Retrofitting Embedded Devices to Enhance Security

by Robert Van Rooyen

August 15, 2013

Prepared By
Barr Group
20203 Goshen Road, #324
Gaithersburg, MD 20879
Website: barrgroup.com
Phone: +1 (866) 65-EMBED

Barr Group Contact: Andrew Girson, CEO
Phone: +1 (301) 802-5527
Many deployed embedded devices, such as legacy medical devices, do not have adequate security measures. Typically, security was either not considered during the requirements phase of the device’s design or was discarded later as “not worth more investment” given other important needs—like getting the product out the door or testing the market. Other products already in the field may contain a set of security features yet suffer from a “weakest link” that has been turned into an exploit by academic researchers or full-fledged attackers.

The question then becomes, “How can I retrofit my insecure device to make it secure, since it is already deployed and the hardware is for all practical purposes immutable”? The answer to this question can be as varied as the devices and industries they address since the solutions are often dependent on the available resources within the device, such as memory, hardware capabilities, and processor performance. If a secure retrofit or “patch” solution can in fact be created, overcoming the technical and logistical obstacles of broad-based distribution and adoption of the retrofit remains a huge challenge. Despite these hurdles, there are at least some ways to increase the overall level of security for devices that have already been deployed.

Ultimately, one must realize however that securing a device even when security was a requirement during the initial design is no guarantee of a secure system. The general goal should be to make it comparatively difficult to break into the device by using a series of best practices, that when combined with the appropriate platform support, will yield an embedded device that is ideally “not worth the investment” from an attacker’s perspective.

A Retrofitting Case Study

In 2011, HP (Hewlett Packard) was confronted with a large security issue that presented a significant security retrofitting problem, given the large installed base of HP products. A group of academic researchers in the Intrusion Detection Systems Lab at Columbia University lead by PhD candidate Ang Cui (partially supported by DARPA and the United States Air Force Research Laboratory) proceeded to embark on a thorough analysis of the HP-RFU (Hewlett Packard Remote Firmware Update) feature, which is a core capability of nearly all HP LaserJet printers (see
http://bit.ly/1db5rTr). Their intent was to publish the results of their work for the betterment of the computing industry by illustrating attack scenarios in a real world product. The team focused on two primary areas; exploiting vulnerabilities that allow the arbitrary execution of malware and an exhaustive search of publicly accessible printers on the Internet with these vulnerabilities. Their study was carried out over the course of two months using publicly available vendor information and a budget of less than $2000.

The initial methods used to determine vulnerabilities included static analysis of binary code, hardware de-soldering to aid in reverse engineering, and serial peripheral interface (SPI) snooping as shown in Figure 1 - Snooping SPI bus.

Since the design relied on SPI Flash for the bootloader, an analysis of the snooped code during system boot led to the discovery of the data structure and compression algorithm that was used in a firmware update image. Further inspection yielded the overall layout of the code that validated and parsed the RFU images, and through the use of IDA Pro (commercially available interactive disassembler application written by Llfaq Guifanov), the team was able to complete their understanding of the actual checksum and compression function implementations. Armed with the knowledge of how to create an update image, the team built a tool that would wrap arbitrary executable code into a RFU compatible package. At that point, they were able to
develop a VxWorks (Real-time Operating System from Wind River Systems) rootkit (form of malware that can provide host system access in addition to the original base feature set) that could be applied as an update via a print job, which is one of several convenient methods to update the firmware (other methods include an update over the network from a printer management application). This of course was a huge security hole as attackers could distribute their rootkit as a payload in any document that an unwitting user might print. The team’s rootkit image included a number of features, such as command and control by way of a secret channel, print job snooping, various forms of reconnaissance, reverse IP tunneling in order to pierce network firewalls, and literal self-destruction in the form of overheating by the toner fixer element. This last feature eventually gave rise to the “Flaming Printer Hack” media headline.

In order to discover the number of vulnerable printers on the Internet, the team created a tool that would scan the Internet for HP printers using a variety of protocols that included HTTP, telnet, SNMP, and raw TCP sockets. Detailed model and firmware version information could be extracted by issuing printer commands over print server sockets as well as SNMP “get” commands using the default “public” community name. The researchers ultimately discovered over 90,000 vulnerable printers with this approach!

The Intrusion Detection Systems Lab eventually published the full details of their research and as a result, HP created a series of security retrofit patches for the various printer models in an attempt to address the issues that were discovered by the research team. After a reasonable period of time for communication of the updates to customers, a subsequent analysis of the printer population concluded that only about 1% had the appropriate update, which implies that nearly 99% of those printers were still vulnerable to this type of attack.

In addition to update adoption hurdles, HP had to contend with the many challenges of developing a good retrofit solution in the presence of immutable hardware. Although the specifics of the workarounds were not published, you can rest assured that it was a non-trivial exercise that required significant creativity.
Threat Assessment

A retrofit security analysis will typically be composed of several vectors that include attacker motivation, a list of implications for a compromised system, and an enumeration of specific hardware/software/mechanical vulnerabilities, which is also referred to as the “attack surface”. This is really the Who, What, and Where of embedded system security.

First, there has to be some underlying motivation for an attacker to spend time and/or money breaking into the specific system. Who are these attackers and how are they motivated? Professional attackers work for money and actively build their reputation, whereas hobbyists are often intrigued by the challenge of solving a complicated puzzle. For well-funded professional attackers, the sky is really the limit in terms of time and resources to get the job done in the context of industrial espionage or geopolitical agendas. Although hobbyists may lack funding, they have nearly unlimited time to spend, which is often a key ingredient in being successful. As we have seen in the earlier case study, a separate category of attackers are made up of academic researchers, whose mission is to understand and expand knowledge in the growing area of computer security. This group is largely responsible for the dramatic improvements we have seen in embedded security and have greatly helped the industry identify and meet the challenges of ever increasing attacker sophistication. You should therefore carefully consider all of the aforementioned points when planning your security retrofit.

Secondly, what are the implications to safety, quality, company reputation, and lost revenue for your product if an attacker is successful? In the case of a medical device, the consequences can quite literally mean life and death. The first publicized case of a successful attack on a cardiac implantable device is a testament to this lethal threat (http://bit.ly/19l4Xrk). In addition to safety, the impression of product quality and your reputation can be significantly impacted, which will ultimately affect your company’s bottom line in terms of lost revenue. As an example, consider the impact to Hewlett Packard and their reputation for quality in the context of such a serious security breach of the LaserJet printer line.
Finally, assessing the hardware, software, and mechanical vulnerabilities of your embedded system should be completed by carefully enumerating all of the methods by which an attacker can interact with the system. The list would include all external physical and logical communication ports, internal system buses, control inputs such as buttons and knobs, secure boot, firmware updates, data storage, power supplies, and physical tampering.

You should pay particular attention to communication ports, as an attacker can monitor transactions by probing signals and interpreting data to find a myriad of vulnerabilities. This general approach would also include viewing data on internal system buses in order to understand data structures and firmware operations. From the printer case study, we learned that the researchers probed the SPI bus, which as it turns out was a key ingredient in disassembling and understanding the boot code and associated encryption algorithms. Your specific product threat assessment must be comprehensive for the embedded system, but should also include the management of security information such as private keys that may be handled by any number of internal personnel and 3rd party service providers, e.g., manufacturing.

Once you have completed the threat assessment, you will likely find that you will be constrained by the inherent limitations of the target platform. You should however be diligent in your initial threat analysis and defer imposing any restrictions until you reach the implementation phase. The design and implementation of the retrofit solution will be dictated by the capabilities of the hardware and its associated resources, but there are many workarounds to common challenges.

**Practical Solutions**

As we have seen with the printer, a number of methods were used to attack the integrated embedded system. Many of these methods can be successfully thwarted by using a series of practical retrofit solutions. A subset of possible countermeasures include using internal system-on-a-chip (SOC) flash memory, one time programming able (OTP) memory, secure boot, private key signed and encrypted firmware images, public/private key encrypted communications, tamper proof key storage, and intrusion
detection. It should be noted however that some of these solutions may require a hardware redesign, which could be cost prohibitive or effectively impossible to roll out to customers given the population of deployed units. This effectively limits a retrofit solution to a firmware update, which in itself will have challenges in terms of customer adoption.

If the hardware is in fact immutable, what can firmware developers do to at least raise the level of security of an existing embedded system through a retrofit? Although the options are limited, there are steps that can be taken to make it more difficult for an attacker. The first step is to determine if there are any inherent security features in the existing hardware that to date have either not been enabled or improperly utilized. Mechanisms such as OTP memory, secure boot, private key storage, and AES offload engines are standard fare in modern processors and could be leveraged with a firmware update that takes advantage of these capabilities. Software libraries with cryptographic functions also can be utilized to handle encryption and signature hashes in the presence of sufficient processing/memory resources.

Securing communication ports can be a tough retrofitting challenge as it likely implies that elements on both sides of the link require an update. If the updates are confined to multiple processing elements internal to the product, that may be significantly easier than scenarios involving external components. Encrypting communications using open standards that rely on private keys is a highly desirable solution if it will work within the confines of your retrofit environment.

A clever and yet simple retrofit mechanism that can be used for intrusion detection is periodic code validation. This can be achieved by doing a checksum or cryptographic hash of the executing code and comparing the results to a secured value, which is often referred to as a “factory measurement”. If the comparison fails, then the code has been modified and therefore compromised. The system could then be rendered inoperative or issue some type of alert through a communication channel. If the hardware does not have a method for properly securing the factory measurement through the use of tamper proof storage, then various obscurity
mechanisms could be used to disperse it in persistent storage. An example would be to store the bytes making up a 128-bit value in vastly different locations of memory with no particular pattern. Another method could involve performing a complex calculation whose result is the factory measurement. These methods are by no means foolproof, but could slow down an attack and in some cases cause the attacker to move on to more fertile ground, which could take the form of attacking another product instead of yours.

A Typical Embedded System Example

Embedded systems come in many forms and are composed of a wide variety of components, but in general they have a number of common elements. These elements include a central processing unit, memory, some form of input/output, and usually a way of updating the firmware in the field, for example via a WIFI connection as shown in Figure 2 - Typical Embedded System

![Figure 2 - Typical Embedded System](image)

These are the base elements and depending on the specific component capabilities, some set of tactical steps can be taken to increase the level of difficulty for an attacker as outlined previously.

Firmware updates should be encrypted and signed so that they can be authenticated as genuine by the embedded system. This is a critical step in securing a system and it implies some level of cryptographic capability by the central processing unit either purely in software or ideally aided by a variety of hardware resources such as an
encryption engine and tamper-proof key storage. Likewise, all accessible data in storage devices should be secured in a similar manner.

Input and output ports are easily probed by attackers and therefore merit immediate attention in terms of their inherent vulnerabilities. Based on the threat assessment, these ports may require an encryption and/or authentication protocol with respect to associated peripherals. This is especially important for both wired and wireless network connections. There are a variety of off-the-shelf solutions such as SSL (secure sockets library) that are based on open standards and can be leveraged to address these types of security challenges.

The ability to dump or probe memory is a common method that attackers use to break into a system. By using internal SOC memory or a secure and perhaps multistage boot strategy, you can greatly increase the level of difficulty for an attacker. Likewise, the use of encryption and signatures for external data storage is equally critical in securing an embedded system.

**Security Testing**

The original system designers are in a unique position to become highly effective attackers given their extensive internal knowledge of the target system. Within the span of a few project meetings and with perhaps an embedded system security expert present, a typical development team could brainstorm a plethora of ways that their system could be compromised. The list of known vulnerabilities then can be sorted by their inherent probability of being exploited by an attacker along with their potential implications to the customer and the company. Armed with this information, a series of countermeasures could be developed along with detailed test plans that can be executed by both internal and 3rd party testing service providers.

Of course, there is always a set of unknown vulnerabilities, which represents a huge challenge going forward in the product lifecycle. There are however at least two effective approaches to deal with this problem. The first method is to reactively deal
with newly discovered security issues in a timely manner by quickly characterizing them and releasing patches that adequately address the vulnerability. The patches should be regression tested to make sure they don’t expose any new security holes and properly communicated to customers. The second approach is through continual testing for conditions that exceed the current product test plan. An effective method for doing this is called “fuzzing”. There are a number of commercial tools available that can help with this effort, but the basic idea is to randomize input and push boundary conditions when interacting with the DUT (device under test) in a semi-automated way. Sending something like a varying sized string with random characters to a serial port repeatedly would be an example of fuzzing. If the system crashes or begins to degrade over time, then a potential “Denial of Service” attack has been uncovered. The team can then develop a proper countermeasure and release a patch and/or include it in the next firmware version.

Device security is a moving target that requires a sustained investment throughout the product lifecycle, which has obvious budgetary implications. The organization must internalize the cost and amortize it over the life of the product in order to avoid unexpected surprises.

**Conclusion**

Retrofitting security in an existing design that is immutable from a hardware perspective is non-trivial. This is compounded by the fact that attackers are increasingly becoming more sophisticated in their approach and attacks are evolving constantly. The case study shows how a team of researchers carefully and systematically uncovered a series of security flaws that allowed them to completely take over an HP LaserJet printer. Furthermore, they were able to gain access to other equipment over the network by exploiting the locality of the embedded system.

Completely securing an embedded system that lacks inherent security support in hardware is effectively impossible, but increasing the level of difficulty for an attacker is possible by methodically using a set of practical and proven solutions.

Embedded system security is an arms race and failing to recognize this fact could be a serious error of judgment on the part of designers, manufacturers, and customers.
Every attempt should be made to continuously increase the level of difficulty for attackers before, during, and after initial product release as lives, reputations, and revenue could be in serious jeopardy.

About the Author

Robert Van Rooyen is a senior engineer at Barr Group. He earned his Master of Science in Computer Science degree at California State University Chico, and his Bachelor of Science in Computer Engineering degree at California State University Sacramento. He has nearly three decades of experience developing electronics and embedded software for products including medical, network, and multimedia devices. Barr Group specializes in the design of reliable and secure embedded systems and provides training, consulting, and engineering services to product makers large and small. Mr. Van Rooyen can be reached by email at rvanrooyen@barrgroup.com.

References

5. S. Jalali, “Trends and Implications in Embedded Systems Development” TATA, 

Acknowledgements

The author wishes to thank Michael Barr from Barr Group for providing guidance throughout the development of this paper as well as Andrew Girson and Dan Smith from Barr Group for their invaluable feedback.