Dragon CTF 2019 – Arcane Sector

Dragon CTF to kontynuacja tradycji CONFidence CTF. Organizowane przez Dragon Sector (najstarszy aktywny polski zespół CTFowy) zawody ponownie odbyły się pod koniec zeszłego roku na konferencji Security PWNing i trzymały bardzo wysoki poziom. Na zawodników czekały interesujące zadania sieciowe, błędy kryptograficzne do wykorzystania, niszowe architektury do analizy oraz usługi czekające tylko, żeby ktoś je wyeksploitował. Ponownie w tej edycji imprezy pojawiła się gra, w której było trochę z wszystkiego – i to ją opiszemy w tym numerze.

CTF	Dragon CTF
Waga CTFtime.org	99 (https://ctftime.org/event/887/)
Liczba drużyn (z niezerową liczbą punktów)	14
System punktacji zadań	Od prostych (100) do trudnych (500)
Liczba zadań	18
Podium	1. p4 (Polska) – 2340 pkt. 3. PPP (Stany Zjednoczone) – 1700 pkt. 3. ALLES! (Niemcy) – 1620 pkt.
Zadanie	Arcane Sector (Drow number guessing, Under the mountain, Expensive flag, Raiding party)

O ZADANIU

Gra Arcane Sector pojawiła się na konkursie Dragon CTF już drugi raz (poprzednią edycję opisywaliśmy w numerze 10/2018). Jak zwykle wyróżniała się na tle innych zadań na kilka sposobów. Po pierwsze, została stworzona w całości (poza większością grafik) na potrzeby CTFa. Składała się ona z wielu powiązanych, ale niezależnych wyzwań. Podobnie jak w 2018 roku, udało nam się w pełni połamać zadanie i znaleźć wszystkie flagi. Na listopadowej konferencji w 2019 roku rozwiązywanie jej przypadło w całości autorowi niniejszego artykułu. Żadna z czterech części nie była zbyt trudna (to w zasadzie nasze główne zastrzeżenie), ale każda była w pewien sposób ciekawa. Ponownie, zanim przejdziemy do rozwiązania, chciałbym wszystkich zachęcić do spróbowania swoich sił samodzielnie (https://github.com/gynvael/arcanesector2) – źródła zostały udostępnione publicznie na githubie, więc serwer z klientem każdy może postawić u siebie. Dla tych, którzy nie mają tyle czasu/cierpliwości, nasze rozwiązanie poniżej.

Do zadania logowaliśmy się bezpośrednio klientem dostarczonym przez organizatorów. Oczywiście mogliśmy go dowolnie przerabiać (co było pomocne, a czasami nawet konieczne do pokonania zadań). Natomiast serwer, na którym stały zadania, był dla nas całkowicie niedostępny, chociaż mogliśmy dla celów testowych postawić swoje instancje.

Zapraszamy na autorskie szkolenia
z zakresu bezpieczeństwa IT

{ Bezpieczeństwo aplikacji WWW }

{ Offensive HTML, SVG, CSS and other Browser-Evil }

{ Wprowadzenie do bezpieczeństwa IT }

{ Szkolenie przygotowujące do egzaminu CEH }
(Certified Ethical Hacker)

www.securitum.pl/oferta/szkolenia

Patroni medialni: sekurak.pl rozwal.to

Zadanie Drow Number Guessing, kategoria: kryptografia

Proste i przyjemne zadanie, chociaż po konkursie okazało się, że rozwiązaliśmy je nie do końca po myśli autora… Ale o tym później. Tak jak w prawie wszystkich zadaniach, żeby dostać flagę, musimy wejść w interakcję z NPCem. W tym przypadku jest to Hazardzista – znajdujemy go w elfiej wiosce i zagadujemy (Rysunek 1).

Rysunek 1. Hazardzista proponuje grę o flagę

Proponuje nam on interesujący układ – my podołamy jego wyzwaniu, a on w zamian za to podzieli się z nami flagą. Niestety, diabeł tkwi w szczegółach (Rysunek 2).

Rysunek 2. O jakiej liczbie między 0 a 10000000000 myślę?

Okazuje się, że wyzwanie, na które się zgodziliśmy, polega na odgadnięciu liczby pomiędzy 0 a 10000000000. Biorąc pod uwagę, że każda próba kosztuje nas 10 sztuk złota, łatwo policzyć, że koszty ataku siłowego szybko przekroczą budżet Warszawy[1]. Musimy znaleźć jakiś lepszy sposób. W tym celu otwieramy plik *npc.py* i czytamy definicję klasy `Gambler` (Listing 1):

Listing 1. Klasa odpowiadająca za Hazardzistę (mniej ważne fragmenty wycięte)

```
class Gambler(Mob):
 def __init__(self, world):
   self.seed = bytearray(os.urandom(32))

def on_heard_talking(self, player, text):
   text = text.lower().strip()

   if "hello" in text:
     self.say("Hello. Do you feel lucky today traveler?")
     self.say("If you beat my challenge, I'll give you a flag.")
     self.note("Say 'gamble' to start the challenge "
         "(costs 10 gp)")
     return

   if "gamble" in text:
     if player.gold < 10:
       self.say("Sorry to say it, but it seem you "
           "don't have anything to bet.")
       return
     player.gold -= 10

     self.say("Perfect! Get ready!")
     self.say("I'm thinking of a number between "
        "0 and 10000000000. Can you guess it?")
     self.gamblers[player.id] = self.gen_number(player)
     return

   v = int(text)

   good = self.gamblers[player.id]
   del self.gamblers[player.id]

   if v != good:
     self.say("Ah, no. I was actually thinking of %i." % good)
     return

   self.say("Well done! Here's your reward.")
   # ...
```

Jest to relatywnie dużo kodu, ale dzieją się tu tylko dwie ważne rzeczy:

- W momencie podejmowania wyzwania gracz traci 10 monet, a w tablicy `self.gamblers` zapisywana jest wybrana liczba.
- W momencie kiedy gracz podaje liczbę, sprawdzane jest, czy zgadza się z wybraną. Jeśli tak, dostaje w nagrodę flagę. Jeśli nie, dowiaduje się, jaka była poprawna odpowiedź i nie dostaje niczego

Cóż, ta część wygląda OK. Pozostaje pytanie, jak generowany jest „losowy" numer, który zgadujemy (Listing 2)?

Listing 2. Generacja kolejnej liczby do zgadywania

```
def gen_number(self, player):
   n = 146410935049

   for v in self.seed[:16]:
     n = (n + v * 186577649851) % 257084507917

   n = (n ^ (int(time.time()) * 117865751629)) % 257084507917

   for v in self.seed[16:]:
     n = (n + v * 186577649851) % 257084507917

   n = (n + player.id * 117865751629) % 257084507917

   return n % 10000000000
```

Tak jak się spodziewaliśmy – algorytm wygląda bardzo podejrzanie. Korzysta on tylko z dwóch zmiennych:

- tablicy `seed` zawierającej 32 bajty silnej losowości.
- wartości `time.time()`, czyli tzw. Unix timestamp (liczba sekund od północy 1 stycznia 1970).

Timestamp jest łatwy do przewidzenia – w końcu wystarczy dokładnie zmierzyć czas podczas rozmawiania z NPC. Gorzej z ziarnem (`seed`) – jego praktycznie nie mamy szans zdobyć.

Popatrzmy jednak, co dokładnie dzieje się z liczbami, które dodajemy. Po pierwsze, wszystko dzieje się modulo 257084507917. Z własności operacji modulo możemy tę operację wyciągnąć na koniec. Dodatkowo kod można podzielić na kilka części:

- Początkowy stan,
- Pierwsze akumulowanie ziarna (pętla `for`),
- Xor z czasem,
- Drugie akumulowanie ziarna (pętla `for`),
- Dodanie `playerid`.

Zakładając, że dobrze zmierzymy czas podczas rozmowy, jesteśmy w stanie przewidzieć tutaj wszystkie operacje, poza drugą i czwartą, które wymagają ziarna. Przenieśmy operację modulo na koniec, a później rozpiszmy tę pętlę dokładniej (Listing 3):

1. 21.4 mld zł na 2020 rok.


Szkolenia
testerzy.pl
Lepsza jakość testowania
Kurs praktyczny
ZAWÓD TESTER
Zostań testerem oprogramowania
- naucz się jak dobrze testować oprogramowanie
- zdaj pożądany egzamin certyfikujący ISTQB!
Więcej na szkolenia.testerzy.pl

Listing 3. Oryginalna zawartość pętli

```
for v in self.seed[:16]:
    n += v * 186577649851
```

Można uprościć to równanie przez wyciągnięcie mnożenia (Listing 4).

Listing 4. Optymalizacja: mnożenie możemy wyciągnąć poza pętlę n_sum = 0

```
for v in self.seed[:16]:
    n_sum += v
n += n_sum * 186577649851
```

Patrząc na takie równanie, okazuje się, że atak *brute-force* nie jest już nierealistyczny. W końcu każdy bajt w tablicy może przyjąć tylko wartości między 0 a 255. To znaczy, że łącznie musimy sprawdzić tylko 4096 możliwości dla pierwszej i tyle samo dla drugiej pętli. Razem 16777216 możliwości – liczba operacji kompletnie trywialna dla współczesnych komputerów

Możemy to już przełożyć na atak. Musimy otrzymać liczbę od elfa, zapisać go do zmiennej `generated_number` oraz zapisać czas zadania pytania do zmiennej `curtime`. Następnie wystarczy przeprowadzić atak siłowy (Listing 5):

Listing 5. Kod znajdujący parametry ziarna użytego podczas generowania liczby

```
curtime = 123  # unix timestamp generacji liczby
generated_number = 456  # wygenerowana liczba
playerid = 0  # id gracza, którego używamy

for s0 in range(0, 16*256):
    for s1 in range(0, 16*256):
        n = 146410935049
        n = (n + s0 * 186577649851) % 257084507917
        n = (n ^ (curtime * 117865751629)) % 257084507917
        n = (n + s1 * 186577649851) % 257084507917
        n = (n + playerid * 117865751629) % 257084507917
        n = n % 10000000000
        ifn == generated_number
            print s0, s1
```

W ten sposób otrzymamy poprawne parametry `s0` i `s1`. Możemy wtedy wygenerować liczbę kolejny raz i podstawić nowe wartości (Listing 6):

Listing 6. Finalne rozwiązanie – podstawiamy parametry s0 i s1 do wzoru

```
curtime = 234  # unix timestamp generacji kolejnej liczby
playerid = 0  # id gracza – którego używamy
s0 = 2312  # suma seed[0..15]
s1 = 1818  # suma seed[16..31]
n = 146410935049
n = (n + s0 * 186577649851) % 257084507917
n = (n ^ (curtime * 117865751629)) % 257084507917
n = (n + s1 * 186577649851) % 257084507917
n = (n + playerid * 117865751629) % 257084507917
n = n % 10000000000
```

W ten sposób uzyskujemy flagę:
`DrgnS{RussianRoulette?ImagineDrowRoulette}`

Na zakończenie ciekawostka. Dało się to zadanie też rozwiązać w znacznie inny, w sumie prostszy sposób. Czy masz jakiś pomysł, jak?

Rozwiązanie, które przewidział autor zadania, polegało na współpracy między graczami i działało tak:

- Gracz 1 podchodzi i zaczyna wyzwanie (mówiąc „gamble”).
- W tym samym momencie Gracz 2 robi to samo.
- Gracz 1 zgaduje dowolną liczbę (np. 12) – nie wygrywa flagi, ale poznaje poprawną odpowiedź…
- Gracz 1 przekazuje poprawną odpowiedź do Gracza 2… Który ją wpisuje i zdobywa flagę.

To rozwiązanie jednak wymagało udziału dwóch graczy. Z tego powodu, mimo że wygląda prościej, mało który zespół rozwiązał je w ten sposób.

Zadanie Flag Under The Mountain, kategoria: błędy logiczne

Kolejne dwa zadania nie wymagają już matematyki, a jedynie znalezienia błędów w samej logice aplikacji. W przypadku zadania Flag Under The Mountain flaga znalazła się w pokoju zamkniętym ze wszystkich stron. Jeśli ktoś brał udział w CTF w roku 2018 (albo czytał nasze rozwiązanie), być może pamięta bardzo podobne zadanie. Amulet teleportacji działał tak, że czar `bind_teleport_ring` zapisywał aktualną pozycję gracza i pozwalał się do niej teleportować w przyszłości. W poprzedniej wersji zadania był w nim błąd, przez to, że niezależnie zapisywał pozycję x i y, co pozwalało teleportować się przez ściany. Niestety, amulet został już naprawiony i nie dało się powtórzyć tej sztuczki.

Z drugiej strony zostało dodane nowe zaklęcie „Ghostly Visions”. Pozwala ono dowolnie przechodzić przez ściany. Niestety, w stanie „duchowym” zabroniona jest jakakolwiek interakcja ze światem, więc nie da się np. przeczytać flagi. Uproszczony kod funkcji `ghostly_visions` jest przedstawiony poniżej (Listing 7):

Listing 7. Uproszczony kod zaklęcia „Ghostly Visions”

```
def spell_ghostly_visions(spell, player, world):
    x, y, d = player.pos_x, player.pos_y, player.direction
    player.block_interactions = True
    player.allow_clip = True
    player.visible = False

    for _ in world.YIELDING_sleep(15.0):  # yield from
        yield _

    world.map.remove_mob(player)
    player.pos_x = x
    player.pos_y = y
    player.direction = d
    world.map.add_mob(player)

    player.block_interactions = False
    player.allow_clip = False
    player.visible = True
    player.send_ground()
```

Jak widać, cały czas kiedy gracz może przechodzić przez ściany (flaga `allow_clip`), gracz ma również niepożądany status `block_interactions`. Nasz pierwszy pomysł był taki, żeby przejść do pokoju jako duch i rzucić zaklęcie wiązania teleportu w środku. Niestety, zostało to przewidziane – w grze nie można rzucać dwóch zaklęć naraz.

I tutaj wchodzi do gry trzeci element, który trzeba było wykorzystać w celu rozwiązania zadania – zapisywanie zaklęć na zwoje. W grze możliwe było zapisanie zaklęcia, tworząc zwój umożliwiający jednorazowe rzucenie tego czaru w przyszłości. Na pierwszy rzut oka nie pomaga to w niczym (np. koszt energii magicznej jest dalej taki sam). Ale w tym przypadku okazało się, że nawet w stanie zablokowanych interakcji dalej można korzystać z posiadanych przedmiotów! Dało nam to prosty plan na zdobycie flagi:

- Rzucenie zaklęcia wiązania teleportu na zwój,
- Rzucenie zaklęcia „Ghostly Visions” na siebie,
- Przejście do pokoju z flagą,
- Przywiązanie pierścienia teleportacji do miejsca, gdzie aktualnie przebywamy (środka pokoju).

Chcesz mieć „Astronomię" w domu?

Nic prostszego – zamów prenumeratę
w naszym sklepie internetowym **www.astronomia.sklep.pl**
lub pod numerem telefonu **515 773 590**

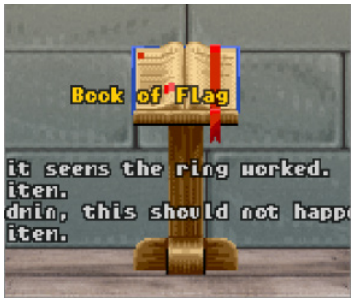
Do nabycia również w sieci Empik, Inmedio oraz Ruch.

Jedyny miesięcznik astronomiczny w Polsce!

» Poczekanie, aż „Ghostly Visions” się zakończą
» Teleportowanie się do środka.

W praktyce wykonanie kilku relatywnie skomplikowanych operacji na touchpadzie autora okazało się trudniejsze niż się wydaje i udało się dopiero za trzecim razem…

Ostatecznie się udało i otrzymaliśmy flagę: **DrgnS{FlagRooms-AreDeathTraps}**. Bardzo prawdziwą – nie pomyśleliśmy, że z pokoju nie ma też jak wyjść[2], i musieliśmy stworzyć nową postać w celu kontynuacji gry.



Rysunek 3. Flaga czekająca na odczytanie z księgi

Zadanie Raid Party, kategoria: błędy logiczne

Ostatnie zadanie, które opiszę, również opierało się na błędzie logicznym. Na pierwszy rzut oka nie było nawet widać, co tu jest zadaniem – postać w grze prosiła nas o zebranie pięciu amuletów, które po prostu leżały na mapie. Haczyk polegał na tym, że po zebraniu czterech amuletów okazało się, że to już wszystkie i więcej nie ma. Wyszło więc na to, że musimy w jakiś sposób wygenerować nowy, albo zduplikować te, które już mamy.

Ciekawą rzeczą, którą zaobserwowaliśmy, było to, że amulety jako jedyne przedmioty miały flagę `blessable` („błogosławialne”). Pozwalało to rzucać na nie czar `rebless`, którego uproszczony kod znajduje się poniżej (Listing 8):

Listing 8. Uproszczony kod zaklęcia „rebłogosławienia”

```
@magic_spell("7071727374757677")
def spell_rebless(spell, player, world):
    for _ in player.YIELDING_select():  # yield from
        yield _

    if player.select_result is None
        player.show_text("Something's not right.")
        return

    target_type, target = player.select_result
    if target_type != "item"
        player.show_text("Something went wrong.")
        return

    item, item_location, item_location_info = target
    if not hasattr(item, "blessable") ornot item.blessable:
        player.show_text("Wrong kind of item")
        return

    item_type = type(item)
    player.show_text("Summoning astral plane beings...")

    for _in world.YIELDING_sleep(1.0):  # yield from
        yield _
    blessing_type = random.choice(["cosmic", "aetheric", ...])
    world.reclaim_item(item)
    item.id = ITEM_NON_EXISTING_ID
    item = world.register_item(item_type(blessing_type))
    player.add_to_inventory(item)
```

Byliśmy prawie pewni, że gdzieś tutaj znajdziemy błąd, który pozwoli nam sklonować przedmioty, więc skupiliśmy się na tej funkcji. W jeszcze większym uproszczeniu można ten kod zapisać jako:

1. Gracz zaznacza coś na ekranie.
2. Jeśli gracz nie wybrał przedmiotu typu „blessable” – kończymy działanie.
3. Zapisujemy wybrany przedmiot i wyświetlamy odpowiednią wiadomość.
4. Czekamy jedną sekundę.
5. Tworzymy nowy przedmiot z odpowiednim typem, dajemy go graczowi oraz usuwamy stary z ekwipunku.

Można zastanowić się chwilę, czy jest tu jakiś problem.

Nam, po jakimś czasie, udało się go w końcu zauważyć. Konkretnie, jeśli gracz rzuci szybko dwa razy zaklęcie na jeden amulet, to sekwencja zdarzeń będzie następująca:

1. Pierwszy czar zapamiętuje sobie amulet w zmiennej `item`.
2. Drugi czar zapamiętuje sobie amulet w zmiennej `item`.
3. Pierwszy czar usuwa amulet `item` z ekwipunku i dodaje nowy.
4. Drugi czar próbuje usunąć amulet, który już został usunięty (więc nic nie zmienia), i dodaje kolejny.

Efektywnie w ten sposób dostajemy dwa nowe amulety z jednego. W celu rzucenia dwóch czarów odpowiednio szybko trzeba było oskryptować grę (albo skorzystać z pomocy kolegi, albo zapisać jeden z nich na zwoju). Ostatecznie jednak się udało i otrzymaliśmy upragniony, piąty amulet, co w efekcie dało nam flagę:
DrgnS{IOweYouFromThatThingWithTheGuyAtThePlace}

ZAKOŃCZENIE

Na tym zakończyła się nasza druga przygoda z grą. Ponownie bardzo nam się podobało, chociaż brakowało nam faktycznie trudnych zadań – wszystkie części gry zostały rozwiązane przez wszystkie topowe zespoły, więc nie dawała ona aż tak dużej przewagi jak rok temu. A szkoda – chętnie stawilibyśmy czoła jednemu lub dwóm wyzwaniom w świecie gry. Tak czy inaczej, liczymy na kolejną edycję za rok!

Jarosław Jedynak

Rozwiązanie zadań zostało nadesłane przez zespół p4, który robi, co może, żeby polskie flagi były wysoko w rankingach drużyn CTFowych.
» https://ctftime.org/team/5152

2. Potrzebne zaklęcia są bardzo kosztowne i nie mieliśmy zapasu potrzebnego na wykorzystanie tej samej sztuczki, żeby wrócić.