



# Vulnerability Analysis

---



# Vulnerability

---

- A defect (bug) in one system component or in the way the component is used
- By exploiting the bug, a threat agent can fire an unexpected behavior of the component
- The behavior allows the agent to violate the security policy = difference between bugs and vulnerabilities



# Vulnerability vs bug

---

- A bug may not result in a behavior that violates the security policy
- A bug that results in such a behavior is a vulnerability

=

any vulnerability is a bug but not the other way around



# Taxonomies

---

- Several vulnerability taxonomies have been defined and may be adopted
- Each taxonomy has a goal (location discovery, evaluate the effects ...)
- Before applying a taxonomy we need to understand whether such a taxonomy satisfies with our goals



# Location of the vulnerability

---

- Actions that are executed
  - Procedural
- People executing the action
  - Organization
- Hardware or software tools
  - ICT tools that are used



# Some examples

---

- Action

- A password communicated in an envelope that is not sealed

- People

- Several administrators for the same machine
- Task assigned to people that are not trained

- Tool

- A password transmitted in clear on a network
- No bound controls on a vector index



# Taxonomy on tool vulns

---

A further classification, useful but not very rigorous

- Specification
  - A tool that is more general than required (more functions, more parameters ...)
- Implementation
  - A coding error in the program of the tool
- Structural
  - The anomalous behavior arises when several components are integrated



# Examples

---

- Specification = programming-in-the-large
  - A library is used that include more functions than those that are required
  - If someone succeeds in invoking some of the “useless” functions, anomalous behaviors may arise
  - Debug functions
  - Code reuse may introduce in a system some vulns in the code that is reused





# Examples

---

- Implementation =
  - Well behaved input
  - No control on input parameters
  - Data and program confusion = jump into a data structure = stack /buffer/heap overflow
- These vulnerabilities strongly depend upon the native control in the language type system and in the language run time system
  - = no overflows with strong data types



# Examples

---

- Structural: due to the composition of several components that are
  - Correct in isolation
  - Uncorrect when composed into a system
- Problems in the TCP/IP stack
- Some components delegate security checks to other ones, their correctness depends upon checks in other components ie they trust the other components



# Another classification

---

- It considers an attack that exploit the vulnerability
  - Who can implement the attack
    - Those who own a local account
    - Those who can interact with the machine
    - ...
  - What can be achieved by the attack



# Searching for vulns

---

- Any system can be seen as a composition of standard and specialized (not standard) components
- Most vulns and exploits for standard components are well known
- The search should focus on
  - Not standard components
  - Structural vulns due to the composition of standard components with not standard ones
  - Vulns in standard components are the last to search



# Vulns and vulnerability scanning

---

- A vulnerability scanner is a tool that returns a set of vulns for each computer node in a network
- The scanner identifies the OS and the applications running on the node through a fingerprinting algorithms
- Then it accesses a database that maps each OS and application into a set of of public vulns
- Vulnerability scanning is the easiest implementation of a vulnerability analysis



# Fingerprint

---

- The main mechanism to identify the OS and the application is the transmission of IP packets that violates the specifications
- All the applications and the OSes reply in a standard way to a standard packet
- Each OS and application has its own reaction to a wrong packet that violates the TCP/IP specification
- Several packets may be required to solve any ambiguity among distinct OSes/Components



# Fingerprint and mapping

---

- The applications are discovered by analyzing the open ports in the node
- After discovering the OS and the applications each of them is mapped into a set of vulnerabilities
- The mapping is implemented by accessing public databases that store any vulnerabilities of the OS or of an application
- This strategy can be defeat by running application on not standard port



# False and true positive

---

- The scanner will signal a vulnerability even if the component has been patched
- This is what is called a false positive
- The only strategy to distinguish false and true positive is to actually implement an attack that exploits the vulnerability (breach and simulation tools)
- Not always possible on production systems





# Security is not a boolean world

---

		Existence (gold standard)	
		Y	N
Test outcome	Y	True positive	False Positive
	N	False negative	True Negative

The problem arises anytime we can only deduce the existence of an object from some symptoms and do not have a direct access to it

# Not a boolean world

		Condition (as determined by "Gold standard")		
		Condition positive	Condition negative	
Test outcome	Test outcome positive	<b>True positive</b>	<b>False positive</b> (Type I error)	Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$
	Test outcome negative	<b>False negative</b> (Type II error)	<b>True negative</b>	Negative predictive value = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$
		Sensitivity = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	Specificity = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	<b>Accuracy</b>

$$\text{accuracy} = \frac{\text{number of true positives} + \text{number of true negatives}}{\text{number of true positives} + \text{false positives} + \text{false negatives} + \text{true negatives}}$$



# Vulnerabilities in non standard components

---

- We consider some tools to search for vulnerabilities in non standard component
- Not always we have the source code of the component available
- Analysis is based on how a module react to some inputs



# Vulnerabilities in non standard components

---

- Web scanner = a tool to search for vulnerabilities in web application
- The term scanner is misleading, essentially it focuses on well known vulnerabilities and attacks
- Some scanners also apply fuzzing (see later)



# Tainting analysis

---

- A preliminary static analysis of the source code that computes the set of program variables that may receive an input variable and so could be overflowed
- It returns a larger set than the actual one, worst case (false positive strike back)
- It can be improved by taking into account the procedure to copy the input value



# Tainting analysis

---

```
if x (y=input)
else (y=z);
w=y
```

- A tainting analysis tell us that w may have been tainted with an input value

```
if x (y=input)
else (y=z);
copy (w, y)
```

- If copy checks the length of y before copying it into w, tainting but less danger



# Discovering overflow

---

- There may a vulnerability anytime an input value is copied into a procedure parameter without checking its length (that is surely a bug)
- There is a vulnerability if the procedure is executed with a large set of rights (bug that is a vulnerability)
- A simple tainting analysis is not sufficient (false positive)



# Fuzzing and fuzzer

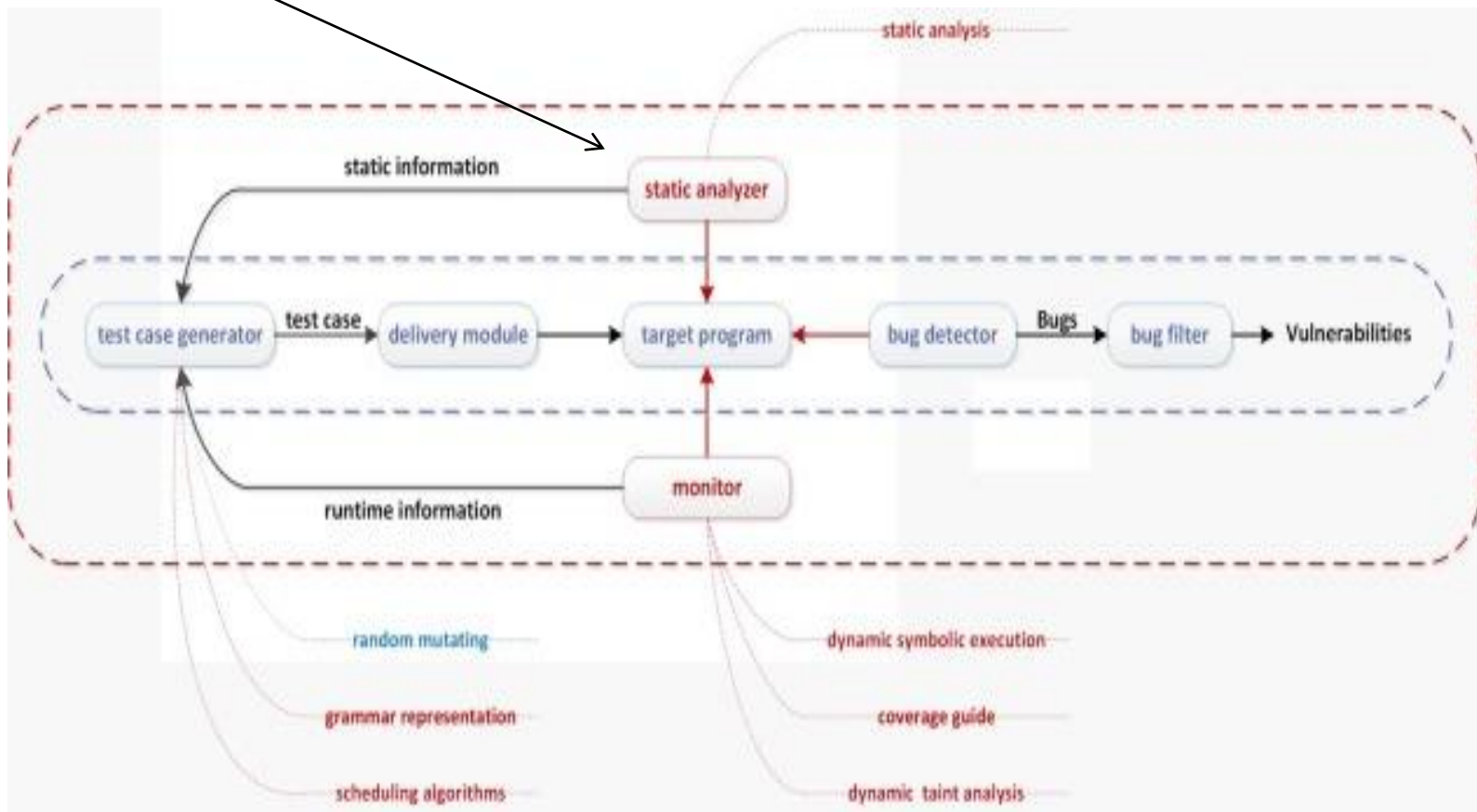
---

- Fuzzing is a search technique that can be applied even if the source code of a module is not available. According to Google and Microsoft it can discover more than 80% of vulnerabilities
- The basic idea is to send malformed input to the module
- If the module crash, then the input is not controlled and a vulnerability is possible
- A fuzzer automates this strategy by testing a huge number of inputs even in parallel



# Fuzzer Architecture

If not possible=black box fuzzing





# Mutation Based Fuzzing

---

- Little or no knowledge of the structure of the inputs is assumed
- Anomalies are added to existing valid inputs
- Anomalies may be completely random or follow some heuristics
- Requires little or no set up time
- Dependent on the inputs being modified
- May fail for protocols with checksums, those which depend on challenge response, etc.
- Example Tools : Taof, GPF, ProxyFuzz, Peach Fuzzer, etc.



# Mutation Based Example: PDF Fuzzing

---

- Google .pdf (lots of results)
- Crawl the results and download lots of PDFs
- Use a mutation fuzzer = introducing small changes to existing inputs that may still keep the input valid, yet exercise new behavior:
- Grab the PDF file
- Mutate the file
- Send the file to the PDF viewer
- Record if it crashed (and the input that crashed it)

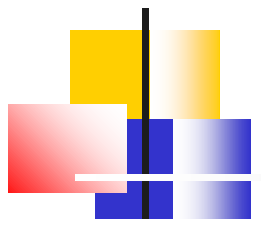


# Generation Based Fuzzing

---

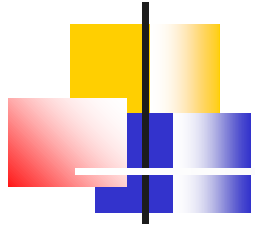
- Test cases are generated from some description of the format: RFC, documentation, etc.
- Anomalies are added to each possible spot in the inputs
- Knowledge of protocol should give better results than random fuzzing
- Can take significant time to set up
- Examples
- SPIKE, Sulley, Mu-4000, Codenomicon, Peach Fuzzer, etc...









# Example Specification for ZIP file



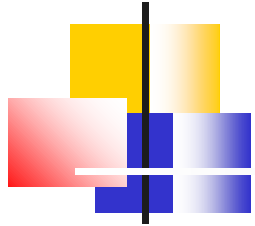
```
1 <!-- A. Local file header -->
2 <Block name="LocalFileHeader">
3   <String name="lfh_Signature" valueType="hex" value="504b0304" token="true" mut
4   <Number name="lfh_Ver" size="16" endian="little" signed="false"/>
5   ...
6   [truncated for space]
7   ...
8   <Number name="lfh_CompSize" size="32" endian="little" signed="false">
9     <Relation type="size" of="lfh_CompData"/>
10  </Number>
11  <Number name="lfh_DecompSize" size="32" endian="little" signed="false"/>
12  <Number name="lfh_FileNameLen" size="16" endian="little" signed="false">
13    <Relation type="size" of="lfh_FileName"/>
14  </Number>
15  <Number name="lfh_ExtraFldLen" size="16" endian="little" signed="false">
16    <Relation type="size" of="lfh_FldName"/>
17  </Number>
18  <String name="lfh_FileName"/>
19  <String name="lfh_FldName"/>
20  <!-- B. File data -->
21  <Blob name="lfh_CompData"/>
22 </Block>
```

# Mutation vs Generation



Mutation-based	Super easy to setup and automate 	Little to no protocol knowledge required 	Limited by initial corpus 	May fail for protocols with checksums, or other complexity 
Generation-based	Writing generator is labor intensive for complex protocols 	have to have spec of protocol (frequently not a problem for common ones http, snmp, etc...) 	Completeness 	Can deal with complex checksums and dependencies 

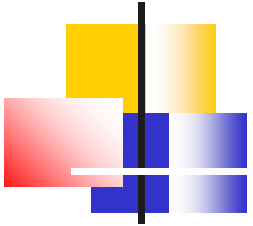
# White box vs. black box fuzzing



- Black box fuzzing: sending the malformed input without any verification of the code paths traversed
- White box fuzzing: sending the malformed input and verifying the code paths traversed. Modifying the inputs to attempt to cover all code paths. Drilling = optimized search of malformed inputs

	Effort	Code coverage	Defects Found
black box + mutation	10 min	50%	25%
black box + generation	30 min	80%	50%
white box + mutation	2 hours	80%	50%
white box + generation	2.5 hours	99%	100%

# Evolutionary Fuzzing



- Attempts to generate inputs based on the program response
  - Autodafe
  - Prioritizes test cases based on which inputs have reached dangerous API functions
- Evolutionary Fuzzing System
  - Generates test cases based on code coverage metrics
  - This technique is still in the alpha stage :)





# Fuzzing and fuzzer: Phases

---

**Identify target**

**Identify inputs**

**Generate fuzzed data**

**Execute fuzzed data**

**Monitor for exceptions**

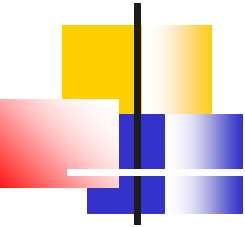
**Determine exploitability**



# Fuzzing and fuzzer: Phases and tools

---

- Command line arguments
  - Environment variables
    - Sharefuzz ([www.immunitysec.com](http://www.immunitysec.com))
- Web applications
  - WebFuzz
- File formats
  - FileFuzz
- Network protocols
  - SPIKE ([www.immunitysec.com](http://www.immunitysec.com))
- Memory
- COM Objects
- COMRaider
- Inter-Process Communication (IPC)



# FileFuzz

**FileFuzz** [Window Title]

File Edit

File Type:   Strip File Extension  Exclude Target Directory

Source File

Target Directory

Target

Byte(s) to Overwrite:

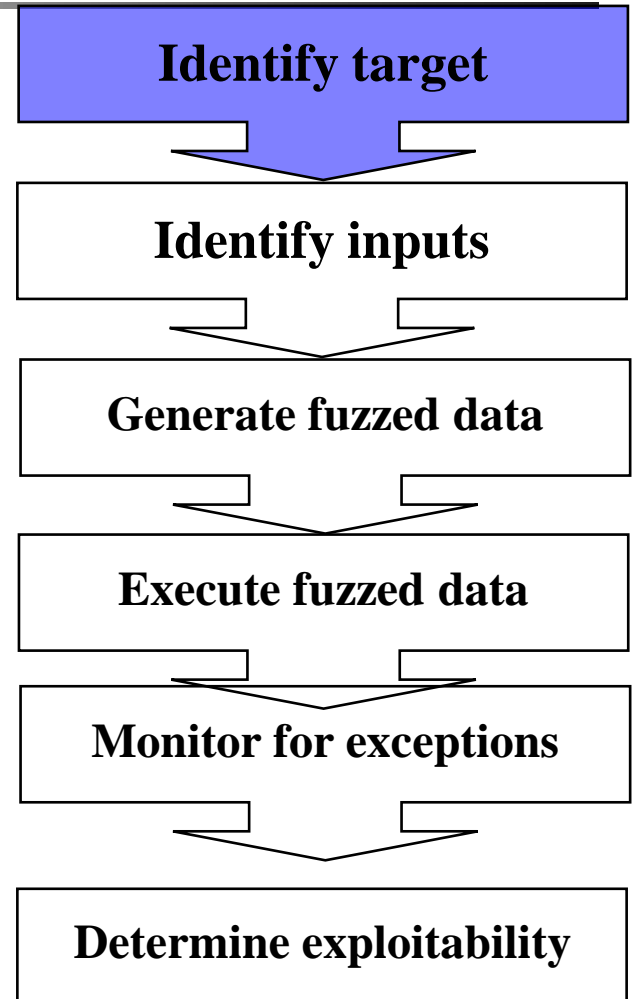
Scope

All Bytes  
 Range Start  Finish   
 Depth Location   
 Match Find   
Replace

```
File Name: BKF
File Description: Windows Backup File
Source File: test.bkf
Source Directory: C:\Program Files\FileFuzz\Attack\
Application Name: ntbackup.exe
Application Description: Windows Backup Utility
Application Action: open
Application Launch: C:\WINDOWS\system32\ntbackup.exe
Application Flags: (0)
Target Directory: c:\fuzz\bkf\
```

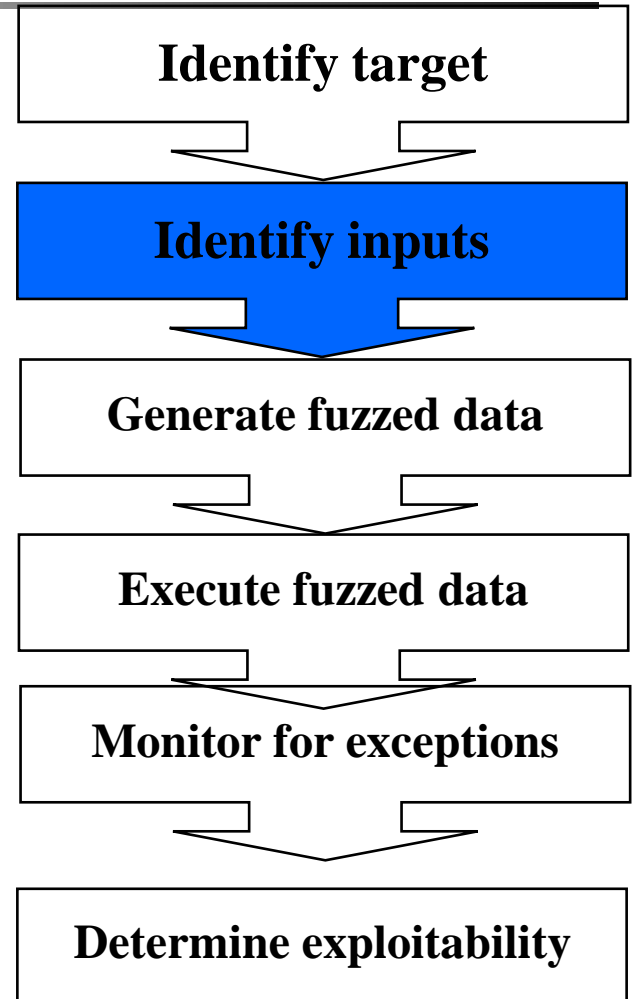
# FileFuzz – Identify Target

- Application vs. file type
  - One file type vs multiple targets
- Vendor history
  - Past vulnerabilities
- High risk targets
- Default file handlers
  - Windows Explorer
  - Windows Registry
- Commonly traded file types
  - Media files
  - Office documents
  - Configuration files



# FileFuzz – Identify Inputs

- Proprietary vs. open formats
- Vendor documents
  - Wotsit.org
- Google
- Binary files
- e.g. images, video, audio, office documents, etc.
- Headers vs. data
- Text files
  - e.g. \*.ini, \*.inf, \*.xml
- Name/value pairs



# FileFuzz – Generate Fuzzed Data

- Binary files

- Breadth (All or Range)

- Identify potential weaknesses

```
FF FF FF FF 00 00 DB FE 0B 00 C5 00 00 01 E8 03 ; yyyÿ..Ûp..Å...è.  
D7 FF FF FF FF 00 00 DB FE 0B 00 C5 00 00 01 E8 03 ; *ÿÿÿÿ.Ûp..Å...è.  
D7 CD FF FF FF FF DB FE 0B 00 C5 00 00 01 E8 03 ; *ÿÿÿÿÛp..Å...è.
```

- Depth

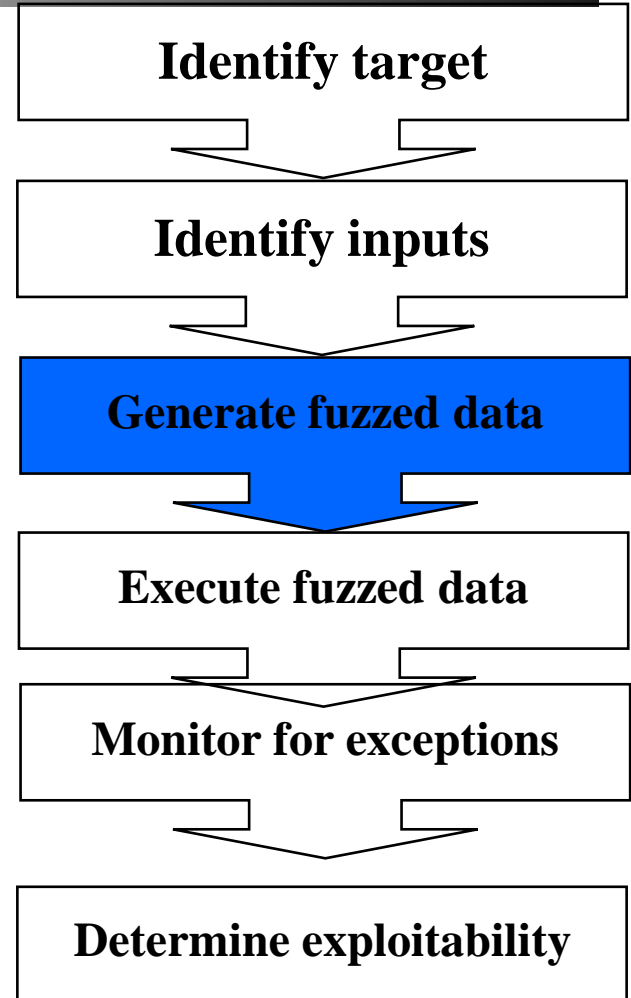
- Determine level of control/influence

```
D7 CD FD 9A 00 00 DB FE 0B 00 C5 00 00 01 E8 03 ; *íÿš..Ûp..Å...è.  
D7 CD FE 9A 00 00 DB FE 0B 00 C5 00 00 01 E8 03 ; *ípš..Ûp..Å...è.  
D7 CD FF 9A 00 00 DB FE 0B 00 C5 00 00 01 E8 03 ; *íÿš..Ûp..Å...è.
```

- Text Files

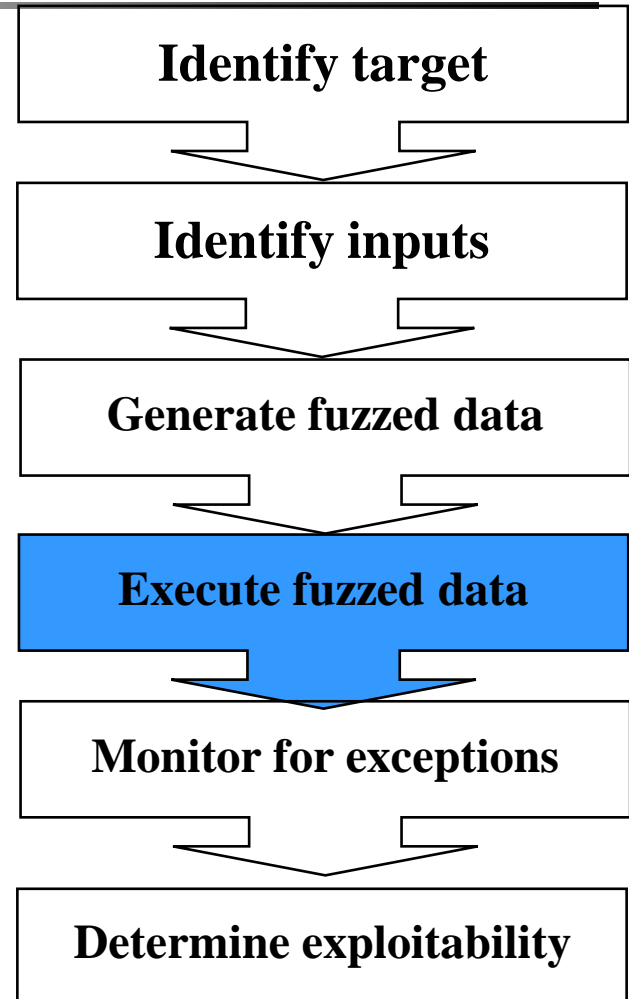
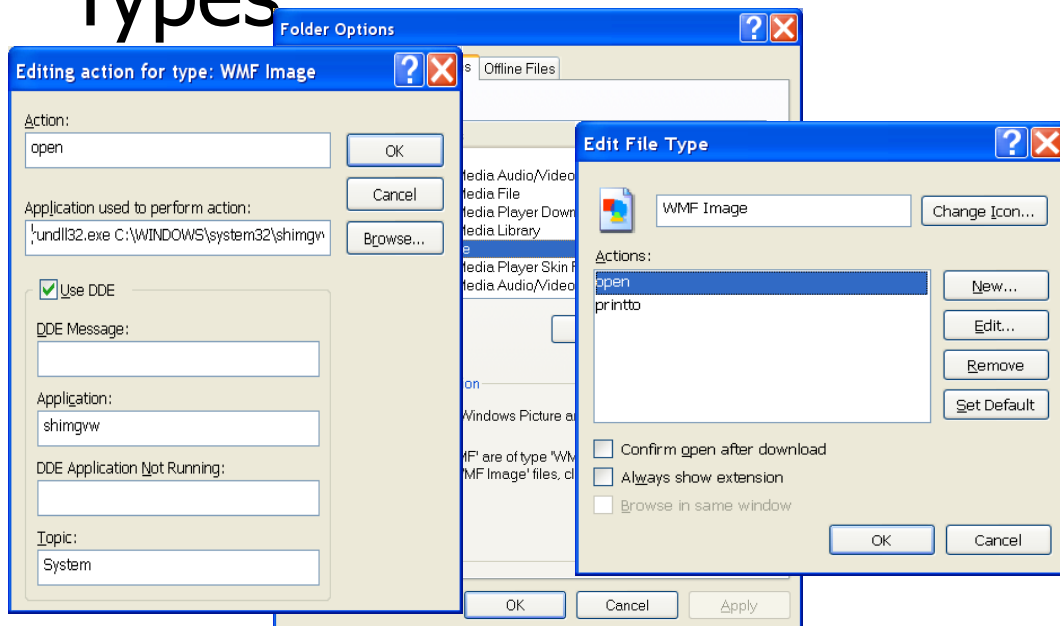
- name = value

```
file_size = 10  
file_size = AAAAA  
file_size = AAAAAAAAAA
```



# FileFuzz – Execute Fuzzed Data

- Command line arguments
  - Windows explorer
- Tools... Folder Options... File Types



# FileFuzz – Monitor for Exceptions

- **Visual**

- Error messages
- Blue screen

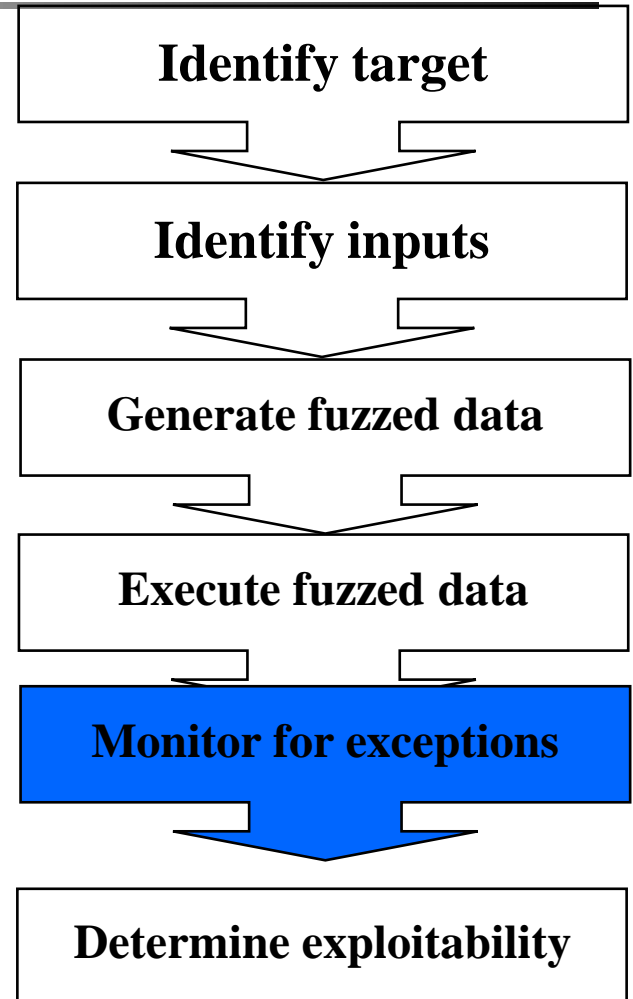
- **Event logs**

- System logs
- Application logs

- **Debuggers/Reverse Engineering**

- **Return codes**

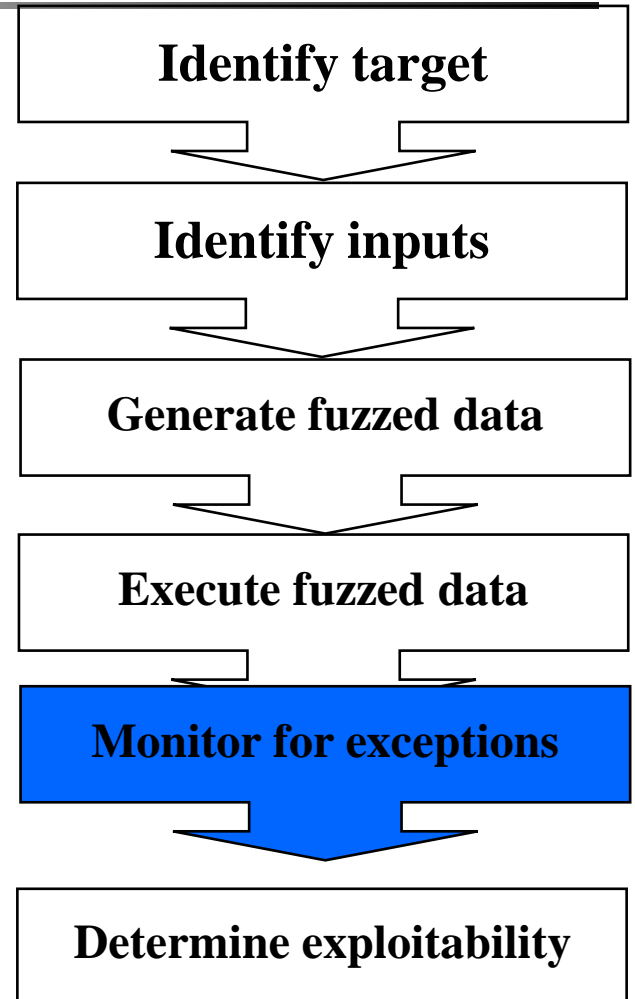
- **Debugging API**





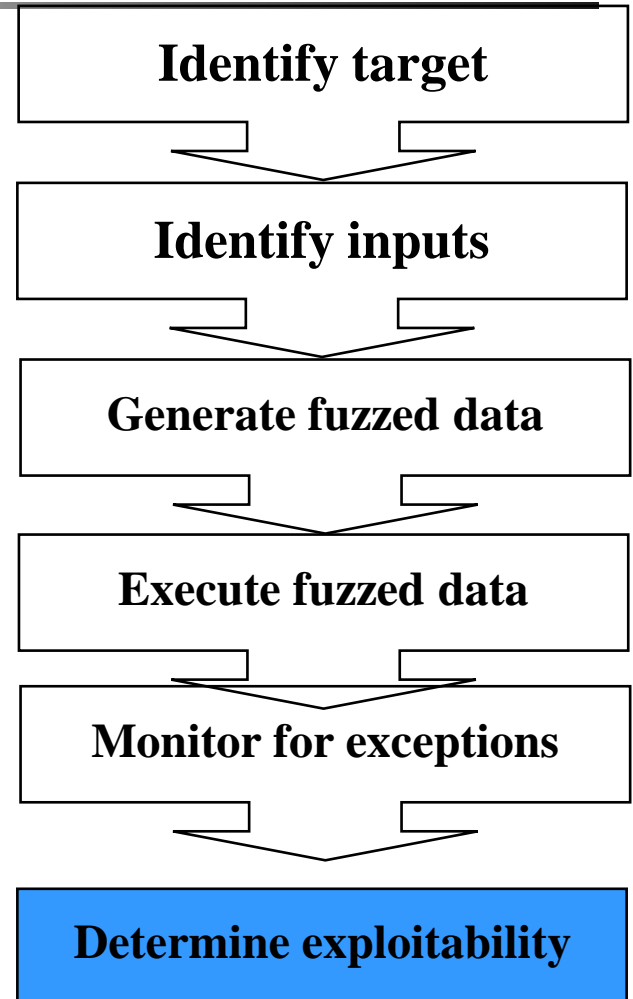
# FileFuzz – Monitor for Exceptions

- Execute
  - Automated and repeated
- Monitor
  - Library – libdasm
- Capture
  - Memory location
  - Registry values
  - Exception type
- Kill
  - Set timeout



# FileFuzz – Determine Exploitability

- Skills = Disassembly + Debugging
- Vulnerability types
  - Stack overflows
  - Heap overflows
  - Integer handling
  - Overflows
  - Signedness
  - DoS
  - Out of bounds reads
  - Infinite loops
  - NULL pointer dereferences
  - Logic errors
  - Windows WMF vulnerability (MS06-001)
  - Format strings
  - Race conditions



# WebFuzz

The screenshot shows the WebFuzz application window. At the top, there are input fields for Host (www.iddefense.com), Port (80), Timeout (Milliseconds) (5000), and a Request button. Below this is the 'Request Headers' section, which displays the following text:

```
GET / HTTP/1.1
Accept: /*
Accept-Language: en-us
Pragma: no-cache
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; InfoPath.1)
Host: localhost
Proxy-Connection: Keep-Alive
```

Below the request headers is the 'Responses' section, which contains a table with the following data:

No.	Status	Host	Request
0	200	www.iddefense.com	GET / HTTP/1.1

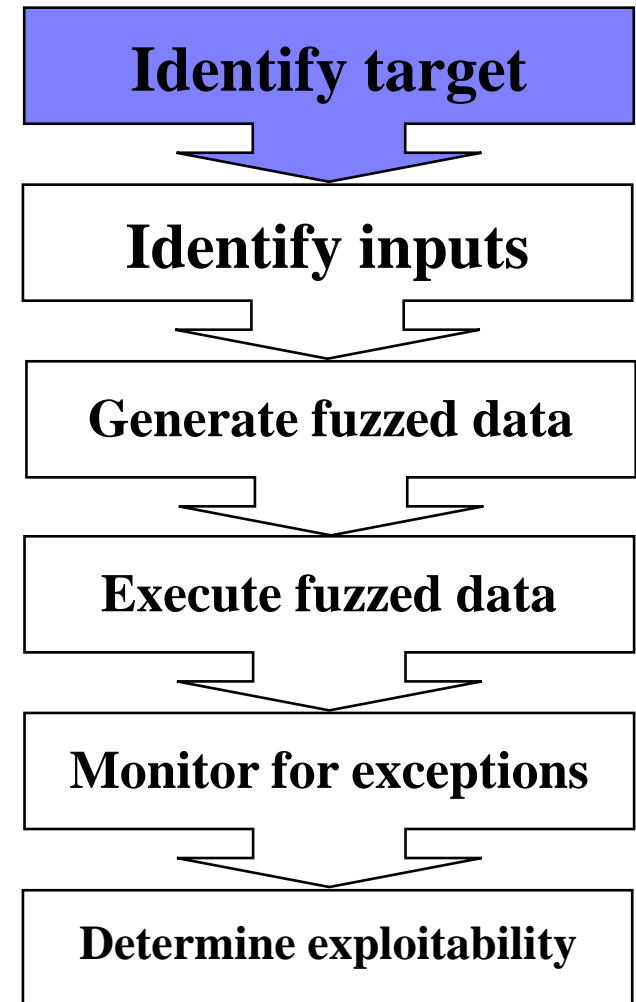
At the bottom of the window, there are three tabs: 'Raw Request', 'Raw Response', and 'HTML'. The 'HTML' tab is selected, showing the following content:

```
HTTP/1.1 200 OK
Date: Fri, 21 Apr 2006 22:24:00 GMT
Server: Apache/2.0.52 (Red Hat)
X-Powered-By: PHP/5.0.4
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1

1fb3
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
  <meta HTTP-EQUIV="expires" CONTENT="0">
  <meta HTTP-EQUIV="pragma" CONTENT="no-cache">
  <meta name="keywords" content=" managed security services, network-based security threats, security vulnerability, protect critical data, infrastructure, cyber attacks, security experts, information security, managed firewall, managed security, managed vpn, penetration testing, network security, potential cyber threats, new malicious code, zero-day exploits, hacker groups, cyber crime, cyber terror, advanced warning, threat protection, critical infrastructure, verisign, iddefense, worm, virus, phishing. mss" />
  <meta name="description" content="VeriSign iDefense services deliver comprehensive, actionable intelligence regarding network-based security threats and vulnerabilities which can help organizations proactively protect critical data and infrastructure from attacks. VeriSign iDefense Security Intelligence Services are an important component of VeriSign's Managed Security Services [MSS].">
  <title>VeriSign iDefense Security Intelligence Services -- Managed Security Services and Information Security for Government and Fortune 500 Organizations // VeriSign iDefense</title>
  <link rel="shortcut icon" href="/favicon.ico" />
  <link rel="icon" href="/favicon.ico" />
```

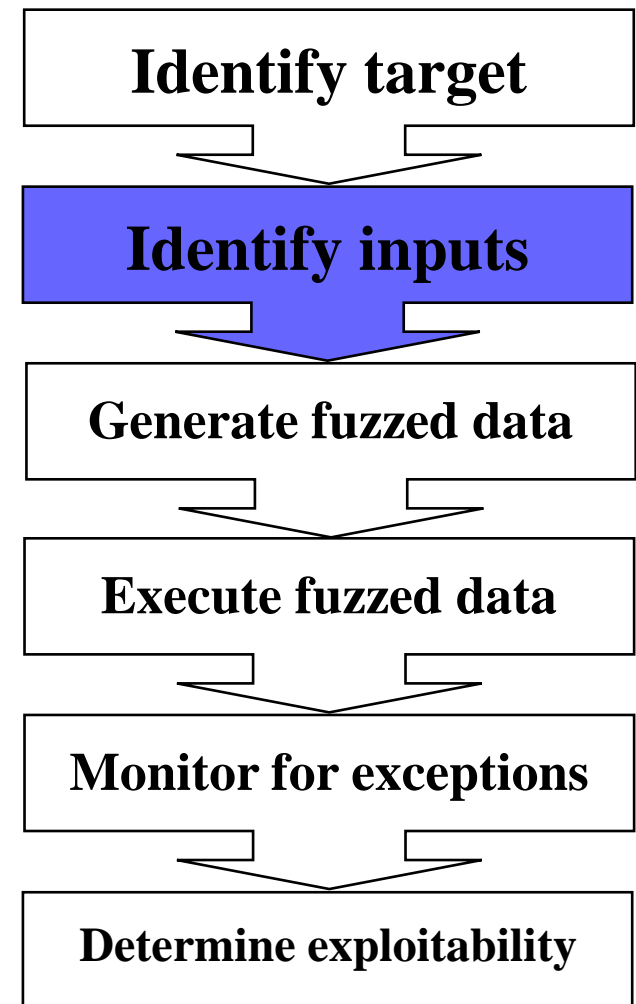
# WebFuzz – Identify Target

- Server vs. Application
- Targeting applications can uncover server vulnerabilities
- Vendor history
- Past vulnerabilities
- High risk targets
- Popular applications
- External applications
- Wikis
- Web mail
- Discussion boards
- Blogs



# WebFuzz – Identify Inputs

- Potential input vectors
  - Method
  - Request-URI
  - Protocol
  - Headers
  - Cookies
  - Post data
- Reconnaissance
  - Web forms
  - Authentication
  - Hidden fields
  - Client side scripting
- Manual Tools
  - Proxies
  - LiveHTTPHeaders
- Automated Tools
  - Spiders



# WebFuzz – Generate Fuzzed Data

## Intelligent fuzzing

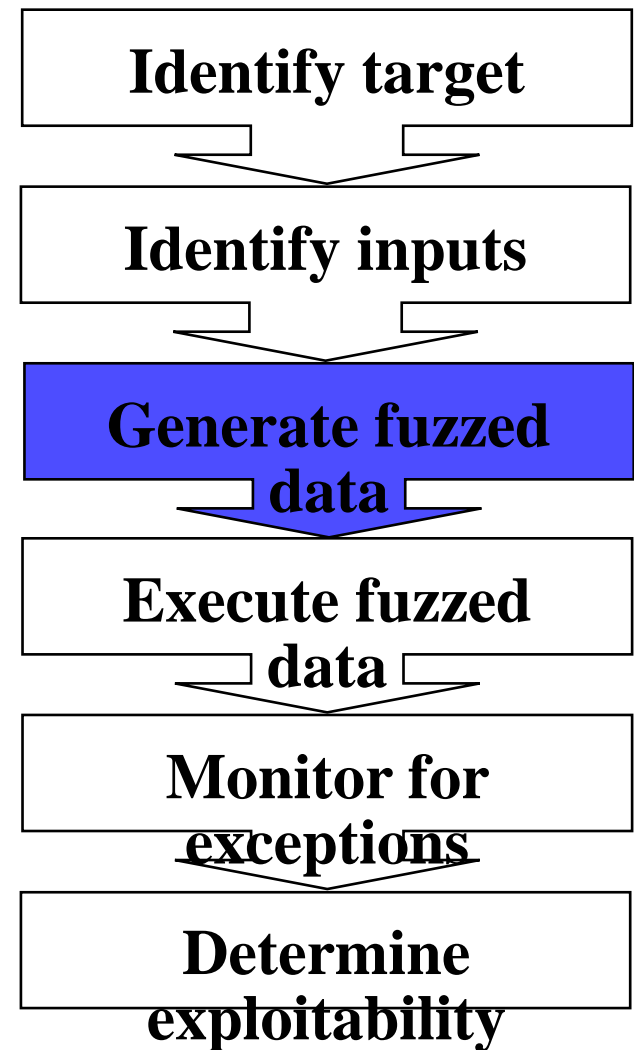
- Start with legitimate web request
- Build template to mutate requests

## Request format

```
[Method] [Request-URI] HTTP/[Major Version].[Minor Version]
[HTTP Headers]
[Post Data]
```

## Fuzz Template

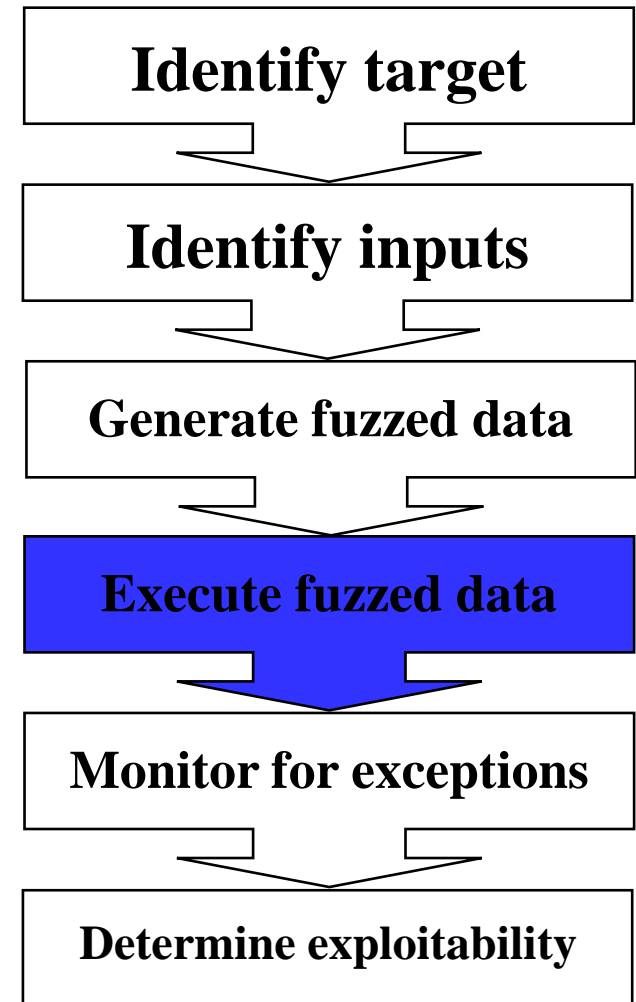
```
[Methods] / [Traversal]/page.html?x=[SQL]&y=[XSS] HTTP/1.1
Accept: */*
Accept-Language: en-us
Pragma: no-cache
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1; InfoPath.1)
Host: [Overflow]
Proxy-Connection: Keep-Alive
```



# WebFuzz – Execute Fuzzed Data

Fuzz classes

- Directory traversal
- Format strings
- Overflow
- SQL Injection
- XSS Injection



# WebFuzz – Monitor for Exceptions

## Execute

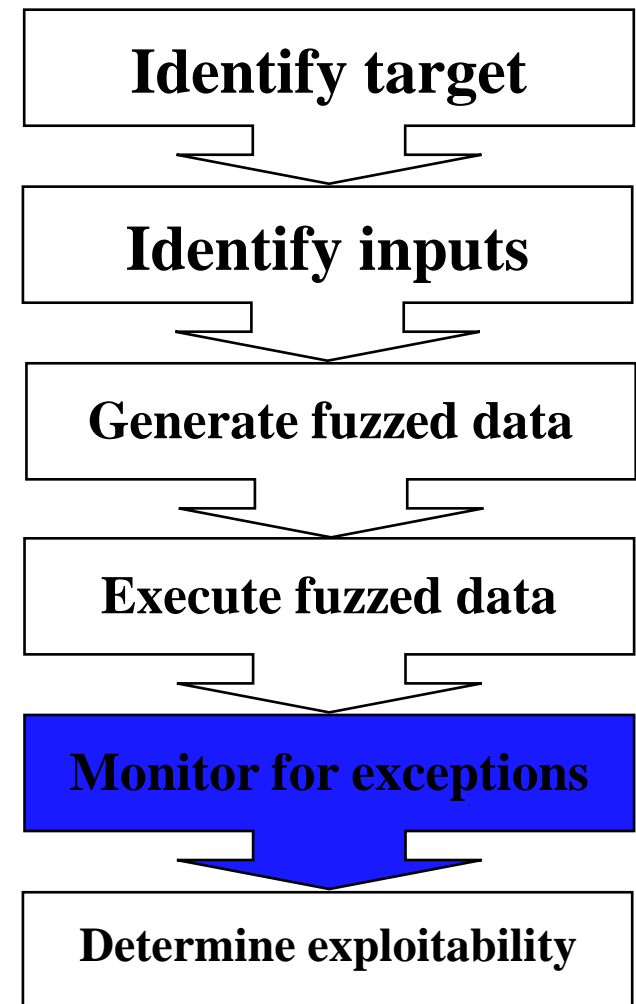
- Automated and repeated

## Monitor

- HTML response
- Error messages
- Raw response
- User input
- Status codes

## Kill

- Set timeout





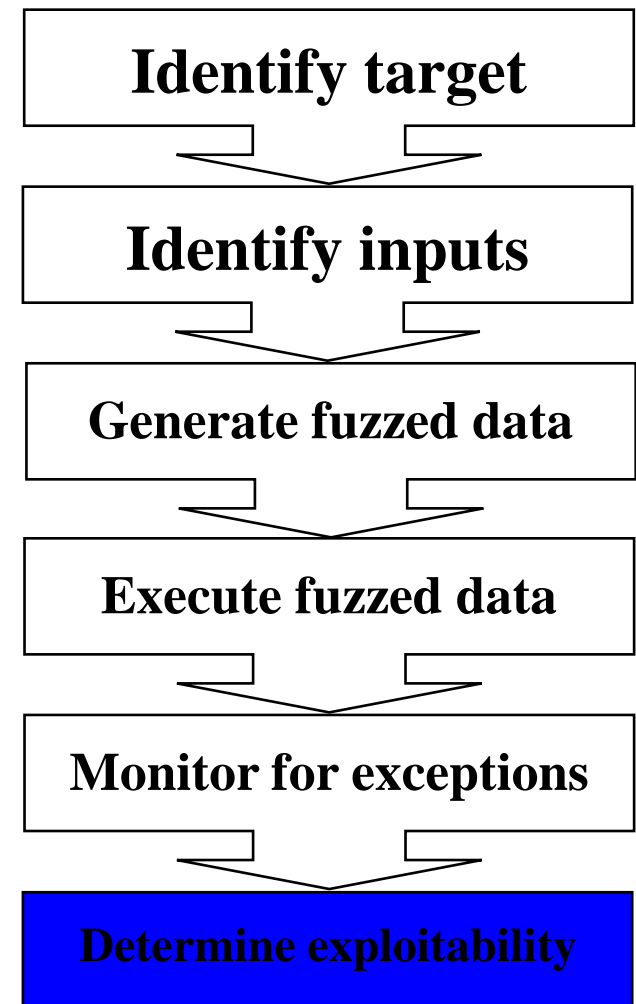
# WebFuzz – Determine Exploitability

## Skills

- HTTP
- HTML
- Client side scripting
- SQL

## Vulnerability types

- Denial of service
- Cross site scripting (XSS)
- SQL injection
- Directory traversal/Weak access control
- Weak authentication
- Weak session management (cookies)
- Buffer overflow
- Improperly supported HTTP methods
- Remote Command Execution
- Remote Code Injection
- Vulnerable Libraries
- HTTP Request Splitting
- Format Strings





# Lessons about Fuzzing

---

- Protocol knowledge is helpful
- Generational beats random
  - better specification make better fuzzers
- Using more fuzzers is better
  - Each one will vary and find different bugs
- The longer you run (typically) the more bugs you'll find
- Guide the process, fix it when it break or fails to reach where you need it to go
- Code coverage can serve as a useful guide

# Interesting Fuzzing Results

## Time to first failure (TTFF)

To measure the relative maturity of a given protocol, the time to first failure (TTFF), or the time to the first instance when that a protocol crashes, is used. It

The average time for first failure of all tests in 2016 was about 1 hour (1.4 hours). This represents a very slight decrease over 2015 (1.7 hours), which might be attributable to a more diverse range of tests run in 2016.

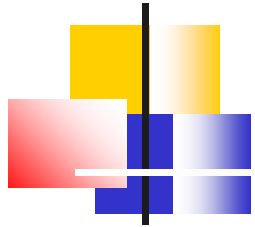
## 5 most mature protocols (average time to first failure, in hours and days)

Protocol suite	2016 time to first failure	2015 time to first failure	2016 test runtime (avg)
TLS Client (Core IP)	9.6 hours	5.4 hours	10.5 hours
SSH2 (Core IP)	6.8 hours	1.9 hours	22.6 hours
SSH Server (Core IP)	4.9 hours	0.7 hours	5.96 hours
802.11 (Core IP)	4.2 hours	1 hour	11.8 hours
ICMPv6 (Core IP)	2.7 hours	1 hour	2.3 days

## 5 least mature protocols (average time to first failure, in minutes and seconds)

Protocol suite	2016 time to first failure	2015 time to first failure	2016 test runtime (avg)
IEC-61850 MMS (ICS)	6.6 seconds	13.2 seconds	5.7 hours
MODBUS PLC (ICS)	1.8 minutes	6 seconds	35 minutes
SNMP Trap (Remote Management)	1.8 minutes	3.6 minutes	57 minutes
MQTT (ICS)	9.9 minutes	2.1 minutes	1.3 hours
DNP3 (ICS)	14 minutes	3.6 minutes	2.6 hours

# Fuzzing Maturity Model



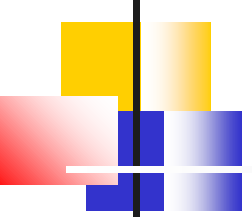
	Types	Test cases	Time (hours)	Instrumentation	Allowed failures	Test harness integration	Processes	Documentation	Other requirements
5 : Optimized	G T	infinite infinite	720	O,A,D	none	Yes	S,C	RR,S,PB,C	U2
4 : Integrated	G T	infinite infinite	168	O,A,D	none		S,C	RR,S,PB,C	
3 : Managed	G T	2,000,000 5,000,000	16	O,A	Tr		S	RR,S,PB	
2 : Defined	G (T)	1,000,000 (5,000,000)	8	O	Tr		S	RR,S,P	
1 : Initial	G/T	100,000	2	O	Tr,As			RR	

0 : Immature

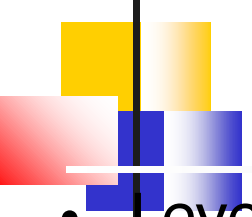
G = Generational fuzzer  
 T = Template fuzzer  
 O = Human observation  
 A = Automated instrumentation  
 D = Developer tooling  
 Tr = Transient failures  
 As = Non-DoS assertion failures

S = Attack surface analysis  
 C = Software component analysis  
 RR = Results and summary  
 P = Test plan  
 B = Document baseline test configuration  
 U2 = Use two different fuzzers per type

# Fuzzing Maturity Model

- 
- 
- Level 0: Immature
    - If no fuzzing has been performed on any attack vector in a target, the target is at FTMM Level 0. If minimal fuzzing has been done, but does not meet the Level 1 requirements, then the target is still at FTMM Level 0.
  - Level 1: Initial
    - Level 1 represents an initial exposure to fuzz testing. Either generational or template fuzzing is used on the known attack vectors of the target, For each tested attack vector, fuzzing should be performed for at least 2 hours or 100,000 test cases, whichever comes first.

# Fuzzing Maturity Model



---

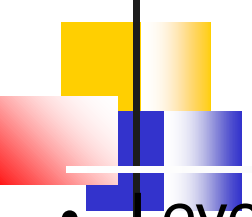
- Level 2: Defined

- The starting point for Level 2 is an attack surface analysis of the target. For each attack vector, a generational fuzzer should be used for 8 hours or 1 million test cases, whichever comes first. If a generational fuzzer is unavailable for an attack vector, a template fuzzer can be used instead, for at least 8 hours or 5 million test cases, whichever comes first.

- Level 3: Managed

- Both generational and template fuzzing must be performed for each attack vector in Level 3. The generational fuzzer must be run for 16 hours or 2 million test cases, whichever comes first, while the template fuzzer must be run for 16 hours or 5 million test cases. Automated instrumentation must be used.

# Fuzzing Maturity Model



---

- Level 4: Integrated

- Level 4 increases the fuzzing time per fuzzer type to one week. There is no longer a minimum threshold for test cases—for each attack vector, a generational fuzzer and a template fuzzer must both be run until the minimum required time is reached.

- Level 5: Optimized

- Level 5 increases testing time to 30 days for each fuzzing type, and requires the use of at least two different fuzzers per fuzzing type. Because fuzzing is an infinite space problem, and because different fuzzers work differently, using two generational and two template fuzzers increases the probability of locating vulnerabilities



# Fuzzing and the IOT

---

- Most IOT systems includes sensors and devices that run proprietary software
- One day the owner of a factory asked this question:
- How can I be sure that the sensors in my factory are reliable, secure and do not send some information to my competitors?
- Fuzzing is one of the few techniques that can be applied



# Fuzzing and fuzzer

## FUZZING

Brute Force Vulnerability Discovery



**MICHAEL SUTTON  
ADAM GREENE  
PEDRAM AMINI**

## Open Source Fuzzing Tools

**Check Your Software for Vulnerabilities and Eliminate Them**

- Complete Coverage of Fuzzing Techniques and How to Integrate Fuzzing into the Development Cycle
- Step-by-Step Instructions for Building Your Own Fuzzer
- A Guide to Open Source Solutions and Commercial Solutions

**Fred Doyle  
Robert Fly  
Aviram Jenik  
Dave Maynor  
Charlie Miller  
Yoav Naveh**

**Featuring an Introduction from  
Dr. Barton Miller**



# Non standard vulns in general

---

- To generalize the search/discovery of vulns in a component, we consider that these vulns defines a systemic property, the **robustness** of the component
- Systemic = it depends upon the component and the relations among components
- There is a relation among
- Search of vulns
- Robustness



# Robustness in ICT

---

• Robustness of a module =

- The module ability of avoiding damage to the overall system even if its specifications are violated
- Violation of the specifications =
  - Inputs differs from the specified one
  - Available resources differs ...
  - ... (enumerating badness)
- A generalization of fuzzing that considers inputs only



# Robustness in ICT

---

• Robustness of a module =

- The module ability of avoiding damage to the overall system even if its specifications are violated
- Violation of the specifications =
  - Inputs differs from the specified one
  - Available resources differs ...
  - ... (enumerating badness)
- A generalization of fuzzing that considers inputs only



# Robustness in biology

---

- Redundancy
- Modularity
- Feedback
- Monitoring of the behavior of a component
- Tuning of the behavior of a component
- Confinement of anomalous behavior
- Uncorrect components are confined and replaced
- No single point of failure

If any of these features is not satisfied a vulnerability is possible



# Robustness vs Vulnerability

---

- Any set of rules that defines how to build a robust module (best solution, best approaches) also defines a set of rules to discover vulnerabilities
- If the rules are violated, then the module is not robust, then there are some vulnerabilities
- As we have seen in fuzzing a crash (violation) signals a potential vulnerability



# Robustness

---

- It differs from performance, efficiency, ease of use, ....
- It can be increased only by decreasing performance, efficiency, ease of use, ...



**No Free Lunch Theorem**



# Robustness

---

- Let us consider a program that given the name of a worker returns the worker's salary
- The program is
  - correct if the salary is correct for any worker
  - high performance if the salary is computed in a very short time
  - easy to use if you learn how to use it in a short time
  - robust ????





# Robust .... what happens if

---

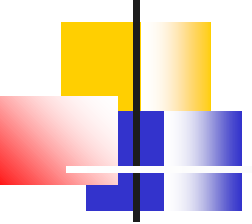
- Wrong record format in the file
- No worker with the name
- The name is 457 characters
- The allocated memory is smaller than expected
- No file with the worker names
- No file with info to compute the salary
- No disk ....



# How much robustness ....

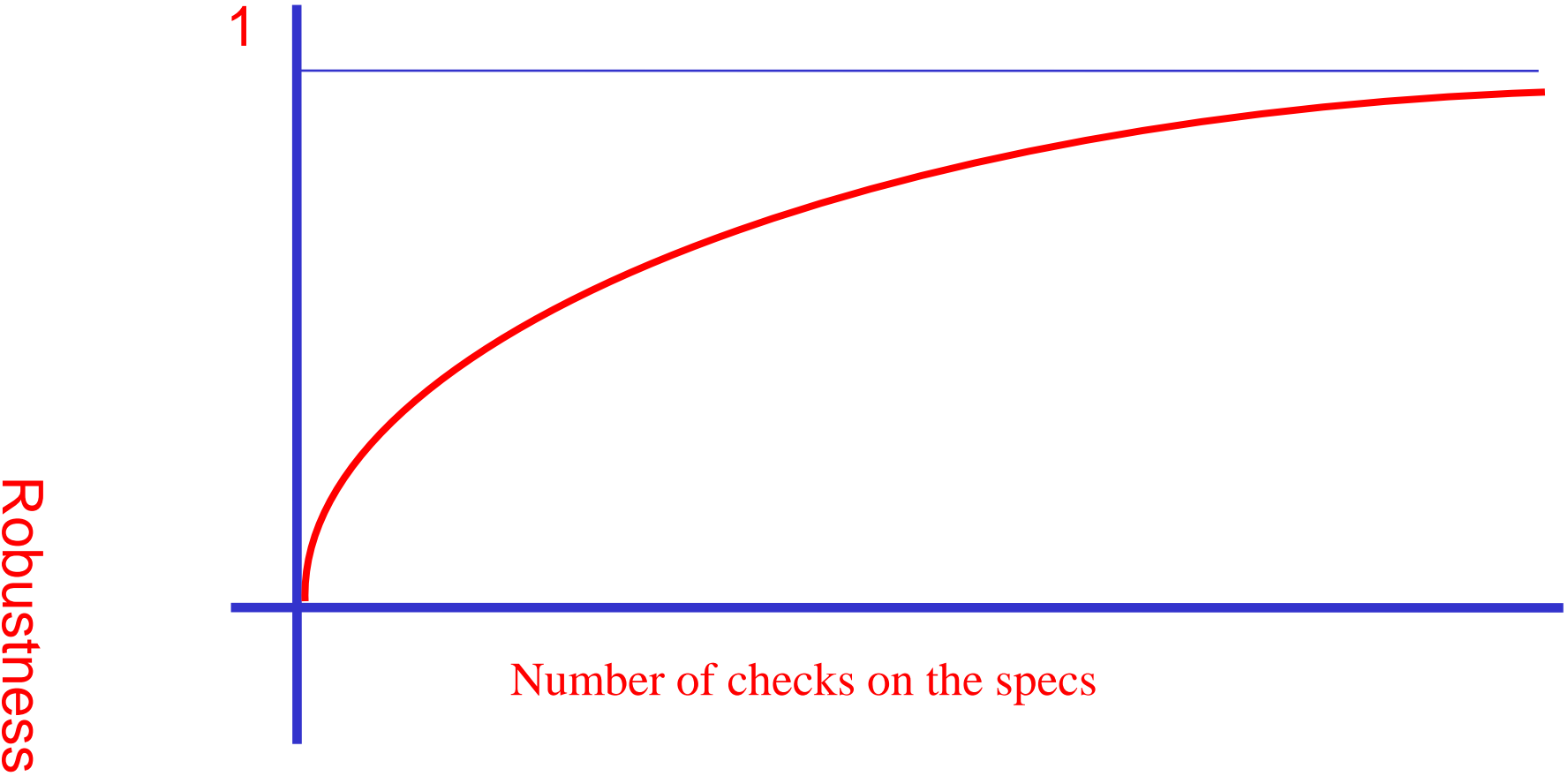
---

- It is almost impossible to define in advance any violation (this proves the weakness of enumerating badness for default allow)
- Robustness is not a 0/1 property
- A robustness measure is a continuous value in the range 0..1 and 1 is an asymptotic value
- Robustness depends upon the number of checks in the module program to discover violations before using an input or a resource



# How much robust?

---





# How much robust?

---

- The value depends upon the number of checks
- Robustness  $\rightarrow 1$  if the number of checks  $\rightarrow \infty$
- Assuming specs are correct, usually checks are useless because the probability the specs are violated is neglectable
- A compromise is required because the number of checks reduces the overall performance
- $\rightarrow$  they slow downs the component execution being implemented through instructions as any other function



# Robustness

---

- It has been experimentally confirmed that even trivial checks can improve the component robustness
- This implies that complex checks should be adopted only after trivial ones
- Most efficient checks are those related to data types (inputs etc)



# Robustness vs Vulns

---

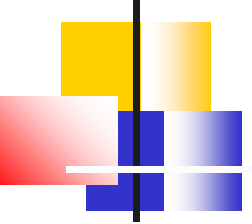
- In an ideal system all the modules implement any control
- The ideal system is the asymptote of a sequence of distinct systems each applying more checks than the previous one
- Any difference between the ideal system and the one being installed and used may be a vulnerability
- If it is a vulnerability depends upon the context and the cost of the control vs the probabilities it detects an anomaly
- Any set of guidelines to build a system also defines the potential vulns of the system



# Robustness vs Vulns

---

- Some differences between the ideal system and the current one cannot be avoided if some controls have not been adopted to satisfy some performance requirements
- Other differences may be unrelated to performance and, hence, controls should be introduced
- The key strategy to discover vulnerabilities **evaluates the cost of missing control** and contrasts it against the required final performance and the risk due to the missing control

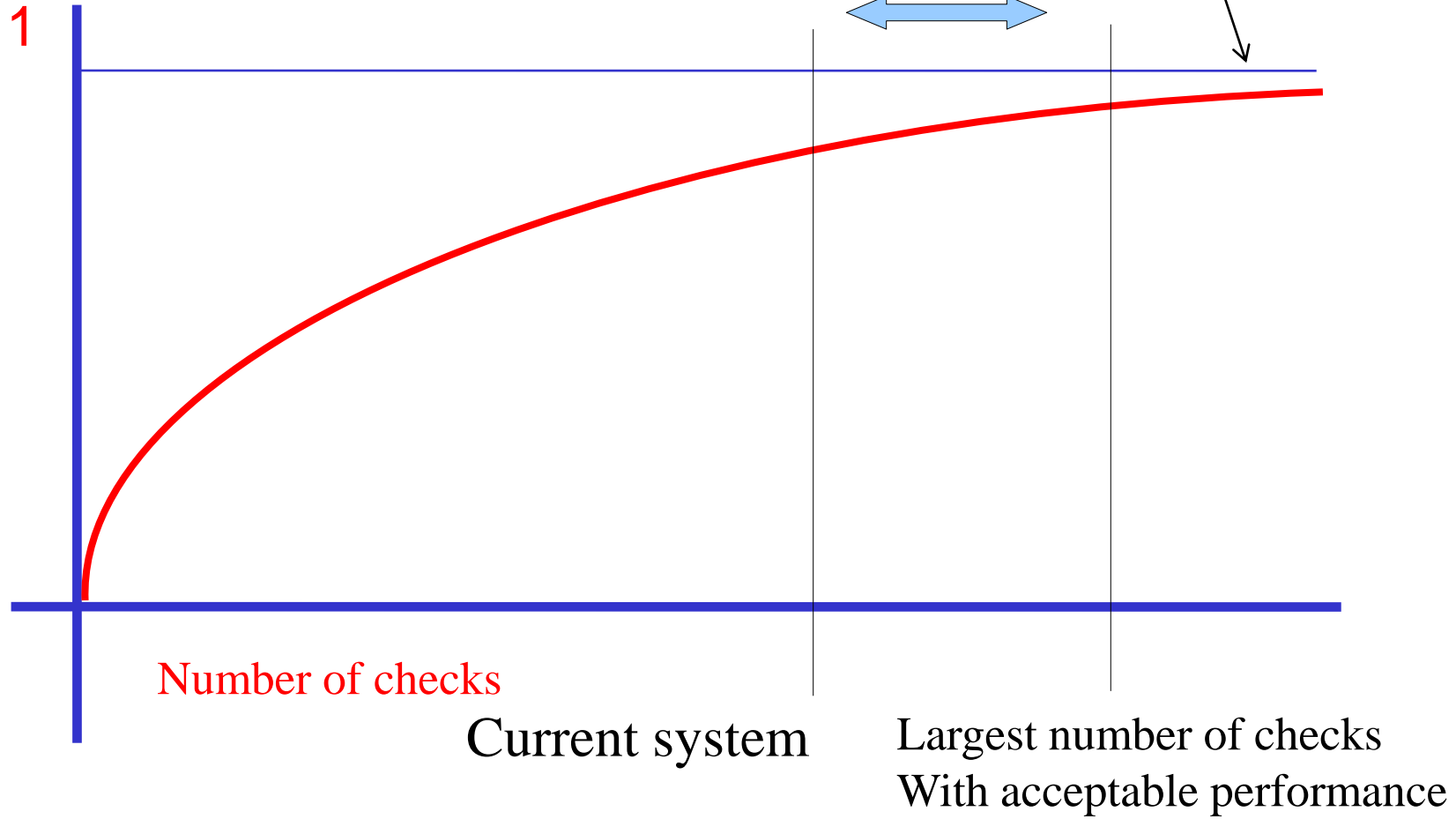


# How much robust?

Asymptotic robustness

Potential vulns

Robustness







# Safety vs Security

---

- Robustness may also be adopted to evaluate the safety of a system
- Security differs because we are interested in robustness with respect to intelligent attacks rather than to random failures



# Safety vs Security

---

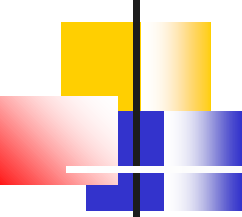
- Safety is proportional to the ratio of anomalous behaviors vs the overall number of behaviours
- A fault results in an anomalous behavior but, if faults are not related with one another, then the ratio shows the cases where faults are not controlled and confined
- In security, the attacker tries to force the system to behave in an anomalous way by attacking those components that influence the behavior of interest
- Safety = random faults / Security = intelligent faults



# Safety vs Security

---

- Both applies the notions of probability and of risk
- Safety is focused on independent probability distribution (law of large numbers)
- Security is focused on conditional probability
  - There are some vulns, hence
  - There are some attackers, hence
  - The attacker can implement the attacks ...



## Design principles for robustness (Saltzer&Schroder) or rules to discover vulnerabilities

---

- Economy of mechanisms
- Fail safe default (Default deny)
- Complete mediation
- Open design
- Separation of privilege
- Least Privilege
- Least common mechanism
- Psychological Acceptability
- Work factor
- Compromise recording



# 8 or 10 principles?

---

- After introducing the first 8 principles, S&S say:  
*Analysis of traditional physical security systems have suggested two further design principles which, unfortunately, apply only imperfectly to computer systems*
- The principles applies to both a system and the mechanisms we introduce to secure the system



# P1-Economy of mechanisms

---

*Keep the design as simple and small as possible  
= keep it simple stupid = kiss rule*

- Simple implies that less things can go wrong and when bugs occur, they are easier to find, understand and fix
- Vulns are proportional to the complexity of a mechanisms and to the code to implement it  
= cyclomatic number to find software bug
- Complexity can be achieved by composition
- SO Hardening = remove OS functionalities useless for applications



# P1- Economy of mechanism

---

- Esokernel and microkernel=
  - Avoid the implementation of complex functions in the kernel
- A strong integration between the OS kernel and the applications not only violates modularity principles but helps the spreading of errors
  - (cascade failures)



# P1- Economy of mechanisms

---

- Simplify the interface
- Complex operations should be implemented by composing simple operations
- If most of the operations are rather complex (and hence powerful), we may be forced to allow a user to invoke a powerful operation just because the simple one it needs is missing
- Hence, users will invoke complex operations even to implement simple operations and this increases their rights  
(related to the least privilege principle)





## P2-Fail safe default (Default deny)

---

*Base access decisions on permission rather than exclusion*

- Burden of proof is on the principal seeking permission
- If the protection system fails, then legitimate access is denied but this also denies illegitimate access
- Induction principle for security anytime the initial state of the system is correct

# Fail safe default & Intel




ITRE SOFTWARE SECURITY DEVOPS BUSINESS PERSONAL TECH SCIENCE EMERG

{\* SECURITY \*}

## 'Unfixable' boot ROM security flaw in millions of Intel chips could spell 'utter chaos' for DRM, file encryption, etc

Although exploitation is like shooting a lone fish in a tiny barrel 1,000 miles away

By Shaun Nichols in San Francisco 5 Mar 2020 at 14:00

145  SHARE ▼



# Fail safe default & Intel

---

It is a fascinating vulnerability, though non-trivial to abuse in a practical sense. It cannot be fixed without replacing the silicon, only mitigated, it is claimed: the design flaw is baked into millions of Intel processor chipsets manufactured over the past five years. The problem revolves around cryptographic keys that, if obtained, can be used to break the root of trust in a system.

Buried deep inside modern Intel chipsets is the Converged Security and Manageability Engine (CSME). It's a miniature computer within your computer.

It has its own CPU, its own RAM, its own code in a boot ROM and access to the rest of the machine.



# Fail safe default & Intel

---

More recently, the CSME's CPU core is 486-based, and its software is derived from the free microkernel operating system MINIX. You can find a deep dive into the technology behind it all, sometimes known as the Minute IA System Agent.

Like a digital janitor, the **CSME works behind the scenes, below the operating system, hypervisor, and firmware, performing lots of crucial low-level tasks**, such as bringing up the computer, controlling power levels, starting the main processor chips, verifying and booting the motherboard firmware, and providing cryptographic functions.

**The engine is the first thing to run when a machine is switched on.**



# Fail safe default strikes back

---

One of the first things it does is set up memory protections on its own built-in RAM so that other hardware and software can't interfere with it. However, these protections are disabled by default, thus there is a tiny timing gap between a system turning on and the CSME **executing the code in its boot ROM that installs those protections,** the form of **input-output memory-management unit page tables.**

During that timing gap, other hardware – physically attached or present on the motherboard – can fire off a DMA transfer into the CSME's private RAM to overwrite variables and pointers and hijack its execution. At that point, the CSME can be controlled for malicious purposes, **all out of view of the software running above it.**



# Threat analysis

---

[https://www.theregister.co.uk/2020/03/05/unfixable\\_intel\\_csme\\_flaw/](https://www.theregister.co.uk/2020/03/05/unfixable_intel_csme_flaw/)



# Threat analysis

---

- A timing attack, very low success probability but the attacker may repeat it at any boot
- Very worrying situation if
  - your boards have been built by a competitor or in another state
  - your cloud provider thinks you have interesting information in your application
  - everything depends upon your threat model
- Supply chain attack = the threat is a supplier



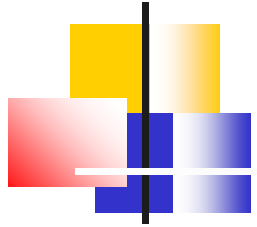
## P3-Complete Mediation

---

*Every access to every object must be checked for authority*

- Usually it is done once, on first access, but if permissions change later on, unauthorized accesses may be possible
- Performance gains achieved by **caching the result of an authority check** should be examined skeptically
- Each operation that is not controlled is a potential vulnerability as it may be invoked when the access rights have been revoked





# P3 - Access control matrix

---

		object	Which object operations the subject is entitled to invoke
subject		rights	Usage of acm is a condition 1.necessary 2.not sufficient for a secure system



# Access control matrix

---

- Security requires this matrix exists for each system implementation layer
- Furthermore, there is also a matrix for each application or virtual machine at the application layer
- Coherency among these matrices
- A matrix may be so large that it has to be stored on a secondary storage



## Rights in $acm[i,j]$ -I

---

- DAC security policy = assigned by the owner of the j-th object
- MAC security policy = they also depends on the levels of the i-th subject and the j-th object
- In both cases the subject may have to actually satisfy further constraints before using the rights that the matrix assigns

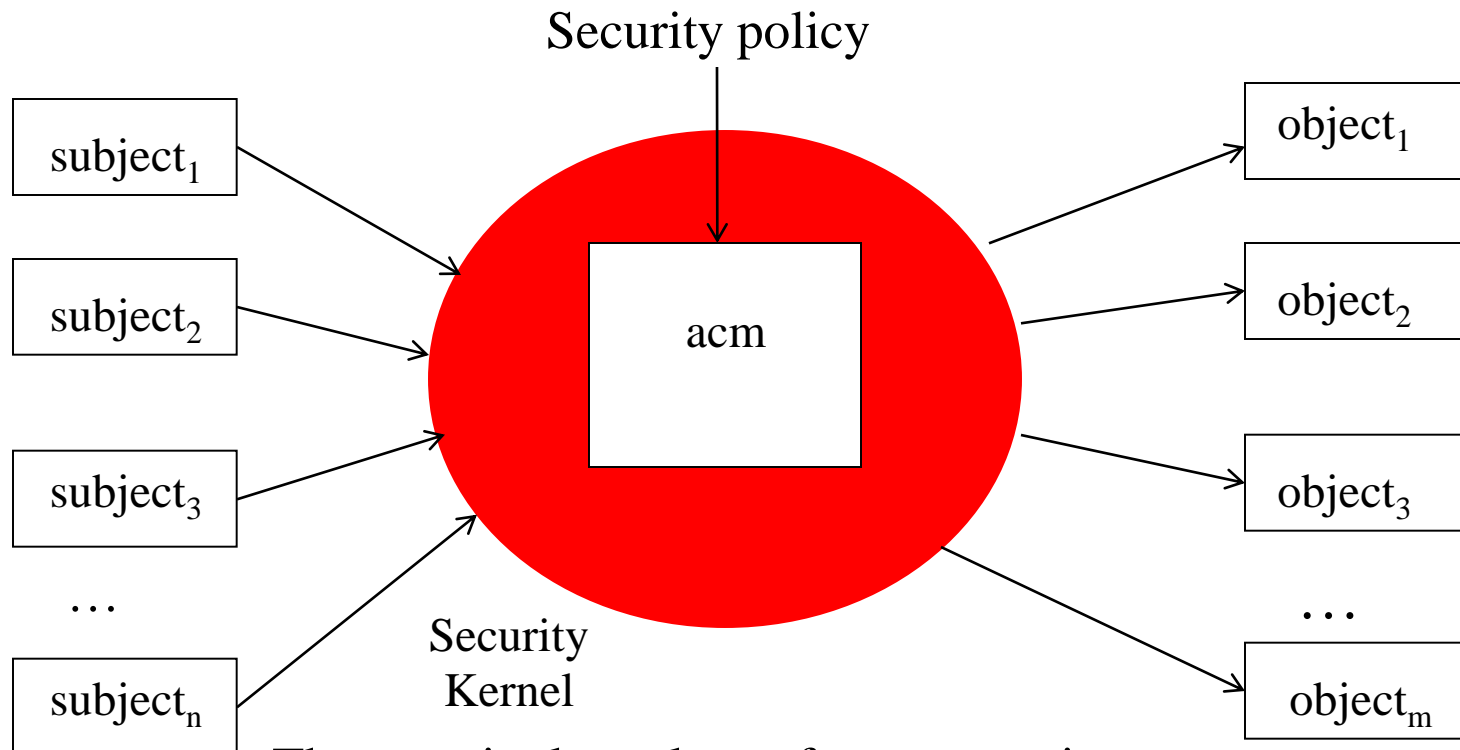


## Rights in $acm[i,j]$ -II

---

- The access control or protection matrix is a **highly dynamic data structure**
- Dynamicity is due to
  - Dynamic creation and destruction of subjects and objects
  - Some security policies dynamically updates the rights of each subject according to the operations the considered subject has invoked

# Acm: a typical implementation



The security kernel or reference monitor (TCB) mediate the subject attempts to invoke the operations defined by the objects



# Access Control Matrix

---

- This is a logical data structure for which a large number of concrete implementations is possible
- Sometime the acm is not implemented by a matrix
- Problems arises when no all the subjects are known in advance (network services)
- In this case, a row of the acm is paired with a class of subjects
- Rules to map each subject into a class have to be defined



# Security Kernel or Reference Monitor

---

- It belongs to the Trusted Computing Base (TCB)
  - = its correctness is a necessary condition for the correct implementation of the security policy
- As small as possible to apply formal techniques to prove its correctness
- A basis for induction proof of security properties
- In some systems it is stored in a tamper proof memory to prevent illegal updates



# Tamper proof

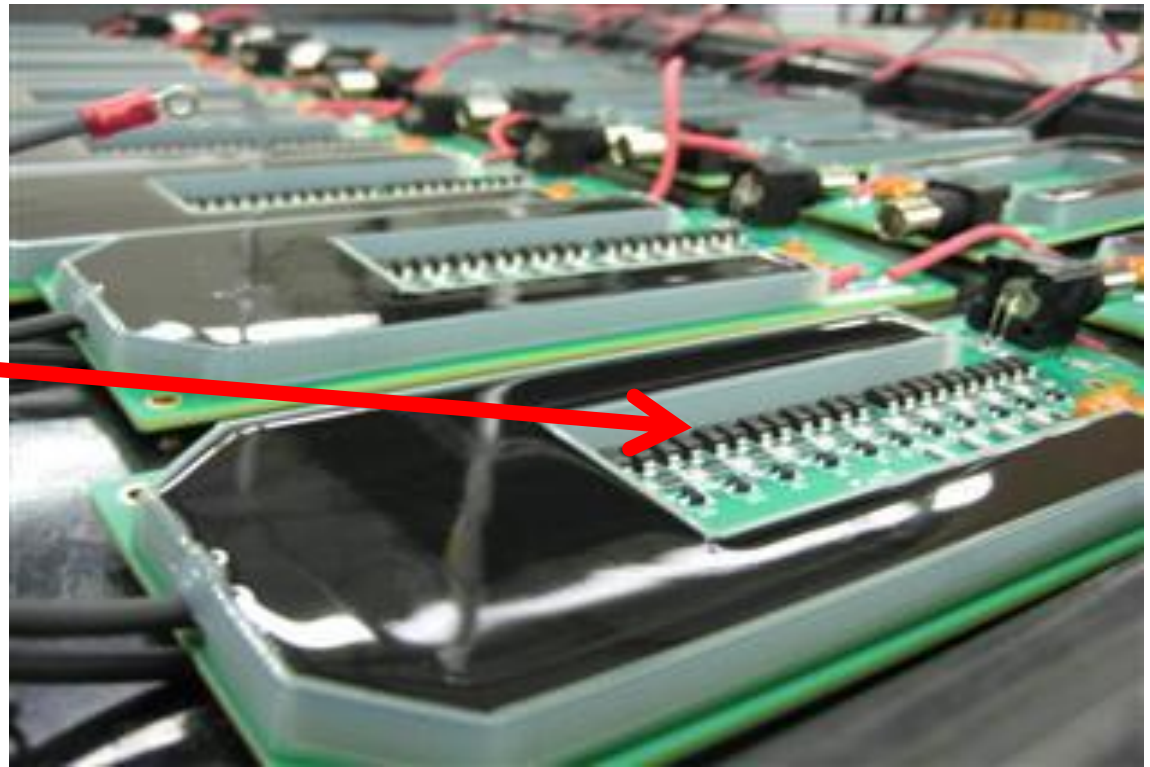
---

- A component where any physical attack is
  - Prevented or at least
  - Detected
- All the components are glued with silicone
- Memory chips are protected by an electrified grid that cancel any information as soon as an attack is attempted



# Silicone tamper proof

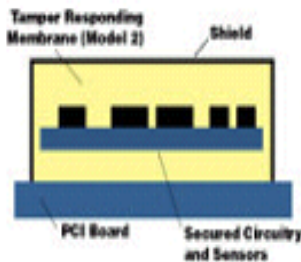
Silicone



# Secure Coprocessor

## IBM 4758 hardware

The IBM PCI Cryptographic Coprocessors are state-of-the-art secure subsystems that you can install in server systems to perform DES and public-key cryptography. You can also load software for highly sensitive processing, such as the minting of electronic postage, which must perform its intended function even when under the physical control of a motivated adversary.



The secure Coprocessor module is mounted on a two-thirds length PCI version 2.1 board and fits in a single slot that provides +5.0VDC +/- 15% and +12VDC +/- 15% power and meets other environmental requirements as listed in chapter 2 of the [General Information Manual](#). (3.3VDC variations of the Models 002 and 023 are also available to OEM customers and used in IBM @server pSeries, and IBM @server iSeries and IBM @server zSeries servers.) The sealed Coprocessor module incorporates physical penetration, power sequencing, temperature, and radiation sensors to detect physical attacks against the encapsulated subsystem. Batteries provide backup power that is active from the time of factory certification until the end of the product's useful life. Any detected tamper event results in loss of power which immediately causes the zeroization of internal secrets and the destruction of the factory certification.



# Complete mediation + fail safe default

---

- If both principles are applied
  - The system starts in a secure state
  - Provided that the security kernel is correct, only secure transitions are enabled
- Induction proofs on reachable states
- If fail safe default does not hold no induction basis exists



# Complete mediation+ fail safe default

---

- Let us assume that to grant a right  $R$  on an operation  $op$  the object  $Ob(op)$  has to be updated
- In the initial state no subject owns the right of updating  $Ob(op)$
  - No subject can grant this right
  - Hence no subject can be granted this right



# Access control matrix

---

- An implicit assumption is that the identity of the subject is checked before accessing the matrix
  - = how can we check that a subject that claims of being A actually is A
- Explicit check in the security kernel
- Password
- One-time password
- Challenge response
- Electronic signature



# One time password

---

- A function  $F$  with at least two parameters
  - $S$  a secret value distinct for each subject
  - $N$  the number of received requests (defined in an implicit or explicit way )
- The subject to be authenticated computes and transmits  $Y = F(S, N)$
- The receiver computes  $X = F(S, N)$  and checks that  $X = Y$
- Synchronization on the value of  $N$



# Challenge - response

---

- Partners agree on a function  $F$  and keep it secret
- $F$  has an input parameter  $x$
- One of the partners sends  $y$  (challenge)
- The receiver computes  $F(y)$  and sends back the result
- Also the challenger computes  $F(y)$  to check whether the received value is correct



# Complete mediation: problems

---

- High performance in the access to acm is required due to the huge number of checks
- An implementation where a centralized data structure is shared among all the subjects and the objects usually cannot achieve an acceptable performance
- A distributed/parallel solution is to be preferred so that the overhead is independent of the number of objects



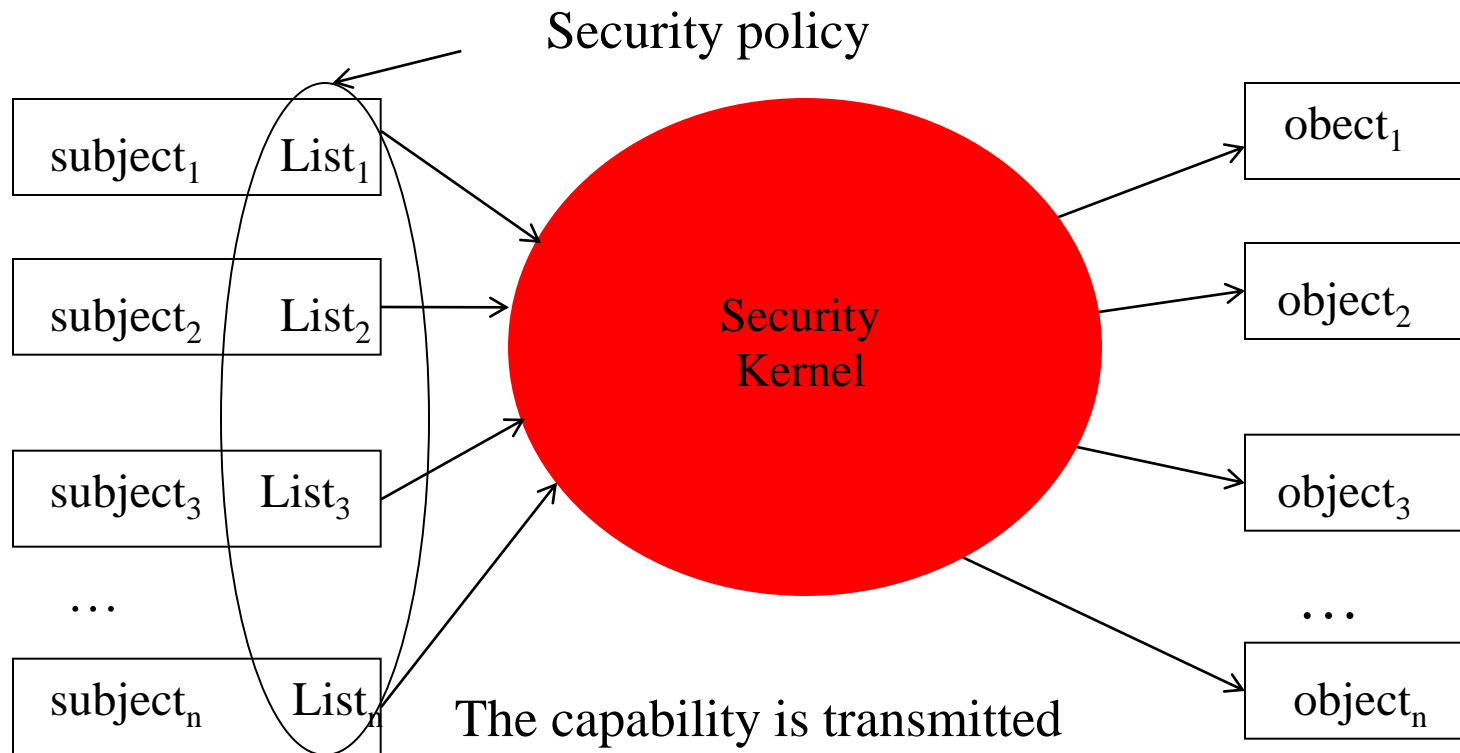


# Solutions - 1

---

- Capability list= a row based organization of the matrix
  - A capability is a pair  $\langle \text{object address} , \text{rights} \rangle$
  - A capability is a generalization of pointer also know as a protected pointer
  - When invoking an operation, the subject specifies which of its capability has to be used for the operation ie it enables the operation

# Acm as capability lists



The capability is transmitted to the security kernel that checks whether it enables the operation  
The SK does not manage the ACM



# Capability -I

---

- Invocation  $op_i(obj_j, par, n)$  = execute the  $i$ -th operation of the  $j$ -th object as enabled by the  $n$ -th capability in the subject list
- If  $S$  transmits a capability to another subject  $S'$  then  $S$  can delegate  $S'$  to invoke an operation  $S'$  is not entitled to
- Capability = ticket for an object
- Delegation increases the number of instances of a given rights that, in turn, increases the complexity of right revocation



## Capability - II

---

- Capabilities are generated by the security kernel that distributes them to the subject
- A subject should only be able
  - to store
  - to read (use)
  - to copy (delegation)
  - but not to update a capability
- Only the kernel can update a capability
- The probability of a successful attack against the security policy increases since rights are stored in the subject



# Capability -III

---

- In some architectures the MMU implements an efficient hw/fw support for capabilities at the OS levels
- The capability list is stored in the MMU
- Capability = physical address + op bits
- The MMU
  - checks the rights in parallel with the address translation
  - prevents a subject from updating its list



## Capability -IV

---

- Address translation exploits a segment/page table that store the physical address
- For each segment/page a bit maps codifies the access rights for predefined operations  
(read, write, fetch)
- Some processors do not check the rights if the segment/page has already been loaded into the cache or if the address has already been translated from logical to physical



# Capture the flag exercise

---

My challenge to you to discover vulnerabilities in a protocol using capability

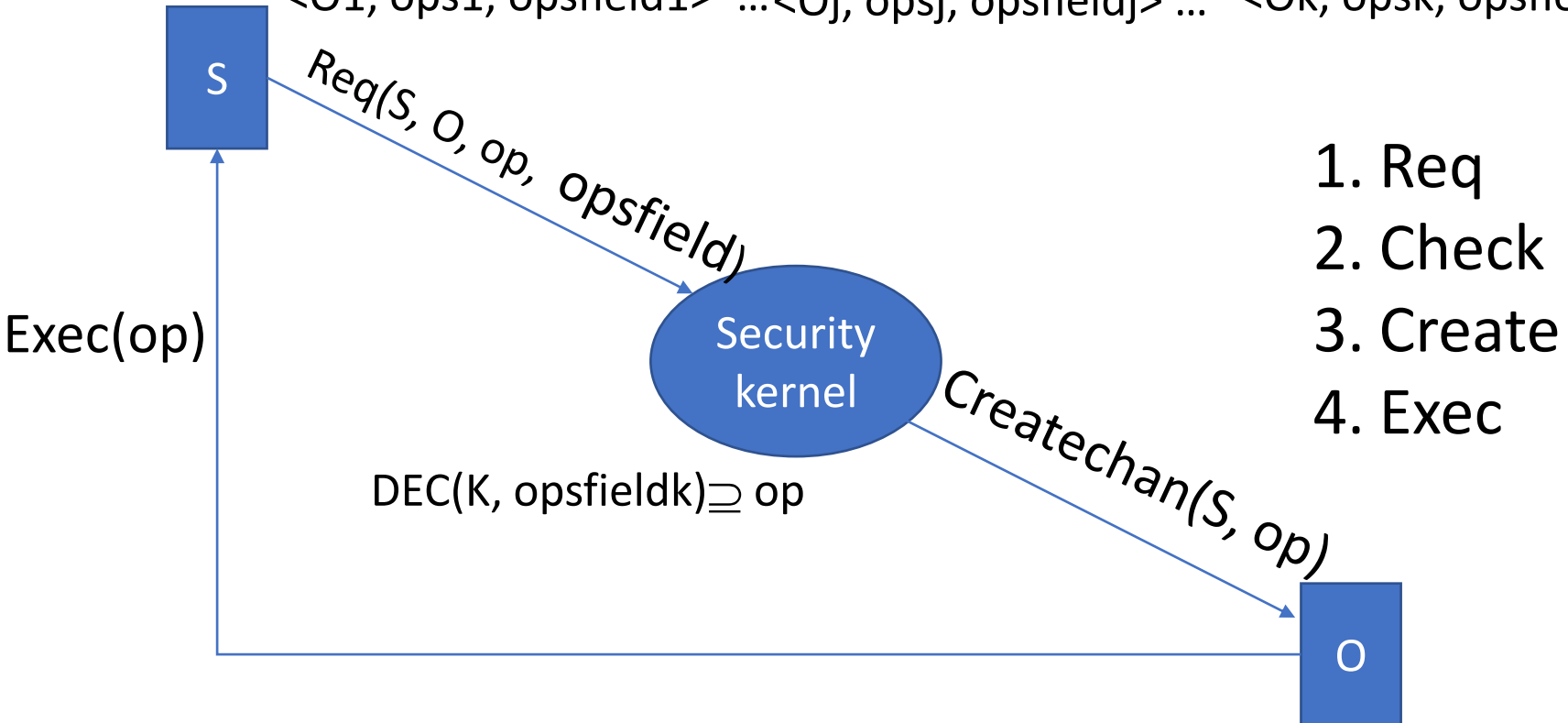
## Outline

- 1) Each subject manages a list with its own capabilities
- 2) The operation field of a capability is encrypted with a key private to the security kernel SK
- 3) To request operation  $Op$  on object  $O$ , a subject  $S$  sends to SK a message with  $S$ ,  $O$ ,  $Op$  and the encrypted capability
- 4) SK decrypts the capability and, if it enables  $Op$  on  $O$ , it asks  $O$  to create a channel with  $S$  to execute  $OP$
- 5)  $O$  destroys the channel when  $Op$  ends

# Capture the flag exercise

$opsfieldm = ENC(K, ops \text{ in } ACM[S, Om])$

$\langle O1, ops1, opsfield1 \rangle \dots \langle Oj, opsj, opsfieldj \rangle \dots \langle Ok, opsk, opsfieldk \rangle$







# Capture the flag exercise

---

Challenge to you

Discover vulnerabilities in the proposed protocol or in the overall system under the assumption that there are no vulnerabilities in the encryption algorithm ie  $K$  cannot be discovered because of mathematical vulnerabilities

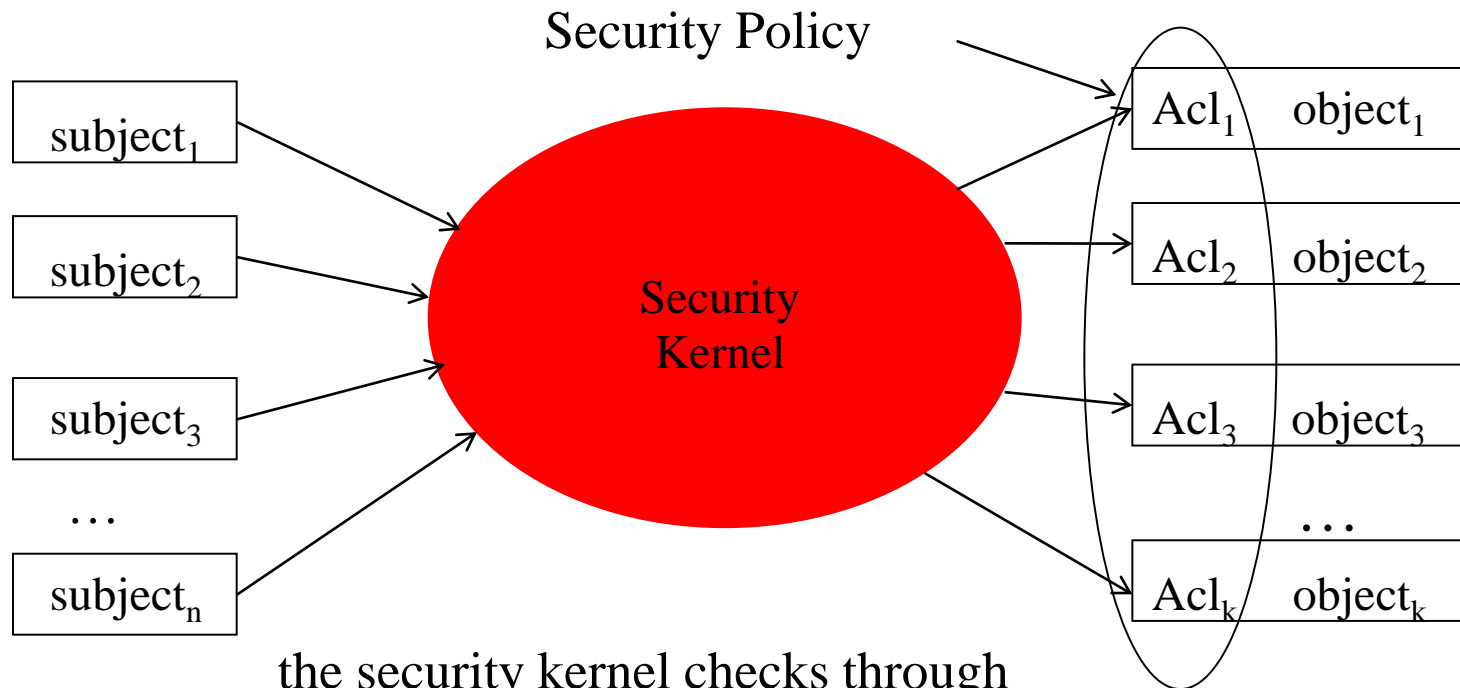


## Complete Mediation - 2

---

- Access control list = a column based organization of the acm
- One list for each object
- Each list element stores the rights of all the subjects on a distinct object
- Now the control can be implemented by the Security Kernel or be delegated to the object
- A centralized structure for each object

# ACM: ACL



the security kernel checks through the object ACL that the security policy is satisfied

The checks may also be implemented by the object



# Access control list

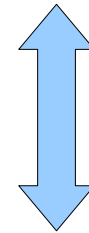
---

- A more flexible solution may be achieved through
  - Partition of the subjects
  - The sequential scanning of the list (no direct access is possible because the subject does not know its position)

*If subject  $\in$  Set1 then {op1, op2}*

*else If subject  $\in$  Set2 then {op3, op4}*

*else {op5}*



this is an ACL!

- the subjects are partitioned into three sets
- this can grant rights even to **subjects not known in advance**. This is not possible for capabilities and it may be adopted to define acs for **web services**



# HW/FW support for ACL

---

- Associative memory where the key may be
  - Subject → set of rights
  - Subject, operation → boolean
- FPGA that implements a function that is a chain of *if statements* about
  - Sets of users
  - Priority among sets



# ACL vs Unix files

---

- Each file is paired with a bit array that defines
  - Owner rights
  - Group owner rights
  - Other users rights
- this is an implementation of the file ACL
  - It adopts classes of users due to missing information on all the system users



# ACL and file descriptor

---

```
struct stat {  
  
    mode_t st_mode; // File type & mode                access control list + set uid bit  
    ino_t st_ino; // i-node number  
    dev_t st_dev; // device number (file system)  
    dev_t st_rdev; // device n. for special files  
    nlink_t st_nlink; // number of links  
    uid_t st_uid; // user ID of owner  
    gid_t st_gid; // group ID of owner  
    off_t st_size; // size in bytes, for reg. files  
    time_t st_atime; // time of last access  
    time_t st_mtime; // time of last modif.  
    time_t st_ctime; // time of last status change  
    long st_blksize; // best I/O block size  
    long st_blocks; // number of 512-byte blocks  
}
```



# Unix/Linux -I

---

- ACL are defined in terms of process identifier

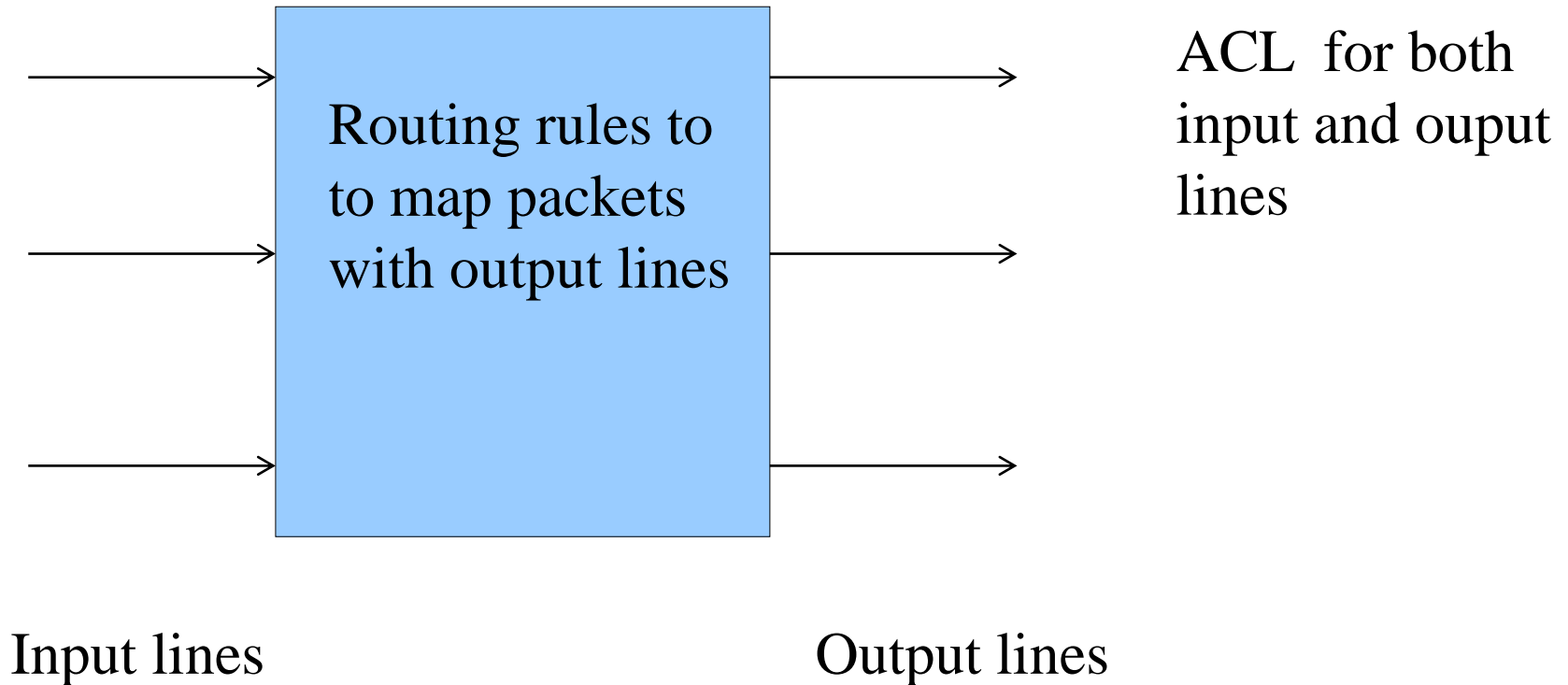
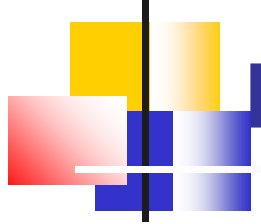
- Real user ID      owner
- Effective user ID
- Saved user ID

in Linux we also have

- File system ID



# ACL for message routing in routers





# ACL for message routing

---

- Router ACLs are built by composing two cases
  - IP Range<sub>1</sub> → route  
packets from these nodes are routed
  - IP Range<sub>2</sub> → drop  
packets from these nodes are dropped
- A list is built for each input/output connection to specify the IP addresses in the packets that can cross the connection
- List = order is important
- Ranges because some addresses may be unknown
- This protects the network where messages are routed



# ACL & Router

---

- ACL of input 1

- 131.114.\*.\* → route
- 131.4.5.6 → route
- 131.4.\*.\* → drop

swapping two rules  
changes everything

Traffic from 131.114.\*.\* is routed and all the traffic from 131.4.\*.\* is dropped but that from 131.4.5.6

- ACL of output 1

- 131.114.\*.\* → drop
- 131.4.\*.\* → drop

No address in 131.4.\*.\* and in 131.4.\*.\* can send traffic to the network connected to output 1



# Routing in Linux: iptables

---

- Input chain: rules for the packets addressed to the node
- Output chain: rules for the packets produced by the node
- Forward chain: rules for the packets that cross the node
- Default allow → transform into a default deny by creating the list of packets to be routed and add “drop all” at the end



# Routing in Linux

---

- Drop
- Route
- Return – return to the invoking chain
- Queue – transmit to user space
- Log
- Reject
- Dnat/Snat/Masquerade

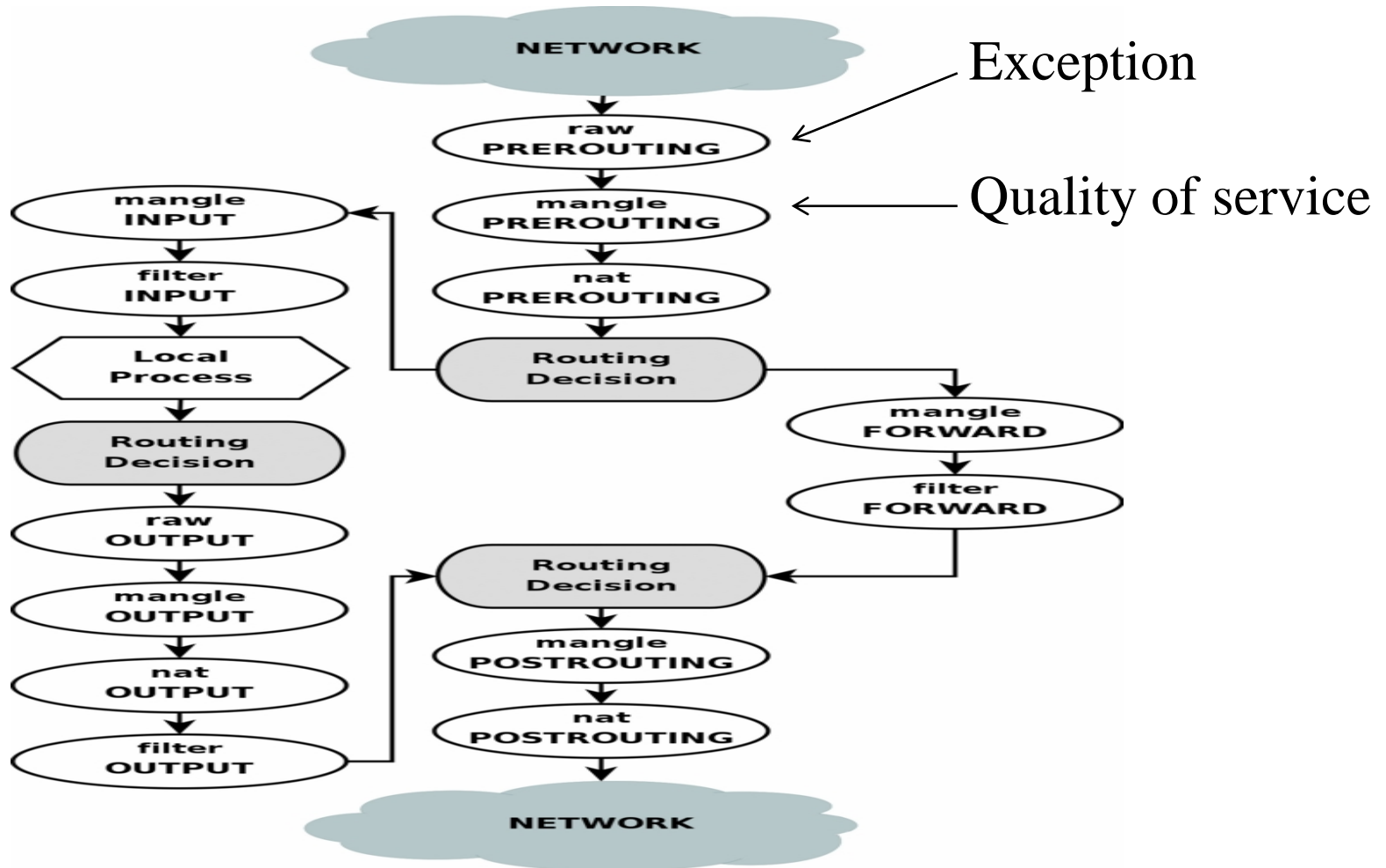


# Nat table

---

- Prerouting chain= any input packet
- Postrouting chain = any output packet
- NAT may change the addresses in a packet
- Applied before INPUT and after OUTPUT/FORWARD

# The overall architecture





# Examples

---

- `iptables -A INPUT -p UDP drop`  
A new rule is inserted in the input chain to drop any UDP packet
- `iptables -A INPUT -p TCP -dport 156 drop`  
Drop any TCP packet addressed to port 156
- `iptables -N newcontrol`  
Create a new chain where new controls can be later inserted





# An important point

---

- Anyone is aware and agrees on the importance of controlling the network traffic that enters a network
- These controls are critical and they are mostly implemented in the border router that connects a network to a public one
- Are there any reasons to check the traffic leaving a network?



# Controlling the output traffic

---

- The control of output traffic is an important mechanism to discover successful attacks against the network (egress filtering)
- If someone is controlling a node X and stealing information in X we can discover the illegal connections of X to some outside network
- These controls can discover Zombies to implement a DDOS



# Egress filtering

---

- It controls the traffic that is attempting to leave the network.
- Before an outbound connection is allowed, it has to pass the filter's rules
- Advantages
  - Discover malware
  - Stop contributing to attacks
  - Block unwanted services



# ACMatrix, subjects and objects

---

- As the number of subjects and objects increases, the complexity of
  - defining the ac matrix
  - checking its correctness
  - achieving full mediation

strongly increases

- Some solutions have been proposed to simplify the definition of this matrix



# Role vs subject

---

- The notion of role is useful when (subject = a final user)
- Role =
  - A professional profile and the corresponding rights
  - Strongly depends upon the applicative environment
- Any role is paired with
  - A set of users that can be assigned that role
  - A set of rights
- Role Based Access Control
- Rights are not assigned to users but to roles
- A user U acquires the rights when a role is assigned to U
- When U leaves the role, it loses the role rights



## Role- II

---

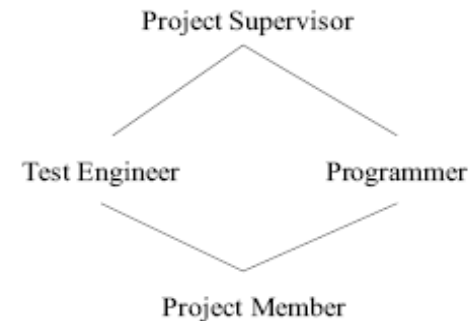
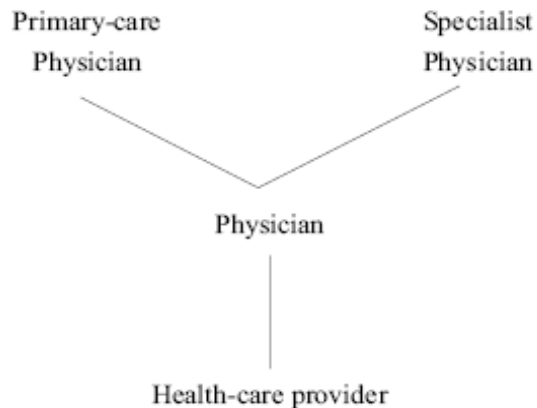
- Rules define when
  - a role may be assigned
  - is lost
- The rules may consider previous operations the users execute
- Any role change may require a password to identify the user and the role



# Role hierarchy - I

---

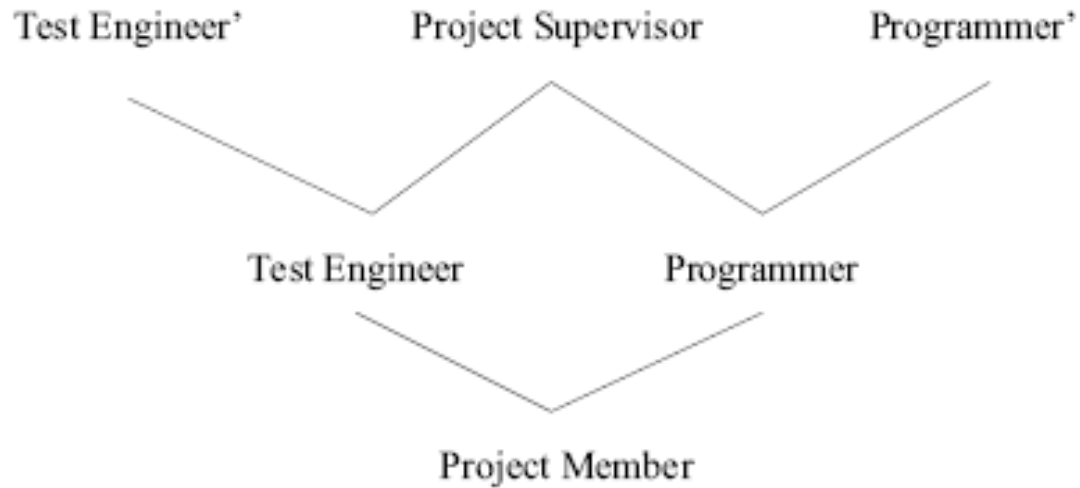
- Role may be partially ordered
- A role R1 is larger than a role R2 if R1 includes all the rights of R2





# Hierarchy II

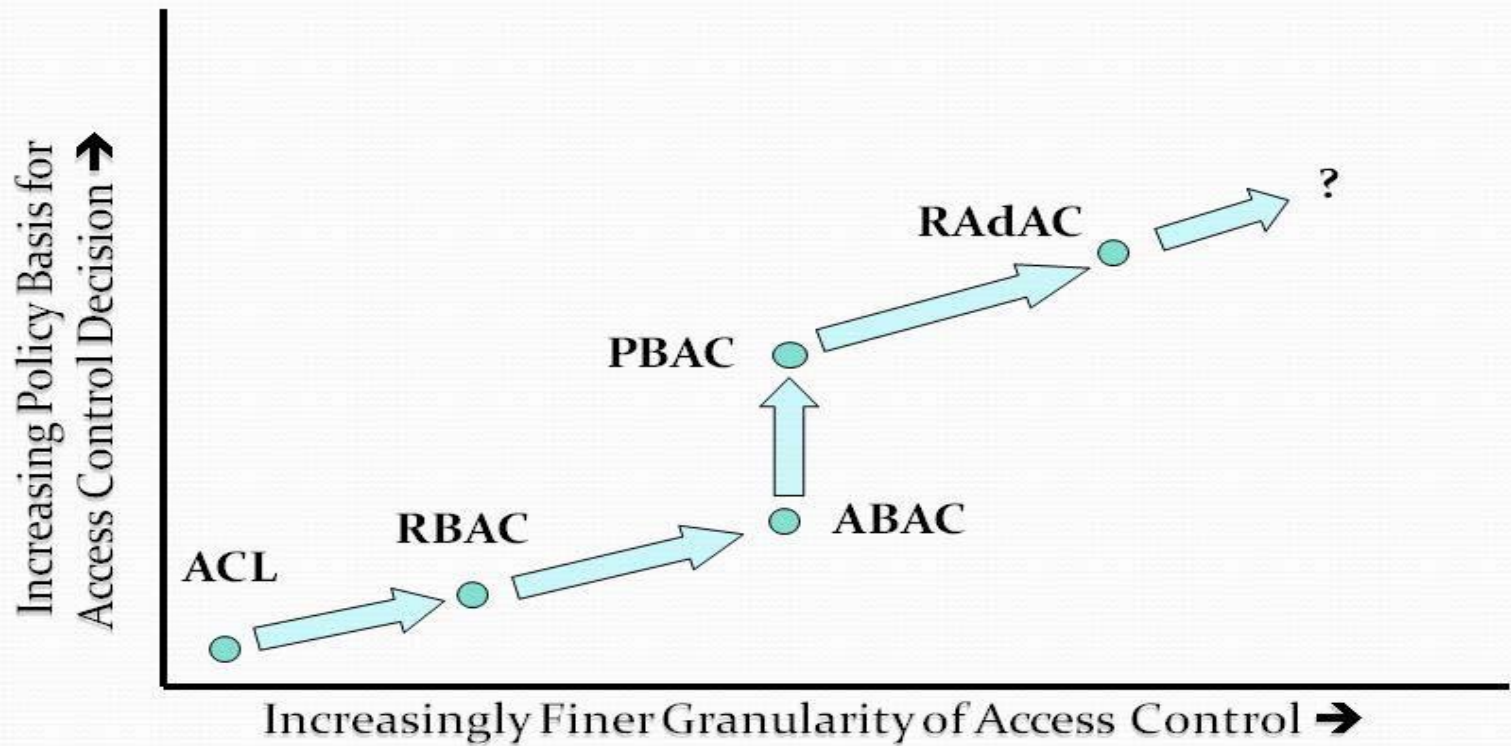
---





# Other models (defined in the following)

## Access Control Models





# Attribute Based Access Control

---

- Each subject is paired with a set of attributes
- The right of invoking an operation is a function of the current values of the attributes
- Not supported by standard OSes but only at the application level (database management systems)
- It could be supported at the OS level provided that a standard set of attributes for all the user is defined



# ABAC

---

Attributes =

- Role
- Security level
- IP address of the user system

As an example the operation can be executed if

- Role= system manager
- (Role= system manager) AND (ip = local)
- (Level > confidential) AND (ip = local) AND (8 <current time <16)



# Risk Based Access Control

---

- The risk posed to the system because of the operation is evaluated
- The evaluation takes into account attributes of the system, of the user etc to decide whether the rights should be granted
- Complex implementation based upon adversary emulation = proactive discovery of what happens if a request is satisfied



# Adversary Emulation

---

- The ability of emulating the behaviour of an attacker before its attack occurs = penetration test
- Obviously in most cases it cannot be done on the real system
- The challenge is the ability of implementing adversary emulation using a digital twin of the target system
- Adoption of the twin tech for ICT security



## P4-Open Design - I

---

*The design should not be secret*

or

*The security should not depend on the secrecy of the design or of the implementation*

- ❑ Popularly misunderstood to mean that source code should be public



## P4-Open Design - II

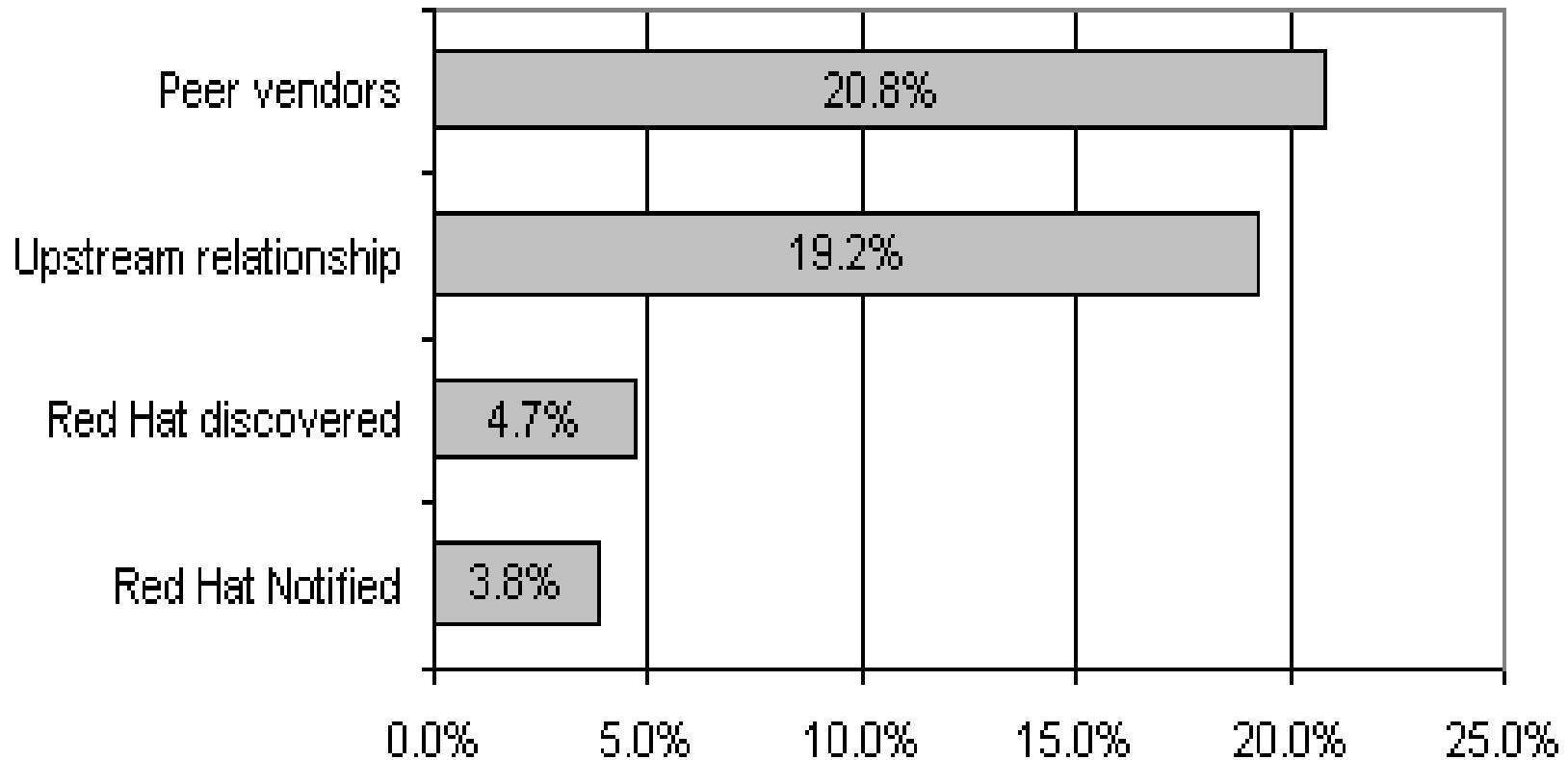
---

- A system peer review is fundamental to discover vulns in the design and/on in the implementation
- An open source implementation is useful only if
  - it results in a peer review
  - any peer that discovers a vulnerability communicates it to the owner
- The open design is useless if
  - no peer review (no peer) or
  - discovered vulns are not revealed to the owner (no review)
- Strength and weakness of open source



# Vulns vs open design

---







# Open design and the attacker work

---

- A benefit of closed design is that the attacker has to acquire the information on the design using scanners and vulnerability scanners
- This slows down the attacker because it increases the amount of work to attack a system due to
  - the overhead of acquisition
  - useless attacks due to unaccurate information



## P5 - Separation of privilege

---

*Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key*

*or*

*Require multiple conditions to grant privilege*

- Separation of duty
- Defence in depth



## P5 – Separation of privilege

---

- A complex operation should be decomposed into simpler operations
- If we assume the complete mediation principle is satisfied then
  - The complex operation and each simple one are enabled by a proper (distinct) rights
  - We can control that the subject owns both
    - The right of invoking the complex op
    - The right of invoking each simple op



# Example

---

- Op = transfer money from account1 to account2
- 5 rights
  - Transfer money
  - Read account1
  - Update account1
  - Read account2
  - Update account2
- If we do not control each operation someone may invoke the complex one and transfer money but not from account1 or to account2

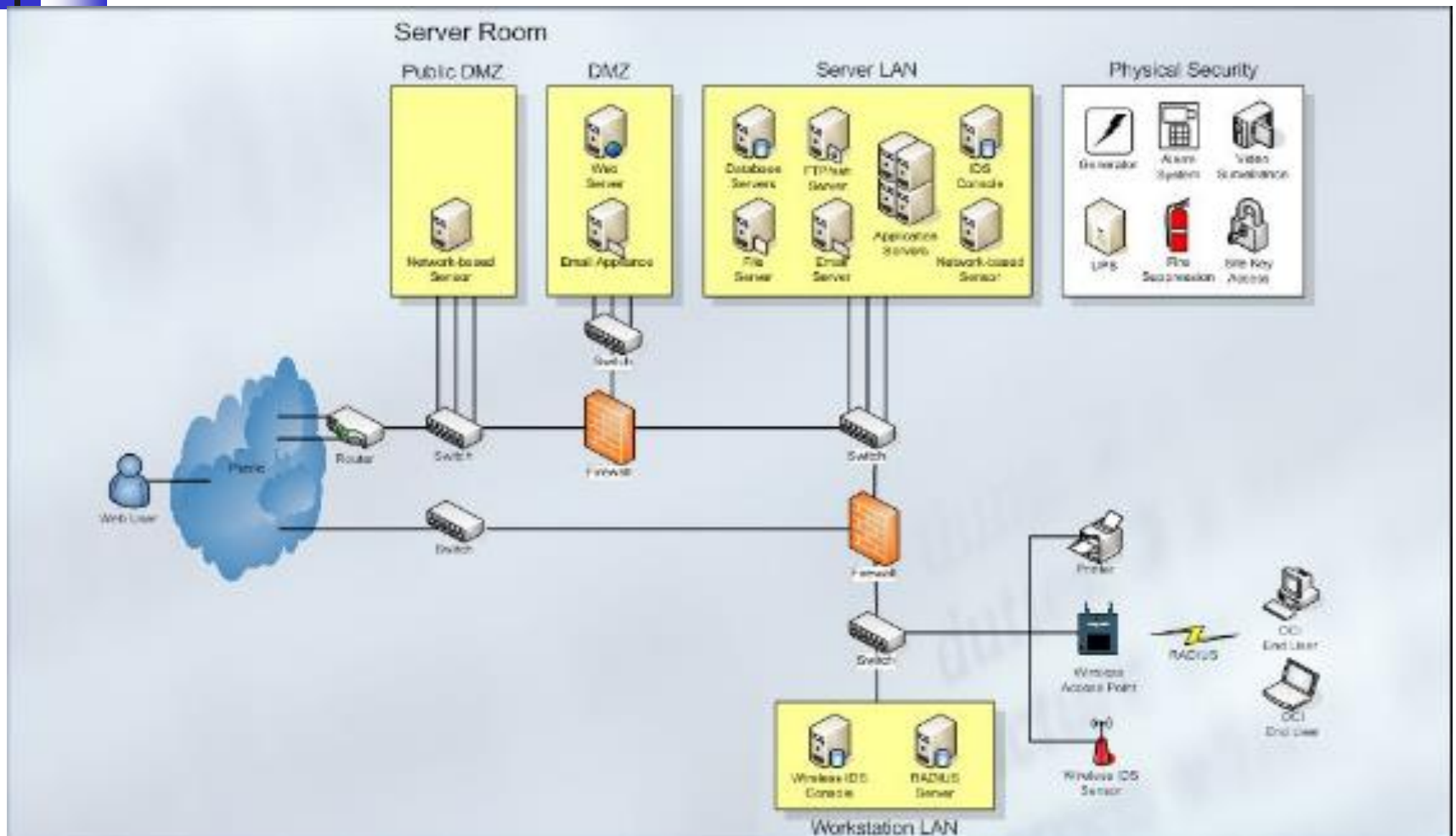


# Defence in depth

---

- Flat network
  - any node can interact with any other one
  - a hub that connect all the nodes
- Segmented network
  - Network is partitioned into subnetwork
  - One hub for each subnetwork
  - Hub are connected by routers
  - Router ACLs determine which traffic can enter or leave a network

# Segmented Network





# Segmented network

---

- A strategy to attack a segmented network adopts pivoting
- Pivoting = a host is attacked just to use it to route traffic to another node
- It forces the attacker to implement a larger number of attacks =  
to execute a larger amount of work



# The lion parable

---

Two guys in a bush meet a hungry lion that starts looking at them as its next lunch. One of the two guys tightens its shoes to prepare to run and the other asks

«Do you believe you can run faster than the lion?».

The other replies

« I do not need to run faster than the lion, I just need to run faster than you»





## P6 – Least privilege - I

---

*Every subject should operate using the least set of privileges necessary to complete its job*

or

*A subject should be given only those privileges it needs to complete its task and only **for the time** to complete it*

- Owning a useless access right is a vuln
- Rights granted as needed, revoked after used
- The ac matrix is a dynamic data structure
- Rights are assigned and revoked as the computation evolves



## P6 – Least privilege - II

---

- This principle should be applied even if the security policy is static as it defines **how rights should be managed** after being assigned to each subject rather than how they are assigned
- If, in a given time interval TI, a subject does not need a right R then it should be revoked and then granted by updating the acm to prevent the subject from using the right in TI
- R is removed at the beginning of TI and then granted at the end of TI
- Rigorous time version of can know/need to know



# Least Privilege - III

---

- Protection Domain Switching = the same subject is executed but the rights in its row of the acm
- Protection Domain Switching = update of an acm row
- A PD switching should occur even without a context switching
- The corresponding overhead is a function of the implementation level and the adopted implementation of the acm (capability vs acl)
- Revoking a right is not simple with capabilities



## Least Privilege - IV

---

- An alternative definition stresses the importance of minimizing the protection domains = the number of the subject rights
- As the size of the protection domain of S decreases, it also decreases the risk due to an attack against S
- If the protection domains are large then revoke access rights when not needed and grant them again when needed



# Least Privilege - V

---

- Notice the strong relation between the two last principle
- As an example, segmented networks force the reduction of protection domains
- Separation of duties enables the implementation of least privilege



# Least Privilege - VI

---

The system designer has to choose the proper compromise because a full application of this principle may result in low performances

- ⇔ for each command that is executed, the acm should to be updated
- ⇔ the asymptotic system is too slow



# Least privilege – In principle

---

When/how the domain switching is fired

- 1) Through further, proper instructions
- 2) Some language constructs also fire the domain switching

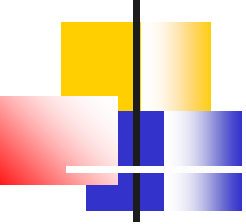


# Least Privilege – Common solution

---

- A classical solution switches a domain when
  - A procedure (method) is called
  - A procedure (method) returns
- A new row is created (call) and destroyed (return) rather than updating a row
  - When the procedure is invoked, a new row that defines its rights is created
  - The row is destroyed when the procedure returns
  - Rights are paired with the instance of a procedure executed by (or on behalf of) a subject rather than with the procedure static code or with the subject



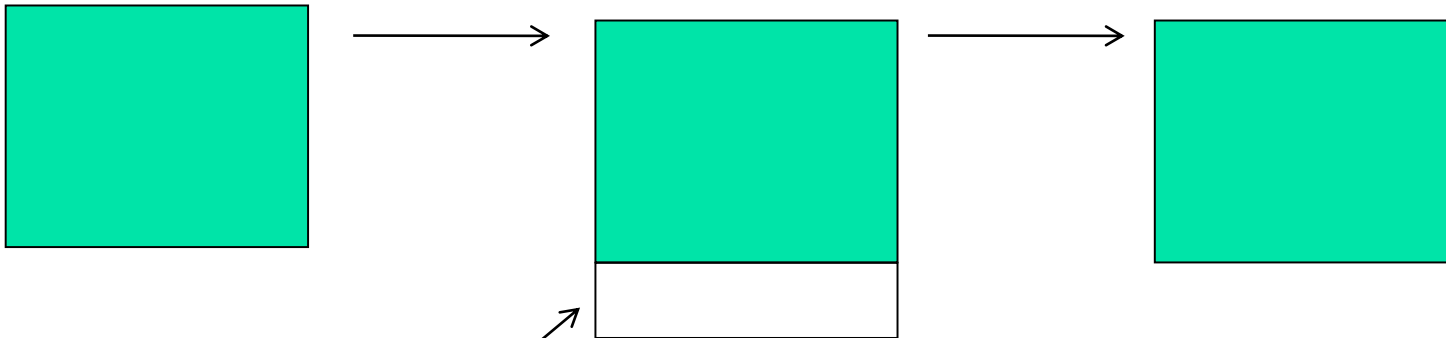


# History of the ac matrix

---

Procedure called

Procedure ends



Row paired  
with the new instance  
subject=new instance

Rows created and destroyed  
Rather than updated



# Least Privilege – Common solution

---

- The rights in the new row are a function of
  - The method private variables (they depends upon the variable types),
  - Input parameters (type of the parameters and the kind of access to the parameters)
- The program structure into classes/ methods defines the strategy to manage the rights granted to the subjects on the program data structures
- The programmer can choose the size of each protection domain
- Domain switching is handled in an automatic way



# Example

---

Op(x, y)

a : ....

- If two subjects (programs) invoke this op, each program has its own row, we have two local copies of a or one copy if shared variables are supported (depends upon the type of a)
- 
- Each row enables the program to access its own parameters and private (non shared) local variables
- If a static acm is adopted, the management of rights is rather more complex and access of a program to the parameters of the other program is simplified



# Least Privilege - Amplification

---

- The encapsulation of information requires that the set of rights of the invoked procedure differs from that of the invoker
- An example is a procedure in the run time support of an object oriented language that needs to know an object implementation
- Rights are amplified: provided that some rights are owned, other may be granted
- Amplification is misleading because we are interested in granting a distinct set of rights rather than a larger set



# Least privilege vs objects

---

- The least privilege principle assumes an object decomposition that is fully coherent with an object oriented methodology
- A simple object defines a small protection domain (a few variables) that can be simply managed
- Even if a simple object is successfully attacked, the attack has a low impact and it can be easily detected
- Sharing among objects should always be minimized



## Least privilege – message passing

---

- In a message passing environment, subjects are processes/threads interacting through ports or channels
- To satisfy the principle
  - Distinct ports supports the implementation of distinct operations,
  - Ports can be opened/closed (created/destroyed) for a set of users
  - If an interaction is legal, then the corresponding port is open/created
  - The port is closed/destroyed as soon as the interaction is no longer possible

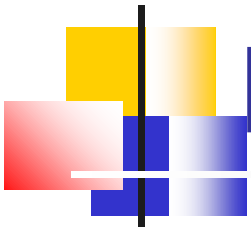


## Least privilege – message passing

---

- Closed port  $\neq$  (open port + mechanism to discard messages)
  - The overhead to discard messages is much lower if the port is closed or if does not exist
  - Messages can be discarded as they are routed
- If discarding is too expensive, the subject can do nothing because it is always busy to discard messages

(Denial of Service)



# Least Privilege – Unix - I

---

- OS like Unix violates this principle because root owns any right (the target of any attack)
- This strongly simplifies attacks, any procedure executed by root is a target, 2 steps escalations
- Management countermeasures such as having distinct administrators for a system
- Further technological countermeasures
  - logging any operation root invokes but not in the node
  - 2F authentication





# Least Privilege – Recording

---

- A log needs a read + append only file
- This may be guaranteed
  - In a physical way = print the file
  - Logical way = hash chain
- Blockchain
  - Replicated data structure
  - Each block includes
    - Hash of the records in the block
    - Hash of the previous and next block
    - Voting or proof of work



# Least Privilege – Unix - II

---

- Chroot constrains the access to the files by defining a new root of the file system
- Jail (BSD) makes it possible to constrain other operations such as network connection
- These are implementation of sandbox = a minimal environment for untrusted application



# Sandbox

---

- Definitely a bad idea
- Any sandbox has been violated
  - Chrome browser from Oct. 2008 more than 40 sandbox related vulnerabilities out of 1523 total
  - When the subject escapes the sandbox, no other countermeasure exist
- Two distinct problems
  - Discover the sandbox
  - Escape the sandbox
  - Current implementation=virtual machine



# Privileged Access Management

---

- They help secure, control, manage and monitor privileged access to critical assets.
- They take the credentials of privileged accounts – i.e. admin – and put them inside a secure repository (a vault) isolating the use of privileged accounts to reduce the risk of those credentials being stolen.
- System administrators need to go through the PAM system to access their credentials, at which point they are authenticated and their access is logged.
- When a credential is checked back in, it is reset to ensure administrators have to go through the PAM system next time they want to use the credential.



## P7- Least common mechanism

---

*Minimize the amount of mechanisms common to more than one user and depended on by all users*

- Mechanisms should not be shared
  - Information can flow along shared channels
  - Covert channels
- Isolation
  - Virtual machines
  - Host and Network Segmentation



## P7- Least common mechanism

---

- A powerful mechanism, if useful, should be decomposed into simpler ones
- When just one mechanism is used to implement distinct operations
  - If several subjects are granted the rights of invoking the mechanism they are also granted all the rights
  - This hides the fact that there are several distinct operations and several distinct rights
  - The least privilege cannot be satisfied



## P7 – Least common mechanism

---

- A solution that decomposes a complex operation into simpler ones can better satisfy separation of privilege and least privilege
- Simpler operations makes it possible to assign to each subject only the rights it needs, and it is entitled to
- Notice all S&S principles dictate some design rules, a design that cannot satisfy a rule will deliver a system with some vulnerabilities



## P8 - Psychological Acceptability

---

*The human interface should be designed for ease of use so that users routinely and automatically accept the protection mechanisms correctly*

or

*Do not adopt policies users will surely violate*

- Security mechanisms should not increase the complexity of accessing a resource
  - Hide complexity introduced by security mechanisms
  - Ease of installation, configuration, use
  - Human factors critical here





# Last two principles

---

- Recall they have been introduced because even if the other principles are satisfied a vulnerability may arise
- The principles are useful if some attacks are successful in spite of the adoption of the previous principles
- Anticipate the presence of vulnerabilities and possible failures



## P9 – Work factor

---

*Compare the cost of circumventing the mechanism with the resources of a potential attacker*

- The probability of a successful attack increases with the resources the attacker can access
- The cost of circumventing a mechanism is the attacker work
- A mechanism is better than another if it can be defeated only through a larger amount of work
- Several mechanisms can be defeated only by indirect strategies, such as waiting for a hardware failure
- Reliable work estimates are very complex anytime several attacks (attack chain) are required to build an intrusion and violate a system



## P9 – Work factor

---

- Most attacks require a privilege escalation
- The number of attacks in escalations and their attributes determine the attacker work
- Attributes
  - Success probability
  - Automated or not
  - Wait for some external condition
- The work also include the one to discover information about the system remember the open design principle



## P9 – Work factor

---

- Adversary emulation returns because to estimate the amount of work we need to mimic the behaviour of the attacker
- Multiple independent emulations are required because several attacks have a success probability smaller than 1 (simulations rather than emulation)
- An accurate evaluation of the work may require a large number of simulations



## P10 – Compromise recording

---

*Mechanisms that reliably record a compromise of information may replace more elaborate ones that completely prevent loss*

- A mechanism supports the discover of unauthorized use if it produces a tamperproof record that is reported to the owner,
- It is difficult to guarantee discovery and forensics after an attack and a control from the outside (more than 250 days to discover an attack)
- Logical damage (and internally stored records of tampering) can be undone by a clever attacker



## P10 – Compromise recording

---

- Useful to collect information about attempted and successful attacks, goals and threat
- Collected information can be used to support
  - a forensics analysis
  - the evaluation of the system robustness
  - more accurate analyses in a risk assessment



# Compromise recording

---

- A log file that records, at least, any of
  - Login attempt
  - Failed login
  - Access to critical resources
- Protection of log file
  - write once memory (e.g. paper)
  - insert a sequence number to discover log manipulation
  - insertion in a record of a value that is a function of all the previous records (hash pointer or blockchain)
- Forensics = the file should be structured so that it can be used to prosecute the attacker and as a legal source of evidence in an investigation (timestamps ...)



# Logging policies

---

What happen when a file is full?

- Throw away – the file is destroyed
- Reset – rotation within a file
- Rotate – rotation among several files
- Compress and archive – stored in a low cost memory

(some laws require that some data are preserved)





# Throwing away log files

---

- The worst solution
  - The files are a source of evidence and of information about security
  - They may also be useful for safety
- Even if the law entitles us to destroy the logs shortly after they are collected, it is better to preserve them for at least one year
  - The average time to discover an intrusion is more than 270 days



# Rotating log files

---

- N distinct files
  - Logfile.1 , logfile.2, ... logfile.n
- Each day a distinct file is used



# Compress and archive

---

- Better solution that takes into account
  - Forensics investigation
  - Commercial problems with clients, suppliers
- Log are copied onto low cost, removable memory devices

The logo for Syslog, featuring a vertical line and several overlapping squares in yellow, red, and blue. The word "Syslog" is written in a blue, sans-serif font to the right of the graphic.

# Syslog

---

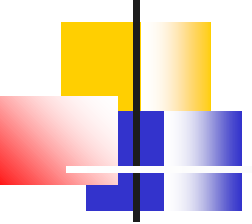
- A logging system to store information produced by the kernel and by system utilities
- It enables a classification of log messages according to the source and the event critical level
- Messages can be addressed to (stored in) several destinations



# Syslog: 3 elements

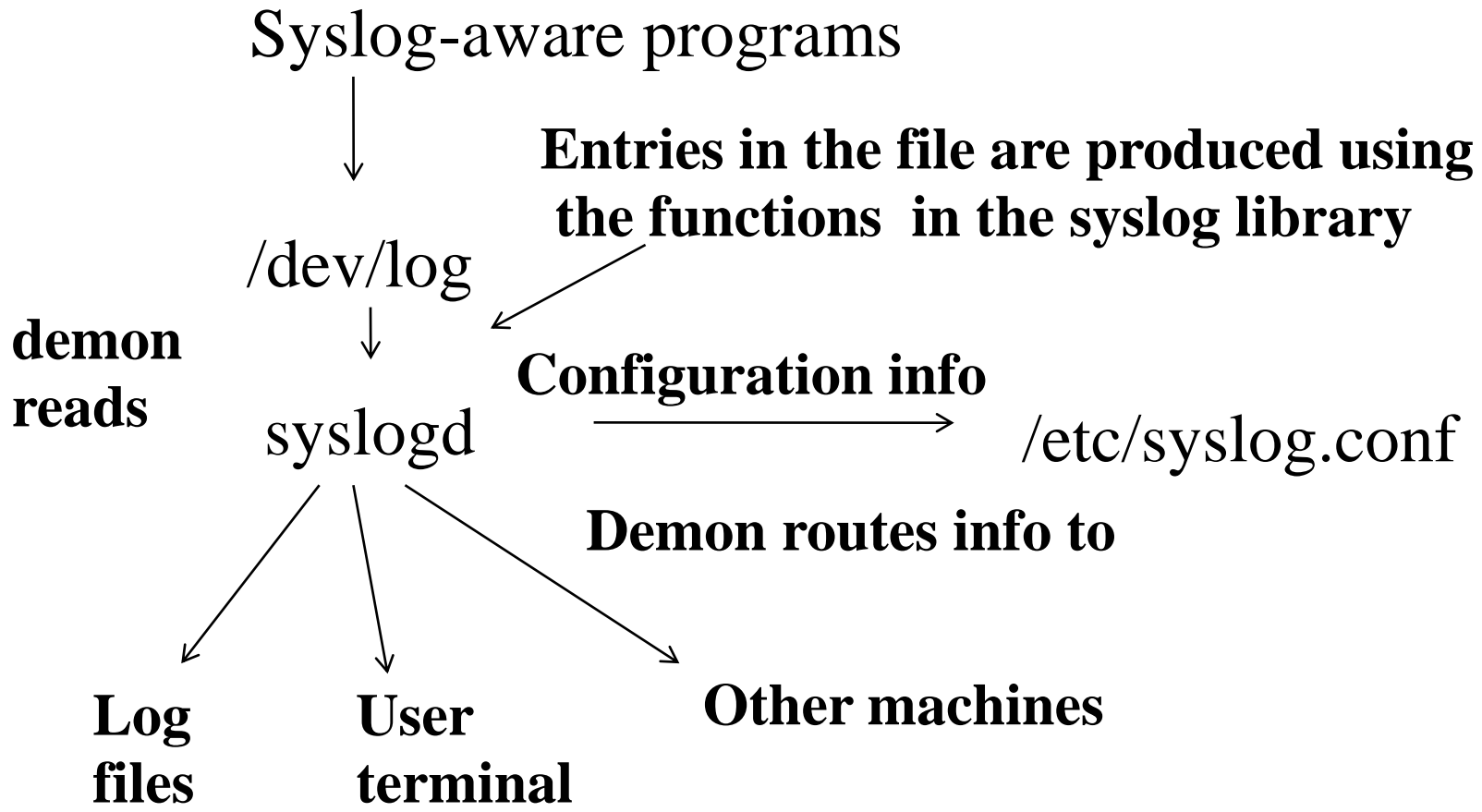
---

- Syslogd /etc/syslog.conf
  - A demon that implement the logging
  - Programmed through a configuration file
- openlog, syslog, closelog
  - Procedures to produce event to be logged
- logger
  - User command to produce a log



# Syslog

---

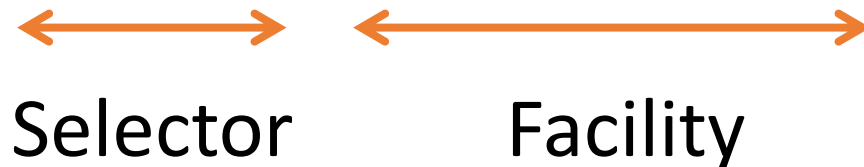




# Syslogd: configuration

---

- Configuration info in `/etc/syslog.conf`
- A text file
  - White lines and those beginning with `#` are ignored
  - ***Selector*** <TAB> ***action***  
as in `mail.info /var/log/maillog`





# Selector

---

- Identifies

- The source – the program (`facility`) that is transmitting the message
- The message `severity level`

- Syntax

- `facility.level`
- facility names and severity levels have to be selected in a predefined set





# Facility names

---

Facility

Used by

kern

kernel

user

user process, default

mail

mail system

daemon

System daemons

auth

Security and authorization  
related commands

lpr

printer spooling system

news

Usenet news system



# Facility names

---

Facility	Used by
uucp	UUCP
cron	cron daemon
mark	Timestamps produced with a fixed frequency
local0-7	local message
syslog	syslog internal messages
authpriv	Private or system messages
ftp	ftp daemon, ftpd
*	further facilities



# Severity level

---

Level

That means approx.

emerg (panic)

Panic situation

alert

Urgent situation

crit

Critical condition

err

other errors

warning

warning

notice

worth an analysis

info

info

debug

debugging info





# Selector

---

- Several facilities separated by ``,'`
  - `daemon,auth,mail.level` action
- The composition of several selectors by ``;'`
  - `daemon.level1; mail.level2` action
- The OR composition of selectors is expressed through ``|'` – `un` a message matches if it matches at least one selector
- ``*' or `none'`, (all or none) can be used



# Selector

---

- The level defines the lowest level of a logged message
  - mail.warning, matches any message from the mail system with a level that is, at least, warning
- 'none' is used to neglect some facilities .
  - \*.level1;mail.none action  
any facility with a level not smaller than level1 but neglect the mail facility



# Action: message handling

---

## Action

## That means

filename

Append the message to a local file

@hostname

send the message to hostname

@ipaddress

send the message to the node with the specified IP address

user1, user2,...

write the message on the screen of any of these users if the user is logged

\*

write the message on any screen



Program	Facility	Levels	Description
amd	auth	err-info	NFS automounter
date	auth	notice	Display and set date
ftpd	daemon	err-debug	ftp daemon
gated	daemon	alert-info	Routing daemon
gopher	daemon	err	Internet info server
halt/reboot	auth	crit	Shutdown programs
login/rlogind	auth	crit-info	Login programs
lpd	lpr	err-info	BSD line printer daemon



Program	Facility	Levels	Description
named	daemon	err-info	Name sever (DNS)
passwd	auth	err	Password setting programs
sendmail	mail	debug-alert	Mail transport system
rwho	daemon	err-notice	remote who daemon
su	auth	crit, notice	substitute UID prog.
sudo	local2	notice, alert	Limited su program
syslogd	syslog, mark	err-info	internet errors, timestamps





# syslog

---

- `openlog ( ident, logopt, facility );`
  - Messages are logged as specified by `logopt`
  - They all begin with `ident`
- `Syslog ( priority, message, parameters... );`
  - `message` is sent to syslog, that logs it according to `priority level`
- `close ( );`



# Logopt

---

- LOG\_CONS  
Write directly to system console if there is an error while sending to system logger.
- LOG\_NDELAY  
Open the connection immediately (normally, the connection is opened when the first message is logged).
- LOG\_NOWAIT  
Don't wait for child processes that may have been created while logging the message.
- LOG\_ODELAY  
The converse of LOG\_NDELAY; opening of the connection is delayed until syslog() is called. (This is the default, and need not be specified.)
- LOG\_PERROR  
(Not in POSIX.1-2001.) Print to stderr as well.
- LOG\_PID  
Include PID with each message.



# Blockchain vs Log

---

- By storing the information on a blockchain rather than in a file we increase the complexity of an attack against the log
- Distinct servers store a copy of the log and each also run the mining algorithm
- Attacking one server is not enough
- The mechanisms that protect the blockchain simplify the usage of its information in a forensics investigation and in a legal one



## Security vs ICT security

---

- All the principles previously discussed do not fully characterize ICT security
- The two peculiar features of ICT security are
  - Automatic attack
  - The virtual machine hierarchy



# Virtual machine hierarchy

---

- Any ICT system is a hierarchy of virtual machines
- Each virtual machine
  - Defines a set of mechanisms = a programming language
  - These mechanisms abstract and hide those of the underlying machine
  - Any machine can be a standard one, with all the consequent implications on vulns



# Why ICT security is difficult?

---

- Vulns may be discovered in the specs and in the implementation of a virtual machine VM
- Vulns cannot be abstracted because a vulnerability in VM results in attacks against any machine of the stack that runs on top of VM
  - ⇔ a vuln in the hardware architecture makes it possible to attack any VM running on it
  - ⇔ a vuln in the OS enables attacks against any application it supports

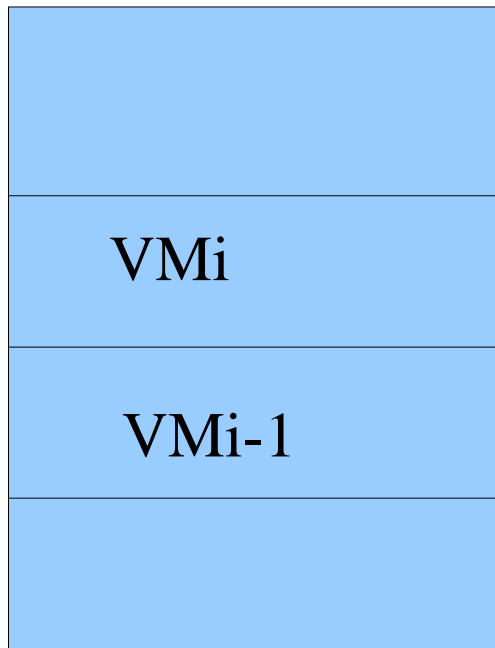


# Going down

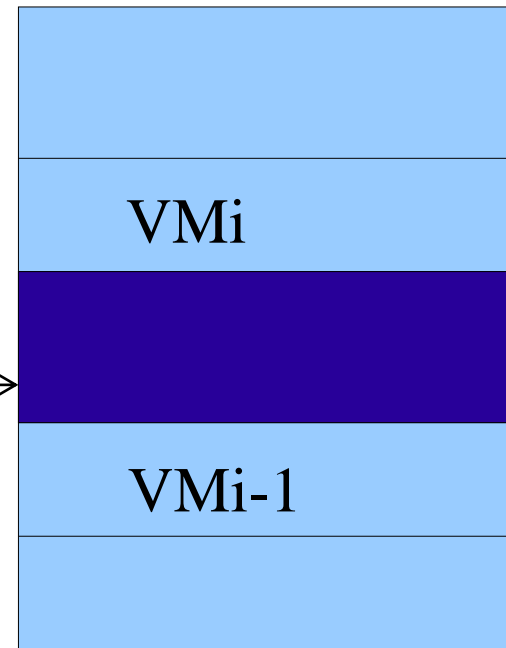
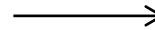
---

- A trend in attack is targeting low level virtual machines
- By controlling a low level of the hierarchy any higher level can be manipulated
- An interesting attack is the one that inserts a further virtual machine in the hierarchy
  - Difficult to be detected
  - High impact from a security perspective

# Blue Pill Attack



New  
Virtual  
Machine







# Blue Pill Attack

The new machine can

- return fake information about the system states to upper layer virtual machines
- transmit to the underlying machines commands that differ from those received by higher VMs
- *Machine in the middle*, a generalization of man in the middle
- Some examples in the next slides

# Another blue pill attack ...




ITRE SOFTWARE SECURITY DEVOPS BUSINESS PERSONAL TECH SCIENCE EMERG

{\* SECURITY \*}

## 'Unfixable' boot ROM security flaw in millions of Intel chips could spell 'utter chaos' for DRM, file encryption, etc

Although exploitation is like shooting a lone fish in a tiny barrel 1,000 miles away

By Shaun Nichols in San Francisco 5 Mar 2020 at 14:00

145  SHARE ▼

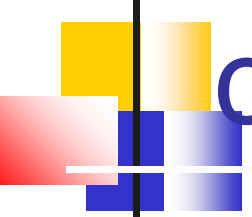


# Hardware vulnerabilities (13<sup>th</sup> of March, 2018)

---

CTS has been researching the security of AMD's latest Zen processors for the past six months and has made concerning discoveries:

1. The AMD Secure Processor, the gatekeeper responsible for the security of AMD processors, contains critical vulnerabilities. This integral part of most of products, including workstations and servers, is currently shipped with multiple vulnerabilities that could allow attackers to permanently install malicious code inside the Secure Processor itself. These vulnerabilities could expose customers to industrial espionage that is virtually undetectable by most security solutions.
2. A set of security vulnerabilities in the Secure Processor could allow attackers to steal network credentials – even on systems guarded by Microsoft's latest Credential Guard technology. This could allow attackers to spread through otherwise secure and up-to-date corporate networks



# Hardware vulnerabilities (13<sup>th</sup> of March, 2018)

---

Secure Encrypted Virtualization, a key security feature that AMD advertises as one of its main offerings to cloud providers could be defeated as soon as attackers obtain malicious code execution on the EPYC Secure Processor.

The Ryzen chipset, a core system component that AMD outsourced to a Taiwanese chip manufacturer, ASMedia, is currently being shipped with exploitable manufacturer backdoors inside. These backdoors could allow attackers to inject malicious code into the chip. The chipset is a central component on the motherboard, responsible for linking the Ryzen processor with hardware devices such as WiFi and network cards, making it an ideal target for attackers.

# Hardware vulnerabilities (13<sup>th</sup> of March, 2018)

---

Vulnerabilities	Impact
<b>MASTERKEY-1</b>	<ul style="list-style-type: none"><li>▪ Persistent malware running inside AMD Secure Processor</li></ul>
<b>MASTERKEY-2</b>	<ul style="list-style-type: none"><li>▪ Bypass firmware-based security features such as <i>Secure Encrypted Virtualization (SEV)</i> and <i>Firmware Trusted Platform Module (fTPM)</i></li></ul>
<b>MASTERKEY-3</b>	<ul style="list-style-type: none"><li>▪ Network credential theft. Bypass <i>Microsoft Virtualization-based Security (VBS)</i>, including <i>Windows Credential Guard</i></li><li>▪ Physical damage to hardware (SPI flash wear-out, etc.)</li><li>▪ Affects: <i>EPYC, Ryzen, Ryzen Pro, Ryzen Mobile</i>. Successfully exploited on <i>EPYC</i> and <i>Ryzen</i></li></ul>

# Hardware vulnerabilities (13<sup>th</sup> of March, 2018)

- RYZENFALL-1 FALLOUT-1**
- Write to protected memory areas, including:
    - Windows Isolated User Mode and Isolated Kernel Mode (VTL1)
    - AMD Secure Processor Fenced DRAM – Allows direct tampering with trusted code running on AMD Secure Processor. Only applicable to select Ryzen motherboards
  - Network credential theft. Bypass *Microsoft Virtualization-based Security (VBS)* including *Windows Credential Guard*
  - Enables memory-resident VTL1 malware that is resilient against most endpoint security solutions
  - Affects: *EPYC, Ryzen, Ryzen Pro, Ryzen Mobile*. Successfully exploited on *EPYC, Ryzen, Ryzen Pro* and *Ryzen Mobile*

- RYZENFALL-2 FALLOUT-2**
- Disable *Secure Management RAM (SMRAM)* read/write protection
  - Enables memory-resident SMM malware, resilient against most endpoint security solutions
  - Affects: *EPYC, Ryzen, Ryzen Pro*. Successfully exploited on *EPYC, Ryzen, Ryzen Pro*. *Ryzen Mobile* is not affected

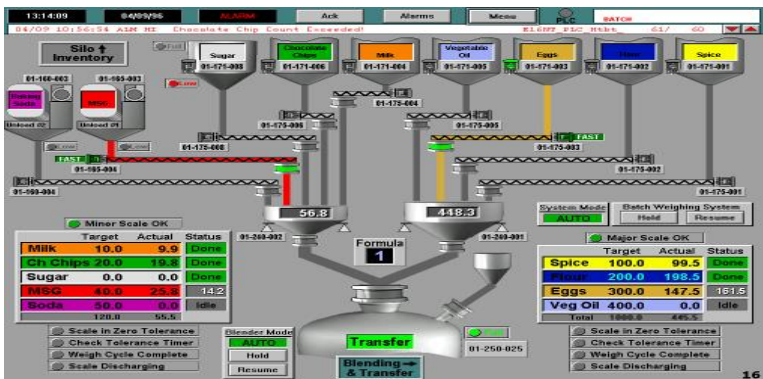
- RYZENFALL-3 FALLOUT-3**
- Read from protected memory areas, including:
    - Windows Isolated User Mode and Isolated Kernel Mode (VTL1)
    - Secure Management RAM (SMRAM)
    - AMD Secure Processor Fenced DRAM. Only applicable to select Ryzen motherboards
  - Network credential theft. Bypass *Windows Credential Guard* by reading secrets from VTL1 memory
  - Affects: *EPYC, Ryzen, Ryzen Pro*. Successfully exploited on *EPYC, Ryzen, Ryzen Pro*. *Ryzen Mobile* is not affected

- RYZENFALL-4**
- Arbitrary code execution on AMD Secure Processor
  - Bypass firmware-based security features such as *Firmware Trusted Platform Module (fTPM)*
  - Network credential theft. Bypass *Microsoft Virtualization-based Security (VBS)*, including *Windows Credential Guard*
  - Physical damage to hardware (SPI flash wear-out, etc.)
  - Affects: *Ryzen, Ryzen Pro*. Successfully exploited on *Ryzen, Ryzen Pro*.

- CHIMERA-FW CHIMERA-HW**
- Two sets of manufacturer backdoors: One implemented in firmware, the other in hardware (ASIC)
  - Allows malware to inject itself into the chipset's internal *8051 architecture* processor
  - The chipset links the CPU to USB, SATA, and PCI-E devices. Network, WiFi and Bluetooth traffic often flows through the chipset as well
  - Malware running inside the chipset could take advantage of the chipset's unique position as a middleman for hardware peripherals
  - Affects: *Ryzen, Ryzen Pro*. Successfully exploited on *Ryzen* and *Ryzen Pro*.

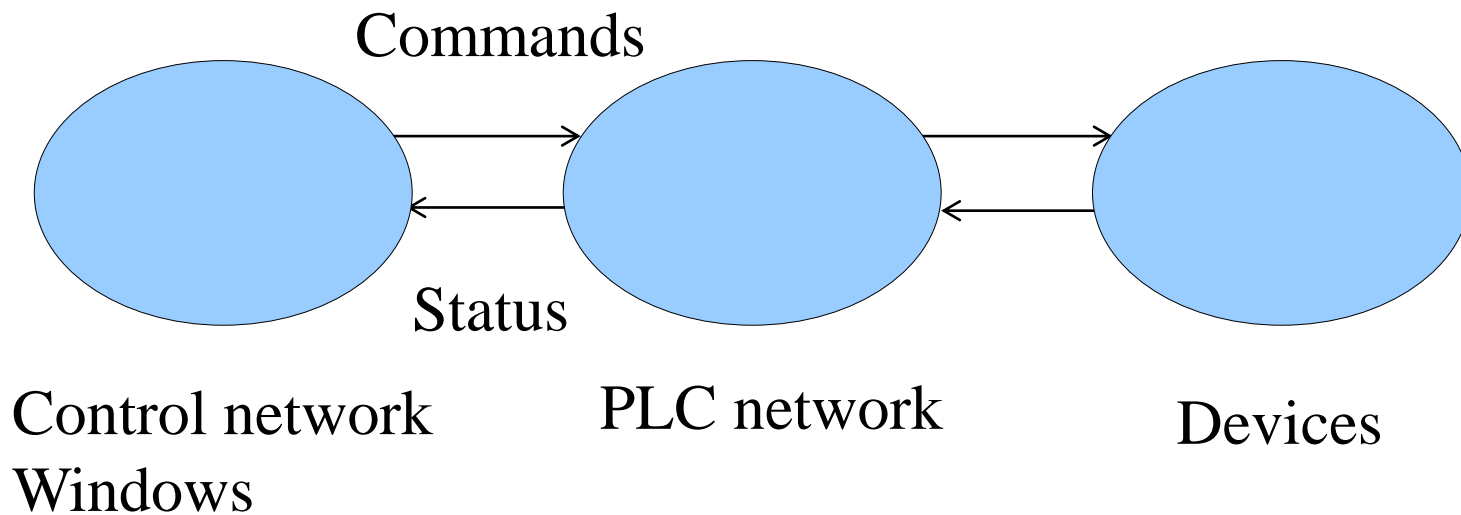
# Industrial Control Systems and IOT

- Run automated processes on factory floors, power and chemical plants, oil refineries, etc.
- Specialized assembly code on PLCs (Programmable Logic Controllers)
- PLCs are usually programmed from Windows
- Not connected to the Internet (“air gap”)



# Industrial Control Systems

- PLC sits inbetween the control network and the actual devices
- Programmed to control the devices







# Cyber physical attacks

---

- Cyber physical attacks cannot be accomplished by just understanding computer code and vulnerabilities.
- To implement a a cyber-physical attack, one has to understand the physical part as well – the design features of the plant to attack and of the process parameters of this plant.
- Different from cyber attacks, a cyber-physical attack involves three layers and their specific vulnerabilities:
  - The IT layer which is used to spread the malware,
  - the control system layer which is used to manipulate (but not disrupt) process control
  - the physical layer where the actual damage is created.



# Stuxnet Attack Vector

---

- Two different attack vectors against centrifuges
- Overpressure Attack
  - Increase centrifuge rotor stress
  - Significantly stronger
  - More stealthy
  - Less documented in literature
- Rotor Speed Attack
  - Increase/decrease rotor velocity
  - Overpressure centrifuge is dormant in this attack
  - Independent from previous attack
  - Less concern about detection



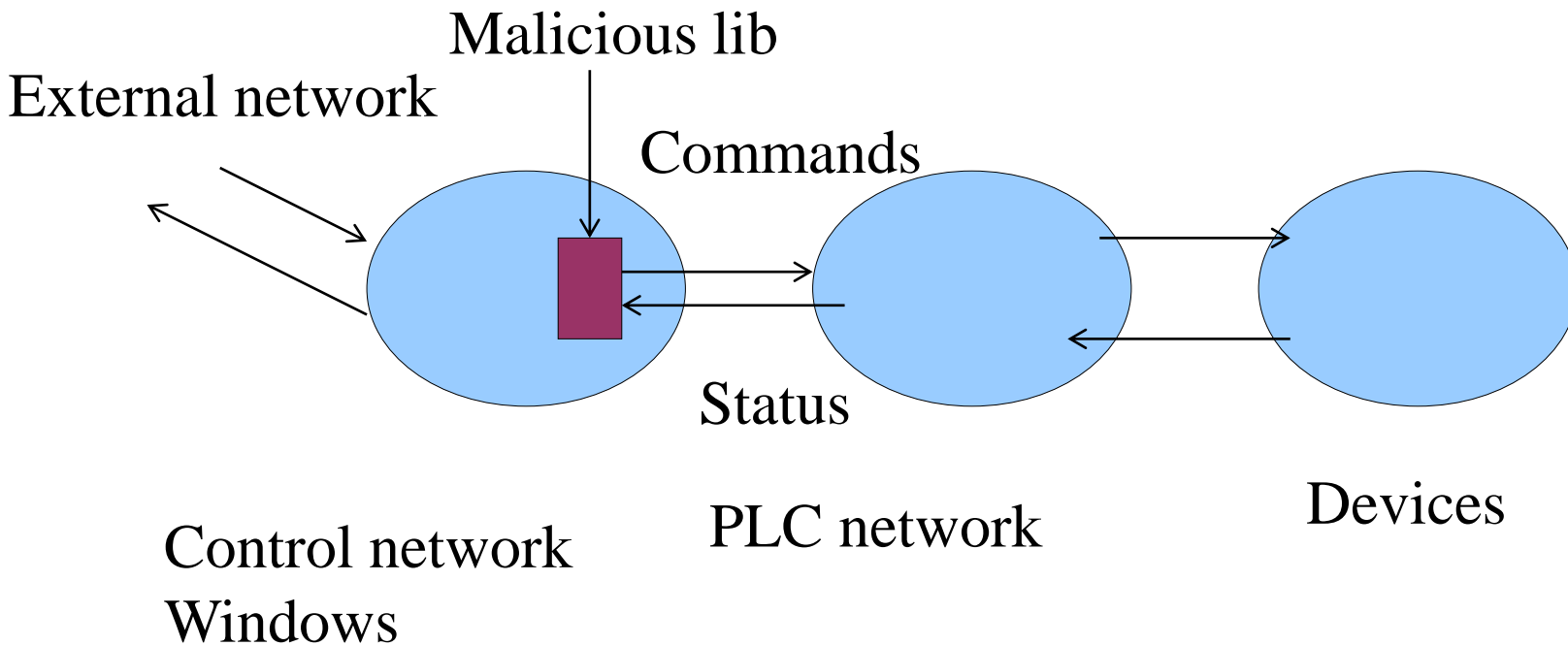
# Not only the Attack Vector

---

- One of the side effect of the attack was the update of a library
- Every time the operator checks the pressure and the speed of the rotor the library returns the correct values it has copied before starting the attack
- In this way the operator has no mechanism to discover the ongoing attack
- The transmission of erroneous commands continues till the rotors are completely crashed (their central axes becomes elliptical)

# In the middle

- Malicious library
- Unreliable interactions





# Spectre and Meltdown

---

If  $x > 0$  and  $x < 100$  then

if  $a[x] = 1$  then  $b[1] = 0$  else  $c[1] = 0$

- Compiled with a branch, assume  $x = 9999$
- Branch prediction invoked but in parallel  $a[x]$  is read and one of the two assignments is executed
- If neither  $b$  nor  $c$  are in the cache one will be loaded
- If condition is violated assignment is undone but the side effect in the cache is not undone
- The following instructions can measure the time to access  $b$  and  $c$  to discover the value of  $a[x]$



# Spectre and Meltdown

---

- Speculative execution may result in the execution of statements that violate a security check
- Undoing the statements is not enough, even if correctly implemented because it may leave some side effects in an architectural component
- Undoing all the side effects strongly reduce the benefits of speculative execution
- Fence = hardware synch that stop speculative execution may result in a 400% overhead



# Side-channel attacks

---

any attack based on information gained from the physical implementation of a computer system, rather than weaknesses in the implemented algorithm itself

– from the Wikipedia definition

## Examples

- Monitoring a machine's electromagnetic emissions (“TEMPEST”-like remote attacks)
- Measuring a machine's power consumption (differential power analysis)
- Timing the length of operations to derive machine state



# Caches as side channels

---

- Caches exist to provide faster access to frequently used data
- Data closer to the cores take fewer cycles to access while data further away take more cycles to access
- This difference can be measured by software
- Consequently it is possible to determine whether a specific address is in the cache
- Calibrate by measuring access time for known cached/not cached data
- Measure time access to a memory location and compare with calibration





# After showing some hw vulns

---

Let us go back to the hierarchy of Vms  
to discuss how it impacts the robustness  
of the overall system

or

Security as an holistic property across  
hierarchy levels



# Hierarchy and robustness - I

---

## Robustness at any level

- Each VM should include the checks on the subjects and the objects of the corresponding level
- The check mapping to the VMs is the simplest way to minimize the overall overhead
- This also guarantees that the checks of a VM cannot be violated by working at a lower level

If this strategy is not applied then either

- A VM does not execute any checks

or

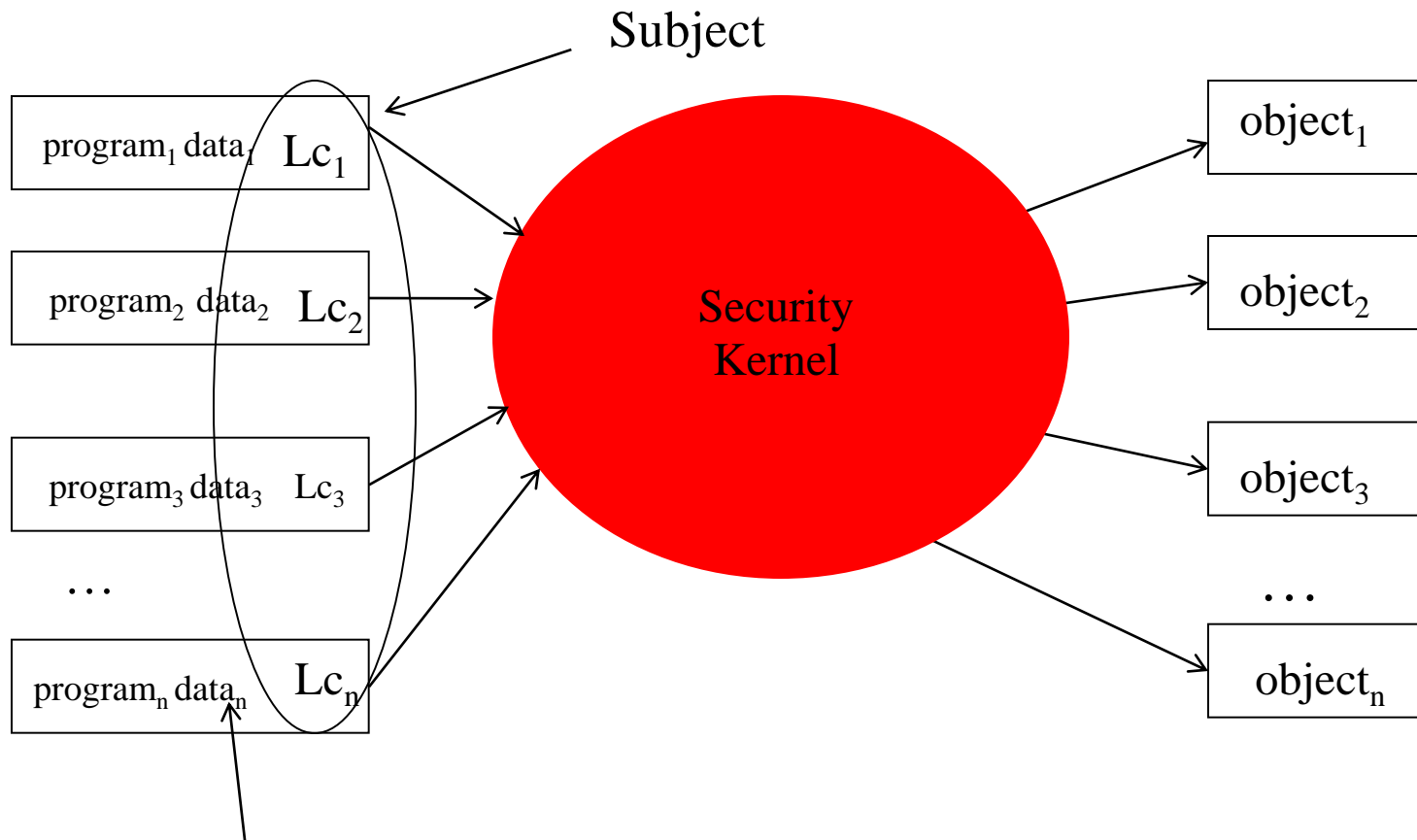
- The checks of a VM are delegated to another one but this increases the overall complexity

Redundancy = checks are repeated in distinct VMs

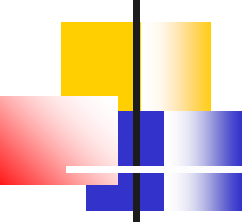


# Capability

---



The implementation of a subject



# Example - Capability

---

- VM(L), the machine at level L adopts a capability based solution to manage the rights of a subject
- VM(L-1), the underlying machine at level L-1
  - Implements the subjects and the objects of VM(L)
  - Manages some further objects that implement the capabilities of VM(L)
- The acm of VM(L-1) should guarantee that the subjects of VM(L) cannot exploit the subjects at level L-1 to manipulate their capabilities at level L



# Hierarchy and robustness - II

---

- Security policy and mechanism modularity in a hierarchy of VMs:
  - any VM defines a set of mechanisms that may be freely composed by the VM users to implement a security policy
  - ⇔
  - Each VM exploits some assumptions on the security of these mechanisms that has to be guaranteed by at least one of the underlying VMs
  - Example: to prevent the manipulation of a capability we can apply
    - Encryption
    - Protection of a memory segment
    - Protection of a data structure
    - ....
- A distributed implementation of the TCB by several VMs



# Hierarchy and robustness - III

---

- The robustness of a VM is a function of the robustness of the underlying VMs
- Even functionally equivalent machines have a very different robustness because of
  - The implementation of the machine
  - The implementation of the underlying machines

Robustness does not agree with abstraction  $\Rightarrow$

Robustness can be evaluated only in terms of the implementation



# A common problem: example

---

- A memory area  $A$ , in some memory of  $VM_i$  is shared among several applications by distinct users of a  $VM_i+k$
- The applications that share the area may be not know in advance because they depend upon the users that are sharing  $VM_i$  (user of a public cloud architecture)
- An application that can access the area  $A$  can read in it some values left by another application or by another user
- This shows why cloud security is a big problem



# Solution

---

- Any memory area that is either
  - released by an application or
  - garbage collected

has to be reinitialed to avoid any illegal information flow between two applications  
(covert channel)

- This holds for any memory area
  - cache,
  - main memory,
  - secondary storage





# Solution

---

- In a system with severe security requirements, all the resources are partitioned into pools each with a distinct level
- The resources in a pool with a security level are shared only among applications run by users with the same security level
- Sharing is constrained to minimize unanticipated flow of information among applications with distinct security levels
- A constrained sharing reduces efficiency and increases cost



# A general principle ...

---

- The previous example shows that sharing should be avoided or at least minimized to improve the security of a system
- A secure system
  - is as simple as possible
  - avoids sharing as much as possible
- This explains why a secure system is more expensive of a less secure one



# Examples

---

- Memory segments are partitioned into subsets, each paired with a security level
- Traffic segregation = it partition network connections into subsets, security critical information is transmitted only along some lines
  - Switchs rather than hubs
  - Partitioning of virtual lines created by tagging or by encryption
  - Distinct transmission frequencies but low security
- It is important to understand that any system manages at least two levels of information



# Two security levels

---

- User information
- Information to implement the security policy
- Distinct mechanisms have to be applied to protect the two kinds of information



# Example

---

- A sniffer on a communication line reads any information transmitted along the line
- If a user information is transmitted the sniffer can read the information
- If a user password is transmitted and read by the sniffer then all the user information is lost



# Sharing and Cloud

---

- Cloud architectures result in large savings because they are based upon pools of resources shared among users
- Elasticity = when a resource is not used it can be granted to any user that requires it
- What happens when a resource passes from one user to another one?



# Cloud Management

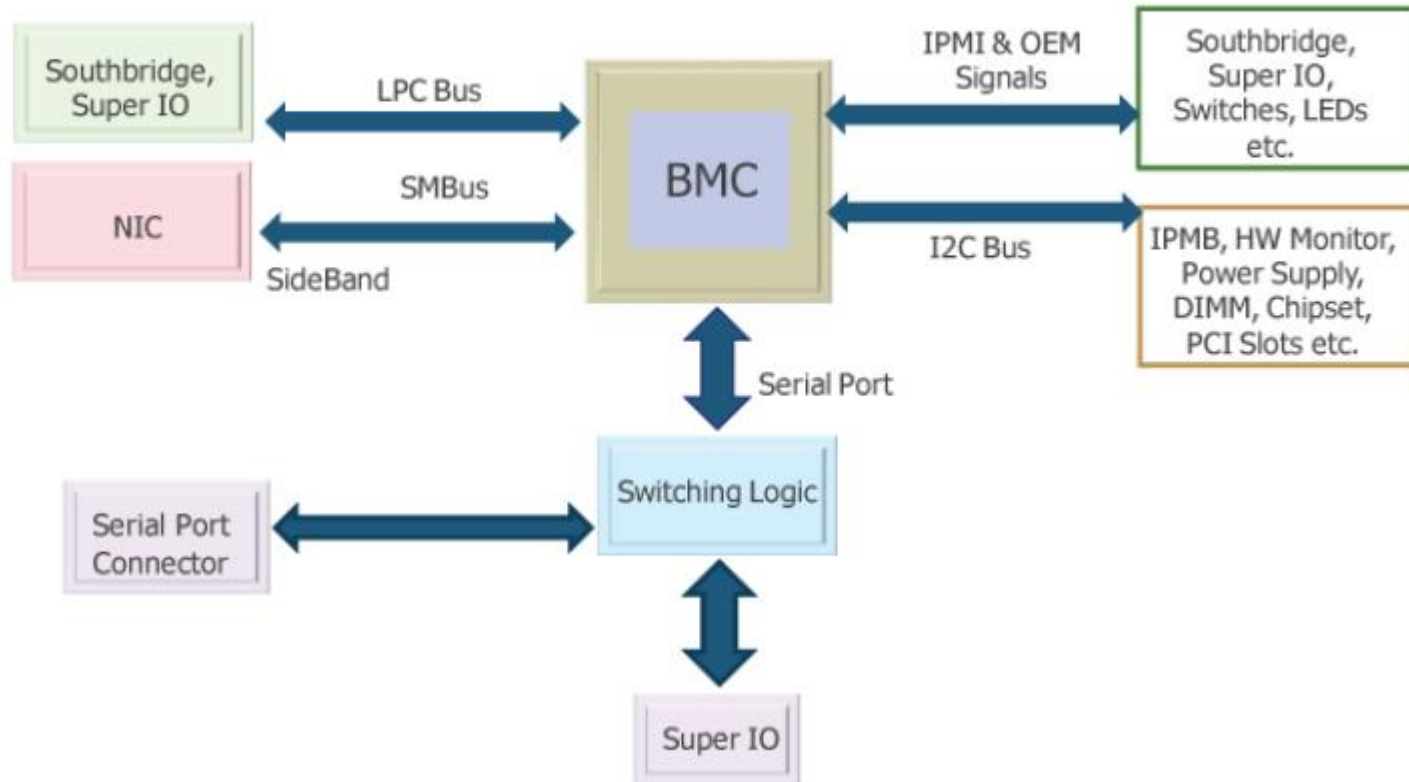
---

- Intelligent Platform Management Interface (IPMI) is a set of computer interface specifications for an autonomous computer subsystem that provides management and monitoring capabilities independently of the host system's CPU, firmware (BIOS or UEFI) and operating system.
- IPMI defines a set of interfaces that system administrators use for out-of-band management and monitoring of computer systems
- IPMI provides a way to manage a computer that may be powered off or otherwise unresponsive by using a network connection to the hardware rather than to an operating system or login shell. Another use case may be installing a custom operating system remotely.

# Cloud Management

## Baseboard Management Controller (BMC)

The embedded micro-controller: the second CPU







# Cloud Management


---

Scanned all public IPs on May 7, 2013 using ZMap\*.

Downloaded all X.509 certs from HTTPS servers.

Used identifying characteristics of default certificates.†

Platform	Devices on Public IPs
Supermicro IPMI	41,545
Dell iDARC	40,413
HP iLO	23,376
Total	105,334



Could root  
all these in  
parallel in  
minutes!



# Cloud and Motherboard

---

- Chinese government agents sneaked spy chips into Super Micro servers used by Amazon, Apple, the US government giving Beijing's snoops access to highly sensitive data, according to a Bloomberg report.
- The story had a huge impact on the markets: Super Micro, saw its share price drop by nearly 50 per cent; Apple's share price dropped by just under two per cent, and Amazon's dropped by more than two per cent.
- According to the report, tiny microchips that were made to look like signal conditioning couplers were added to Super Micro data center server motherboards manufactured by sub-contractors based in China.
- Those spy chips contained enough memory and processing power to effectively backdoor the host systems so that outside agents could, say, meddle with the servers and exfiltrate information.



# Cloud and Motherboard

---

- The spy chip could have been placed electrically between the baseboard management controller (BMC) and its SPI flash or serial EEPROM storage containing the BMC's firmware. When the BMC fetches and executes its code from this memory, the chip can intercept the signals and modify the bitstream to inject malicious code into the BMC processor, allowing its masters to control the BMC
- The BMC is a crucial component on a server motherboard. Using it the administrators can remotely monitor and repair machines, typically over a network, without having to find the box in a data center, physically pull it out of the rack, fix it, and re-rack it.
- The BMC and its firmware can power-cycle the server, reinstall or modify the host operating system, mount additional storage with malicious code and data, access a virtual keyboard and terminal connected to the computer, and so on. Who can reach the BMC and its software, total controls the box.

# The chip ...

The Big Hack

