SAPIENZA
UNIVERSITÀ DI ROMA

Seminar in Advanced Topics in Computer Science

Master Degree in Engineering in Computer Science

IoT Honeypots: State of the Art

Author:

Biagio Botticelli 1212666

September 2, 2017

Contents

Table of Contents **2**

1 Introduction **4**
1.1 Internet of Things . . . 5
1.2 IoT Security Risk . . . 6

2 Attacks to IoT Devices: Botnets **8**
2.1 From Botnets to Thingsbots: Mirai Example . . . 9
2.2 Defense Techniques against Botnets . . . 10

3 Denial of Service (DoS) and Distributed Denial of Service (DDoS) **11**
3.1 DDoS Motivation . . . 15

4 DDoS Taxonomy **16**
4.1 Bandwidth Depletion Attacks . . . 16
4.1.1 Flood Attacks . . . 16
4.1.2 Amplification Attacks . . . 17
4.2 Resource Depletion Attacks . . . 18
4.2.1 Protocol Exploit Attacks . . . 18
4.2.2 Malformed Packet attacks. . . . 19

5 IoT Malware **20**
5.1 Linux.Hydra . . . 20
5.2 Psyb0t . . . 20
5.3 Chuck Norris . . . 21
5.4 Tsunami . . . 21
5.5 LightAidra/Aidra . . . 22
5.6 Carna . . . 22
5.7 Linux.Darlloz . . . 23
5.8 Qbot/Qakbot . . . 23
5.9 Linux.Wifatch . . . 23
5.10 TheMoon . . . 24
5.11 Spike/Dofloo . . . 24
5.12 BASHLITE . . . 24
5.13 KTN-RM / Remaiten . . . 25
5.14 Mirai . . . 25
5.15 Linux/IRCTelnet . . . 26
5.16 Hajime . . . 27
5.17 Amnesia . . . 27
5.18 BrickerBot . . . 27

6 DDoS Defense Techniques **30**

7 IoT Honeypot **34**
7.1 Advantages and Disadvantages . . . 35
7.2 Types of Honeypots . . . 36
7.2.1 Research Honeypots . . . 36

7.2.2 Production Honeypots . . . 36
7.2.3 Low Interaction Honeypots . . . 36
7.2.4 High Interaction Honeypots . . . 37
7.2.5 Hybrid Approach: Low + High Interaction . . . 38
7.2.6 Medium Interaction Honeypots . . . 38

8 IoT Honeypot: State of the Art 40
8.1 Low Interaction Honeypots . . . 40
8.1.1 Nepenthes . . . 40
8.1.2 Dionaea . . . 41
8.1.3 Glastopf . . . 41
8.1.4 HoneyD . . . 43
8.1.5 HoneyBOT . . . 45
8.1.6 Honeytrap . . . 45
8.1.7 Bluepot . . . 46
8.1.8 Shockpot . . . 47
8.1.9 HoneyC . . . 48
8.1.10 Monkey-Spider . . . 49
8.1.11 YALIH . . . 50
8.1.12 MTPot . . . 51
8.1.13 Telnet IoT Honeypot . . . 52
8.2 Medium Interaction Honeypots . . . 53
8.2.1 Kippo . . . 53
8.2.2 Cowrie . . . 55
8.2.3 IoTPOT . . . 57
8.3 High Interaction Honeypots . . . 59
8.3.1 HoneyPHARM . . . 59
8.3.2 SIPHON (Scalable high-Interaction Physical HONeypot.) . . . 60

Abstract

The quantity of IoT devices is exponentially growing and it's estimated that within the 2020, the number of connected devices will exceed 40 billions and it will have a total potential economic impact of $3.9 trillion to $11.1 trillion a year.
Internet of Things has been called the next Industrial Revolution and it will impact the way all businesses, governments, and consumers will interact with the physical world experiencing an increased connectivity. However, if on one hand we have the idea of a world in which devices are all digitally connected, allowing us to be able to do things we never before imagined, on the other one there is a downside: IoT is becoming an increasingly attractive target for cybercriminals because of lack of security. More connected devices mean more attack vectors and more possibilities for hackers to target users and companies; unless this growing security concern would be fast addressed, there will be an intricate question.

"Whatever can go wrong, will go wrong" - Murphy's Law

The aim of this document is to present the IoT Security Problem under different points of view: there will be an introduction on the aspects which make IoT so attractive for hackers, a presentation of the main IoT attack methodologies (with particular emphasis on botnets and Distributed Denial of Service (DDoS) attacks) a description of IoT Malware families and an introduction on the techniques that could be use against them.
The last part of the project will be focused on the concept of IoT honeypot in general and then focusing on the single technologies having particular interesting features. Moreover a comparison between the different existing Honeypot technologies will be made according to different features.

Keywords: Internet of Things, IoT Security, Honeypots, IoT Malware, Malware Collection, Denial of Service (DoS), Distributed Denial of Service (DDoS), DDoS Defense Techniques, Botnets, Network Attacks, Intrusion Detection.

1 Introduction

In Tuesday, September 20th 2016, `KrebsOnSecurity.com` blog was targeted by an extremely large and unusual **Distributed Denial-of-Service attack** (**DDoS**) of over 660 Gbps of traffic.
The attack seems to have been designed to knock offline the website of the investigative cybercrime journalist Brian Krebs in retaliation for the arrest of the owners of vDOS attack-for-hire service.
The attack did not succeed, but according to Akamai it was nearly double the size of the largest attack they had ever seen and it orders of magnitude more traffic than is typically needed to knock the most of sites offline.

Figure 1: Krebs' tweet after the DDoS Attack

The most interesting aspect of this attack is that it was <u>not</u> performed by using traditional reflection/amplification DDoS, but with **direct traffic** instead: the attack was carried out by a Botnet (or Zombie Network) of hacked devices.
While the total number of devices involved was not known for sure, it was sure that hundreds of thousands of compromised devices were related to the Internet of Things (mainly home routers, IP security cameras, Digital Video Recorder boxes and printers).
The IoT devices became infected with malware by very simple `Telnet` dictionary attacks and were made part of the botnet that will made the DDoS attack.
A couple of days after the attack, the source code of the malware dubbed `Mirai` was published by its creator (the hacker Anna-senpai) on `hackforums.net` hacking community.

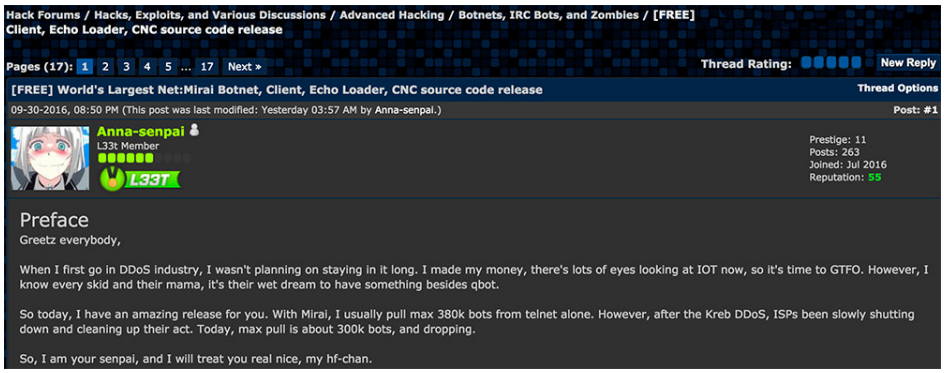
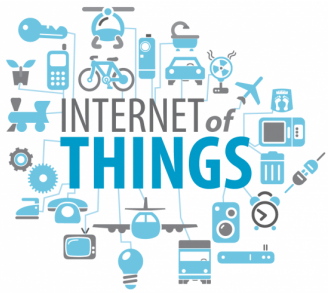


Figure 2: Mirai source code release on `hackforums.net`

The same month, a Mirai-based attack against the French webhost OVH broke the **record** for the largest recorded DDoS attack with at least 1.1 Tbps of traffic.

1.1 Internet of Things

The Internet of Things could be defined as the vision where real-world objects become part of the Internet: every device is uniquely identified and accessible to the network, its position and status known; services and intelligence are added to this expanded Internet, fusing the digital and physical world into one single entity.



IoT is a system architecture with interrelated computing devices, objects, entities, digital and mechanical machines and people with exceptional skills.
Systems can exchange information with a network, but not limited to the need of any human-to-computer or human-to- human interaction (human is not the center of the system).

The merging of micro electro-mechanical and wireless systems, Internet and micro-services has contributed to the rise of IoT: there is a huge variety of smart objects which are being introduced at each layer of Information Technology for a precise purpose and with specific characteristics and the number and diversity of these connected devices is expected to increase *exponentially*.

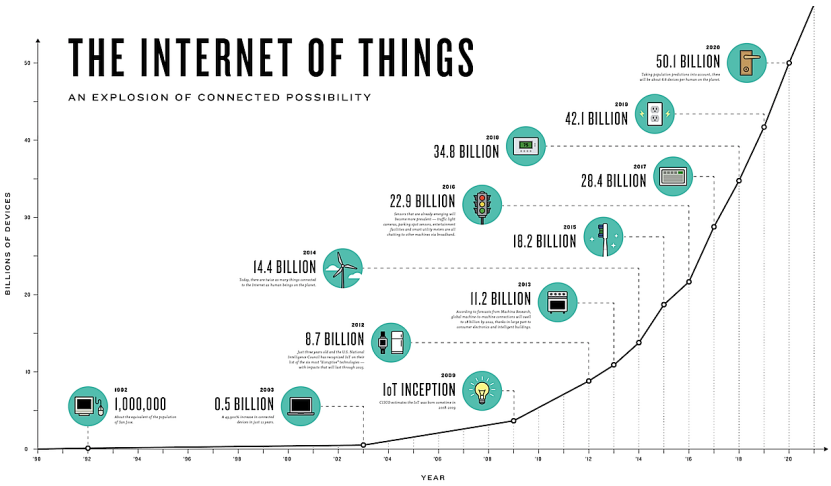


Figure 3: Exponential growth of the number of IoT devices

The connected "things" (for example, sensors or mobile devices) monitor and collect all types of environment data. They enable the collection of real-time data about properties, individuals, plants, and animals offering personalized services to users.
However, an increased connectivity leads to an exponential increase in the **threat surface**: more smart technology we add, more likely it's to be hacked from the point of view of security.

Typically IoT devices are poorly secured (sometimes, not secured at all) and the interconnected nature of these smart objects means that every poorly secured device that is connected online, it potentially affects the security and resilience of the Internet globally.
So, there is an important **Security Risk** for what concerns IoT and since the amount of IoT devices is *growing*, this problem will only become *bigger* in the next future.

1.2 IoT Security Risk

The main problem with IoT devices is that the majority of them has lack of even elementary security and they present some interesting features which make them an ideal target for hackers. In fact, compared to common systems, IoT systems are at **higher security risk** for several reasons:

- ◇ **Fast Connection:** IoT devices are typically connected to fast Internet networks;
- ◇ **Always Online:** in contrast to human-controlled devices, IoT ones are typically operating 24 hours a day, 7 days a week;
- ◇ **Perimeters:** IoT systems do not have well defined perimeters and perimeters continuously change due to device and user mobility;
- ◇ **Heterogeneity:** IoT systems are highly heterogeneous with respect to communication medium, protocols, platforms, and devices;
- ◇ **Devices Control:** IoT devices could be autonomous entities that control other IoT devices and/or they can be controlled by different parties;
- ◇ **Expanded Internet:** IoT systems might include objects originally not designed to be connected to the Internet;
- ◇ **Physical Protection:** IoT systems, or portions of them, could be physically unprotected;
- ◇ **Permissions:** unlike smartphone applications which require permission (e.g. for installation and for other user interactions), granular permission requests might not be possible in IoT systems because of the large number of devices.

So, the main challenges for IoT security come from the *heterogeneity* and the *large scale* of objects.

In addition, there are several security issues related to IoT, which are related to each IT OSI Layer (from the `Physical`, arriving to the `Application` one):

- **Identification and Location:** the main security issue is to *uniquely identify* an IoT device. An ideal identification methodology not only identifies the objects uniquely, but also reflects the property of the object. Although IP addressing may still be used in the future Internet, `Named Data Networking` (`NDN`) could replace it as new standard.
- **Authentication and Authorization:** *authentication* is traditionally achieved by methods such as ID/password, pre-shared secrets, and public-key crypto systems; while *authorization* can be achieved by database-based or crypto-based access control.
  Due to the heterogeneity and complexity of the objects and networks in IoT, traditional authentication and authorization methods may not be applicable.
- **Privacy:** in traditional Internet, information about user behavior is collected while browsing the Web, to enrich the user experience. In IoT, the collected data also includes information about a user's daily routine, so that the IoT devices can cooperate to provide better services. Owning this information that describes the user in detail, it involves a security need since the data must be protected in order to preserve the user privacy.

- **Security Protocol:** there are many resource-constrained IoT devices which have limited computing power and battery capacity (e.g. sensor nodes and wearable devices).
  Although many conventional security protocols are considered secure and robust, they may not be suitable for these kind of devices.

- **Software Vulnerabilities:** during the development of a software, programming bugs could be accidentally created by developers. Bugs that result in security incidents are known as software vulnerabilities. Upon discovery of a *new software vulnerability* (also called `0-Day`), attackers can leverage this knowledge to exploit a large number of machines.
  While in the traditional PC industry, security awareness is relatively easy to be enforced with proper education, in the heterogeneous IoT context, diversified hardware platforms makes it very difficult to educate on security awareness.
  The main issue with IoT software vulnerabilities is that they can lead to the installation of a **backdoor** to control the device: due to resource constraints of IoT devices, security mechanisms (such as IDS or antivirus) are not applicable in IoT. Therefore, it is relatively easy for attackers to inject backdoor into victim's device.

- **IoT Malware:** in November 2013, Symantec confirmed the finding of the first IoT malware (`Linux.Darlloz` worm), which raises the issue for IoT malware security.
  IoT allows a great connectivity among various devices and this is attracting attackers: once connected to a victim user, a hacked IoT device could infect any other device held by the victim. The malware threat and related countermeasure in IoT will become a critical problem.

The IoT Security problem has been analyzed also by the Open Web Application Security Project and they identified the 10 most common IoT vulnerabilities which are shown in the following table:

Vulnerability	Exploitability	Detectability	Security Impact	Examples
Insecure Web Interface	EASY: Attacker uses weak credentials, captures plain-text credentials or enumerates accounts to gain the access to the web interface. Attack could come from external or internal users.	EASY: An insecure web interface can be present when account enumeration, lack of account lockout or weak credentials are present. Issues with the web interface are easy to discover when examining the interface manually along with automated tools to identify other issues such as cross-site scripting.	SEVERE: Insecure web interfaces can result in data loss or corruption, lack of accountability, or denial of access and can lead to complete device takeover.	Inability to change default usernames and passwords; weak passwords; lack of robust password recovery mechanisms; exposed credentials; lack of account lockout; susceptibility to cross-site scripting, cross-site request forgery, and/or SQL injection.
Insufficient Authentication/ Authorization	AVERAGE: Attacker uses weak passwords, insecure password recovery mechanisms, poorly protected credentials or lack of granular access control to access a particular interface. Attack could come from external or internal users.	EASY: Authentication may not be sufficient when weak passwords are used or are poorly protected. Many Issues with authentication/authorization are easy to discover when examining the interface manually and can also be discovered via automated testing.	SEVERE: Insufficient authentication/authorization can result in data loss or corruption, lack of accountability, or denial of access and to complete compromise of the device and/or user accounts.	Privilege escalation; lack of granular access control.
Insecure Network Services	AVERAGE: Attacker uses vulnerable network services to attack the device itself or bounce attacks off the device. Attack could come from external or internal users.	AVERAGE: Insecure network services may be susceptible to buffer overflow attacks or attacks that create a denial of service condition leaving the device inaccessible to the user. Insecure network services can be detected by automated tools such as port scanners/fuzzers.	MODERATE: Insecure network services can result in data loss or corruption, denial of service or facilitation of attacks on other devices.	Vulnerability to denial-of-service, buffer overflow, and fuzzing attacks; network ports or services unnecessarily exposed to the Internet.
Lack of Transport Encryption/Integrity Verification	AVERAGE: Attacker uses the lack of transport encryption to view data being passed over the network. Attack could come from external or internal users.	EASY: Lack of transport encryption allows data to be viewed as it travels over local networks or the internet. Many Issues with transport encryption are easy to discover simply by viewing network traffic and searching for readable data. Automated tools can also look for proper implementation of common transport encryption such as SSL and TLS.	SEVERE: Lack of transport encryption can result in data loss and depending on the data exposed, could lead to complete compromise of the device or user accounts.	Transmission of unencrypted data and credentials.
Privacy Concerns	AVERAGE: Attacker uses insufficient authentication, lack of transport encryption or insecure network services to view personal data which is not being properly protected or is being collected unnecessarily. Attack could come from external or internal users.	EASY: Privacy concerns generated by the collection of personal data in addition to the lack of proper protection of that data is prevalent. Privacy concerns are easy to discover by reviewing the data that is being collected as the user sets up and activates the device. Automated tools can also look for specific patterns of data that may indicate collection of personal data or other sensitive data.	SEVERE:Collection of personal data along with a lack of protection of that data can lead to compromise of a user's personal data.	Collection of unnecessary user data; exposed personal data; insufficient controls on who has access to user data; sensitive data not de-identified or anonymized; lack of data retention limits.
Insecure Cloud Interface	AVERAGE: Attacker uses insufficient authentication, lack of transport encryption and account enumeration to access data or controls via the cloud website. Attack will most likely come from the internet.	EASY: An insecure cloud interface is present when easy to guess credentials are used or account enumeration is possible. Insecure cloud interfaces are easy to discover by simply reviewing the connection to the cloud interface and identifying if SSL is in use or by using the password reset mechanism to identify valid accounts which can lead to account enumeration.	SEVERE: An insecure cloud interface could lead to compromise of user data and control over the device.	Inability to change default usernames and passwords; weak passwords; lack of robust password recovery mechanisms; exposed credentials; lack of account lockout; susceptibility to cross-site scripting, cross-site request forgery, and/or SQL injection.
Insecure Mobile Interface	AVERAGE: Attacker uses insufficient authentication, lack of transport encryption and account enumeration to access data or controls via the mobile interface. Attack will most likely come from anyone who has access to the mobile application.	EASY: An insecure mobile interface is present when easy to guess credentials are used or account enumeration is possible. Insecure mobile interfaces are easy to discover by reviewing the connection to the wireless networks and identifying if SSL is in use or by using the password reset mechanism to identify valid accounts which can lead to account enumeration.	SEVERE: An insecure mobile interface could lead to compromise of user data and control over the device.	Inability to change default usernames and passwords; weak passwords; lack of robust password recovery mechanisms; exposed credentials; lack of account lockout; susceptibility to cross-site scripting, cross-site request forgery, and/or SQL injection.
Insufficient Security Configurability	AVERAGE: Attacker uses the lack of granular permissions to access data or controls on the device. The attacker could also use the lack of encryption options and lack of password options to perform other attacks which lead to compromise of the device and/or data. Attack could potentially come from any user of the device whether intentional or accidental.	EASY: Insufficient security configurability is present when users of the device have limited or no ability to alter its security controls. Insufficient security configurability is apparent when the web interface of the device has no options for creating granular user permissions. Manual review of the web interface will reveal these deficiencies.	MODERATE: Insufficient security configurability could lead to compromise of the device whether intentional or accidental and/or data loss.	Lack of granular permissions model; inability to separate administrators from users; weak password policies; no security logging; lack of data encryption options; no user notification of security events.
Insecure Software/ Firmware	DIFFICULT: Attacker uses capturing update files via unencrypted connections, the update file itself is not encrypted or they are able to perform their own malicious update via DNS hijacking. Attack could come from the local network or the internet.	EASY: The lack of ability for a device to be updated presents a security weakness on its own. Devices should have the ability to be updated when vulnerabilities are discovered and software/firmware updates can be insecure when the updated files themselves and the network connection they are delivered on are not protected. Software/firmware issues are easy to discover by simply inspecting the network traffic during the update to check for encryption or using a hex editor to inspect the update file itself for interesting information.	SEVERE: Insecure software/firmware could lead to compromise of user data, control over the device and attacks against other devices.	Lack of secure update mechanism; update files not encrypted; update files not verified before upload; insecure update server; hardcoded credentials.
Poor Physical Security	AVERAGE: Attacker uses vectors such as USB ports, SD cards or other storage means to access the Operating System and potentially any data stored on the device. Attack could come from anyone who has physical access.	AVERAGE: Physical security weaknesses are present when an attacker can disassemble a device to easily access the storage medium and any data stored on that medium. Weaknesses are also present when USB ports or other external ports can be used to access the device using features intended for configuration or maintenance.	SEVERE: Insufficient physical security could lead to compromise of the device itself and any data stored on that device.	Device easy to disassemble; access to software via USB ports; removable storage media.

Figure 4: OWASP Top 10 List of IoT Vulnearbilities

2 Attacks to IoT Devices: Botnets

A **botnet** is a robot network of hacked machines, or *bots* (usually infected by `trojan`, `backdoors` or `worms`), which run malicious code under the remote command and control (C&C) of a *botmaster*.

Botnets have a wide range of malicious purposes including email spam delivery, distributed denial-of-service (DDoS) attacks, permanent denial of service (PDoS), password cracking, key logging, and crypto currency mining.

Bots can automatically scan entire network ranges and propagate themselves using known vulnerabilities and weak passwords on other machines. Once a machine is compromised, a small executable is installed for future activation by the botmaster.
At a certain time, the botmaster can instruct the bots to execute a **DDoS attack** by sending requests to a target website trying to render it unable to serve requests by legitimate users.
Early botnets was organized in a *hierarchical centralized architecture*, in which the bots relied on the `Internet Relay Chat` (`IRC`) for communicating with their botmaster, that could reside on one or more central C&C servers.
As detecting strategies evolved to monitor or block `IRC`, botnets shifted their communication channels to `HTTP`. Then, when detection strategies adapted to `HTTP` communication with a single server, disabling botnets by shutting down these single points of failure, botnets adapted by changing their organization from *hierarchical* to an alternative *distributed peer-to-peer* (P2P) architecture.
Examples of P2P botnets are: `GameOver Zeus`, `Sality`, `ZeroAccess`, and `Kelihos`.

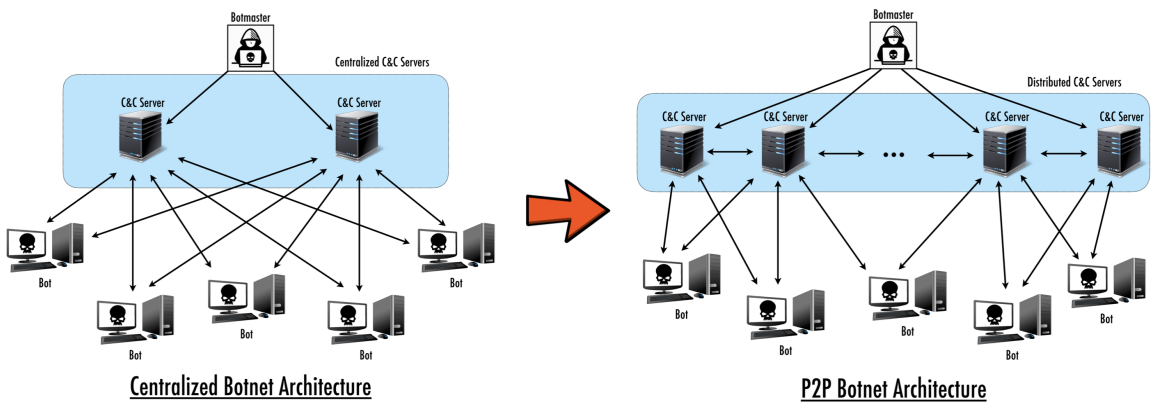


Figure 5: From Centralized to P2P Botnets

Moreover, early botnet-based DDoS attacks were *manual* and attacker had to implement many steps before the launch of final attack, which includes port scanning, identifying compromised machines or zombies in the internet to create botnet, deploying malware etc.
Nowadays, sophisticated automated DoS or DDoS attack tools been developed to assist attackers in implementing some (or all) steps automatically, with minimal human effort: the attacker has to configure just some attack parameters and the attack is managed by the automated tool.
Some of these tools are: `TFN`, `Trinoo`, `TFN2K`, `Shaft`, `Stacheldraht`, `Knight` and `Trinity`.

So, botnets have continued to evolve and adapt to detection techniques making it difficult to consistently and accurately identify their presence on networks.

2.1 From Botnets to Thingsbots: Mirai Example

A conventional botnet is composed by computers that have been remotely accessed without the owners' knowledge and set up to forward transmissions to other computers on the Internet.
The Internet of Things (IoT) is made up of not only dedicated computers, but also of many other objects such as sensors and devices equipped with IP addresses and the ability to transmit data over a network. In the IoT context, these are the so-called *things*.
With the rise of the IoT, many objects and devices are in danger of being (or are already) part of **thingsbots**, which are botnets that incorporate independent connected IoT devices (*things*).
With more and more devices joining IoT, cyber-attacks are only bound to increase, but not only in terms of number attacks, but also in terms of the technology complexity of the attacks.
This threat is very much plausible, because devices, sensors, vehicles, and machines that compose the IoT have rarely been designed with security in mind and, since many of the devices are there to track user habits or to allow remote control, they collect interesting information not only for the final user, but also desirable for hackers.

The term **Botnet of Things** (**BoT**), or **thingsbot**, refers to a botnet of compromised IoT devices which are infected and controlled to exploit cyber-crime activities, such as DDoS attacks.

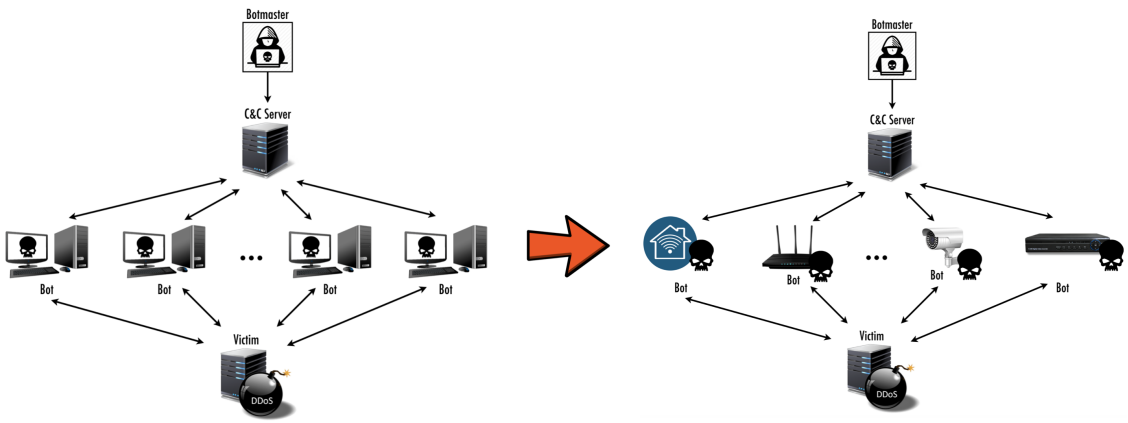


Figure 6: From Botnets to ThingsBots

While hackers have been infecting all sorts of devices for many years, with IoT on the rise and more devices available for hacking, there is a new opportunity for profit for them: there are so many devices available which can be abused and a thingsbot based DDoS attack, where hackers take control of connected devices to attack and overwhelm the an Internet critical infrastructure, is now a real menace.
The recent largest scale DDoS attacks against `KrebsOnSecurity.com` and `Dyn DNS` was exploited by the Mirai botnet, which can be considered a thingsbot.
The Dyn attack generated about 1.2 terabits of attack traffic and paralyzed Dyn's DNS hosts for several hours resulting in numerous major Internet services such as Twitter, Netflix, and Reddit becoming unusable across Europe and the United States.
The most interesting aspect of these attacks was the extensive use of IoT devices as *bots* rather than more conventional hosts such as desktops and networking infrastructure.
By compromising large numbers of IoT devices, the Mirai botmaster was able to generate massive amounts of attack traffic from legitimate network hosts, controlling the infected devices from the Mirai C&C servers to take down the victim's network.

Attack traffic included `GRE` and `TCP SYN` packets, together with `HTTP GET` and `POST` requests and all of these traffic types often appear as *legitmate traffic*, making it difficult to filter.

The success of the Mirai botnet was primarily due to the large number of available IoT devices with remote access credentials stored in their firmware.
These devices exposed `Telnet`, `SSH`, and `web services` to the external Internet on their interfaces protected only by the default firmware credentials and these credentials (`username` and `password`) cannot be changed without an update to a new firmware from the manufacturer. Furthermore, not even the remote access can be disabled without modifying the firmware of the IoT device or altering the running code on the device.
So, to protect the device from remote access, the IoT device owner would require the manufacturers involvement to patch vulenrabilities (very difficult to achieve for each single device).

2.2 Defense Techniques against Botnets

Defending against botnets and thingsbots is an area of active research in the security community. The most common defense strategies against botnets are disabling C&C servers and sinkholing. The C&C server takedown addresses the problem from the point of view of where the attack traffic is controlled; while sinkholing the attack traffic, it addresses the problem from the point of view nearest to the victim.

Disabling the C&C servers is often referred to as *"cutting the head off of a Hydra"* since many botnets are able to bring replacement of C&C servers which are online very quickly.
In order to takedown a botnet's C&C server, the server's location must be determined with either traffic analysis or disassembly of the bot code.
Current bots use encryption for both hiding this information inside the code and for encrypting their communication channels in order to evade detection and discovery of the C&C servers; and even if the servers are discovered, takedowns may not be very effective due to the ability of botnets to dynamically alter their C&C servers quickly.
Moreover, physically disabling the C&C servers is often difficult (occasionally impossible) because the servers are typically located outside of the victim's law enforcement jurisdiction.
A server takedown across jurisdictions may take an extended period of time to complete, all while the victim is unable to operate normally.

Sinkholing is a process where the traffic is filtered using packet filtering upstream from the victim. It requires the cooperation of Internet Service Providers (ISPs) and depends on the traffic having a *unique signature* that allows filtering the attack traffic, while allowing legitimate traffic to pass.
So, in sinkholing context, ISPs play a fundamental role in preventing DDoS attacks.
Best practices for network operators have been described in over 200 separate documents by the Internet Engineering Task Force (IETF). In particular tactics such as restricting forged (spoofed) traffic, preventing hosts from being used as reflectors, managing routing advertisements are all useful in preventing some of DDoS attacks.

Despite the research and industry efforts at defeating botnets, hackers have been very successful at evading defense tactics by continuously modifying their bot design and C&C architecture.

3 Denial of Service (DoS) and Distributed Denial of Service (DDoS)

Availability is one of the main components of IT security, along with *confidentiality* and *integrity*. Denial of Service (DoS) and Distributed Denial of Service (DDoS) are considered serious threats mainly because they are attacks which are quite easy to be carried out today and because they exhaust the computing and communication resources of a victim system within a short period of time with little or no advance, making these resources *unavailable* for legitimate users.
The aims of DoS and DDoS attacks is to render a network incapable of providing normal services by targeting the victim with a stream of useless traffic packets (which seems legitimate) that slows or even stops its connection or its processing capabilities.
An important aspect of these attacks is that they do not cause any corruption to the data, but they damage the victim by forcing it to not provide the resources required by the final user.
The main targets of these attacks are servers, default gateways and even end-user systems.

Definition - Denial of Service: *A non-authorized user of a specific service is said to deny a service to another authorized user if the former user makes the specified service* ***unavailable*** *to the latter one, for a period of time which exceeds the intended (and advertised) waiting time.*
The use of single computer to carry out these attacks is termed as ***Denial Of Service*** *attack whereas the use of multiple computers (called zombies) to carry out a coordinated attack on one or more systems or services is termed as* ***Distributed Denial of Service*** *attack.*

It's easy to understand that among the two attacks, DDoS represent the major threat.
A DDoS attack is launched in two phases:

Step 1: an attacker builds an attack network which is distributed and consists of thousands of different computers which are infected by using Trojans, worms, etc. These compromised machines are named as *zombies*, *bots*, or *attacking hosts* while the main controller machine is called *master*. The master-zombie relationship works similar to the client-server architecture.

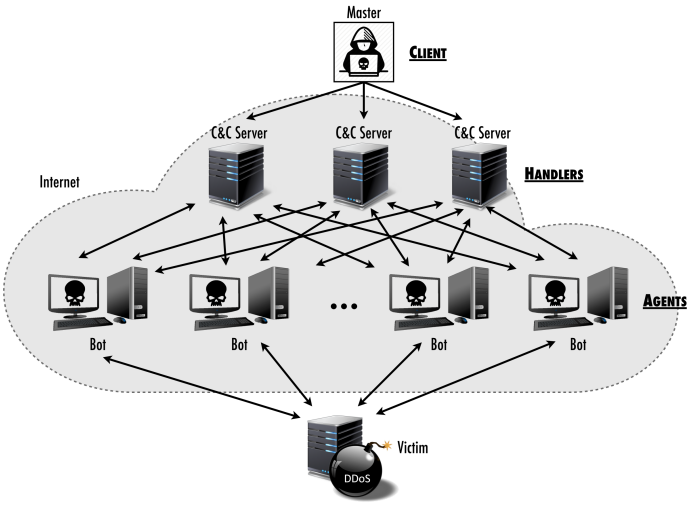


Figure 7: Example of an attack network

Step 2: the attacker exploits the attack against one or more systems by flooding a tremendous volume of traffic towards victims either under the command of the master, or automatically.

The main issue is that it can be very difficult to detect DDoS attacks because the zombies may be situated across the globe. As a result, they cannot be differentiated from the legitimate traffic.
This is because in a DDoS attack, it's possible to differentiate two different types of victims: the services under attack are the *primary victims*, while the compromised systems used to launch the attack are often called the *secondary victims*. The use of secondary victims in a DDoS attack provides the attacker with the ability to exploit a much larger and more disruptive attack while remaining anonymous since the secondary victims actually perform the attack making it more difficult for network forensics to track down the real attacker.
The master-zombie architecture is an **Agent-Handler** model that consists of *client* (attacker), *handlers* (C&C servers), and *agents* (zombies).

- **Client:** the system used by the attacker to communicate with the rest of the DDoS network.
- **Handlers:** software packages located in some servers throughout the Internet that the attacker's client uses to communicate with the agents.
- **Agents:** software that exists in zombie systems that will eventually carry out the attack. Depending on how the attacker configures the DDoS attack network, agents can be instructed to communicate with a single handler or multiple handlers.

The attacker communicates with any number of handlers to identify which agents are up and running, when to schedule attacks, or when to upgrade agents. Meantime, the owners and users of the agent systems typically have no knowledge that their system has been compromised and will be taking part in a DDoS attack. Usually, attackers will try to place the handler software on a compromised router or network server that handles large volumes of traffic. This makes it harder to identify messages between the client and handler and between the handler and agents.

The DDoS attack network can be modeled also with another type of architecture, the so called **IRC-based model**: the IRC-based DDoS architecture is similar to the Agent-Handler model except that instead of using a handler program installed on a network server, an *Internet Relay Chat* (IRC) communication channel is used to connect the client to the agents.

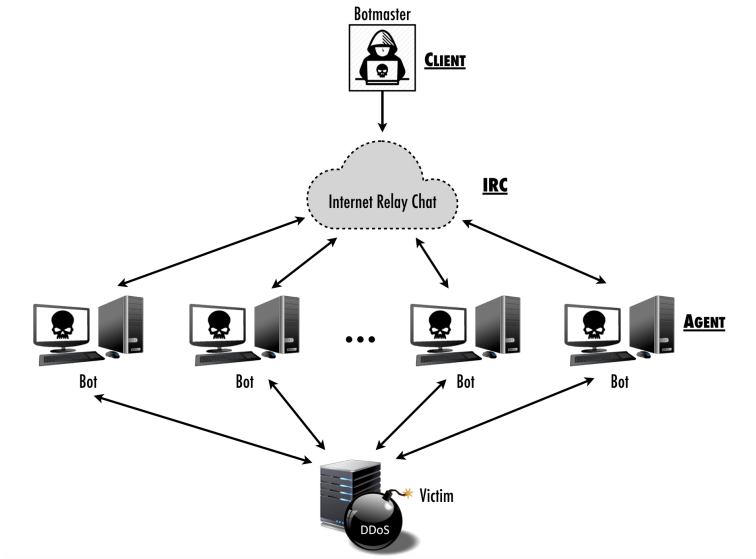


Figure 8: DDoS IRC-based Attack Model

An IRC channel provides an attacker with additional benefits such as the use of legitimate IRC ports for sending commands to the agents. This makes tracking the DDoS command packets more and more difficult. Additionally, IRC servers tend to have large volumes of traffic making it easier for the attacker to hide his presence and also if a zombie system is discovered, it may not reveal the identification of other zombies or the attacker. Moreover, to further avoid discovery, the attacker can frequently switch IRC channels.
Another advantage is that the attacker does not need to maintain a list of the agents, since he can log on to the IRC server and see a list of all available agents which are currently running, since the agent software installed in the IRC network usually communicates to the IRC channel to notify the attacker when the agent is up and running.
In both IRC-based and Agent-Handler DDoS attack models, we refer to the agents (zombies) as *secondary victims* and the target of the DDoS attack as the *primary victim*.
Well-designed agent software uses only a small proportion of resources (memory and bandwidth) of the infected system so that the users of secondary victim experience minimal performance impact when their system participates in a DDoS attack.

To build an attack network, the attacker typically looks for computers that are poorly secured, such as those not having been properly patched. In general, a vulnerable host can be compromised via two types of approaches.

- **Approach 1:** entice users to run malicious programs, such as a `Trojan horse`, a `virus` or a `spyware` carried in malicious emails, files, or web pages.
- **Approach 2:** use automated malicious programs, such as `worms` that can automatically scan vulnerable remote computers. The vulnerability in these computers is then exploited to allow the attacker to break into and install DoS attacking programs that further automatically scan other hosts, install backdoors and flood packets, and so on. Because of the self-propagation, a large attack network can quickly be built to include hundreds or thousands of computers.

The most common types of DDoS attacks experienced today are:

- **Volumetric Attacks**: they are those attacks that aim to consume all the *available bandwidth* in a network. It's the most frequent type of DDoS attack making up for about 65% of the total reported (according to Arbor Studio Report [2]).

  Volumetric attacks use multiple zombie systems to flood the network layers with a substantial amount of seemingly legitimate traffic.
  This depletes the bandwidth of the network causing **congestion** and drives network operations to become excessively slow (or simply not working).

- **Application-Layer Attacks**: they target web application packets in order to disrupt the transmission of data between hosts. Application-layer attacks include about 17% of all the reported DDoS attacks [2]. An important aspect of application layer attacks is that they can come from a single machine, which causes less traffic to be generated, allowing these attacks going under the radar of **detection systems**.

  An example of Application-layer attack is the so-called **HTTP Flood**: it uses multiple zombie systems to force a target web server or application to consume an excessive amount of resources when responding to an `HTTP request` (`GET` or `POST`).
  From an attacker's point of view, an HTTP Flood attack is an attractive threat since it does not consume a great deal of bandwidth to pull down a server and since it has mimic human-user behavior; so, it's very difficult to detect and block.

The primary targets of Application-Layer Attacks are `DNS` and `HTTP` services, but also `HTTPS` and `SMTP` are also commonly targeted [2].

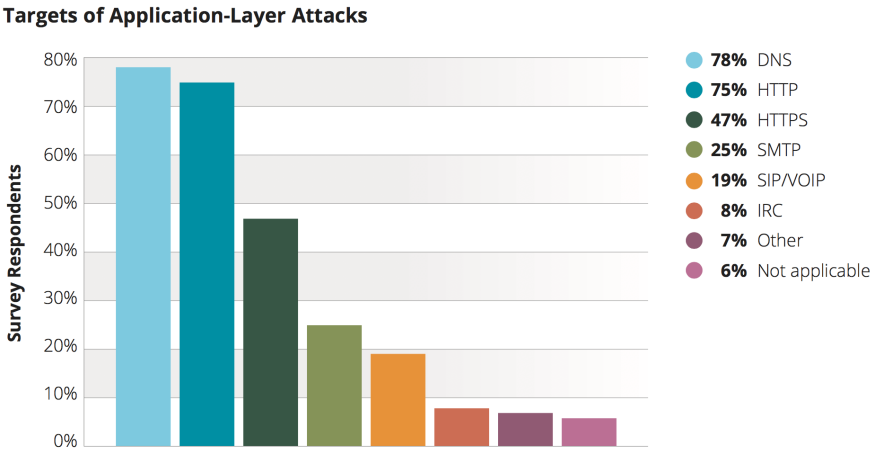


Figure 9: Percentage of Application-Layer Attacks against OSI Layer 7 protocols.

- **State-Exhaustion Attacks**: also known as *protocol attacks*, they target the connection state tables in firewalls, web application servers, and other infrastructure components.

  The most common state-exhaustion attack is the **Ping of Death (PoD)**: type of attack in which the attacker floods the victim system with malformed or malicious pings. The aim of the attacker is to destabilize or halt the victim system using the malformed and oversized packets using just a simple ping command. The maximum packet length of an IP packet is 65.535 bytes. So, during the attack, a ping packet of 65.536 bytes is defragmented into multiple IP packets or fragments and the they're sent to the target as fast as possible.
  When the target server reassembles the large packet, a *buffer overload* typically occurs: the target attempts to respond to the pings, consuming more and more bandwidth, eventually causing memory *buffer overflows* allocated for the packet and leading to denial of service for legitimate packets or eventually to the crash of the victim system.

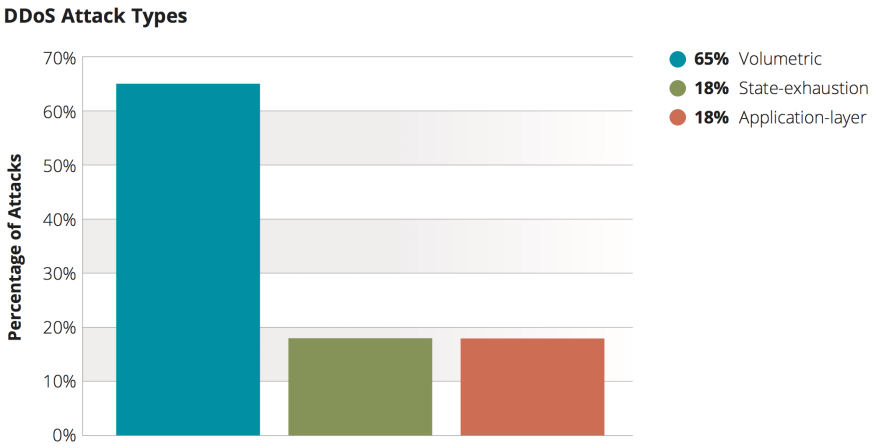


Figure 10: Percentage of DDoS Attack Types

3.1 DDoS Motivation

The reasons motivating attackers to exploit a DDoS attack against a company/organization and its related services are various. They can be categorized according to their nature in different classes:

- **Financial/Economical Gain:** attacks launched for financial gain are often, the most dangerous and difficult to stop. These are mainly concern of corporations and require more technical skills and experience.

  An interesting example is the Linux *worm* named `Linux.Darlloz`, earlier used to target Internet of Things (IoT) devices (i.e. routers, TV boxes, security cameras, printers, etc.) and now upgraded and equipped with an open source crypto currency mining tool called `cpuminer` that could be used to mine Crypto Currencies (like Bitcoins).

- **Invariably Slow Network Performance:** the attacker launches an attack to block the resources of victim system, which slowdowns the performance of the system and consumes the network bandawidth.

- **Revenge:** attackers of this kind are normally with lower technical skills and are frustrated individuals, carry out these as a response to a perceived injustice.

- **Ideological Belief:** attackers in this category are inspired by their ideological beliefs to attack their targets. This category is currently one of the major incentives for the attackers to launch DDoS attacks.

- **Intellectual Challenge:** the attack targets some systems for experiment and learn how to launch various attacks. They are usually young hacking enthusiasts who want to show off their competencies.

- **Service Unavailability:** the attacker overloads the services offered by the victim system through unwanted or fake traffic.

- **Cyberwarfare:** this kind of attackers normally belongs to the military or terrorist organizations of a country and they are politically motivated to attack a wide range of critical sections of another country.

4 DDoS Taxonomy

Since there is a wide variety of DDoS attacks, it's useful to classify them in the following taxonomy:

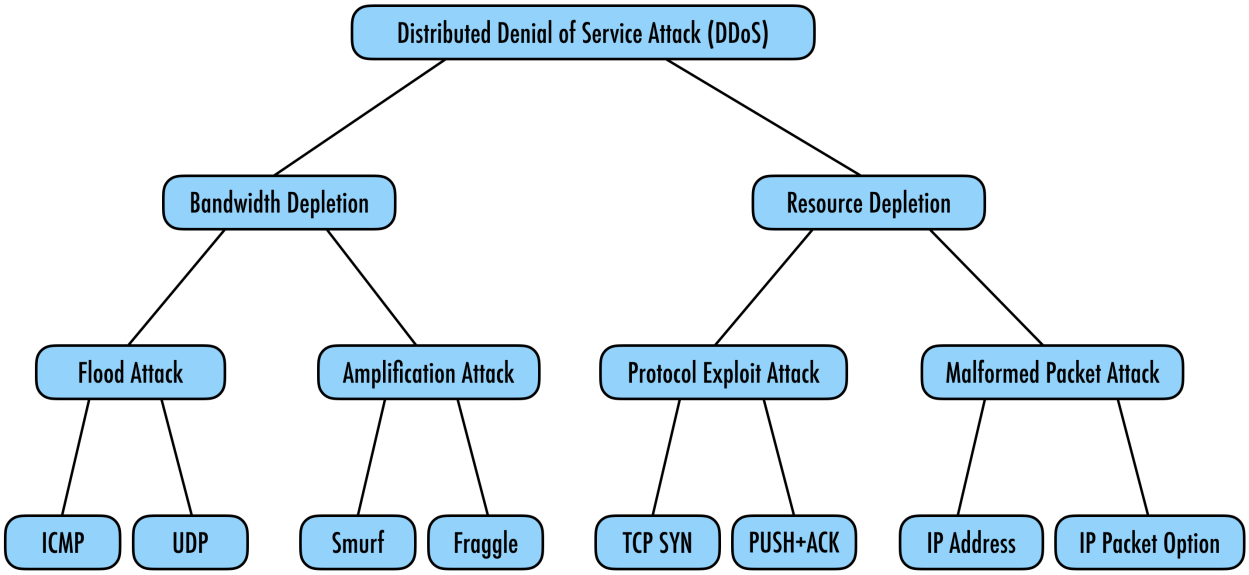


Figure 11: Classification of DDoS Attacks.

There are two main classes of DDoS attacks: **bandwidth** and **resource depletion** attacks.

1. **Bandwidth Depletion:** the attack is designed to flood the victim network with unwanted traffic that prevents legitimate traffic from reaching the victim system by exhausting all the available network bandwidth.

2. **Resource Depletion:** attack is designed to bind all the available resources of the target system (such as CPU, memory, I/O bandwidth, disk space, or any combination of them), making the victim unable to process legitimate requests for service.

4.1 Bandwidth Depletion Attacks

4.1.1 Flood Attacks

A Flood Attack involves zombie systems sending large volumes of traffic to a victim system to congest the victim's available network bandwidth with IP traffic. The victim system slows down, crashes, or suffers from saturated network bandwidth, preventing access by legitimate users.
Flood attacks have been launched using both **User Datagram Protocol** (**UDP**) and **Internet Control Message Protocol** (**ICMP**) packets.

- ◇ **ICMP Flood Attack:** also called **Ping Flood Attack**, it simply exploits ICMP, which enables users to send an `echo` packet to a remote host to check whether it's alive.
  An ICMP flood attack involves zombie systems flooding the victim's network with `ping` packets for a reply from the victim. In general, sending packets as fast as possible without waiting for replies, it results in the saturation of the victim's network bandwidth (both outgoing and incoming), resulting in a significant overall system slowdown.
  During the attack, the source IP address of the ICMP packet may also be spoofed helping

to hide the identity of the secondary victims since return packets from the victim system are not sent back to the zombies, but to the spoofed addresses (thereby anonymizing the attack).

- ◇ **UDP Flood Attack:** a large number of `UDP` packets are sent to either random or specified ports on the victim system. The victim system tries to process the incoming data to determine which applications have requested data. If the victim system is not running any applications on the targeted port, it will send out an `ICMP` packet to the sending system indicating a `destination port unreachable` message. If the number of packets being delivered to ports of the victim is large, the host resources will be depleted and it will lead to *unavailability*.
  Often, the attacking UDP Flood DDoS tool will also spoof the source IP address of the attacking packets. This helps hide the identity of the secondary victims since return packets from the victim system are not sent back to the zombies, but to the spoofed addresses (thereby anonymizing the attack).
  UDP flood attacks may also fill the bandwidth of connections located around the victim system. This often impacts not only the victim, but also the systems located near the target system. `UDP Unicorn` is one of commercially-available software packages that can be used to perform a UDP flood attack.

4.1.2 Amplification Attacks

An amplification attack involves the attacker (or zombies) sending messages to a broadcast IP address, causing all systems in the subnet reached by the broadcast address to send a reply to the victim system. In this kind of attacks, the *broadcast IP address* is used to amplify and reflect the attack traffic, and thus reducing the victim's bandwidth.
The attacker can send the broadcast message directly, or use the agents to send the broadcast message to increase the volume of attacking traffic. If the attacker decides to send the broadcast message directly, this attack provides the attacker with the ability to use the systems within the broadcast network as zombies without needing to infiltrate them or install any agent software.

- ◇ **Smurf Attack:** this attack is similar to the **ICMP Flood** [4.1.1], since also the Smurf attack uses the `echo` response mechanism of ICMP and flooding the victim with `ICMP echo-reply packets`. This attack uses a *network amplifier* (a system supporting broadcast addressing) and *IP broadcasting* in which, when a packet is sent to an IP broadcast address from a machine on the local network, then the packet is delivered to all machines on that network. The attacking hosts forge `ICMP echo requests` (similar to `ping`) having the spoofed victim's address as source address and the broadcast address of these remote networks as the destination address. Since the packets are sent at broadcast address, the packets are received by all the nodes within the broadcast address range in the network. Then, each node responds back to the victim machine (since the source IP address is spoofed). This creates a large amount of `echo response packets` sent to the target (the original packet is amplified tens or hundreds of times), thereby making the victim's network unstable and causing network congestion.
- ◇ **Fraggle attack:** in the Fraggle attack the attacker sends packets to a *network amplifier*, sending `UDP echo packets` to the port that supports character generation (`chargen`, port 19 in Unix systems), with the return address spoofed to the victim's echo service (`echo`, port 7 in Unix systems) creating an infinite loop. The UDP Fraggle packet will target the character generator in the systems reached by the broadcast address. These systems each generate a character to send to the echo service in the victim system, which will send an echo packet back to the character generator, and the process repeats. This attack can generate more bad traffic and cause more damage than a Smurf attack.

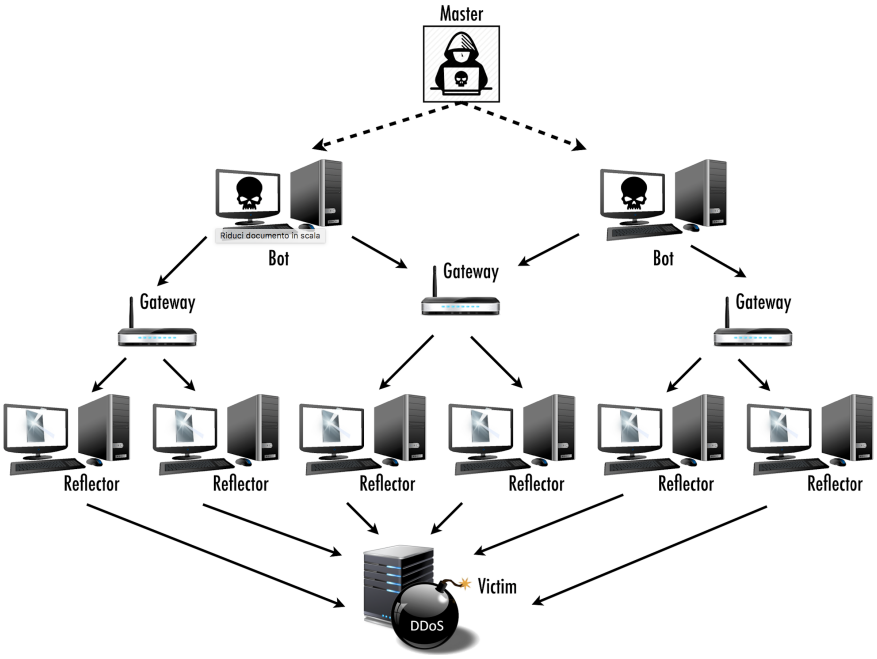


Figure 12: ICMP Smurf and UDP Fraggle Attack.

4.2 Resource Depletion Attacks

DDoS Resource Depletion attacks involve the attacker sending packets that misuse network protocol communications or are malformed.

4.2.1 Protocol Exploit Attacks

There are two main attacks which exploit protocol vulnerabilities: one misusing the `TCP SYN` (`Transfer Control Protocol Synchronize`), and the other misusing the `PUSH+ACK` protocol.

- **TCP SYN Attack:** in a TCP SYN attack, the attacker instructs the zombies to send false `TCP SYN requests` to a victim server in order to tie up the server's processor resources, and hence prevent the server from responding to legitimate requests.
  The TCP SYN attack exploits the three-way handshake between the sending system and the receiving system by sending large volumes of TCP SYN packets to the victim system with spoofed source IP addresses (since the return address can be overridden and falsified), so the victim system responds to a non-requesting system with the `ACK+SYN response` and it waits for an `ACK` from the client that will never come.
  When a large volume of `SYN requests` are being processed by a server and none of the `ACK+SYN responses` are returned, the server eventually runs out of processor and memory resources, and is unable to respond to legitimate users.
  In practice, the SYN Flood attack floods the victim with *half open connections*: this kind of incomplete connection is stored in the Backlog Queue. After 75 seconds the incomplete connection is removed from Backlog Queue and it is disconnected. However, if the client sends lots of `SYN packets` before the server removes incomplete connections from Backlog Queue, then Backlog Queue in the server is **overflowed**. In this case the server cannot accept other TCP connections at all.

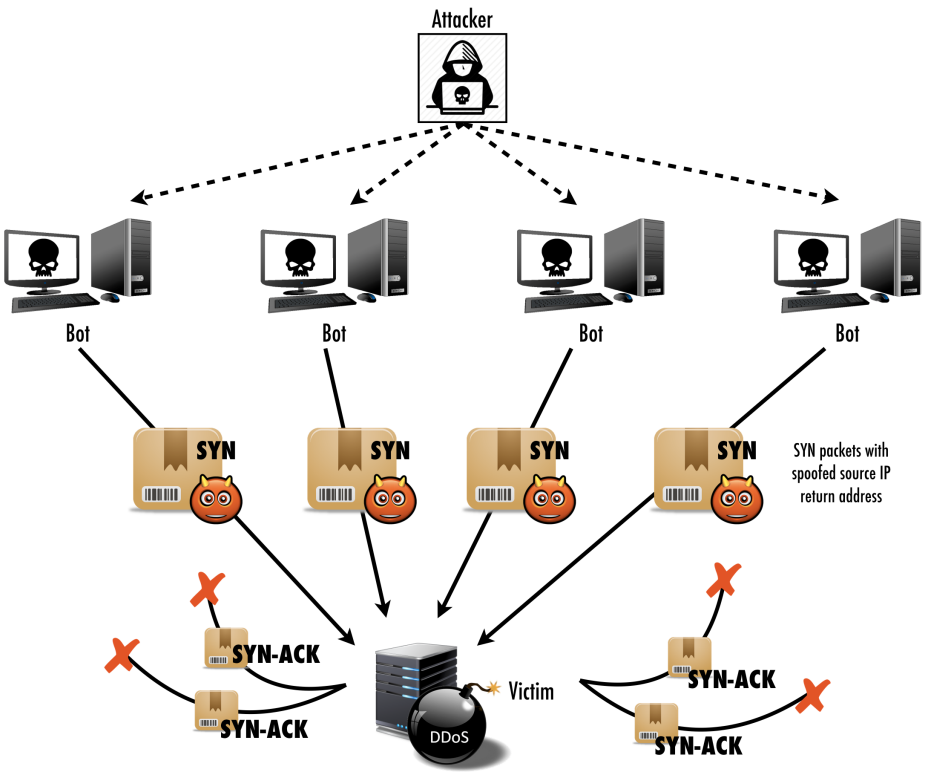


Figure 13: TCP SYN Flood Attack.

- **PUSH+ACK Attack:** the attacking agents (zombies) send TCP packets with the `PUSH` and `ACK` bits set to one. These triggers in the TCP packet header instruct the victim system to unload all data in the TCP buffer (regardless of whether or not the buffer is full) and send an acknowledgement when complete. If this process is repeated with multiple agents, the receiving system cannot process the large volume of incoming packets and the victim system will crash.

4.2.2 Malformed Packet attacks.

A malformed packet attack is an attack where the attacker instructs the zombies to send incorrectly formed IP packets to the victim system in order to crash it.
There are at least two types of malformed packet attacks: `IP Address Attack` and `IP Packet Options Attack`.

- **IP Address Attack:** in the IP address attack, the packet contains the same source and destination IP addresses. This can confuse the operating system of the victim system and can cause the victim system to crash.

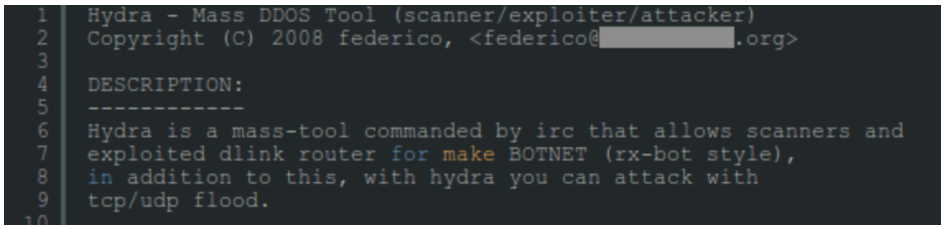
- **IP Packet Options Attack:** in the IP packet options attack, a malformed packet may randomize the optional fields within an IP packet and set all quality of service bits to one so that the victim system must use additional processing time to analyze the traffic. If this attack is multiplied, it can exhaust the processing ability of the victim system.

5 IoT Malware

The definition of IoT Malware directly follows by the devices which are targeted by the attack: what makes IoT malware different from any other similar types of malware is that it targets a specific class of devices, the Internet of Things.
With the growth of IoT, malware targeting this class of devices has naturally grown with it, finding a recent success thanks to lacks of IoT security.
This has led to the origin of different **malware families** properly designed to infect IoT devices and to include them as part of thingsbots exploiting IoT-based Distributed Denial of Service attacks.
The IoT malware is still evolving and new versions are daily found by Cyber Security engineers.
A list of IoT botnet malware is discussed below, but obviously it cannot be complete, because young hackers and cyber criminals works day by day modifying and updating the known malware to exploit new vulnerabilities or to infect different types of IoT devices.

5.1 Linux.Hydra

`Hydra` is the earliest known malware targeting IoT devices. It is a router malware that operated automatically appeared in 2008 (then published on the author's blog in April 2011) in the form of an open-source botnet framework. It was managed by Internet Relay Chat (IRC) and its main target was to access routers using brute force methods for the purposes of performing DDoS attacks.
Getting access to the router was possible by either using a built-in list of default passwords or with the use of a D-Link authentication bypass exploit.
Before compiling the code, the malicious user had to edit one of the source files in order to provide the URL address of the C&C IRC server as well as the link to download the malicious binary.
All further `MIPS` malware found in the wild is more or less based on Hydra.


```
Hydra - Mass DDOS Tool (scanner/exploiter/attacker)
Copyright (C) 2008 federico, <federico@[redacted].org>

DESCRIPTION:
------------
Hydra is a mass-tool commanded by irc that allows scanners and
exploited dlink router for make BOTNET (rx-bot style),
in addition to this, with hydra you can attack with
tcp/udp flood.
```

Figure 14: Description of Hydra included in Readme file.

5.2 Psyb0t

`Psyb0t` was found in-the-wild targeting routers and DSL modems and it was discovered by the Australian researcher Terry Baume in January 2009, when he noticed strange attacks in his firewall logs. It drew still more attention in March of the same year when a botnet consisting of infected routers performed a DDoS attack on the DroneBL portal. It's estimated that over 100.000 compromised devices were infected by this malware.
Psyb0t is an Internet Relay Chat (IRC) bot based on Command&Control servers, which connects to the server, joins the channel and looks up the topic for commands. When it was discovered and the bot was still active, the command included in the topic was to scan for all network devices in a specified address range in the network and try to gain access to routers/modems by using the `Telnet` and `SSH` accesses using simple brute force attack with predefined 6.000 usernames and 13.000 passwords. If the attempt was unsuccessful, the bot tries a password gained through a known

authentication bypass exploit. This meant that the malware self-proliferated across the network.
After gaining access to the shell, Psyb0t searches the list of running processes looking for its own process. If the router is not infected already, Psyb0t uses either `wget` or `tftp` to download its body from the server and execute it. Next it sends a message about the successful attack to the botnet owner and goes on to the next host.
Apart from code for proliferating, Psyb0t also contains numerous commands for initiating various types of DDoS attacks, brute forcing the router password, executing shell commands and brute forcing access to other services like MySQL, PHP MyAdmin, FTP servers, etc.
As the router firmware is usually read-only, Psyb0t resides exclusively in RAM and a simple reboot results in the machine being disinfected. Nevertheless, vulnerable devices will be re-infected all over again unless the password has been changed and/or the firmware patched to new version.
Growth in public interest was probably the reason of Psyb0t deactivation: on March 2009, the creator changed commands in the IRC channel topic to make all bots to kill processes and exit.
Psyb0t was designed to work on Linux-based systems running on the `MIPS` platform (e.g. `OpenWRT`, `DD-WRT`, `MontaVista`, `VxWorks`, etc.) mainly used in embedded devices (i.e., network devices).
It used also a primitive obfuscation to prevent analysis: the binary itself was packed with the UPX packer; so, the headers were corrupted and they could not be analyzed.

5.3 Chuck Norris

`Chuck Norris` is a IRC bot found infecting routers and DSL modems that in many ways reminded of `Psyb0t`. It was discovered by researchers of the Masaryk University (Czech Republic) in the beginning of 2010. The C&C centre of this new botnet was located in Italy and the strings and function names inside the malicious binaries also indicate that the code is of Italian origin. The malware contained the text *"In nome di Chuck Norris"*, a phrase which occurs several times in the code, and thus the botnet's nickname became `Chuck Norris`. Kaspersky Lab's signature for this malware is `Trojan.Linux.Uteltend`, which comes from another string in the file: a command used to prevent the `utelnetd` service from running on an infected machine.
The Chuck Norris and Psyb0t binaries have in common: same brute forcing method (set of default passwords), same platform (`Linux MIPSel`), same packer (`UPX`), some headers corrupted in exactly the same way, both have IRC functionality, same method of encrypting the most sensitive information, similar proliferation and similar use of D-Link routers' authentication bypass exploit.

5.4 Tsunami

In early March 2010 a new IRC bot was discovered in Latin America and it was classified as a variant of `Backdoor.Linux.Tsunami` as it shared many of the distinguishing features associated with the `Linux Kaiten/Tsunami` open source DDoS tool.
However, some parts of its code are also very similar to the Chuck Norris binaries: the function for obfuscating strings is the same and also the encryption key. Moreover, some of the strings and IP addresses are in fact identical.
The innovations introduced by Tsunami were the attempts to change the DNS settings and to block the range of ports from 22 through to 80. It was mainly a Distributed Denial-of-Service (DDoS) flooder. However, it was also capable of performing other actions such as downloading additional files and executing shell commands in an infected system. These actions could grant the bot master almost a full control of the infected system.

5.5 LightAidra/Aidra

`LightAidra/Aidra` is an IRC-based mass scanning and exploitation tool supported on several architectures (`MIPS, MIPSEL, ARM, PPC, x86/86-64` and `SuperH`) that was discovered in 2013. According to its author, Federico Fazzi, the malware was introduced in early 2012 as an IRC-based tool to create botnets and, in addition to this, the attacker could perform some DDoS attacks with TCP flood. The malware was designed to search for target IoT devices (randomly or recursively) that run embedded forms of Linux with active Telnet connectivity and open telnet ports that could be accessed using known default credentials (or no password at all) and requires two servers to manage the entire botnet; one server that will host the various binaries used for infection and an IRC server to control the botnet actions. Some variants of Aidra can retrieve router passwords through a bug found on some outdated D-Link and Netgear devices.
The malware attempts to connect to a telnet port using default credentials and if it succeeds, it downloads and executes a script which removes other malware binaries and prevents the device from being compromised by other competing malware. Some variants attempt also to change the device credentials. The malware binaries are downloaded from the HTTP server into the RAM, and then the malware can be removed by rebooting the device (since the directories are stored in RAM). At this point, the infected device connects to an IRC C&C server, joins a channel, reads a topic, and follows the instructions.
Also if the LightAidra DDoS malware was developed in 2012, it appears to be still doing quite well in the wild: mainly because this bot relies on the fact that many consumer network devices that this bot targets use the default username and password (e.g., `admin/admin` or `root/admin`) and also because its source code is freely available on the Internet as open source project.

5.6 Carna

`Carna` is a botnet of 420.000 devices created by an anonymous hacker to measure the extent of the Internet and to get an estimation of the IP address usage in what the creator called the *"Internet Census of 2012"*. The data was collected by infecting Internet enabled IoT devices (in particular routers) with empty or default credentials. Collected data was compiled into a `.gif` image to display Internet use around the world over the course of 24 hours. The data gathered included only the IPv4 address space and not the IPv6 address space: of the 4,3 billion possible IPv4 addresses, Carna found a total of 1,3 billion addresses in use, including 141 million that were behind a firewall and 729 million that returned reverse domain name system records. The remaining 2,3 billion IPv4 addresses are probably not used.

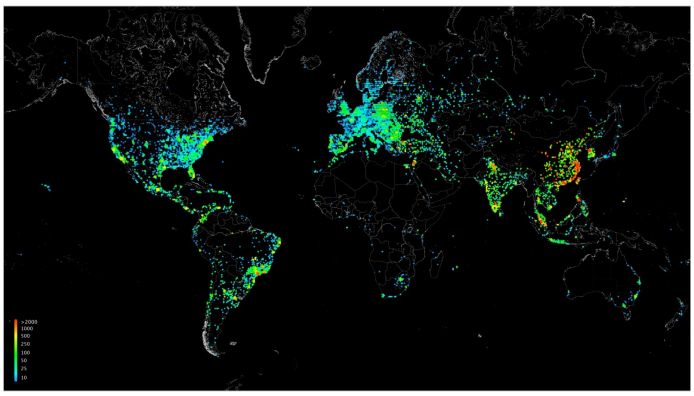


Figure 15: 420.000 Carna botnet clients active from March to December 2012.

5.7 Linux.Darlloz

`Linux.Darlloz` is an IoT worm discovered by Symantec in late 2013 that attacked IoT devices by exploiting an old PHP vulnerability (CVE-2012-1823) to access the system, to make a privilege escalation through default and common credential lists, to propagate through the network, and to establish a backdoor on the infected system. While the original malware only infected computers running Intel `x86` chip architectures, later versions were designed to target `ARM`, `PPC`, `MIPS` and `MIPSEL` chip architectures commonly used in IoT devices. After infecting the device, it drops the `Telnet` traffic via `iptables` configuration and terminates the `telnetd` process to block users from accessing the infected device using that protocol. The worm also scanned systems searching for Linux.Aidra malware and attempted to remove any files related to the threat and to block any ports used by Aidra for communication. Symantec found that Darlloz has infected more than 31.000 devices by February 2014. A newer version of Darlloz uses infected devices to *mine crypto-currencies* (Bitcoin, Mincoins and Dogecoins).

5.8 Qbot/Qakbot

`Qbot` is a network-aware worm capable of discovering credentials and creating backdoors of IoT systems. The malware, first discovered around 2009, continues to be adapted and employed by script kiddies and cybercriminals. Qbot leverages the Rig Exploit Kit against vulnerable websites to gain write access on the backend and to inject malicious JavaScript onto the site. To avoid suspicion, the malicious JavaScript may be appended onto the beginning or end of a legitimate JS. The Rig exploit kit is a two-tier model consisting of a gate and a landing page. The malware target Windows users which sessions are injected with the malware via a watering-hole attack, a drive-by download, or by delivery of malicious emails. Once installed on the system, the malware runs a network speed test and it sends an initial beacon, containing a list of installed software, user privileges, and the infected network external IP address, to the FTP server. The malware injects itself into a running `explorer.exe` process and it infects processes as they start up. The bot injects a DLL into processes that will extract its strings, configuration, APIs, and critical strings block into heap-allocated buffers, when run. The malware places system-wide inline hooks to intercept or modify network traffic, to modify or redirect browser queries, to infect new processes, and to hide its presence. Upon installation, Qbot mutate the appearance of the malware by self-recompiling or self-re-encrypting the malware as a type of *polymorphism*, an obfuscation mechanism meant to confound anti-malware application and research efforts. This polymorphism enables Qbot to avoid most anti-virus products because the malware updates itself to a new version every few days, and re-encrypts itself to remain undetectable for long periods of time. The malware can also detect whether it is running in a Virtual Machine sandbox and it can alter its behavior to avoid detection. Over a two-week investigation, BAE Systems discovered over 54.517 machines infected by the Qbot botnet (85% located in the United States).

5.9 Linux.Wifatch

`Wifatch` is an open-source malware which infects IoT devices witn embedded Linux (`ARM` 83%, `MIPS` 10% and `SH4` 7%) with weak or default credentials and that has been noted because is not used for malicious actions, instead for attempting to secure devices from other malware.
It was first discovered at the beginning of 2014 and it has been active since at least November 2014. Once infected, it removes other malwares and disables `Telnet` access while logging the message: *"Telnet has been closed to avoid further infection of this device. Please disable telnet, change telnet passwords, and/or update the firmware."* in the device logs.

Wifatch operates in a manner similar to a computer security system and it uses peer-to-peer network to update the malware definition and deletes the rest of malware which remain in the IoT devices.

5.10 TheMoon

`TheMoon` is an IoT P2P worm discovered by Johannes Ullrich of SANS in February 2014. This malware specifically targets Linksys and Asus routers and exploits a command execution vulnerability on their hard-coded `iptables` rules while parsing `ttcp_ip` parameter value sent in a `POST` request. This vulnerability allows the malware to send malicious `UDP` packages to vulnerable routers, bypass authentication procedures, and execute code on the device taking it over from its real owner. In subsequent exploitation steps, TheMoon will add new firewall rules to ensure the hacker can access the device from his desired location and to block other IoT malware from exploiting the same flaw and hijacking the device from his botnet.

5.11 Spike/Dofloo

`Spike/Dofloo` is a backdoor/DDoS malware discovered around mid-2014 found targeting 32-bit and 64-bit Windows, Linux based PCs as well as IoT devices based on MIPS and ARM architectures. It has been used in several attacks aimed at organizations in Asia and the United States. Akamai observed that one of the attacks to have peaked at 215 Gbps. This malware appears to be developed by China based group and can launch DDoS attacks with various payloads including SYN floods, UDP floods, DNS query floods, and GET floods against targeted organizations.

5.12 BASHLITE

`BASHLITE` (also known as `Gafgyt`, `Lizkebab`, `Torlus`, `LizardStresser`, `Bash0day` and `Bashdoor`) is a malware which infects Linux systems in order to launch DDoS attacks, but it's also designed to easily cross-compile to various computer architectures. Originally it was also known under the name `Bashdoor`, but nowadays this term refers to the exploit method used by the malware. It has been used to launch attacks of up to 400 Gbps. In 2015 its source code was leaked, causing a proliferation of different variants, and by 2016 it was reported that 1.000.000 devices have been infected. Of the identifiable devices participating in these botnets in August 2016 almost 96% were IoT devices (of which 95% were cameras and DVRs), roughly 4% were home routers and less than 1% were compromised Linux servers. The majority of the infected devices were located in Taiwan, Brazil, and Columbia. DVRs are high value bots because the devices are configured with open Telnet and other web interfaces, often rely on default credentials (or no credential at all) and are able to process high bandwidth, as is required to stream video.
The original version in 2014 exploited a flaw in the `bash` shell (the `Shellshock` software bug) to exploit devices running `BusyBox`. In 2015 the source code was leaked causing a proliferation of over a dozen variants. Now, exact capabilities of the malware differ between variants; the most common features generate several different types of DDoS attacks: it can hold open TCP connections, send a random string of junk characters to a TCP or a UDP port, or repeatedly send TCP packets with specified flags. They may also have a mechanism to run arbitrary shell commands on the infected machine. There are no facilities for reflected or amplification attacks.
BASHLITE uses a client-server model for command and control and the protocol used for communication is essentially a lightweight version of Internet Relay Chat (IRC). Even though it supports multiple command and control servers, most variants only have a single C&C IP-address hardcoded. The malware propagates via brute forcing, using a built-in dictionary of common usernames and passwords and it connects to random IP addresses and attempts to login, with successful logins

reported back to the C&C server. The malware conducts two scans to discover vulnerable devices to infect. The first attack vector utilizes the bots to port scan IP ranges for telnet servers and then it instructs them to brute force credentials in order to access and infect the device. The second attack vector employs external scanners to detect vulnerable devices and then infects those devices by using brute force on the credentials, by exploiting known security vulnerabilities, or by leveraging another attack vector. Once the attacker has compromised a device, the malware tools execute the `busybox wget` and `wget` commands to retrieve the DDoS payloads.
The malware does not identify the architecture of the compromised device; instead, it attempts to run different versions that have been compiled for different architectures, until one executes. Most BASHLITE attacks are simple UDP and TCP floods, though the malware does support a less used feature to spoof source addresses and some variants support HTTP attacks. BASHLITE is a predecessor to Mirai, and the botnets are now in direct competition for a diminishing pool of vulnerable IoT devices.

5.13 KTN-RM / Remaiten

`Remaiten` is a IoT Malware which combines the features of `Tsunami` and `BASHLITE` (`Gafgyt` variant). It infects embedded Linux IoT devices by brute forcing using frequently used default usernames and passwords combinations from a list. The C&C server interacts with the bots using IRC communications by an actual IRC channel, HTTP and Telnet. Remaiten is more sophisticated than its predecessor malwares: it first discovers the victim's system architecture and then download the appropriate malicious code; so, in practice, it can adapt itself based on the IoT device and the type of attacks it wants to launch. The infection is exploited in different phases: first of all, the malware choose a random public IP address and tries to connect to it through `Telnet`; then it makes a brute force dictionary attack trying common username/password credentials; now, if it's logged in, Remainten discovers the victim's architecture by issueing the command `cat $SHELL` and by parsing the output (since `cat $SHELL` prints the ELF executable); at the end, it downloads and executes the corresponding malware.
The bots receive instruction from C&C server through the IRC channel to execute different malicious activitues such as: download binaries/sample from remote IP and/or from Internet, execute shell commands, exploit DoS and DDoS attacks, list and terminate existing processes (also kill other bots), send files to remote IP and scan for other vulnerable devices on port 23 (`Telnet`).

5.14 Mirai

`Mirai` is one of the most predominant DDoS IoT botnet in recent times derived by `BASHLITE` (`Gafgyt` variant). This botnet is definitely the next step in IoT DDoS botnet malwares, however not as sophisticated as Remaiten but most effective. At its peak, Mirai infected 4.000 IoT devices per hour and currently it is estimated to have little more than 500.000 infected active IoT devices. Mirai botnet is famous for being used in the currently record DDoS attack with about 1,1Tbps of traffic and with 148.000 IoT devices involved.
Mirai targets vulnerable IoT devices (mostly CCTV cameras, DVRs, and home routers).
The source code for Mirai has been published by the author Paras Jha using his online pseudonym Anna-senpai on the hacking community Hackforums as open-source. Since the release of the Mirai source code, the number of infected devices increased from 213.000 to 483.000 in just two weeks.
Based on the IP addresses one can identify that the Mirai infected IoT devices are distributed in over 164 countries with highest densities in Vietnam, Brazil, US, China and Mexico.

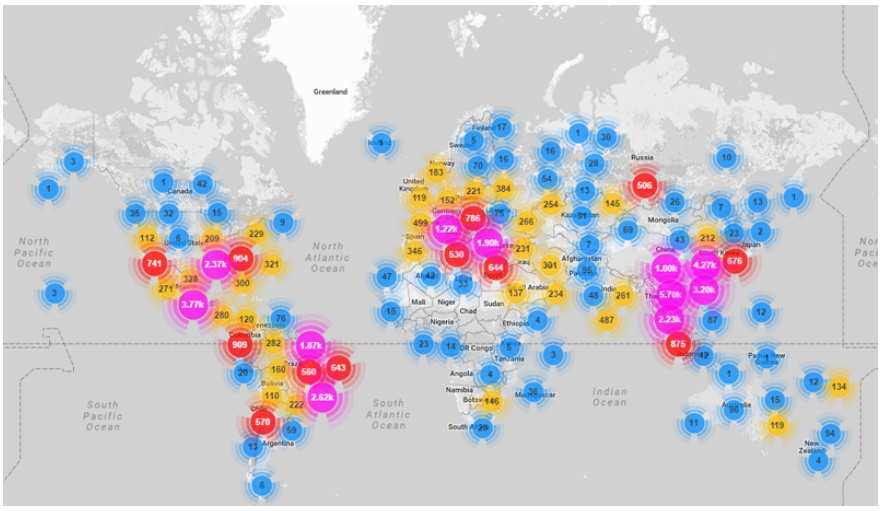


Figure 16: Locations of Mirai-infected devices.

The infection of the device is made by a brute force dictionary attack on `Telnet` using an hardcoded list of default username/passwords. Once Mirai has infected the device, the botnet kills `Telnet`, `SSH` and `HTTP` services to avoid other remote connections to the device, preventing the device to be infected from other malware. It also scan the device searching for other malware (found in known locations) to eventually erase or block it. The botnet self-propagate itself by searching for other vulnerable IoT devices; to prevent detection, Mirai has an hardcoded list of IPs that should be avoided (Loopback, U.S. Postal Office, Department of Defense, Multicast, etc.).
Mirai can exploit different DDoS types of traffic, generating `GRE IP`, `GRE ETH`, `SYN+ACK`, `STOMP`, `DNS`, `UDP`, or `HTTP` traffic against a target. The majority of these data types are used to flood traffic on the network (OSI Layer 3-4).
An important novelty introduced by Mirai (which is also a difference from its predecessor `BASHLITE`) is that Mirai uses an encrypted channel for communication between C&C server and bots.
Recently, Mirai has been found to be enhanced to infect Windows devices, helping hackers hijack even more devices. This enhanced Mirai malware could also identify and compromise database services like `MySQL` and `MicrosoftSQL` running on different ports, allowing hackers to steal the DB.

5.15 Linux/IRCTelnet

`IRCTelnet` (also known as `newAidra`) is the first discovered successor to Mirai and it's a IRC botnet ELF malware aimed at IoT devices with IPv6 capabilities. At the time of its discovery (Italian origin), Linux/IRCTelnet infected an average of 700 bots per day.
IRCTelnet has been created by using the base source code of `LightAidra/Aidra` to build a new botnet malware redesigned and modified with respect to the modern IoT landscape, borrowing some key concepts of the most important botnets like `Tsunami` (for IRC protocol), `BASHLITE` (for the infection techniques, i.e. telnet brute force access and code injection) and `Mirai` (for the botnet's IoT credential list). The botnet is using UDP and TCP flood along with other series of DDoS attack methods in both IPv4 and IPv6 protocol. Like the other IoT botnets, the malware does not establish persistence and it can be removed from infected devices by rebooting the device and clearing the RAM. The malware improves upon its predecessors with two new features: the ability to attack systems running IPv6 and the ability to communicate with bots over an IRC channel, rather than through traditional C&C infrastructure.

5.16 Hajime

`Hajime` is an IoT worm that has been discovered by Sam Edwards and Ioannis Profetis, two security researchers for Rapidity Networks, in October 2016 when they found what appeared to be a Mirai infection, but in reality, it was something new entirely. The worm uses a three-stage infection mechanism, and self-propagates. Stage 0 occurs on an already infected system, from where Hajime starts to scan random IPv4 addresses. The worm launches a brute-force attack on port 23, trying to log in on the other end with a series of username and password combinations hardcoded in its source code. If the IP doesn't have port 23 open, or the brute-force attack fails, Hajime moves on to a new IP. In the case of a successful connection, Hajime will execute four commands (`enable`, `system`, `shell`, `sh /bin/busybox ECCHI`) that will allow the worm to tell if it infected a Linux-based system. Hajime can target the following platforms: `ARMv5`, `ARMv7`, `Intel x86-64` and `MIPS`. At this point, the worm enters Stage 1, and it starts the download of a foreign binary, a 484-byte ELF program (Linux binary). The Hajime worm launches this binary, which has only one purpose, to open a connection to the attacker's server and write the bytes it receives to a new binary, launching it into execution when the data transfer finishes. Then there is the Stage 2: the binary uses the `DHT` protocol to connect to a P2P botnet, from where it downloads further payloads (binaries, modules) via the `uTP` protocol (both DHT and uTP are BitTorrent protocols). Once the target has been infected Hajime blocks access to ports 23, 7547, 5555, and 5358 to prevent other malware, such as Mirai, to infect the target and it does not perform other malicious actions. Hajime's mode of operation outdoes Mirai's (list of built-in username/password combinations to brute-force random IPs) and seems to have borrowed many tricks from other IoT malware as well (multi-stage infection mechanism, P2P botnet). Hajime also uses several obfuscation techniques including deleting itself from the infected device's filesystem, changing its process name after execution, and disguising itself as a common Telnet daemon program. So, Hajime seems to have included the best parts of other IoT malware, and it is far more sophisticated than anything else seen targeting IoT devices today. Based on the hardcoded credentials included in the worm's source code, Hajime targets routers, DVRs, and CCTV systems, just like Mirai.

5.17 Amnesia

`Amnesia` botnet targets an unpatched remote code execution vulnerability that was publicly disclosed over a year ago in March 2016 in DVR (digital video recorder) devices made by TVT Digital and branded by over 70 vendors worldwide. This vulnerability affects approximately 227.000 devices around the world with Taiwan, the United States, Israel, Turkey, and India being the most exposed. Amnesia is the first Linux malware to adopt *virtual machine evasion techniques* to defeat malware analysis sandboxes. Virtual machine evasion techniques are more commonly associated with Microsoft Windows and Google Android malware. Similar to those, Amnesia tries to detect whether it's running in a `VirtualBox`, `VMware` or `QEMU` based virtual machine, and if it detects those virtualized environments it will delete itself and will try to delete Linux root directory ("/"), User home directory (" /") and the current working directory ("./"). Amnesia exploits the remote code execution vulnerability by scanning for, locating, and attacking vulnerable systems. A successful attack results in Amnesia gaining full control of the device and attackers could potentially use the Amnesia botnet to launch broad DDoS attacks (similar to Mirai ones).

5.18 BrickerBot

Over a four-day period, Radware's honeypot recorded 1.895 Permanent Denial of Service (PDoS) attempts performed from several locations around the world. Its sole purpose was to compromise

IoT devices and corrupt their storage. Besides this intense, short-lived bot (called `BrickerBot.1`), Radware's honeypot recorded attempts from a second, very similar bot (called `BrickerBot.2`) which started PDoS attempts on the same date.
Also known as *phlashing*, PDoS is an attack that damages a system so badly that it requires replacement or reinstallation of hardware. By exploiting security flaws or misconfigurations, this type of cyber attack can destroy the firmware and/or basic functionalities of a system. It is a contrast to the DDoS attack, which overloads systems with requests meant to saturate resources through unintended usage.
The BrickerBot PDoS attack used `Telnet` brute force dictionary attack (same as Mirai) to breach a victim's devices. The malware, once the device is infected, it does not try to download a binary, but it performs a series of commands that would ultimately lead to corrupted storage (e.g., reduce partitions to 1 sector), followed by commands to disrupt Internet connectivity (e.g., delete default gateway), device performance (e.g., delete the entire file system), and the wiping of all files on the device (e.g., limit the maximum number of kernel threads to 1). The use of the `busybox` command means this attack is targeted specifically at Linux/BusyBox-based IoT devices which have their Telnet port open and exposed publically on the Internet. These are matching the devices targeted by Mirai or related IoT botnets. The first PDoS attempts originated from a limited number of IP addresses spread around the world.
In parallel, Radware's honeypot recorded over 333 PDoS attempts with a different command signature. The source IP addresses from these attempts are TOR Nodes and hence there is no identifying the actual source of the attacks. It is worth noting that these attacks are still ongoing and the attacker/author is using TOR nodes to conceal its bot(s). The commands executed by this second version of the PDoS bot at the end are identical to the first one and try to remove the default gateway, wipe the device and disable TCP timestamps as well as limiting the max number of kernel threads to one. However, extra commands were added to flush all iptables firewall and NAT rules and add a rule to drop all outgoing packets.
It also executes a *fork bomb* (also called *rabbit virus* or *wabbit*): it's a DoS attack in which a process continually replicates itself to deplete available system resources, slowing down or crashing the system due to resource starvation. So the second attack is potentially more dangerous and harmful.

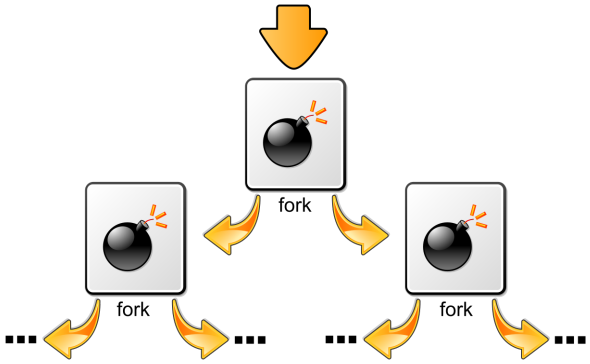


Figure 17: Fork Bomb - Processes continually replicate themselves, potentially causing a DoS.

Both PDoS attacks started the same day and approximately the same time (March 20, 2017), but while the BrickerBot.1 attack have stopped, the attacks from BrickerBot.2, which are less dense but better concealed using TOR nodes, are still active and ongoing.

Malware	Target	Platform	Access Method	Purpose	First Appearance
Hydra	Routers	Linux	Brute-Force (Default Passwords on Telnet) or Authentication Bypass	DDoS Attacks	2008
PsybOt	Routers and DSL Modems	Linux MIPSel	Brute Force (Default Passwords on Telnet/SSH) or Authentication Bypass	DDoS Attacks	2009
Chuck Norris	Routers and DSL Modems	Linux MIPSel	Brute Force (Default Passwords on Telnet/SSH) or Authentication Bypass	DDoS Attacks	2010
Tsunami	Routers and DSL Modems	Linux MIPSel	Brute Force (Default Passwords on Telnet) or Authentication Bypass	DDoS Attacks, DNS Redirection, Command Execution	2010
LightAidra	IoT Devices with Embedded Linux	MIPS, MIPSEL, ARM, PPC, x86/86-64, SuperH	Brute Force (Default/No Passwords on Telnet) or Bug Exploitation on D-Link devices	DDoS Attack, Command Execution	2013
Carna	IoT Devices	Linux	Brute Force (Default/No Passwords)	"Internet Census"	2012
Darlloz	IoT Devices with Embedded Linux	Intel x86 (1st version) x86, ARM, MIPS, MIPSEL, PPC (Later)	PHP Vulnerability (CVE-2012-1823)	Crypto-Currency Mining, Command Execution	2013
QBot	Vulnerable Websites	Windows	Rig Exploitation Kit	Command Injection Command Execution	2009
Wifatch	IoT Devices	ARM, MIPS, SH4	Brute Force (Default/No Passwords on Telnet)	Security Improvement	2014
TheMoon	Linksys and Asus Routers	Firmware iptables	Command Execution Vulnerability	Command Execution	2014
Spike/Dofloo	Windows, Linux PCs, IoT devices	32 and 64-bit Windows, Linux, MIPS and ARM	Brute Force (Default/No Passwords)	DDoS Attacks, Command Execution	2014
BASHLITE	IoT Devices, Routers, Linux Servers	Designed to be compiled for all Embedded Linux architectures	Brute Force (Default/No Passwords) Shellshock Vulnerability (BusyBox), Bashdoor	DDoS Attacks, Command Execution	2014
Remaiten	IoT Devices with Embedded Linux	Embedded Linux (tipically MIPS and ARM)	Brute Force (Default/No Passwords on Telnet)	DDoS Attacks, Command Execution	2016
Mirai	IoT Devices with Embedded Linux, Windows	Embedded Linux, Windows	Brute Force (Default/No Passwords on Telnet)	DDoS Attacks	2016
IRCTelnet	IoT Devices with IPv6 Capabilities	Embedded Linux	Brute Force (Default/No Passwords on Telnet) Code Injection	DDoS Attacks	2016
Hajime	IoT Devices with Embedded Linux	Embedded Linux, ARM, Intel x86-64, MIPS	Brute Force (Default Passwords on Telnet)	Security Improvement	2016
Amnesia	TVT Digital Digital Video Recorders	Embedded Linux	RCE Vulnerability, Remote Code Execution	DDoS Attacks Command Execution	2016
BrickerBot	Linux/Busybox based IoT Devices	Embedded Linux	Brute Force (Default Passwords on Telnet)	PDoS	2017

Figure 18: Overview of IoT Malware.

6 DDoS Defense Techniques

The security policy used to protect a network requires to be regularly updated in order to defend against new attack patterns and malwares that exist on the Internet.
The KNOW YOUR ENEMY policy could greatly applies to IoT security:

> The *more knowledge* we have of the different emerging **malware**, the *better* it's possible to identify the new **threat**.

In order to obtain this knowledge, it is of crucial importance to collect accurate, concise, and high-quality information about malicious activities. This activity is known as **Malware Collection**. The analysis of individual pieces of the collected malware samples provides a more in-depth understanding of their functionality and their purpose.

The traditional lines of defense (such as Intrusion Detection Systems (IDSs) and Dynamic Firewalls) does not provide a coverage on detecting novel and emerging patterns of attacks. This is because they are typically obtained by Signature-based and Behavior-based approaches.

- ◇ **Signature-based Detection**: it's performed by IDS using a set of predefined signatures and attack patterns to detect intrusions. It's a fast detection method, but it presents the limitation of detecting only known attack patterns and may not detect intrusions exploited by attacks with new behaviours.

  Furthermore, the detection of some malware may also be missed due to the high complexity level of the sample (e.g. malware packers, which obfuscate the malicious code by encryption).

- ◇ **Behavior-based Detection**: also known as **Dynamic Detection**, it observes the normal behavior of the system and detects any deviation from usual patterns. The usual patterns are extracted by analyzing the functionalities requested by the standard usage of the system.

  If the Behavior-based IDS detects a sudden increase of particular activity (e.g. network traffic, CPU usage, etc.), then it will raise an alarm. This allows also for new attacks to be detected.

  - \+ **Pro**: maybe some of new detected attacks are the same ones that have not been detected by Signature-based IDSs (not having previous pattern signatures available).
  - − **Cons**: it has high false alarm rate (it can falsely detect normal activity as intrusion).

Malware Collection and Malware Analysis in a large scale can provide enhance statistics of different attack patterns, trends and rates of attacks that exist in daily network traffic.
There are several methods that can be applied to collect malware, such as the use of disk forensics, online malware repositories, anti-virus applications and honeypots.

- ◇ **Disk Forensics:** it involves forensic analysis of disks searching for any traces of malware. This could take a significant amount of time depending on the size of the disk and it could become a very time consuming task. It also requires the acquisition of the infected disk.

- ◇ **Online Malware Repositories:** web pages that acts as shared repositories containing a large amount of malware samples which are been uploaded from different sources.
  This includes manual uploads made by users or automatic ones by different IDS sensors.
  Since the sources may be *distributed*, the use of such online sources of information can provide a huge amount of new samples of from all around the world.

Typically, they are free to use, they can provide in-depth malware analysis and they can offer the comparison of the obtained results with anti-virus vendors ones.

The most popular shared repositories are: `VirusTotal`, `VirusSign`, `KernelMode`, and the `Kaspersky Security Lab`.

- **Anti-virus:** programs installed and running directly on the system which try to detect hidden malware. Typically, each antivirus uses a single type of detection and most of them uses the so called virus scanners and scanning typically means searching into the host for known code patterns.
  This is an inherent **weakness**: a scanner can either give a false alarm, if a pattern code happens to match part of a normal file's code, or worse, it might not detect the code of a new sample at all. To avoid this, an antivirus program's database must be periodically updated. Another problem is due to the fact that installing and running an antivirus software can use a lot of computer memory and hard disk space, slowing down the PC.

- **Honeypot:** well designed system that attracts hackers (or botnets) into it. By luring the intruder into the system, it is possible to monitor, analyze, understand, watch and track the processes that are started and executed on the system by the attacker.
  In other words, an honeypot is a trap machine which looks like a real system in order to attract the attacker. By monitoring this machine, it's possible to gain a lot of informations which can be used to create more secure systems.

  The most important feature of honeypots is that they allow cyber security engineers to discover **new threats**: since they are systems that want to be attacked emulating different types of vulnerabilities, the hackers (or botnets) will attempt to exploit the honeypot.
  The methods and tactics used for executing such exploits are logged and stored.
  So, the security engineer can obtain relevant information directly from this real-time obtained data by using forensic tools and, maybe, discovering new samples of malware that have never been seen before.

Malware Collection and Malware Analysis offer a methodology to discover and study an attack when it had been just executed (or during the execution of the exploit). But the network security of an environment can be improved by using some technologies that could detect and block an intrusion. The main systems that can be adopted to this objective are: Intrusion Detection Systems (IDS), Firewalls, Anti-Virus (AV), Intrusion Prevention Systems (IPS) and Honeypots. They can be classified by *areas of operation* which are:

- **Prevention:** any action that discourages hackers and makes exploits to the system infeasible.
- **Detection:** process of locating threats against confidentiality, integrity and availability of the system.
- **Reaction:** execution of reactive measures after detecting dangerous unauthorized actions.

Traditionally, firewalls monitor and control network traffic based on predetermined, static rules. They act as a barrier between secure and insecure networks. IDS analyze whole packets (both header and payload) looking for malicious signatures; when a known event is detected an alert is generated. IPS can be understood as an extension of IDS which not only issue alerts, but actively reject packets and dynamically add new rules. Anti-Virus software is also signature based, but it acts on a file-level and tries to capture infections by a trojan or virus. Complete log-monitoring (application logs, service logs, system logs etc.) increases the awareness of network infrastructure.

Problems are detected by evaluating the log files statistically for known text patterns that indicate important events. Cyber security standards prevent attacks by defining security guidelines for the software and infrastructure, such as a clean configuration, hardened kernel and recent updates. Nowadays a clear distinction between these concepts became difficult, as security solutions are hybrids that combine several concepts in order to minimize the individual trade-off.

Prevention

- **Honeypot:** it adds a little value to prevention as it does not protect against security breaches. At most, it inhibits attacks because attackers are wasting time attacking the honeypot instead of the target production systems. However, it fails against automated toolkits and worms which scan and try to exploit every vulnerable element found in the network.

- **IDS:** it might deter, however it will not actively prevent automated attacks.

- **Firewall:** it is the best concept for prevention, since it hides services and blocks communication on unused ports or from suspicious IP-address-ranges; so, intruders are kept out or do not find any services to communicate with.

- **Cyber Security Practices:** the best prevention is achieved by good cyber security standards and practices, (such as disabling unnecessary or insecure services, frequent update and patch policies, strong authentication mechanisms, etc.).

Detection

- **Honeypot:** it is very difficult to detect attacks on production systems because the attacks simply submerge in the vast amount of production activity. Honeypots can simplify the detection process since all their incoming connections are suspect by nature and therefore detected as unauthorized (with almost no false positives and negatives).

- **IDS:** as the name suggests, intrusion detection systems are designed to detect attacks. However, their effectiveness is highly dependent on the signature quality and if not properly maintained, administrators can be overwhelmed with false positives, starting to ignore all messages (also true positives). On the other hand, false negatives are also possible if no appropriate signature exists as new exploits are used.

- **Firewall:** considering a traditional firewall concept, without advanced abilities such as flow statistics (e.g., packet count), a firewall is significantly limited in the detection of attacks.

Reaction

- **Honeypot:** it accelerates the reaction to attacks because the study of an attack is substantially easier, as attack data is not mixed with data of production activity. Furthermore, honeypots can be taken off-line for the review, which enables a full forensic analysis, trying to understand the exploit, which is the first step to patch the corresponding vulnerabilities.

- ◇ **IDS:** it's primarily passive, that means that it analyzes packets and creates alerts. Traditionally, it does not alter the network packet flow and it only informs administrators about malicious activities. The reaction to new attacks may include the creation of new signatures.

- ◇ **Firewall:** since it does not detect attacks, it cannot react dynamically to them. However, administrators can define new firewall rules manually, which are usually simpler then IDS rules and are based on packet meta-information rather than payload and packet flow.

As discussed above, each different security concept tend to have a specific objective: primarily, honeypots implement reaction, IDSs achieve detection, and firewalls accomplish prevention. But, each of these concepts do not exclude each other (they are complementary). To achieve all objectives (prevention, detection, reaction), hybrid solutions such as IPSs have been developed.
In the following figure, the level of contribution of security systems to each area of operation is denoted by the number of "+" (general, not focused on single implementations).

Objective	Prevention	Detection	Reaction
Honeypot	+	++	+++
Firewall	+++	++	+
Intrusion Detection Sys.	+	+++	+
Intrusion Prevention Sys.	++	+++	++
Anti-Virus	++	++	++
Log-Monitoring	+	++	+
Cyber Security Standard	+++	+	+

Figure 19: Contribution of security systems to areas of operation.

According to the argumentations made before, it can be seen that the use of specific honeypots is the most promising technique for acquiring malware samples and to study emerging threats against IoT devices. Hence, the analysis of honeypots and of their contribute for gathering information about different IoT malware samples will be the next focus of this project.

7 IoT Honeypot

To counter the problem of finding a methodology which can provide a coverage on detecting novel and emerging patterns of IoT botnets exploiting DDoS attacks, a branch of research concerning the Cyber Security is deepening the study of particular systems called **honeypots**.
An honeypot is a sort of trap set to detect or deflect attempts at unauthorized use of information systems. Generally, it consists of a computer that appears to be part of the network but which is actually isolated and protected, and which seems to contain information or functionalities that would be of value for attackers.
A few months ago, Arbor Networks (the Cyber Security division of Netscout) has strengthened its network by introducing a *cloud-based honeypot* that monitors the activities of malicious scan for IoT devices which could be compromised by IoT botnets.
The results are impressive: in only two weeks, the honeypot has registered 1.027.543 login attempts (18.054 login attempts/hour) of which 819.198 failed, from 92.317 different IP addresses. Moreover it was found that `Telnet` protocol is subject to almost 4 times higher number of attacks compared to the `SSH` protocol: 756 attempts/hour for `SSH` against 2.762 attempts/hour for `Telnet`.
It is also interesting the **geographical spread** of the DDoS attacks: the top two target countries are the USA and China. The third most common target is France, which may come as a surprise to some. However, data centers in France provide hosting for many organizations based all around the world. Looking at the sources of DDoS attacks, we can see that the USA, South Korea and China hold the top positions.

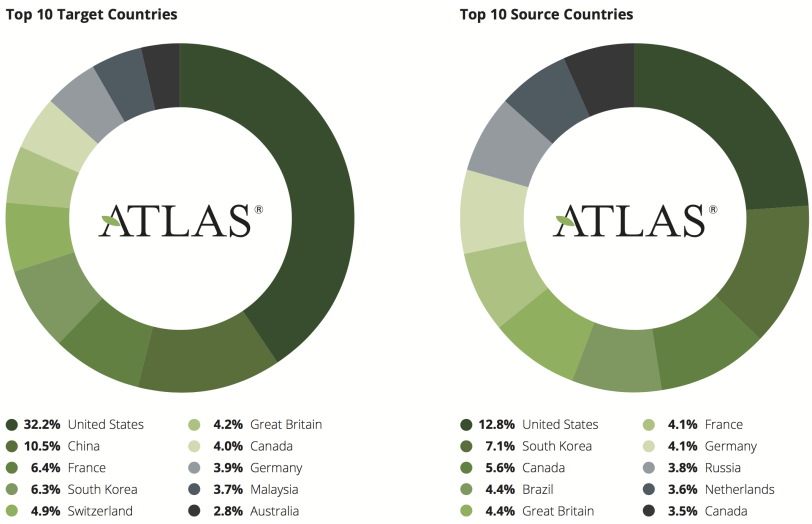


Figure 20: Top 10 Target/Source nations of DDoS attack attempts.

The Arbor Networks honeypot is an example of the fundamental informations that these kind of systems can gain when exposed over the Internet. Under this point of view, an honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource and that does not have production value, since the primary value given by the honeypot is information about malware samples and patterns which is possible to obtain.
In general, an honeypot is a system in which there are files and directories just like a real computer. However, the aim of the honeypot is to attract hackers to fall into it trying to watch and follow their behavior once they think they've exploited a vulnerable system and trying to capture the samples downloaded by botnets once they've infected a system/device.

An honeypot can be seen as a fake system that looks like a real one and which main functions are:

- ◇ Mislead the attention of the attacker from the real network, in a way that the main information resources could be not compromised;
- ◇ Capture new viruses or worms samples for future analysis;
- ◇ Create attacker profiles in order to identify their preferred attack methods in order to identify a pattern behavior of the hacker;
- ◇ Detect new vulnerabilities and threats of various operating systems, devices, environments and programs which are not clearly identified at the moment.

So, honeypots are a different technology with respect to other security systems since they are not only finding one solution to a particular problem, but also they are eligible to apply on a variety of security problems and finding several approaches for them (e.g., they can be used to log malicious activities in a compromised system, or they can be used to learn new threats, etc.).

7.1 Advantages and Disadvantages

Advantages

+ Honeypots capture information about attacks into logs and by exploring them, it is possible to find hidden information about the exploits and attack patterns.
+ Honeypots are designed to capture anything that interacts with them, including tools or tactics never seen before (0-Days) and new solutions can be created by analyzing them.
+ Honeypots only deal with incoming malicious traffic; so, the collected information is smaller (no huge data storage) and higher-value, since it is only a part of the whole traffic.
+ Fewer false positives with respect to other security solutions (e.g., IDS), since only attack traffic is detected (no legitimate traffic).
+ Honeypots require minimal resources: any computer (e.g., Pentium III,128 Mb RAM) can be used to build an honeypot system. Thus, it does not cost additional budget to companies.
+ Honeypots are simple to understand, to configure and to install.
+ Do not require known attack signatures (unlike IDS).
+ Honeypots will detect an IPv6-based attack the same way it does with an IPv4 attack.

Disadvantages

− Honeypots have a limited vision: an honeypot can only scan and capture data when hacking activity is destined to interact directly with them. If there is an attack against neighboring systems, the honeypot will not be able to identify it unless the attacker or the threat interacts directly with the honeypot at the same time.

− Honeypots suffer of fingerprinting: it is easy for an experienced hacker to understand if he's attacking a fake system or a real one and fingerprinting allows the attacker to distinguish between these two.

− Honeypots lead to a risk: an hijacked honeypot system (if not properly designed) may be used by the attacker to reach other systems in the same network, trying to infect them. This can be very dangerous.

7.2 Types of Honeypots

Honeypots can be classified according to their *aims* and *level of interactions*.
A first distinction between different systems can be made by considering their *aims*: there are two different types of honeypots, which are **Research Honeypots** and **Production Honeypots**.

7.2.1 Research Honeypots

Research honeypots are mostly used by military, research and government organizations, capturing a huge amount of information. Their aim is to discover new threats and learn more about Blackhat hackers' motivations and techniques.
The objective of this class of honeypots is to learn and study how to better protect a system from all sorts of attack patterns and threats and not to bring any direct value to the security of an organization, since they are not implemented to protect networks.
A great deal of current attention is focused on Research honeypots, which are used to gather information about the intruders' actions. An interesting example is the `Honeynet Project` that is a non-profit research organization focused in voluntary security using honeypots to gather information about threats in cyberspace.

7.2.2 Production Honeypots

Production honeypots are used to protect organizations in real production operating environments, to protect the company from attacks. They are implemented inside the production network of a company (parallel to data networks or IT Infrastructures), to improve the overall security. They are capturing a limited amount of information (tipically, *low interaction* honeypots are used). Thus, security administrator watches the hacker's movements carefully and tries to lower the risks that may come from it towards the company.
Using production honeypots, it could imply a **risk**: while testing the security of honeypot in the existing systems in an organization, unexpected actions may happen (such as misusing other systems using honeypot features). If the network administrator is not aware of this problem, they put organization in a big trouble.
Production honeypots are constantly gathering more importance due to the detection tools they provide to companies and because of the way they can complement network and host protection.

A second classification among honeypots can be made by considering their *level of interation*.
By considering this feature as distinctive element, there are three main categories of honeypots available today: **Low Interaction**, **Medium Interaction** and **High Interaction**.
These classes are defined based on the services (*interaction level*) provided by the system to hackers.
In practice, the level of interaction stands for how much the hacker will be able to interact with the system once infected.
More amounts of data the analysts would like to gather, they require more level of interaction; but more level of interaction brings more risks into the network security as well. So, it's important to make the right choice of system that will be used by considering the level of interaction based on the purpose of the experiment that one would like to examine.

7.2.3 Low Interaction Honeypots

Low Interaction honeypots present to hackers emulated services with a limited subset of the functionality they would expect, with the intent of detecting sources of unauthorized activity. With

these kind of system, the analyst can get the least amount of data compared to other honeypots. Generally, the implementation process of a Low Interaction honeypot consists of installing an OS emulation software (i.e. `VMWare Workstation`), choosing the OS and services to be emulated, establishing a monitoring strategy and let the software operate by itself in a normal manner. For example, an FTP service (listening on port 21) emulated on a Low Interaction honeypot is probably emulating an FTP login or will possibly support additional FTP commands but it does not represent a target of critical importance due to the fact that it is not possibly linked to a FTP server containing sensitive information. Depending on purpose, a low interaction honeypot can be a very simple (e.g. only monitoring for incoming connections) or a more complicated system dedicated to downloading new samples of malware. Since these components are restricted, the risk of being compromised by an intruder is not big: first of all, there is no operating system to deal with and the system works exclusively emulating operating systems and services (not offering real functionalities). In practice, attacker's activities are limited to the honeypot's level and quality of emulation: emulated services mitigate the risk of penetration, containing the intruder's activities; the attacker can never gain access to a real OS that could be used to damage other systems inside the network. The main advantage of a Low Interaction honeypots is due to their simplicity and to the fact that they tend to be easy to understand, configure, use and maintain, with minimum risks. In addition, there is better control of attacker activities, since the hacker is constrained by the emulation of a software running on a host operating system. On the other side, it is relatively simple for an attacker to detect a Low Interaction honeypot due to the fact that a skilled intruder can detect emulation capabilities of services.

7.2.4 High Interaction Honeypots

High interaction are the most advanced honeypots and they let the hacker interact with a real operating system, allowing the attacker to perform quite anything with the goal of capturing the maximum amount of information on the attack techniques. Any command or end-user application which is expected to be installed, it's available on the system (enriched with monitoring functions); in general, there is little (or no) restriction placed on what the hacker can do once the system is compromised. So, more information can be captured on the hacker's activities and behaviors. High interaction honeypots constitute a complex solution because they involve the utilization of operating systems and real applications implemented in real hardware running in a normal way, without using emulation software. For example, if the honeypot needs to be implemented on a real Linux system running a FTP server, a real Linux OS needs to be built on the computer and a real FTP server will need to be configured. Deploying a high interaction honeypot is ideal and advantageous if caching, tracking and monitoring the activities and the motivations of a human attacker is intended, since this approach allows the hacker to download files and install tools, eventually capturing large amounts of information. After a successful compromise of the honeypot, most of the attackers install a root-kit to hide their actions and to remotely control the compromised system. In addition, IRC-Bots are a common tool used to control the compromised system. Generally, the capture of this data is limited on systems that use low-interaction honeypots. However, in the era of automatic threats (i.e., worms, e-mail viruses, auto-rooters, etc.), high interaction honeypots used for gathering of samples of malicious code are not efficient and very risky: if the attacker detects and disables all the monitoring mechanisms, the system can be used for other hostile activity directed at internal systems which are not part of the honeypot, turning bait into a weapon. As a result of this, there is a need to implement additional technologies which prevent the attacker from damaging non-honeypot systems that denies the compromised system of its capabilities of becoming a platform to launch potential attacks. In addition, High Interaction honeypots are very time consuming and difficult

to maintain (due to low scalability): after each successful infection, the system must be cleaned and reinstalled (very intensive task) in order to observe subsequent attacks that are independent on the initial attack. This process, even with the support of virtualization, it's relatively slow and the honeypot is still unusable while reinstalling.

7.2.5 Hybrid Approach: Low + High Interaction

To cope with the problems concerning the low scalability of high interaction honeypots, hybrid approaches have been proposed. The Hybrid Approach introduces a two step structure by combination of low and high interaction components: in the first step, all traffic is processed by low interaction components that allow to cover a broad range of IP addresses. The low interaction honeypot filters out all previously unseen attacks and relay them to the network of the high interaction one. This seems to be an important advantage because the problem of the high deployment effort of high interaction honeypots can be alleviated. Thus, all previously known attacks can be discarded without elaboration by the high interaction system. In the second step, selected connections are relayed to high interaction component where they are analyzed. Only unknown attacks which are much more interesting are analyzed on the high interaction honeypot. In addition, hybrid approaches are able to alleviate the problem of denial of service attacks against high interaction honeypots because these attacks can be blocked at the first stage.
Thus, hybrid approaches try to combine the advantages of the two different technologies.

7.2.6 Medium Interaction Honeypots

Medium-interaction honeypots provide an expanded interaction from low interaction honeypots but less than high interaction ones: they provide partial implementation of services and do not allow full interaction with the system (as high-interaction honeypots). In medium interaction honeypots still operating system does not exist; but this time, more information and more complicated attacks from the hacker can be obtained. For example, a medium interaction system could partially implement the HTTP protocol to emulate a well-known vendor's implementation (e.g., Apache). As it is more advanced, it has more security holes (with respect to lower interaction honeypots) so that hacker can access the system.

Low-Interaction Honeypot	**High-Interaction Honeypot**
Emulates OSs and services	No Emulation; real OS & services provided
Simple and easy to install and deploy.	Can be complex to install and deploy
Minimal risk as the emulated services control what attackers can and cannot do.	Increased risk as attackers are provided with real operating systems to interact with.
Captures limited information which is mainly transactional data and very limited interaction.	Can capture far more information including new tools, communication and attacker keystrokes

Figure 21: Comparison of Low and High Interaction Honeypots.

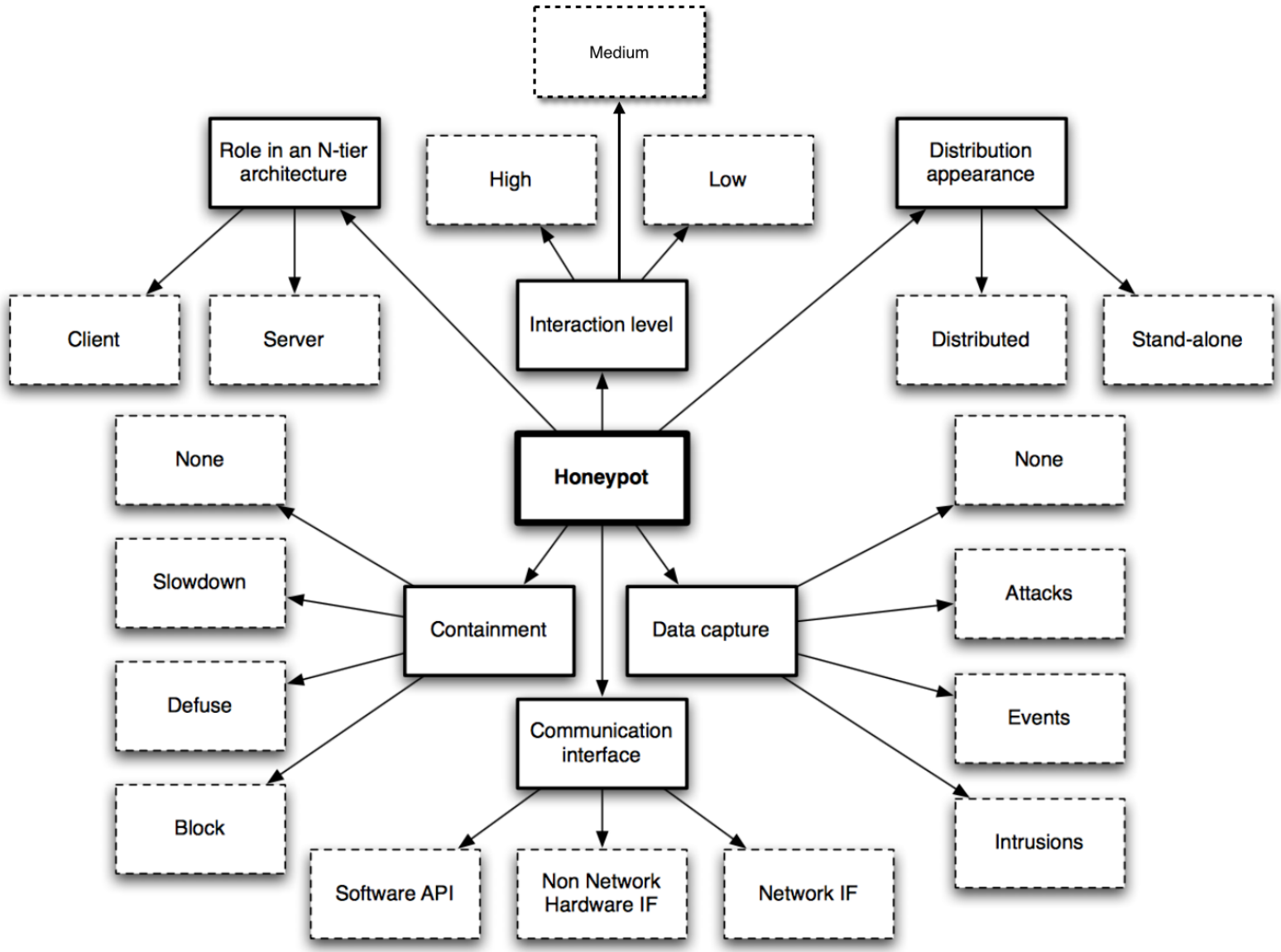


Figure 22: IoT Honeypot Taxonomy.

8 IoT Honeypot: State of the Art

A complete description of all the existing honeypot technologies is infeasible since new systems are introduced after few time and also because there is a big number of honeypots, each one with different contexts and purposes.
To have a (quite) complete list of all the available honeypot solutions, follow this link to GitHub:

https://github.com/paralax/awesome-honeypots

In this chapter I will present the most famous honeypots and the free ones which I've found more interesting according to their features, classified according to the **interaction level** offered.

8.1 Low Interaction Honeypots

8.1.1 Nepenthes

Nepenthes is a low interaction honeypot introduced in 2006 that inherits the scalability of low-interaction honeypots but at the same time offers a high degree of expressiveness. It's designed to emulate vulnerabilities (offering common services like `FTP`, `HTTP`, `TFTP` and `MSSQL`) that are exploited by botnets during an attack, and to detect and capture the related worms.
Nepenthes is based upon a flexible modular architecture which can be easily extended by adding new vulnerabilities (vulnerability modules) to configure the honeypot to adapt to other different types of threats. In classical terms, Nepenthes still realizes a low-interaction honeypot since it emulates the vulnerable services. The core (the actual daemon) of the honeypot handles the network interface and coordinates the actions of the other modules. The real work is carried out by several modules, which register themselves in the nepenthes core. There are several different types of modules: *vulnerability modules* which emulate the vulnerable parts of network services, *shellcode modules* that analyze the received `shellcode` by one of the vulnerability modules and extract information about the propagating malware, *fetch modules* which use the information extracted by the shellcode modules to download the malware from a remote location, *submission modules* that take care of the downloaded malware (e.g., save the binary on HD, store it in a DB, or sending it to anti-virus vendors, etc.) and *logging modules* which log information about the emulation process and help in getting an overview of patterns in the collected data.

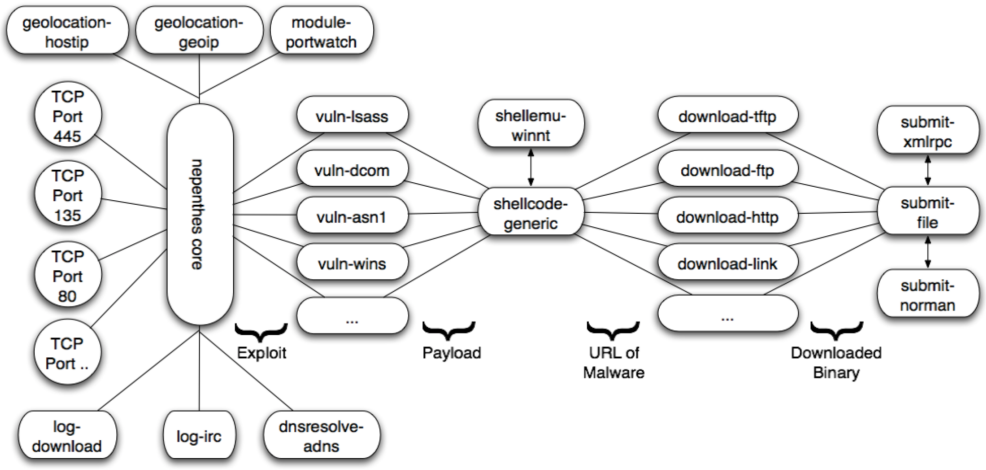


Figure 23: Nepenthes platform.

Vulnerability modules are one of the most important components of the whole Nepenthes architecture since they take care of the emulation process while logging modules allows to detect changes in attack patterns (indicator of new trends) and help to develop new modules. In case of a 0day (a vulnerability for which no information is available), this can enable a fast analysis since the first stages of the attack have already been captured and logged.
Although Nepenthes needs expert knowledge to emulate new vulnerabilities and to conduct a successful interaction with malware, developing a new vulnerability module to emulate a novel threat (or to capture a 0day exploit) is a straightforward process and demands only little effort. On average, writing of less than 500 lines of C++ code is required to implement the needed functionality.

8.1.2 Dionaea

The Dionaea low interaction honeypot is the successor of Nepenthes introduced in 2009. Its main purpose was easy gathering of worm samples, especially those directed to the Microsoft Windows. So, it implements full Windows remote procedure call (RPC) stack used by various Windows network services. Additionally, system can simulate other protocols, such us `MongoDB`, `MSSQL`, `MySQL`, `SIP`, `SMB`, `FTP`, `http` and `https`; so, it can be also suitable to emulate IoT devices.
Dionaea was innovative in the sense that it doesn't emulate specific network vulnerabilities, but rather the whole underlying network protocol stack. The main protocol offered for exploitation to threat agents is `SMB` (also known as `CIFS`), which uses the well known port 445. For analysis purpose and exploit detection, the suspicious `shellcode` payloads can be analyzed by a `libemu` library while the profiling is offered by `GetPC`. All the shellcodes run inside a `libemu VM` and the API calls made by the system get recorded. All data concerning activity of the systems are logged in `Postgress/SQLite` database stored locally. Dionaea can also send them to online malware analysis services (VirusTotal, CWSandbox, Anubis, Normal Sandbox). This DB can grow very large in size and essentially can make the analysis of the results a great challenge for the security researcher.
Thus, data visualization solutions can play a vital role in helping the defenders get a quick and detailed overview of the honeypot's captured activity. Dionaea utilizes Nepenthes' `XMPP` (eXtensible Messaging and Presence Protocol) backend in order to present some statistics of the honeypot's activity. The creator of Dionaea has included some rather basic visualization tools in the software, but these proved to be infective when the dataset grows large in size. So, easy access to data is provided by further developed Web panel called `carniwwwhore`, a web interface for Dionaea's DBs. Using Dionaea the analyst can identify new attempts of attacks and once the interaction with the malware is established, the system can download the malicious code using various protocols (e.g., `FTP`, `TFTP` and `http`) from the web address extracted by the analysis of the shellcode. During experiments, Dionaea proved to be very effective campturing a large number of samples also when used as single honeypot. The main drawback of the Dionaea was a poor emulation of `http` service.

8.1.3 Glastopf

Glastopf is an easy to deploy low-interaction honeypot web server introduced in 2008. It collects information about web application-based attacks such as local and remote file inclusion or SQL injections. Furthermore, Glastopf downloads files from links that are included in incoming requests and tries to respond such that the expectations of the attacker is satisfied and subsequent attacks are initiated. An important aspect of Glastopf honeypot is that it's developed by implementing a vulnerability type emulation instead of specific vulnerability emulation. Once a vulnerability type is emulated, unknown attacks of the same type can be handled. While the implementation may be more slow and complicated than the implementation of the single vulnerability, on the long-term

analysts benefit from this abstraction. The principle behind it is very simple: reply to the attack using the response the attacker is expecting from his attempt to exploit the web application. The attacker sends a malicious request, the honeypot processes the request and writes to a database or to the file system, and then it replies to the attacker; but the goal is to provide a proper reply for every request from the attacker, convincing the hacker that the system is vulnerable. So, Glastopf is a web-app honeypot capable of emulating thousands of vulnerabilities to gather data from attacks.

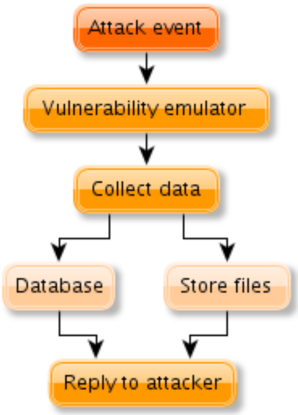


Figure 24: Glastopf functionality overview.

Glasif is the web interface for the Glastopf web application honeypot. It can be used to visualize and analyze data collected by the honeypot in real-time. The dashboard is the main page and provides the user with a quick overview about the activity of Glastopf. Various graphs show the attack and request volume of the wanted elapsed time. The Glasif dashboard shows statistics including the number of unique hits since installation and the uptime of the honeypot. It provides also an overview of the last five attacks as well as the top five attacking IP addresses.

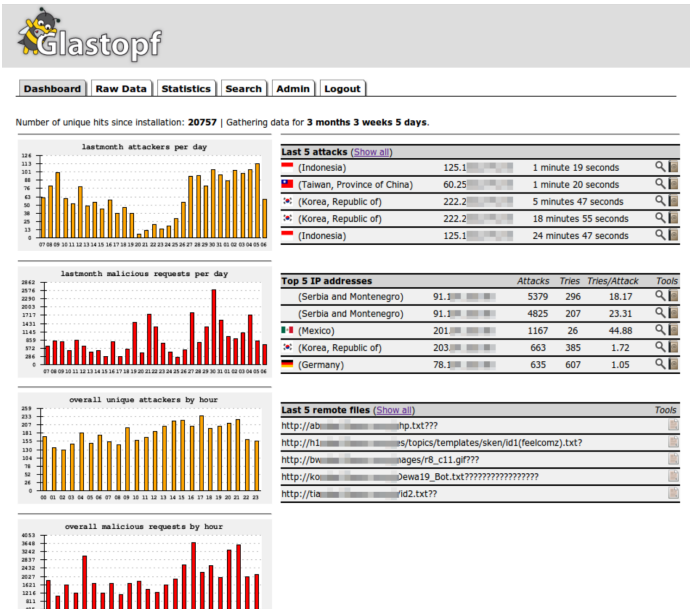


Figure 25: Glastopf Web Interface (Glasif).

8.1.4 HoneyD

Honeyd is is one of the best-known and most seminal low interaction honeypot implementations, in particular concerning the virtualization of hosts in Unix systems. Introduced in 2003, HoneyD is a small daemon that simulates thousands of virtual hosts at the same time on a single system.

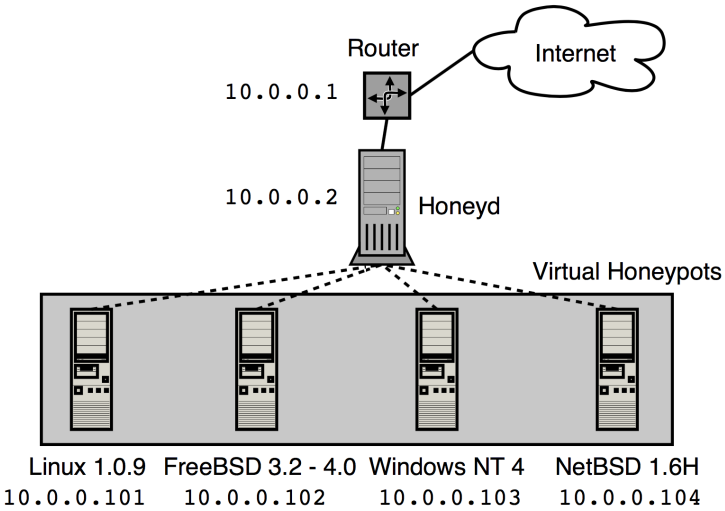


Figure 26: HoneyD overview.

The hosts can be configured to run arbitrary services on the specified open ports, and their fingerprint can be adapted so that they appear to be running certain operating systems (also IoT architectures). Different operating systems are simulated on the TCP/IP stack level by learning TCP fingerprints from reading `nmap/xprobe` fingerprint files, which results in an effective deception of these common tools. Not only hosts are simulated, but also any arbitrary routing topology of the network (including dedicated routes and routers). Moreover, the routes can be annotated with latency and packet loss to improve realistic characteristics of the topology. It is possible to ping or to trace route all virtual machines. As hacker will not find any real computer with real operating system, the key point is to configure the virtualized network stack. Therefore, the attacker believes to have access to a complete realistic network.
Honeyd provides the functionality of a low-interaction honeypot running an arbitrary operating system and network services are simulated by using external programs which mimic the basic behavior of the service. The architecture consists of the following components: a configuration database, a central packet dispatcher, protocol handlers, a personality engine, and an optional routing component Incoming packets are processed by the central packet dispatcher: it first checks the length of an IP packet and verifies the packet's checksum. The framework is aware of the three major Internet protocols: `ICMP`, `TCP` and `UDP`. Packets for other protocols are logged and silently discarded. Before it can process a packet, the dispatcher must query the configuration database to find a honeypot configuration that corresponds to the destination IP address. If no specific configuration exists, a default template is used. Given a configuration, the packet and corresponding configuration is handed to the protocol specific handler. The ICMP protocol handler supports most ICMP requests and by default, all honeypot configurations respond to `echo` requests and process `destination unreachable` messages. For TCP and UDP, the framework can establish connections to arbitrary services which are external applications that receive data on `stdin` and send their output to `stdout` (*service scripts*, *python services* and *subsystems*).

The behavior of the single service depends entirely on the external application. Therefore, it is easy to integrate new programs/applications into the framework. In addition, these handlers can redirect connections to remote systems on which native services run (*subsystems*). All outgoing traffic is processed by the Personality Engine: it allows each honeypot in the framework to impersonate an arbitrary operating system. Thus, it can be used to fool an attacker (e.g., making him that he is attacking a Cisco router). The Honeyd distribution also includes scripts which simulates the behavior of virus back-doors (e.g., `kuang` and `mydoom`) and many network services (e.g., `HTTP`, `FTP`, `POP3`, and `SMTP`). The honeypot allows to simulate up to 65.535 different systems on a single host; however, the performances depend on the complexity of the simulated services which run on each honeypot instance.
There are two other major advantages to use Honeyd: first of all, it can capture the connection on any port; this utility makes detection of the network traffic easier and better. The second advantage is that Honeyd is able to change services and handle several different OSs at the same time.
However, even if Honeyd is maybe the most popular low interaction honeypot, its main problem is the age: the project is open-source but part of it is outdated and nobody in the community seems to upgrade it. Another limitation is that scripts bind to the different ports: with a basic scan it is possible to find which ports are open but as soon as the attacker tries to actually connect on a specific port, he will realize that the service is fake. For example, if a script is used for the connection to a web-server by using the Telnet protocol, the server should reply back specific messages; but in reality, with HoneyD nothing happens. So, the hacker can quickly understand that there is something wrong with the target service and will abort his attack.

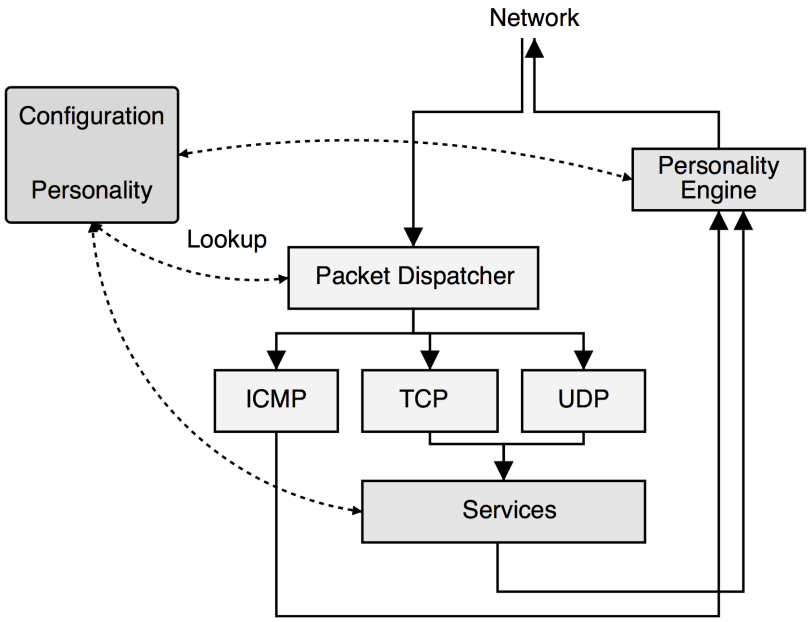


Figure 27: HoneyD Architecture: incoming packets are sent to the correct protocol handler. For TCP/UDP, the configured services receive new data and send responses, if necessary. All outgoing packets are modified by personality engine to mimic the behavior of the configured network stack.

8.1.5 HoneyBOT

HoneyBOT is a low-interaction honeypot for Windows introduced in 2002, which opens a range of roughly 1.000 `UDP` and `TCP` listening sockets emulating vulnerable Windows services. When attackers connect to these services, they are fooled into thinking they are attacking a real server. The honeypot safely captures all communications with the attacker and logs these results for future analysis and if the attacker attempts an exploit or upload a malware to the server, then the honeypot will safely store these files on your computer for analysis and submission to antivirus vendors. The log tool is quite sophisticated, as raw packet level data, keystrokes, and malware (trojans, rootkits) are saved for analysis. Furthermore, a report to a centralized collection point via the `syslog` utility is optional. It is closely tied with a GUI, which offers a classification of log events based on port number or attacker IP address. HoneyBOT emulates many of the main services/protocols, such as: `echo`, `ftp`, `shh`, `telnet`, `smtp`, `http`, `pop3` and more.

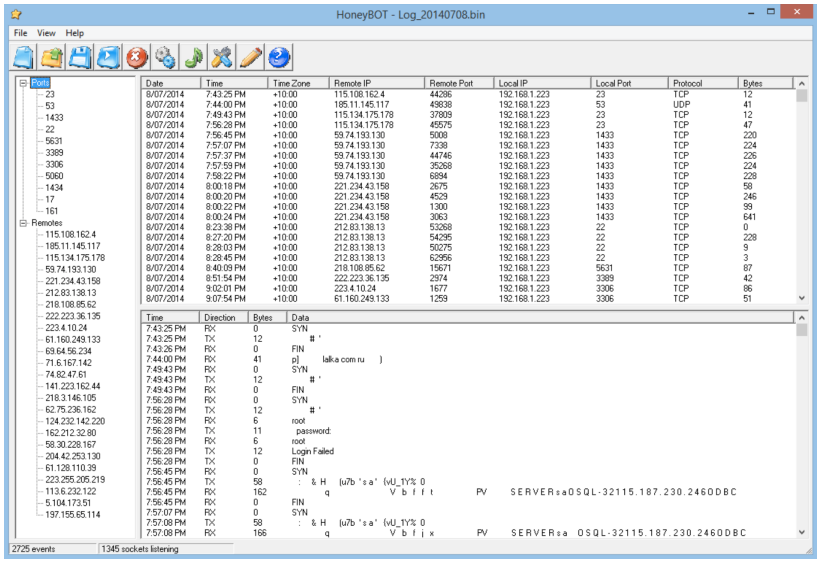


Figure 28: HoneyBOT GUI.

8.1.6 Honeytrap

Honeytrap is a low-interaction honeypot daemon which distinguishes itself from other honeypots by implementing a dynamic server concept. It is a network security tool written to observe attacks in particular against TCP or UDP services.

In contrast to other honeypots, which often focus on malware collection, honeytrap aims for catching the initial exploit collecting and further processing attack traces. It uses a `PCap` based sniffer to check the network stream for incoming packets and starts appropriate listeners on demand. Each listener can handle multiple connections and terminates after some time. This concept targets completely unknown attacks which might occur on random ports and unknown protocols (no predefined emulation is required). The arriving data is assembled to a string (called *attack string*) and written to a database file. Honeytrap can parse an attack string for commands, advising the server to download a file from another host. If a download command is found, the server tries to retrieve the related file automatically. A downloaded file is stored locally with an `md5` checksum in its name.

Service emulation is not the main focus, however some basic emulation is provided (`ftp` and `tftp`). If no emulation for a protocol exists, the default response is a single newline character, which, according to the developer, is a simple but surprisingly successful approach. Honeytrap distinguishes strictly between data capture, which is implemented by the core system, and attack analysis, which is implemented by plugins. It supports several modes, including a mirror mode, which replies with exactly the same packets as received, hence emulating services implicitly. If the connection fails, default replies are used. Furthermore, Honeytrap can be used in proxy mode (also called *meta-honeypot*), which relays incoming connections to other hosts or services while still logging the communication. If ports are configured in the ignore mode, it simply does not handle those ports.

```
$ sudo ./sbin/honeytrap -C etc/honeytrap/honeytrap.conf -D -t 5
honeytrap v1.0.0 - Initializing.
  Loading plugin ftpDownload v0.5.3
  Loading plugin tftpDownload v0.4.1
  Loading plugin httpDownload v0.0.2
  Loading plugin SaveFile v0.2.0
  Loading plugin ClamAV v0.1.0
  Loading plugin SpamSum v0.4.0
  Port 22/tcp is configured to be handled in ignore mode.
  Port 80/tcp is configured to be handled in proxy mode with target 85.214.26.53:80/tcp.
  Port 443/tcp is configured to be handled in proxy mode with target 85.214.26.53:443/tcp.
  Servers will run as user nobody (65535).
  Servers will run as group nogroup (65535).
  Loading default responses.
  Connections will be handled in mirror mode by default.
  Logging to /var/log/honeytrap.log.
  Initialization complete.

honeytrap v1.0.0 Copyright (C) 2005-2007 Tillmann Werner <tillmann.werner@gmx.de>
[2007-10-28 16:37:10]    445/tcp          Handling tcp connection request from 217.232.215.22:3039 to 217.232.59.63:445.
[2007-10-28 16:37:10]    445/tcp          Connection from 217.232.215.22:3693 accepted.
[2007-10-28 16:37:10] <> 445/tcp        * 4348 (of 4348) bytes copied from client connection to 217.232.215.22:445.
[2007-10-28 16:37:11] <> 445/tcp        * 51 (of 51) bytes copied from mirror connection to 217.232.215.22:27918.
[2007-10-28 16:37:11] <> 445/tcp        * 1440 (of 1440) bytes copied from client connection to 217.232.215.22:445.
[2007-10-28 16:37:11] <> 445/tcp        * 832 (of 832) bytes copied from client connection to 217.232.215.22:445.
[2007-10-28 16:37:11] <> 445/tcp        * 133 (of 133) bytes copied from mirror connection to 217.232.215.22:27918.
[2007-10-28 16:37:11] <> 445/tcp        * 127 (of 127) bytes copied from client connection to 217.232.215.22:445.
[2007-10-28 16:37:12] <> 445/tcp        * 39 (of 39) bytes copied from mirror connection to 217.232.215.22:27918.
[2007-10-28 16:37:12] <> 445/tcp          Connection closed by foreign host.
[2007-10-28 16:37:12] <> 445/tcp          Mirror connection closed.
[2007-10-28 16:37:12]  * 445/tcp          7698 bytes attack string from 217.232.215.22:3693.
[2007-10-28 16:37:12] SpamSum - MD5 hash appended to signature file.
[2007-10-28 16:37:12] SpamSum - Calculating spamsum.
[2007-10-28 16:37:12] SpamSum - Spamsum match score is 97.
[2007-10-28 16:37:12] SpamSum - Locality sensitive hash value appended to signature file.
[2007-10-28 16:37:17]    23454/tcp        Handling tcp connection request from 217.232.215.22:3535 to 217.232.59.63:23454.
[2007-10-28 16:37:17]    23454/tcp        Connection from 217.232.215.22:4274 accepted.
[2007-10-28 16:37:17] <> 23454/tcp        Mirror connection rejected, falling back to normal mode.
[2007-10-28 16:37:18]    23454/tcp      * 152 bytes read.
[2007-10-28 16:37:19]    23454/tcp        Connection closed by foreign host.
[2007-10-28 16:37:19]  * 23454/tcp        152 bytes attack string from 217.232.215.22:4274.
[2007-10-28 16:37:19] FTP download - Requesting 'jnjguezoh.exe' from 217.232.215.22:23289..
[2007-10-28 16:37:19] FTP download - Faking Windows FTP dialogue.
[2007-10-28 16:37:19] FTP download - Sending 'USER 1'.
[2007-10-28 16:37:19] FTP download - Sending 'PASS 1'.
[2007-10-28 16:37:19] FTP download - Sending 'TYPE I'.
[2007-10-28 16:37:20] FTP download - Sending 'PORT 217,232,59,63,4,56.
[2007-10-28 16:37:20] FTP download - Sending 'RETR jnjguezoh.exe'.
[2007-10-28 16:37:22] FTP download - Successfully downloaded jnjguezoh.exe.
[2007-10-28 16:37:22] FTP download - Sending 'QUIT'.
[2007-10-28 16:37:22] FTP download - jnjguezoh.exe attached to attack record.
[2007-10-28 16:37:22] SaveFile - Malware sample dumpfile jnjguezoh.exe saved.
[2007-10-28 16:37:22] ClamAV - Scanning sample 1.
[2007-10-28 16:37:22] ClamAV - Sample 1 is infected with 'Exploit.DCOM.Gen'.
[2007-10-28 16:37:22] SpamSum - MD5 hash appended to signature file.
[2007-10-28 16:37:22] SpamSum - Calculating spamsum.
[2007-10-28 16:37:22] SpamSum - Spamsum match score is 51.
[2007-10-28 16:37:22] SpamSum - Locality sensitive hash value appended to signature file.
[2007-10-28 16:37:23] ---- honeytrap stopped ----
```

Figure 29: Example of execution of Honeytrap.

8.1.7 Bluepot

The bluetooth honeypot Bluepot was developed in 2010 to capture attacks on bluetooth devices. It is designed to accept and store any malware sent to it and to interact with common bluetooth attacks such as `Blue Bugging` (remotely taking over control of the target device) and `Blue Snarfing` (remotely reading files from another device).
This honeypot does not require any specific device, offers the support for multiple Bluetooth devices and for multiple vulnerabilities, has the ability to emulate fake devices, it is written in `Java` (and distributed as a `.jar` file) and runs on any Linux-based machine, but it does obviously demand from the user to possess at least one active bluetooth interface.

The main support for malware collection is support for the `OBEX` (`Object Exchange`) push protocol, enabling a computer to automatically download all files sent to it whilst under the appearance of a smart phone or printer, `RFCOMM` (`Radio Frequency Communication`) protocol used for serial communication and the `L2CAP` (`Logical Link Control and Adaptation Protocol`) used for transmission control.. This honeypot has also some graphical dashboard which allows monitoring of attacks and presents some simple graphs and lists.

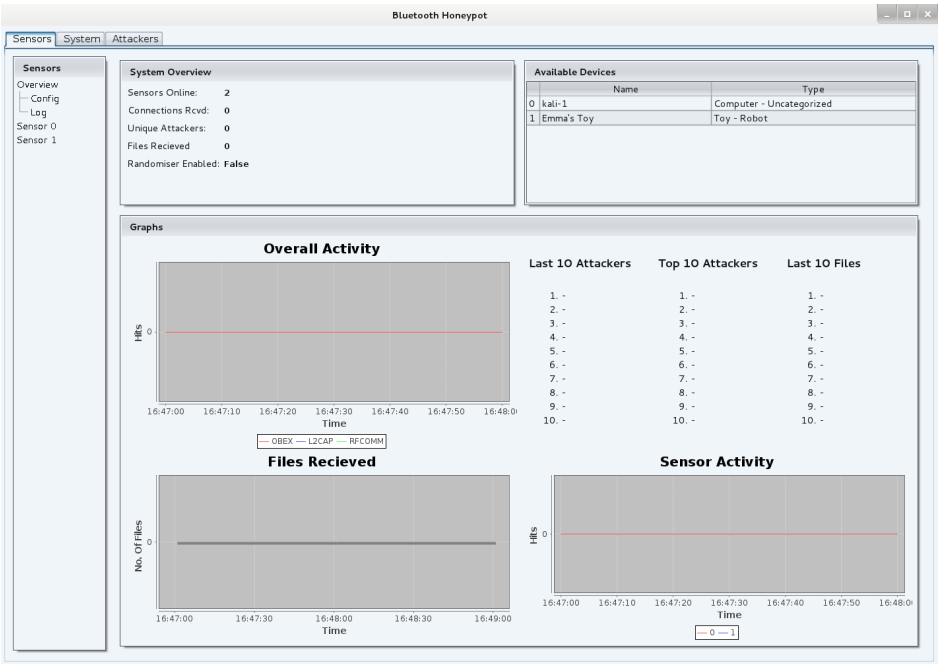


Figure 30: Bluepot Dashboard.

8.1.8 Shockpot

The honeypot Shockpot is designed around a single, critical vulnerability called `Shellshock` (CVE-2014-6271) for the bash shell. ShellShock is a virus which can exploit a vulnerability in one of the operating computer's ports and then enters unauthorized commands. This program, which is used to enter commands was discovered in September 2014 to create a port through which hackers can attack. Namely, the computer is made to spread the virus to other networks by creating viral botnets. These commands can be deployed individually by the hacker, making ShellShock more akin to a back door than a wild-running virus. Having more control over the proliferation allows attackers to be stealthier than they were with earlier Trojans. Once built, botnets can be commanded to drive huge amount of traffic to a specific network, making a DDoS attack. The vulnerability was very far-reaching and severe, since it is one of the most commonly deployed command line software shells and puts millions of systems at risk to local and remote code execution. So, the study of its exploitation became interesting. The pure Python implementation emulates an Apache-server, that appears to processes trailing strings after function definitions in the values of environment variables. This allowed remote attackers to execute arbitrary code via a crafted environment in the original vulnerability. The honeypot logs all HTTP requests to port 80 by default and detect ShellShock attacks while logging the exploit code and scripts. Then, it's easy to extract the URL's and files from the ShockPot to submit to online malware repositories (e.g., VirusTotal).

8.1.9 HoneyC

Client honeypots are necessary because they are able to detect client side attacks, an attack vector that typically server honeypots are unable to detect. Client side attacks are assaults of clients that originate from malicious servers. Client honeypots crawl the Internet to find and identify servers that exploit client side vulnerabilities. Traditionally, these servers are identified by the client honeypot monitoring state changes that result from a server interaction; these high-interaction client honeypots are slow and expensive to operate because they require a real browser within a real OS to be executed and to and monitor the state of the entire system. Two examples are:

- **Honeyclient:** it detects successful attacks by monitoring changes to a list of files, directories and system configuration after the Honeyclient honeypot has interacted with a server.
- **Honeymonkey:** it also detects intrusions by monitoring changes to a list of executable files and registry entries, but Honeymonkey goes a step further by adding monitoring of the child processes to its repertoire to detect client side attacks.

HoneyC was developed to introduce a new low-interaction client honeypot that emulates only the essential features of client honeypots, applying signature matching to allow fast static analysis of server responses. A client honeypot has to fulfill some fundamental tasks: first of all the client honeypot needs to interact with the server; this usually requires making a request to the server, consuming and processing the response. So, a portion of the client honeypot needs to create a queue of server requests to execute. Then, another portion could employ several algorithms to create such a queue of server requests (e.g., crawling or search engine integration) and in the end, the client honeypot needs to analyze the server response searching for illegal messages.
HoneyC implements all these requirements in three components that map to the functional requirements outlined above: *Queuer*, *Visitor*, and *Analysis Engine*. Each of these components supports additional modules to suit specific needs.

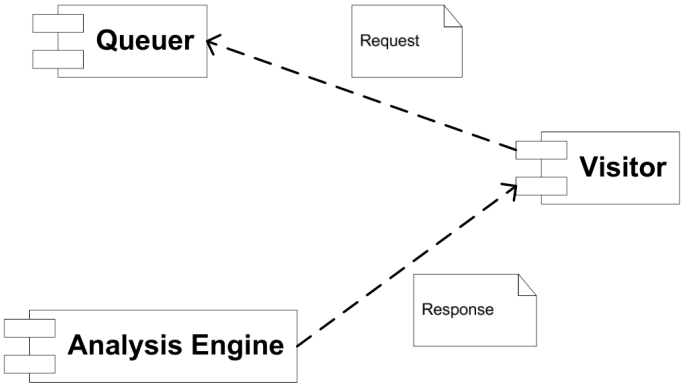


Figure 31: HoneyC Overview.

In practice, the HTTP requests that are generated by the Queuer component are consumed by the Visitor component. The corresponding HTTP responses that are generated by the Visitor are analyzed by the Analysis Engine valuating whether security policies have been violated. The HoneyC framework automatically links each component to ensure appropriate flow of information. The analysis made by the Analysis Engine uses predefined (or customized) Snort IDS Rules so that the client honeypot is able to analyze the response, searching for matches with a set of Snort signatures. HoneyC is expandable in a variety of ways: it can use different visitor clients, search schemes, and analysis algorithms.

8.1.10 Monkey-Spider

Monkey-Spider is a crawler based low-interaction client honeypot utilizing anti-virus solutions to detect malware. Its main idea is to first crawl the contents of a website and then analyze the crawled content for maliciousness to expose threats to Web clients. It is claimed to be fast and expandable with respect to other detection mechanisms. It uses many existing freely available software systems, the well-known scalable `Heritrix` crawler to create `arc` files and then pass them along to anti-virus scanners such as `ClamAV` and `Avast`. Monkey-Spider detection is done only by external signatures. URLs, binaries and malicious JavaScript are extracted and saved in a threats database. The architecture is divided in functional blocks that can either be located on different computers for scalability and performance reasons or used on a single one for the ease of use.

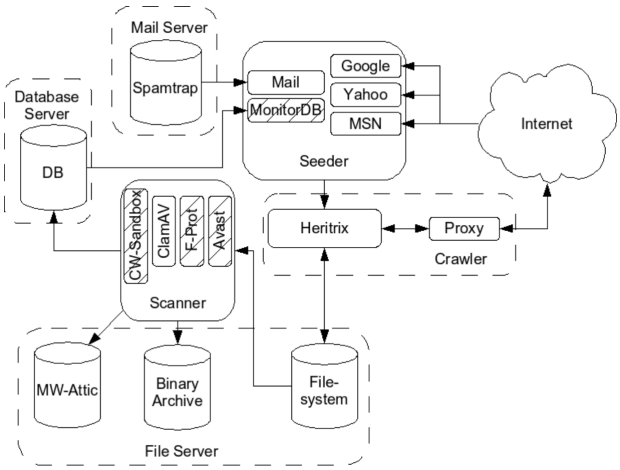


Figure 32: Monkey-Spider Overview.

Queue/Seed Generation: Every run begins with the Seeder block which generates starting URLs for the crawler (the queue). One can either generate seeds with the Web search seeders and/or generate seeds out of spam mails with the mail seeder. Another possibility is to use the MonitorDB seeder to re-queue previously detected malicious sites. The Web search seeders use the search engines Yahoo, Google and MSN Search for searching and URL extraction. The mail seeder extracts URLs out of collected spam mails from a spamtrap. In addition, the MonitorDB seeder is used to constantly re-seed previously found malicious content over time from our malware DB.

Web Crawling: Heritrix Web crawler project is used from the Internet Archive, to crawl the URLs extracted by the Seeder and download content from the World Wide Web. Heritrix queues the generated URLs from the seeder and stores the crawled contents on the file server while generating detailed log files. Optionally we can use Heritrix through an interconnected Web proxy: the Web proxy can be used to increase performance and avoid duplicate crawling.

Malware Analysis: the actual downloaded content is stored by the crawler in the so called `arc` file format. The information from these files is extracted and then the content is analyzed with different anti-virus solutions and malware analysis tools. Identified malware and malicious websites are stored in the Malware Attic Directory. In addition, information regarding the malicious content is stored in a database and every binary and JavaScript file is stored into an additional archive directory.

8.1.11 YALIH

The false negative detection rate of client honeypots increases significantly with obfuscated attacks that exploit obfuscation techniques that scripting languages provide (e.g., JavaScript simple substitution, string splitting). Obfuscation allows attackers to bypass detection and deliver exploits for which a signature already exists. So, Signature-based detection engines that do not de-obfuscate the script, they will suffer from false negatives regardless of the number of attack signature is used. Clint honeypots mostly rely on a single analysis engine for attack detection: for example, Monkey-Spider uses the ClamAV antivirus signature engine while HoneyC makes use of Snort intrusion detection system signatures. The reliability on a single analysis engine makes them susceptible to an unknown number of false negatives that refers directly to the quality and freshness of the attack signatures. YALIH (Yet Another Low Interaction Honeyclient, 2014) is a client honeypot designed to detect malicious websites by integrating a combination of multiple antivirus engines and pattern matching using string or regular expressions for detection. It aims to provide both a high rate of detection and a very low scanning time using multiple analysis engine and low resources. It deliberately does not address the problem of zero-day attacks. In general, client honeypots are formed by three main components that manage the entire process of website retrieval and analysis: the URL Collector (Queuer); the Visitor; and the Analysis Engine.

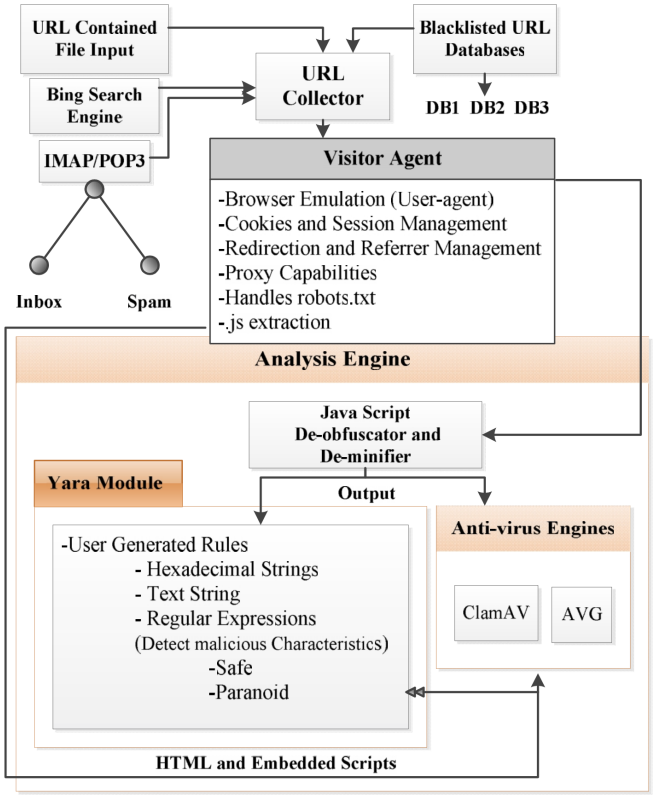


Figure 33: Overall design of YALIH.

The URL Collector module is responsible for gathering the hyperlinks of potential malicious websites to be visited by the honeyclient. Depending upon the implementation, these hyperlinks can be gathered from single or multiple sources. For example, HoneyC and Monkey-Spider gather hyperlinks from search engines while Monkey-Spider also employs crawlers. A problem with using web crawlers is that it increases the chances that a client honeypot may be detected by a malicious

web server since they initiate a high number of connections and fetch many URLs in a very short period of time. To this purpose, YALIH uses the `Mechanize` visitor module that provides built-in functions to crawl and extract links from the visited websites to a depth defined by the user without using crawling functionalities. The Mechanize visitor agent allows a user to set different features in the form of browser that may represent various browser characteristics (such as type, version, underlying OS, language and extensions). Mechanize module in YALIH honeypot is also configured to provide various capabilities to minimize detection of virtual browsers through cookies, redirection and session handling. URL collection and queuing can be achieved in different ways:

1. Direct Feed: users may input an URL directly, or use a file containing a list of URLs to visit.
2. Search Engine API Integration: YALIH incorporates the Bing search engine for URL collection.
3. Spam & Phishing: YALIH has a component that integrates URL harvesting from emails (IMAP).
4. Malicious Website Database: blacklisted malicious URLs are collected by a module from databases.

The analysis engine on YALIH also relies on several methods:

a) De-Obfuscation & De-Mining of JavaScript: YALIH parses, extracts and downloads the `.js` embedded within the visiting websites. The downloaded JS file is rendered, de-mining and de-obfuscating the JS scripts encoded. Once the output is generated, it is written to a file that is then scanned by multiple antivirus and signature/pattern matching tools.

b) Multiple Antivirus Signatures:YALIH utilizes a selection of available detection engines at its core. These engines comprise of an open source (i.e. ClamAV) and a commercial antivirus attack signature database (i.e. AVG). Other antivirus engines available for Linux operating system were also tested but then removed to reduce the scanning time, as they did not provide significant detection improvement, considering the delay they introduced.

c) Malicious Pattern Detection: use of Regular Expressions allows the system to detect any variations of some characteristics which are typical of legitimate and benign websites.

The developers compared the client honeypot's effectiveness using a malware URL database and determined 80% less false negatives than HoneyC and 35% than Monkey-Spider while still requiring only a moderate scanning time.

8.1.12 MTPot

In 2016 Cymmetria Research released a new open source low intearction honeypot specifically designed for the detection of the `Mirai` botnet. It's a simple and very specific tool built to match what the Mirai malware expects, directly based on its source code.
The decision to develop the honeypot followed the huge DDoS attack ("DynDOS") that occurred in October and performed using IoT devices infected by the MIrai botnet. The attack has overwhelmed Internet service providers and caused multiple disruptions, making DDoS one of the key concerns of security as well as businesses world-wide. So there was the need of a very lightweight honeypot with which it's possible to collect verified mirai Indicators of Compromise (IoCs), in particular IP addresses trying to compromise IoT systems and the malware samples which are downloaded during the infection. The target of MTPot is to detect and study the infection attempt that Mirai makes in order to build the botnet that will be used for the Distributed Denial of Service attack.
To capture the infection payload the MTPot honeypot includes abilities such as:

- ◇ Detect incoming connections on any port using telnet (equivalent to listening on that port).

- ◇ Specifically ID the mirai version (open source one), based on the commands requested from the service.
- ◇ Alter parameters to ID mirai (port and commands).
- ◇ Report to a syslog server to collect the fingerprinted IPs.
- ◇ Collect the malware samples mirai tried to infect the user with.

The usage of the honeypot is quite simple, but much like any other low interaction honeypot, it can be fingerprinted if a talented hacker has enough time to deal with it. In the actual version, MTPot has some problems with the collection of malware samples; so, actually it is not programmed to download and execute the obtained malware.

8.1.13 Telnet IoT Honeypot

Telnet-IoT-Honeypot is a new low interaction honeypot introduced in 2016. It has been developed to catch botnet binaries which are currently attacking IoT devices after the Mirai source code release. The project implements a Python telnet server trying to act as an honeypot for IoT Malware which simulate the very insecure default passwords of telnet services of IoT devices on the internet. It's fast and easy to use since it does not provide a full (via `chroot`) or simulated behavior of an Embedded Linux system; the honeypot instead is designed with the goal of just collect statistics about IoT botnets and their patterns. So, the honeypot requires that it must work with every form of automated telnet session, which may try to infect the system with malware. Luckily, these IoT malwares infection processes are quite simple (just using `wget` to download something and then running it) and the honeypot succeeds in detecting and collecting the vast majority of the samples. Telnet-IoT-Honeypot has a client/server architecture, with a client (the actual honeypot) accepting telnet connections and a server aggregating connection data and sample analysis. The analysis of the collected sample is delegated to the Virus Total online malware repository, while the analyzed data is store into a SQLite database. For local deployments, the application can also be run in local mode to eliminate the need to run a client and server locally.

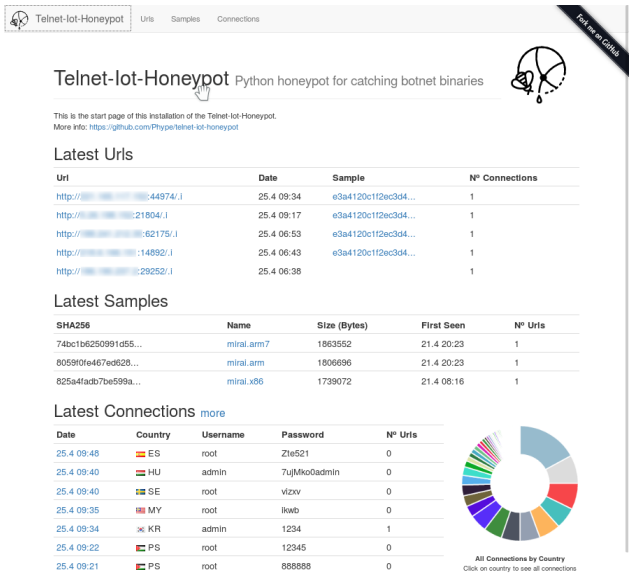


Figure 34: Telnet-IoT-Pot Dashboard.

8.2 Medium Interaction Honeypots

8.2.1 Kippo

Unauthorized access to servers occurs on a daily basis. Attackers are constantly searching the Internet for servers that they can use for malicious activity. An easy target for an attacker is a server on which an administrator has set up a service which allows users to control the server remotely but has a weak password set. A common way to provide remote shell access on an operating system is using the `Secure Shell` (`SSH`) network protocol. By using SSH, an user can perform a secure remote login over an insecure network to access a remote shell. In order to login to the remote shell, the user has to successfully authenticate itself to the SSH server. An attacker can authenticate itself by getting the password from a dictionary attack or brute force attack.
In order to study the activities performed by attackers after they compromised a system with an SSH server, the Kippo honeypot was introduced in 2009. Kippo is is a medium interaction honeypot designed mainly to be used to log the SSH brute force attacks and, most importantly, the entire shell interaction performed by the attacker against a target system. It provides a fake file system, simulating a Debian Linux server with the ability to add and remove files. The honeypot allows the attacking entity (hacker or botnet) to attempt a login to the system, believing it is entering into a legitimate SSH session with the real server. Upon successful guessing of the password (deliberately trivial), the attacker is then moved into a fake system in which the attacker can navigate around, interact with (but can't do any real damage) and use some basic tools (such as `cat`, `wget`, etc.) that are integrated to fool the attacker. In this fake system, all interactions with the shell are monitored and recorded with session logs stored in an UML compatible format, so that they can be replayed with the original timings of the prompt (`.tty` logs). Moreover, the system also allows the use of `wget` and other commands commonly used to fetch or download files and manipulate it on the infected host. It also saves the downloaded samples obtained by `wget` for later investigation.
The main objective of Kippo is to bring to the attacker the impression of navigating a real compromised system. By analyzing the hacker's interaction with the shell of the device, it's possible to understand for what purpose the system was hacked, which data the attacker was trying to get from it and what techniques were used. Undoubtedly, it can be also a source of learning how the administrator in the next future should strengthen the system security.
The main interesting features that Kippo provides are:

- ◇ Fake filesystem resembling a Debian 5.0 installation with the ability to add/remove files.
- ◇ Possibility of adding fake file contents so the attacker can `cat` files (such as `/etc/passwd`).
- ◇ Possibility to add fake command output (e.g., `/sbin/ifconfig`, `vi`, `apt-get`, etc.).
- ◇ Session logs stored in an UML compatible format for easy replay with original timings.
- ◇ Stores all the downloaded files (using `wget` and `curl`) during the SSH session.
- ◇ Stores logs about attacker's behavior (commands invoked) in a format allowing to replay it.
- ◇ Simulates finishing of the SSH session, while the honeypot does not really stop the connection.

The filesystem cannot be damaged since Kippo stores it in a `fs.pickle` file: pickle is the standard mechanism for object serialization; it uses a stack-based virtual machine that records the instructions used to reconstruct the object.

A pickle file is a partial memory dump that allows the developer to easily write objects to the hard disk and once written, the file can be reload again into main memory at a later point. So, every time a new user connects to the honeypot, Kippo loads the file system into memory. The attacker can do whatever they want to the file system, but when the next person logs in, Kippo loads the original file system pickle file; that way every new connection starts with a fresh environment.
Once inside the system, an attacker can try try add a new user of the OS (typical practice) by using the `useradd` command; however, Kippo's `useradd` command provides a more interesting feature: it tries to collect all the information about the attacker.

To monitor the honeypot activity, an interesting script has been developed, called `Kippo-graph`. It can be used to visualize fundamental statistics from the Kippo honeypot such as: Top 10 usernames/passwords attempted, Top 10 username-password combinations, Success ratio, Number of connections per unique IP, Top 10 input, wget commands, etc.

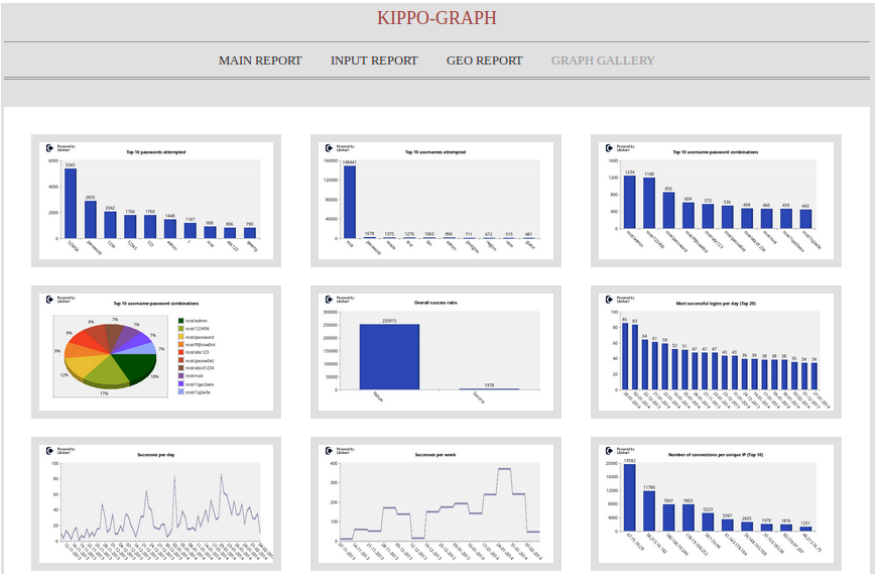


Figure 35: Kippograph Dashboard.

One of the advantages of using Kippo is that it gives a little time response to an intrusion attempt and reconnaissance with whom we are dealing. A skilled person probably detects the honeypot, but this would deter script kiddies and inexperienced hackers. Maybe the main advantage is given by the added value represented by the analysis of the Kippo's logs: they link the sample obtained by `wget` command to the SSH session of the attacker; in addition, logs stores information about an attacker's behavior in the operating system in a format allowing replaying it in a screen-cast form.

The major disadvantage of Kippo is that it can be easily detected by fingerprinting the honeypot. This is the main weakness of the system as the attacker would not connect to the honeypot if they noticed this. Attackers can figure out that they are connected to a Kippo by using this simple trick: they can use a command not implemented in the honeypot filesystem (and neither defined by the analyst, because not so common) and see which is the response; if the response is different from the one expected on a real OS, they deduce that they in a fake device.

Example: the `file` command is not implemented in Kippo (neither by users); so, if it's prompted:

```
# file /sbin/init
bash:  file:  command not found
```

While on a real system, by typing the `file` command, I would get:

```
# file /sbin/init
/sbin/init:  ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
    dynamically linked (uses shared libs), for GNU/Linux 2.6.24,
         BuildID[sha1]=0x7aa29ded613e503fb09fb75d94026f3256f01e7a, stripped
```

So, more advanced attackers easily know that they are in a Kippo honeypot and even if they exploit the login on the system, then they exit immediately.

8.2.2 Cowrie

The developer of Cowrie was working on extensions and contributions to the Kippo honeypot. In 2014, when Kippo project were closed, he forked the repo and continued the development of the honeypot. To distinguish this new project from the original one, it was renamed into Cowrie.

Cowrie, being the successor of Kippo, is itself a low interaction honeypot that emulates the `SSH` service, with an additional introduction of the emulation for the `Telnet` protocol.
The emulation involves both `SSH/Telnet` session establishments and, in the case of user's successful authorization, also shell interaction with the attacker. The honeypot can be used to register brute force attacks aimed at obtaining users `SSH/Telnet` service passwords.
Moreover, it allows for analyzing further actions of the attacker who manages to obtain the password and thus has access to the emulated system. This is possible because Cowrie inherited the property of Kippo to register all shell commands inputted by the user and to save in specific directory all malicious files downloaded by means of emulated `wget` and `curl` commands. Cowrie has also introduced the support for `SCP` and `SFTP` protocols for the file upload; so, also malicious samples uploaded using these services are placed in the same download folder.
As Kippo, Cowrie emulates services so a degree that may give an attacker the impression of using real protocols. Thus, data gathered by the honeypot may provide many valuable information about the purpose and the way of performing Telnet/SSH dictionary attacks. These data may be useful for many different IT security systems.
Likewise the Cowrie honeypot is forked from the Kippo project, Cowrie is considered a better honeypot than Kippo since it has fixed some common issues within the latter one.
An example are some issues found in Kippo which are related on how it handles established sessions made to the honeypot: when a client exits the honeypot, he's left in a trapped shell; the reason is because Kippo doesn't send the client an exit status. So, the client is effectively trapped in a shell that is unusable. Instead, Cowrie has solved this problem by sending to the client an `exit code 0`, which means a successful disconnect.
Another problem related to Kippo is that some SSH attacks were not logged in complete detail; also this issue is solved by Cowrie developers by introducing additional logging capabilities (e.g., `.log`, `.tty` and `.json` logs).

The main additional functionalities introduced by Cowrie over Kippo are:

- `SFTP` and `SCP` support for file upload: uploaded files are placed into the download directory (like files that were downloaded by `wget`).
- Support for SSH exec commands (run remote commands; e.g., `ssh user@'host cat /etc/passwd'`).
- Support for TCP/IP tunneling.
- Logging of direct-tcp connection attempts (ssh proxying).
- Forward SMTP connections to SMTP Honeypot (e.g. mailoney)
- Logging in JSON format for easy processing in log management solutions
- SSH fingerprint logging: some adversaries have public keys configured on attack hosts (**!!!**). The SSH client attempts to login to the honeypot with this public key. This feature allows Cowrie to accept, log and then deny public-key authentication.
- Better support for common linux commands (e.g., `ls` lists all entries in alphabetical order, `wget` can download data using ports other than 80, etc.).
- Additional commands: `sleep`, `sync`, `uname -r`, `dir`, `halt`, `reboot`, `users`, `killall`, etc..
- ELK stack integration.

An interesting feature that Cowrie inherited from Kippo are `.tty` logs: each session initiated by an attacker is logged to a `.tty` log file. These log files are stored in a specific directory and can be executed (with the `./opt/cowire/bin/playlog` command) to "watch" a reproduction of the interaction of the attacker with the honeypot shell in real time.
As said before, Cowrie has introduced better log methodologies; in particular `JSON` logging: Cowrie has a JSON dblog plugin that logs the same information as the general `.log` file, but in `JSON` format, so it can be easily picked up by log analysis tools. Indeed, since `JSON` logging is available, the developers introduced in Cowrie the support for the ELK stack (`Elasticsearch`, `Logstash` and `Kibana`), to display honeypot data in a `Kibana` dashboard.

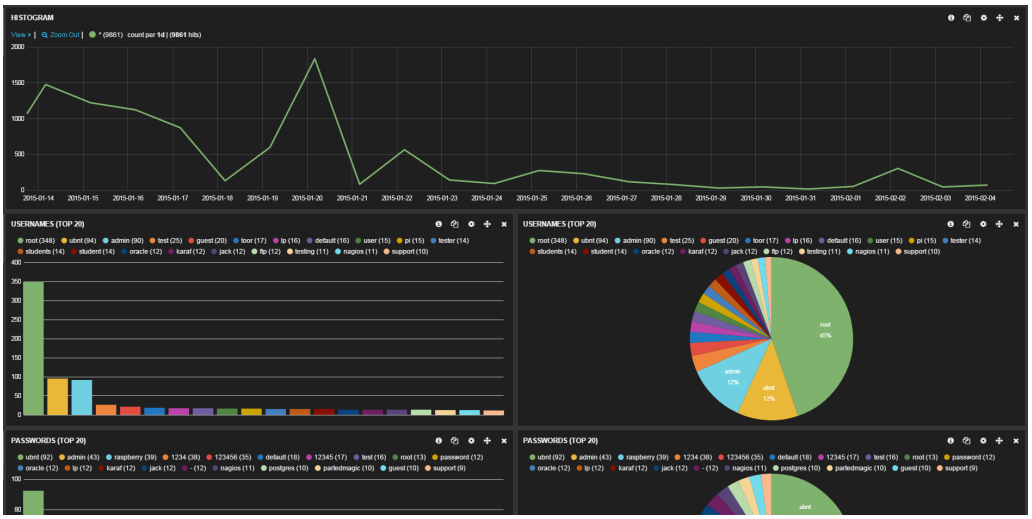


Figure 36: Cowrie data loaded into ELK stack and visualized by Kibana Dashboard.

8.2.3 IoTPOT

In 2014, a team of researchers from Japan introduced IoTPOT, a novel honeypot to emulate Telnet services of various IoT devices running on different CPU architectures (such as `ARM`, `MIPS`, and `PPC`) to analyze the increasing threats constituted by ongoing attacks against various IoT devices.
IoTPOT is a innovative medium interaction honeypot (IoT honeypot and sandbox), which attracts and analyzes Telnet-based attacks consisting of a front-end low-interaction responder cooperating with backend high-interaction virtual environments (called IoTBOX). IoTBOX operates various emulated systems commonly used by embedded systems for different CPU architectures. Moreover, it is a malware analysis environment for IoT devices.
IoTPOT supports all the features inherited by the Telnet protocol interaction: protocol options that the attacking clients could choose to use, realistic welcome message and login prompt usually displayed by the service, and the possibility to login (since it's interesting to study the characteristics of the authentication interactions (e.g., sequences of usernames/passwords). Furthermore, independent from the Telnet protocol, the honeypot supports multiple CPU architectures to capture malware across devices.
During authentication, IoTPOT supports various tactics: it can be configured to reject any authentication credentials to observe login attempts, to allow immediate authentication regardless of the login, to accept only certain credentials, or reject the first attempts and eventually accept a login. During the command interaction with the attacker, IoTPOT chooses the correct emulated system on which redirect the traffic among a set of environments simulating the fact that each IoT device runs on different CPU architecture. So, there is a set of embedded linux OS on different CPU architectures to handle the interactions of attackers with various devices.
The architecture of IoTPOT can be represented as formed by blocks: the heart of the system is the Frontend Responder that handles incoming TCP connection requests, banner interactions, authentication, and command interactions with a set of device profiles, acting as different IoT systems.

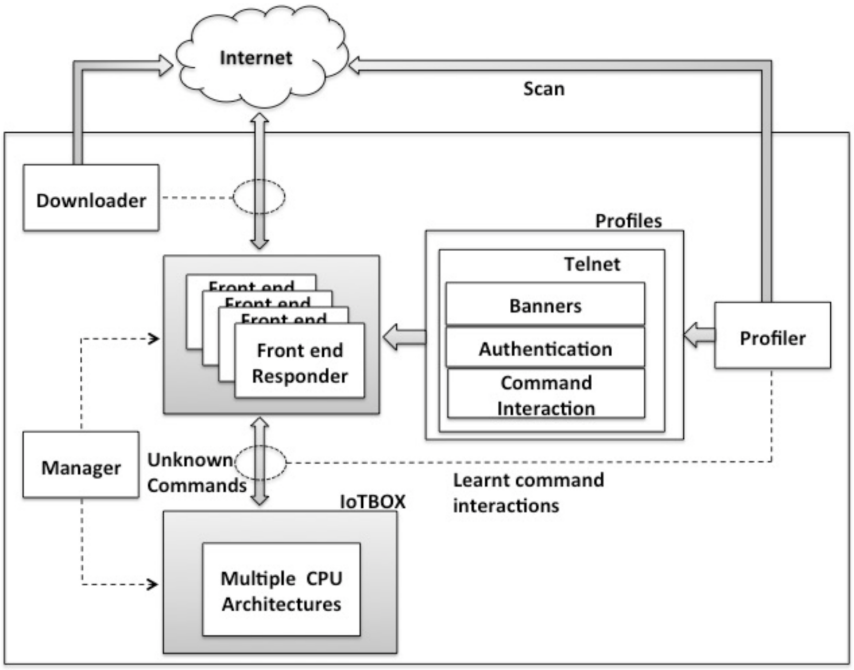


Figure 37: Overview of IoTPOT architecture.

A device profile consists of a banner, an authentication and a command interaction profiles.

- ◇ **Banner Profiles:** they determine the responses of the honeypot for banner interactions (i.e., Telnet options, welcome messages and login prompts).
- ◇ **Authentication Profiles:** they define how to reply to incoming authentication challenges.
- ◇ **Command Interaction Profiles:** they determines the responses to incoming commands, consisting of a set of commands and their corresponding responses.

When an incoming command is not known yet, the Frontend Responder establishes a Telnet connection with a backend IoTBOX and forwards the command to it. IoTBOX is a set of sandbox environments that run Linux OS for embedded devices supporting the malware analysis on 8 different CPU architectures: `MIPS`, `MIPSEL`, `PPC`, `SPARC`, `ARM`, `MIPS64`, `SH4` and `x86`. To run malware binaries of different CPU architectures, the respective platforms (OS) are run on an emulated CPU using `QEMU`, an open source processor emulator. Then, the respective `OpenWRT` platforms are used to run on the emulated CPU environment. `OpenWRT` is a highly extensible GNU/Linux distribution for embedded devices (typically, wireless routers). Then, the Frontend Responder forwards the response from IoTBOX to the client. The OS image occasionally requires a reset since the incoming commands forwarded to IoTBOX may cause malware infections or system alteration. Another important part of the system is the Profiler: it parses the interaction between Frontend Responder and IoTBOX, extracts the incoming command and corresponding response, and updates the command interaction profile so that Frontend Responder can further handle the same command without interacting with IoTBOX. The Profiler also executes the collection of banners from devices in the Internet. It operates in two banner grabbing modes: in Active scan mode, the Profiler scans different networks to collect banners from various devices; instead in Visitor scan mode, the Profiler only connects back to hosts who visit honeypot. The Downloader component is the block of IoTPOT architecture which examines the interactions for download triggers of remote files (e.g., malware binaries). In particular, the files downloaded are obtained from all URLs observed via commands such as `wget`, `ftp`, and `tftp`. The last two blocks are, the Manager which handles configuration of IoTPOT such that it links IP addresses to specific Device Profiles and the Access Controller which controls all network related operations. The Analysis Report block outputs the results of `.pcap` analysis showing total number of packets, start time and end time of packet captures, data byte rate, average packet size and total number of victim IP addresses for each attack.

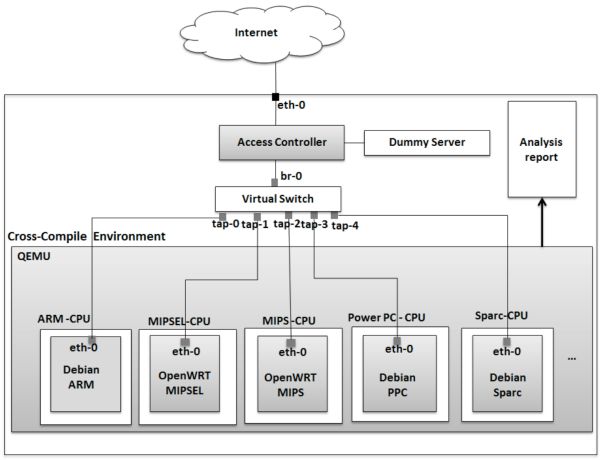


Figure 38: Overview of IoTBOX.

8.3 High Interaction Honeypots

8.3.1 HoneyPHARM

HoneyPHARM is an high interaction honeypot introduced in the last months which is not directly designed for the world of IoT; however it presents an innovative idea which could be a starting point for new IoT honeypot projects.
The idea is quite simple: honeypots are designed to be infected, so they wok collecting malware. Almost any software captured by honeypots is malicious by definition, since honeypots do not run any real services. Thus, *the more honeypots there are in different networks and infrastructures, the higher the chances are to capture new malware samples.*
HoneyPHARM, which is born as an extension Nepenthes Pharm, gives the ability to collect malware from the distributed Nepenthes nodes in a central repository. Findings can be analyzed and actions can be taken accordingly in order to support the proactive early warning notification of threats. The HoneyPHARM is a client/server system: Nepenthes honeypot is installed on each client node in order to collect the malware and related data. It will then send them to the server, which is the central repository. In addition to the malware itself, the client sends details about where the malware has propagated from and the time when the malware is captured. The client also conducts frequent health checks in order to ensure that Nepenthes is running and the connection with the server is always established.
The server, on the other hand, receives these details from different clients, and feeds them to the associated tables in a central database. Through a web portal, the information gathered is represented in multiple forms. A statistical view of information (e.g., most infecting countries, most used ports, etc.), illustrates to the user an executive summary of the trend of malicious activities. In addition, detailed information about these activities is also available to assist in further investigation and analysis; these details consist of the IP addresses and their geographical location as well as the name of the captured malware. This information are normalized and stored in a log file. In practice, HoneyPHARM is a client/server system that manages, reports, and analyzes all distributed honeypot instances to one central repository. The system architecture satisfies four needs: *collection*, *submission*, *analysis* and *presentation*.

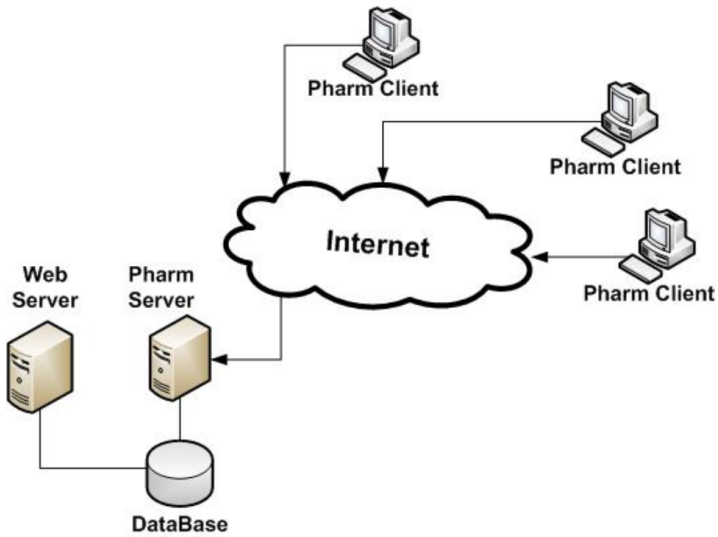


Figure 39: Overview of HoneyPHARM.

The Pharm Client, which is installed on the Nepenthes Nodes, emulates vulnerabilities that attract the attacker; it also collects the malware and related details (hashing it to a unique identifier); then sends them to the Pharm Server through a TCP/IP connection. The setup on the server side is designed to accept connections from specific IP addresses to a specific destination port. This is in order to increase security level of the system.
The Pharm client consists of a mixture of application and scripts to effectively ensure carrying out its role of collecting and submitting the malware and the logs to the server without interruptions. This is done through continuous monitoring and status checking of the client components and its communication with the server. The server receives malwares/logs and it stores them in a DB.

8.3.2 SIPHON (Scalable high-Interaction Physical HONeypot.)

The last honeypot that I want to introduce in my project is maybe the more interesting one. It's a sort of improvement of the concepts that were introduced by IoTPOT (and partially by HoneyPHARM): IoTPOT presents a medium interaction honeypot for IoT devices (IP cameras and Digital Video Recorders) by emulating Telnet services. However, no real devices were used to deploy the honeypot and the goal was to capture telnet based attacks and analyze them with respect to the concerned IoT devices. This implementation is limited in the sense that it typically captures only automated attacks (exploited by botnets) since this kind of honeypots are easily detectable by expert hackers. So, there is the need to find a way how to develop high-interaction IoT honeypots that will attract also skilled attackers.
SIPHON is a prototype of scalable and reliable high interaction honeypot which goal is to learn about existing and novel attack vectors (both human and automatic) targeting IoT devices. There is a first problem to face: IoT devices offer a rich set of potential attack vectors to the attacker and typically honeypots use virtualization or emulation approaches to replicate the device under attack. But how we've just seen before, this approach is really working only with automated attack tools (botnets). Only high interaction honeypots will convince skilled attackers to use advanced attack methods while exploiting zero-day vulnerabilities. This could be implemented by offering advance services to attackers (e.g., allowing an user to move a camera) or allowing the hacker to directly interact with a real existing IoT device. Obviously, this could be very challenging.
A framework for deploying IoT honeypots should:

1) be able to provide high-interaction in order to motivate skilled attackers to perform their activity and expose the vulnerabilities they are exploiting;
2) make it possible to easily place the IoT devices in a wide range of geographic locations;
3) easily scale to devices of multiple vendors and of different kinds in order to keep track of a dynamic threat landscape.

All these features are included in SIPHON allowing the (prototype) honeypot to deploy over 80 high interactive devices with a diverse set of IPs located in different regions of the world. Its architecture is scalable and distributed: SIPHON uses only 7 IoT devices (5 IP cameras, 1 NVR and 1 IP printer) physical IoT devices that are connected (and replicated) to various *warmholes* (geographical locations) on a large scale distribution with 85 instances in 16 cities (9 countries). The network traffic is forwarded on an infrastructure of cloud service providers consisting of Amazon EC2 instances, DigitalOcean and Linode.

<u>Observation</u>: Low-interaction honeypots emulate well-known vulnerable network services by partially implementing the TCP/IP stack, in order to attract attackers. They do not give an attacker access to a real system, but rather to some sort of emulated system. Since usually they do not run a fully featured operating system, they are considered to be safer in terms of remote exploitation. However, due to their simplicity, the main disadvantage of low-interaction honeypots is that they can be detected as artificial by a human attacker. Thus, they are by construction **not optimal** to capture *zero-day vulnerabilities*, since such vulnerabilities are unknown at the time of the deployment of the honeypot and thus cannot be simulated.
High interaction honeypots help to observe attacks in a more realistic setting: they do not emulate services, cut they offer real functionalities with a real base operating systems not restricting the attacker's behavior. So, they allow to learn more about the tools, tactics, and motives of the attacker. As a result, high interaction honeypots are better suited to monitoring complex interactions between the attackers and the system.
On the other hand, there are two main disadvantages by using high interaction honeypots: first, they are usually difficult and costly to deploy and maintain; moreover, they expose the system to a higher security risk, since attackers can stop normal operation of the device after a successful privilege escalation and may use the device at their will.

In the context of wireless communications, the term *wormhole* refers to malicious forwarding of communication by the attacker, often through an out-of-band channel. In particular, wormholes can be used to influence routing schemes in dynamic sensor networks and they are typically very challenging or impossible to detect.
In the SIPHON project, the term *wormhole* is intended in a different way: instead of offensive wormholes used by an attacker, wormholes are defensively used by the honeypot operator. To attackers, the presence of wormhole (in particular the location of the other end) should be hidden. The researchers made an assumption: attackers are using the Internet and in particular Shodan.io to identify potentially vulnerable or unsecured IoT devices such as webcams, smart fridges, etc. Shodan is a search engine that lists IoT devices found on Internet by using port scanning on IPs. Analyzing the HTTP responses of the devices, it identifies the type of device found and it obtains other useful informations (such as geo-locations, accessible ports, etc.) using fingerprinting heuristics. Moreover, the Shodan API provides a service called `Honeyscore` that tries to estimate if a device is a honeypot or not. The `Honeyscore` uses a heuristic to classify devices as potential honeypots, giving a score between 0 (not a honeypot) and 1 (honeypot).

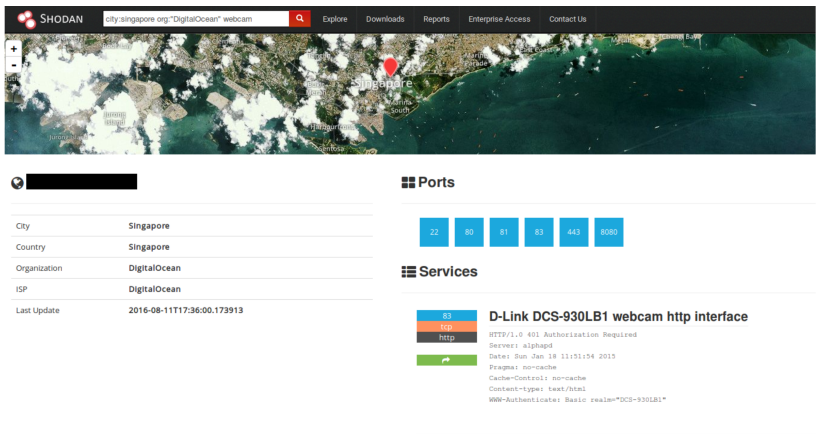


Figure 40: Example of device found by Shodan.io.

The devices under attack are assumed to be reachable directly from the Internet (i.e., on public IP addresses) exposing one or more services on open ports (e.g., `HTTP`, `Telnet`, `SSH`, etc.) and they're embedded in a real environment (i.e. cameras show live images, fridge contains real food, etc.). Devices are also configured with minimal effort (i.e., default passwords) and no dedicated security solutions are present (e.g., firewalls or IDS).
SIPHON is based on real physical IoT devices which are exposed to the Internet through a large number of tunnels that forward traffic from wormholes (remote public IP addresses) to the local physical IoT devices.

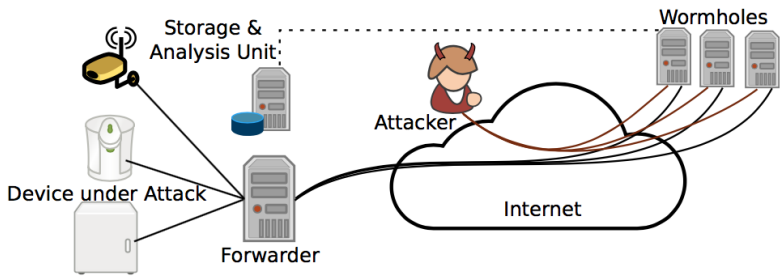


Figure 41: Overview of SIPHON.

- **Wormhole:** it presents a number of open ports to the Internet on a public IP address. Incoming traffic is either logged locally, or transparently forwarded to a specific port on a remote physical IoT device via the Forwarder. More than a physical device can be associated to each wormhole, but each service forwarded will need a unique port on the wormhole.
- **Forwarder:** it ensures that the traffic between the wormhole and the IoT device is rewritten in real-time to hide the fact that devices are physically located somewhere else, or communicating with many other tunnels.
- **Device under Attack (DUA):** the local IoT device under attack can be a normal commercial off-the shelf device. It will expose one or more services via the network (e.g., over `TCP`) and it's able to maintain a number of concurrent connections.
- **Storage & Analysis Unit (SAU):** the SAU obtains traffic records and general logs from the wormholes and aggregates the data for offline analysis.

As IoT devices are embedded in the physical world and interact with it, the researchers made the assumption that the physical location of the devics is correlated to its attractiveness to the attacker. So, wormholes should have public IP addresses that are localized in a range of physical locations, ideally spanning multiple countries or even continents.
Tunnels between the forwarder and wormholes can be realized using different forwarding techniques:

- **Application-Layer Proxy:** if the honeypot should represent only few services, it's possible to use proxies on the wormholes (e.g., `HTTP proxy`).
- **Encapsulation:** on transport and application layers, services such as `SSH` and `TLS` can be used to encapsulate and forward the selected traffic with low configuration overhead. SIPHON implementation uses an `SSH tunnel`.
- **Network-Layer VPN:** a number of network-layer VPN services are available (e.g., `IPSec`). Such services might have an advantage over selective forwarding using `SSH`, as large numbers of ports could be forwarded with little configuration.

The SIPHON architecture exposes real devices to the attacker and it could be possible for the attacker to fully compromise a device during an attack. Once compromised, the infected devices could be used to attack other devices or to distribute malware. There are several options to mitigate or prevent such scenarios:

- **Periodic Reset:** a simply approach is to setup periodically resets of the configuration and the firmware of IoT devices in the honeypot.
- **IPS:** Intrusion prevention systems like the open source `Suricata` could be used to prevent compromises. In addition, an IDS/IPS could be used to detect the changed behavior of an infected device to identify that a compromise has happened.
- **Full Instrumentation:** the most reliable detection of a compromise could be achieved through low-level instrumentation of IoT devices. The setup and maintenance of this methodology could be challenging and expensive.

For n physical IoT devices and m wormholes, the theoretical number of devices presented to the attacker by the SIPHON architecture (without exposing the same device twice in the same wormhole) can be up to $n \times m$. The number of services per wormhole is limited to around 65.000 due to port numbering restrictions while the number of wormholes per physical device is limited by the amount of parallel traffic that can be handled by each physical device. However in the prototype implementation of the honeypot lower values for both n and m were been chosen, due to budget and space constraints.

A first implementation of the SIPHON architecture provided the utilization of only a specific class of IoT devices: IP Cameras used for home or public surveillance. These devices have been chosen since the number of connected cameras is increasing daily and also because it has been proven that IP cameras are very vulnerable to attacks; thus, attracting more illegitimate traffic than other types of IoT devices in the market.

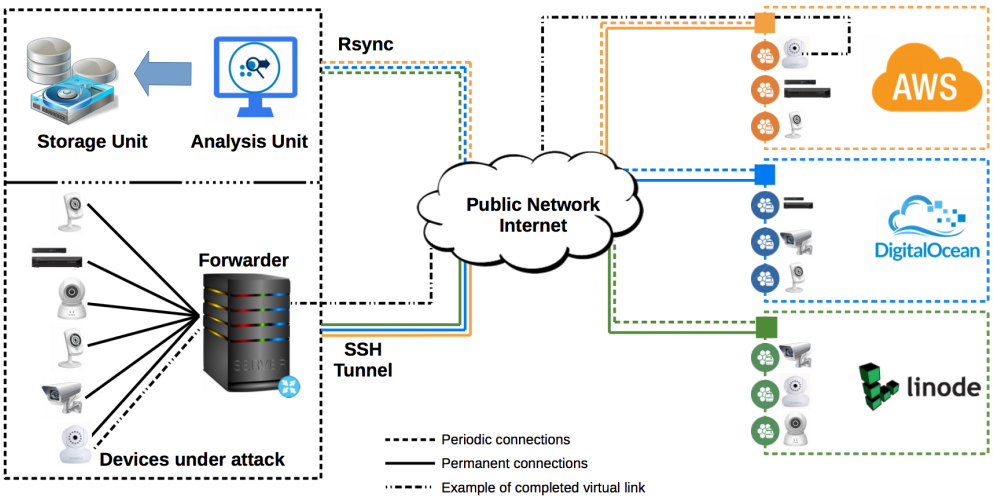


Figure 42: SIPHON prototype implementation.

All the devices included in the architecture expose an administration web interface protected by a password (weak or default password) in order to attract attacks which will be captured in order to analyze the complex interactions with the attacked devices. Administration web interfaces expose

to the attacker many services (e.g., camera motion, wi-fi network scanning, etc.) and multiple device interaction scenarios.
An important **difference** from other honeypot technologies is due to the choice to not consider the discovery and analysis of vulnerabilities exposed by `Telnet` and `SSH` ports. This has been made mainly because `Telnet/SSH` would not provide a high interaction environment and also because the chosen IoT devices used in the honeypot do not have open `Telnet/SSH` ports. Moreover, vulnerabilities over `Telnet/SSH` were already explored by IoTPOT project.
Summarizing, the IoT devices are connected to a total of 39 cloud servers in various cities (each one with a distinct public static IP address); each of the cloud servers is designated as a wormhole with up to 3 connected devices connected on different ports of the cloud server. In total, 85 devices were visible to the attackers on the Internet (since some of them are linked to different servers), while only 8 physical devices were used. Thus, in average about 10 virtual connections to each physical device are exposed.
All the traffic inside this architecture is captured using `tcpdump` on each wormhole server and the resulting `.pcap` samples are stored locally for offline analysis.
This setup relies on a network gateway scheme: to implement the forwarder, a local Linux Ubuntu server is used to manage all the TCP communications between the m wormholes to the n IoT devices. The connection between the forwarder and the cloud instances is established with reverse ssh tunnels that redirect traffic of a specific port (i.e., port 80) on the wormhole to a port in the forwarder. Once the traffic has reached the forwarder, it's redirected to the IoT devices and vice versa. This process is automatized by using a different local forwarder port for each wormhole.
Under the point of view of attackers, IoT devices are placed in cities where the actual wormhole is placed; physically, they can be located in a common area together with the forwarder.
The first SIPHON experimental deployment covered 16 cities in 9 different countries (San Jose, Boardman, Ashburn, New York, San Francisco, Dallas, Fremont and Newark in U.S.A., Toronto in Canada, Frankfurt, London, Dublin and Amsterdam in Europe, Singapore and Bangalore in Asia and Sydney in Australia) having a total cost (39 wormhole instances) of about 12 US $ per day.

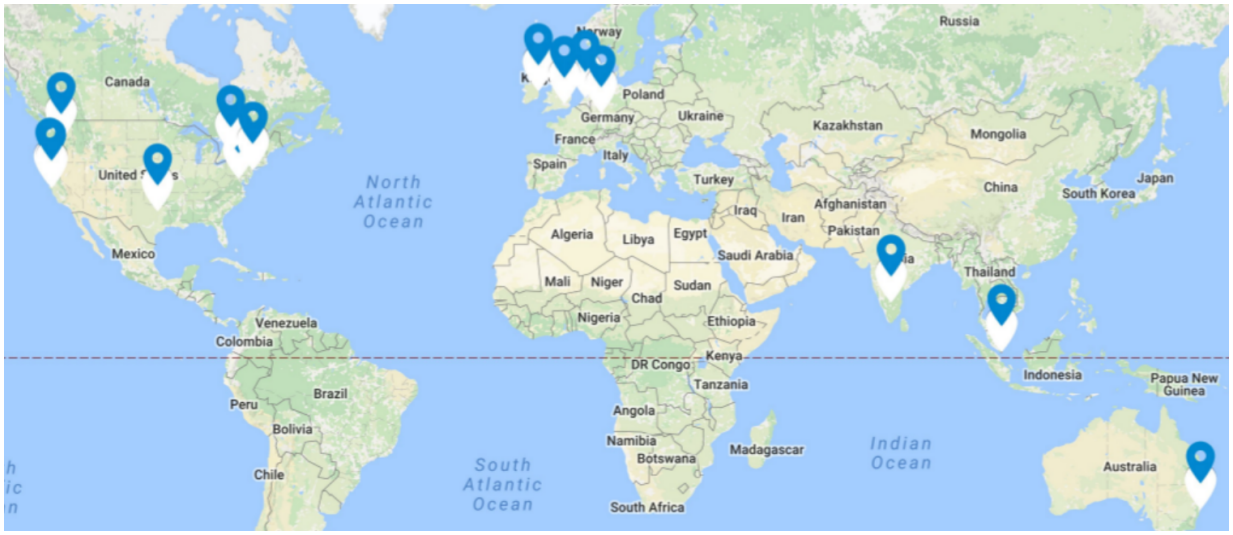


Figure 43: First SIPHON prototype locations.

The honeypot architecture was checked whether its IP addresses were marked as *"honeypot"* by Shodan or not: during the test, IP addresses were mostly not detected as honeypots; in particular

the `Honeyscore` value was in the range [0.0, 0.5] (not honeypot), with an average score of 0.16.

Interests of attackers may vary depending on several factors such as:

- **Location:** attackers may prefer to invest more time in certain cities than others. They search for vulnerable devices in a particular city motivated by a commercial interest of the location's IP for re-selling after infection, or as a starting point for targeted attacks.
- **Device type:** attackers may target particular models or device types that might have known vulnerabilities (e.g., Chinese Wireless IP Camera (P2) WIFICAM).
- **Shodan Listing:** attackers may target devices listed on Shodan.
- **High Interaction vs. Low Interaction:** attackers may behave differently when exposed to a low interaction implementation of IoT devices when the real device should perform a different action (IP camera showing a static image instead of video).

The target of SIPHON is to understand how many TCP connections are received per each wormhole, what kind of services are consulted, whether interactions are performed using known scripts or new ones, which is the number of different ports accessed by the same wormhole, as well as the user agent used in HTTP sessions. Moreover, since to have access to the admin interface of IoT devices the attacker needs to authenticate, the honeypot also counts the number of brute-force login attempts (successful or not), to further investigate what happens after a successful login.
All the network traffic is collected in raw `.pcaps` obtained with `tcpdump` and stored in an SQL database on a local machine. Then an offline analysis is performed by parsing the `.pcaps` samples obtaining different useful informations. Analyzing the data concerning the samples obtained by the honeypot during a period of two months, the researchers observed the following results:

- **Distinct remote IPs:** incoming TCP connections were obtained from over 13.000 distinct remote IP addresses.
- **Connections to wormhole per city:** wormholes in some cities received more attention than other ones in different locations.

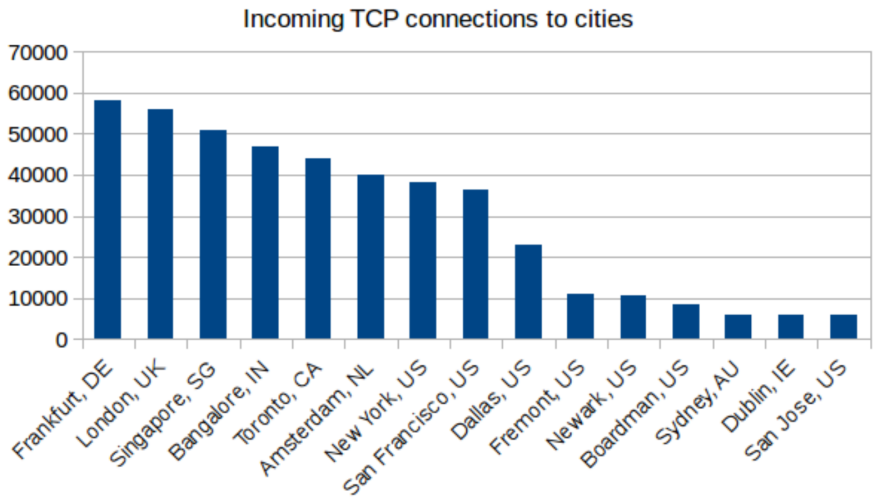


Figure 44: Distribution of TCP connections to wormholes per city.

◇ **Wormhole ports:** most of the incoming connections (97,45%) were directed on `SSH` port 22 of the wormholes, whilst `HTTP/HTTPS` ports 80 and 8080 received just 1,27% and 1,12% of total incoming TCP connections respectively. Other `HTTP` ports got only 0.16% of the total.

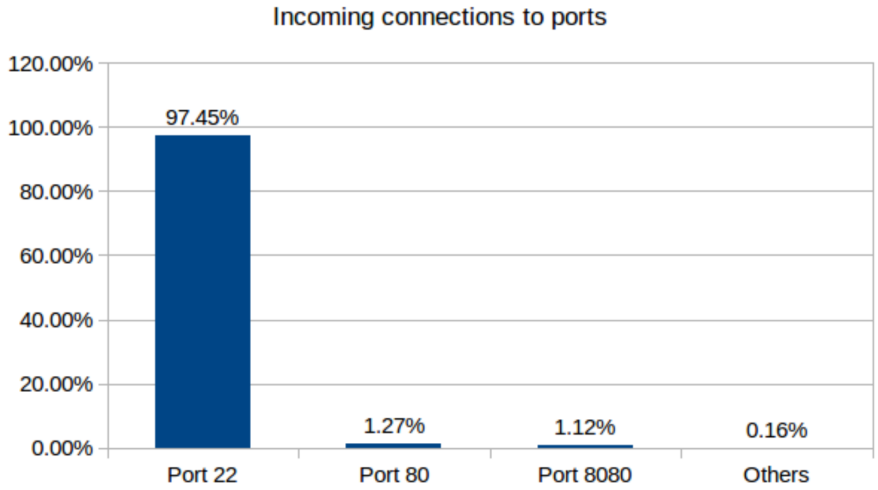


Figure 45: Connections to ports.

◇ **Location of attackers:** more than 70% of the attacks originated in China followed by 8% connections originating in USA. However, this is not a reliable indicator since attackers can easily tunnel through IPs located anywhere in the web.

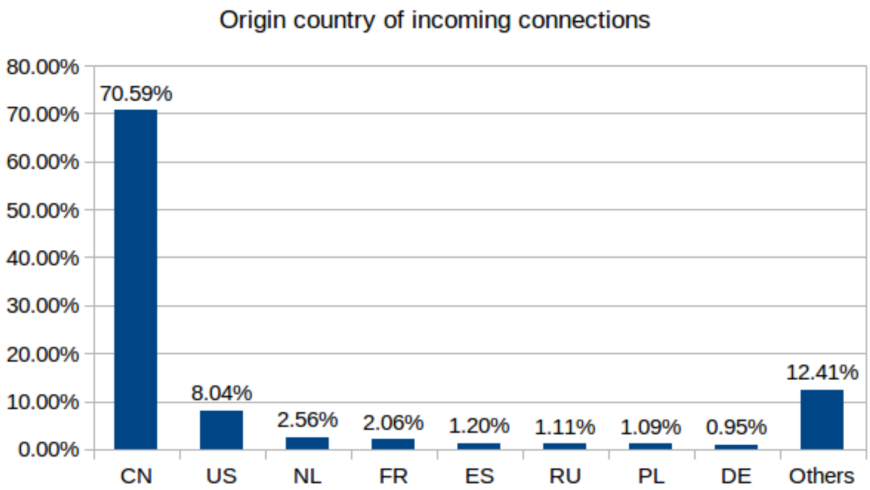


Figure 46: Country of origin of attacking connections.

◇ **Brute force login attempts:** different levels of password difficulties were configured on IoT devices. A total of 404 brute force attempts were observed on all the devices, exploited from 137 distinct remote IPs. A login attempt was considered as a brute force attempt if the same remote IP tried to perform more than 3 login attempts in an HTTP session. Of the total 404 brute force attempts, only 11 attempts succeed. All the successful login attempts were on devices with easy passwords that are present in most of the dictionaries used by brute-force attack tools; on devices with hard passwords instead, no successful login was recorded.

◇ **User agents used by attackers:** most of the connections (76%) were made using `Mozilla Firefox` as the user agent. Other user agents used were: `Chrome`, `Python Request`, `wget`, `Curl`, `Scanbot`, `Telesphoro` and `Masscanagents`. Moreover, around 9.000 Shell shock attempts on our devices were detected.

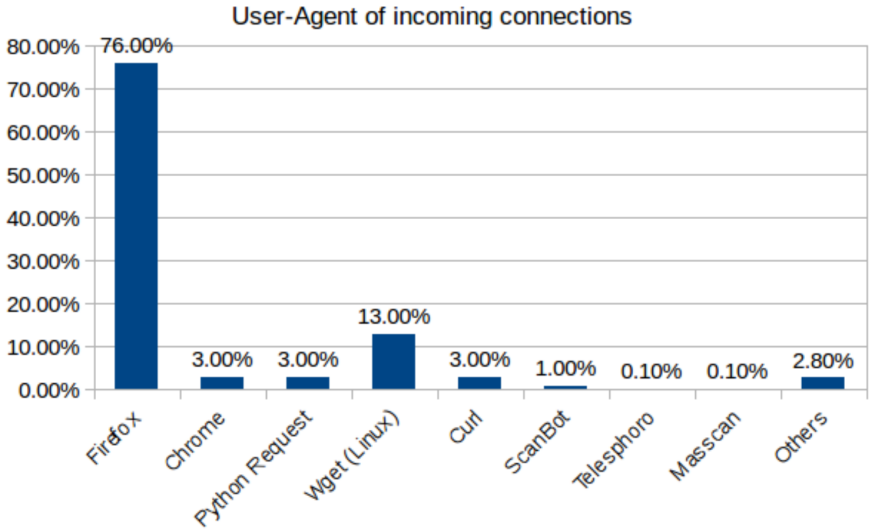


Figure 47: Incoming connections according to user agents.

◇ **Attacked devices:** the majority of traffic was directed towards the `DLink DCS-930L` IP camera maybe due to a report about the presence of vulnerabilities in its firmware that attracted the most attention from attackers on this particular device.

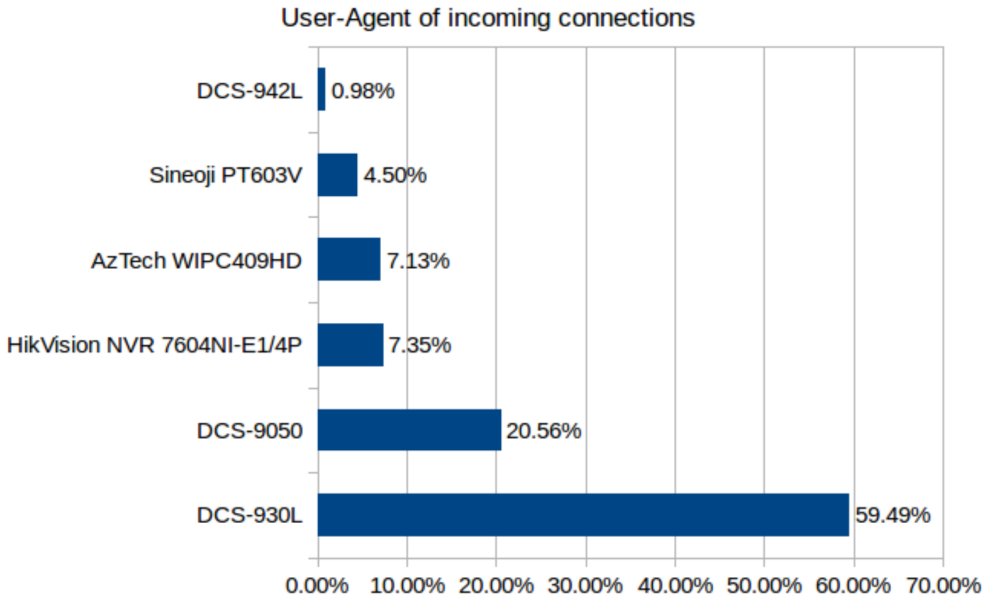


Figure 48: Targeted IoT Devices.

Another goal of the analysis of the data obtained during two months of execution was to investigate the impact of the listing of a wormhole on Shodan. So, the number of TCP connections received by wormholes were monitored before and after their Shodan listings. In particular, the impact of Shodan listing was measured according to the number of average connections the wormholes attracted per week before and after the listing: as expected, an increase of incoming connections was evident after that wormholes were listed on Shodan. In the long term, the number of incoming connections on that wormholes decreases (but it's still more than the beginning).

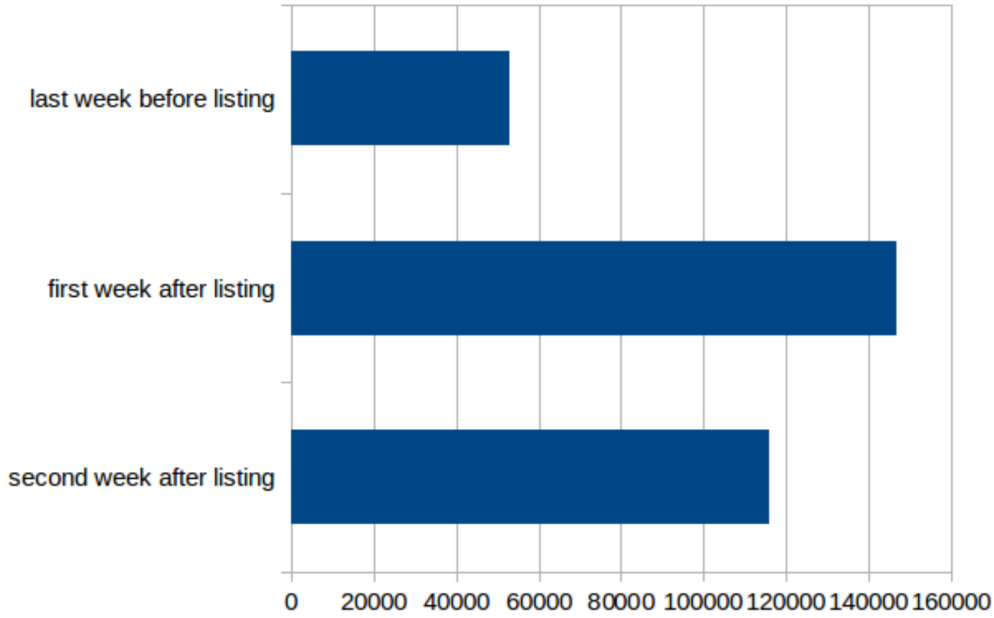


Figure 49: Impact of Shodan listing.

The first implementation of SIPHON included only devices representing a small subset of existing IoT devices (IP cameras and a NVR). However it obtained very promising results about IoT attacks. So, the researchers tried to add another IoT device (an IP printer) connected to a cloud instance located in London for about four weeks. After analyzing the obtained network traffic data from the IP printer, the results confirmed the trends observed before:

- **Wormhole ports:** 99% of the connections were on `SSH` port 22 of the wormholes, whereas `HTTP` port 80 received only the 1%. So, the results were similar to the ones IP cameras' case.

- **Location of attackers:** similar to the observation in the case of IP cameras, the highest percentage of incoming connections to the IP printer (79%), originated in China.

- **Brute force login attempts:** the credentials of the IP printer were changed to an easy password (`admin:1234567890`); only 1 brute force login attempt and a total of 4 successful login attempts were detected from 3 distinct remote IPs (all from China).

- **User agents used by attackers:** as in the case IP cameras, all the 4 successful login attempts were made using the `Mozilla Firefox` user agent.

So, the second experiment demonstrated that the architecture is scalable and allows the researchers to successfully introduce new devices.

The high interaction design of SIPHON obtained good results capturing interaction from human hackers... however it does not obtained interactions from automated botnets. This was due to the fact that low interaction honeypots are typically targeted by this type of attacks since botnets are not capable to distinguish between real or virtual fake devices.
So the research team implemented a low interaction honeypot using the same methodologies as of the high interaction one. However, instead of a real physical device they used the `Trendnet` camera emulator to emulate IP cameras in the low interaction honeypot. The wormhole (entry point) to the low interaction honeypot has been created from the cloud service located in Singapore. Then the researchers test the low interaction system for six weeks obtaining the following results:

- **Wormhole ports:** 87% of the connections to the fake IP camera were on `SSH` port 22, whereas `HTTP/HTTPS` ports 8080 and 80 received 12% and 1% of the total incoming connections respectively. So, the observed behavior is consistent with the trends observed with both high interaction honeypots.

- **User-agents used by attackers:** the user agent that were detected are a part of the same ones observed by the high interaction honeypots; the maximum percentage of incoming connections having `Mozilla Firefox` as the user agent and then followed by `wget`, `python` and `masscan`).

- **Login attempts:**. 3 successful login attempts with the default credentials `admin:admin` were detected from three distinct remote IPs. One successful login attempt was made each from China, Russia and Iran. Another interesting aspect was analyzed: in this case attackers did not spend enough time to explore the device or its functionality spending on an average only around 30 seconds (compared to about a minute or more in the case of real cameras in high interaction honeypot). It is understood that the duration of a session is correlated to human intervention: session time will be longer when the attacker will try to explore functionality of the device. Human beings can easily distinguish between a still image and a streaming video unlike automated bots. Thus, we can more easily determine manual attacks using this low interaction honeypot compared to our earlier high interaction honeypot.

By analyzing the traffic of the last experiment with a low interaction honeypot, the researchers demonstrated that the proposed architecture for a distributed and scalable high-interaction IoT honeypot consisting of real IoT devices was successful in attracting a high number of connections, confirming its overall design. That setup allows to capture and analyze real traffic and attacks across different geolocations and measure attacker behavior with respect to different locations. Thus, the SIPHON prototype demonstrated to be maybe the most promising technology which could used in order to improve the actual knowledge of the threat landscape in IoT security for what concerns both the human or automated nature of attacks.

Type	Honeypot	Year	Services	Feature
LOW	**Nepenthes**	2005-2009	FTP, HTTP, TFTP, MSSQL	Capture worm payload, Flexibility
	Dionaea	2009-2013	SMB, FTP, SIP, MYSQL, Telnet, HTTP/HTTPS	Capture worm payload, Nepenthes successor
	Glastopf	2011-2015	HTML, PHP, SQL	Web applications, Vulnerability types
	HoneyD	2003-2008	HTTP, POP3, SMTP, FTP	Heterogeneous Networks Emulation
	HoneyBOT	2002-2007	FTP, SSH, Telnet, SMTP, POP3, HTML, HTTP	Windows vulnerabilities and GUI
	Honeytrap	2005-2015	HTML, FTP, Dynamic Server Emulation	Attacks via TCP/UDP, Dynamic emulation
	Bluepot	2010-2015	Bluetooth	Bluetooth Malware
	Shockpot	2014-2015	Apache, Bash	Shellshock Vulnerability
	HoneyC	2004-2007	HTTP Client	Identify malicious servers by Snort IDS signatures
	Monkey-Spider	2007-2009	HTTP Client, JavaScript, Crawling	Threat DB creation, Anti Virus support
	YALIH	2014-2015	HTTP Client, JavaScript, IMAP	Advanced Combination of Analysis Methods
	MTPot	2016-2017	Telnet, Mirai Fingerprinting	Mirai Detection
	Telnet IoT Honeypot	2016-2017	Telnet	Telnet Emulation, Malware Collection
MEDIUM	**Kippo**	2009-2014	SSH, XML, MySQL, XMPP	Emulate shell interaction, Brute-Force on SSH
	Cowrie	2014-2017	Same of Kippo + Telnet, SFTP, SCP	Kippo's successor (Bug fixes, Better Logging)
	IoTPOT	2014-2015	Telnet	Low-Intereraction Front-End + High-Interation Virtual Environments
HIGH	**HoneyPHARM**	2016-2017	HoneyNet Implemenation	Network of Distributed Nepenthes honeypots
	SIPHON	2016-2017	Real Devices over a Distributed and Scalable Architecture	0-day Exploits Identification, Distributed HoneyNet, Attack Detection (both human and automantic)

Figure 50: Table of IoT Honeypots with related features.

References

[1] Brian Krebs Website.
https://krebsonsecurity.com/

[2] Arbor Networks.
Worldwide Infrastructure Security Report, Vol.XI.

[3] OWASP Top 10 IoT Vulnerabilities.
https://www.owasp.org/index.php/Top_IoT_Vulnerabilities

[4] CERT (Computer Emergency Response Team) Nazionale Italia.
https://www.certnazionale.it

[5] Paloalto Networks Research Center.
https://researchcenter.paloaltonetworks.com

[6] NJ Cybersecurity & Communications Integration Cell.
https://www.cyber.nj.gov

[7] Malware Must Die Blog.
http://blog.malwaremustdie.org

[8] Softpedia - Security News.
http://news.softpedia.com/cat/Security/

[9] TrendLabs Security Intelligence Blog.
http://blog.trendmicro.com/trendlabs-security-intelligence/

[10] N. Provos, T. Holz.
Virtual Honeypots: From Botnet Tracking to Intrusion Detection.

[11] L. Spitzner.
Honeypots: Tracking Hackers.

[12] E. Peter and T. Schiller.
A Practical Guide to Honeypots.

[13] K. Cabaj, P. Gawkowski.
HoneyPot systems in practice.

[14] N. Weiler.
Honeypots for Distributed Denial of Service Attacks.

[15] D. Akkaya and F. Thalgott.
Honeypots in Network Security.

[16] S. M. Specht, R. B. Lee.
Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures.

[17] K. M. Prasad, A. R. M. Reddy, K. V. Rao.
DoS and DDoS Attacks: Defense, Detection and Traceback Mechanisms - A Survey.

[18] S. Farraposo, L. Gallon, P. Owezarski.
Network Security and DoS Attacks.

[19] S. R. Oh, Y. G. Kim.
Smart World of Internet of Things (IoT) and it's Security Concerns.

[20] T. J. Charity, H. J. Hua.
Security Requirements Analysis for the IoT.

[21] M. Nawir, A. Amir, N. Yaakob, O. B. Lynn.
Internet of Things (IoT): Taxonomy of Security Attacks.

[22] F. A. Alabaa, M. Othmana, I. A. T. Hashema, F. Alotaibi.
Internet of Things security: A survey.

[23] P. Baecher, M. Koetter, T. Holz, M. Dornseif and F. Freiling.
The Nepenthes Platform: An Efficient Approach to Collect Malware.

[24] L. Rist.
Glastopf: a dynamic, low-interaction web application honeypot.
Github page: https://github.com/mushorg/glastopf

[25] N. Provos.
Honeyd: A Virtual Honeypot Daemon.
Github page: https://github.com/provos/honeyd

[26] A. L. Podhradsky, C. Casey and P. Ceretti.
The Bluetooth Honeypot Project.
Github page: https://github.com/andrewmichaelsmith/bluepot

[27] C. Seifert, I. Welch and P. Komisarczuk.
HoneyC - The Low-Interaction Client Honeypot.
Github page: https://projects.honeynet.org/honeyc

[28] A. Ikinci, T. Holz and F. Freiling.
Monkey-Spider: Detecting Malicious Websites with Low-Interaction Honeyclients.
Github page: http://monkeyspider.sourceforge.net/

[29] M. Mansoori, I. Welch and Qiang Fu.
YALIH, Yet Another Low Interaction Honeyclient.
Github page: https://github.com/Masood-M/yalih

[30] K. Koltys.
An in-depth look at Kippo: an integration perspective.
Github page: https://github.com/desaster/kippo

[31] Y.M. Pa Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama and C. Rossow.
IoTPOT: Analysing the Rise of IoT Compromises.

[32] A. Hassan and M. A. Ali.
Collecting malware from ditributed honrypots: HoneyPHARM.

[33] J. Guarnizo, A. Tambe, S.S. Bhunia, M. Ochoa, N. Tippenhauer, A. Shabtai and Y. Elovici.
SIPHON: Towards Scalable High-Interaction Physical Honeypots.