Appendix A - On the Internet of Things and Vulnerability Analysis

This appendix is adapted from two CERT/CC Blog Posts [1,2].

IoT Vulnerability Discovery

In 2014 CERT performed a study of vulnerability discovery techniques for IoT systems. As we reviewed the literature, we found a number of techniques in common use. Here they are, ranked in approximately descending order of popularity in the research we surveyed:

Vulnerability Discovery Technique	Description
Reading documentation	This includes product data sheets, protocol specifications, Internet Drafts and RFCs, manufacturer documentation and specs, patents, hardware documentation, support sites, bug trackers, discussion forums, FCC filings, developer documentation, and related information.
Reverse engineering	In most cases, this consists of reverse engineering (RE) binary firmware or other software to understand its function. However, there are instances in which merely understanding a proprietary file format is sufficient to direct further analyses. Hardware RE appears in some research, but has not been as prevalent as RE of software or file formats. As security researchers develop more hardware knowledge and skills (or as individuals with those skills become security researchers) we expect the prevalence of hardware RE to increase in the security literature.
Protocol analysis	Understanding the communication protocols used by a system is vital to identifying remotely exploitable vulnerabilities. This technique can take the form of simply sniffing traffic to find mistrusted input or channels, or reverse engineering a proprietary protocol enough to build a fuzzer for it. Decoding both the syntax and semantics can be important. In wireless systems, this technique can also take the form of using a software defined radio (SDR) to perform signal analysis, which for this purpose is essentially protocol analysis at a lower level of the stack.
Modeling and simulation	Threat modeling from the attacker perspective was mentioned in a handful of papers, as was modeling and simulation of either the system or its protocols for further analysis using mathematical techniques such as game or graph theory.
Fuzzing	Generating randomized input is a common way to test how a system deals with arbitrary input. Fuzzing of network protocols is a common method cited in a number of reports.
Input or traffic generation and spoofing	Unlike fuzzing, spoofing usually consists of constructing otherwise valid input to a system to cause it to exhibit unexpected behavior. Constructing bogus input from a valid or trusted source also falls into this category.
Scanning	Because most IoT are composed of multiple components, each of which may have its own architecture and code base, it is often the case that a researcher can find known vulnerabilities in systems simply by using available vulnerability scanning tools such as Nessus or Metasploit.
Hardware hacking	This technique involves interfacing directly with the electronics at the circuit level. It is a form of physical-level reverse engineering and can include mapping circuits and connecting with JTAG to dump memory state or firmware.
Debugging	This technique uses software-based or hardware-based debuggers. JTAG is a common hardware debugging interface mentioned in many reports.
Writing code	This technique involves developing custom tools to assist with extracting, characterizing, and analyzing data to identify vulnerabilities.
Application of specialized knowledge and skills	In some cases, just knowing how a system works and approaching it with a security mindset is sufficient to find vulnerabilities. Examples include RFID and ModBus.

Many of the techniques listed above are common to vulnerability discovery in the traditional computing and mobile world. However, the low-hanging fruit appears to hang much lower in the IoT than in traditional computing. From a security perspective, even mobile systems have a head start, although they are not as far along as traditional computing platforms. The fact is that many of the vulnerabilities found thus far in IoT would be considered trivial—and rightly so—in the more mature market of servers and desktop computing. Yet the relative scale of the IoT market makes even trivial vulnerabilities potentially risky in aggregate.

IoT Vulnerability Analysis

In our review of recent security research that focused on vulnerability discovery in the Internet of Things, we identified several key differences between IoT and traditional computing and mobile platforms, including

Difference between Traditional and IoT Vulnerability Analysis	Description
Limited instrumentation	The vulnerability analyst's ability to instrument the system in order to test its security can be limited. Many of the systems comprise embedded devices that are effectively black boxes at the network level. On the surface, this limitation might appear to be beneficial to the security of the system; if it's hard to create an analysis environment, it might be difficult to find vulnerabilities in the system. However, the problem is that while a determined and/or well-resourced attacker can overcome such obstacles and get on with finding vulnerabilities, a lack of instrumentation can make it difficult even for the vendor to adequately test the security of its own products.
Less familiar system architectures	IoT architectures are often different from those most often encountered by the typical vulnerability analyst. In short, ARM is neither x86 nor IA64, and some embedded systems are neither. Although this limitation is trivially obvious at a technical level, many vulnerability researchers and analysts will have to overcome this skill gap if they are to remain effective at finding and remediating vulnerabilities in IoT.
Limited user interfaces	User interfaces on the devices themselves are extremely limited—a few LEDs, maybe some switches or buttons, and that's about it. Thus, significant effort can be required just to provide input or get the feedback needed to perform security analysis work.
Proprietary protocols	The network protocols used above the transport layer are often proprietary. Although the spread of HTTP/HTTPS continues in this space as it has in the traditional and mobile spaces, there are many extant protocols that are poorly documented or wholly undocumented. The effort required to identify and understand higher level protocols, given sometimes scant information about them, can be daunting. Techniques and tools for network protocol inference and reverse engineering can be effective tactics. However, if vendors were more open with their protocol specifications, much of the need for that effort would be obviated.
Lack of updatability	Unlike most other devices (laptops, PCs, smartphones, tablets), many IoT are either non-updateable or require significant effort to update. Systems that cannot be updated become less secure over time as new vulnerabilities are found and novel attack techniques emerge. Because vulnerabilities are often discovered long after a system has been delivered, systems that lack facilities for secure updates once deployed present a long-term risk to the networks in which they reside. This design flaw is perhaps the most significant one already found in many IoT, and if not corrected across the board, could lead to years if not decades of increasingly insecure devices acting as reservoirs of infection or as platforms for lateral movement by attackers of all types.
Lack of security tools	Security tools used for prevention, detection, analysis, and remediation in traditional computing systems have evolved and matured significantly over a period of decades. And while in many cases similar concepts apply to IoT, the practitioner will observe a distinct gap in available tools when attempting to secure or even observe such a system in detail. Packet capture and decoding, traffic analysis, reverse engineering and binary analysis, and the like are all transferable as concepts if not directly as tools, yet the tooling is far weaker when you get outside of the realm of Windows and Unix-based (including OSX) operating systems running on x86/IA64 architectures.
Vulnerability scanning tool and database bias	Vulnerability scanning tools largely look for known vulnerabilities. They, in turn, depend on vulnerability databases for their source material. However, databases of known vulnerabilities—CVE [3], the National Vulnerability Database (NVD) [4], Japan Vulnerability Notes (JVN) [5] and the CERT Vulnerability Notes Database [6] to name a few—are heavily biased by their history of tracking vulnerabilities in traditional computing systems (e.g., Windows, Linux, OSX, Unix and variants). Recent conversations with these and other vulnerability database operators indicate that the need to expand coverage into IoT is either a topic of active investigation and discussion or a work already in progress. However, we can expect the existing gap to remain for some time as these capabilities ramp up.
Inadequate threat models	Overly optimistic threat models are de rigueur among IoT. Many IoT are developed with what can only be described as naive threat models that drastically underestimate the hostility of the environments into which the system will be deployed. (Undocumented threat models are still threat models, even if they only exist in the assumptions made by the developer.) Even in cases where the developers of the main system are security-knowledgeable, they are often composing systems out of components or libraries that may not have been developed with the same degree of security consideration. This weakness is especially pernicious in power- or bandwidth-constrained systems where the goal of providing lightweight implementations supersedes the need to provide a minimum level of security. We believe this is a false economy that only defers a much larger cost when the system has been deployed, vulnerabilities are discovered, and remediation is difficult.
Third-party library vulnerabilities	We observe pervasive use of third-party libraries with neither recognition of nor adequate planning for how to fix or mitigate the vulnerabilities they inevitably contain. When a developer embeds a library into a system, that system can inherit vulnerabilities subsequently found in the incorporated code. Although this is true in the traditional computing world, it is even more concerning in contexts where many libraries wind up as binary blobs and are simply included in the firmware as such. Lacking the ability to analyze this black box code either in manual source code reviews or using most code analysis tools, vendors may find it difficult to examine the code's security.

Unprepared vendors	Often we find that IoT vendors are not prepared to receive and handle vulnerability reports from outside parties, such as the security researcher community. Many also lack the ability to perform their own vulnerability discovery within their development lifecycle. These difficulties tend to arise from one of two causes: 1. The vendor is comparatively small or new and has yet to form a product security incident response capability. 2. The vendor has deep engineering experience in its domain but has not fully incorporated the effect of network-enabling its devices into its engineering quality assurance (this is related to the inadequate threat model point above). Typically, vendors in the latter group may have very strong skills in safety engineering or regulatory compliance, yet their internet security capability is lacking. Our experience is that many IoT vendors are surprised by the vulnerability disclosure process. We frequently find ourselves having conversations that rehash two decades of vulnerability coordination and disclosure debates with vendors who appear to experience something similar to the Kübler-Ross stages of grief during the process. (The Kübler-Ross stages of grief are denial, anger, bargaining, depression, and acceptance. See http://www.ekrfoundation.org/)
Unresolved vulnerability disclosure debates	If we have learned anything in decades of CVD at the CERT/CC, it is that there is no single right answer to most vulnerability disclosure questions. However, in the traditional computing arena, most vendors and researchers have settled into a reasonable rhythm of allowing the vendor some time to fix vulnerabilities prior to publishing a vulnerability report more widely. Software as a service (SAAS) and software distributed through app stores can often fix and deploy patches to most customers quickly. On the opposite end of the spectrum, we find many IoT and embedded device vendors for whom fixing a vulnerability might require a firmware upgrade or even physical replacement of affected devices. This diversity of requirements forces vendors and researchers alike to reconsider their expectations with respect to the timing and level of detail provided in vulnerability reports based on the systems affected. Coupled with the proliferation of IoT vendors who are relative novices at internet-enabled devices and just becoming exposed to the world of vulnerability research and disclosure, the shift toward IoT can be expected to reinvigorate numerous disclosure debates as the various stakeholders work out their newfound positions.

IoT Parting Thoughts

Although vulnerability analysis for IoT has much in common with security research in traditional computing and mobile environments, there are a number of important distinctions outlined in this appendix. The threats posed by these systems given their current proliferation trajectory are concerning. Even as they become more common, it can be difficult to identify the threats posed to a network by IoT either alone or in aggregate. In the simplest sense one might think of it as a "hidden Linux" problem: How many devices can you find in your immediate vicinity containing some form of Linux? Do you know what their patch status is? Do you know how you'd deal with a critical vulnerability affecting them? Furthermore, while the hidden Linux problem isn't going away any time soon, we believe the third-party library problem will long outlast it. How many vulnerable image parsers with a network-accessible attack vector share your home with you? How would you patch them?

Dan Geer [7] puts it thus:

> *An advanced persistent threat, one that is difficult to discover, difficult to remove, and difficult to attribute, is easier in a low-end monoculture, easier in an environment where much of the computing is done by devices that are deaf and mute once installed or where those devices operate at the very bottom of the software stack, where those devices bring no relevant societal risk by their onesies and twosies, but do bring relevant societal risk at today's extant scales much less the scales coming soon.*

We agree.

< 9. Conclusion | Appendix B - Traffic Light Protocol >

References

1. A. Householder, "What's Different About Vulnerability Analysis and Discovery in Emerging Networked Systems?" 6 January 2015. [Online]. Available: https://insights.sei.cmu.edu/cert/2015/01/-whats-different-about-vulnerability-analysis-and-discovery-in-emerging-networked-systems.html. [Accessed 16 May 2017].
2. A. Householder, "Vulnerability Discovery for Emerging Networked Systems," 20 November 2014. [Online]. Available: https://insights.sei.cmu.edu/cert/2014/11/-vulnerability-discovery-for-emerging-networked-systems.html. [Accessed 16 May 2017].
3. MITRE, "Common Vulnerabilities and Exposures," [Online]. Available: https://cve.mitre.org/. [Accessed 16 May 2017].
4. National Institute of Standards and Technology, "National Vulnerability Database," [Online]. Available: https://nvd.nist.gov/. [Accessed 16 May 2017].
5. JPCERT/CC and IPA, "Japan Vulnerability Notes," [Online]. Available: https://jvn.jp/en/. [Accessed 16 May 2017].
6. CERT/CC, "Vulnerability Notes Database," [Online]. Available: https://www.kb.cert.org/vuls. [Accessed 16 May 2017].
7. D. Geer, "Security of Things," 14 May 2014. [Online]. Available: http://geer.tinho.net/geer.secot.7v14.txt. [Accessed 16 May 2017].