is available in figure B.3.

5.3 Nmap

The program *Nmap* is a popular network discovery program, and was chosen as the engine for the service scanner implementation. The XBox 360 gaming console was chosen at Inteno's suggestion, with the motivation that it is one of the devices that end users have had the most issues with, in regards to port forwarding. Nmap is capable of detecting several operating systems, embedded devices and network services.

Using Nmap is quite intrusive and could be detected as an attack by intrusion detection software, used to monitor the network for illicit behavior. An alternative approach is using passive fingerprinting of network traffic, one such utility is P0f which uses passive scanning of traffic.[7] However, in order to provide low latency, the Inteno routers are configured with cut-through switching, which would render the passive fingerprinting of P0f ineffective. Cut-through switching, as opposed to store and forward, starts forwarding the packets before it has been fully received, this hides hides the packet information from software processing and analyzing techniques.

5.3.1 Wrapper

Executing the Nmap scanning utility and returning results, is implemented as a shell script. In the development process of the wrapper, a shell script was written to test the functionality and extract data about the detected services. The original intent was to replace this with a Lua script, for a more consistent codebase in regards to the rest of the system. Due to lack of time, the rewrite was cancelled and a quick adaptation was made to to the script to return valid JSON for the JavaScript frontend.

While testing the service identification features of Nmap, there is no way for Nmap to positively identify an XBox 360. This failure was due to an inconclusive fingerprint, but using the flag called version scan – run with arguments *-sV* – Nmap interrogates ports more thoroughly and returns more information than a regular operating system scan. The extra scan using the Nmap *version scan*, was successfully used to identify the XBox 360. Whenever the service is identified as *LSA-or-nterm*, the TCP ports 1026 and 1027, were scanned, either of these are in use by the XBox 360.[11] A more thorough version scan is issued for the device and then matched for *XBox 360 UPnP*, which the wrapper is set to interpret as a positive match and returns its service name[3].

5.4 Preset library

The preset library consists of common services and port data, that the user would want to set up forwarding rules for. Details of these ports and protocols are provided by the application developers,

[3] The XBox 360 is labeled *xbox360* in the configuration presets

specifically for address translation reasons.

Using the *Unified Configuration System*, which is included in the OpenWrt distribution, all the basic commands for configuring the firewall rules were prototyped and explored. A set of *XBox 360* port forwarding rules from the preset configuration file is shown in figure A.1 in appendix A. The rules were formatted to fit the UCI configuration file format and returned as JSON to the JavaScript frontend in an AJAX call through the Lua dispatcher.[4]

For the scope of this project the following services is added to the preset library:

- Xbox360
- HTTP traffic
- POP3 email
- FTP traffic
- SSH traffic
- IMAP email
- IMAP3 email
- IMAPS email
- POP3S email

Applying the rules requires the user to select the desired service from a list, and pressing a button which runs a JavaScript function, performing an AJAX call to the Lua backend, issuing the UCI calls. See section 4.3.3 for examples of how JavaScript is used in the LuCI module.

[4]The dispatcher is the *Controller* in the MVC framework

Chapter 6

Results

6.1 Operating system scan

The results of the basic operating system scan of XBox 360 is shown in figure 6.1. Note that the line:

`No exact OS matches for host (test conditions non-ideal).`

We observe that the scan is unable to identify the correct operating system against the XBox 360 and had a duration of 35.78 seconds. A more thorough scan is required to detect the gaming console in question.

```
root@Inteno:~# time nmap -O --osscan-guess --fuzzy 192.168.1.218

Starting Nmap 5.51 ( http://nmap.org ) at 2013-05-28 18:03 CEST
Nmap scan report for 192.168.1.218
Host is up (0.00061s latency).
Not shown: 999 filtered ports
PORT     STATE SERVICE
1026/tcp open  LSA-or-nterm
MAC Address: 00:22:48:40:11:FE (Microsoft)
Warning: OSScan results may be unreliable because we could not find at
least 1 open and 1 closed port
Device type: general purpose|switch
Running (JUST GUESSING): IBM OS/2 4.X (92%), HP OpenVMS
Aggressive OS guesses: IBM OS/2 Warp 2.0 (92\%), HP OpenVMS
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop
OS detection performed. Please report any incorrect results at
http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 41.67 seconds
real 0m 35.78s
user 0m 18.00s
sys  0m 2.37s
```

Figure 6.1. Raw output of first Nmap scan of XBox 360, failing to guess target operating system.

6.2 Version scan

By issuing a version scan, this run of Nmap is able to positively identify the service *XBox 360 XML UPnP (Serial number 757502283805)* in 13 seconds, as shown in figure 6.2.

```
root@Inteno:~# time nmap -sV -p 1026-1027 192.168.1.218

Starting Nmap 5.51 ( http://nmap.org ) at 2013-05-28 18:06 CEST
Nmap scan report for 192.168.1.218
Host is up (0.00081s latency).
PORT     STATE    SERVICE VERSION
1026/tcp open upnp XBox 360 XML UPnP (Serial number 757502283805)
1027/tcp filtered IIS
MAC Address: 00:22:48:40:11:FE (Microsoft)
Service Info: Device: game console

Service detection performed. Please report any incorrect results at
http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.89 seconds
real 0m 13.02s
user 0m 4.09s
sys  0m 1.44s
```

Figure 6.2. Raw output of deeper Nmap scan of XBox 360, positively identifying it as *XBox 360 UPnP*.

This exhaustive scan takes longer to complete per port, but is limited to a small range. The results shows that the system manages to identify the XBox 360 gaming console, using the extra scan issued by the wrapper, the device can positively be identified correctly. Its services in terms of ports are well known and defined in the preset part of the implemented system.

6.3 Linux server

Scanning a faily standard GNU/Linux operating system[1] running on the Raspberry Pi installed with the options *web server*, *mail server* and *ssh server*, detect these services and ports as shown in figure 6.3. The mail server option in the installer, enables identification on ports 110, 143, 993 and 995 because of the various email delivery protocols. Every major network service is discovered, on the GNU/Linux system.

```
PORT     STATE SERVICE   REASON  VERSION
22/tcp   open  ssh       syn-ack OpenSSH 6.0p1 Debian 4 ...
80/tcp   open  http      syn-ack Apache httpd 2.2.22    ...
110/tcp  open  pop3      syn-ack Dovecot pop3d
111/tcp  open  rpcbind   syn-ack 2-4 (RPC #100000)
143/tcp  open  imap      syn-ack Dovecot imapd
993/tcp  open  ssl/imap  syn-ack Dovecot imapd
995/tcp  open  ssl/pop3  syn-ack Dovecot pop3d
MAC Address: B8:27:EB:0C:A5:70 (Raspberry Pi Foundation)
```

Figure 6.3. Nmap version scan of the Raspberry Pi, identifying available services on the open ports.

[1]The operating system running on the Raspberry Pi is the recommended Raspbian, a Debian based distribution

The Raspberry Pi running the Debian GNU/Linux distribution is successfully detected as such, all services selected during the installation are successfully detected. The front-end with select the first service from the dropdown list of presets and present the user with the choice to apply its forwarding rules.

Chapter 7

Conclusions

The preset system will simplify the port forwarding procedure and provide the novice user with helpful hints, in an otherwise complex graphical environment. This part of the system could be made production ready and included by default in the iopsys platform. Service detection fits with the preset data and the combination of these results results in a qualified guess about services the users might want forwarded. Detection is currently slow and tweaking the scan arguments any more were outside the scope of this project.

The identification process of the XBox 360 is not a generic Nmap solution, it requires a workaround implemented in the shell script wrapper. This behaviour is not optimal but perhaps acceptable, depending on the frequency of the issue of future devices not delivering sufficient data for Nmap fingerprinting. Considering the CPE operators various needs, we are unable to draw any conclusions as to weather it should be implemented as part of the default distribution or not.

The choice of not employing usability testing and working in an iterative process while developing the design was a mistake. The scope of usability enhancement were however relatively small and we conclude that the results of the improvements made to the graphical user interface are positive in terms of usability.

7.1 Further development

The solution using Nmap could be interpreted as illegal activity and attempts at exploiting the systems of network administrators and intrusion detection systems, this is a risk which could render the proposed solution undesirable. A fix for this would be to implement a less intrusive way of identifying services, reviewing the ARP tables which contains the cache of the address resolution protocol. One could filter possible devices by their manufacturers MAC address, with an already populated ARP table this procedure is unintrusive and fast. A more detailed Nmap *version scan* could then be performed according to a configuration file, mapping MAC addresses of known device manufacturers to more lightweight Nmap scans. It would be a better fit for our reference model, but more error-prone and less generic than using Nmap.

The execution time of Nmap is an issue, on local networks with several devices the latency would be deemed too high for several use cases. To address this issue one could adjust the service to preload the automatic identification results and have it run in the background, to provide a more responsive user experience using the cached results. But allowing such background jobs that are not system-critical will most likely be deemed too wasteful on embedded systems.

Appendix A

Configuration files

```
config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp'
option src_dport '80'
option dest_ip '192.168.1.214'
option dest_port '80'
option name 'Web server'
```

Figure A.1. Port forwarding section in the UCI *firewall* configuration file.

```
config device 'xbox360'
option name 'xbox360'
option description 'Xbox 360 Live'

config redirect
option device 'xbox360'
option proto 'udp'
option port '88'

config redirect
option device 'xbox360'
option proto 'tcp udp'
option port '3074'

config redirect
option device 'xbox360'
option proto 'tcp udp'
option port '53'

config redirect
option device 'xbox360'
option proto 'tcp'
option port '80'
```

Figure A.2. Preset for port forwarding sections in the UCI *preset* configuration file.

Appendix B

Programming and shell scripting

```
# create a named pipe, if it does not exist
[ ! -p /tmp/fifo ] && mkfifo /tmp/fifo
# scan target, pipe the output into a named pipe
nmap -O --osscan-guess --fuzzy $1 > /tmp/fifo &
while read -r line
do
  case "$line" in
    *"open"*) # on open ports, perform:
      # extraction of service name and ports
      SERVICE=$(echo "$line" | awk '{print $3}')
      PORT=$(echo "$line" | awk -F "/" '{print $1}')
      # check if port is a number
      if [ "$PORT" -eq "$PORT" ] 2>/dev/null; then
        # append to RESULT
        RESULT="$RESULT\"$SERVICE\": $PORT, "
      fi
    ;;
  esac
done < /tmp/fifo
rm /tmp/fifo
# return RESULT
```

Figure B.1. Wrapper for Nmap, by using a named pipe to redirect the output of one program to a temporary file and reading it line by line, the wrapper parses the output and formats it as a JSON string.

```
xhr.get('<%=luci.dispatcher.build_url("admin","wizard")%>/get_list/'+argument, null,
    function(reply) {
        // Perform operation on 'reply'
    }
);
```

Figure B.2. Example of calling the LuCI dispatcher from client-side JavaScript with an argument, using xhr.js.

```
#!/bin/sh
#update hosts file to enable access through "http://login.lan/" from LAN PC

string="login.lan login"
HOSTS=/etc/hosts
IP=`uci get network.lan.ipaddr`

reload_dns() {
    /etc/init.d/dnsmasq reload &
}

main() {
    ROW=0
    while read LINE; do
        ROW=$((ROW+1))

        case $LINE in
            # Already up to date, exit
            "$string")
                exit;;

            # Change to current IP, reload
            *"$string")
                sed -i -e "$ROW s/.*/$IP\ $string/g" $HOSTS
                reload_dns
                exit;;
        esac
    done < $HOSTS

    # Add new entry
    echo "$IP $string" >> $HOSTS
}

main
```

Figure B.3. Script that keeps the IP address translation for the local network up to date, allowing users to access the gateway by visiting *http://login.lan* in their web browsers address bar.

Bibliography

[1] dnsmasq Linux man page. http://linux.die.net/man/8/dnsmasq. Accessed: 2013-08-08.

[2] Inteno GPL support page. http://www.inteno.se/Support/GPL.aspx. Accessed: 2013-05-21.

[3] New business possibilities with Open Source software. http://www.inteno.se/Portals/0/IntenoFiles/ProductDocs/241/689/iopsys_white_paper.pdf_20121015135755.pdf. Accessed: 2013-04-29.

[4] Open Source Initiative homepage. http://opensource.org/licenses/mit-license.php. Accessed: 2013-06-05.

[5] Open Source Initiative homepage. http://opensource.org/licenses/mit-license.php. Accessed: 2013-06-05.

[6] OpenWrt structure and design. http://wiki.confine-project.eu/_media/soft:openwrt-talk-2012-06-01.pdf. Accessed: 2013-04-29.

[7] p0f homepage. http://lcamtuf.coredump.cx/p0f3/. Accessed: 2013-05-22.

[8] Port Forward homepage. http://portforward.com/. Accessed: 2013-05-14.

[9] R. Ierusalimschy. *Programming in Lua*. Lua.org, 2006.

[10] S. Krug. *Don't make me think!: a common sense approach to Web usability*. Voices That Matter Series. New Riders, 2006.

[11] Halvar Myrmo. Game consoles - are they secure? Master's thesis, Gjøvik University College, 2007.

[12] D.A. Norman. *The Design of Everyday Things*. Basic Books, 2002.