

ECE 1724: Security Concept & Definitions

Richard Reiner & David Lie
Department of Electrical and Computer Engineering
University of Toronto

1

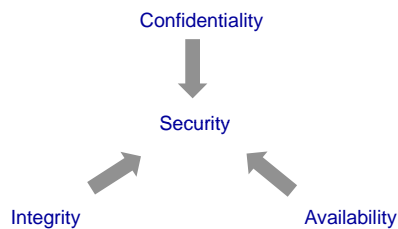
Definition of Security

- Security is a very nebulous term -- what does it mean to be secure?
 - Varying definitions, but in the end often one has to rely on intuition
 - From real life, people have an intuitive idea of what is secure, and these can for the most part be applied to computer systems
 - However, as a science, one would like a more formal definition



2

Components of Security



3

Confidentiality

"The protection of information or resources from exposure."

- There are 2 aspects of information or resources that are often important to conceal:
 1. The *content* of a piece of information or resource. Ex: most people want to keep their account passwords confidential. In many cases, content may also cover things such as configuration, cost or other attributes.
 2. The *existence* of information or resource. In this case, the mere knowledge that something exists is damaging. Ex: a company may want to keep the existence of a new product secret until it is ready for commercial introduction to the market.



4

Integrity

"The trustworthiness of information or resources."

- Integrity tells you how ready you should be to believe something. Like confidentiality, there are 2 aspects of integrity that are important:
 1. The correctness of the *contents* of a piece of information or resource. Ex: by altering the contents of a piece of e-mail, you are violating the integrity of its contents.
 2. The correctness of the *origin* of a piece of information. Ex: by altering the sender's e-mail address, you can mail an e-mail look like it's coming from someone else. Ensuring that something came from where you think it did is called *authentication*.



5

Availability

"The ability to access or use information or resources as desired"

- In real terms, a resource is available if it is accepting and responding to requests. Information is available if the service which stores that information is up and running. Availability is generally more difficult to deal with for 2 reasons:
 1. In automata theory, availability is not a finite property. Unlike confidentiality or integrity, you can't always say that at time t some object has become unavailable.
 2. Many systems deal with availability in a probabilistic fashion. Components in any system have some inherent level of unreliability and the unreliability of the system is the composition of unreliability of these components. However, availability cannot be treated probabilistically for security since and active adversary will cause unlikely events to occur more often (i.e. network flooding)



6

CIA = Security

- Any system that can be called secure provides all three (Confidentiality, Integrity & Availability) of these attributes to some degree. Remember that no system is absolutely secure, and so no system can provide all or any of the three absolutely.
- Measures:
 - Confidentiality & Integrity are often provided by cryptographic algorithms. Their strength is often measured in terms of complexity (how long will it take to break the algorithm). Ex: 256-bit keys are considered more secure than 128-bit keys.
 - Availability is very hard to measure. Traditional measures (probabilistic) measure availability in terms of percentage of time a system is accessible. A system with 99.999% (five 9's) availability is only down 0.001% of the time. Unfortunately, this doesn't apply well to measuring security.



7

Assessing Security: Threats

- A *threat* is a potential vector for a system's security to be compromised.
 - When an attacker exercises a threat, it becomes an *attack*.
 - If the attack is successful, a system's security is then *compromised*.
- Threats can come in many forms, and a good security practitioner learns to identify and assess their seriousness:
 - A system hooked up to the Internet. Network traffic that arrives from the Internet and is accepted by the system is a threat.
 - The University allows students to hook laptops up to their wireless network. The laptops pose a threat because they may transport viruses they are infected with.



8

Assessing Security: Vulnerabilities

- A *vulnerability* is a flaw in a system that has a security implication.
- Vulnerabilities are very difficult to identify (especially after any reasonable amount of testing). They are almost always serious:
 - A unchecked string copy allows an attacker to overflow a buffer and execute arbitrary code in a privileged program.
 - During configuration, a system administrator forgets to disable debug mode on a program, allowing an attacker to gain administrator privileges
 - A naïve user does not change the default password on their router from the factory default. An attacker who is experienced with the router guesses their password and gains access to their network.



9

Threats and Vulnerabilities

- Compromises occur when an attacker matches a threat (think of these as the attacker's arsenal), with a vulnerability (these are weaknesses in the system).
- In the previous slide, all vulnerabilities were accompanied by an attack the attacker used. By removing the threat, a security practitioner can prevent the attack:
 - Even though the router's password was not changed, the router is behind a firewall that prevents the attacker from connecting to the router directly. The vulnerability cannot be exploited.



10

Human Factors: User Awareness & Assurance

- Humans are actually the leading causes for computer security breaches. They are prone to making mistakes:
 1. Humans make configuration, design and implementation errors. To counter this, people try to find their own (and other's) mistakes. The amount of checking that has been done to remove errors is defined by the level of *assurance*.
 2. Humans are not all equal in terms of knowledge and education. Security knowledgeable users will create fewer vulnerabilities than unknowledgeable users. *When assessing a system, always keep in mind who created it and who is going to use it.*



11

Trust

- Trust defines how much exposure a system has to a particular interface. The more a system trusts a component, the more likely that component will be a serious threat, and the more likely that threat will find a vulnerability.
- The danger is that rather than actively assigning trust, trust in system is usually assigned via the assumptions the designer makes.



12

Introduction to Cryptographic Mechanisms

13

Reasons we need Cryptography

- Cryptography is an indispensable tool for security
 - Provides very powerful guarantees
 - Much encryption seems impossible to break (though surprisingly no concrete proof)
 - However, it's not infallible, must understand it to use it
- Cryptographic techniques can provide
 - Confidentiality/Secrecy
 - Integrity
 - Authentication
 - Nonrepudiation



14

Basic Terminology

- Plaintext or Cleartext:
 - Data that is plainly readable and understandable by anyone
- Ciphertext:
 - Data that has been processed so that only authorised principals can read it
- Encryption:
 - The process of making plaintext into ciphertext
- Decryption:
 - The opposite of encryption
- Key:
 - A value that is used with encryption or decryption to make the particular process unique
- Channel:
 - A means of communicating data between parties



15

Cryptographic Mechanisms Overview

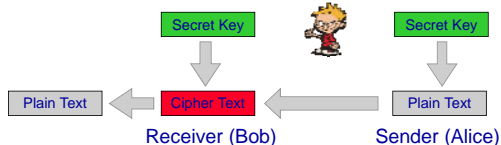
- Ciphers:
 - Symmetric or Private-Key Ciphers
 - Asymmetric or Public-Key Ciphers
- One Way Hashes:
 - Message Authentication Codes
 - Digital Signatures



16

Symmetric Ciphers

- Examples AES (Rijndael) and DES
 - Single key used for encryption and decryption
 - Pretty fast when implemented in hardware



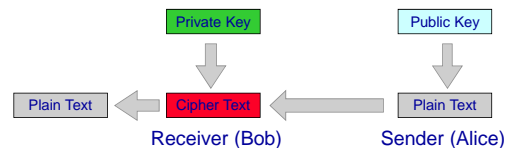
- How do we securely distribute the keys?



17

Asymmetric Ciphers

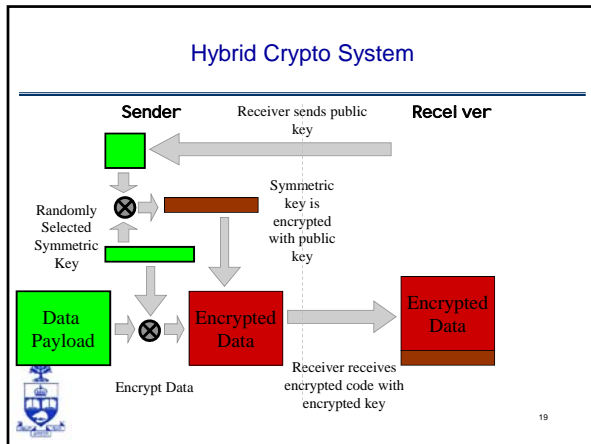
- RSA and DSA are the common ones
 - Pairs of keys
 - Public key and is used to encrypt data
 - Private key and is used to decrypt data



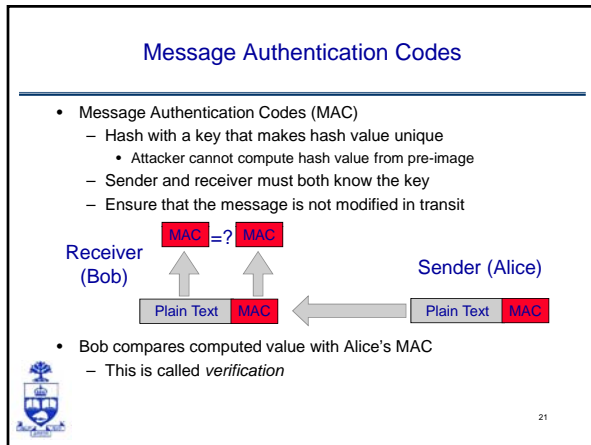
- Public-key ciphers are very slow
 - Typically use hybrid systems with both ciphers together



18



- ### Cryptographic One-way Hashes
- Hash Function (CRC, Parity)
 - Converts a *pre-image* into a *hash value*
 - Lossy compression function
 - Hash value is shorter than pre-image
 - This means you have the problem of collisions (two pre-images with the same hash value)
 - One-way hash function (SHA, MD5)
 - Difficult to reverse
 - Hard to find a pre-image that matches a given hash value
 - Collision-resistant/collision-free
 - Hard to find two pre-images that have the same hash value
- 20



- ### Digital Signatures
- Combine public-key cryptography and One-way Hashes
 - Provides Authentication, Integrity and Nonrepudiation
 - The message has not been tampered with
 - The message is from the person we think sent it
 - Alice creates a hash of the message and then encrypts it with her private key
 - Bob can decrypt the hash with Alice's public key and verify it against the message
 - Since only Alice has the private key, only she could have produced the signature
 - So since the message is signed by Alice, and only Alice has the key, then she cannot deny that she signed the message
 - This is useful in digital contracts
- 22

Example Attacks

23

- ### Cross Site Scripting (XSS)
- **MySpace, October 2005:** "Samy" had **amassed over 1 million** friends on the popular online community... How did Samy transcend his humble beginnings of only 73 friends to become a veritable global celebrity? The answer is a combination of **XSS tricks and lax security in certain Web browsers.**
 - **Google, January 2007:** ... **Gmail XSS (cross-site scripting)** security problem. Using a **small piece of JavaScript** you can put on any server, the **user's contact names & email addresses are revealed ...**
- 24

Cross Site Scripting (XSS)

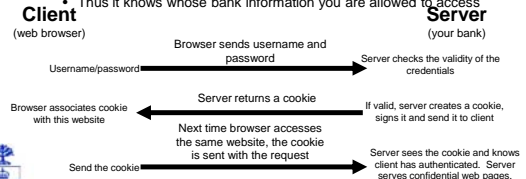
- **Cisco, April 2007:** "A **cross-site scripting (XSS) vulnerability** in the online help system distributed with **several Cisco products** has been independently reported to Cisco by Erwin Paternotte from Fox-IT and by Cassio Goldschmidt. The vulnerability would **allow an attacker to execute arbitrary scripting code** in a user's web browser if the attacker is successful in enticing the user to follow a specially crafted, malicious URL."



25

Web Authentication

- A Http "Cookie" is used to track whether you have logged in or not yet:
 - Some web sites (i.e. a bank) require you to **authenticate**:
 - This is, you have to prove your identity
 - Usually done via a username/password
 - Once you have logged in, the web site send you a cookie so when you make a request, the bank knows *who* is making the request:
 - Thus it knows whose bank information you are allowed to access



26

Authentication Cookies

- If an attacker wants to impersonate you, all she has to create a cookie that says they are you:
 - Several safeguards to prevent this
- Attacker can try to **forge** a cookie:
 - Create a fake cookie that says they are someone else.
 - Servers **digitally sign** the cookies, attacker cannot fake the digital signature
- Attacker can try to steal the cookie from you:
 - Cookies are encrypted before sending, attacker cannot sniff network traffic and get the cookie that way



27

Stealing Cookies

- Instead, what the attacker does is trick your browser into sending the cookie to them instead of to the bank web site:
 - Once the attacker has the cookie, they can just send it to the bank web site and the bank and convince the bank they are you
 - Attacker can do this by tricking you into visiting their website (offer some free wallpapers, celebrity pictures, etc...)
 - When on their site, the site send your web browser some javascript code which your web browser executes...



28

Same Origin Policy

- This is not completely straight forward because browsers implement the **Same Origin Policy (SOP)**:
 - SOP says that javascript from a web site can only access cookies from the web site the javascript came from
 - Attacker's javascript will only be able to access cookies from their own web site!



29

XSS Demo



30

The Vulnerability of the Decade

- In 2000 Crispin Cowan wrote "[Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade](#)" (available on CCNET)
 - Outlined some defenses against this attack
- 6 years later, the attack is still very prominent, but for the most part the technical problem is solved:
 - A combination of hardware support (non-executable pages) and software to utilize the hardware support will stop most attacks (implemented in Windows XP SP2 patch)
 - However, before we go into how to defend against it, let us examine and understand how the attack works

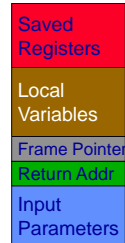


31

Program Stack Review

```

movl $param,(%esp); pass a input
call MySub      ; call MySub
MySub push %ebp      ; push frame pointer
    mov %esp,%ebp
    sub $0x4,%esp    ; allocate local vars
    push %eax       ; save registers
    
```



- Remember how subroutine calls worked:
 - Push Input Parameters onto stack
 - Push Return Address onto stack
 - Push Frame Pointer onto stack
 - Allocate room on stack for Local Variables
 - Saved Registers



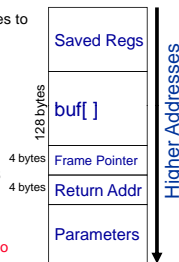
32

Buffer Overflow

- Remember that stack grows from high addresses to low addresses
 - But arrays are filled in from lower addresses to higher ones
- Say you have code like this:


```

void func(char *str) {
    char buf[128];
    strcpy(buf, str);
}
            
```
- What happens if `str` is longer than `buf`?
 - `strcpy` will keep copying until it hits a null character. In this case, `str` has to be 136 bytes (128 + 4 + 4) to overwrite the return address with the contents of `str`.
- Is there a safer alternative to `strcpy`?
 - `strncpy`, you can specify how many bytes to copy



33

Buffer Overflow

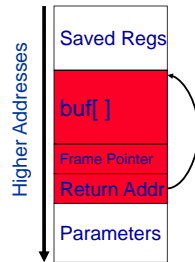
- Once return address is changed, when the subroutine returns, it will return to wherever the return address is changed to. The attacker can hijack the program by altering the instructions that program executes.
- The vulnerability requires:
 1. A string that is input from the attacker.
 2. A buffer that is located on the stack (meaning it's a local variable in a subroutine).
 3. A bug where the programmer copies the string from the attacker into a buffer without checking that the input string will fit into the buffer.
- Because the buffer is on the stack, and the attacker overwrites values on the stack, this is commonly referred to as a "**Stack Smashing Attack**".



34

Arbitrary Code Execution

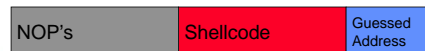
- We've seen how the attacker can redirect the execution of the program by changing the return address. However, to have the vulnerable program execute arbitrary code, the attacker needs somewhere to put the code.
 - Put it in the buffer that was vulnerable!
 - What kind of "arbitrary code" does the attacker want to execute?



35

Putting it all Together

- Now assume we (the attacker) have some code we want to inject (call it shell code). How do we use it?
 - We can control where the program will return to by overwriting the return address, but we don't know where the shellcode will sit in memory, so we have to guess.
 - We put our guesses at the end since it's the end of the buffer that will overwrite the return address
 - We pad the front of the shell code with `nop`'s (called a `nop sled`). This way if we jump into any address in that region, we will eventually execute the shellcode.



36

Sample Buffer Construction Program

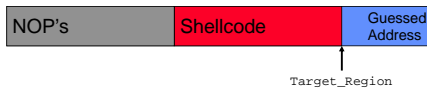
```
ptr = (unsigned long *) attackBuffer;

/* fill the buffer with the return address */
for ( i = 0; i < 21; i++)
    *(ptr + i) = TARGET_ADDR;

/* copy in shellcode */
for ( i = TARGET_REGION - shellLength; i < TARGET_REGION; i++)
    attackBuffer[i] = shellcode[i - TARGET_REGION + shellLength];

/* fill the front with NOPS */
for ( i = 0; i < (TARGET_REGION - shellLength); i++)
    attackBuffer[i] = NOP;

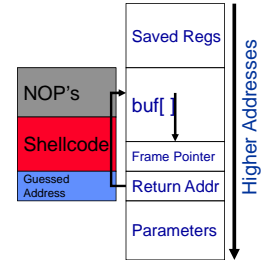
/* terminate the buffer */
attackBuffer[BUFSIZE-1] = '\0';
```



37

Putting it all together

1. Program copies input buffer which overflows the buffer on the stack
2. Function eventually exits and jumps to the return address on the stack which as been overwritten. This causes the program to land somewhere within the nop sled
3. The program executes the nops and eventually hits the shellcode and executes it. A shell is spawned!



38

ECE1724: Course Information

David Lie

39

Overview

- Main source of information: Course Web page
 - The course involves reading and discussing papers, understanding what makes a good paper.
 - Independent research on a security project
- Course deliverables:
 1. Paper presentations: Each student is responsible for making an oral presentation of (30%)
 2. Course Project Proposal: Outline what you intend to do for your course project. Projects will be done in groups (5%)
 3. Midterm Project Update: Midway through the term project groups will present what they have achieved so far and their plans going forward (20%)
 4. Final Project Presentation & Report: Groups will present the results of their project and submit a 10-page report. (45%)



40

Readings and Presentations

- Students are expected to read the papers every week
- Each week a group of 2-3 students will present the papers for the week. Format for each paper presented will be:
 1. Summary of the paper (~10 minutes)
 2. A presenter will argue for the paper (Pro)
 3. A presenter will argue against the paper (Con)
 4. Discussion
- Remember that you can separate the paper *idea* from the paper's *execution* (writing, implementation, evaluation).
 - Many papers may have good ideas, but research them poorly and vice-versa



41

Projects

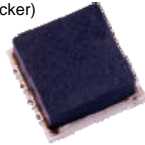
- Students should work in groups to pursue a research oriented project:
 - Project should solve an interesting problem in the area of computer security
 - Students should place their work in the context of existing related work, it should be clear how their work should improve on existing solutions
- Students are encouraged to think of their own projects **related to their research areas**:
 - Some "suggested" research projects exist as well



42

Potential Projects: TPM Attestation

- Machines must interact with other untrusted hosts over the internet. Attestation is a mechanism that allows one host to prove its integrity to another host.
 - This is done via a special piece of hardware called a TCG TPM, present on most mother boards
 - TPM can store cryptographic keys and sign messages with them
 - Also used for hard drive encryption (i.e. Bitlocker)



43

Potential Projects: Detecting Spyware

- Spyware reports information on a user's web surfing habits to a remote attacker:
 - Spyware is often difficult to identify, many processes running on a system, how do we know which is malicious?
 - However, there must be a statistical correlation between the behavior of spyware on a system and the amount of web surfing that has occurred
 - Can also feed web browser with known commands and look for correlations among processes
 - Almost all spyware is windows based, so knowledge of Windows OS is a plus



44

Potential Project: VOIP Security

- Many people assume that VOIP is as secure as traditional Plain Old Telephone Systems (POTS). This is not the case:
 - VOIP systems may be more susceptible to traffic analysis and eavesdropping.
 - Cheaper and easier to send voicemail spam
 - Clients themselves are susceptible to attacks



45