Security Part 2

O. Aktouf

Based on material from M. Cherfaoui (CISCO)

Security

Security is often associated with encryption (confidentiality) and username/password verification (authentication). We have seen these applications (as well as integrity) in a previous module.

Other areas where security has also a lot of press:

- Software viruses
- Breaches (think Target and Home Depot)
- Information disclosure (Snowdown, Wikileaks)
- Web defacing

Why Do Customers Worry about Security?

- When a company buys a product or subscribes to a Cloud service, it needs to make sure the product or the service cannot be compromised.
- A compromise of the product/service can lead to a direct financial or/and intellectual property (IP) loss. It can ultimately lead to a serious business impact.

Vendors Should also Worry about Security

- A compromised product (cloud or not) can expose a vendor to a serious liability.
- Customer confidence erodes and revenue can be lost as a result.

Security of Non-Cloud Software (1)

Security has always been a concern for companies buying software, even before Cloud computing became ubiquitous. The next slides will focus on on-prem software. Some steps companies take before buying an on-prem product:

- They make sure the vendor is reputable/viable.
- Look at the history of product security breaches
- Does the vendor have an internal process for quality and security? One such process is called SDL. We will talk about it later.
- Does product allow secure configuration (e.g. create users with different permissions)?
- Does the product comply with industry and/or government security standards?

Security of Non-Cloud Software (2)

- Some on-prem products are sold as a bundle of software, *middleware*[1], OS and hardware. Examples include routers, switches and telephony systems.
- When access to the OS is restricted, the product is an appliance.
- For this type of products, additional security requirements apply. For example:
  - The product uses an OS and middleware that up to date on security patches.
  - OS and middleware are *hardened*[2].
- OS and middleware are referred to as infrastructure.

1: middleware is the third party software stack used by an application: Apache, Tomcat, MySQLm RabbitMQ, etc.
2: hardening will be introduced in a future slide.

Security of Non-Cloud Software (3)

Once a company vets a product and decides to buy it, there are additional security steps the company needs to implement once the product is bought and deployed internally:

- Physical security: make sure access to the product is physically protected. For example a stolen hard drive can disclose valuable company secrets.
- Network protection via firewalls and ACLs.
- Monitoring of alerts and logs for security violations/attacks
- Ensure that a server crash does not impact the business => regular backups and failover (redundancy).

Operations

- Operations (or simply ops) refer to the activities that a company performs to ensure that its IT assets are working as expected.
- The security activities that a company performs to make sure that its product assets are secure are part of operations.
- Note that for Cloud products, operations are the responsibility of the vendor not the customer!

Security of Cloud Products (1)

- Similarly to on-prem products, customers want to make sure that the product is designed to be secure and provides all the expected security features.
- What changes with Cloud is that customers also want to make sure the vendor's operations adhere to the best security practices.

Security of Cloud Products (2)

Security evaluation of a cloud product looks at the following aspects:

- Application security
- Infrastructure (OS, middleware and network) security
- Operational security
- Physical Security
- Administrative security
- Compliance with applicable standards.

Application Security

We will cover the following:

- Overview of the major software vulnerabilities
- The Secure Development Lifecycle (SDL) to protect against this vulnerabilities.

Software Vulnerabilities

There are many software vulnerabilities that can be exploited by attackers. We will review a few:

- The infamous buffer overflow
- Web applications vulnerabilities
- Poor cryptography
- Hard-coded secrets in source code and obscurity as a means to protect secrets.

Note: we will focus on cloud software. Obviously, end user applications (think browser, pdf acrobat reader, Flash, etc.) interest everyone but are not in scope for this module. However, root causes are similar in both scenarios.

All Input is Evil!

- As we go through the next slide, you will notice a recurring theme: poor validation of input leads to security problems.
- Input is usually thought of as something supplied by an ill-intentioned user while interacting with the application. However, many input vectors exist:
  - Input from IP packets and any protocol on top of it (HTTP, SIP, SNMP, XMPP, etc.)
  - Data read from a database
  - Data read from a file (JPEG, PDF, etc.)
- Bad input can also be supplied by an "honest" user by mistake!

Buffer Overflow

- Buffer overflow refers to the fact that a variable meant to be stored in memory “overflows” beyond the allocated space.
- When the variable overflows, the function return address can get smashed.
- It results from a developer not properly performing bounds checking when assigning data to memory.
- The vulnerability can be exploited in C and C++ programs. Java programs are not exposed.

Buffer Overflow Illustrated (1)
(from Wikipedia)

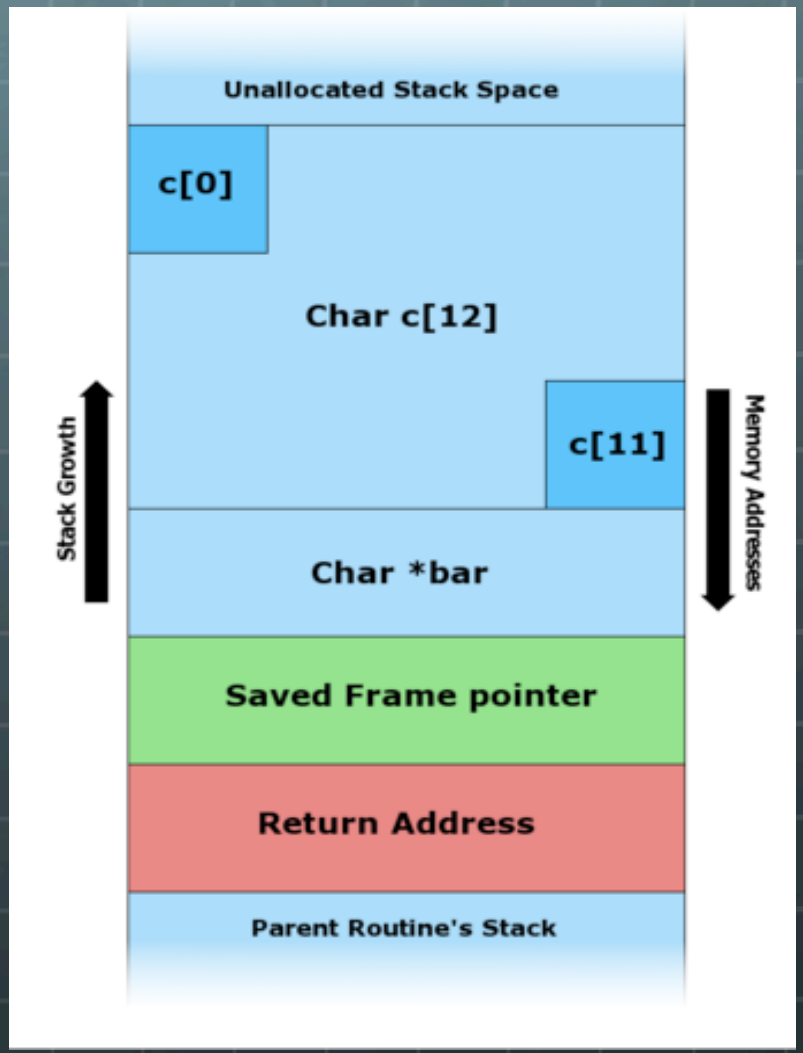
Consider the following program:

```
#include <string.h>
void foo (char *bar) {
        char c[12];
        strcpy(c, bar); // no bounds checking
}

int main (int argc, char **argv)
{
         foo(argv[1]);
}
```

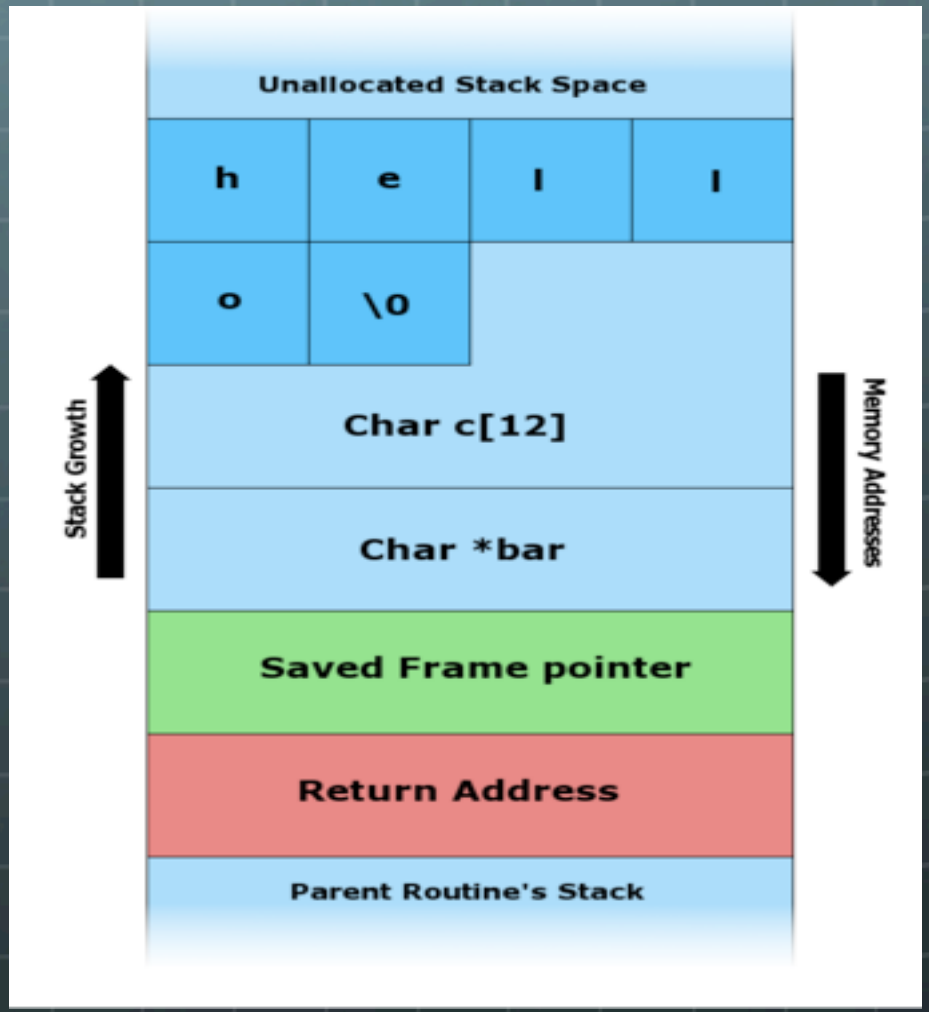
Buffer Overflow Illustrated (2)
(from Wikipedia)

Memory Layout



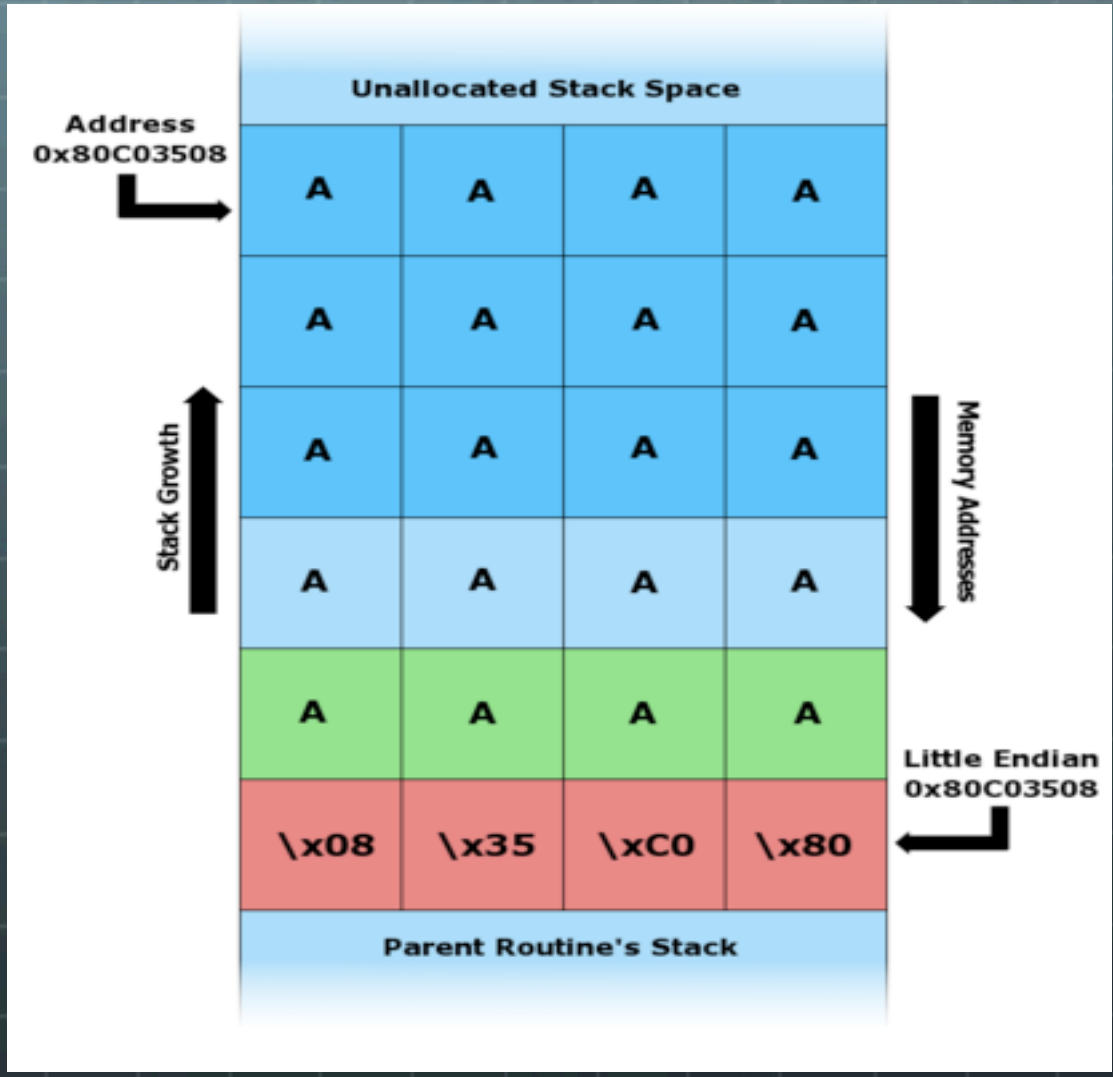
Buffer Overflow Illustrated (3)
(from Wikipedia)

With Valid Input ("hello")



Buffer Overflow Illustrated (4)
(from Wikipedia)

With Invalid Input ("AAAAAAAAAAAAAAAAA\x08\x35\xC0\x80")



Consequences of a Buffer Overflow

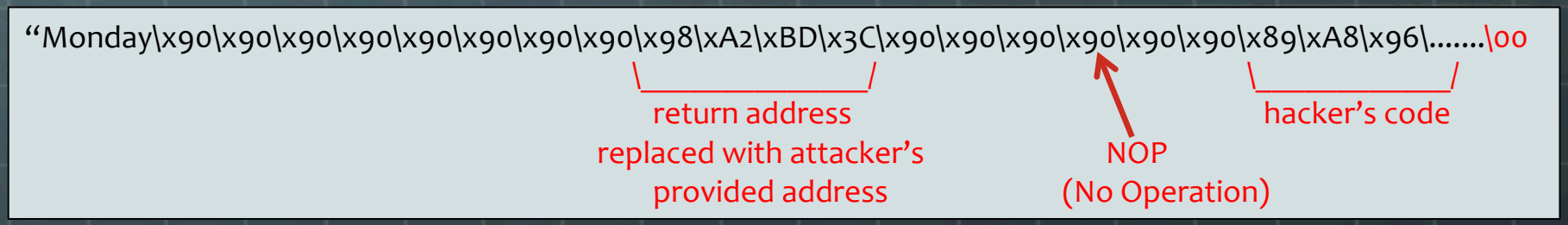
- In the most benign case, smashing the function return address causes the application to crash, which leads to a Denial of Service (DoS) scenario. When exploited, DoS impacts the application availability.
- A worse outcome is when an attacker crafts an input in a way that the return address points to his code. This leads to a code execution, which can lead to the attacker completely owning the server on which the application is running.

A Quick Word on DoS

- Denial of Service (DoS) is a condition where a product is no longer able to perform the intended service. An application crash is one such condition. It can either result from a poor design/coding or an active (DoS) attack.
- Poor coding examples include 1) a developer not freeing allocated resources (e.g. memory) after use, 2) improper check of error conditions, e.g. division by 0, 3) access of resources that require elevated privileges (OS routines). These flaws can be exploited by attackers.
- An example of an active DoS attack is when a website is flooded with malicious traffic preventing it from servicing legitimate traffic. A variant of this attack is called DDoS (Distributed DoS) where the attack originates from multiple sources (e.g. bots).

Remote Code Execution Example

- An application written in C++ reads a day of the current week (e.g. “Monday”) from a TCP packet and returns a list of events that happened that day.
- An attacker crafts a TCP packet and sends the following string:



- This will cause the attacker’s code to be executed on the server where the application is running!

Buffer Overflows Protection (1)

Buffer overflows can be prevented at different stages:

- Development: developer awareness, code reviews, use of Safe C library (http://sourceforge.net/projects/safeclib/)
- System configuration, compilation and runtime: DEP, Canary, ASLR

Buffer Overflow Protections (2)

- Safe C library: provides alternate safe functions to standard C/C++ functions that could lead to BO (strcpy, strcmp, strcat, etc.)
- DEP (Data Execution Prevention): OS feature that prevents code execution in memory areas that should only contain data.
- Canary value: a “marker” that is inserted between the stack variables and the return address of a function. If the value is changed after the function call ends, a BO is detected.
- ASLR (Address Space Layout Randomization): a technique that randomly changes the position of the executable, stack and libraries in memory to make it harder for attackers to know what to change the function return address to.

Web Application Vulnerabilities

- There are many!
- The OWASP (Open Web Application Security Project) organization maintains an ordered list of the top 10 web vulnerabilities (https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)
- We'll review the following:
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - Cross-Site Resource Forgery (CSRF)
  - Broken Authentication and Session Management

SQL Injection

- Websites routinely use input from users to do a database lookup. For example, you enter a product name and the website does a lookup to find the product details in the database.
- SQL Injection refers to the fact that an attacker crafts his input in such a way that the database query performs an action not intended by the developer.

SQL Injection Example

- A hospital website takes an "SSN" input from users and returns the SSN, patient name and date of the next visit. The website performs the following operations:
  1. ssn = patientForm.ssn
  2. query = "select ssn, name, date from patients where ssn=\"' + ssn + "\"";
  3. return an html page with ssn, name and date.
- An attacker could send the following input:

  563899367\' or 1=1 --
- The resulting query will be:

  select ssn, name, date from patients where ssn='563899367' or 1=1 –'
- The webserver will return all the rows in the database!

SQL Injection Damage is Real!

A few “exploits” from wikipedia:

- On March 7, 2014, officials at Johns Hopkins University publicly announced that their Biomedical Engineering Servers had become victim to an SQL injection attack carried out by an Anonymous hacker named "Hooky" and aligned with hacktivist group "RaptorSwag". The hackers compromised personal details of 878 students and staff, posting a press release and the leaked data on the internet.
- In August 2014, Milwaukee-based computer security company Hold Security disclosed that it uncovered a theft of confidential information from nearly 420,000 websites through SQL injections. The New York Times confirmed this finding by hiring a security expert to check the claim.

SQL Injection Protections

- Developer awareness! Validate input! Leverage a validation framework (like OWASP's ESAPI)
- Parameterized (or prepared) statement
- Escape input (e.g. chars like ' and -)
- Use tools to find potential SQL Injections.

Cross-Site Scripting (1)

- A Cross-Site Scripting (or XSS) condition exists wherever input from a user is reflected back to her in the response page.
- For example, you enter your name in a form (e.g. Jamal) and the response page would include it at the top (e.g. Jamal's page).
- If the input is not properly validated by the website developer, a script could be inserted in the input. For example: Jamal<script>alert('I can run!')</script>
- Why would anyone do that?

Cross-Site Scripting (2)

- A user injecting a script in a form has no useful purpose!
- However, if an attacker tricks a user into clicking on a link that contains script injection, then damage can be done! The trick is a case of phishing. A phishing message example: "Hey, check out this link: it's so cool" ☺
- The injected script can do nasty things in the victim's browser. Things such as: send the session cookie to the attacker!
- Links sent to victims are often encoded to hide the real effect(s) of the link.

XSS Protections

- Developer awareness! Check input! Leverage a validation framework (like OWASP's ESAPI)
- Encode output
- Tools can help find potential XSSs

CSRF

- Consider the following URL that results from a money transfer on a bank website:

https://www.myverysecurebank.com/transfer?amount=500&to=someonesaccount

- This is a perfectly valid use case. After all, people do transfer money.
- However consider an attacker crafting the following URL:

https://www.myverysecurebank.com/transfer?amount=50000000&to=attackersaccount

- The attacker can then trick a user into clicking the crafted link (another phishing scenario). If the user is authenticated at that time, the transfer will be validated by the website!

CSRF Protections

- Developer awareness!
- The website sends a random token with the transfer page. The token will be returned by the user and matched by the website. The website won't trust a link crafted by an attacker because the link will either contain a token an the website has not generated one or, if it had, the tokens will likely not match.
- Tools!

Broken Authentication and Session Management

- Authentication and Session management can be flawed in a number of ways exposing legitimate accounts to attackers.
- Examples of flaws include sending sessions ids in the URL or in non-encrypted channels and weak account creation, change and recover password.
- Impact can be very damaging
- Protection includes using sound design principles (such those defined by OWASP).

Weak Cryptography

- Weak cryptography (encryption, signing, hashing, random number generation, secure protocols) in an application can leak valuable information.
- Examples include weak password protection and use of outdated cryptography
- Protection includes using the latest strong vetted cryptographic algorithms by the cryptography community. NIST (National Institute of Standards and Technology) also maintains a list of approved algorithms.

Hard-Coded Secrets and Obscurity

- Inexperienced developers sometimes hard code keys and passwords in the source code assuming binary code cannot be reverse-engineered.
- How about cloud applications which are not shipped to customers? It's still a dangerous assumption and code can leak out (intentionally or not).
- Obscurity refers to the fact of hiding data by using ad hoc means (hidden files, inconspicuous directories, etc.). Obscurity is not security! Use vetted cryptography!

Security Development Lifecycle (SDL)

- The purpose of an SDL is to prevent application vulnerabilities (such as the ones we have seen).
- SDL is a list of activities that development (including testing) need to incorporate the development process in order to build secure software.
- It's a known fact that the cost of catching and fixing defects during development is much cheaper than after.

SDL Activities

- Security Awareness and training
- Code reviews and use of a security checklist.
- Static source code analysis
- Vulnerability scanning
- Use of safe libraries (e.g. Safe C)
- Keeping up to data with security patches of third party software (e.g. openssl, jdk/jre, etc.)
- Threat modeling
- Use of safe options/flags in software build
- Complying with a security baseline

Security Awareness and Training

- A security training should be part of the trainings given to newly hired developers.
- The training can be instructor led or through an online offering. Instructor led classes are preferable.
- All developers should re-take the training when there is a significant change in the training material.
- The training focuses on common vulnerabilities and the SDL process.

Code Reviews

- Peer code reviews are a good practice to identify issues early on (security, performance, bad programming, missing logs, etc.)
- Using a checklist for things to look for is a good practice.
- CMU has good checklists for major languages (https://www.securecoding.cert.org/confluence/display/seccode/CERT+Coding+Standards)

Static Source Code Analysis

- Static analysis tools scan source code for a variety of issues, including security ones.
- Available tools include JTest, Coverity, Findbugs, Checkstyle, PMD and Fortify.
- Examples of security issues that can be caught by SA tools include: use of uninitialized variables, use of unsafe C/C++ functions, lack of bounds checks, XSS, SQL injection, etc.
- Development teams have to build a process where source code is regularly scanned and findings are tracked and remediated.

Static Analysis Example

```
MCHERFAO-M-T2JD:bin mcherfao$ cat try.c
#include <stdio.h>
#include <string.h>


int main (int argc, char** argv)
{
        char* buffer;

        strcpy (buffer, "Something");
}
MCHERFAO-M-T2JD:bin mcherfao$
MCHERFAO-M-T2JD:bin mcherfao$
MCHERFAO-M-T2JD:bin mcherfao$ gcc try.c
MCHERFAO-M-T2JD:bin mcherfao$
MCHERFAO-M-T2JD:bin mcherfao$
MCHERFAO-M-T2JD:bin mcherfao$ ./clang --analyze try.c
try.c:9:9: warning: Function call argument is an uninitialized value
        strcpy (buffer, "Something");
        ^~~~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/include/secure/_string.h:83:3: note: expanded from macro 'strcpy'
  __builtin___strcpy_chk (dest, src, __darwin_obsz (dest))
  ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 warning generated.
```

Compilation is OK

Static Analysis is not!

Application Vulnerability Scanning (1)

- Static Analysis tools operate on the source code. Vulnerability scanning tools require running the application.
- There are 2 types of vulnerability scanning tools:
  - Tools that analyze the binary code at run time to identify memory management and library calls issues. They usually need to "instrument" the code. Tools include Purify and Valgrind.
  - Tools that inject some form of input into the application and observe the resulting output. We will focus on this kind of tools in the next slide.
- Running vulnerability scanning tools in development and/or testing should be part of any good SDL process.
- Vulnerability scanning can be also done against the OS and the middleware but we'll see this application later.

Application Vulnerability Scanning (2)

- Most popular tools deal with web applications scanning. Tools include AppScan, WebInspect, Burp Suite and others.
- To be effective, these tools need to "crawl" the pages of entire web application and be configured with credentials to access pages that require authentication.
- These tools can run automatically or assisted by a developer or tester. They are generally very good at finding potential injections, XSS, CSRFs and other web vulnerabilities. Trickier vulnerabilities require manual testing.

Application Vulnerability Scanning (3)

- If the product has support for a protocol (SIP, XMPP, SOAP, etc.), protocol testing is needed.
- Protocol testing uses a technique called fuzzing. Fuzzing creates variations on the fields/format expected by the protocol and observes how the product reacts. Codenomicon is the most popular tool. Python Scapy is another tool.
- Fuzzing can also be applied to applications that parse file formats (e.g. pdf/jpeg viewers).

Use of Safe Libraries

- The standard C library for string and memory manipulation is notoriously known to potentially lead to vulnerabilities (buffer overflow).
- Safe C library (http://sourceforge.net/projects/safeclib/) is a good alternative to the standard library.
- Safe libraries alternatives should be used whenever possible

Keeping up with Security Patches

- Any large application (cloud or not) uses 3rd party libraries at some point. Examples of 3rd party libraries include: openssl, jre, etc.
- Like any other software, vulnerabilities in 3rd party libraries can exist and can be exploited by attackers.
- The product development team needs to monitor the vendor's alerts and security bulletins and apply security patches. Critical patches should be applied immediately.

Monitoring 3rd Party Software Vulnerabilities Announcements

- Some websites are dedicated to tracking vulnerabilities and providing information on them (criticality, availability of patches, etc.)
- CVE (https://cve.mitre.org/) is one such website.

Threat Modeling

- Threat Modeling is an activity that product teams perform at design time in order to identify potential attack vectors and plan for mitigations or solutions.
- Tools exist to assist with Threat Modeling, like Microsoft's (http://www.microsoft.com/en-us/download/details.aspx?id=42518). However, the activity can also be done blackboard style or by going through a checklist.

Safe Build Options/Flags

- Compiler flags can protect against certain types of attacks.
- The fstack-protector and fstack-protector-all flags tell gcc (GNU Compiler) to protect against buffer overflow. Stack protection is enabled by default in MS Visual Studio but can be disabled with /GS- flag.

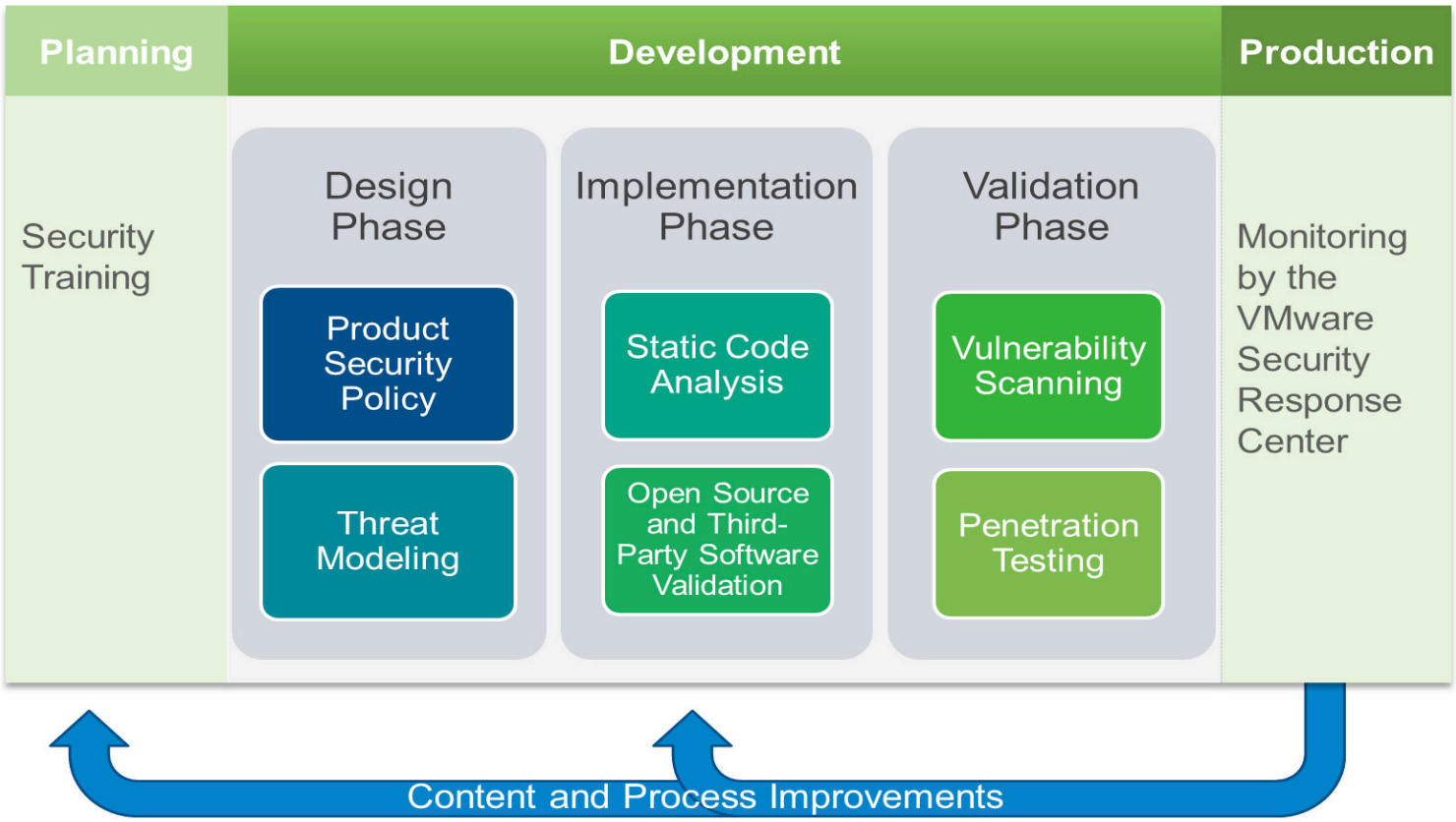
Compliance with a Security Baseline (1)

- A company can define a set of minimal security requirements that every product in the company must comply with.
- Each product does a gap analysis at the beginning of the development cycle to identify what needs to be done to meet the baseline requirements.

Compliance with a Security Baseline (2)

- Examples of requirements could be:
  - Don't use secrets/keys in source code.
  - All external communications must be TLS protected.
  - Enforce long passwords
- Some of the requirements in a baseline can also be found in code review and threat modeling checklists. However, the Security Baseline's goal is to identify existing security gaps whereas code review and threat modeling address new development.

Example: VMWARE's SDL



Security of Cloud Products

Security evaluation of a cloud product looks at the following aspects:

- Application security
- Infrastructure (OS, middleware and network) security
- Operational security
- Physical Security
- Administrative security
- Compliance with applicable standards.

Infrastructure Security

- Infrastructure refers to the OS, the middleware stack (e.g. Tomcat, Oracle, Apache, RabbitMQ, etc.) and the network used by a cloud application. Obviously these should be secured as much as the application itself.
- Securing the OS and middleware involve similar activities. We'll start with them then look at network security.

OS and Middleware Security

Securing the OS and Middleware involves the following activities:

- Hardening. The goal of hardening is to *reduce the surface attack* as much as possible.
- Monitoring security patches

In a cloud context, this activities straddle development and operations.

Hardening Steps (1)

- Remove any OS and middleware packages that are not absolutely necessary for the product. In fact, the right approach is to start with the minimum required packages and add packages as needed.
- Enable services only when needed. An example is ssh.
- Run services with the minimal privilege required so that damage caused by a compromise is contained.

Hardening Steps (2)

- Disable default accounts in middleware. Middleware (e.g. Oracle) often comes with default accounts whose passwords can be googled!
- Disable the root account in the OS and define service accounts per job role (troubleshooting, log collection, etc.). If elevated privilege is required, use sudo.
- Use industry benchmarks to harden OS and Middleware, e.g. CIS benchmarks (Center for Internet Security).

Monitoring Security Patches

- Similar to 3rd party development libraries, vulnerabilities can be found in OS and middleware (shellshock anyone?)
- Product teams need to monitor security bulletins and apply patches ASAP.

Network Security

Network Security involves 2 activities:

- The Security of the network elements (routers, switches)
- Hardening of the cloud application network design.

Security of Network Elements

- Network elements such as routers, switches, firewalls and others need to be secured. Steps to secure them include:
  - Staying up to date on security patches
  - Disabling default accounts
  - Enforcing strong authentication (strong passwords, disabling telnet for the more secure ssh, etc,.)
  - Routers and switches hardening (beyond the scope of this module)
- Note that if the cloud application is run in a public IaaS/ PaaS cloud, this activity is not required.

Hardening the Network Design

Hardening the network involves the following activities:

- Establishing strict control of the traffic between the public internet and the cloud application network.
- Segment the network on top of which the cloud application runs into security zones. This allows to contain a security compromise within the zone.
- Segment off the cloud application network from the corporate network, again to contain damage.

Security Zones

- Security zones are implemented creating a dedicated sub-network for each function (e.g. Front end, application logic, data, logging, management zones). Type of traffic between these networks is strictly controlled via network Access Control Lists (ACLs) and firewalls.
- Sub-networks are usually implemented via VLANs.
- If the cloud application runs on an IaaS/PaaS, it can leverage constructs provided by the IaaS/PaaS like security groups and VPCs in AWS.

Security of Cloud Products

Security evaluation of a cloud product looks at the following aspects:

- Application security
- Infrastructure (OS, middleware and network) security
- Operations security
- Physical Security
- Administrative security
- Compliance with applicable standards.

Operations Security

The operations team takes over security activities once the product development is done and the product is publicly available for use.

Note that OS and middleware security are the responsibility of both development and operations teams.

Operations Security Activities

- Logs Collection
- Monitoring
- Logs and Events Analysis
- Vulnerability Scanning
- Business Continuity (and Disaster Recovery)
- Incident Response

Logs Collection

- Logs are critical to flag to identify and investigate security incidents and for forensics.
- All events that significant from a security perspective need to be logged. For example: privileged actions.
- Logs are generated by the application, the OS, the middleware and network nodes (routers, switches, firewalls, etc.)
- Log must be consolidated in one location for easy access. Syslog and Syslog-ng are common mechanism for logging to a remote server.
- Logs need to be protected and only authorized personnel can have access to them. A retention period needs to be defined.

Monitoring

- This refers to the activity of monitoring the cloud application and its infrastructure (OS, middleware, network) for security incidents.
- There are basically 2 strategies for monitoring and they complement each other:
  - NIDS (Network-based Intrusion Detection System)
  - HIDS (Host-based Intrusion Detection System)
- NIDS and HIDS send alerts to the operations control center and events to the log store.

Network and Host IDS

- Network IDSs are network nodes that monitor the network traffic and look for malware/attack signatures. Snort is one the popular tools.
- The NIDS function can integrated in firewalls and routers.
- Host IDSs monitor the traffic entering a specific host and look for attack signatures. They also monitor systems logs, disk and cpu usage. Popular solutions include OSSEC and Nagios.

More on Network Monitoring

- On top of monitoring the contents of the network traffic for attack signatures, monitoring the network traffic usage can in itself be useful.
- A tool like Netflow gathers data about the network traffic like traffic volume per time period, percentage of TCP vs. UDP traffic, traffic type (audio stream, http, etc.).
- Netflow allows to build a profile of the traffic of a typical day. By monitoring deviations from the typical profile, attacks in progress can be detected.

Logs and Events Analysis

- A cloud application can generate huge volumes of logs and events (such as those generated by NIDS and HIDS). Analyzing them manually is not a good option.
- There are dedicated tools that can parse the logs, establish correlations and detect attack patterns. These tools are called **SIEM** (Security Information and Event Management).
- **Splunk** is one of the most popular SIEMs.

Vulnerability Scanning

- On top of monitoring patches announcements par third party providers, operations need to actively scan the infrastructure for vulnerabilities.
- This activity allows to pro-actively look for vulnerabilities instead of waiting for security bulletins. Also, operators can intentionally install software that is unpatched and this activity will flag it.
- Tools that scan OS and middleware include Qualys, Nessus and NMAP. Other tools target specific middleware like McAfee's MVMD for Oracle.

Business Continuity

- The product needs to be resilient in the face of crashes and compromises.
- This is usually addressed by performing regular backups so that business can resume in case of an incident. Backups need to have the same level of protection as the original data.
- If business continuity is desired even in the case of a major disaster (e.g. earthquake) then geographic redundancy is a must. Consider replicating data between the data centers.
- If a company can't afford a redundant data center at all times, it needs to consider options like mobile data centers.

Incident Response

- Incident response is a process that defines how a company reacts to a breach.
- The process needs to specify the steps involved and the owners/contacts for each one.
- Some of the steps involved in an incident response:
  - Detection
  - Containment
  - Analysis/Investigation
  - Remediation
  - Lessons learned

Security of Cloud Products

Security evaluation of a cloud product looks at the following aspects:

- Application security
- Infrastructure (OS, middleware and network) security
- Operations security
- Physical Security
- Administrative security
- Compliance with applicable standards.

Physical Security

- Physical security involves strengthening the data center buildings and the corresponding environment.
- Note that if the product is deployed in an IaaS or PaaS, physical security is the responsibility of the IaaS/PaaS provider.

Security of Cloud Products

Security evaluation of a cloud product looks at the following aspects:

- Application security
- Infrastructure (OS, middleware and network) security
- Operations security
- Physical Security
- Administrative security
- Compliance with applicable standards.

Administrative Security (1)

Humans are often said to be the weakest link in security. The following principles strengthen the security posture:

- **Training and raising awareness**: Security training are a must.
- **Least privilege**: operators in operation must only have the privileges they need for their job, not more. For example, an operator in charge of backups should not have the permission to manage accounts.
- **Separation (or segregation) of duties**: for security critical business processes that involve many tasks, assign different operators to the tasks to make it harder to abuse the process.

Administrative Security (2)

Additional principals:

- **Job Rotation**: rotate operators between different jobs so that operators are discouraged from unethical behavior.
- **Background Checks**: Newly hired operators need to undergo a background check, especially those tasked with sensitive responsibilities (e.g. access to customers data).
- **Non-Disclosure Agreements**: every employee must sign an agreement stating that company secrets will not be shared.
- **Change Management/Control**: the process that ensures that all changes are reviewed and tracked.

Security of Cloud Products

Security evaluation of a cloud product looks at the following aspects:

- Application security
- Infrastructure (OS, middleware and network) security
- Operations security
- Physical Security
- Administrative security
- Compliance with applicable standards.

Compliance

- Compliance with industry and government standards and regulations becomes relevant as soon as a product gains a large reach.
- Most of the standards require that the product provider maintains a security policy.
- A security policy is a corporate set of policies and procedures that defines how the company handles business continuity, incident response, access control, etc.

Standards

- CSA (Cloud Security Alliance) maintains a list of security requirements for cloud applications. It's not binding but complying with its requirements allows to meet most of the existing standards.
- ISO 27001 and SSAE16 are the most requested certifications by cloud consumers. Government consumers sometimes require compliance with FedRamp. Europe requires US companies to comply with the Safe Harbor directive in order to store EU customers data that have privacy implications.
- Depending on the industry sector, other standards might apply: PCI DSS for credit cards, HIPAA for the health sector, SOX for the financial sector, etc.
- Certifications against these standards are done by auditors.