SIEMENS
Ingenuity for life

Siemens Digital Industries Software

A Guide to Minimizing Device Security Vulnerabilities

Executive summary

The security landscape is one that is constantly changing, with new exploits being publicized almost every day. This paper will discuss the processes that make this information available, the mechanisms that you should be thinking about before, during and after your product's development to minimize the impact of this changing landscape, how to minimize the likelihood of your devices being the cause of data or security breaches, and how to be prepared to swiftly respond in the likely case that some security flaw will be discovered in your devices after their release to the public.

If you are responsible for the development of a product that includes complex software and connectivity to the Internet, you have many things that you are concerned about. Is the quality there? Are the features going to meet the needs of the market and excel against your competition? After time and effort, you and your teams solve all of those problems, complete the product, release it to the market, and it is a success. Congratulations! But...

Shortly after your product is deployed, you get a midnight phone call from your company's CEO letting you know that the company's product will be on the front page of tomorrow's Wall Street Journal because hackers have figured out how to access customer sensitive data from your device. How did this happen? What do you do? How do you manage to keep your job?

Some Background

A security vulnerability is a programming error (or defect, or bug) that opens a device up to be affected by some external or internal application that was not intended. Security vulnerabilities are in every product. By following the guidelines described in this paper, you will learn how to protect your device from issues that are known, make it difficult for vulnerabilities that you do not know about to be exploited, and establish processes for known issues to be discovered and fixed.

The most important part of this process is known as Common Vulnerabilities and Exposures, or CVEs. CVEs were originally defined in 1999. The database is a repository of known exploitable security issues that exist (or existed) in products. CVEs are published and maintained in a joint effort between the MITRE Corporation and the US National Vulnerability Database (NVD), which is maintained by the US Department of Homeland Security. Every significant security vulnerability you have probably heard of has been documented as a CVE, from Heartbleed (CVE-2014-060) to Shellshock (CVE-2014-6271) to URGENT/11 (11 CVEs discovered in 2019).

These CVEs are discovered either as a result of damage caused (a postmortem of an adverse effect discovered the underlying issue), or as a result of a conscientious engineer who discovers a potential exploit. The good news is that most exploits are discovered without causing damage, the bad news is once an exploit is communicated to the world through the CVE process, it can be easily exploited by hackers worldwide, so time is of the essence. Fortunately, the CVE process is designed to give product or software package developers time to fix the exploit and have that fix available to customers before the exploit is announced to the world. While it does not always happen, the opp-ortunity is there for product developers focused on keeping their devices secure to act rapidly.

When a CVE is discovered, it is assigned a CVE identification ID. If the CVE is determined to be an issue, it is generally assigned a Vulnerability Score by the NVD. This is a number between 1 and 10; the higher the number, the more serious the vulnerability will be to devices that contain the vulnerability. The NVD also contains any other known information about the issue, as well as links to pertinent sites that further describe the issue, and the existing fixes available for it.

The main benefit of the CVE reporting process is knowledge; knowledge of the issue, of potential fixes, and of the severity and risk the issue might have to your products. As you probably know, security vulnerabilities can expose your devices, your customers, and yourself to several adverse consequences, including:

- Loss of confidential business data
- Exposure of customer or end-user data, which could lead to identity theft, HIPAA violations, and other serious consequences
- Infiltration of the device by malicious actors which can cause loss of property, injection of ransomware, etc.

Not only are these potential results bad for your customers (and their customers), the impact can be embarrassing, costly, and damaging to you and your company's reputation.

Security Issues and Product Design

When we think of how to develop devices that are as secure as possible, it helps to think about the different types of potential security issues, and the approaches to prevent them:

- Closing the Barn Door - Protecting your devices from known types of issues
- Keeping the Barn Door Shut – Protecting your devices from the future
- Make the Door Hard to Open – Apply preventative development techniques

We will look at these more closely.

Protecting your Devices from Known Types of Issues

As discussed above, many potential exploits that are used by hackers to break into devices are already known by the worldwide security community and have already been fixed. It would be unfortunate, and career limiting, if something that is already fixed shows up in your devices and is exploited by hackers. How do you prevent this from happening?

First, each CVE that is shown to be an exploit is searchable in both the NVD and CVE databases. You can search by component name, CVE ID or any keyword of interest. For example, assume your device uses some distribution of Linux. Searching for Linux vulnerabilities, you will find a number of issues; as a specific example, consider CVE-2019-11683. This is a critical severity issue that should not be in your product, since it is both well-known and allows remote denial of service attacks, or "unspecified other impacts". Looking at the entry for this defect, we see that it is resolved in Linux kernel version 5.0.13 or later, which means that you should upgrade to that version in your product if your Linux kernel is an earlier version.

There are a number of tools available to help you identify if important CVEs are included in your product, the most important of which is called cve-check (https://github.com/clearlinux/cve-check-tool). This tool generates reports that includes which packages contain CVEs that are not resolved in the versions you are using, by performing version checking like we did above in the Linux example. You can use this information to determine if any preemptive action is required before your product images are considered complete.

Device manufacturers do not want to do this kind of checking and updating as a matter of course while trying to get a product designed and developed. Most device manufacturers would rather have their engineers solve product problems instead of managing and maintaining a Linux distribution. However, there is a price for the extreme capabilities, stability and community that using open source gives to you (do you want to write an SSL layer or use the one that is used in devices all over the world?). Either the device manufacturer must take this task on, or use a commercial Linux distribution and hold their vendor responsible for doing this work for them.

Note that monitoring and making sure known exploits are addressed is not the only thing to think about at this phase. A few other concerns that, if not addressed, can cause your device to be exploited include:

- Access Control – Have you designed in the ability to define roles that can access various types of data (userlevel, management level, maintenance level, etc.), and are you certain that only authorized roles access the data? Is it difficult to access data with higher levels of access control from the internet compared with having physical access to the device? Linux provides at least 2 ways to manage access control; Discretionary (DAC), which is the standard Linux access control model, and Mandatory (MAC) which is more complex and more secure (and is part of the SELinux package).

- Encryption – Is the data stored on your device (both in memory and in storage), as well as that transmitted between your device and others, protected and encrypted so that it can only be deciphered by those meant to see it? Many potential exploits that allow outside actors to see data still would need the proper keys to decrypt it, so you need to make sure that a different mechanism must be overcome to access the keys than to simply access the encrypted memory.

- Hardware security assistance – Many features of modern processors help in ensuring the security of your devices and applications, but it is the responsibility of the system designer to take advantage of them. Features such as TrustZone, Cryptographic Acceleration, Trusted Platform Modules (TPMs), etc. are on modern microprocessors and are designed to both accelerate and assist in the development of secure designs. But, much like having a lock on your front door that you forget to lock, having these features in hardware is useless if you do not make use of them.

Protecting your Devices from the Future

You have released your product, and you know that at the time it is released that it is not subject to known exploits, and you have done everything to protect the data on your device. Congratulations, you have already done more than some manufacturers to make your device secure. However, once the product is released, your work is not complete, just because you have protected yourself against all known exploits. The number of known exploits increases every day; in 2019, there were 12,174 CVEs created, which is actually less than there were in 2018. That is over 30 per day. Of course, most of these do not turn out to be issues, and, of the ones that are issues, many will not apply to your device (many CVEs are reported against versions of open source components older than what you might deploy, or will be against components you are not using). That said, even against the Linux kernel, there were 170 CVEs issued in 2019, and some of them will lead to exploits against your device.

While there is no way to prevent this from happening, you need to know that it WILL happen, and you need to make sure your device is prepared. The time to prepare your device for the future is during development, so that you can prepare the device to be updated as new exploits (and significant product defects) are found and fixed.

As an example, in 2016, an exploit commonly known as the Mirai botnet caused extensive internet outages by taking over small IoT devices such as webcams and routers and used them to execute Distributed Denial of Service attacks (DDoS attacks) against both US and French web infrastructure providers, as well as several more targeted attacks including ones against Rutgers University and a popular cybersecurity focused website. Most of the owners of these infected devices are unaware that their systems are infected, and Mirai (and derivatives of it) is still a threat today. There are even devices made today that are susceptible to it, even though the underlying cause, which was as simple as trying to access a root level access account with 64 well known default login/passwords such as user/user, or user/password. As most users of these devices were unaware or unable to change these simple defaults, the Mirai botnet was able to take control of these systems.

The considerations that should be made during development to future-proof your device are many, but the most important is the ability to securely update your system. The methods and facilities to support this are many and complex and outside the scope of this paper.

Applying Preventative Development Techniques

If you are using Linux and other open source software as part of your product's design, then there will be vulnerabilities in your device that you don't know about when you release your device. As a result, you do not only want to eliminate as many known vulnerabilities as possible at the time you release, but you must also assume that, at some point, some bad actor will be able to get unauthorized access to the device through as-yet unknown vulnerabilities in the common open-source packages you are using.

When that happens, you will not be able to prevent unauthorized access to the device until you fix the underlying security flaw; consider an example where you are building a consumer-grade device that uses a customer's Amazon username and password to configure. If this is already compromised, then it may be possible for an external entity to compromise the device, insert malware, etc. Once the malware is running on the device, it would be possible to attempt to probe and attack the applications you have included on the device.

There is no perfect defense from determined hackers armed with the knowledge of exploits in your device, but you want to make it as difficult as possible for them to do so. You cannot protect yourself from vulnerabilities in the open source modules you are using, but you can limit potential vulnerabilities in your own applications. Of course, you should consider the techniques mentioned above to design in greater protection, but what about the way your applications are developed?

As mentioned above, most exploits in open source and application software are due to repeated developmental flaws that happen over and over again. Things like NULL pointer de-references, freeing already freed memory, overflowing a fixed length buffer, etc. are the kinds of coding errors that can be easily exploited by hackers to compromise your devices. However, there are several approaches that can be used to help. Specific techniques are beyond the scope of this paper, but places to start looking are:

- Static (and dynamic) analysis. The first static analysis you are likely to see are the warnings that come from your compiler. It is surprising how many organizations ignore this valuable diagnostic tool in a misguided rush to get something released. Beyond that, the open source community provides several useful static analysis tools such cppcheck and clang, and there are many commercial solutions available. All will detect issues that are easily missed in code reviews and, as long as the reports from these tools are managed, you can prevent several major classes of potential exploits in your applications.

- Use of a coding standard. In general, the MISRA C and C++ coding standards (https://misra.org.uk/) are the gold standard for these, and provides many well thought-through recommendations for securing your applications. While its genesis is from the automotive industry, there is nothing automotive specific about it, and can (and should) be considered by any device manufacturer looking to secure their applications. Note that most static analysis tools also greatly ease the checking of applications against MISRA rules. While there are other coding standards available, MISRA combines common sense with good practice in a way that can be implemented by organizations of all sizes.

- Another useful coding standard comes from the Software Engineering Institute at Carnegie Mellon; known as SEI CERT C (https://wiki.sei.cmu.edu/confluence/display/seccode). There is significant overlap between this and MISRA, but the SEI standards extend beyond C and C++ and into Android, Java and Perl.

There are many other useful sources of information to consider when developing secure software, but if you are not already employing the above techniques, start there and consider expanding your thinking once you have smart coding standard and static analysis paradigm in place.

Product Testing for Known Security Issues

Given all of the above, how do you test for this large number of issues that the community knows about at any given time? There are several things to think about that can help protect you from the known issues that might stil be in your product:

- As mentioned above, most CVEs have a root cause analysis that generally maps to issues that can be identified with static analysis (also mentioned above). Besides running this analysis on the software that your teams write, you should also run it on any third-party software that you acquire (either open-source or proprietary) and see what it tells you. This is said with a word of caution; most foundational software (such as Linux or a proprietary RTOS) will perform activities that will be flagged by static analysis, such as significant pointer arithmetic and accessing data structures using offsets generated by this arithmetic. This is likely to be acceptable; it is not possible to write a highly-performant operating system without doing these activities, but looking at the results will also alert you of potential issues you should ask your vendor about if it is not clear.
- If you are using Linux, use a Linux vulnerability scanner. There are many such scanners on the market, both open-source and proprietary, and several of the most popular embedded Linux distributions (such as Yocto, which is the foundation of Siemens Embedded Linux FlexOS) include scanners that check your package versions against the National Vulnerability Database, and will inform you if there are open issues and if they have been fixed. You should run such scanners during development, while you can still update the module versions in your device, and as part of your maintenance process, so that you can determine what must be updated when you release your products.
- Penetration or Fuzz testing can also be used to identify already known vulnerabilities, but since these are your main protection against the future, we will discuss them in detail, below

Product Testing for Unknown Security Issues

As has been said previously, many of the challenges in securing devices is protecting your device from vulnerabilities that were unknown when the product was released. Several of the techniques mentioned above (structured code analysis, reviews, etc.) will make your device harder to exploit in the field, but you can try to duplicate the techniques used by hackers before your product is released. There are two major methods to do this:

Penetration testing – When we think of devices being exploited, we think about them being exploited by the "bad guys" (criminals, governmental actors, industrial espionage, etc.) for nefarious means. However, the techniques that these "black hat hackers" use are not rocket science; they are well known in the community. These techniques, along with ingenuity possessed by all software developers are how these exploits are found. There is nothing stopping an organization from doing the same on their own devices; having engineers or "white hat hacker" consultants attempt to exploit your devices before they are released to the market. This process is referred to as penetration testing; where you allow a cybersecurity attack against your devices but, instead of the results being used against you, the results are reported to you while you can do something to prevent those exploits in the field.

Fuzz testing – One of the kinds of analysis that hackers perform is to probe a device with a large amount of valid and invalid Ethernet packets that they can control and see what happens to the device. Some of those sequences of packets will duplicate known exploits, but if you have been following the guidance in this paper, you are protected from those. However, many of these sequences will be more random such as malformed or semi-malformed packets, just to see what happens. This kind of testing is known as fuzz testing, and while not as effective as penetration testing, fuzz testing is much easier to implement as part of the standard testing of your product. Additionally, there are several products on the market that performed fuzz testing and can be integrated into an automated testing framework.

Generally, penetration testing can only be performed once or twice during a product's development and testing, while fuzz testing can be performed periodically (for example, once a week), and as part of any formal testing regimen.

Product Maintenance in a Constantly Changing Security Landscape

As mentioned above, you must consider product maintenance during its development, so that the product may be updated safely and securely when the inevitable issues arise. These issues can be product upgrades, resolutions to product functionality issues, or resolutions to newly found exploits regardless of the source (additional penetration testing, new CVEs, or from other means).

You should establish a regular update frequency for your released products (quarterly, twice a year, whatever makes sense for your teams and your customers). Having a regular update frequency allows your customers to schedule predictable and minimal downtimes. Since many devices are mission critical for your customers, striking a balance between update frequency and downtime is a business decision between you and your customers, but it needs to be more often than "never."

What Should You be Asking Your Operating System Provider?

Using an operating system (OS) provider such as Siemens Embedded can be a great benefit to your product development. There are several benefits:

- The OS provider focuses on the OS as a product unto itself. As a result, they will develop, test and release the product to a degree that is beyo-nd what your teams can do themselves. You should ask your OS provider how they test the OS and related products and packages, including board support packages, drivers and other soft-ware necessary to deploy the OS on your target.

- Since the OS provider is an expert on the OS side, they can provide services and support to you that will accelerate your product develop-ment, including the security of your device. Ask them about their experience in working with customers to secure their devices

- The OS provider will maintain their products, providing regular updates to their customers. Ask your OS provider about how often they up-date their products, and what kinds of improve-ments should you expect to see in those up-dates. At the minimum they should include updates for newly found issues, including CVEs.

- The OS provider should be strongly focused on security vulnerabilities, including CVEs. Besides the regular updates mentioned above, they should be able to help you manage the constant CVE load that we have talked about in this paper. Your OS vendor should be able to provide grea-ter service around these issues, which is espe-cially important if your device has to undergo regulatory approval (such as in medical or auto-motive devices)

Conclusions

At the beginning of this paper, we described a scenario, where a newly released product has been successfully infiltrated by hackers at great expense to your company, both in terms of reputation and dollars. By following the guidance in this paper, your product will be:

- More difficult to successfully exploit
- Protected against known and unknown exploits when released
- Faster to update to close any newly found exploits
- More secure, giving your customers confidence that they are protected EVEN IF something goes wrong

The last point is especially important. Customers are aware that there is no device that is completely free of bugs. What they want to know is how you are minimizing defects and their impact, and how ready are you when something inevitably goes wrong. The methods in this paper will not prevent all potential future security issues, but they will put you in a good position to quickly resolve those issues when they arise.

So, when your CEO calls you telling you that your product issues will be publicized, you can tell him that not only have you done everything possible to limit the impact, but that the issue is already well on its way to closure. This will not only save your job but could get you a bonus as well.

Author's biography

Robert Bates is Siemens Embedded's chief safety officer responsible for the safety, quality and security aspects of Siemens Embedded's product portfolio targeting the medical, industrial, automotive, and aerospace markets. In his role, Rob works closely with customers and certification agencies to facilitate the safety certification of devices to IEC 61508, IEC 62304, ISO 26262 and other safety certifications. Before moving to Siemens, Robert was a software development director at Wind River, where he was responsible for commercial and safety certified operating system offerings, as well as both secure and commercial hypervisors. Robert has over 30 years of experience in the embedded software field, most of which has been spent developing operating system and middleware components for device makers around the world.

Siemens Digital Industries Software

Headquarters
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

Americas
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

Europe
Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific
Unit 901-902, 9/F
Tower B, Manulife Financial Centre
223-231 Wai Yip Street, Kwun Tong
Kowloon, Hong Kong
+852 2230 3333

About Siemens Digital Industries Software

Siemens Digital Industries Software is driving transformation to enable a digital enterprise where engineering, manufacturing and electronics design meet tomorrow. Xcelerator, the comprehensive and integrated portfolio of software and services from Siemens Digital Industries Software, helps companies of all sizes create and leverage a comprehensive digital twin that provides organizations with new insights, opportunities and levels of automation to drive innovation. For more information on Siemens Digital Industries Software products and services, visit siemens.com/embedded or follow us on LinkedIn, Twitter, Facebook and Instagram.

Siemens Digital Industries Software – Where today meets tomorrow.

siemens.com/embedded

MGC 02-21 TECH14500-wp 5/21