Secure By Design

Chapter 10

(Some slides are taken from M. Howard, Microsoft Research)

Outlines

- ★ Security & Privacy (Revisit)
- ★ Security Components & Security Policy (Revisit)
- ★ Secure by design
- ★ Tools
  - Input Validation & Threats Modeling
- ★ Secure by design in actions
  (A case from Microsoft.)

Security & Privacy (Revisit)

- Security
  - Who can do what when?
- Privacy
  - The freedom to control access to our personal information

What do we need to create a secure system?

Security Components

- ★ Authentication
  - “Who are you? Are you really the person whom you claim to be?”
- ★ Authorization
  - “Do you have the authority to do what you are trying to do?”
- ★ Accounting (Auditing)
  - “What did you do?” log

Supporting Concepts

- ★ Integrity
  - Integrity (n) “the quality or state of being complete or undivided”
- ★ Software Engineering & Threat Modeling
  - “Threat modeling is a method of addressing and documenting the security risks associated with an application.”
- ★ Validation of Input
  - “All input is evil until proven otherwise”

Integrity as the forth A

- ★ Integrity is sometime referred as Authenticity—hence it is sometime mentioned as the forth “A” of security components.
- ★ How can we preserve the integrity of data?

What have we learned?

- ★ Authentication
- ★ Authorization
  - Confidentiality
  - Integrity
  - Availability
- ★ Auditing

What is secure by design?

- ★ Plan more than just functionality (Plan for Security)
- ★ Attack Surface Reduction/Analysis
- ★ Threats & Risk Modeling

Secure By Design in “Login Program”

Prog 1.	Prog 2.	Prog 3.

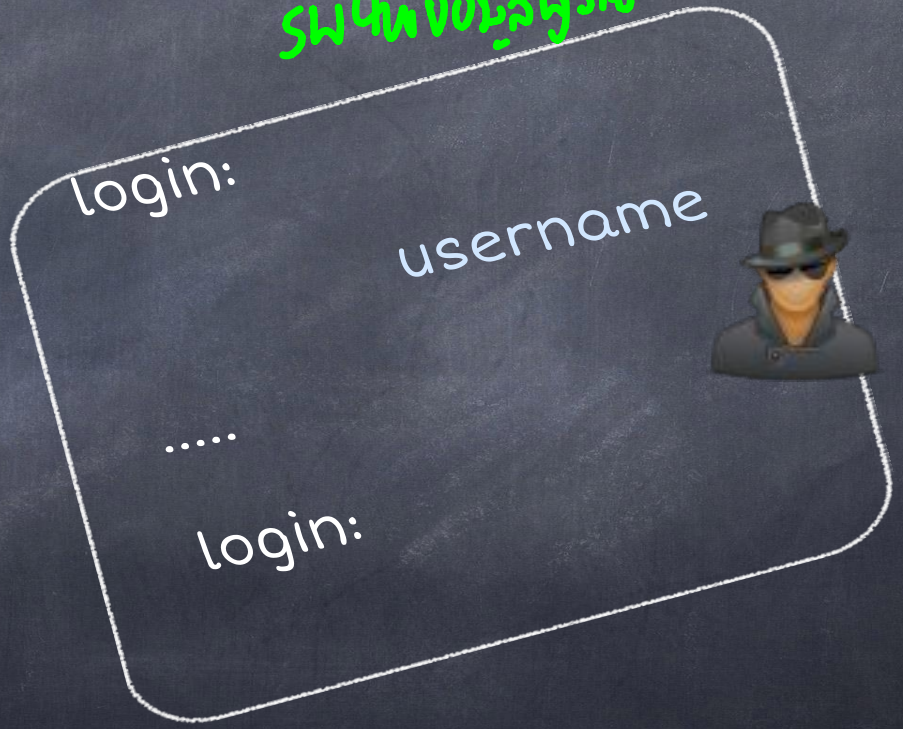
1. Input [login name]
2. Fetch [saved password]
3. If no entry then
Exit
4. Input [password]
5. Compare passwords.
6. If valid then
start session
else
exit
End if
	1. Input [login name]
	2. Input [password]
	3. Fetch [saved password]
	4. If no entry then
	Exit
	5. Compare passwords.
	6. If valid then
	start session
	else
	exit
	End if
		1. Input [login name]
		2. Input [password]
		3. Fetch [saved password]
		4. If no entry then
		[saved password] <-
		random
		5. Compare passwords.
		6. If valid then
		start session
		else
		exit
		End if

Prog 1.

ผู้ชายคนนี้ user นี้

1. Input [login name]
2. Fetch [saved password]
3. If no entry then
   exit
4. Input [password]
5. Compare passwords.
6. If valid then
   start session
   else
   exit
   End if

รหัสผ่านของผู้ชาย



Prog 2.

1. Input [login name]
2. Input [password]
3. Fetch [saved password]
4. If no entry then
   exit
5. Compare passwords.
6. If valid then
   start session
   else
   exit
   End if

เร็ว : Username ผิด

ช้า : PW ผิด

login:

username

Password:

****

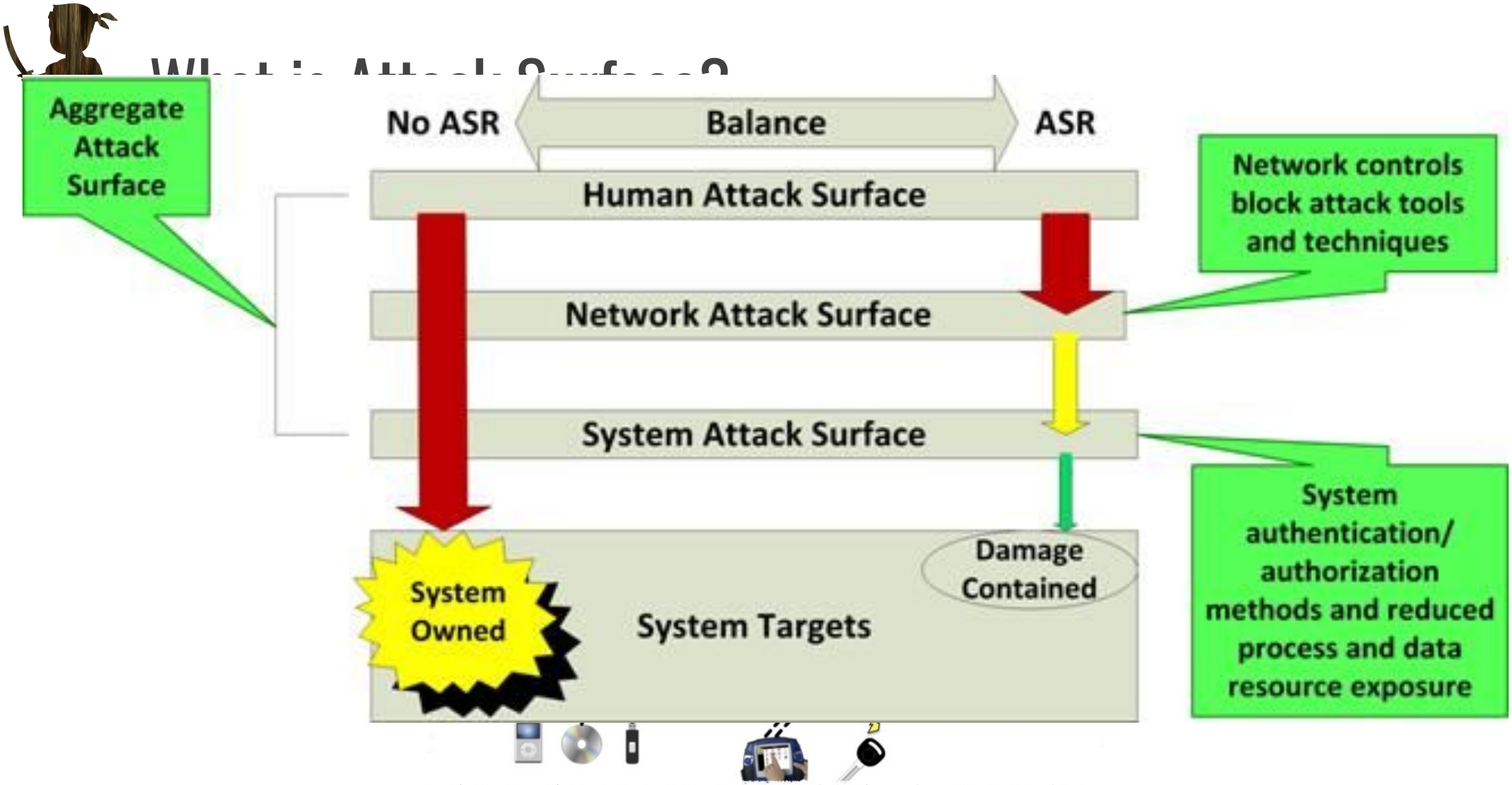
.....

login:

Prog 3.

1. Input [login name]
2. Input [password]
3. Fetch [saved password]
4. If no entry then
   [saved password] <- random
5. Compare passwords.
6. If valid then
   start session
   else
   exit
   End if

ห้าม exit : ขัดเวลาไม่ได้


Aggregate Attack Surface
No ASR
Balance
ASR
Human Attack Surface
Network Attack Surface
System Attack Surface
Network controls block attack tools and techniques
System authentication/ authorization methods and reduced process and data resource exposure
System Owned
Damage Contained
System Targets

Attack Surface Reduction

↳ where attacker can entry or extract data from a system

- ★ Defense in Depth
- ★ Least Privilege
- ★ Secure Defaults
  - Less code running = less stuff to attack

Facts

Microsoft have been using Secure By Design since 2004.

- ★ Before Windows 2003 Server (Windows XP Service Pack 2), Microsoft was notoriously called evil empire.
- ★ Microsoft used to sell functions, not security.
- ★ Mike Howard and his team leaded Microsoft for Secure By Design.

Let's learn from Microsoft

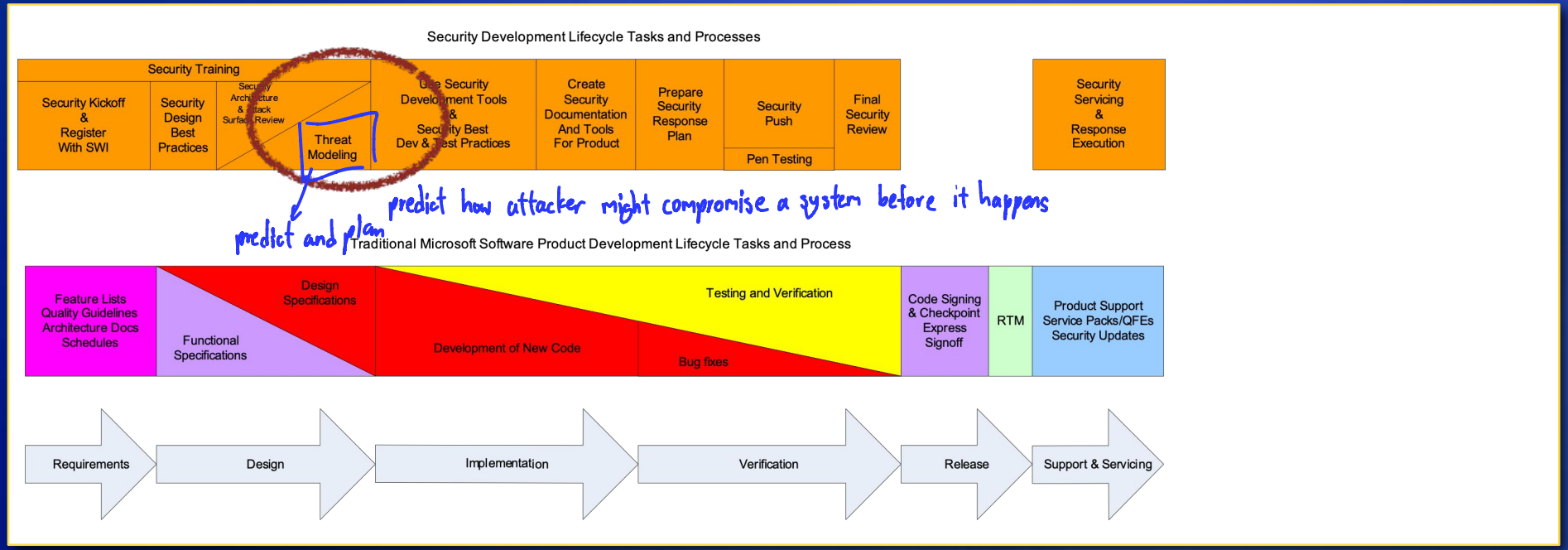
The Security Engineering & Communications Group

- Help you secure your products
- "Security-as-in-threats" NOT "Security-as-in-crypto"

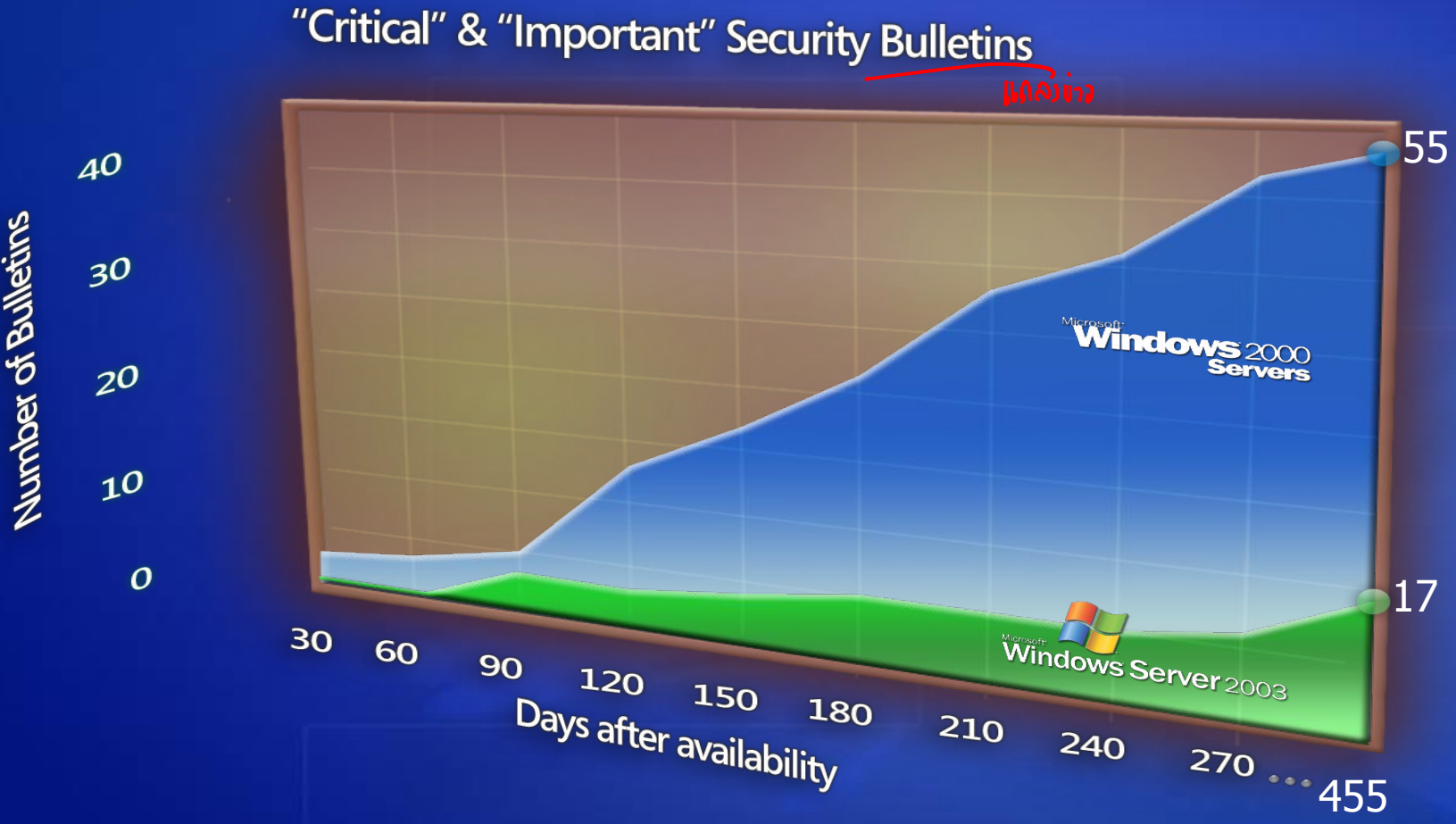
not just อย่างนี้เท่านั้น

mailto:switeam
http://swi
http://msnsecurity/sdl

Security Development Lifecycle



Early Results of the SDL



A Case Study: MS04-011

VULNERABILITY IDENTIFIERS	IMPACT OF VULNERABILITY	WINDOWS 2000	WINDOWS XP	WINDOWS SERVER 2003
LSASS Vulnerability - CAN-2003-0533	Remote Code Execution	Critical	Critical	Low
LDAP Vulnerability – CAN-2003-0663	Denial Of Service	Important	None	None
PCT Vulnerability - CAN-2003-0719	Remote Code Execution	Critical	Important	Low
Winlogon Vulnerability - CAN-2003-0806	Remote Code Execution	Moderate	Moderate	None
Metafile Vulnerability - CAN-2003-0906	Remote Code Execution	Critical	Critical	None
Help and Support Center Vulnerability - CAN-2003-0907	Remote Code Execution	None	Critical	Critical
Utility Manager Vulnerability - CAN-2003-0908	Privilege Elevation	Important	None	None
Windows Management Vulnerability - CAN-2003-0[illegible]	Privilege Elevation	None	Important	None
Local Descriptor Table Vulnerability - CAN-200[illegible]	Privilege Elevation	Important	None	None
H.323 Vulnerability* - CAN-2004-0117	Remote Code Execution	Important	Important	Important
Virtual DOS Machine Vulnerability - CAN-2004-0118	Privilege Elevation	Important	None	None

Secure Design

- Reduce Attack Surface
  - Defense in Depth
  - Least Privilege
  - Secure Defaults

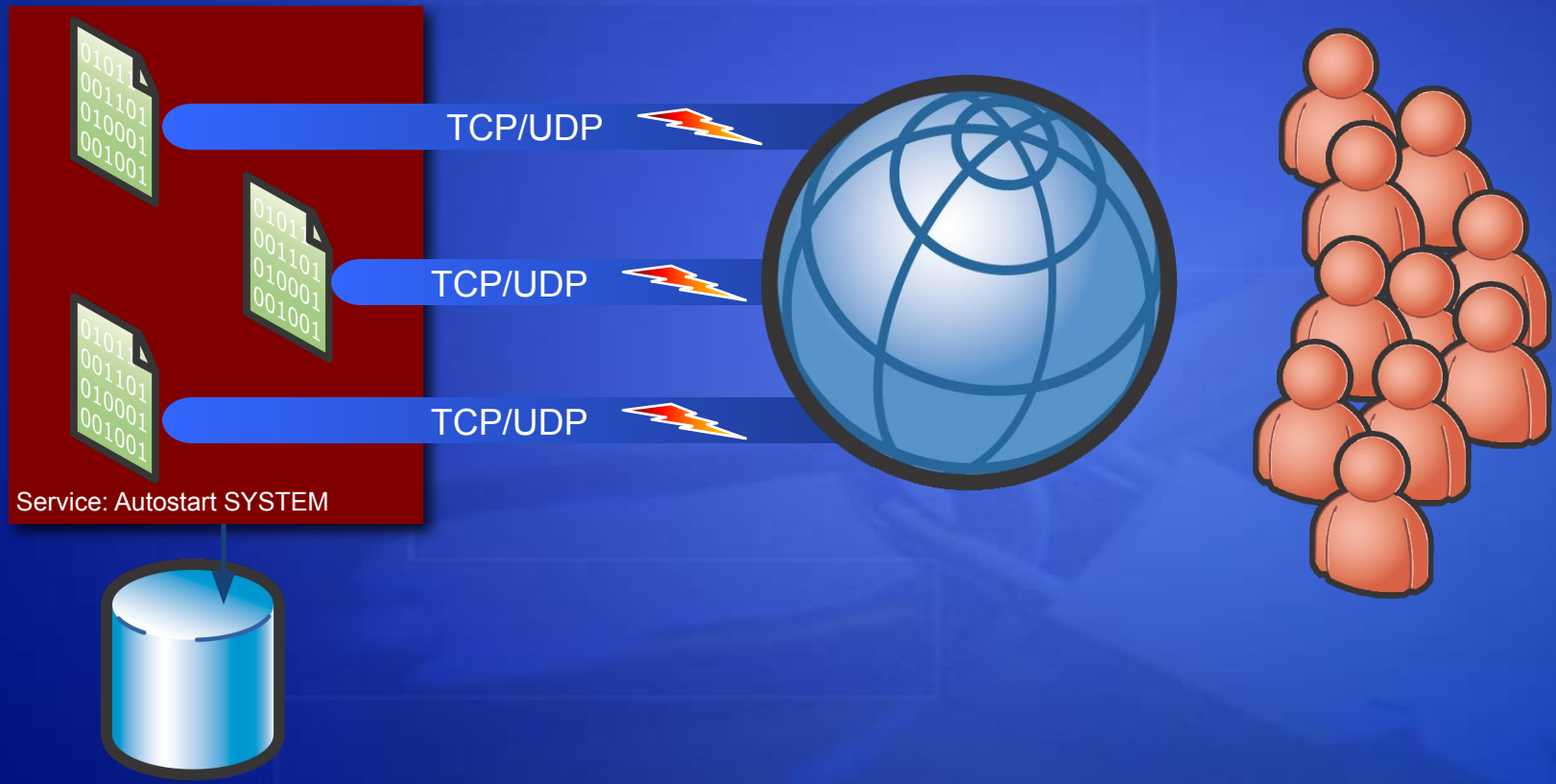
Defense in Depth (MS03-007)
Windows Server 2003 Unaffected

The underlying DLL (NTDLL.DLL) not vulnerable	Code fixed during the Windows Security Push
Even if it was vulnerable	IIS 6.0 not running by default on Windows Server 2003
Even if it was running	IIS 6.0 doesn't have WebDAV enabled by default
Even if it did have WebDAV enabled	Default maximum URL length (16kb) prevented exploitation (>64kb needed)
Even if the buffer was large enough	Process halts rather than executes malicious code, due to buffer-overrun detection code (-GS)
Even if it there was an exploitable buffer overrun	Would only 'network service' privileges – commensurate with a normal user

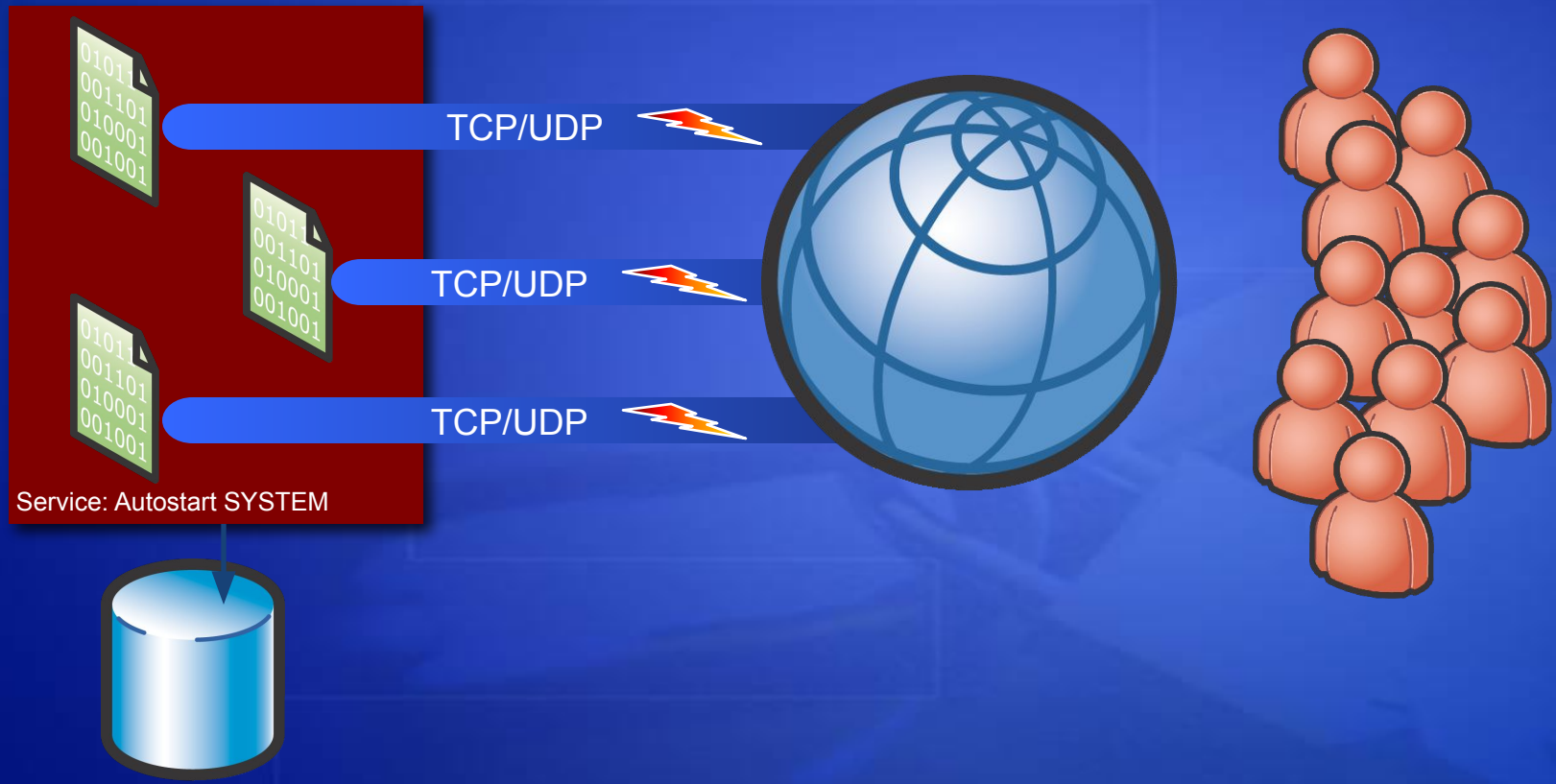
Secure Defaults

- Less code running by default = less stuff to attack by default
- Slammer & CodeRed would not have happened if the features were not enabled by default
- Reduces the urgency to deploy security fixes
  - A 'critical' may be rated 'important'
- Defense in depth removes single points of failure
- Reduces the need for customers to 'harden' the product
- Reduces your testing workload

Attack Surface Reduction (ASR) Ideas

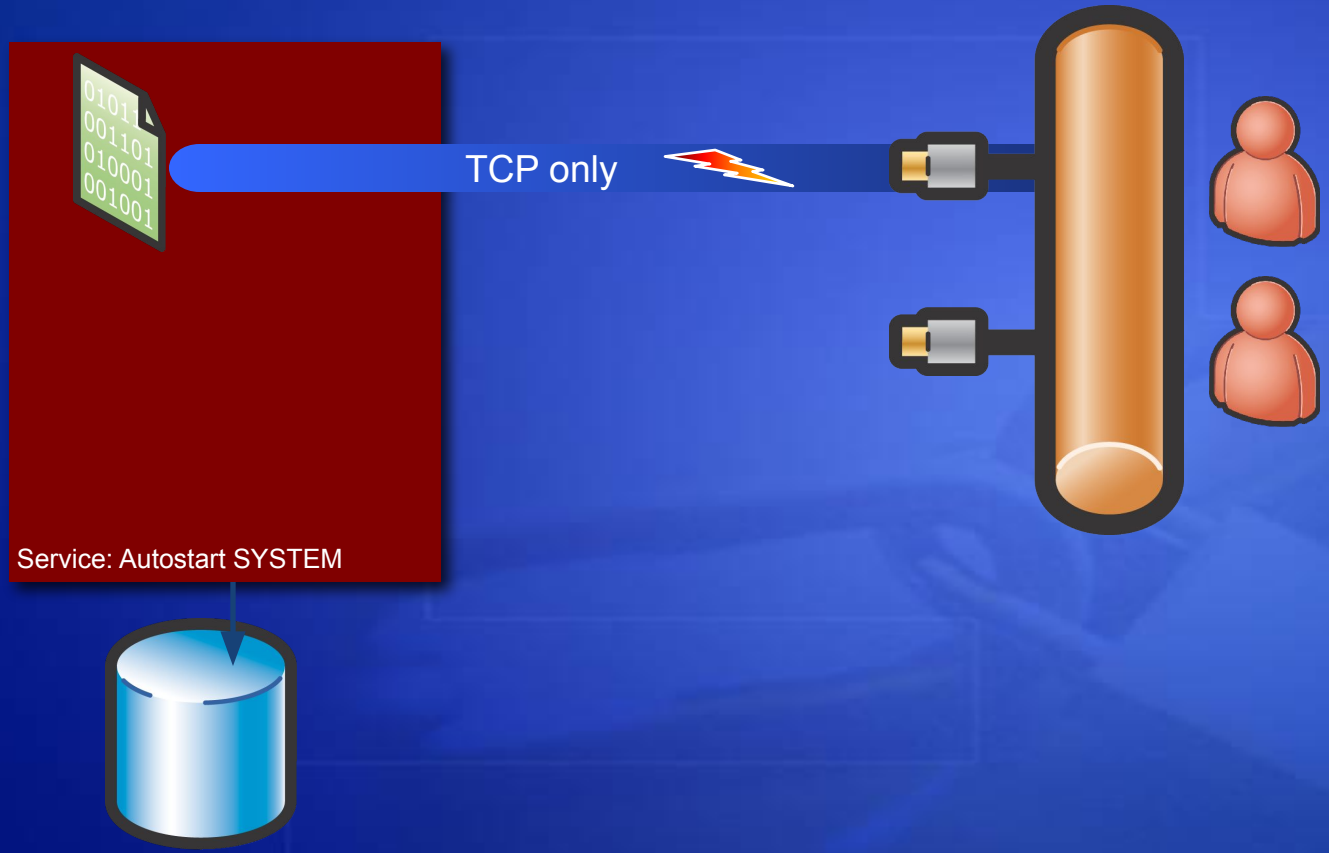


Turn off less-used ports

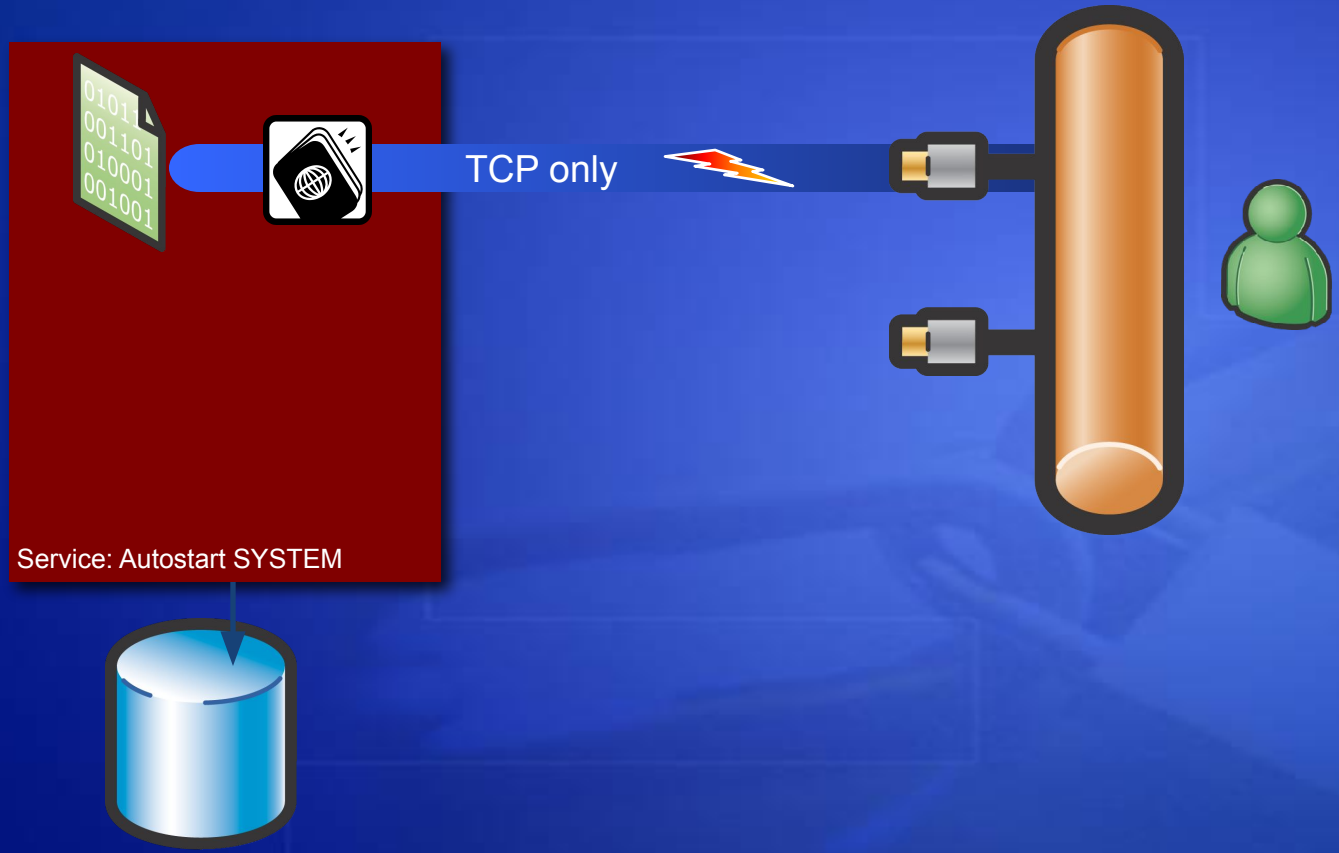


Turn off UDP connections

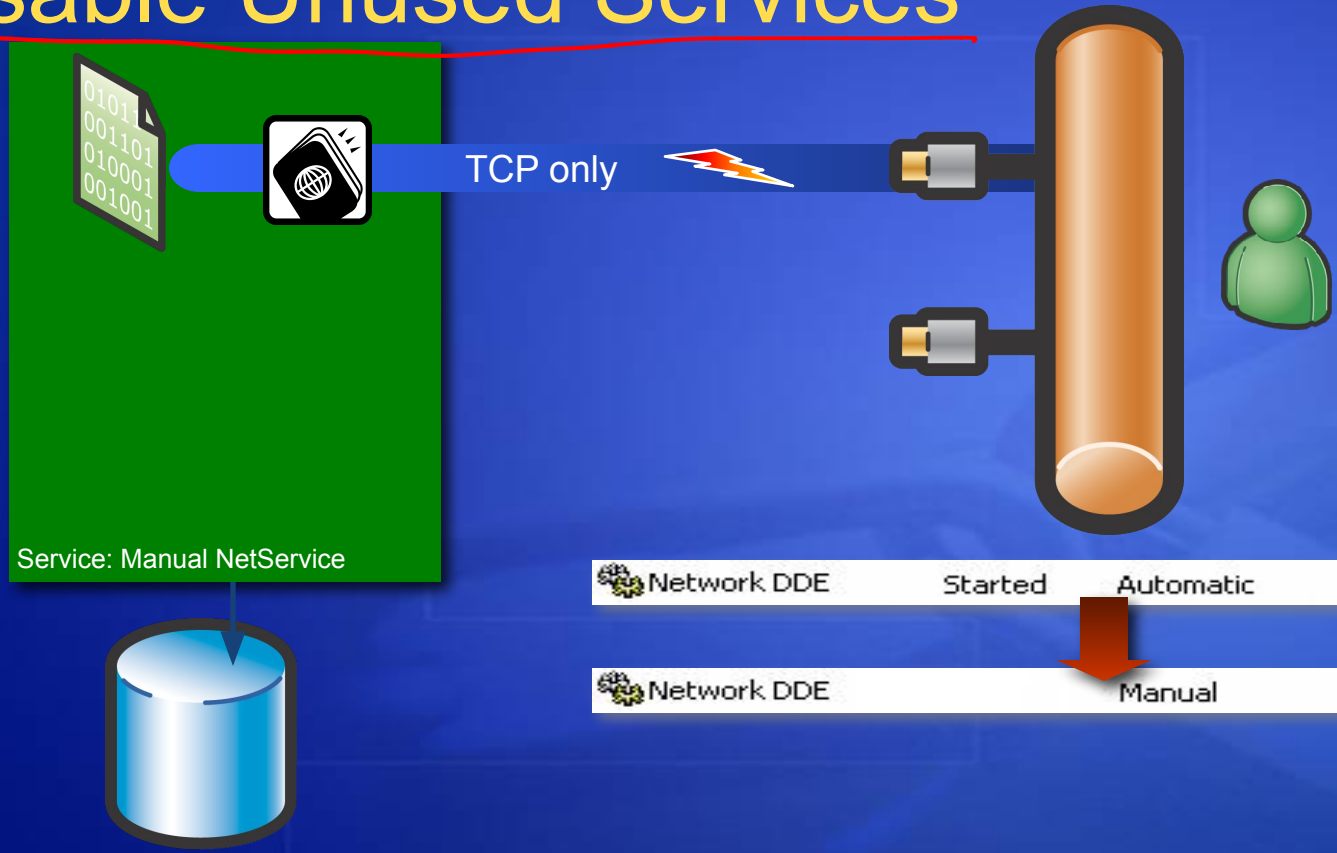
Restrict requests to a small IP range and subnet



Authenticate Connections

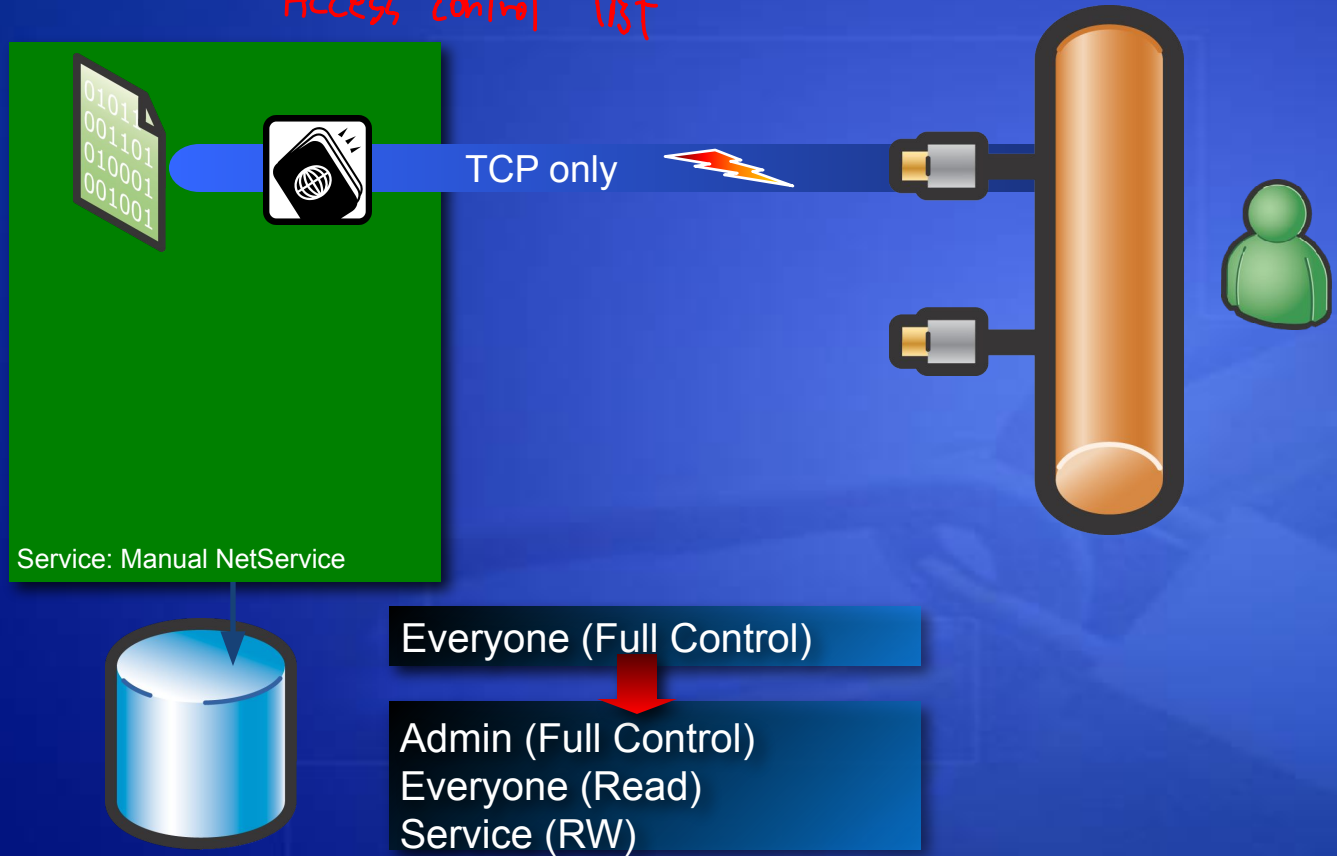


Reduce Privilege and Disable Unused Services



Harden ACLs

Access control list



Increased Attack Surface means Increased Security Scrutiny…

- On by default
- Running as SYSTEM
- Open, unauth TCP socket

- Off by default
- Running with least priv
- Open, TCP socket limited to local subnet

Threat Modeling plan & predict

★ Think like a bad guy.
(but do not be a bad guy yourself)

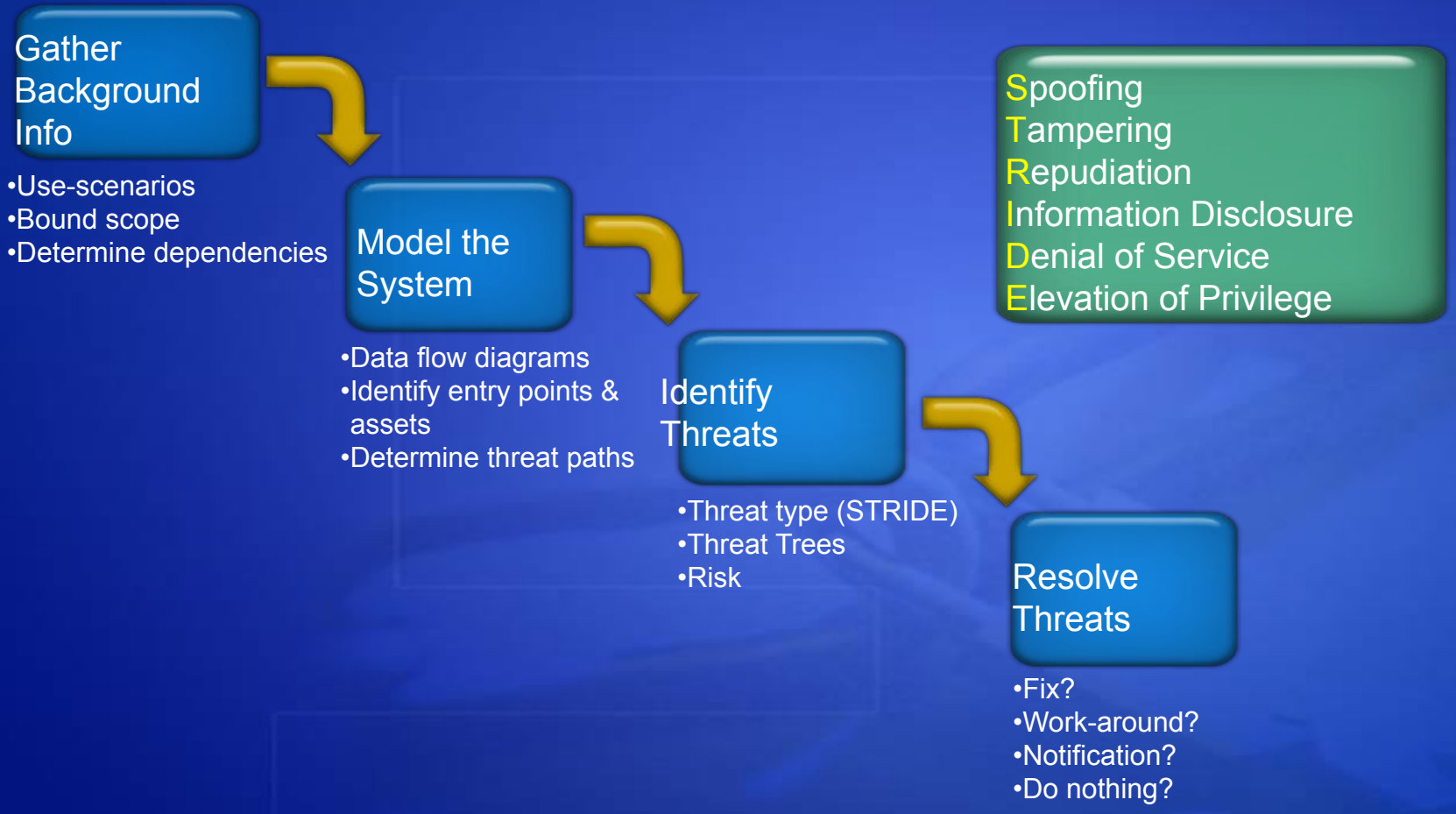
★ What will a bad guy do to your software/system?

Threat Analysis

Threat Analysis

- Secure software starts with understanding the threats
- Threats are not vulnerabilities
- Threats live forever, they are the attacker's goal(s)

A Threat Modeling Process

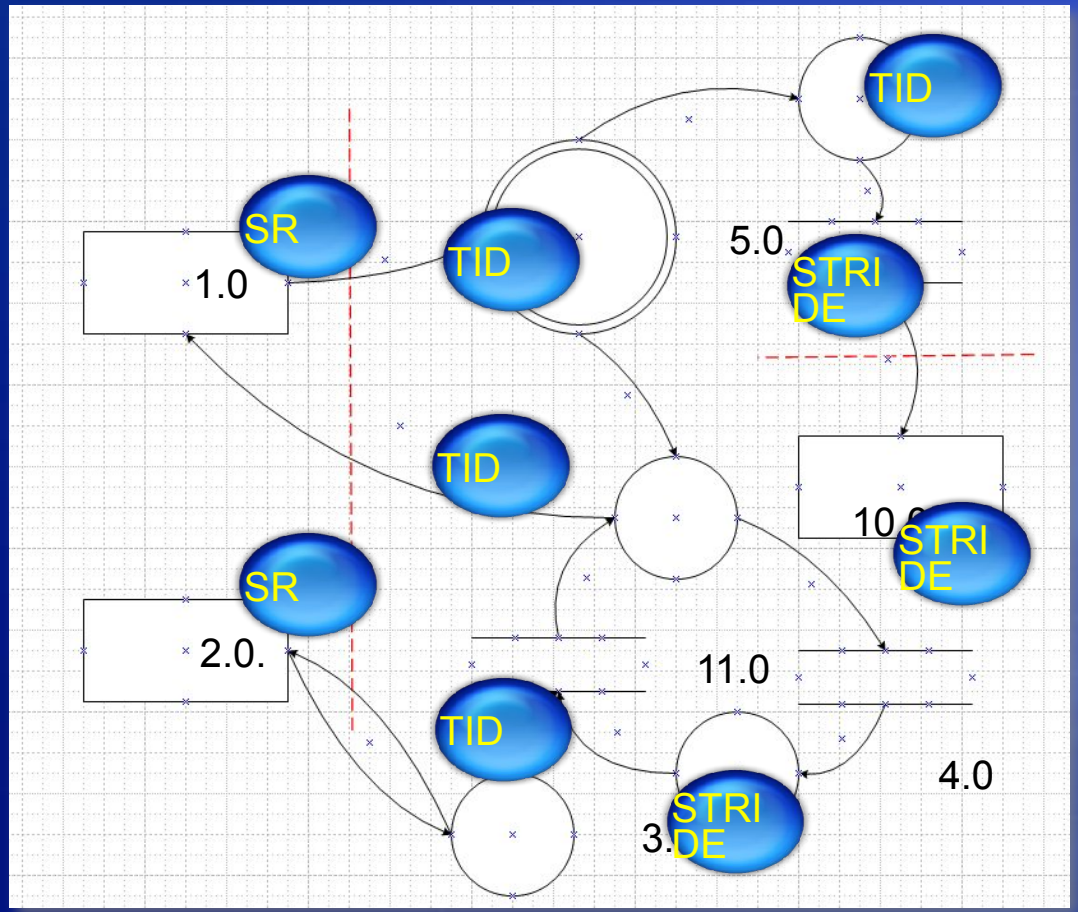


Threats in Software/System

- ★ Spoofing : ปลอมตัว
- ★ Tampering : แฮะเปลี่ยน
- ★ Repudiation : ปฏิเสธการกระทำของตัวเอง
- ★ Information Disclosure : เปิดเผยข้อมูล
- ★ Denial of Services : ทำงานไม่ได้
- ★ Elevation of Privilege : ยก Privilege ขึ้นสูง เป็นสัมส์

Picture taken from http://www.threatgeek.com/2013/11/threattoons-trick-or-treat.html

Determining Threat Types



6.0 TID

7.0

8.0 TID

9.0

Each element in the DFD is susceptible to one or more threat types

DFD Elements are Threat Targets
A "Work list"

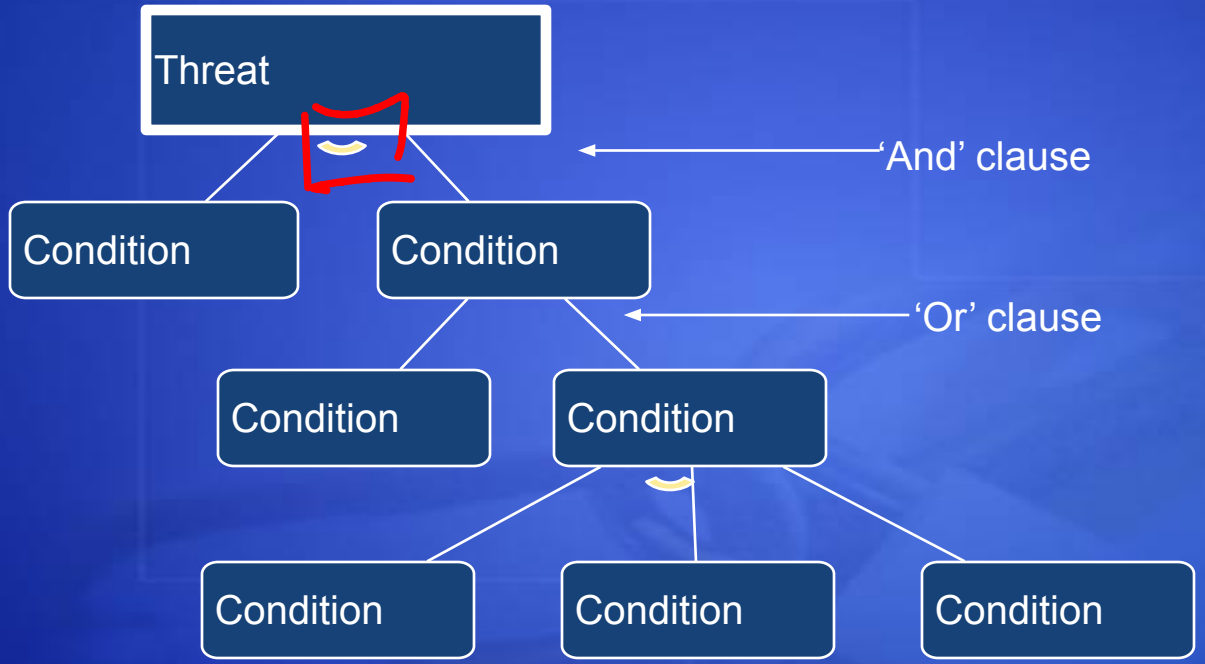
Data Flow	S	T	R	I	D	E
1 → 5		✓		✓	✓	
5 → 6		✓		✓	✓	
6 → 7		✓		✓	✓	
7 → 8		✓		✓	✓	
Data Store						
7		✓		✓	✓	
9		✓		✓	✓	
11		✓		✓	✓	
Interactor						
1	✓		✓			
2	✓		✓			
8	✓		✓			
Process						
3	✓	✓	✓	✓	✓	✓
4	✓	✓	✓	✓	✓	✓
5	✓	✓	✓	✓	✓	✓
6	✓	✓	✓	✓	✓	✓
10	✓	✓	✓	✓	✓	✓

Each ✓ is a potential threat to the system.

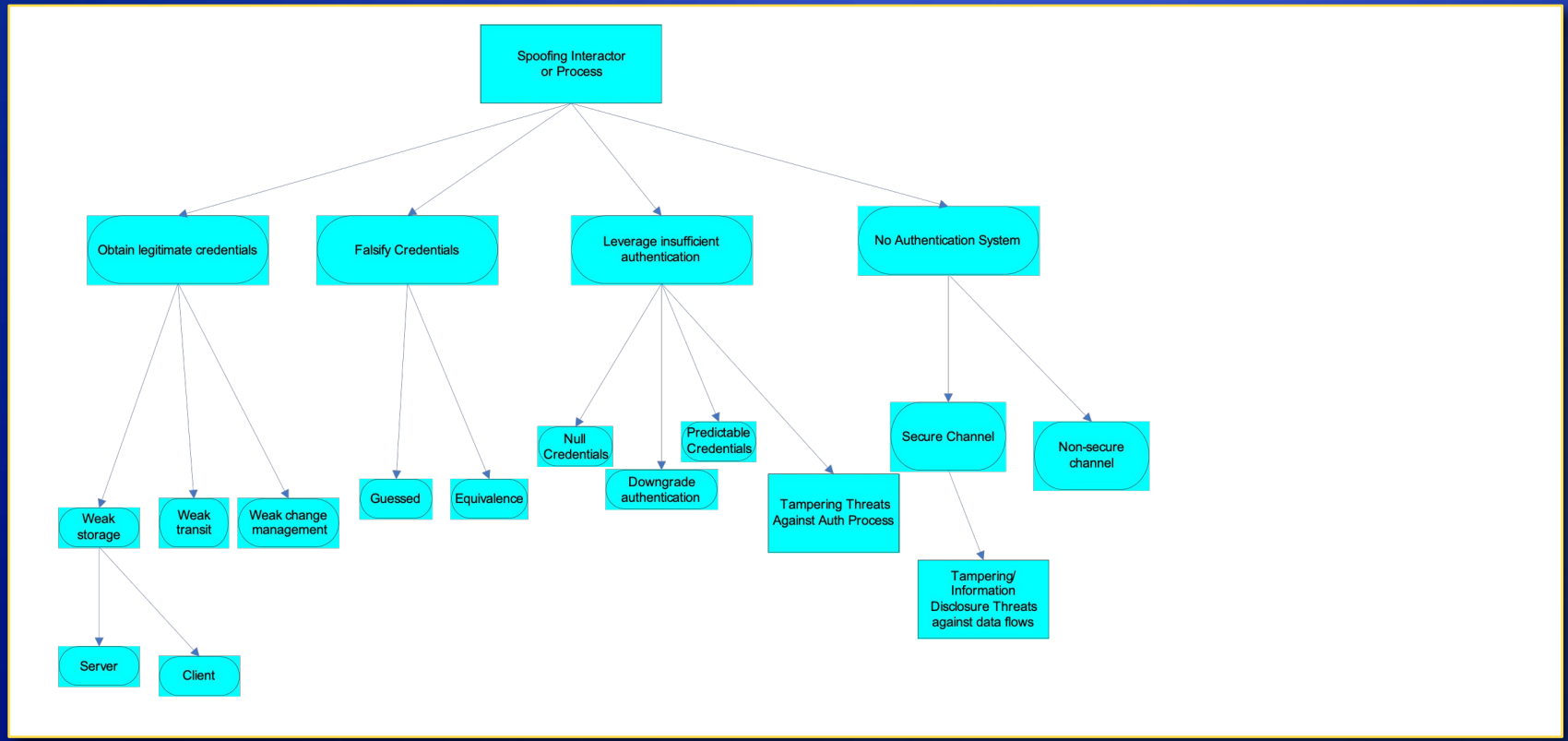
Each threat is governed by the conditions which make the threat possible

Threat Tree Format

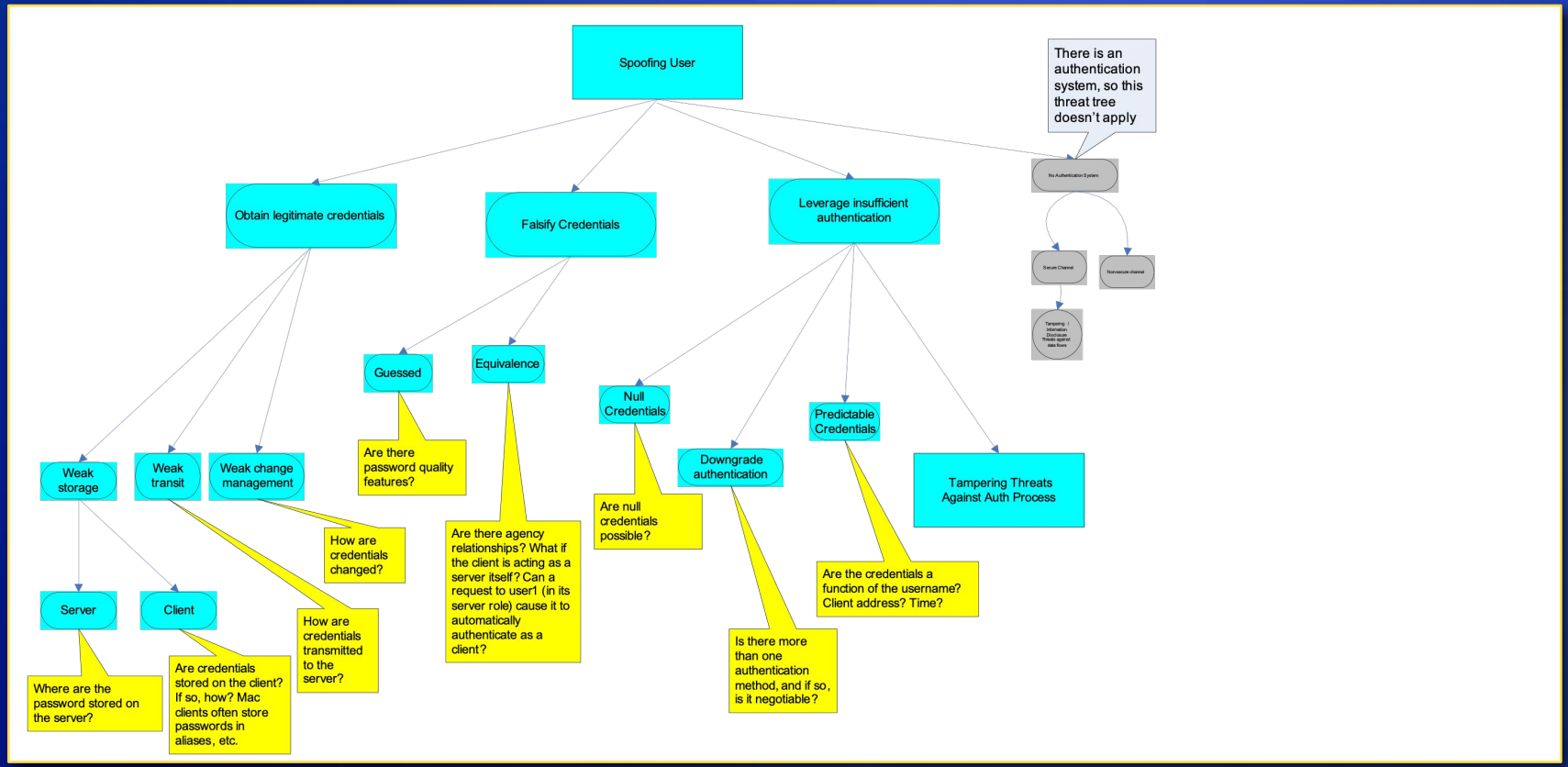
Attack tree



Threat Tree Pattern Examples
Spoofing



Threat Tree Pattern Examples
Thinking Like a Security Pro!



Calculating Risk with Numbers

- DREAD etc.
- Very subjective
- Often requires the analyst be a security expert
  - On a scale of 0.0 to 1.0, just how likely is it that an attacker could access a private key?
- Where do you draw the line?
  - Do you fix everything above 0.4 risk and leave everything below as "Won't Fix"?

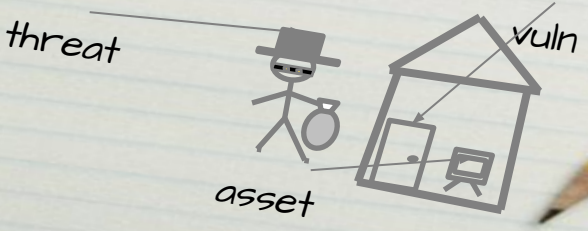
Mitigation Techniques

Threat	Mitigation Feature
Spoofing	Authentication
Tampering	Integrity
Repudiation	Nonrepudiaton
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

Attend “Secure Design Principles”

Threat Model Checklist

- [x] No design is complete without a threat model!
- [x] Follow anonymous data paths
- [x] Every threat needs a security test plan
- [x] Check all information disclosure threats - are they privacy issues?
- [x] Be wary of elevated processes
- [x] Use the threat modeling tool



Input Validation

“All input is evil, until proven otherwise.”

–Michael Howard
Chief Security Office, Microsoft

Why “Input does matter” ?

★ AAA talks about “Who do What and When?” without any data involves.

★ DATA can be harmful.

SELECT * FROM car WHERE pl

Sample Input

- SQL Injection
- Cross-Site scripting
- Buffer-Overflow Attacks

Why It's Wrong

```
sqlstring="SELECT HasShipped" +
        " FROM Shipment WHERE ID='" + Id + "'";
```

Good Guy

Enter a Shipping ID: 1001

```
SELECT HasShipped
FROM Shipment
WHERE ID='1001'
```

Not so Good Guy

Enter a Shipping ID: 1001' or 2>1 --

```
SELECT HasShipped
FROM Shipment
WHERE ID= '1001' or 2>1 -- '
```

Why It's Wrong

```
sqlstring="SELECT HasShipped" +
        " FROM Shipment WHERE ID='" + Id + "'";
```

Really Bad Guy

Enter a Shipping ID: 1001' drop table orders –

```
SELECT HasShipped
FROM Shipment
WHERE ID= '1001' drop table orders -- '
```

Downright Evil Guy

Enter a Shipping ID: 1001' exec xp_cmdshell('net user add URHacked Pa$sw0rd') –

Enter a Shipping ID: 1001' exec xp_cmdshell('net localgroup admins URHacked /add') –

```
SELECT HasShipped
FROM Shipment
WHERE ID= '1001' exec xp_cmdshell('...') -- '
```

Don’t do this.

Listing 3. A Simple “Harmful SQL Commands” Filter

```
<?php
function filter_sql($input) {
     $reg = "(delete)|(update)|(union)|(insert)";
     return(eregi_replace($reg, "", $input));
}
?>
```

LINUX® JOURNAL

DELDELETEETE

Input Validation

- ★ Validate all input.
- ★ Type Checks (e.g. numeric only)
- ★ Length Checks
- ★ Range Checks (e.g. A-z)
- ★ Format Checks (e.g. email)

The security testing is beyond the scope of these slides.

Conclusion

★ Every design should be secure from the ground up.

End of Chapter 10