BAI3110 Security Introduction

As the Internet grows in importance, applications are becoming highly interconnected. Prior to the Internet computers were mainly stand alone with little, if any, interconnectivity. In those days it didn't matter if your application was insecure, the worst you could do was attack yourself, and as long as application performed its tasks successfully, most people didn't care about security.

Times have changed. Now virtually all computers, consisting of, servers, desktop pc's cell phones, and pocket sized devices and imbedded systems are interconnected. Although this creates incredible opportunities it also means that these interconnected systems can be attacked.

The internet is a hostile environment, so you must design all code to withstand attack. Secure systems are quality systems. Code designed and built with security as a prime feature is of a much higher quality (and value) than code written with security as an afterthought. Secure products are also more immune to media criticism, more attractive to users, and less expensive to fix and support.

If you setup a server on the Internet today, in a matter of days it is discovered, probed, and attacked. I have setup computers on a Friday, and come back to school on a Monday finding the machine compromised.

Computers setup to do nothing but be exposed to attacks are called ***honeypots.*** Too learn more about this topic visit http://project.honeynet.org/

Some attackers are very highly skilled and very clever. They have deep computer knowledge and ample time on their hands. They have the time and energy to probe and analyze computer applications for vulnerabilities. Attackers doing this in an ethical manner are known as *white-hats*, or good guys. The best white-hats work closely with the software vendors to discover and remedy serious security issues prior to the vendor issuing a security bulletin or patch.

Most attackers are simply foolish vandals and are known as *script kiddies*. They have little knowledge of security and can attack insecure systems only by using scripts written by more knowledgeable attackers who have posted their exploits on bulletin boards or websites. Script kiddies can be dangerous though, if just through their sheer numbers. One smart hacker can find an exploit and tens of thousands of script kiddies can take advantage of it.

Anatomy of an Attack

A hacker:

1. Pings an existing server to get its IP address.
2. Runs a port scanner (such as Nmap or sitescan) looking for new systems that have open ports. A port scanner allows them to search a block of IP addresses. It's common to search within a block of known addresses found by the previous ping command.
3. If the hacker finds port 80 open, they can issue an HTTP HEAD request to read back the servers Operating System information. And determine if it has been patched.

4 If it is unpatched, they can then run an exploit to compromise the machine or publish their findings on a website such as http://www.slashdot.org

5. The system will now be flooded with IP-level attacks.

Security Vulnerabilities are Expensive to Fix

Like all engineering changes, security fixes are expensive to make late in the developmental process. The price of making a fix includes:

- The cost of the fix coordination, someone has to create a plan to get the fix created.
- Developer cost to find the vulnerable code.
- Developer cost to fix the vulnerable code.
- The cost of testers testing the fix.
- The cost of testing the setup of the fix.
- The cost of creating and testing international version.
- The cost to post the fix to your Web site.
- The cost of writing support documentation.
- The cost of handling bad public relations.
- Bandwidth and download costs to distribute your fix

The potential cost of making one security fix could easily in the tens, if not hundreds, of thousands of dollars. If only you had security in mind when you designed and built the product in the first place!

> The Microsoft Security Response Center believes a security bug that requires a security bulletin costs in the neighbourhood of $100,000.

Nominate a Security Evangelist

Having one or more people to evangelize the security cause, people who understand and appreciate the importance of security, works well. These people will be the focal point for all security-related issues. Their main goals are:

- Stay abreast of security issues in the industry.
- Interview people to build a competent security team.
- Provide security education to the rest of the development team.
- Hand out awards or incentives for the most secure code or the best fix of a security bug. Examples include cash, time off, a close parking spot, whatever it takes!
- Provide security bug triaging to determine the severity of bugs and offer advice on how to fix them.

Be aware of Security Issues

Two of the best resources of up-to-date information are NTBugTraq and BugTraq. NTBugTraq discusses Windows security specifically and BugTraq is more general. You can sign up at http://www.ntbugtraq.com. Bugtraq, the most well-known of the security vulnerability and disclosure mailing lists, is maintained by SecurityFocus, which is now owned by Symantec Corporation. On average you will see about 20 postings per day. It should be part of the everyday routine for a security guru to see what's going on in the security world by reading postings from both of these lists.

If you're really serious, you should also consider some of the other SecurityFocus offerings, such as Vuln-Dev, Pen-Test, and SecProg.

Build a Security Team

In many larger organizations you will find that your security experts will be quickly overrun with work. It's important that security work scales out so that people are responsible for the security of the feature that they are creating. To do this, you must hire people who not only are good at what they do, but also take pride and enjoy building a secure, quality product.

When interviewing people for security positions look for a number of different qualities:

- A love for the subject, "having the fire in your belly".
- A deep and broad range of security knowledge. Cryptography, authentication, authorization, vulnerabilities, prevention, accountability, and much more.
- An intense desire to build secure software that fulfills real personal and business requirements.
- The ability to apply security theory in novel yet appropriate ways to mitigate security threats.
- The ability to define realistic solutions, not just problems. Anyone can come up with a list of problems, that's the easy part!
- The ability to think like an Attacker
- Often, the ability to act like an attacker. Yes, to prevent the attacks, you really need to be able to do the same things that an attacker does.

The primary trait of a security person is a love for security. Good security people love to see IT systems and networks' meeting the needs of the business without putting the business at more risk than the business is willing to take on. The best security people live and breathe the subject, and people usually do their best if they love what they do. (Pardon my mantra: if people don't love what they do, they should move on to something they do love.)

Another important trait is experience. People that have had to make security fixes in the wild will understand the pain and anguish involved when things go awry and will implant that concern in the rest of the company.

If you find someone with these traits, hire that person.

Providing Ongoing Security Education

A key job function for the Security Evangelist is to make sure that your fellow employees stay attuned to their security education. For example, the Secure Windows Initiative team at Microsoft employs a number of methods to accomplish this, including the following:

- Create an intranet site that provides a focal point for security material. This should be the site people go to if they have any security questions.
- Provide white papers outlining security best practices. As you discover vulnerabilities in the way your company develops software, you should crate documentation about how these issues can be stamped out.
- Perform daylong security bug-bashes. Start the day with some security education, and then have the team review their own product code, designs, test plans, and documentation for security issues. Bug hunting is like homework, it strengthens the knowledge they learned during the morning. Finding bugs is the icing on the cake.
- Each week send an email to the team outlining a security bug and asking people to find the problem. Provide a link in the email to your website with the solution, details about how the bug

could have been prevented, and tools or material that could have been used to find the issue ahead of time. This approach helps to keep people aware of security issues each week.
- Provide security consulting to teams across the company. Review designs, code, and test plans.

Provide Bug Triaging

There are times you have to decide whether a bug will be fixed. Sometimes you will come across a bug that rarely manifests itself, that has low impact, and that is very hard to fix. You might opt not to remedy this bug but rather document the limitation. However, you will also come across serious bugs that should be fixed. It's up to you to determine the best way to remedy the bug and the priority of the bug fix.

The Attackers Advantage and the Defender's Dilemma

We've outlined the requirement to build more secure applications and looked at some simple ways to help build a more secure culture. However, we should not overlook the fact that as software developers we are always on the back foot. We, as the defenders must build better quality systems because the attacker almost certainly has the advantage.

Once software is installed on a computer, especially an Internet-facing system, it is in a state of defense. The code is open to potential attack 24 hours a day and 7 days a week from any corner of the globe.

Let's look at some of the reasons why the attackers can have fun at the defender's expense.

Principle #1: The defender must defend all points; the attacker can choose the weakest point.

Imagine you are the lord of a castle. You have many defenses at your disposal: archers on the battlements, a deep moat full of stagnant water, a drawbridge, and 5 foot thick walls of stone. As the defender, you must have guards constantly patrolling the castle walls, you must keep the drawbridge up most of the time and guard the gate when the drawbridge is down, and you must make sure the archers are well trained and well armed. You must be prepared to fight fires started by flaming arrows, and you must also make sure the castle is well stocked with supplies in case of a siege.

The attacker, on the other hand, need only spy on the castle to look for one weak point, one point that is not well defended.

The same applies to software. The attacker can take your software look for just one weak point, while we, the defenders, need to make sure that all entry points into the code are well protected. Of course, if a feature is not there, that is, not installed, then it cannot be attacked

Principle #2: The defender can only defend against known attacks; the attacker can probe for unknown vulnerabilities.

Now imagine that the castle you defend includes a well that is fed by an underground river. Have you considered that an attacker could attack the castle by accessing the underground river and climbing up the well? Remember the original Trojan horse? The residents of Troy did not consider a gift from the Greeks as a point of attack, and many Trojan lives were lost.

Software can be shipped with defenses only for pre-theorized or pre-understood points of attack. The attacker will spend a lot of time looking for many permutations of input and altering data in hundreds of ways to try to bypass the system.

The only way to defend against unknown attacks is to disenable features unless expressly required by the user. In the case of the Greeks, the Trojan horse would have been a nonevent if there was no way to get the “gift” into the city walls.

Principle #3: The defender must be constantly vigilant; the attacker can strike at will.

The defender’s guard must always be up. The attacker’s life on the other hand is much easier. They can remain unnoticed and attack whenever they like. The attacker might wait for just the right moment before attacking, while the defender must consider every moment as one in which an attack might occur.

This might be a problem for sysadmins, who must always monitor their systems, review log files, and look for and defend against attack. Hence, software developers must provide software that can constantly defend against attack and monitoring tools to aid the user in determining whether the system is under attack.

Principle #4: The defender must play by the rules; the attacker can play dirty.

This is not always true on the world of software, but it’s more true than false. The defender has various well understood white-hat tools (for example: firewalls, intrusion detection systems, audit logs, and honeypots) to protect their system.

The attacker can use any intrusive tool or information they can find to determine the weaknesses of the system, including tools and information created **after** your application goes live.

We Have Met the Enemy and He is Us

Risks don't always come from outside attackers, those that would seek to do harm may come from within. A CSI/FBI Computer Crime and Security Survey revealed that over 75 percent of companies cited disgruntled employees as a likely source of hacking attacks, and 45 percent of businesses reported unauthorized access by insiders. A study from a threat management and risk assessment firm discovered that the majority of security and human resources executives say that workplace violence is a bigger problem now than it was two years ago, and 23 percent of those surveyed said that employees had intentionally downloaded viruses over the past 12 months. Outsourcers too are facing similar threats, particularly as companies shift towards a lean operating model and jobs shift to outsourcing agencies. Eighty-eight percent of outsourcers surveyed said employee backlash is their primary concern.

A study conducted by the U.S. Secret Service and CERT Coordination Center/SEI highlighted the prevalence of insider threats in the banking and finance sectors. In one example cited, an employee of an international financial services company who had quit over a dispute concerning his annual bonus planted a "logic bomb," which deleted ten billion files in the company's network. In fact, insiders may well pose a greater threat than unknown attackers, because of their often very detailed knowledge of the company systems.

Many of the "insider" cases cited by the Secret Service study actually involved little technical skill, and in fact most cases involved insiders who were not employed in technical positions. Several of the insider attacks were able to be executed simply because of poor password and account management practices, and in one case, a company assigned default employee passwords that everyone knew to be the employee's office number.

Anyone inside the company can be a data thief, and the Secret Service study indicated that perpetrators did not share a common profile. Many insiders who had perpetrated attacks did not hold technical positions, and had no history of making any type of hacking attacks. Most had not been perceived as "problem" employees. Insiders ranged from 18 to 59 years of age, 42 percent were female, and they came from a variety of racial and ethnic backgrounds and family situations. Inside attackers with a known history of electronic abuses accounted for only nine percent of the incidents.

Incidents from within can also include an employee purposefully attempting to introduce spyware or other malware to the network via the Internet. Filtering can help reduce the risk of such inside attacks by blocking access to sites that host malware, thereby preventing intentional downloads of viruses and other harmful software to infect the network. Strong internet filtering is indispensable as part of a multi-tiered strategy to stop this and other threats from occurring.

Summary

As you can see, the world of the defender is not a pleasant one. As defenders we must build applications and solutions that are constantly vigilant, but the attackers always have the upper hand and insecure software will be quickly defeated. In short, we have to work smarter to defeat the attackers.

We will never be able to “defeat” internet vandals, simply because there are so many vandals and so many servers to attack. Nevertheless, we can raise the bar substantially, to a point where the attackers will find software more difficult to attack and use their skills for other purposes (or look for softer targets).

Finally, be aware that security is different from other aspects of computers. Few, if any, outsiders are actively looking for scalability or internationalization issues in your software. However, plenty of people are willing to spend time, money, and sweat looking for security vulnerabilities. The Internet is an incredibly complex and hostile environment, and your applications must survive there.

Site	URL
Microsoft Security Response Center Blog	http://blogs.technet.com/msrc
Hackers Hangout	http://slashdot.org/
System Administration, Networking and Security Institute	http://sans.org/
Honeynet Project	http://project.honeynet.org
BTBugTraq	http://www.ntbugtraq.com
BugTraq	http://www.securityfocus.com

Security Principles to Live By

Application security must be designed and built into your solutions from the start, and in this section we will focus on how to accomplish this goal by covering tried and tested security principles you should adopt. This includes design issues that should be addressed primarily by designers, architects and program managers. Let's look at some high level concepts:

SD³: Secure by Design, by Default, and in Deployment

Secure by default, design, and deployment can help long term and short term security goals and help shape the development process to deliver secure systems.

Secure by Design

If a system is secure by design it means you have taken appropriate steps to make sure the overall design of the product is sound from the outset. The steps you need to take to achieve this include the following:

- Assign a go-to person for security issues. This is the person who signs off on the product being secure. They get the big bucks for doing so. They are not a scapegoat, but someone who can sit at a meeting and say whether the product is secure enough to ship and, if its not, what needs to be done to rectify the situation.
- Make sure Threat Models are in place by the time the design phase is complete. We will discuss threat Models later on in the course.
- Adhere to design and coding guidelines. Again, this will be discussed in more detail later on in the course.
- Fix all bugs that deviate from the guidelines as soon as possible. Remember, that attackers don't care if the code is old or new. If the code has a flaw, it is flawed, regardless of age.
- Make sure the guidelines evolve. Update them as you learn new vulnerabilities and learn new best practices for mitigating them.
- Develop regression tests for all previously fixed vulnerabilities. This is an example of learning from past mistakes. When a security flaw is discovered, distill the attack code to its simplest form and go look for the other related bugs in other parts of your code.
- Simplify the code, and simplify our security model. This is hard to do, especially if you have a large software base. However, you should have plans in place to simplify old code by shedding unused and insecure features over time. Code tends to be more chaotic and harder to maintain over time. Code degeneration is often called *code rot*.
- Perform penetration analysis before you ship. Have people try to break the application. Install test servers, and invite the team and external entities to break it. From my experience, unless the penetration team does nothing other than penetrations and are experts in their field, penetration testing will yield marginal results at best.

Secure by Default

The goal of secure by default is to ship a product that is secure enough out of the box. Some ways to achieve this include:

- Do not install all features and capabilities by default. Apply only those features used by most of your users, and provide an easy mechanism to enable other features.
- Allow **least privilege** in your application; don't require your code be used by members of the administrators group when it does not require such elevated capabilities.
- Apply appropriate protection for resources. Sensitive data and critical resources should be protected from attack.

Secure in Deployment

Secure in deployment means the system is maintainable once your users install the product. You might create a very well designed and written application, but if it's hard to deploy and administer, it might be hard to keep the application secure as new threats arise. You should follow a few simple guidelines:

- Make sure the application offers a way to administer its security functionality. Without knowing the security settings and configuration of the application, the administrator cannot know whether the application is secure.
- Create good quality security patches as soon as feasible. Fix it as soon as possible, but take enough time to make sure you are not introducing more errors, make sure you perform regression testing on the fix.
- Provide information to the user so they can understand how to use the system in a secure manner. This could be through online help, documentation, or cues online.

Security Principles

The rest of this chapter builds on the SD^3 principles. We will spend the rest of the course talking about these principles in more detail, this is just an overview:

- Learn from mistakes
- Minimize your attack surface
- Use defense in depth
- Use least privilege
- Employ secure defaults
- Backward compatibility will always give you grief
- Assume external systems are insecure
- Plan on failure
- Fail to a secure mode
- Remember that security features != secure features
- Never depend on security through obscurity alone
- Don't mix code and data
- Fix security issues correctly

Learn From Mistakes

You might have heard the saying "what does not kill you makes you stronger", but I swear that in the world of software engineering we do not learn from mistakes readily.

- If you find a security problem in your software or learn of one in your competitor's products, learn from the mistake. Ask questions like:
- How did the security error occur?
- Is the same error replicated in other parts of the code?
- How could we have prevented this error from occurring?
- How do we make sure this type of error does not happen in the future?
- Do we need to update education or analysis tools?

A Hard lesson

About four years ago, an obscure security bug was found in a Product I was close to. Once the fix was made, I asked the product team some questions, including what had caused the mistake. The development lead indicated the team was too busy to worry about such a petty, time wasting exercise. During the next year, outside sources found three similar bugs in the product. Each bug took about 100 man hours to remedy.

I presented this to the new development lead, and pointed out that if 4 similar issues were found in the space of 1 year, it would be reasonable to expect more. He agreed and we spent four hours determining what the core issue was. The issue was simple: some developers had made some incorrect assumptions about the way a function was used. We looked for similar instances in the entire code base and found four more and fixed them all.

Next we added some debug code to that function that would cause the application to stop if the false assumption condition arose. Finally we sent email to the entire development organization explaining the issue and the steps to take to make sure the issue never occurred again. The entire process took less than 20 man hours.

The issue is no longer an issue. The same mistake is sometimes made, but the team catches the flaw quickly because of the newly added error-checking code.

Minimize Your Attack Surface

When you install more code and listen on more network-based protocols, you quickly realize that attackers have more potential points of entry. It's important that you keep these points of entry to a minimum and allow your users to enable functionality as needed.

If a feature is not running it cannot be attacked. The standard rule, known as the Pareto Principle is known as the 80-20 rule. The 20% of functionality used by 80% of your users should be enabled by default. The remaining 80% should be set off by default.

Use Defense in Depth

The security in your application should be layered appropriately. There should be no single point of failure that compromises or totally opens the system.

Look at the way security is performed at a bank for instance. When was the last time you entered a bank to see a bank teller sitting on the floor in a huge room next to a big pile of money?

To get at the big money in a vault requires that you go through multiple layers of defense.

Think of the different security concepts we are learning as being gates. We will put as many gates in an application in order to protect the application appropriately. A company address book application would require far fewer gates than the payroll system. As appealing as it is to use every security feature we are learning in every application, we need to set the bar appropriately.

Use Least Privilege

All applications should execute with the least privilege to get the job done and no more. Do not design applications that require Local Admin access or the Local System account, for instance.

The reason for this is that if an attacker can find a vulnerability in your code and manage to inject their own code, it should not be running with elevated privileges, and be able to, say, format your hard drive.

Backward Compatibility Will Always Give You Grief

If you find a vulnerability in your code that you rectify by releasing a new version of your program, any communication or access to the old code will be problematic. And your old vulnerabilities will live forever as long as you support the previous version.

Assume External Systems Are Insecure

Assuming external systems are insecure is related to defense in depth, the assumption is actually one of your defenses. Consider any data you receive from a system you do not have complete control over to be insecure and a source of attack. This is especially important when accepting input from users.

Don't assume that your application will always communicate with an application that limits the commands a user can execute from the user interface or Web-based client. Many server attacks take advantage of the ease of sending malicious data to the server by circumventing the client altogether.

Plan on Failure

Stuff fails, and stuff breaks. Make a plan to handle if the firewall is breached or your website is defaced. That will never happen? Wrong answer. It's like having an escape plan in case of a fire, you hope it will never happen, but if it does you have a better chance of getting out alive.

Fail to a Secure Mode

So, what happens when you do fail? You can fail securely or insecurely. Failing to a secure mode means that the application has not disclosed any data that would not be disclosed ordinarily, and that the data still cannot be tampered with. An attacker might try to make your application fail and bypass security mechanisms, for example by relying on exception handling that leaves the process running as System Admin.

Remember That Security Features != Secure Features

Remember that sprinkling some magic security pixie dust on your application does not make it secure. We must make sure to include the correct features and to use them correctly to defend against attack. Using 256 bit encryption is useless for example, if your password is always set to "password". It doesn't matter how many bits you encrypt something if the attacker can guess your password within 3 tries.

Never depend on security through obscurity alone

Always assume the attacker knows everything you know and has access to all of the source code and designs. It is trivially easy for an attacker to determine obscured information. Obscurity is a useful defense, so long as it is not your only defense.

As an example, you may think it is amazingly smart to hide some data in the registry under the screen saver key but the attacker who is running a Registry Monitor program will take about 5 seconds to find that information.

Don’t mix code and data

Data is data and code is code. Once you add code to the data you give the user the ability to inject their own code into your systems. Allowing users to mark up bulletin board comments with HTML for example also gives them the ability to execute JavaScript programs. Allowing users to specify the value of a WHERE clause in SQL like this:

```
“Select * from Users Where CustID=’” & Textbox1.Text & “’”
```

allows a crafty hacker to inject their own code into the system, think about what would happen if they type this into the textbox:

```
1'; DROP DATABASE; --
```

Fix security issues correctly

If you find a security code bug or design issue, fix it and go looking for similar issues in other parts of the application. You will find more like it. Security flaws are like cockroaches: you see one in the kitchen and get rid of it. The problem is that the creature has many brothers, sisters, grandkids, cousins, nieces, nephews and so on. If you have a cockroach, then you have a cockroach problem. Unfortunately, the same holds true with security bugs, the person making the mistake probably made the same mistake elsewhere.