Week 4 write up

web

Comment

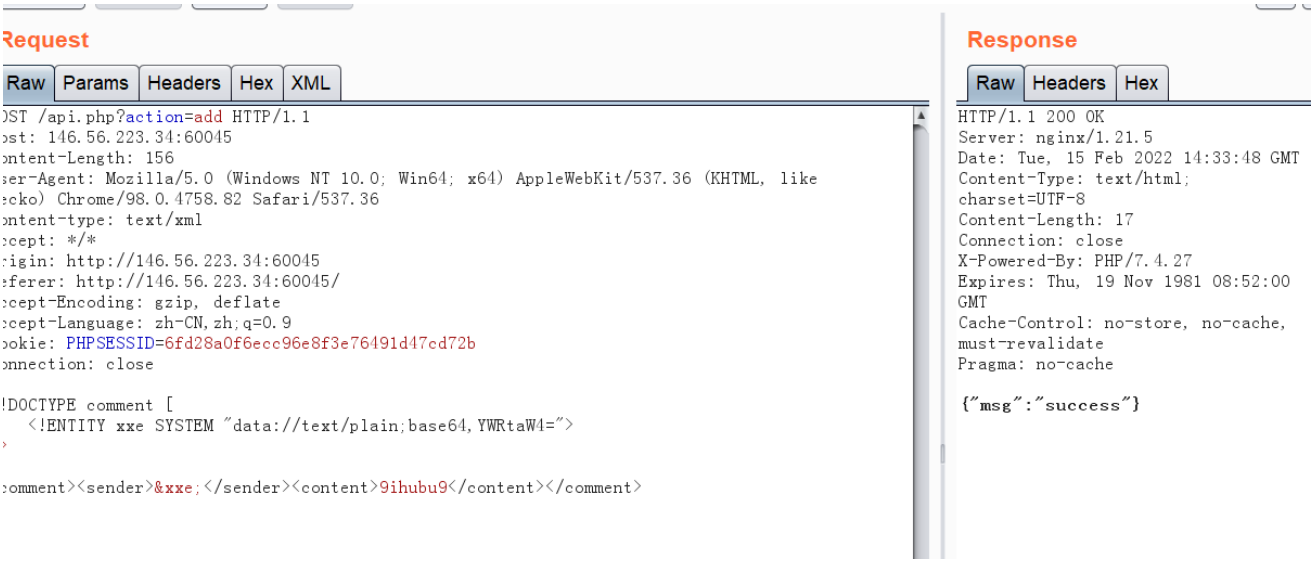
```
switch ($_GET['action']) {
    case 'get':
        get();
        break;
    case 'add':
        save();
        break;
    case 'info':
        echo json_encode(['unique_id' => $_SESSION['unique_id']]);
        break;
    default:
        http_response_code(400);
        echo json_encode(['error' => 'no such action']);
        break;
}
```

根据源码大致三个状态，同时根据hint 得知要用到伪协议

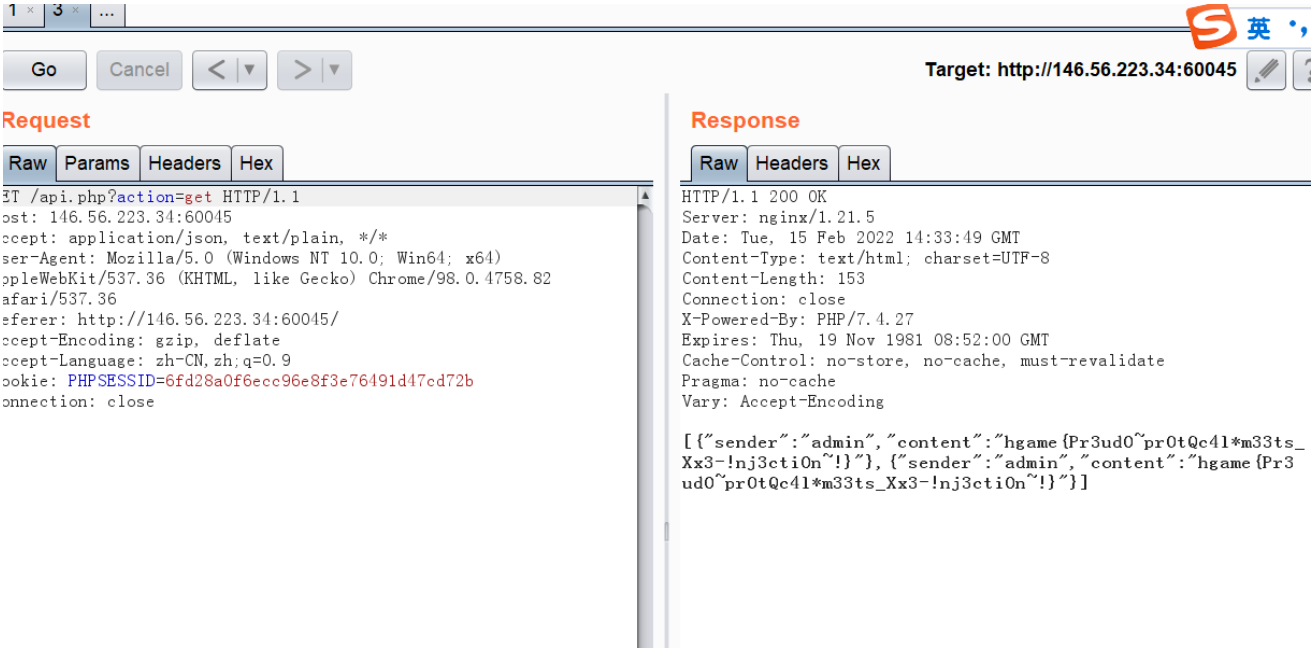
观察代码

```
function waf($str): bool {
    if (preg_match('/file|glob|http|dict|gopher|php|ftp|ssh|phar/i',
$str)) {
        return true;
    }
    return false;
}



    if ($attrs->sender == 'admin' && !preg_match('/admin/i', $str)) {
        $flag = 'hgame{xxxxx}';
        $attrs->content = $flag;
    }
```

发现伪协议中 data没有被过滤，而且在$str中不包含‘admin’但是sender中有于是用伪协议做封装数据



拿到flag



crypto

ECC

得知是椭圆曲线加密于是学习相关知识后可以用sagemath解题

```
SageMath version 9.3, Release Date: 2021-05-09
Using Python 3.7.10. Type "help()" for help.

sage: p = 749970215594340659752724316266187207258384730917219366165603590006486518915 07
....: a =  617390437303328599782364690079486669975105442123623866290620320949253535196 57
....: b =  878217828184778176098825263164797214909198150136680967719923600024676578273 19
....: k = 9365387427217610758445998205852708160408387118279781620477264450962327106123 1
....: E = EllipticCurve(GF(p),[a,b])
....: c1 = E(144556136662118995760188351651324381020119882646071465119382497448719649460 84,255065825705812897146126404932582998138031575617962473306937681467630357919 42)
....: c2 = E(375548711626194567091835091226739296364576222518801992350547345237824838699 31,713920555406167365392679609893042870836292885303984745907823663848738144778 06)
....: m = c1*c2
....: cipher_left = 68208062402162616009217039034331142786282678107650228761709584478779998734710
....: cipher_right = 27453988545002384546706933590432585006240439443312571008791835203660152890619
....: print(cipher_left/m[0])
49303314923700944603626 0
sage: print(cipher_right/m[1])
12748090025655102209539391 7
```

```
C:\Users\pc\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/pc/PycharmProjects/pythonProject/16.prng.py
b'hgam'b'e{me'b'Rsen'b'ne!t'b'WisT'b'ER~i'b'S^A*'b'WIDe'b'lY-U'b'SEd^'b'pSEU'b'Do&r'b'AndO'b'M:nU'b'mBEr'b'!GeN'b'ErAT'b'Ion?'b'Algo'b'rITh'b'M}'
Process finished with exit code 0
```

PRNG

此题是梅森旋转算法

观察脚本后将第一次旋转后的数据代入

然后即可模拟之后的输出

```
data=[888058162, 3094055443, 1404990361, 1012543603, 448723884,
2580444236, 201608779, 1062995809, 1348787313, 2980019361, 2245025385,
494977308, 4042503808, 275744301, 406611131, 142226472, 3871761759,
3888795536, 2617489687, 1220227074, 342928858, 3728958896, 1477077966,
1433151407, 1119405037, 330145973, 3547181160, 2123007249, 3739964132,
1794129718, 2739743522, 2291585121, 3013727731, 1536788463, 247633572,
408079265, 3025555185, 1604681922, 2848997116, 3646041955, 1059445774,
2849764176, 2638965889, 1232303180, 759521642, 2257008531, 3932082254,
1052428413, 4017559916, 3505694223, 1418363972, 477751107, 4266295945,
3832138928, 245251422, 1964323108, 2453472918, 3029032760, 323619451,
2548825339, 3410027991, 278143595, 816124107, 245705463, 4173686519,
4100831820, 3599257115, 2274885516, 3954736394, 198254482, 1050449178,
3933150558, 899220021, 597474632, 1823539097, 3340511318, 2144918682,
2310527451, 264391694, 69923676, 3266017310, 3199627722, 4035962745,
932969905, 2832951013, 2182887504, 1374919242, 2978944795, 1840647233,
3510878043, 3250544991, 4255542321, 804377010, 1286980519, 1980427321,
2893246724, 1745353148, 1406140332, 4101848568, 3227434698, 1869729934,
2638181242, 1270111849, 2387910792, 3411542702, 2793303435, 2455337459,
2802808043, 2418872990, 1043274549, 144911746, 2312236858, 780373658,
1527499811, 3402753408, 2617924770, 1659648360, 2714315441, 4202103851,
244677433, 1963258902, 3851363324, 3454195559, 813228826, 3944899734,
3697685234, 1613584167, 1874570879, 1592343033, 4194241173, 551902434,
3460909265, 4122075287, 176665387, 152849760, 3593212904, 952880017,
1793357635, 2052902220, 807859486, 334839380, 3485132343, 2113403566,
3259106798, 1443078482, 2434820318, 1347902400, 2344061487, 141766876,
2641586235, 287277458, 2385094526, 1510128758, 348957861, 2861038633,
1135611795, 4289024199, 1021202791, 2460872523, 3837050794, 4092005952,
52622948, 387056916, 3102913460, 4098715316, 154916530, 2890197932,
1441566957, 2368779800, 271808452, 3566810840, 2227022452, 316480679,
603893066, 2121889912, 4208763743, 3098334580, 721958838, 3848020801,
1029884135, 832405094, 2276817394, 981553190, 246940442, 1069231974,
3275216531, 58945988, 4100121200, 230446475, 2396021649, 4608139,
3468707911, 3249498323, 315898153, 3280797960, 388108494, 1110548082,
2357147660, 2336724751, 4047583630, 2108667879, 2784078579, 1170844412,
3920262445, 3564073655, 590490534, 1645945278, 2487463163, 434409966,
1563546251, 888601967, 1913513318, 1327448740, 2504517969, 304688984,
1443685450, 4040619940, 3601250858, 4097529433, 4260590151, 575843085,
1114360271, 2186035374, 2821388594, 3763206347, 4283149630, 4097168778,
1924538037, 3272064650, 1689166701, 1352331676, 520525342, 2954296222,
2629516330, 3674317458, 231784130, 1930132422, 4169222397, 1638784833,
1245667959, 1253759350, 1154928813, 66021172, 3217915692, 4159785573,
3798512628, 2945489695, 700725579, 3940231312, 1960713279, 3289722468,
2970919839, 1356139680, 1141841193, 629177162, 3696263539, 1084872874,
4294077062, 1115547807, 3421092527, 611575307, 7808529, 2784523837,
1267307982, 1538837032, 4038330055, 3262951566, 3139820067, 1249725729,
757191354, 3025188720, 291705345, 575676661, 3023956263, 1045504889,
205204207, 1777650027, 1956698897, 996147619, 1470431, 2275722398,
2666078800, 470333070, 1306693906, 2968672077, 2476023772, 2645573325,
3939390068, 2874886754, 4226430090, 2290851636, 3707585663, 109770347,
```

127373916, 815817847, 1565834917, 636869794, 4062053412, 583594822,
3782553071, 3293311273, 2801932604, 2647080862, 1514083254, 3534640458,
342361004, 3266111849, 2157351044, 1851728420, 3412596866, 2793236910,
3758306563, 1799548561, 952631672, 912455646, 2894404493, 2194084105,
119615608, 2071058651, 1524462411, 900936180, 3697554830, 3501838982,
2874465656, 2501478689, 1069024222, 3135689372, 1168458702, 1966524629,
36400028, 2704775319, 4030739700, 3985599923, 2778920518, 2669538325,
1951594393, 795749079, 665593501, 3007338649, 1535343068, 2031873237,
3202423789, 560224943, 1290838890, 2545130826, 695695377, 3048615291,
1957903923, 1986693779, 2594986519, 3396211122, 2625687092, 575329062,
2852671310, 3472799759, 715985207, 1660331651, 958648594, 305711662,
75621441, 548447557, 2473842353, 2110558182, 3321750402, 2415793078,
815198061, 1258834500, 972966677, 3267046345, 2923564883, 518207679,
1662309775, 278933232, 4294256390, 2444117793, 2241879973, 3915962245,
3836532482, 3449260219, 1092128833, 3177300913, 874588042, 1185436845,
2064537788, 364292705, 3802247898, 3122264959, 186651829, 2789447523,
797964681, 897671294, 1504956985, 2294012382, 3916152546, 177325516,
2741945226, 4188655695, 2738134558, 557326292, 1625014790, 2945266389,
1843516240, 644046640, 3853456819, 3456105042, 3467742754, 2885173326,
812088996, 1238970324, 766072156, 2675925963, 1667463511, 2808303112,
1638756770, 260047996, 1117661655, 346883777, 2268712532, 1904918136,
513102466, 1024624509, 2154277089, 4147814745, 3681688842, 2233642964,
3135674550, 1259551210, 3286048484, 4271168802, 4227197378, 3310884772,
2063705584, 791399172, 4069266828, 1511606526, 1047713396, 615906401,
2805111822, 499223767, 740832370, 351782725, 2258776891, 1837046713,
3969757168, 2873152110, 4214869805, 3416771254, 2527945969, 3279116532,
1217038009, 4014402228, 3696705795, 1877774112, 3928347956, 959715122,
1612979629, 4045688071, 2403021083, 424891533, 1887765641, 2090726432,
2897940431, 268403838, 3447542890, 575011346, 2559143209, 532649938,
3625398853, 2077769196, 1598653066, 3104923961, 3594500739, 675029389,
579180583, 2024117612, 1351780728, 654841863, 769835263, 1431012736,
2369300321, 4157341752, 1968305076, 2086919554, 3075265115, 2128974418,
3144501489, 3993066430, 1121959765, 1373765135, 4232964375, 2264170351,
11814235, 1797654983, 3382686935, 2541491040, 3540726136, 1330685654,
4123114026, 2521290625, 3357439706, 3331159097, 2298656231, 3446738535,
290996369, 3020977553, 849241175, 3469792522, 4119898263, 1339695718,
2125209134, 3620160106, 1063375386, 1656465852, 2505508266, 3958528861,
3497875682, 3112358345, 3675237811, 1109625127, 2672368219, 1983461371,
3579663373, 1969195060, 225618775, 653511251, 3508369415, 4127429853,
828877800, 4286770015, 1474706143, 870777512, 804917422, 3913439258,
2433991646, 2742831709, 4289045475, 2899508500, 185462457, 4178107803,
2671443073, 2701796854, 1170522896, 1599880638, 1410722361, 3977867960,
1263177666, 2159508450, 2704509681, 1540819416, 1836499452, 1667451095,
3799477506, 157146600, 3717470672, 89865758, 3815588203, 1929105788,
861643665, 684514017, 3519778437, 2712956097, 1004423983, 1540346552,
2617389519, 2754800020, 870994822, 1702399767, 3526294475, 3251290865,
2365820957, 1915675760, 1828371367, 3737352795, 2511512700, 1080446781,
2565191059, 2412448535, 3719988291, 1434643780, 4163492408, 1359345746,
1457543102, 2389534435, 2800945892, 2646700564, 1719588203, 999665519,

```
3120652917, 1800116770, 3247314137, 4261164550, 1503462948, 3017762189,
263481701, 1754772485, 869168639, 604192231, 498759780, 2602535702,
3346756344, 2836267314, 2073734260, 3445425559, 4051271696, 1647518162,
401835417, 1968629992, 2854677838, 2381566661, 3144829468, 519547510,
3058642603, 3944140819, 1248923220, 1050321901, 3218172519, 376999645,
184187381, 3837095155, 3363256702, 751966993, 3419533016, 4028456468,
1156797460]
class MersenneTwister:
    __n = 624
    __m = 397
    __a = 0x9908b0df
    __b = 0x9d2c5680
    __c = 0xefc60000
    __kInitOperand = 0x6c078965
    __kMaxBits = 0xffffffff
    __kUpperBits = 0x80000000
    __kLowerBits = 0x7fffffff

    def __init__(self, seed = 0):
        self.__register = [0] * self.__n
        self.__state = 0

        self.__register[0] = seed
        for i in range(1, self.__n):
            prev = self.__register[i - 1]
            temp = self.__kInitOperand * (prev ^ (prev >> 30)) + i
            self.__register[i] = temp & self.__kMaxBits

    def __twister(self):
        for i in range(self.__n):
            y = (self.__register[i] & self.__kUpperBits) + \
                    (self.__register[(i + 1) % self.__n] &
self.__kLowerBits)
            self.__register[i] = self.__register[(i + self.__m) %
self.__n] ^ (y >> 1)
            if y % 2:
                self.__register[i] ^= self.__a
        return None

    def __temper(self):
        if self.__state == 0:
            self.__twister()

        y = self.__register[self.__state]
        y = y ^ (y >> 11)
        y = y ^ (y << 7) & self.__b
        y = y ^ (y << 15) & self.__c
        y = y ^ (y >> 18)
```

```
        self.__state = (self.__state + 1) % self.__n

        return y

    def __call__(self):
        return self.__temper()

    def load_register(self, register):
        self.__state = 0
        self.__register = register


class TemperInverser:
    __b = 0x9d2c5680
    __c = 0xefc60000
    __kMaxBits = 0xffffffff

    def __inverse_right_shift_xor(self, value, shift):
        i, result = 0, 0
        while i * shift < 32:
            part_mask = ((self.__kMaxBits << (32 - shift)) &
self.__kMaxBits) >> (i * shift)
            part = value & part_mask
            value ^= part >> shift
            result |= part
            i += 1
        return result

    def __inverse_left_shift_xor(self, value, shift, mask):
        i, result = 0, 0
        while i * shift < 32:
            part_mask = (self.__kMaxBits >> (32 - shift)) << (i * shift)
            part = value & part_mask
            value ^= (part << shift) & mask
            result |= part
            i += 1
        return result

    def __inverse_temper(self, tempered):
        value = tempered
        value = self.__inverse_right_shift_xor(value, 18)
        value = self.__inverse_left_shift_xor(value, 15, self.__c)
        value = self.__inverse_left_shift_xor(value, 7, self.__b)
        value = self.__inverse_right_shift_xor(value, 11)
        return value

    def __call__(self, tempered):
        return self.__inverse_temper(tempered)
```

```
class MersenneTwisterCracker:
    __n = 624
    def __init__(self):
        inverser  = TemperInverser()
        register  = [inverser(data[i]) for i in range(self.__n)]
        self.__mt = MersenneTwister(0)
        self.__mt.load_register(register)

    def __call__(self):
        return self.__mt()
from libnum import n2s
if __name__ == "__main__":
    a=[]
    flag=[3437104340, 508103176, 1635844121, 878522509, 1923790547,
1727955782, 1371509208, 3182873539, 156878129,1757777801, 1472806960,
3486450735, 2307527058, 2950814692, 1817110380, 372493821, 729662950,
2366747255,774823385, 387513980, 1444397883]
    mtc = MersenneTwisterCracker()
    for i in range(21):
       print(n2s(flag[i] ^ mtc()),end="")
```

解密即可