EMBEDDED DEVICE VULNERABILITY ANALYSIS CASE STUDY USING TROMMEL

Madison Oliver
Kyle O'Meara

November 2017

Executive Summary

Researching embedded devices is not always straightforward, as such devices often vastly differ from one another. Such research is difficult to repeat and results are not easily comparable because it is difficult to conceive a standard approach for analysis. This document proposes an initial research methodology for vulnerability analysis that can be applied to any embedded device. This methodology looks beyond preliminary research findings, such as open ports and running services, and takes a holistic, macro-level approach of the embedded device, to include an analysis of the firmware, web application, mobile application, and hardware. In addition, TROMMEL, an open source tool, was created to help researchers during embedded device vulnerability analysis.

This document provides security researchers with a repeatable methodology to produce more thorough and actionable results when analyzing embedded devices for vulnerabilities. As a case study, we analyzed a Wi-Fi camera as a class of embedded devices to demonstrate this methodology is more encompassing than standard research. This methodology can be applied to all embedded devices and should be expanded as the landscape of embedded device evolves.

1. Introduction

Embedded devices are becoming more ubiquitous. Embedded devices can be thought of as “a paradigm that considers pervasive presence in the environment of various things that through...connections are able to interact and cooperate with other connected things to create seamless communication and contextual services...” [Misra 2017]. Embedded devices have multiple components, both hardware and software, that can contain vulnerabilities. All of these different components can contain exploitable vulnerabilities and provide additional entry points for attackers.

These devices have multiple components that are often overlooked during analysis because researchers typically focus on one area or part of the device, such as changing memory contents, or have not applied their findings to a physical device [Seshadri 2004; Papp 2015].

The proposed methodology ensures that a researcher considers all components of the device during analysis and broadens the footprint of the results. This methodology is meant to be a living document and should be adapted to include additional steps as the embedded device ecosystem evolves. We outline the following methodology to take during research: embedded device list curation and identification, general information gathering, firmware analysis, web application analysis, mobile application analysis, hardware analysis, and vulnerability analysis.

2. TROMMEL

We created a custom, open-source tool using Python called TROMMEL to assist researchers during embedded device vulnerability analysis [TROMMEL 2017]. TROMMEL sifts through directories of files to identify indicators that may contain vulnerabilities. Specifically, TROMMEL identifies the following indicators:

- Secure Shell (SSH) key files
- Secure Socket Layer (SSL) key files
- Internet Protocol (IP) addresses
- Uniform Resource Locators (URLs)
- email addresses
- shell scripts
- web server binaries
- configuration files
- database files
- specific binaries files (for example, Dropbear, BusyBox, and others)
- shared object library files
- web application scripting variables
- Android application package (APK) file permissions

Also, TROMMEL integrates vFeed, which is a database wrapper that pulls in content from Common Vulnerabilities and Exposures (CVE) database, Exploit-DB, and Metasploit [Toolswatch 2017]. vFeed offers a free, downloadable Community Database for non-commercial users [Toolswatch 2017]. This integration allows for further in-depth vulnerability analysis of identified indicators. TROMMEL significantly lessens the manual analysis time of the researcher by automating much of the vulnerability discovery and analysis process.

3. Methodology

We created this methodology to enable other security researchers to follow a more uniform and scientific research process when reviewing embedded devices. Given that a methodology does not presently exist for embedded devices, our goal was to provide a way for researchers to apply a repeatable, macro-level process to embedded device research or analysis. Researchers should apply this methodology to analyze disparate types of embedded devices, and as the ecosystem evolves, should add additional components to account for the changing landscape.

We provide a case study using a Wi-Fi camera to highlight the key points for each part of the process.

3.1. Embedded Device List Curation and Identification

Given a particular challenge or set of requirements, researchers should develop a list of common embedded devices. To further narrow this list of candidate devices, researchers should then consider:

1. Ease of physical access to the device. Physical access allows for more complete results.
2. Ease of access to the firmware. Firmware access is necessary to have comprehensive results. Researchers should consider the number of firmware versions, the availability of this firmware, and whether the firmware is still being maintained.

3.2. General Information Gathering

After selecting a device, researchers should perform significant information gathering on the device and the device class. Additional research points should be:

1. Vulnerabilities, past or present, that have been associated with the vendor and with vendor devices that are similar to the selected device. These vulnerabilities are likely still present on older devices and on older firmware versions that are still used by consumers.
2. Exploits. There are multiple Internet resources, such as Exploit-DB and VirusTotal, that can be used to discover exploits for the vulnerabilities found.

3.3. Firmware Analysis

After information gathering is complete, the contents of the device firmware image should be extracted and analyzed for potential vulnerabilities. There are multiple tools that can be used, but for our analysis, we chose open-source tools that included Binwalk and TROMMEL.

3.3.1. Binwalk

Binwalk is an open source tool that is used to analyze, reverse engineer, and extract firmware images [devttys0 2017]. Though Binwalk has many advanced features, two standard options are useful:

1. Pass the firmware image to Binwalk with no options. This allows review of the contents of the firmware image before extraction and reveals whether:

    a. The firmware contains data that can be extracted.

    b. The firmware contains a file system. Common file systems found in the firmware of embedded devices are: SquashFS, compressed ROM file system (CramFS), Journalling Flash File System version 2 (JFFS2), Yet Another Flash File System (YAFFS2), and second extended filesystem (ext2).

2. After a file system is identified, pass the firmware image to Binwalk with extraction options. This will extract all content identified by Binwalk, including—but not limited to—the file system directory. The researcher will need to review and recursively search this new directory for the file system folder and perform follow-on analysis.

3.3.2. TROMMEL

We chose to use TROMMEL for firmware analysis because it supports identification of indicators, mentioned above, that are found in file systems extracted from embedded device firmware.

3.4. Web Application Analysis

In many cases, embedded devices usually contain a web application interface that the user can access to control the device. These interfaces are usually administrative in nature and are used to adjust settings, either through a web, cloud, or mobile platform [OWASP 2016]. This interface should be thoroughly tested for vulnerabilities such as information leakage or unauthorized access. The interface can be tested manually or with automated tools. We chose to use automation, owing to improved test speed and reliability.

There are many open-source web application tools available. Using a variety of tools is recommended, as results may vary from tool to tool. Web application scanners, which typically test for a wide array of web application vulnerabilities, are the most common type of tool used in web application analysis.

Some examples of the most commonly used scanners are [Infosec Institute 2017; eHacking 2011]:

- Burp Suite
- Grabber
- OWASP ZAP
- Nikto
- W3af
- Wapiti
- Vega

In this case study, we used two different open source tools because of they are extremely informative and easy to use:

1. OWASP ZAP uses a proxy to intercept website requests and then to scan actively and passively for vulnerabilities [OWASP 2017].
2. Nikto is a web server scanner that searches for potentially malicious files, checks for outdated server versions, and looks for specific known problems in the server versions found [Sullo 2017].

We suggest using the default settings in each tool, both tools were given the administrative webpage IP address, and Nikto was given administrative login credentials. These tools are also able to thoroughly crawl the entire website and return results about potential vulnerabilities, such as cross-site scripting or Structured Query Language injections (SQLi).

3.5. Android Mobile Application Analysis

Mobile application analysis includes the process of analyzing the decoded package files of an Android mobile application. While this is not an exhaustive list, researchers should attempt to analyze the Android mobile application with Apktool and TROMMEL.

3.5.1. Apktool

Apktool is a tool for reverse engineering Android Package Kit (APK) files [Wiśniewski 2017]. To decompile the APK file, the researcher provides Apktool with the decode option and the absolute path to the APK file. Apktool saves the decoded content to a directory named [APK file name].out.

3.5.2. TROMMEL

We chose to use TROMMEL for mobile application analysis because it supports identification of indicators, mentioned above, that can be found in decompiled APK files.

3.6. Hardware Analysis

There are multiple steps that can be taken for the hardware analysis process, but we recommend the following five steps.

3.6.1. Step 1: Identify all markings on the case of the device

The researcher should research all markings on the case of the device, then compare findings against the vendor's website and against any information collected during the Information Gathering phase of the methodology. Information that can be identified on the case of the device might include product number, model number, serial number, media access control (MAC) address, Wi-Fi service set identifier (SSID), and Wi-Fi password. If the device has any radio communication, the case might have regulatory numbers for countries it was registered with such as the Federal Communication Commission (FCC) or Industry Canada (IC). These regulatory agencies store device information in a publicly searchable database. Typically, the database will have documentation that was provided by the vendor during the registration process. This documentation may include datasheets and can be

used to further identify or verify internal components of the hardware device, as well as supplement the data collected during the Information Gathering phase.

3.6.2. Step 2: Identify the components internal to the device located on printed circuit board (PCB)

The researcher must open the device to expose the PCB with the goal of identifying the flash memory component. Any datasheets discovered during the first step of hardware analysis will be useful in identifying the components on the PCB. If no datasheets were discovered, the goal is to identify any labeling on all components present on the PCB. This labeling will include alphanumeric strings and potentially a vendor logo, which the researcher can use to search for datasheets and further information from the respective vendor website or the Internet at large. It may be difficult to locate datasheets, but they are necessary to identify the pin layout of the flash memory component.

3.6.3. Step 3: Dump memory from flash memory component to firmware image binary

The third step requires more advanced skills, but the goal is to dump the contents of the flash memory component to a firmware image binary. There are many different tools that can be used to extract firmware images from devices. After identifying the pin layout of the flash memory component, we recommend the following tools for attempting memory extraction: a Linux machine with flashrom installed, jumper wires, ultra-fine test clips, a Bus Pirate, and a Bus Pirate cable [FireEye 2017**]**. We chose this setup due to the integration between BusPirate and flashrom, as well as the fact flashrom supports a variety of makes and models of flash memory components. If flashrom (in conjunction with the BusPirate) can identify the flash component, it will extract the contents of the component to a binary file in the working directory.

3.6.4. Step 4: Extract and analyze the content of firmware image binary

The fourth step is to take the firmware image binary dumped from the flash memory component in step 3 and follow the steps in section 3.3.1. regarding the use of the Binwalk tool.

3.6.5. Step 5: Compare the contents extracted from the flash memory component to relative firmware

If applicable, in the fifth step, the researcher should compare the contents extracted from the flash memory binary to firmware versions gathered during the Embedded Device List Curation and Identification phase of the methodology.

3.7. Vulnerability Analysis

After the appropriate information is gathered in the above steps, researchers should proceed to vulnerability testing, exploit testing, and coordination.

3.7.1. Vulnerability and Exploit Testing

When testing, we suggest two separate steps:

1. Review the firmware updates for patches against the vulnerabilities that were found during background research to determine if and when these vulnerabilities were fixed. This information can

typically be found in any release notes associated with the update and requires manual review. If there is no documentation that the vulnerabilities were fixed, the researcher should test for these vulnerabilities on every firmware version to determine if the device is still vulnerable. If the vendor claims that the vulnerabilities have been fixed with a software or firmware update, the researchers should then check these fixes before and after the update to ensure that the vulnerability was fixed.

2. Test the actual exploits against the device. All exploits gathered during the Information Gathering section that are associated with other devices from the same vendor should be tested against the chosen device to determine if there is any crossover of vulnerabilities. Researchers should be cautious when downloading exploits, review the exploit code before testing it against the chosen device, and use isolated systems/networks.

3.7.2. Vulnerability Disclosure

Novel vulnerabilities found during this process can be disclosed in a few different ways. Cencini, Yu, and Chan define three different forms of vulnerability disclosure: non-disclosure, full disclosure, and responsible disclosure [Cencini 2005]. Non-disclosure is when a researcher keeps a vulnerability secret and does not disclose it to anyone. Full disclosure is when a researcher informs the public of the vulnerability. Responsible disclosure, or limited disclosure, is when the researcher informs the software vendor and gives them a specific amount of time that they must issue a response. After this time passes, the researcher can disclose the vulnerability to the public with or without a patch from the vendor [Cencini 2005].

The CERT Guide to Coordinated Vulnerability Disclosure defines Coordinated Vulnerability Disclosure as "the process of gathering information from vulnerability fixers, coordinating the sharing of that information between relevant stakeholders, and disclosing the existence of software vulnerabilities and their mitigations to various stakeholders, including the public" [Householder 2017]. The CERT Guide encourages responsible disclosure and can assist researchers with disclosing to organizations that do not have a process set up yet.

4. Case Study Application and Results

This section outlines the results of the proposed methodology using a DCS-935L D-Link camera. Some of the results are limited, as we are currently coordinating the vulnerabilities with the vendor.

The following sections correspond to the methodology described in Section 3.

4.1. Embedded Device List Curation and Identification

After compiling a list of nearly 300 different embedded devices, we chose a D-Link camera because D-Link makes their firmware readily available, and although there has been significant analysis completed on D-Link routers, there has been less analysis on their cameras. Table 1 includes a list of the DCS-935L firmware versions analyzed.

Table 1: D-Link DCS-935L Firmware Versions

Camera Model	Firmware Version
935L	1.04
	1.06
	1.08
	1.09
	1.10

4.2. General Information Gathering

The Appendix contains the list of known vulnerabilities, exploits, and corresponding CVE pairings for the DCS-935L and similar devices.

4.3. Firmware Analysis

The following results are for firmware v1.10.01 for DCS-935L, which was downloaded directly from the D-Link website. This was the newest available firmware at the time of writing this whitepaper.

4.3.1. Binwalk

Binwalk identified a file system embedded in the firmware v1.10.01, which was then extracted for follow-on analysis using TROMMEL.

4.3.2. TROMMEL

Table 2 shows two indicators that contained extensive product information, which aid the researcher in vulnerability analysis. These files provided extensive wireless details about the product that was previously not found anywhere else during the General Information Gathering phase.

Table 2: TROMMEL Results: Files of Interest from Firmware Image

File name and location
/root/etc/Wireless/wscd.conf
/root/etc/Wireless/RTL8192CD_static.dat

With the integration of vFeed into TROMMEL, v1.22.1 of BusyBox was identified, which is out of date. The newest version available as of this writing is v1.27.2 [BusyBox 2017]. vFeed also assisted in providing further vulnerability analysis on the out-of-date version of Busybox:

- CVE-2016-2148 affects BusyBox versions before 1.25, CVE-2016-2147 affects BusyBox version 1.25, and CVE-2014-9645 affects BusyBox version 1.23 [Toolswatch 2017].
- At the time of writing this whitepaper, no Exploit-DB entries or Metasploit modules exist for the above three CVE [Toolswatch 2017]. However, this does not negate the fact that the current BusyBox version in firmware v1.10.01 contains known vulnerabilities.

4.3.3. Web Application Analysis

As stated in the Web Application Analysis section, both tools used in this phase, OWASP ZAP and Nikto, are open-source web application testing tools.

OWASP ZAP spidered the web pages and performed active scans against the administrative web interface while returning the results detailed in Table 3. OWASP ZAP returned four different findings when testing the administrative webpage. Two of these findings are related to best practices of securing a web interface. The other two findings are vulnerabilities that have not been previously reported. We contacted the vendor and followed the steps outlined in the Vulnerability Disclosure phase to properly disclose these vulnerabilities.

Nikto confirmed most of our findings from OWASP ZAP, including one of the newly discovered vulnerabilities. It also found one additional vulnerability that OWASP ZAP did not find.

The results, of previously reported findings, of these scans are included in Table 3 and Table 4.

Table 3: OWASP ZAP Scan Results

Finding/Vulnerability	Status
The X-XSS-Protection Header is not defined	Best Practice
The X-Content-Type-Options header is not set.	Best Practice

Table 4: Nikto Scan Results

Finding/Vulnerability	Status
The X-XSS-Protection Header is not defined	Best Practice
The X-Content-Type-Options header is not set.	Best Practice
/crossdomain.xml contains a full wildcard entry.	Best Practice

4.3.4. Android Mobile Application Analysis

4.3.4.1. Apktool

We downloaded the APK file com.dlink.mydlink from the Google Play store. D-Link customer service identified this file as the Android application used to administer the DCS-935L D-Link camera. We then successfully decoded the APK using Apktool to the specified output directory. This directory was used for follow-on analysis by TROMMEL.

4.3.4.2. TROMMEL

TROMMEL was executed on the decoded APK output directory. The indicators were reviewed and verified. No vulnerabilities were identified.

4.3.5. Hardware Analysis

4.3.5.1. Identify all markings on the physical case of the device

The physical case for the DCS-935L provided the following information:

- Firmware Version
- Hardware Version
- MAC Address
- Model
- Model Number
- mydlink Number
- Part Number
- QR Code
- Regulatory Identification Numbers for United States, Canada, Europe, and China
- Serial Number
- Wi-Fi Password
- Wi-Fi SSID

The documents in these regulatory databases did not provide any information that was beneficial to this analysis.

4.3.5.2. Identify the components internal to the device located on printed circuit board (PCB)

After we opened the case, we examined the components on PCB with the main goal of identifying the flash component. We identified five components of interest, then found their respective datasheets and information from vendor websites on the Internet. See the Appendix for links to these datasheets.

- RealTek RTS5826 EAC33H1 GE412
- RealTek RTL8881AB D9C47P5 GE38 Taiwan
- Winbond W9751G6KB-25 64535P900B02 516PUA TWN
- Macronix MXIC MX 25L12835F M21-10G 8B17230 L151
- Macronix MXIC 25L1006EMI-10G
- Skyworks SKY11 85703 517W5

Upon reviewing the datasheets further, we identified the flash memory component as the Macronix MXIC MX 25L12835F M21-10G 8B17230 L151.

4.3.5.3. Dump memory from flash memory component to firmware image binary

We used a BusPirate in conjunction with flashrom running on a Linux virtual machine to successfully extract the firmware image from one of the Macronix flash memory components. This involved identifying the pins on the Macronix flash memory component, connecting them to the proper BusPirate connections, plugging the BusPirate into the Linux virtual machine, and running flashrom.

Flashrom successfully identified the Macronix flash memory component and successfully extracted the contents of the Macronix component to a binary file. We labeled this extracted firmware as v1.04 because this was the firmware version listed on the case of the DCS-935L.

4.3.5.4. Extract and analyze the content of firmware image binary

We used the same techniques outlined in 3.3.1. Binwalk and 3.3.2. TROMMEL to analyze the firmware v1.04. Similar to the firmware v.1.10.01, firmware v1.04 contained an out-of-date BusyBox version (v1.13.4). The newest version as of this writing is 1.27.2 [BusyBox 2017]. vFeed also assisted with further vulnerability analysis of the out-of-date version of Busybox:

- Six CVEs were found to affect BusyBox version v1.13.4:
  - CVE-2016-2148, CVE-2016-2147, CVE-2014-9645, CVE-2013-1813, CVE-2011-5325, and CVE-2011-2716 [Toolswatch 2017].
- At the time of writing this paper, no Exploit-DB entries or Metasploit modules exist for the above six CVE [Toolswatch 2017]. This does not negate the fact that the current BusyBox version in firmware v1.04 is vulnerable.

4.3.6. Compare the contents extracted from the flash memory component to relative firmware

We compared files between v1.04 and v1.04.06, the earliest firmware available from D-Link's website, and between v1.04 and v1.10.01, the latest firmware available from D-Link's website, using the MD5 hash values of the respective extracted files. The goal was to see how many files did not change between versions. Firmware v1.04 shared 4.3% of the files found in firmware v1.04.06 and 3.7% of the files found in v1.10.01. The most common files across all firmware versions were shared object library files. This could be a future concern if a vulnerability is found in a library file because it will affect multiple versions.

4.3.7. Vulnerability Analysis

4.3.7.1. Vulnerability and Exploit Research and Testing

To determine when and if vulnerabilities had been fixed, we first reviewed the firmware updates and documentation that had been posted publicly by the vendor. We manually reviewed these documents from version 1.04 to version 1.10 for the DCS-935L. This helped us to determine that two of the vulnerabilities found that had not previously been reported were not fixed in any of the newer firmware versions.

We then attempted known exploits against the DCS-935L running firmware v1.04. Our research found 16 different vulnerabilities (some with exploits) for many different model cameras, not including the camera being tested. A list of these vulnerabilities is located in the Appendix. The vulnerabilities with exploits were tested against the DCS-935L; however, none of these exploits were successful.

4.3.7.2. Vulnerability Disclosure

We are following the process outlined by the vendor to properly disclose the newly discovered vulnerabilities. Like many vendors, D-Link has a dedicated vulnerability disclosure process and we have submitted our findings to them [D-Link 2017].

5. Future Work

This paper presents a research methodology that can be applied to many different embedded devices. However, this methodology is meant to evolve and grow. Future work could consist of applying this methodology to other embedded devices, including other D-Link cameras, cameras from different vendors, and completely different types of embedded devices.

We also intend to develop future phases of the methodology. These phases should include radio frequency (RF) analysis, advanced hardware analysis, and binary analysis of ARM and MIPS files. These additional phases should provide an even greater analysis footprint for use in analyzing embedded devices.

6. Conclusion

This methodology was developed to create a holistic, macro-level approach to vulnerability analysis of embedded devices. Our goal was to create a methodology that researchers could follow to create more comprehensive and actionable results. Our methodology includes embedded device list curation and identification, general information gathering, and vulnerability analysis of the firmware, web application, mobile application, and hardware.

We tested this methodology on a class of embedded devices (Wi-Fi camera) and found vulnerabilities that had not yet been published. We also developed and published an open-source tool, TROMMEL, to aid researchers in embedded device vulnerability analysis. This methodology can be applied to other embedded devices such as refrigerators, door locks, light switches, and automobiles.

This methodology should be treated as a living document. We expect the methodology to not only expand with our future work but also with evolution of the embedded device landscape.

Appendix

Table 4: D-Link Known Camera Vulnerabilities

Vulnerability	Model Affected	Link
Remote Code Execution	930L	https://www.exploit-db.com/exploits/39437/
CVE-2015-2049/ VU#377348 Arbitrary File Upload	931L	https://www.rapid7.com/db/modules/exploit/linux/http/dlink_dcs931l_upload
Unauthenticated Remote Access	Non-specific	https://www.exploit-db.com/exploits/24442/
CVE-2014-9645 Bypass Authentication	Non-specific	http://www.cvedetails.com/cve/CVE-2014-9645/
CVE-2016-6301 Denial of Service	Non-specific	https://nvd.nist.gov/vuln/detail/CVE-2016-6301
VU#377348 Unrestricted Upload	93*L Family	http://www.kb.cert.org/vuls/id/377348
CVE 2017-7852 CSRF	933L, 5020L	https://nvd.nist.gov/vuln/detail/CVE-2017-7852
SSL Certificate Vulnerability (patched)	5020L	http://news.softpedia.com/news/D-Link-Fixes-Persistent-SSL-Certificate-Vulnerability-in-DCS-IP-Cameras-429622.shtml
CVE-2014-9517 XSS	2103	https://nvd.nist.gov/vuln/detail/CVE-2014-9517
CVE-2012-4046 Information Exposure	932L	http://www.cvedetails.com/cve/CVE-2012-4046/
CVE-2012-5306 Denial of Service	5605	https://www.cvedetails.com/cve/CVE-2012-5306/
CVE-2004-1650 Remote Access	900	https://www.cvedetails.com/cve/CVE-2004-1650/
CVE-2014-9238 Remote Access	2103	https://www.cvedetails.com/cve/CVE-2014-9238/

Vulnerability	Model Affected	Link
CVE-2014-9234 Directory Traversal	2103	https://www.cvedetails.com/cve/CVE-2014-9234/
CVE-2006-5536 Directory Traversal	Non-specific	https://nvd.nist.gov/vuln/detail/CVE-2006-5536
CVE-2015-2048 CSRF	931L	https://www.cvedetails.com/cve/CVE-2015-2048/

Table 5: PCB Components and Datasheets

Component	Datasheets
RealTek RTS5826 EAC33H1 GE412	http://www.realtek.com/press/newsViewOne.aspx?NewsID=336
RealTek RTL8881AB D9C47P5 GE38 Taiwan	http://www.realtek.com/press/newsViewOne.aspx?NewsID=336
Winbond W9751G6KB-25 64535P900B02 516PUA TWN	http://www.winbond.com/resource-files/da00-w9751g6kbg1.pdf
MXIC MX 25L12835F M21-10G 8B17230 L151	http://www.macronix.com/Lists/Datasheet/Attachments/6228/MX25L12835F,%203V,%20128Mb,%20v1.6.pdf
MXIC 25L1006EMI-10G	http://www.macronix.com/Lists/Datasheet/Attachments/6189/MX25L1006E,%203V,%201Mb,%20v1.4.pdf
SKY11 85703 517W5	http://www.skyworksinc.com/uploads/documents/SKY85703_11_202991A.pdf

References

[BusyBox 2017]
BusyBox.net downloads. https://busybox.net/downloads/ October 18, 2017 [accessed]

[Cencini 2005]
Software Vulnerabilities: Full-, Responsible-, and Non-Disclosure. https://courses.cs.washington.edu/courses/csep590/05au/whitepaper_turnin/software_vulnerabilities_by_cencini_yu_chan.pdf October 24, 2017 [accessed]

[D-Link 2017]
D-Link. Report a Suspected Security Vulnerability. http://support.dlink.com/ReportVulnerabilities.aspx November 10, 2017 [accessed]

[devttys0 2017]
devttys0. binwalk: fast, easy to use tool for analyzing, reverse engineering, and extracting firmware images. *GitHub.* https://github.com/devttys0/binwalk October 18, 2017 [accessed]

[eHacking 2011]
Top 6 Web Vulnerability Scanner Tool. https://www.ehacking.net/2011/08/top-6-web-vulnerability-scanner-tool.html October 18, 2017 [accessed]

[Householder 2017]
Householder, Allen D.; Wassermann, Garret; Manion, Art; & King, Chris. "The CERT® Guide to Coordinated Vulnerability Disclosure." *Software Engineering Institute. Carnegie Mellon University.* August 2017. CMU/SEI-2017-SR-022. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=503330 October 18, 2017 [accessed]

[InfoSec Institute 2017]
14 Best Open Source Web Application Vulnerability Scanners. [Updated for 2017]. *InfoSec Institute*. May 24, 2017. http://resources.infosecinstitute.com/14-popular-web-application-vulnerability-scanners/#gref October 18, 2017 [accessed]

[Misra 2017]
Misra, Sridipta; Maheswaran, Muthucumaru; & Hashmi, Salman. *Security challenges and approaches in internet of things*. Springer Briefs in Electrical and Computer Engineering. Springer, 2017. https://link.springer.com/content/pdf/10.1007/978-3-319-44230-3.pdf October 18, 2017 [accessed]

[OWASP 2016]
IoT Testing Guide. *Open Web Application Security Project* (OWASP). May 14, 2016 https://www.owasp.org/index.php/IoT_Testing_Guides October 18, 2017 [accessed]

[OWASP 2017]
OWASP Zed Attack Proxy Project. *Open Web Application Security Project* (OWASP). August 9, 2017 https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project October 18, 2017 [accessed]

[Papp 2015]
Papp, Dorottya; Ma, Zhendong; & Buttyan, Levente. "Embedded systems security: Threats, vulnerabilities, and attack taxonomy." *13th Annual Conference on Privacy, Security and Trust (PST).* pp. 145-152. 2015. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7232966 October 18, 2017 [accessed]

[Seshadri 2004]
Seshadri, A.; Perrig, A.; Van Doorn, L.; & Khosla, P. (2004, May). Swatt: Software-based attestation for embedded devices. *Proceedings 2004 IEEE Symposium on Security and Privacy*. pp. 272-282. http://ieeexplore.ieee.org/document/1301329/ October 18, 2017 [accessed]

[Sullo 2017]
Sullo, Chris & Lodge, David. Nikto2. *CIRT.net.* https://cirt.net/nikto2 October 18, 2017 [accessed]

[Toolswatch 2017]
Toolswatch. vFeed The Correlated Vulnerability and Threat Intelligence Database Wrapper. *GitHub.* https://github.com/toolswatch/vFeed October 18, 2017 [accessed]

[TROMMEL 2017]
TROMMEL. TROMMEL sifts through directories of files to identify indicators that may contain vulnerabilities. *GitHub.* https://github.com/CERTCC-Vulnerability-Analysis/trommel October 19, 2017 [accessed]

[Wiśniewski 2017]
Wiśniewski, Ryszard & Tumbleson, Connor. Apktool - A tool for reverse engineering Android apk files. https://ibotpeaches.github.io/Apktool/ October 18, 2017 [accessed]

Bibliography

[CERT 2017]
Vulnerability Disclosure Policy. 2017 CERT. *Software Engineering Institute. Carnegie Mellon University.* https://www.cert.org/vulnerability-analysis/vul-disclosure.cfm October 18, 2017 [accessed]

[Chahid 2017]
Chahid, Y.; Benabdellah, M.; & Azizi, A. "Internet of things security," *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, Fez, 2017, pp. 1-6. http://ieeexplore.ieee.org/document/7934655/ October 18, 2017 [accessed]

[Chirgwin 2017]
Chirgwin, Richard. Dishwasher has directory traversal bug. *The Register*. March 26, 2017 https://www.theregister.co.uk/2017/03/26/miele_joins_internetofst_hall_of_shame/ October 18, 2017 [accessed]

[FCC 2014]
FCC Filings for D-Link HD Wi-Fi Camera, model DCS-935L. FCC ID=KA2CS935LA1. *U.S. Federal Communication Commission.* https://apps.fcc.gov/oetcf/eas/reports/ViewExhibitReport.cfm?mode=Exhibits&RequestTimeout=500&calledFromFrame=N&application_id=sQUCgOD%2BFyf%2FGKTeMBTYWg%3D%3D&fcc_id=KA2CS935LA1 October 18, 2017 [accessed]

[FireEye 2017]
Embedded Hardware Hacking 101 – The Belkin WeMo Link. *FireEye.* August 22, 2016 https://www.fireeye.com/blog/threat-research/2016/08/embedded_hardwareha.html November 10, 2017 [accessed]

[flashrom 2017]
flashrom: a utility for identifying, reading, writing, verifying and erasing flash chips. *flashrom.* September 5, 2017. https://www.flashrom.org/Flashrom October 18, 2017 [accessed]

[IC 2014]
IC 4216A-CS935LA1 by D-LINK CORPORATION for Wifi Cube H.264 Network Camera IC ID: 4216A-CS935LA1 / 4216ACS935LA1. Model: DCS-935LA1. Industry Canada. October 27, 2014. https://industrycanada.co/4216A-CS935LA1 October 18, 2017 [accessed]

[Indiana University 2010]
ARCHIVED: What is firmware? Knowledge Base. *University Information Technology Services (UITS).* Indiana University. March 31, 2010. https://kb.iu.edu/d/ahtw October 18, 2017 [accessed]

[InfoSec Institute 2013]
Reversing Firmware Part 1. *InfoSec Institute*. http://resources.infosecinstitute.com/reversing-firmware-part-1/ October 18, 2017 [accessed]

[Murphy 2009]
Firmware and You: A Comprehensive Guide to Updating Your Hardware. *PC World*. June 29, 209 http://www.pcworld.com/article/165867/firmware_guide.html October 18, 2017 [accessed]

[SENRIO 2017]
Home, Secure, Home? SENRIO blog. June 8, 2016 http://blog.senr.io/blog/home-secure-home October 18, 2017 [accessed]

[Where Labs 2011]
Bus Pirate: an open source hacker multi-tool that talks to electronic stuff. Dangerous Prototypes. *Where Labs, LLC.* http://dangerousprototypes.com/docs/Bus_Pirate October 18, 2017 [accessed]

Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone: 412/268.5800 | 888.201.4479
Web: www.sei.cmu.edu | www.cert.org
Email: info@sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.