



# Alternative approaches

---

- Unconditional security
  - Defend a system against any threat agents
- Conditional security (risk management)
  - Protect what from whom?
  - What resources, information we want to protect
  - Protect from which adversaries??
  - These adversaries (agents) what attacks can do?  
Adversary emulation
  - How we stop these attacks?



# Risk analysis

---

A modern approach to security:

1. Asset analysis (resources to be protected)
2. Vulnerability analysis
3. Attack analysis
4. Threat analysis (threat intelligence)
5. Impact analysis (damages)
6. Risk management =
  1. Classify risk
  2. Define acceptable risk
  3. Select and implement countermeasures



# Asset Analysis

---

- Which logical and physical resources of the ICT system we want to protect
- Who is entitled to access these resources and which operation they are entitled to invoke
  - Who is entitled to read an information
  - Who is entitled to update an information
  - Who is entitled to run a given application
  - .....
- The analysis defines the goal of our strategy: which resources are we going to defend



# Risk analysis and management

---

- Not all the attacks are worth preventing
- Economy driven solution = Which attacks
  - can be prevented
  - is worth preventing = defence cost less than impact
- Very few complete and quantitative methodologies are currently available and several under development
- Quantitative approaches are fundamental
- Several partial solutions to be integrated



# Attack and intrusion

---

- Some preliminary definition
- Some classification and terminology
- Main differences between cybersec and security in distinct fields
- Intrusion = the whole sequence of actions of the attacker (adversary) to reach a goal
- Attack = a single action



# Local vs remote attack

---

- An attack is
  - Local if it can be executed only when and if the attacker (threat agent) can access a local account
  - Remote if it can be executed from another node and so the attacker does not need a local account
- A remote attack is obviously more dangerous and it is the basis for worms etc.



# Automated attack

---

- No human action is required, to execute the attack, just code, the exploit, is executed
- This is the most dangerous kind of attacks
- Automated attacks characterize ICT security with respect to security in other fields
  - The time to execute an automated attack is neglectable
  - An attacker with no know how or abilities can execute an exploit
  - An attack platform is a software tool to implement an intrusion without involving a human

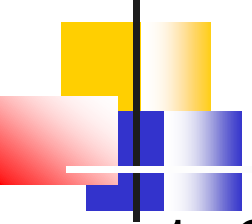


# Automated Attack and Malware

---

- A malware is a software designed to attack a system and install some other software
- Sometimes this installation requires the user cooperation (phishing) to install the malware on the target system (attack vector is a human)
- A computer worm has an attack vector that is a software that remotely attack other nodes to replicate itself onto these nodes
- Payload of a worm = A software module installed on a node by the attack vector at at the end of the attack





# The steps of an intrusion (kill chain)

---

1. Collection of information about a system
2. Discovery of system vulnerabilities (can be automated)
3. Search or build of a program (=exploit) to implement the attack (even partially)
4. **Implementation of the attack** Execution of the exploits + Execution of human actions
5. Install tools to control the system (persistence)
6. Remove any attack trace on the system
7. Access, update, control a subset of the system information

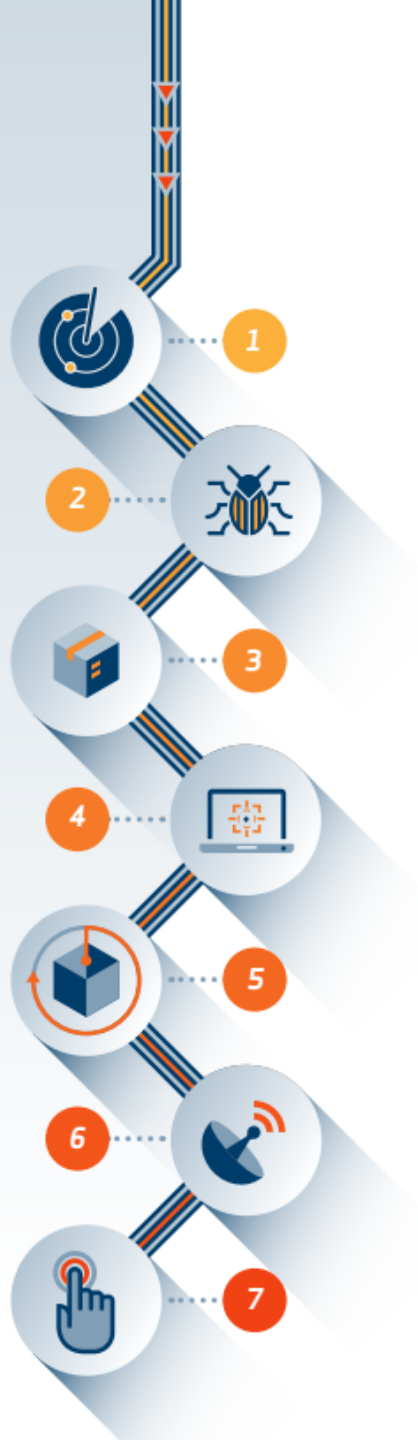
# THE LOCKHEED MARTIN CYBER KILL CHAIN®

The Cyber Kill Chain® framework is part of the Intelligence Driven Defense® model for the identification and prevention of cyber intrusions activity. The model identifies what the adversaries must complete in order to achieve their objective.

Stopping adversaries at any stage breaks the chain of attack! Adversaries must completely progress through all phases for success; this puts the odds in our favor as we only need to block them at any given one for success. Every intrusion is a chance to understand more about our adversaries and use their persistence to our advantage.

The kill chain model is designed in seven steps:

- ▶ Defender's goal: understand the aggressor's actions
- ▶ Understanding is Intelligence
- ▶ Intruder succeeds if, and only if, they can proceed through steps 1-6 and reach the final stage of the Cyber Kill Chain®.





# Description of the seven steps

---

Step 1: Reconnaissance. The attacker gathers information on the target

Step 2: Weaponization. The attacker creates a malicious exploit .

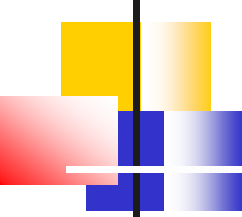
Step 3: Delivery. The attacker sends the malicious exploit to the victim by email or other means.

Step 4: Exploitation. The actual execution of the exploit

Step 5: Installation. Installing malware on the infected computer is relevant only if the attacker used malware. The installation is a point that takes months to operate.

Step 6: Command and control. The attacker creates a command and control channel in order to operate his internal assets remotely.

Step 7: Action on objectives. The attacker performs the steps to achieve his actual goals inside the victim's network.



# The steps of an intrusion (kill chain)

---

- The actions it describes are needed but they are not so well separated
- As far as concern Reconnaissance we can distinguish at least two moments where it is executed
  - Before the initial attack against the target system to discover how to implement the initial penetration
  - After the initial penetration to collect information on the various nodes and develop or acquire tools to attack the nodes
- This shows that the model can be used to understand the various phases of an intrusion but not to emulate the attacker behavior in full details

What the adversary does

How to defend

## RECONNAISSANCE *Identify the Targets*

### ADVERSARY

*The adversaries are in the planning phase of their operation. They conduct research to understand which targets will enable them to meet their objectives.*

- ▶ Harvest email addresses
- ▶ Identify employees on social media networks
- ▶ Collect press releases, contract awards, conference attendee lists
- ▶ Discover internet-facing servers

### DEFENDER

*Detecting reconnaissance as it happens can be very difficult, but when defenders discover recon – even well after the fact – it can reveal the intent of the adversaries.*

- ▶ Collect website visitor logs for alerting and historical searching.
- ▶ Collaborate with web administrators to utilize their existing browser analytics.
- ▶ Build detections for browsing behaviors unique to reconnaissance.
- ▶ Prioritize defenses around particular technologies or people based on recon activity.

1



## WEAPONIZATION *Prepare the Operation*

---

### ADVERSARY

*The adversaries are in the preparation and staging phase of their operation. Malware generation is likely not done by hand – they use automated tools. A “weaponizer” couples malware and exploit into a deliverable payload.*

- ▶ Obtain a weaponizer, either in-house or obtain through public or private channels
- ▶ For file-based exploits, select “decoy” document to present to the victim.
- ▶ Select backdoor implant and appropriate command and control infrastructure for operation
- ▶ Designate a specific “mission id” and embed in the malware
- ▶ Compile the backdoor and weaponize the payload

### DEFENDER

*This is an essential phase for defenders to understand. Though they cannot detect weaponization as it happens, they can infer by analyzing malware artifacts. Detections against weaponizer artifacts are often the most durable & resilient defenses.*

- ▶ Conduct full malware analysis – not just what payload it drops, but how it was made.
- ▶ Build detections for weaponizers – find new campaigns and new payloads only because they reused a weaponizer toolkit.
- ▶ Analyze timeline of when malware was created relative to when it was used. Old malware is “malware off the shelf” but new malware might mean active, tailored operations.
- ▶ Collect files and metadata for future analysis.
- ▶ Determine which weaponizer artifacts are common to which APT campaigns. Are they widely shared or closely held?

2



## DELIVERY *Launch the Operation*

---

### ADVERSARY

*The adversaries convey the malware to the target. They have launched their operation.*

- ▶ Adversary controlled delivery:
  - ▶ Direct against web servers
- ▶ Adversary released delivery:
  - ▶ Malicious email
  - ▶ Malware on USB stick
  - ▶ Social media interactions
  - ▶ “Watering hole” compromised websites

### DEFENDER

*This is the first and most important opportunity for defenders to block the operation. A key measure of effectiveness is the fraction of intrusion attempts that are blocked at delivery stage.*

- ▶ Analyze delivery medium – understand upstream infrastructure.
- ▶ Understand targeted servers and people, their roles and responsibilities, what information is available.
- ▶ Infer intent of adversary based on targeting.
- ▶ Leverage weaponizer artifacts to detect new malicious payloads at the point of Delivery.
- ▶ Analyze time of day of when operation began.
- ▶ Collect email and web logs for forensic reconstruction. Even if an intrusion is detected late, defenders must be able to determine when and how delivery began.

3



## EXPLOITATION *Gain Access to Victim*

---

### ADVERSARY

*The adversaries must exploit a vulnerability to gain access. The phrase “zero day” refers to the exploit code used in just this step.*

- ▶ Software, hardware, or human vulnerability
- ▶ Acquire or develop zero day exploit
- ▶ Adversary triggered exploits for server-based vulnerabilities
- ▶ Victim triggered exploits
  - ▶ Opening attachment of malicious email
  - ▶ Clicking malicious link

### DEFENDER

*Here traditional hardening measures add resiliency, but custom capabilities are necessary to stop zero-day exploits at this stage.*

- ▶ User awareness training and email testing for employees.
- ▶ Secure coding training for web developers.
- ▶ Regular vulnerability scanning and penetration testing.
- ▶ Endpoint hardening measures:
  - ▶ Restrict admin privileges
  - ▶ Use Microsoft EMET
  - ▶ Custom endpoint rules to block shellcode execution
- ▶ Endpoint process auditing to forensically determine origin of exploit.

Not clear where and how the required information is collected

4





# INSTALLATION *Establish Beachhead at the Victim*

---

## ADVERSARY

*Typically, the adversaries install a persistent backdoor or implant in the victim environment to maintain access for an extended period of time.*

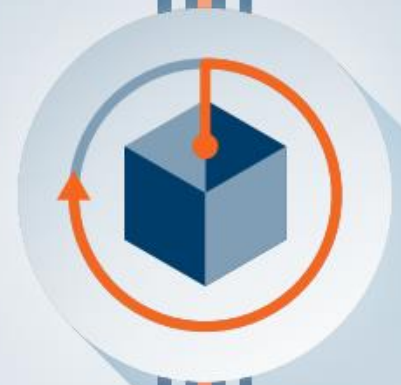
- ▶ Install webshell on web server
- ▶ Install backdoor/implant on client victim
- ▶ Create point of persistence by adding services, AutoRun keys, etc.
- ▶ Some adversaries “time stomp” the file to make malware appear it is part of the standard operating system install.

## DEFENDER

*Endpoint instrumentation to detect and log installation activity. Analyze installation phase during malware analysis to create new endpoint mitigations.*

- ▶ HIPS to alert or block on common installation paths, e.g. RECYCLER.
- ▶ Understand if malware requires administrator privileges or only user.
- ▶ Endpoint process auditing to discover abnormal file creations.
- ▶ Extract certificates of any signed executables.
- ▶ Understand compile time of malware to determine if it is old or new.

5



## COMMAND & CONTROL (C2) *Remotely Control the Implants*

---

### ADVERSARY

*Malware opens a command channel to enable the adversary to remotely manipulate the victim.*

- ▶ Open two way communications channel to C2 infrastructure
- ▶ Most common C2 channels are over web, DNS, and email protocols
- ▶ C2 infrastructure may be adversary owned or another victim network itself

### DEFENDER

*The defender's last best chance to block the operation: by blocking the C2 channel. If adversaries can't issue commands, defenders can prevent impact.*

- ▶ Discover C2 infrastructure thorough malware analysis.
- ▶ Harden network:
  - ▶ Consolidate number of internet points of presence
  - ▶ Require proxies for all types of traffic (HTTP, DNS)
- ▶ Customize blocks of C2 protocols on web proxies.
- ▶ Proxy category blocks, including "none" or "uncategorized" domains.
- ▶ DNS sink holing and name server poisoning.
- ▶ Conduct open source research to discover new adversary C2 infrastructure.

6



## COMMAND & CONTROL (C2) *Remotely Control the Implants*

---

### ADVERSARY

*Malware opens a command channel to enable the adversary to remotely manipulate the victim.*

- ▶ Open two way communications channel to C2 infrastructure
- ▶ Most common C2 channels are over web, DNS, and email protocols
- ▶ C2 infrastructure may be adversary owned or another victim network itself

### DEFENDER

*The defender's last best chance to block the operation: by blocking the C2 channel. If adversaries can't issue commands, defenders can prevent impact.*

- ▶ Discover C2 infrastructure thorough malware analysis.
- ▶ Harden network:
  - ▶ Consolidate number of internet points of presence
  - ▶ Require proxies for all types of traffic (HTTP, DNS)
- ▶ Customize blocks of C2 protocols on web proxies.
- ▶ Proxy category blocks, including "none" or "uncategorized" domains.
- ▶ DNS sink holing and name server poisoning.
- ▶ Conduct open source research to discover new adversary C2 infrastructure.

6



## ACTIONS ON OBJECTIVES *Achieve the Mission's Goal*

---

### ADVERSARY

*With hands-on keyboard access, intruders accomplish the mission's goal. What happens next depends on who is on the keyboard.*

- ▶ Collect user credentials
- ▶ Privilege escalation
- ▶ Internal reconnaissance
- ▶ Lateral movement through environment
- ▶ Collect and exfiltrate data
- ▶ Destroy systems
- ▶ Overwrite or corrupt data
- ▶ Surreptitiously modify data

### DEFENDER

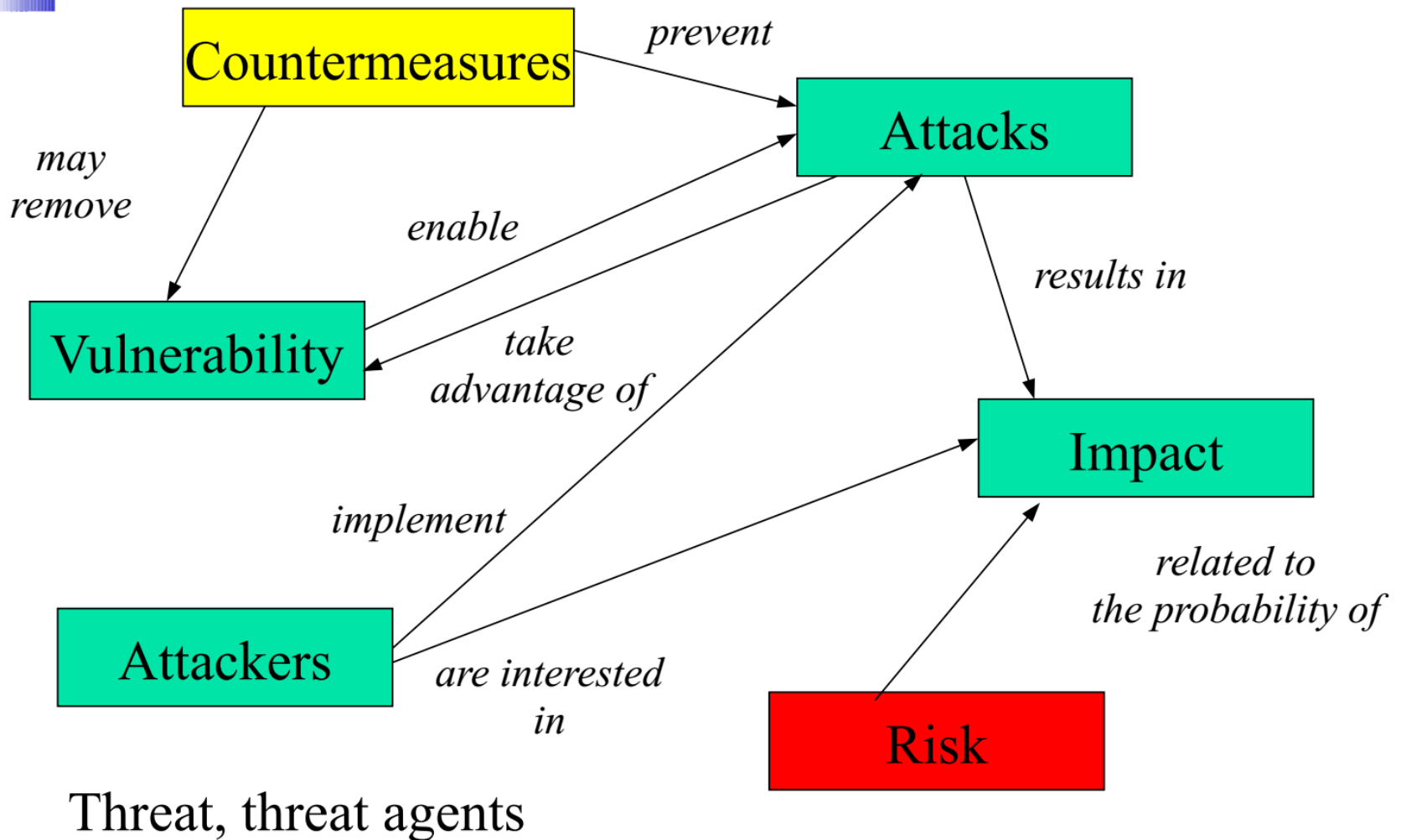
*The longer an adversary has CKC7 access, the greater the impact. Defenders must detect this stage as quickly as possible by using forensic evidence – including network packet captures, for damage assessment.*

- ▶ Establish incident response playbook, including executive engagement and communications plan.
- ▶ Detect data exfiltration, lateral movement, unauthorized credential usage.
- ▶ Immediate analyst response to all CKC7 alerts.
- ▶ Forensic agents pre-deployed to endpoints for rapid triage.
- ▶ Network package capture to recreate activity.
- ▶ Conduct damage assessment with subject matter experts.

7



# Terminology and relations ...





# Partial points view on sec– I

---

- Security = Confidentiality  $\Leftrightarrow$  Cryptography
- A set of algorithms to hide information so that only those who know another information (the key) can access it
- A fundamental but partial property because it cannot guarantee availability
- Crypto is a powerful tool to simplify not to solve problems
- *If you think cryptography by itself solve your problem either you do not understand cryptography or you do not understand your problem*



# Partial points of view – II

---

- Several security problems are related to the triple  $\langle \text{user, resources, rights=operations on the res} \rangle$  that determines who /how a resource is manipulated
- Several security mechanisms are related to the solution of these problems
  - Identifying the user
  - Identifying the resource
  - Discover the user rights on the resources
- Sophisticated identification system (biometrics etc.) can solve 1 but neither of the other ones
- You cannot change your fingerprint ...



# Partial point of view - III

---

- Safety  $\neq$  Security as it considers random threat agents
- In a system with  $10^n - 1$  safe states and 1 unsafe state where the threat agents work randomly,
  - the probability of an unsafe behavior =  $1/10^n$
  - system safety increases with  $n$
- If a system has one not secure state out of  $10^n$ , an intelligent threat agent tries to force the system to enter that state
- Security depends upon the agent success probability rather than on the overall number of states
- Security adversaries are intelligent, adaptive and not random





# Safety vs Security

---

- 3/5 modular redundancy is a standard strategy to increase safety that introduces 3/5 instances of each module
  - Any input is broadcasted to each module
  - The modules work in parallel
  - Vote on the output and selection of the output with the largest number of votes
- If a module is affected by a vulnerability then the attacker has 2/4 more opportunities to be successful



# Safety vs Security

---

- To make thing worst in the IOT you cannot have safety without security or a lack of security results in a lack of safety
- If terrorist controls a smart semaphore, the traffic can become rather unsafe and result in several security problem
- A robot that is not secure can kill workers and so on



# Partial point of view - IV

---

- Red team exercises aka penetration test
- You pay someone for attacking your system
  - If the attack fails, you assume your system is ok
  - If the attack is successful you improve it
- Inconsistent approach because you cannot be sure that
  - Your improvement is effective (Braess paradox)
  - The red team has find all the possible attack
  - A red team failure has a large number of reasons ..



# Some examples

---

- Vulnerability
- Attack
- Some countermeasures

We describe a stack overflow, a popular attack that is an instance of buffer overrun



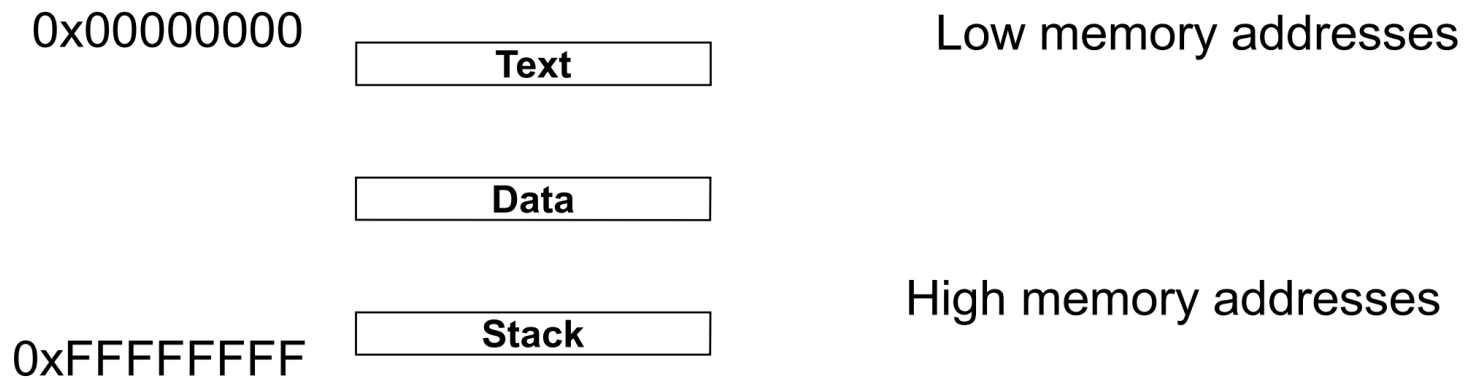
# Buffer overflow

---

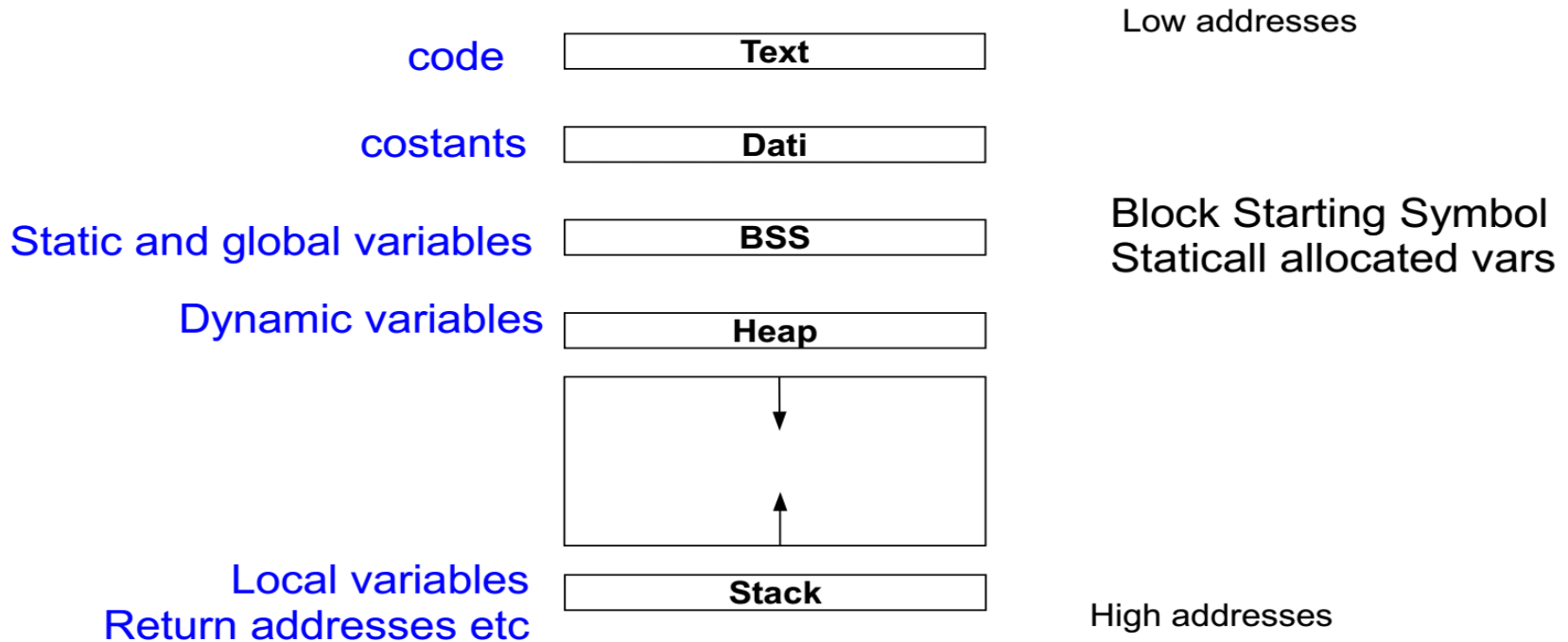
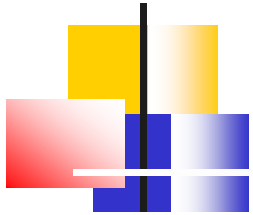
- The most common problem due to language controlled memory management
- It does not arise in high level languages where the programmer is not involved in memory management or with strong data types
- The most important security issue in the last 10 years (currently replaced by web vulnerabilities and phishing + malware)
- A **forced write of some data with a size larger than expected**. If the type system does not detect this inconsistency, some data is replaced in memory.
- It inserts some program (code injection) into a system that can, among other execute some shell command. If the program is executed at root level, then the attacker fully control any system function.
- A buffer overflow can exploit any of the following areas **stack, heap e bss (block started by symbol) static variables that are allocated by the compiler.**

# A process memory

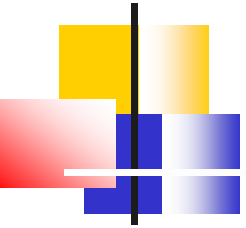
- Fundamental to understand buffer overflow, is the structure of a process memory that is partitioned into three segments: *text*, *data* and *stack*.
- The *text segment* is fixed, stores the program code and it is read only. Any write attempts results in a segmentation error (segmentation fault – core dump)
- The *data segment* stores the process static and dynamic variables
- The *stack segment* stores the data to manage function calls and returns



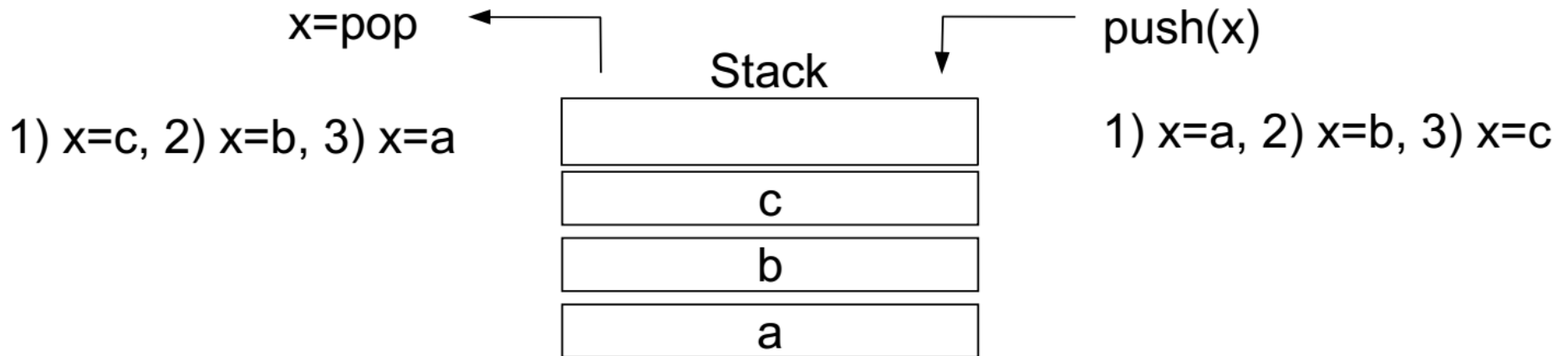
# A process memory



# Stack



- A Lifo (Last In First Out) dynamic data structure
- It is used to manage function calls and returns (call assembly instruction).
- The stack memory area is logically partitioned into records (stack or activation frame) one for each call







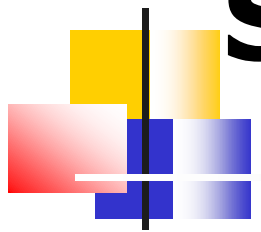
# Stack and system registers

---

- The memory address of the instruction to be executed is stored in the **EIP** (Extended Instruction Pointer) register
- **EBP** (Extended Base Pointer) points to the beginning of a *stack frame* while **ESP** (Extended Stack Pointer) points to the end of the stack frame
- When a function is called, the system pushes onto the stack
  - the return address = **EIP+4**,
  - the base address of the current frame = **EBP**

then it copies **ESP** into **EBP** to initialize the new *stack frame*.

# Stack and system registers



Grow in this direction

activation  
record

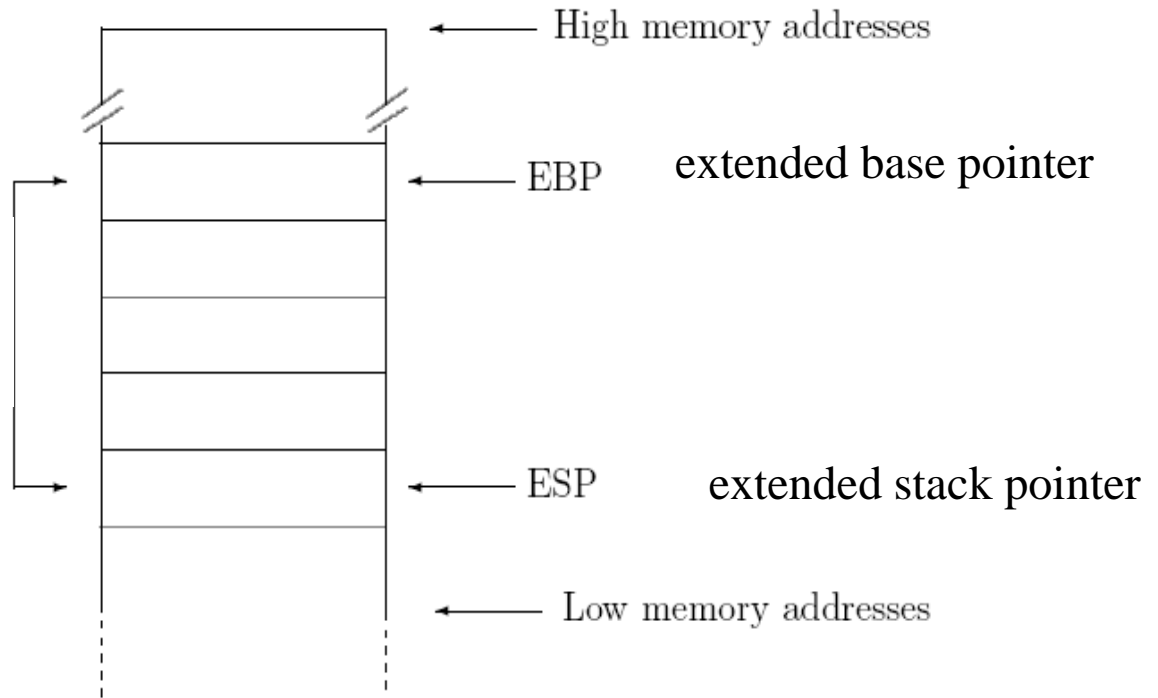
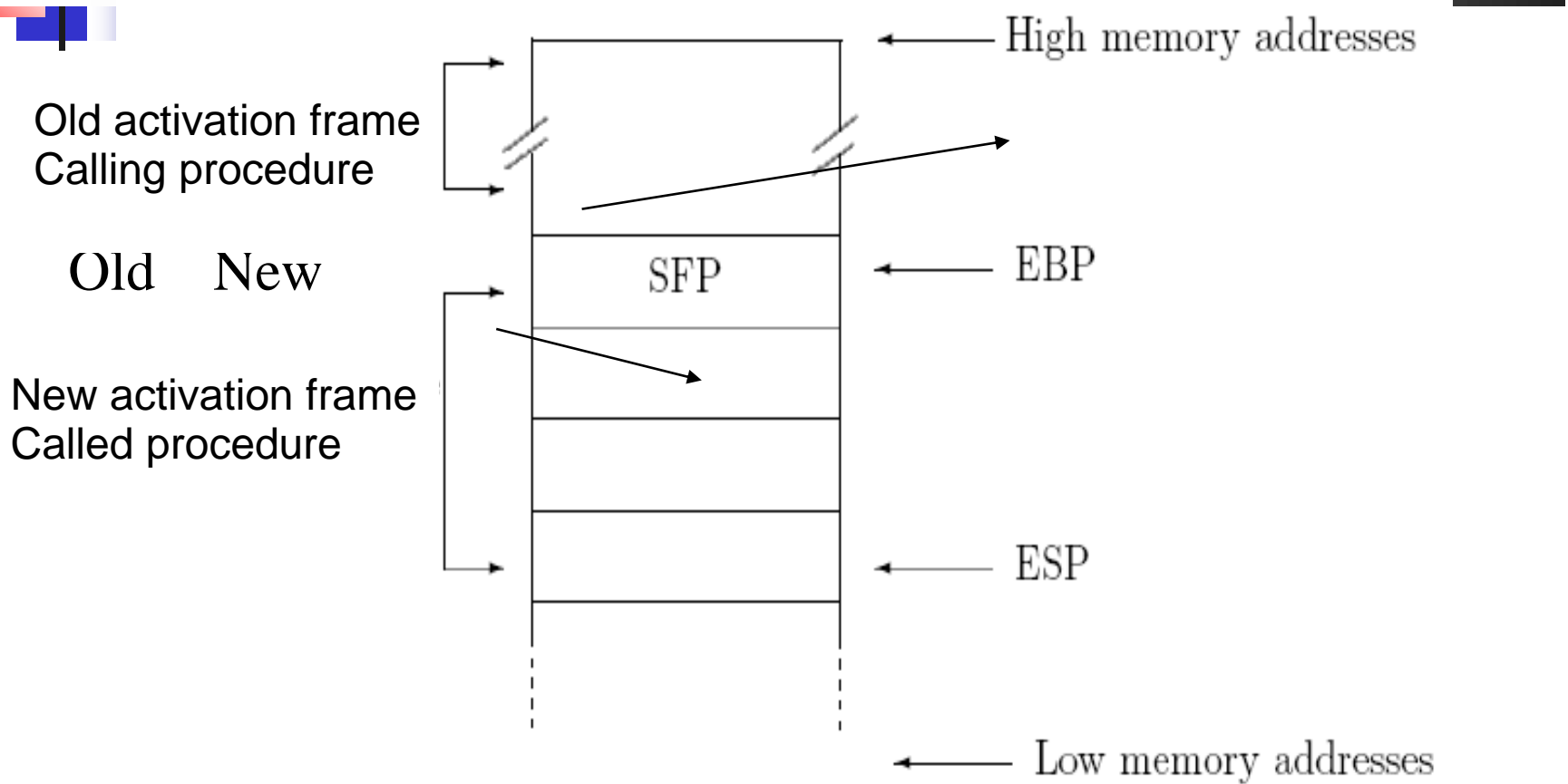
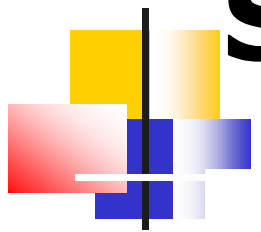


Figura 4: Stack pointer e frame pointer

# Stack and system registers

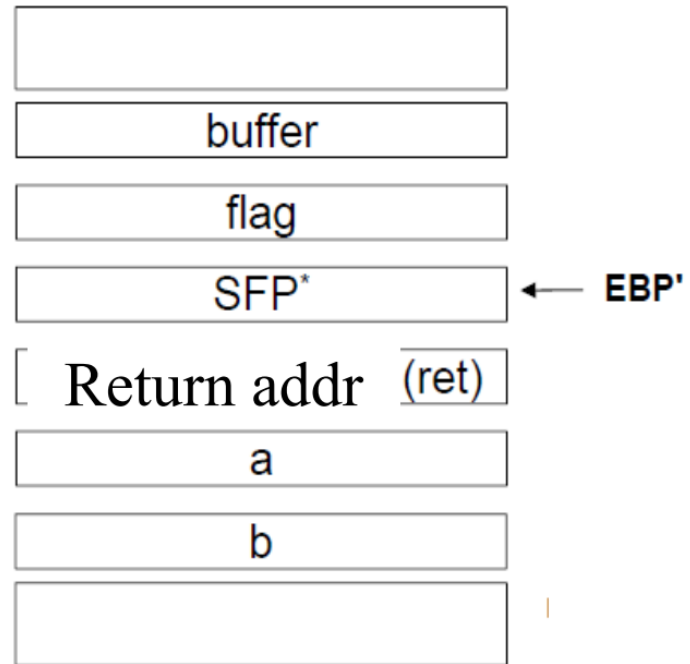


# C: an example

This is a simple example to see how all the stuff works

```
void test_function (int a, int b)
{
  char flag;
  char buffer[10];
}
```

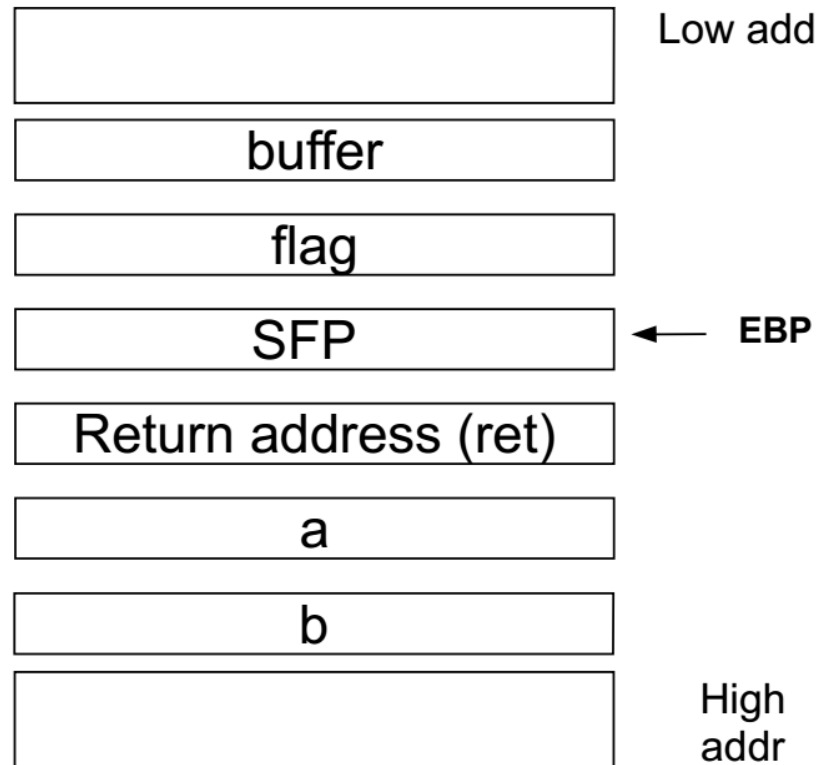
```
int main()
{
  test_function (1,2);
  exit(0);
}
Return address = EIP + 4 byte
```



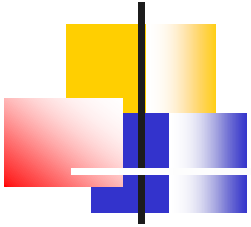
SFP = saved frame pointer = it is used to restore the original value of EBP on a return

# The stack frame

- Local variable of *test\_function* are addressed by subtracting a displacement from **EBP** while function parameters by a positive displacement
- This is independent of the value of the stack pointer that may change
- When a function is called EIP points to the function code.
- The stack stores both local variables and parameters of a function. When the function ends, the whole stack frame is removed before returning (**ret**).

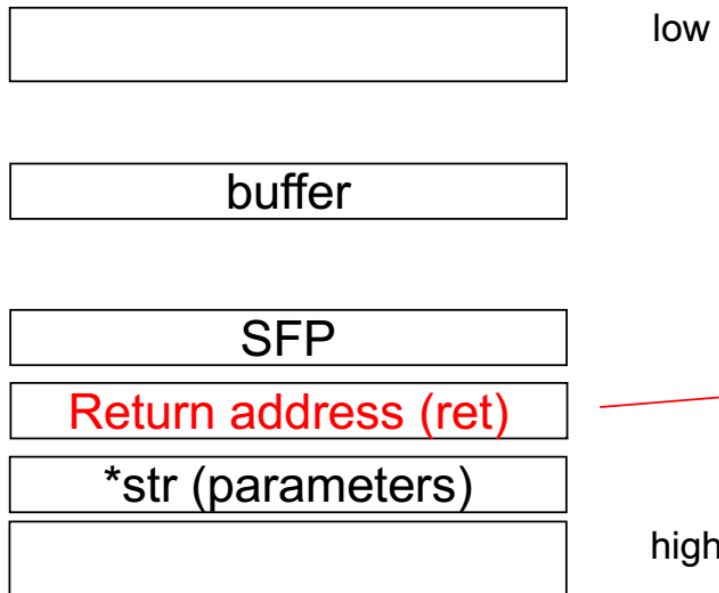


# Segmentation fault

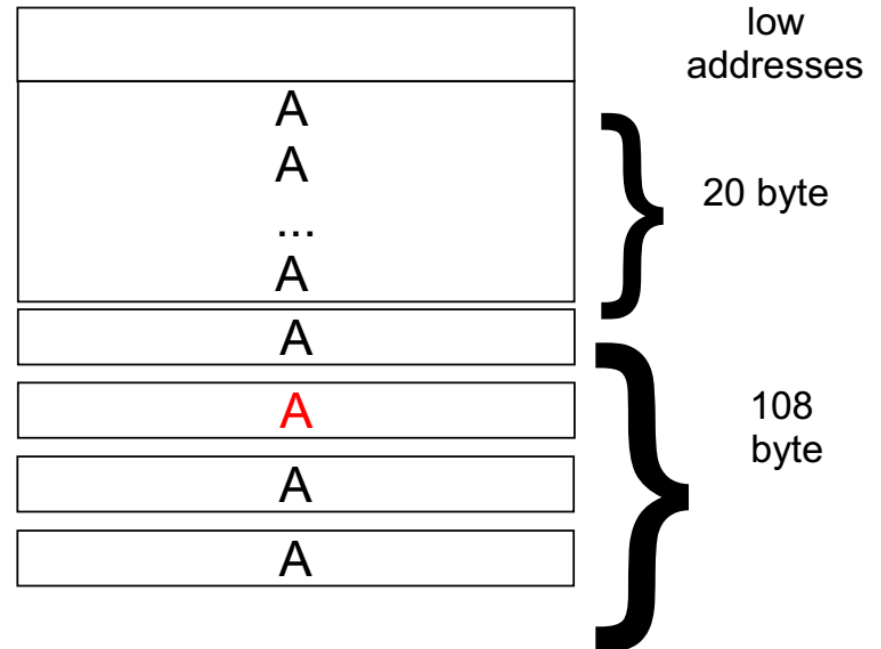


The previous code results in a segmentation fault

1) The first call to *overflow\_function* correctly initializes the stack frame



2) When *overflow\_function* ends, the return address has been overwritten by the character **A** (segmentation fault!)





# Buffer (stack) overflow

---

What happens if the return address (`ret`) stores a valid memory address?

- No exception is signalled and the process continues by executing the instruction pointed by `ret`.
- A *stack based buffer overflow* exploits this by replacing `ret` with a pointer to some code injected by the attacker maybe into the stack itself
- How can we manipulate the return address and inject some code into the system?



# A Buffer Overrun

---

- It occurs when some variable is larger than expected and it overwrite other variables
- It may be implemented if the language lacks a typing system
- Very popular among computer worms
- Four kinds:
  - Stack based buffer overrun
  - Heap based buffer overrun
  - V-table and function pointer overrun
  - Exception handler overrun





# Stack Overflow

---

- By copying a string into the stack we destroy (update ??)
  - The return address
  - Other values on the stack
- The values that are copied codify a program
- The new return address points to the program we have copied onto the stack
- Overall result: an administrative shell
- This is possible only if the target procedure runs as root

A local fully automated attack, remote if present in a network service



# Stack overflow

---

Vulnerability = alternative perspectives

1. Lack of control on the size of program variables
2. Bad type system
3. Incorrect memory operation
4. Growth direction of the stack
- 5....



# Overflow: countermeasures

---

- Strong typing
- Controls on string lengths
- Insert a “canary” into the stack
- Not executable memory
- Data Execution Prevention (DEP)
- Ad hoc checks in the compiler
- ASLR: address space layout randomization



# Canary

---

- A value that differs at each invocation
- Inserted into the stack before any parameter so that any overflow that overwrites the return address also overwrites the canary
- Before returning we check that the canary has not been updated
- Randomly chosen at each invocation so that the attacker cannot know its value



# Not executable stack

---

- Controls when fetching an instructions, they can be supported by the MMU
- No data structure can store instructions
- NX bit (the last one) introduced in AMD processors
- It does not work with Linux that stores some drivers in the stack to manage i/o devices



# Data Execution Prevention (DEP)

---

- A system-level memory protection feature introduced into the operating system starting with Windows XP and Windows Server 2003.
- DEP enables the system to mark one or more pages of memory as non-executable.
- This means that code cannot be run from that region of memory, which makes it harder for the exploitation of buffer overruns.

# Address Space Layout

## Randomization ASLR

---

- The starting point of the various segment is selected randomly
- The attacker cannot know in advance the starting address of data structures of interest
- The first step of the attack has to compute the starting address
- Attack more complex and slower



# ASLR – entropy

---

Type	Description	Protection	Granularity of Rebasing
Free	Free space	Inaccessible	Not rebased
Code	Executable or DLL code	Read-only	15 bits
Static data	Within executable or DLL	Read-Write	15 bits
Stack	Process and thread stacks	Read-Write	29 bits
Heap	Main and other heaps	Read-Write	20 bits
TEB	Thread Environment Block	Read-Write	19 bits
PEB	Process Environment Block	Read-Write	19 bits
Parameters	Command-line and Environment variables	Read-Write	19 bits
VAD	Returned by virtual memory allocation routines	Read-Write	15 bits
VAD	Shared Info for kernel and user mode	Unwritable	Not rebased





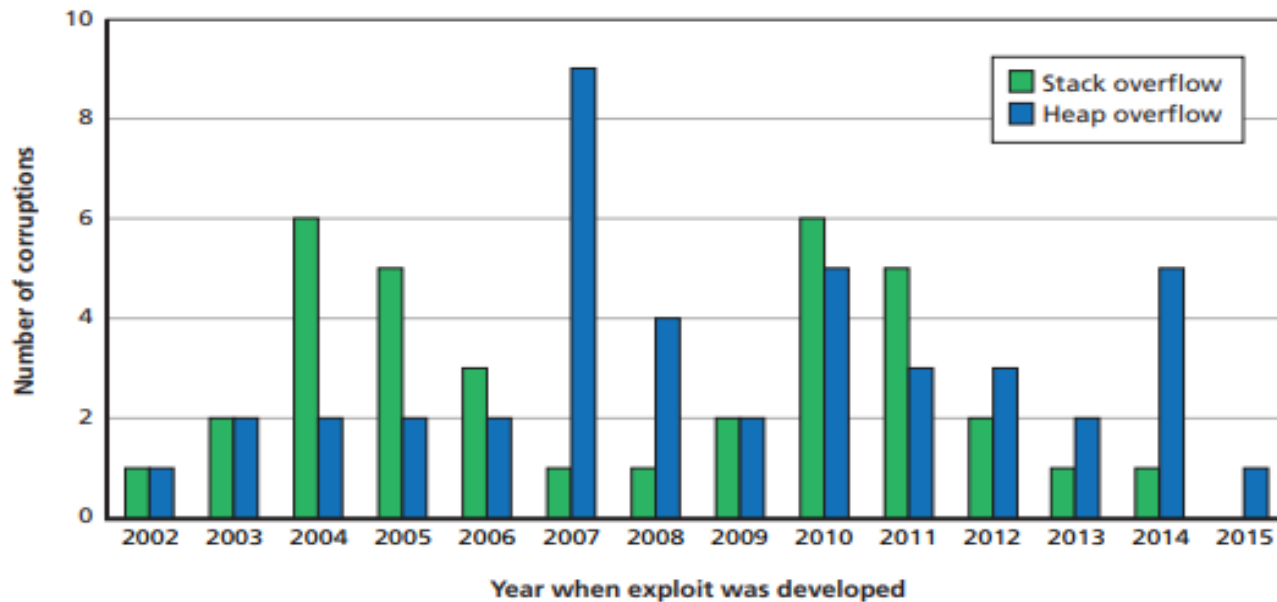
# Cost of the countermeasures

---

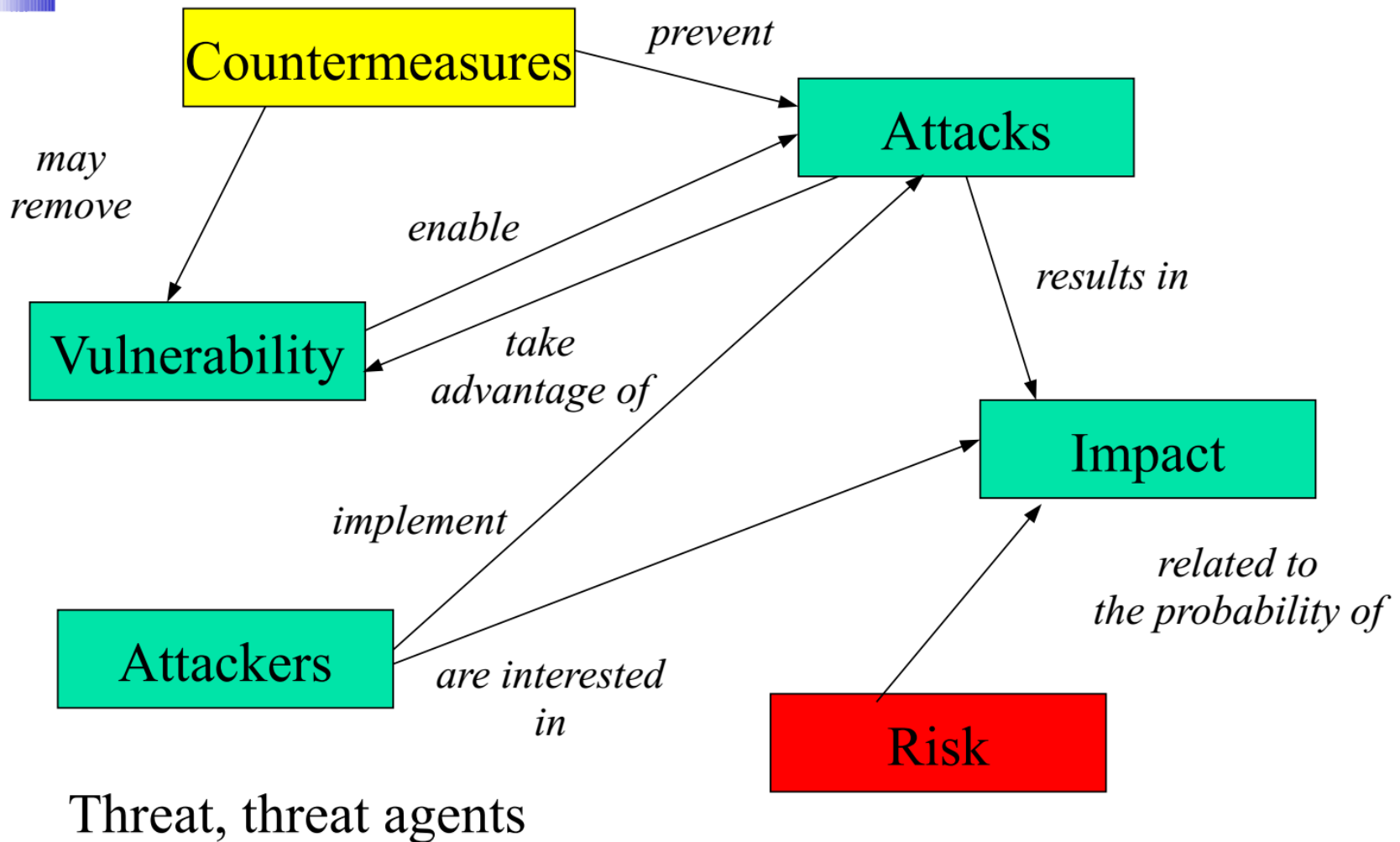
- Each countermeasure has a distinct cost
  - Strong typing = 10-30% run time overhead
  - Checks on string length = large cost but lower than the previous one
  - Canary = specialized control, low cost
  - ASLR supported by MMU translation low cost
  - Not executable stack = lowest cost because it exploits an hardware/firmware support

# Stack vs heap

Figure C.1  
Type of Memory Corruption, Counts by Year (n = 101)



# Terminology and relations ...





# Structural vulnerability TCP/IP

---

- When the TCP/IP stack has been defined, the main goal was resilience against physical attack against the network (attack = bombing)
- Main goal = availability
  - Some mechanism defined to discover which nodes are alive and reachable
  - No mechanism is available to guarantee \ (authenticate) the source of a message



# Structural vuln: an Example

---

1. A node can send an ECHO message to check whether another node is alive and reachable, The receiver replies by returning the same message.
2. The sender can specify a **partial IP address** to broadcast a message **to check a set of nodes**
3. There is no control on the fields of an IP packet a node sends



## All together now ..

---

1. R is a network with 1000 nodes, X is a partial IP address matching the addresses of all nodes of R
2. A sends a ECHO message to the address X but it specifies the address of B as the packet sender address
3. Any node in R replies to B
4. B cannot interact with other nodes because its communication lines are overflowed by the ECHO messages

***Distributed Denial of Service***



# All together now +IOT

---

OVH France-based hosting provider, was the victim of a wide-scale DDoS attack carried via network of over **152,000 IoT devices**.

According to OVH the DDoS attack reached nearly **1 Tbps at its peak**. Of those IoT devices participating in the DDoS attack, **they were primarily comprised of CCTV cameras and DVRs**.

Many of these types devices' network settings are improperly configured, which leaves them ripe for the picking for hackers that would love to use them to carry out destructive attacks.

**OVH originally stated that 145,607 devices made up the botnet, but recently confirmed that another 6,857 cameras joined in on the attack**. The DDoS peaked at 990 Gbps on September 20th thanks to two concurrent attacks, and according to OVH, the original botnet was capable of a **1.5 Tbps DDoS attack** if each IP topped out at 30 Mbps.



# Security as an holistic property

---

- The security of a system is not implied by (cannot be deduced from ) the one of each of its modules
- The overall system may be unsecure even when each module is secure
- In a virtual machine hierarchy the security of a machine may be destroyed by a vulnerability in an underlying machine





# Impact and countermeasures

---

- The DDOS impact
  - depends upon the numbers of nodes, zombies, whose address matches that in the message
  - may be amplified by further messages
- Very few effective countermeasures because B is aware of the attack **only** when it starts receiving messages (discover the reconnaissance phase is impossible for B )
- Global hygiene of the Internet environment
- This is a **structural vulnerability**, it depends not upon the building blocks but upon the block composition



# Design approaches vs vulns

---

When designing and building a system we may adopt one of two approaches

- a) pretend there are no vulnerabilities in the components (penetrate and patch)
- b) be aware that there are vulnerabilities and try to anticipate them even if we still do not know which vulnerabilities (proactive or predictive approach)



# Penetrate and patch

---

- Vulnerabilities have not been anticipated
- Since we have assumed there are no vulnerabilities, we should remove (patch) a vulnerability (more in general deploy a countermeasure) as soon as a vulnerability is discovered.
- There is a competition between
  - discovering and exploiting vulnerabilities
  - patching the system to remove them



# Security Patch (wikipedia)

---

- A security patch is a change applied to an asset (OS, application, ...) to correct the weakness described by a vulnerability.
- This corrective action will prevent successful exploitation and remove or mitigate a threat's capability to exploit the vulnerability to attack an asset.
- Security patches are the primary method of fixing software vulnerabilities. Currently Microsoft releases its security patches once a month, and other operating systems and software projects have security teams dedicated to release reliable software patches as soon after a vulnerability announcement as possible.
- Security patches are closely tied to responsible disclosure

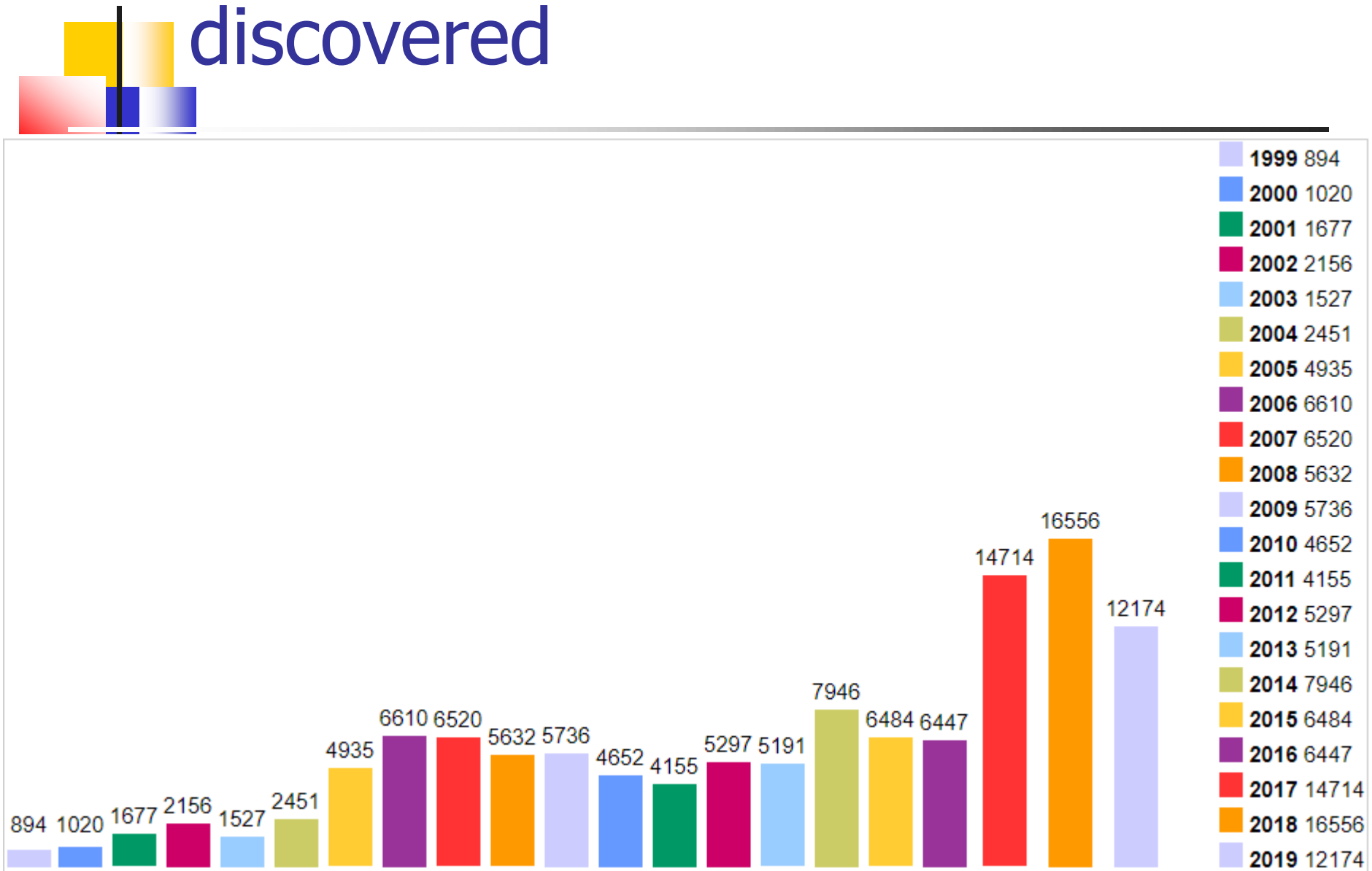


# Patches: problem

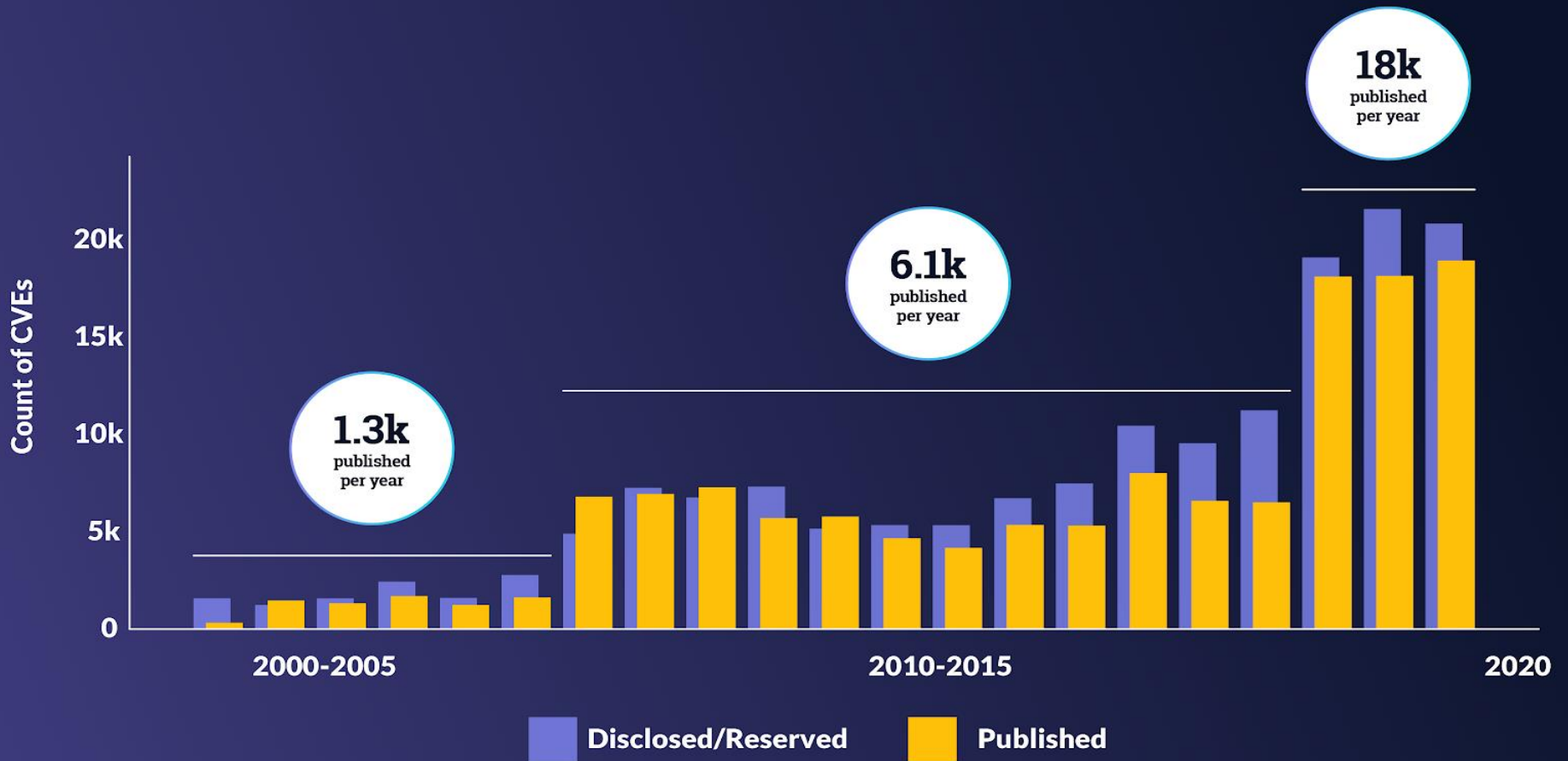
---

- Any patching updates a software component and changes its behaviour
- The change may influence the users
- A patch can be applied only after checking that the changes can be accepted
- Some system cannot be patched because they are certified and a patch invalidates the certification (power production, gas and power distribution)

# Number of vulnerabilities discovered

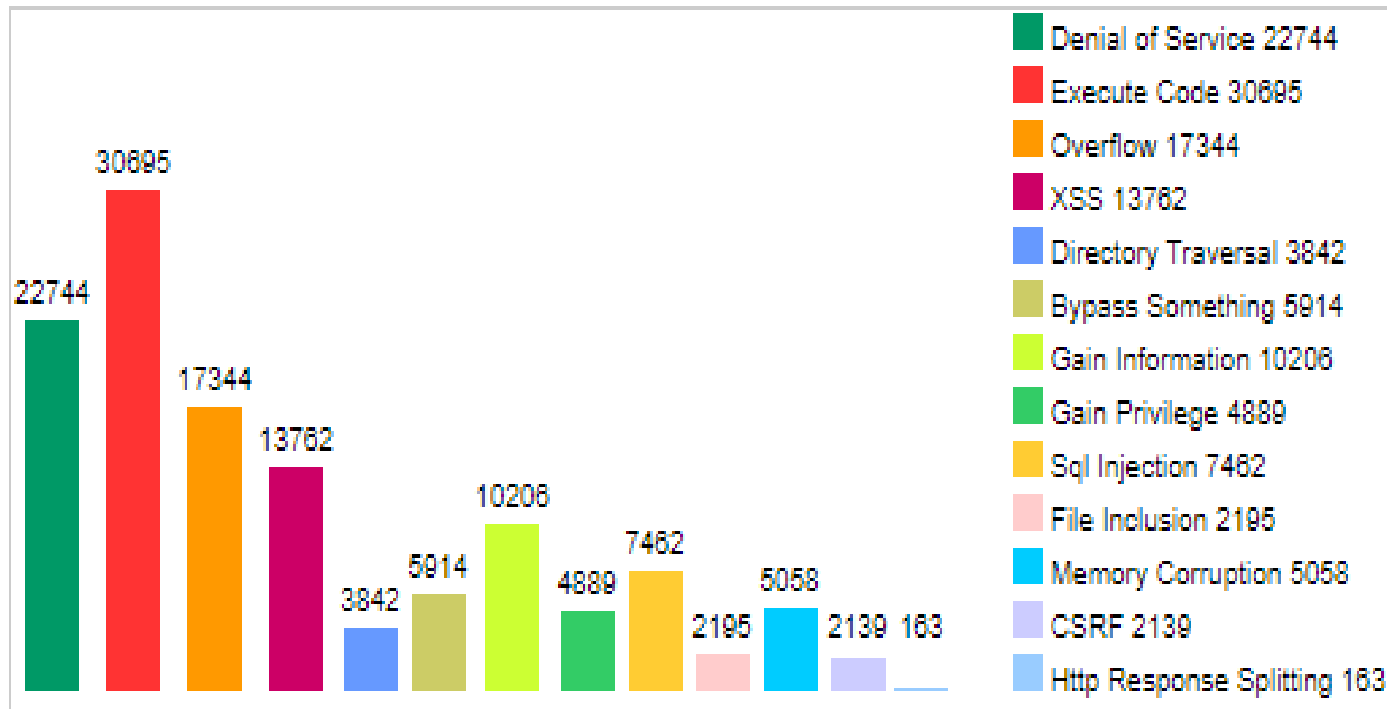


# Number of vulnerabilities discovered



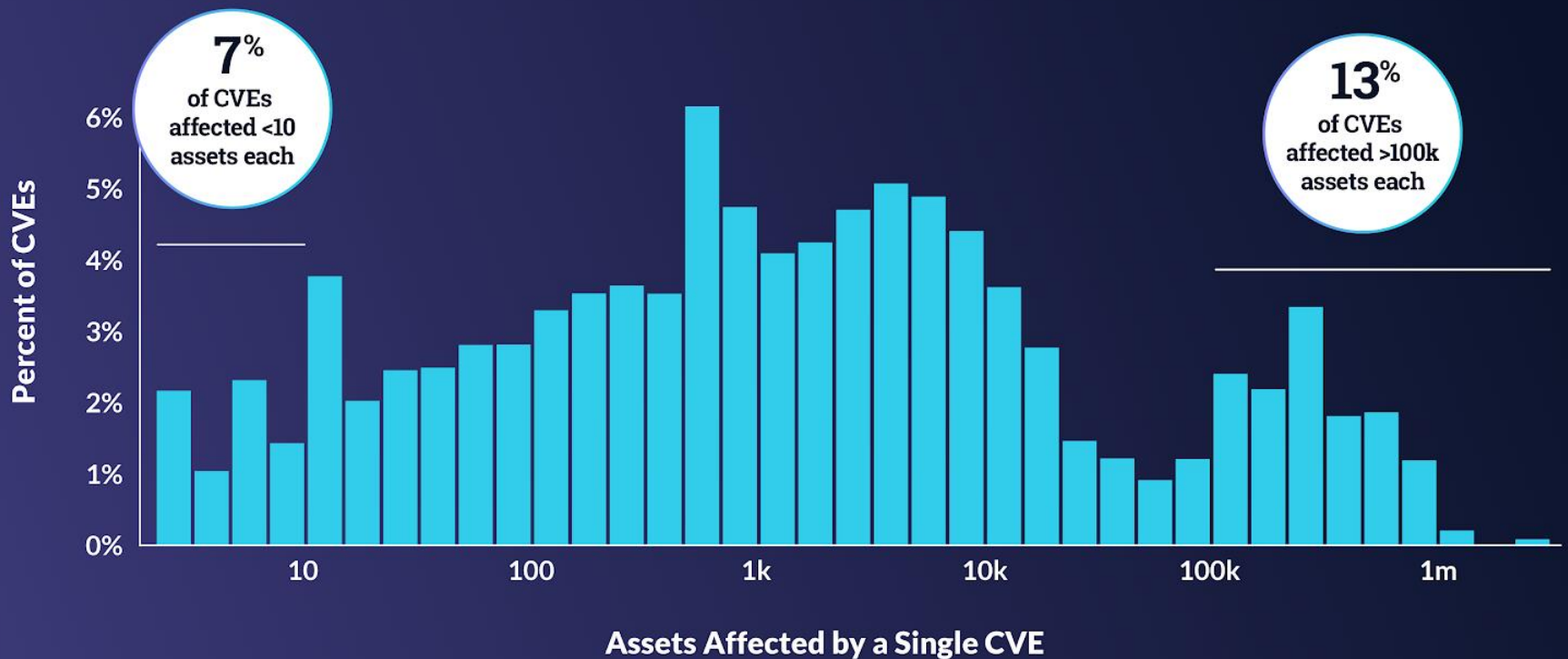
# Number of vulnerabilities discovered

Vulnerabilities By Type

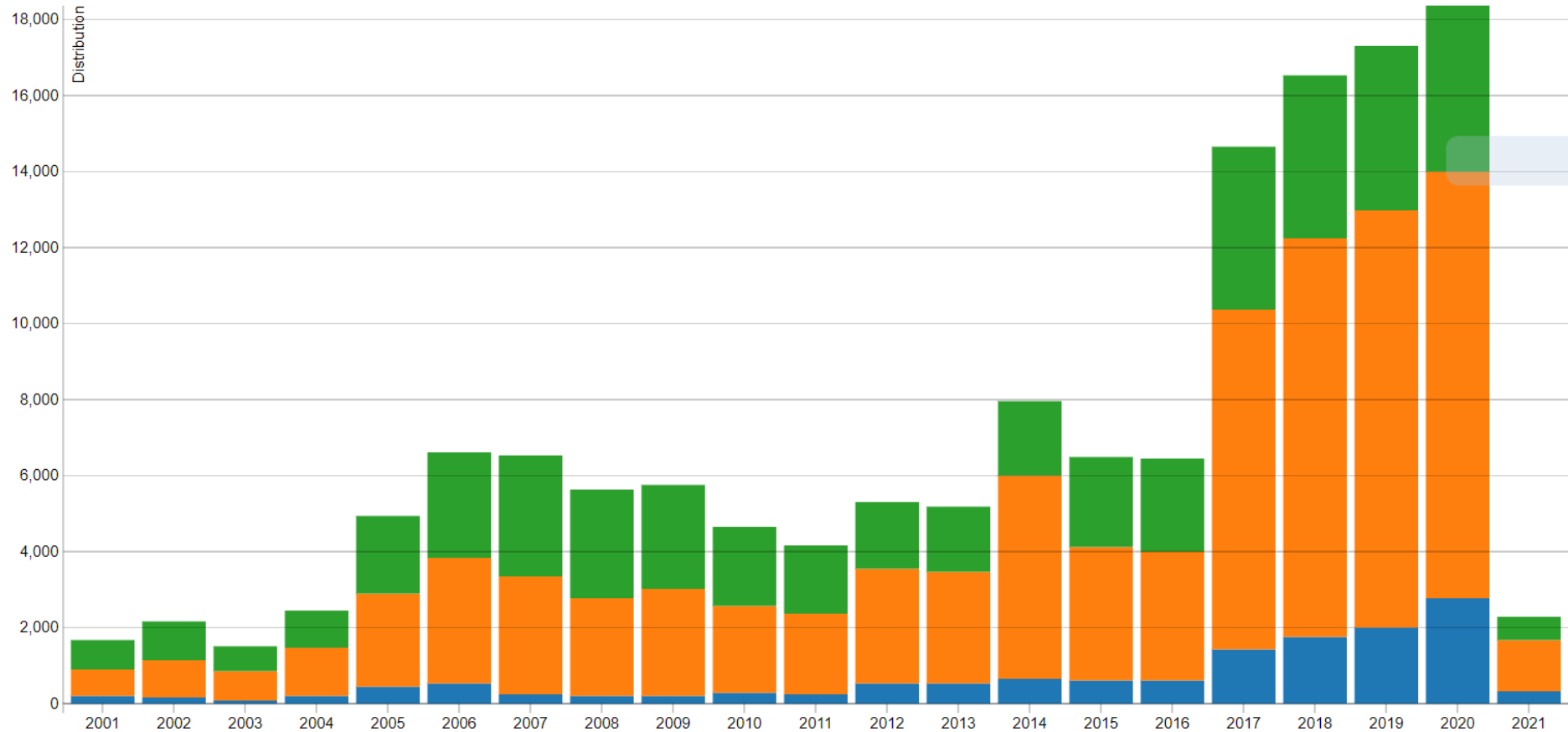
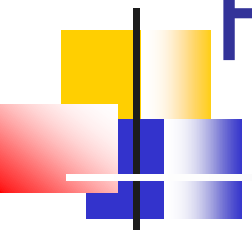




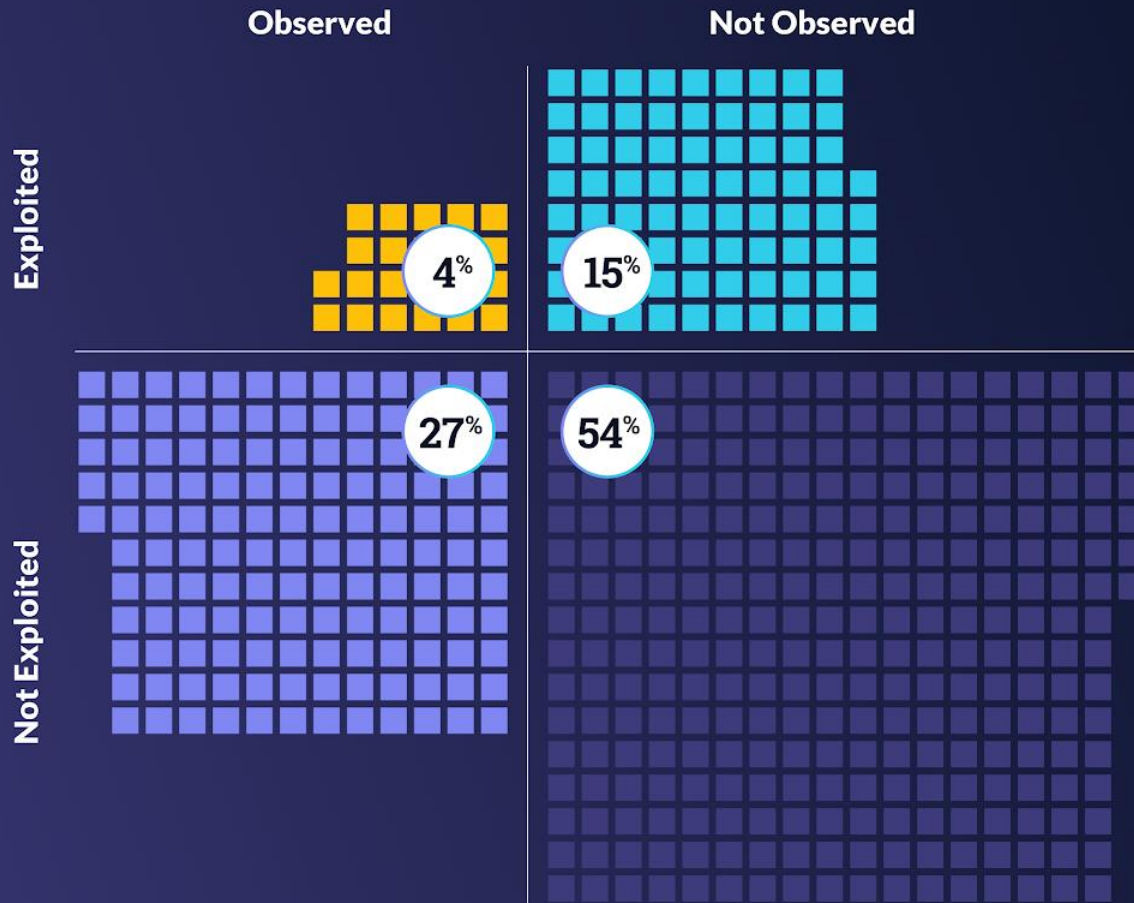
# Number of vulnerabilities discovered



# How dangerous (not risk)



# How dangerous (not risk)





# Top 10 Vulnerabilities - Windows Systems

---

1. Internet Information Services
2. Microsoft SQL Server
3. Windows Authentication
4. Internet Explorer
5. Windows Remote Access Services
6. Data Access Components(MDAC)
7. Windows Scripting Host
8. Outlook and Outlook Express
9. Peer to Peer File Sharing
10. Simple Network Management



# Top 10 Vulnerabilities - Unix Systems

---

1. BIND Domain Name System
2. Remote Procedure Calls (RPC)
3. Apache Web Server
4. Accounts with No Passwords or Weak Passwords
5. Clear Text Services
6. Sendmail
7. Simple Network Management Protocol
8. Secure Shell (SSH)
9. Misconfiguration of NIS/NFS
10. Open Secure Sockets Layer (SSL)



# Other lists - I

---

## ■ Top Vulnerabilities in Windows Systems

- W1. Windows Services
- W2. Internet Explorer
- W3. Windows Libraries
- W4. Microsoft Office and Outlook Express
- W5. Windows Configuration Weaknesses

## Top Vulnerabilities in Cross-Platform Applications

- C1. Backup Software
- C2. Anti-virus Software
- C3. PHP-based Applications
- C4. Database Software
- C5. File Sharing Applications
- C6. DNS Software
- C7. Media Players
- C8. Instant Messaging Applications
- C9. Mozilla and Firefox Browsers
- C10. Other Cross-platform Applications



## Other lists - II

---

- Top Vulnerabilities in UNIX Systems
  - U1. UNIX Configuration Weaknesses
  - U2. Mac OS X
- Top Vulnerabilities in Networking Products
  - N1. Cisco IOS and non-IOS Products
  - N2. Juniper, CheckPoint and Symantec Products
  - N3. Cisco Devices Configuration Weaknesses



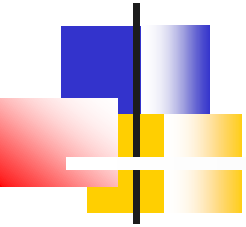
# Hippa vulnerabilities

---

- Firewall and System Probing
- Network File Systems (NFS) Application
- Electronic Mail Attacks
- Vendor Default Password Attacks
- Spoofing, Sniffing, Fragmentation and Splicing
- Social Engineering Attacks
- Easy-To-Guess Password
- Destructive Computer Viruses
- Prefix Scanning (Illegal Modem)
- Trojan Horses



# Life cycle of a vulnerability in a penetrate and patch world



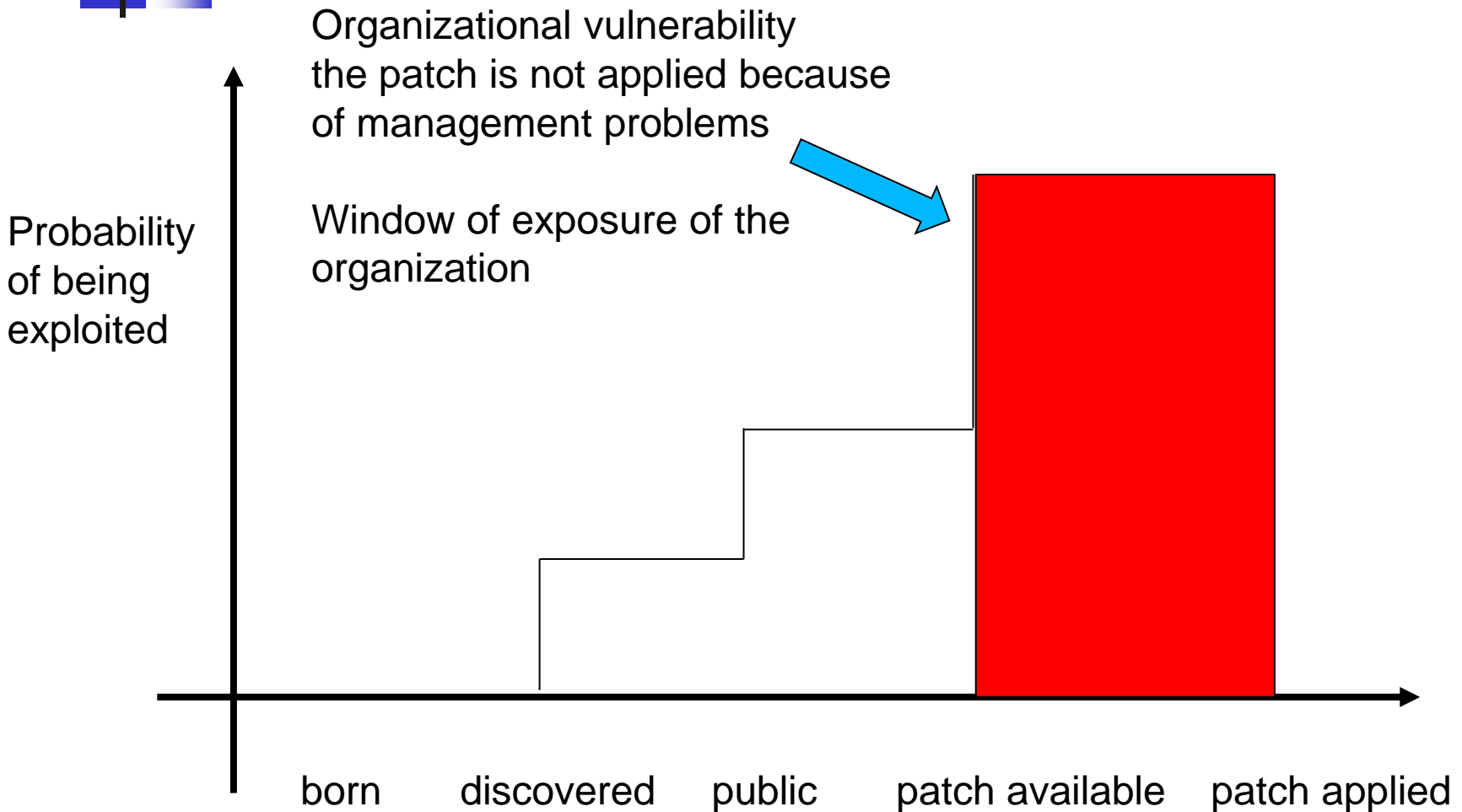
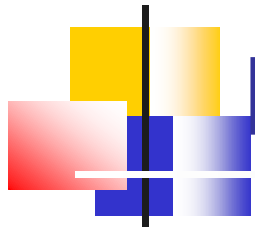


# State of a vulnerability - 1

---

1. The vulnerability is born
2. The vulnerability is discovered
3. Both the vulnerability and an exploit that takes advantage of the vulnerability are discovered
4. Both the vulnerability and a patch that removes the vulnerability are discovered (a race with 3)
5. The vulnerability, the exploit and the patch have been discovered

# Potential impact of a vulnerability

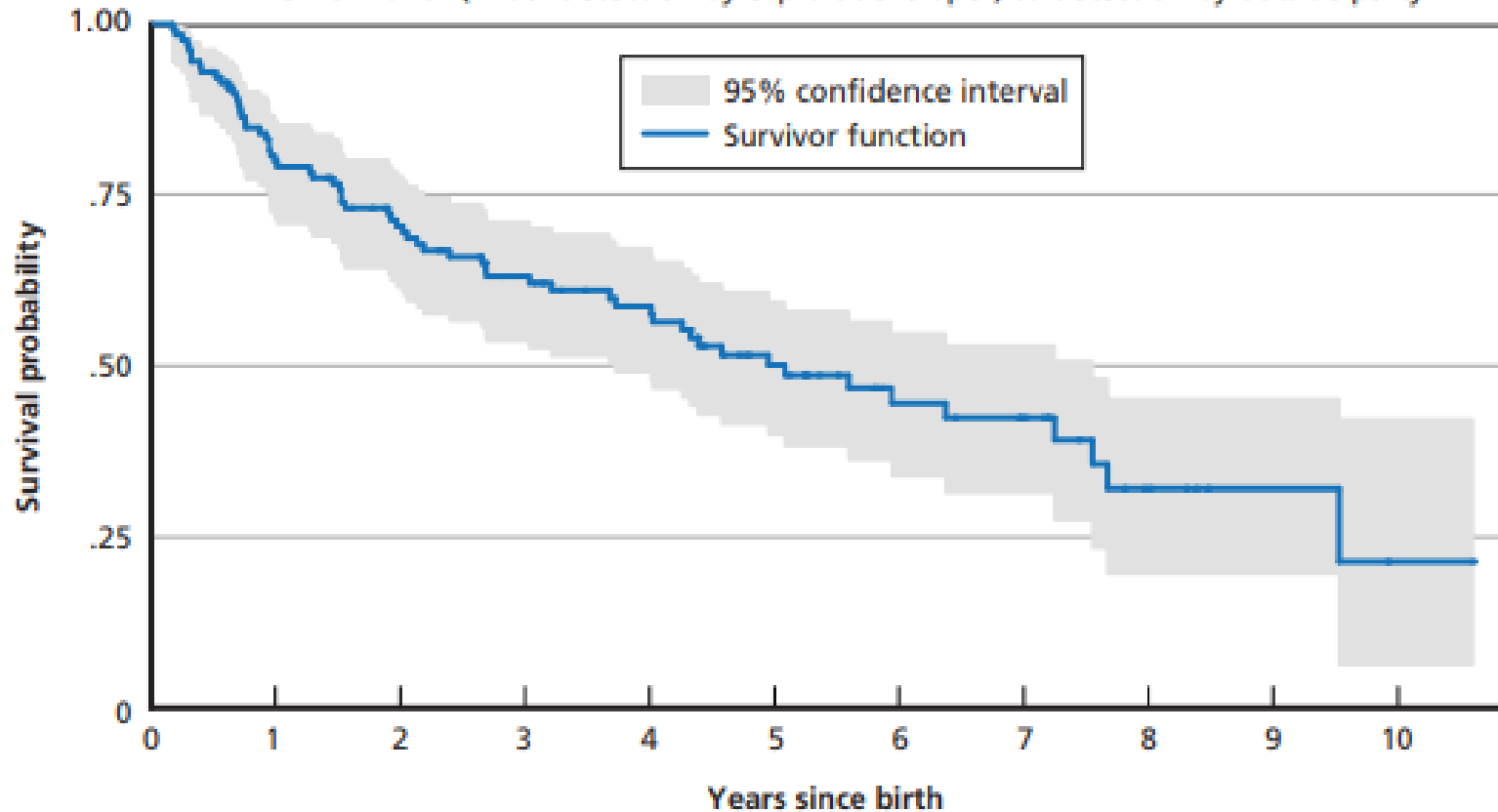


# Survival of a vulnerability

Kaplan-Meier Survival Probability Estimates (n = 127)

A vulnerability is not public after its discovery

Time from birth (initial detection by exploit developer) to detection by outside party

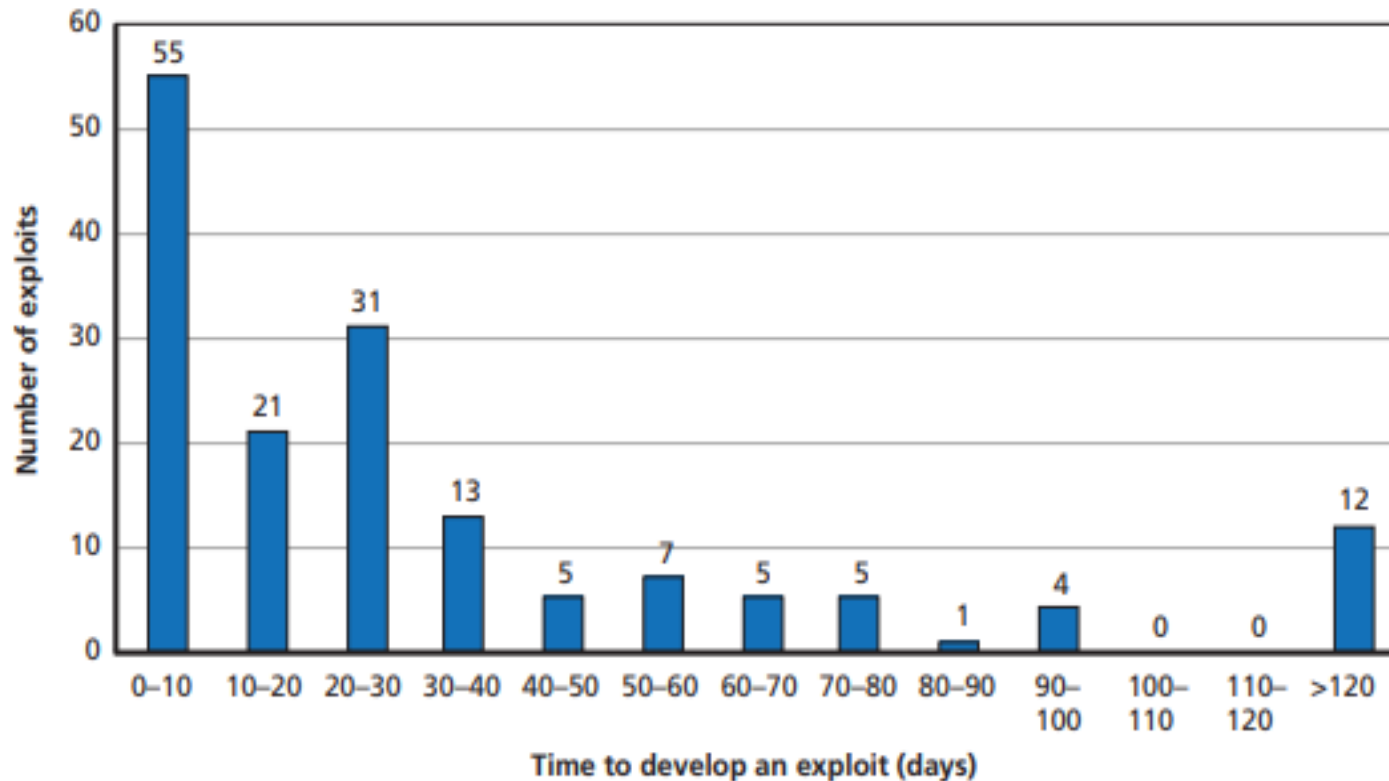




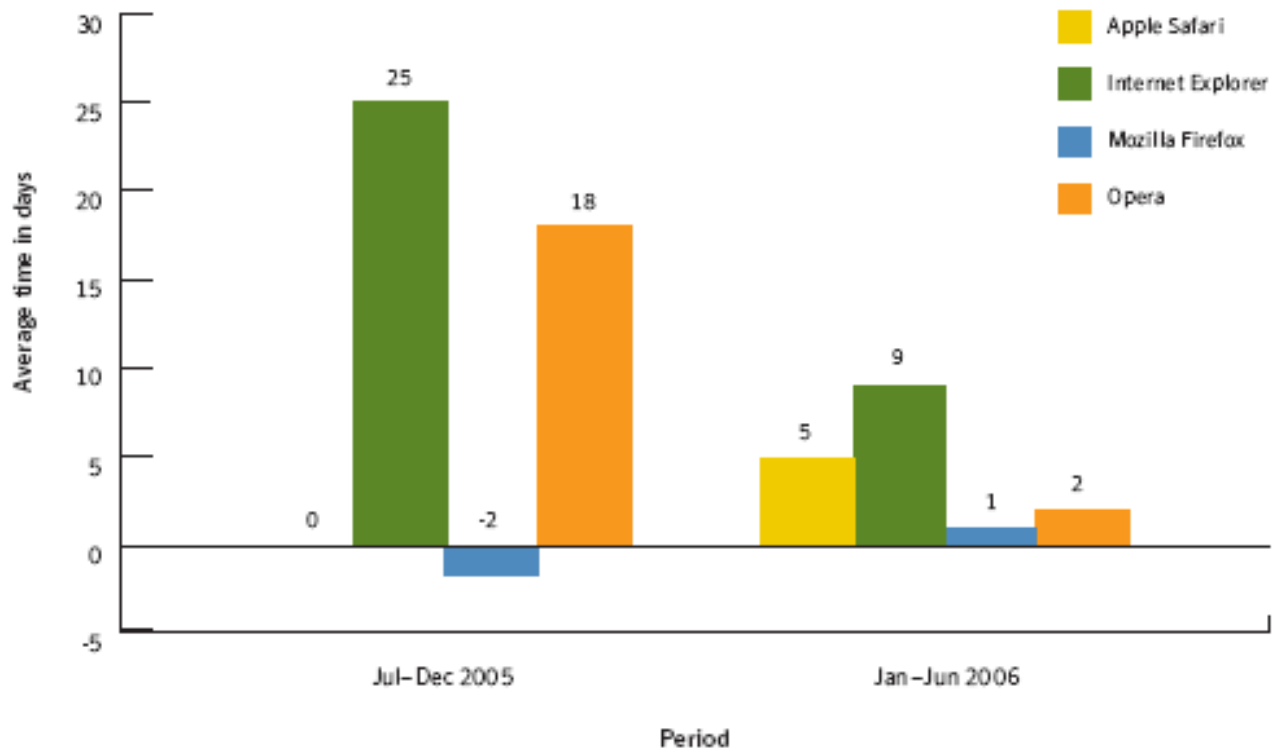
# Time for an exploit

---

Frequency Count of Time to Develop an Exploit (n = 159)



# Time to develop a patch



**Figure 4. Web browsers window of exposure**

*Source: Symantec Corporation*



# Zero day exploit

---

- An exploit for a vulnerability that has been discovered but not disclosed to all the users
- Sometimes those who discover a vulnerability sell it to those interested in attacking the system (black market of vulnerabilities)
- The only way to defeat the market is to design a system that resists attacks even when a vulnerability is discovered?



## State of a vulnerability - 2

---

- Sometimes a system is attacked even if vulnerability is in the last status ie a patch is available
- It is well known that some system owners do not apply a patch even if it is available (60% of attacks exploit a vulnerability that can be patched)
- Asymmetry between the owner and the software supplier (applying the patch is a responsibility of the owner rather than of the supplier)



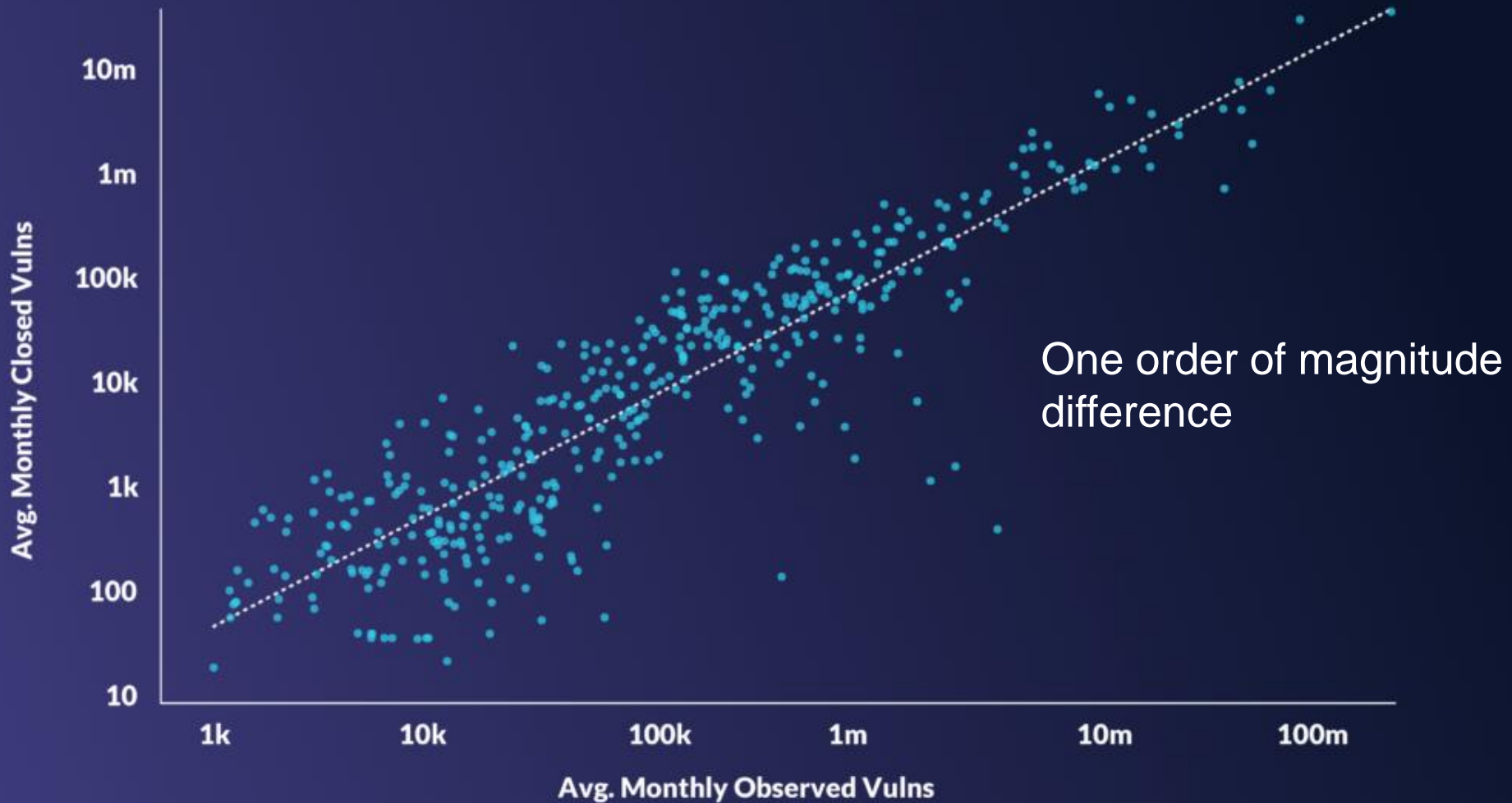
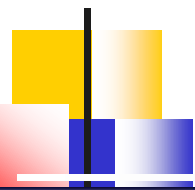


## Potential impact

---

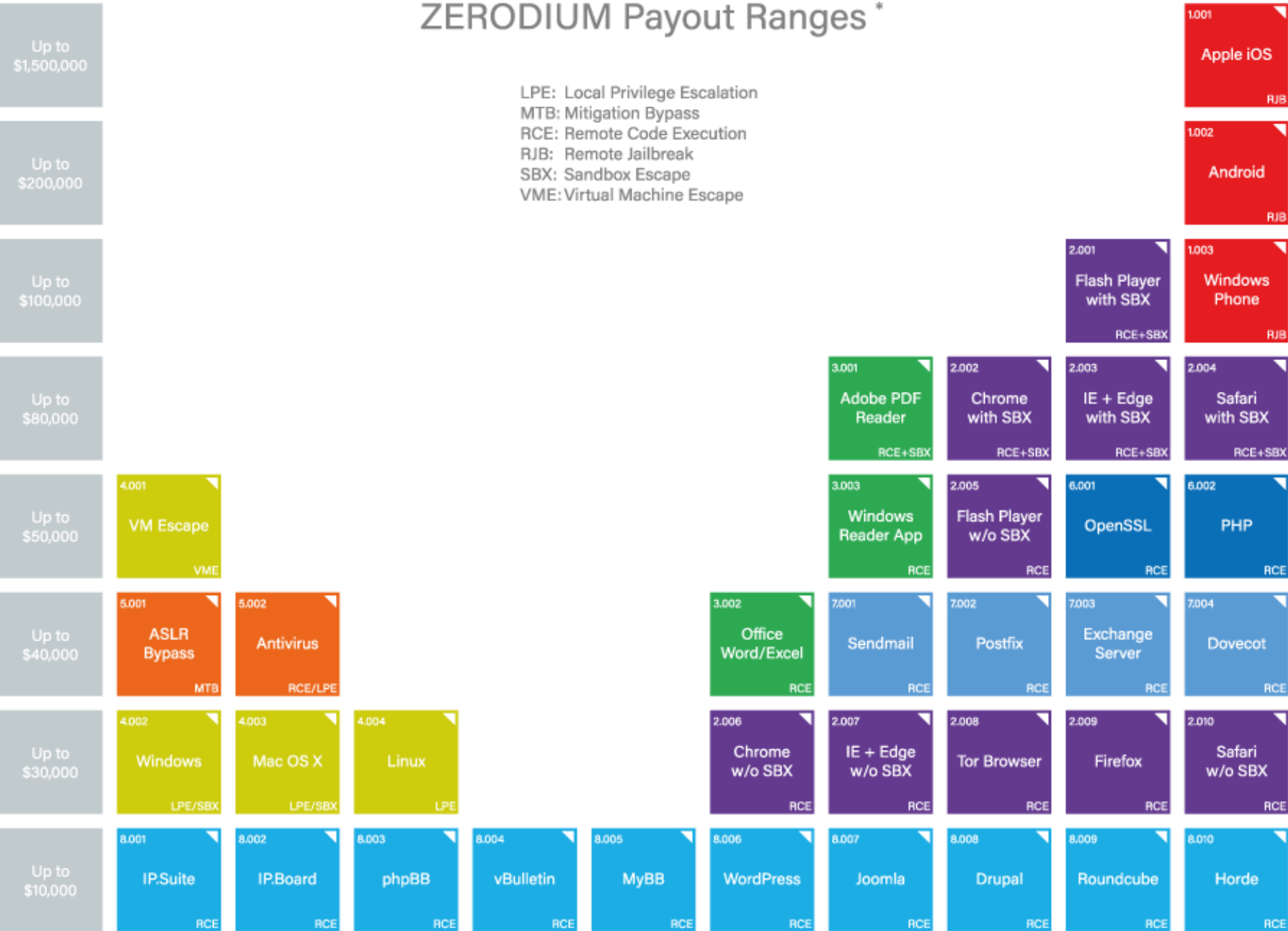
- In the best case, a patch is available before an attack is known
- If the owner does not apply the patch, then any benefit of discovering the patch before the attack is lost
- It is the application of the patch not its definition that reduces the danger

# Patch ability



# ZERODIUM Payout Ranges \*

LPE: Local Privilege Escalation  
 MTB: Mitigation Bypass  
 RCE: Remote Code Execution  
 RJB: Remote Jailbreak  
 SBX: Sandbox Escape  
 VME: Virtual Machine Escape



\* All payout amounts are chosen at the discretion of ZERODIUM and are subject to change or cancellation without notice.

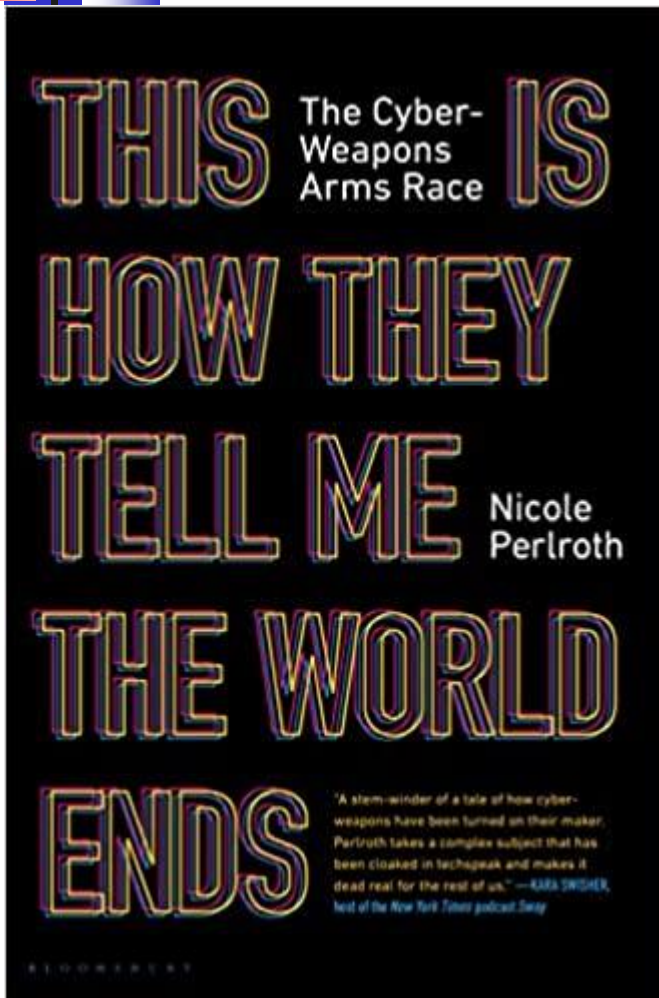


# Other buyer ...

---

ADOBE READER	\$5,000-\$30,000
MAC OSX	\$20,000-\$50,000
ANDROID	\$30,000-\$60,000
FLASH OR JAVA BROWSER PLUG-INS	\$40,000-\$100,000
MICROSOFT WORD	\$50,000-\$100,000
WINDOWS	\$60,000-\$120,000
FIREFOX OR SAFARI	\$60,000-\$150,000
CHROME OR INTERNET EXPLORER	\$80,000-\$200,000
IOS	\$100,000-\$250,000

# A book about zero day ...



Just published, 9 February

In a few words :  
a market for zero day  
is really a bad idea



# Number of vulnerability vs quality

---

- The number of vulnerabilities **discovered (= known)** in a module is always lower than **existing ones**
- This number depends upon
  - the availability of the source code
  - the number of applications and of people using the module
  - the expected benefit of an attack against the module
- If a module is scarcely used, very few vulnerabilities are known but this does not imply they do not exist
  - ⇒ The number of disclosed vulnerability cannot be used to evaluate the quality of the module code



# Genetic difference

---

- A system is more robust if it composes modules from distinct suppliers
- The joint existence of vulnerabilities and a monopoly in the module supply results in several problems because
  - all the instances of a module are affected by the same vulnerabilities
  - the same vendor tends to repeat a vulnerability
- How much configuration influences vulnerabilities (??!!)



# Defence in depth

---

- Any system component can be affected by a vulnerability
- A security expert
  - Does not need to know any vulnerability
  - Can design a system so that the discovery of a vulnerability in a component does not make the whole system useless
  - Layered defence or defence in depth = redundancies and diversities in the controls
- proactive approach vs penetrate and patch





# Adopted Approach - I

---

- A solution that tries to anticipate any vulnerability in any component has an huge cost
- Hence some vulnerabilities cannot be anticipated
- According to their potential impact we want to understand which vulnerabilities
  - should be accepted
  - should be anticipated
  - should be patched asap
- Problem: how to classify each vulnerability



# Adopted Approach - II

---

- A vulnerability classification (handling) depends upon the corresponding risk
- Risk
  - 1) Average impact if the vulnerability is successfully exploited
  - 2) Risk of a vulnerability =  $F(P_{\text{attsucc}}, \text{Imp})$
- $P_{\text{attsucc}}$  = probability of a successful attack
  - Probability an intrusion exploits a vulnerability
  - Probability the attack is successful
- $\text{Imp}$  = impact due to a successful attack



# Adopted Approach - III

---

$P_{\text{attsucc}}$  is a function of several parameters

- Do exist threat agents that
  - are interested in implementing the attack
  - have the know how and the resources to implement the attack
- Complexity of the implementation (automated ?)
- Are there other vulnerabilities that an intrusion can exploit to reach the same goal?
- Are these attacks more or less complex?
- Less than 10% of vulns are actually exploited



# Probability and impact

---

- Approx are requires because detailed evaluations of the probability an attack ( is attempted) and (is successful) both are extremely complex
  - No historical information available
  - Quick hardware/software evolution
  - Human factor
- Similar problems arise for the impact because of loss of new clients, damage to the reputation



# Probability and impact

---

- An interesting approach that is currently emerging is the one of adversary emulation
- Mimic in an automated way the action of an attacker against the system
- The ability of repeating the attack a large number of times can support a detailed evaluation of the system robustness and resilience



## Probability - II

---

- Sometimes both the success probability and the impact are approximated  
{low, medium, high} or  
{low, medium-low, medium ...}
- We also need a risk matrix to compute the risk given the approximated input values

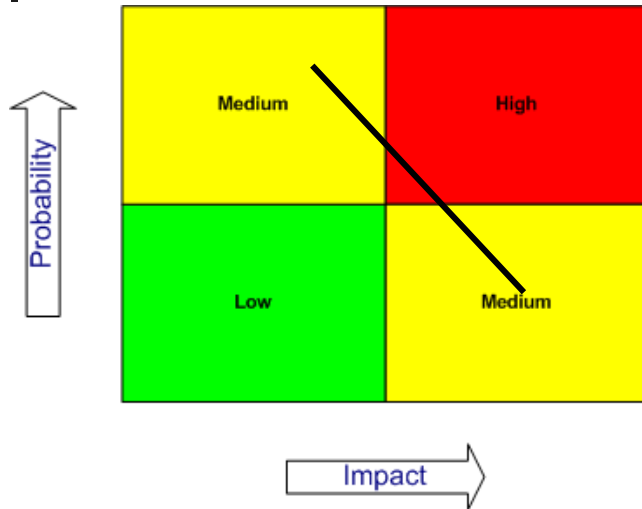


# Risk Matrix

---

Prob	VL	L	M	H	VH	
Impact	VH	H	H	VH	VH	
	H	M	H	H	H	
	M	L	L	M	M	
	L	L	L	M	M	
	VL	VL	L	M	H	VH

# Risk Matrix and Coherence



A risk matrix with more than one “colour” (level of risk priority) for its cells satisfies weak consistency with a quantitative risk interpretation if points in its top risk category (red) represent higher quantitative risks than points in its bottom category (green)

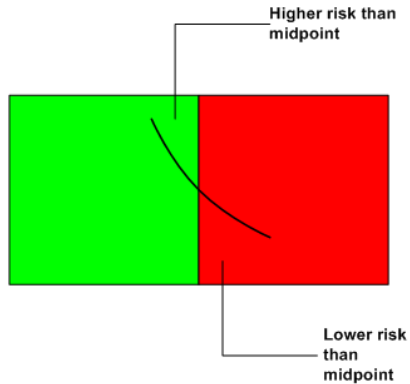
**Cox’s First Lemma:** If a risk matrix satisfies weak consistency, then no red cell (highest risk category) can share an edge with a green cell (lowest risk category).

**Cox’s Second Lemma:** if a risk matrix satisfies weak consistency and has at least two colours (green in lower left and red in upper right, if axes are oriented to depict increasing probability and impact), then no red cell can occur in the bottom row or left column of the matrix.



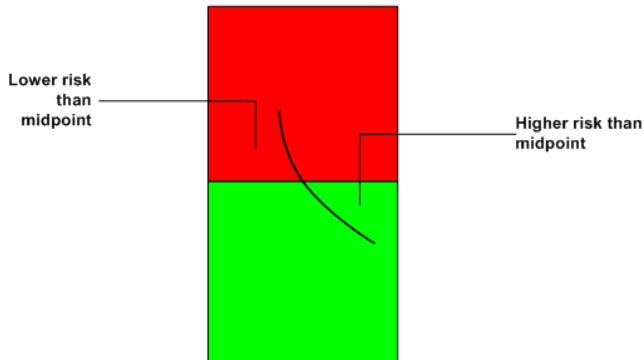
# Risk Matrix

---



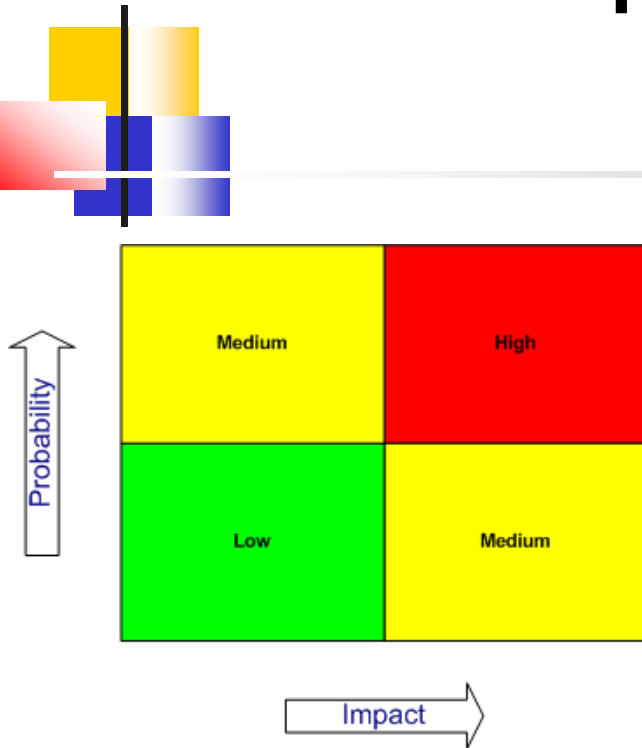
**Cox's First Lemma:** If a risk matrix satisfies weak consistency, then no red cell (highest risk category) can share an edge with a green cell (lowest risk category).

**Cox's Second Lemma:** if a risk matrix satisfies weak consistency and has at least two colours (green in lower left and red in upper right, if axes are oriented to depict increasing probability and impact), then no red cell can occur in the bottom row or left column of the matrix.



Betweeness A risk matrix satisfies this axiom of between-ness if every positively sloped line segment that lies in a green cell at its lower end and a red cell at its upper end must pass through at least one intermediate cell (i.e. one that is neither red nor green).

# Risk Matrix



In a risk matrix satisfying weak consistency, between-ness and consistent colouring:

- All cells in the leftmost column and in the bottom row are green.
- All cells in the second column from the left and the second row from the bottom are non-red.

A tricoloured  $3 \times 3$  or  $4 \times 4$  matrix that satisfies weak consistency, between-ness and consistent colouring can have only the following (single!) colour scheme:

- Leftmost column and bottom row coloured green.
- Top right cell (for  $3 \times 3$ ) or four top right cells (for  $4 \times 4$ ) coloured red.



# A critical problem

---

- In most cases probability is computed using some information about the past behavior of a system and of attackers
- From this information we can extrapolate the future behavior under a continuity assumption
- A breakthrough in the technology for the attacker or the owner can invalidate the continuity assumption and results in distinct probabilities (eg moving from meter to smart meter changes the vulnerabilities)



# Return on investment ROI

---

- The security analyst should be able to justify the cost of the countermeasures that are selected to be implemented (deployed)
- A countermeasure should be adopted only for those vulnerabilities that enable attacks that have both/at least one of
  - A large success probability
  - A large impact

= they have a large risk
- An interesting debate about both/at least one
- Black swan = big impact, low probability



# Return of investment

---

- It is the difference between
  - The overall risk before the countermeasures
  - The overall risk after the adoption of countermeasures
- The difference arises because of a lower success probability and/or a lower impact of an attack
- The case where a vulnerability is removed or patched ( $0$  = success probability) is a particular one



# Return of investment=Earning

---

- The difference between the potential impact and the cost of countermeasures
- The difference should be at most zero
- An alternative definition consider the ratio between the ROI and the cost of countermeasure
- The ratio should be larger than 1



# Summing Up

---

- A risk attitude is defined by two parameters
  - Penetrate and patch/Proactive (choose one)
  - Conditional/Unconditional (choose one)
- In penetrate and patch
  - each vulnerability may be critical, in proactive is critical if it has not been anticipated
  - the number of critical vulnerabilities (there is a risk) is much higher than in proactive
- If a vulnerability is critical
  - conditional sec = assess the risk and remove only if
    - there is a non zero risk= $f(\text{Probsucc}, \text{Impact})$
    - if it is cost effective ( $\text{risk} > \text{cost to remove}$ )



# Evaluating risk with no data

---

- Current research is focused on risk evaluation even if no data is available
- Solutions exist to produce accurate and realistic data to replace historical one that, in general, is not available or is not public
- Adversary simulation to understand how a system can be attacked and attack success probability





# Risk Based Approach

---

The approach we have described:

1. Asset analysis
2. Vulnerability Analysis
3. Attack Analysis
4. Threat Analysis
5. Impact Analysis
6. Risk Evaluation
7. Risk Management =

Risk  
Assessment

which countermeasures are to be adopted



# Risk Assess & Management

---

- The most modern approach to ICT security
- It consider the overall risk for an organization and it frames cyber risk with other risks
- A larger context has to be considered because ICT security should not be seen as a technological problem only



# Next steps

---

- Asset analysis
- Security policy
- Vulnerability Analysis
- Possible countermeasures
- Attack Analysis
- Risk Management = selection of the countermeasures to deploy



## Next Steps - II

---

- In principle, the security policy is a countermeasure
- In practice, it is defined independently of, and before, risk assessment because it defines the goals of an organization and the rules for its ICT resources
- Its satisfaction is an assessment goal
- Without a policy you do not know if you are secure