Lecture Notes

Chapter 10:
Conclusion - Miscellaneous Security Issues and the Future of Embedded Applications Security

CYENG 351: Embedded Secure Networking

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)

Chapter Overview and Programming Languages and Security

- This chapter covers a few of the more esoteric aspects of security, including the impact of programming language choice on security, dealing with and recognizing attacks, and the future of security, including quantum cryptography and legal issues.
- Choosing a hardware platform and designing security features are only part of the work in developing a secure application.
- Development tools and languages factor into the security.
- Some languages, such as Java, provide some built-in safeguards that help with overall security, such as strict typing.
- The features of different languages and compilers are something to look at when choosing the software for your system.
- The predominant language for most embedded applications is some variant of the C language.
- Unfortunately, C suffers from some inherent problems that detract from overall security, notably weak typing and problems with buffer overflow.
- C is such a popular language for embedded developers because of its close ties to the underlying hardware, but this can detract from the security of an application because it lets programmers do things they should not.
- The C++ language, being a descendant of C, also suffers from similar shortfalls.
- C++, however, has better typing (though not perfect) and a good object-orient design approach can help with the buffer overflow and pointer issues found in C.

Programming Languages and Security (Cont.)

- In any case, using C or C++ makes security a difficult job, but there often isn't a choice.
- Applying good engineering principles to your design is pretty much the only way to avoid the security pitfalls in C and C-like languages.
- Fortunately, however, almost every security protocol or algorithm that you may want to use will inevitably have a reference implementation in C.
- An alternative to C that has been gaining in popularity is the Java language.
- Java is good from a security perspective because it enforces fairly strict typing and virtually eliminates some of the buffer overflow problems seen in C.
- Java exception handling is also quite nice for handling errors that represent possible attacks.
- The problem, though, is that unless you have a CPU equipped to run Java Bytecode directly, you have to rely on a Java Virtual Machine (JVM), which is most likely written in C.
- If you can verify that your JVM is secure, then it is easier to verify your application.
- Another problem with using a JVM, besides the security issues, is performance.
- Since Java is a partially interpreted language, it suffers tremendous performance penalties for certain applications.
- For this reason, Java is typically only available on higher-performance platforms.

Programming Languages and Security (Cont.)

- Other than C (and various flavors of C) and Java, the language that may be most familiar to the reader is BASIC, available for some hobbyist-targeted hardware, such as the Parallax
- BASIC Stamp. BASIC does not have the same backing as C and Java, and it hides a lot of functionality (such as typing) from the software engineer.
- It should be possible to implement various cryptographic algorithms and even full security protocols in BASIC, but it is likely that there isn't the wealth of reference code available that can be found for languages like C and Java.
- You should also look at the development tools themselves.
- If your compiler generates incorrect code, you may introduce a security hole without knowing it.
- If you are using inexpensive development tools, there is a greater possibility of incorrect code generation, but compilers are complex applications and even world-class tools can still suffer from problems.
- Inspect generated code in critical sections of your applications and be sure to keep up with the latest bug reports for the tools you use.
- There are numerous languages that provide software support for different hardware platforms, but other than the languages mentioned here, they are likely aimed at niche markets or are specific to a particular vendor.
- When using a proprietary language to develop a secure embedded application, be sure that you understand what the language and the development tools are doing.
- The better you understand the languages and tools that you use, the less likely you are to make mistakes that will lead to security breaches later.

Dealing with Attacks

- Any system connected to a network (the Internet or proprietary) will be subject to attacks - both intentional (i.e., malicious hackers) and inadvertently (i.e., heavy network traffic leading to Denial of Service.
- The first line of defense is a good security policy and verification or assurance of the security of the implementation of that policy.
- If you can be relatively certain that you application and policy are sound, then you have likely prevented the lion's share of future attacks.
- No matter how well you implement your application there will likely be a problem at some point that will allow an attacker to gain an advantage.
- How do we recognize an attack? How do we deal with an attack that is happening?
- Recognizing that an attack has occurred or is occurring can be extremely difficult.
- Eavesdropping or man-in-the-middle attacks are impossible to detect, since they happen outside the system that is under your control.
- However, protocols like SSL will let you know when a possible attack is occurring (in the case of SSL, the attack notification is called an alert).
- Your application must treat every possible attack as the real deal, or you will eventually be attacked without knowing it (this is sort of a variant of the boy who cried wolf).
- Attacks that exploit your own system are a little easier to recognize, especially after the fact.
- If your application is behaving erratically, there is the possibility it was attacked, but remember that an attack and a defect can have the same symptoms, so don't jump to any conclusions.
- How you deal with an attack that has already occurred depends largely on what your application does.

Dealing with Attacks (Cont.)

- When designing your system, pay close attention to the actions you don't want to happen.
- Treat each of those actions as the result of an attack, and working backwards, identify as many paths to those end results as you can.
- This exercise will help greatly in developing a more secure application.
- Handling a current attack is a different matter.
- Even if you have high confidence that your application is safe, there may still (and likely do) exist vulnerabilities that can be exploited.
- For this reason, a mechanism for intrusion detection may be desirable.
- You should identify any paths that lead to an action that is undesirable.
- If you cannot prevent an action from occurring, such as a Denial-of-Service condition, you may want to consider adding some logic to detect the early stages of an attack and react accordingly.
- However, designing an intrusion-detection system (or IDS) is trickier than it sounds. IDS development is a big business and a difficult problem to solve, even on machines with the horsepower to run complex programs that can identify various intrusions.
- An IDS can actually harm your application as well, if implemented improperly.

Dealing with Attacks (Cont.)

- There are many different ways to handle the aftermath of an attack, and nearly all of them are application-specific.
- Basically, recovering from a successful attack will generally amount to damage control.
- We won't consider the business or human factors in here, but there are some things you can do for your system.
  - First and foremost, you will want to update other systems to prevent repeat attacks as soon as possible.
  - Using redundancy (if one device is compromised, protect a redundant copy) can help to heal a broken system, and frequent backups help in recovering lost data or application state.
  - Generally speaking, however, your application should be designed to prevent attacks, not recover from successful ones.
  - Doing so would be the equivalent of preparing to deal with being mugged walking down the street, rather than taking measures to prevent being mugged in the first place.

The Future of Security

- Truly the only way to provide any level of security for an application is a constant state of vigilance and attention to developments within the security community.
- With prevention being the best solution, knowledge is your best weapon.
- The Internet is a fantastic resource, since information on many attacks and security flaws is almost immediately accessible. → Open Source Availability.
- With AES reference implementations easily downloaded, anyone with a connection to the Internet can access strong encryption to protect their communications.
- This presents a paradoxical challenge - without security mechanisms, we cannot keep our own data and communications safe, but if we make it publicly available, then our enemies can use it against us.
- Elliptic Curve Cryptography (ECC) Algorithms derive their security from the mathematic principles of (you guessed it) elliptic curves. → They need more explorations and developments.
- One of the most exciting (and potentially frightening) areas of interest for security researchers today is quantum computing.
- Utilizing quantum mechanics as a computation mechanism, quantum computers will theoretically make almost all existing forms of cryptography obsolete overnight.
- For now, a full quantum computer is purely theoretical, but if implemented, it should be able to factor large numbers in a fraction of the execution time any conventional hardware could ever dream of achieving.
- Quantum Computing → Usage for creating new attacks and defenses.
- Despite its useful positive and negative properties and the promise of a new order in computer security, quantum cryptography is still a very new concept and technology that utilizes quantum mechanics is extremely expensive.
- Security will need to be an integral part of all electronic devices and applications unless human nature is changed and we all start being nice to one another.

Assignment

- **Reading Assignment:**
  - Stapko, T., 2011. **Practical embedded security: building secure resource-constrained systems**. Elsevier.
    - ✓ "Chapter 10: Conclusion - Miscellaneous Security Issues and the Future of Embedded Applications Security", Pages 179-185.

GANNON UNIVERSITY

Questions?