Pipelight

Netflix and more via Wine

Michael Müller Sebastian Lackner

February 2, 2014

$ whoami

Michael Müller

Email: michael@fds-team.de

studying computer science at the university of Heidelberg, Germany

Sebastian Lackner

Email: sebastian@fds-team.de

studying physics at the university of Heidelberg, Germany

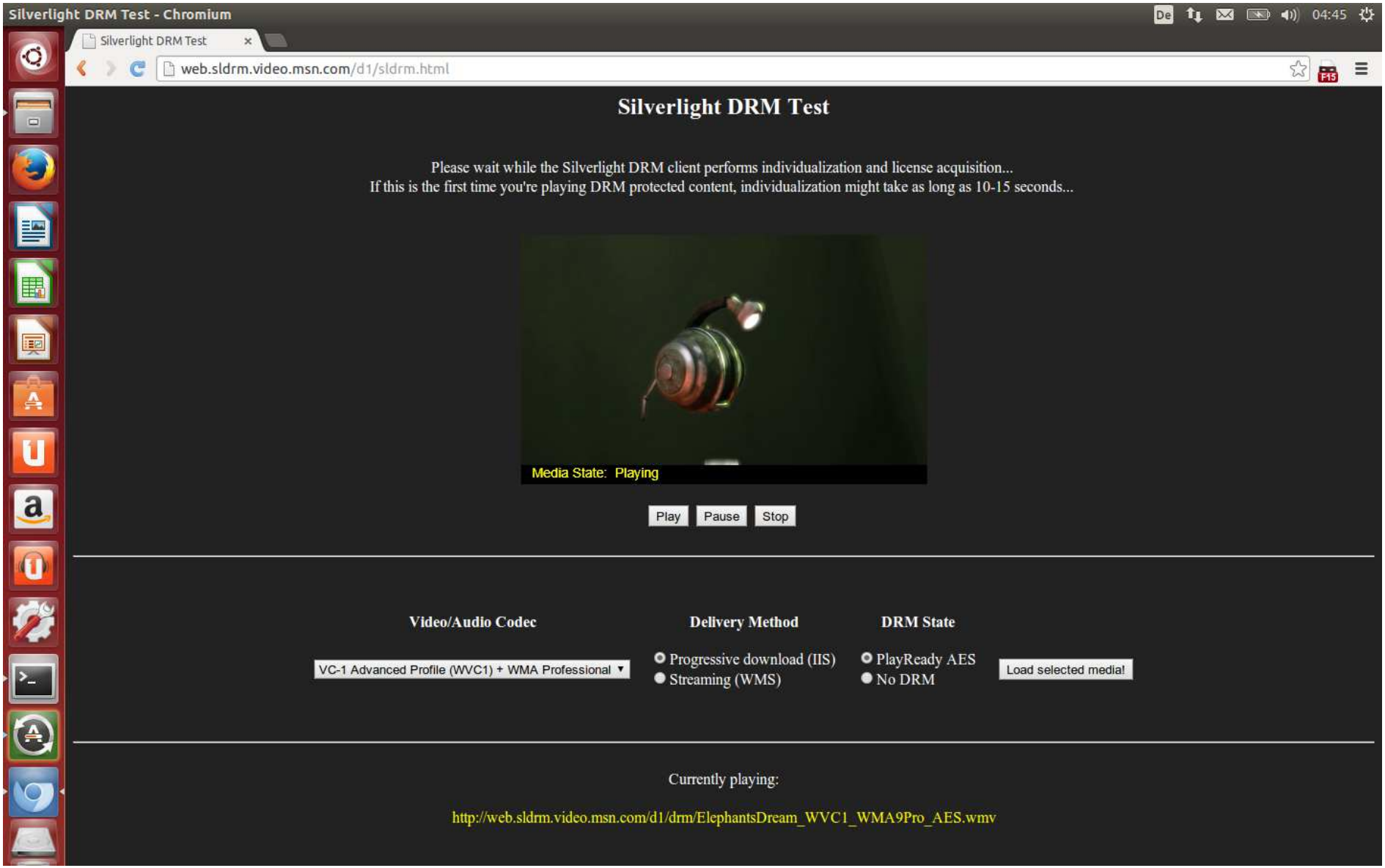

Silverlight DRM Test - Chromium
04:45
Silverlight DRM Test
web.sldrm.video.msn.com/d1/sldrm.html
Silverlight DRM Test
Please wait while the Silverlight DRM client performs individualization and license acquisition...
If this is the first time you're playing DRM protected content, individualization might take as long as 10-15 seconds...
Media State: Playing
Play
Pause
Stop
Video/Audio Codec
Delivery Method
DRM State
VC-1 Advanced Profile (WVC1) + WMA Professional
Progressive download (IIS)
Streaming (WMS)
PlayReady AES
No DRM
Load selected media!
Currently playing:
http://web.sldrm.video.msn.com/d1/drm/ElephantsDream_WVC1_WMA9Pro_AES.wmv

Table of contents

Overview about Pipelight

Idea behind Pipelight

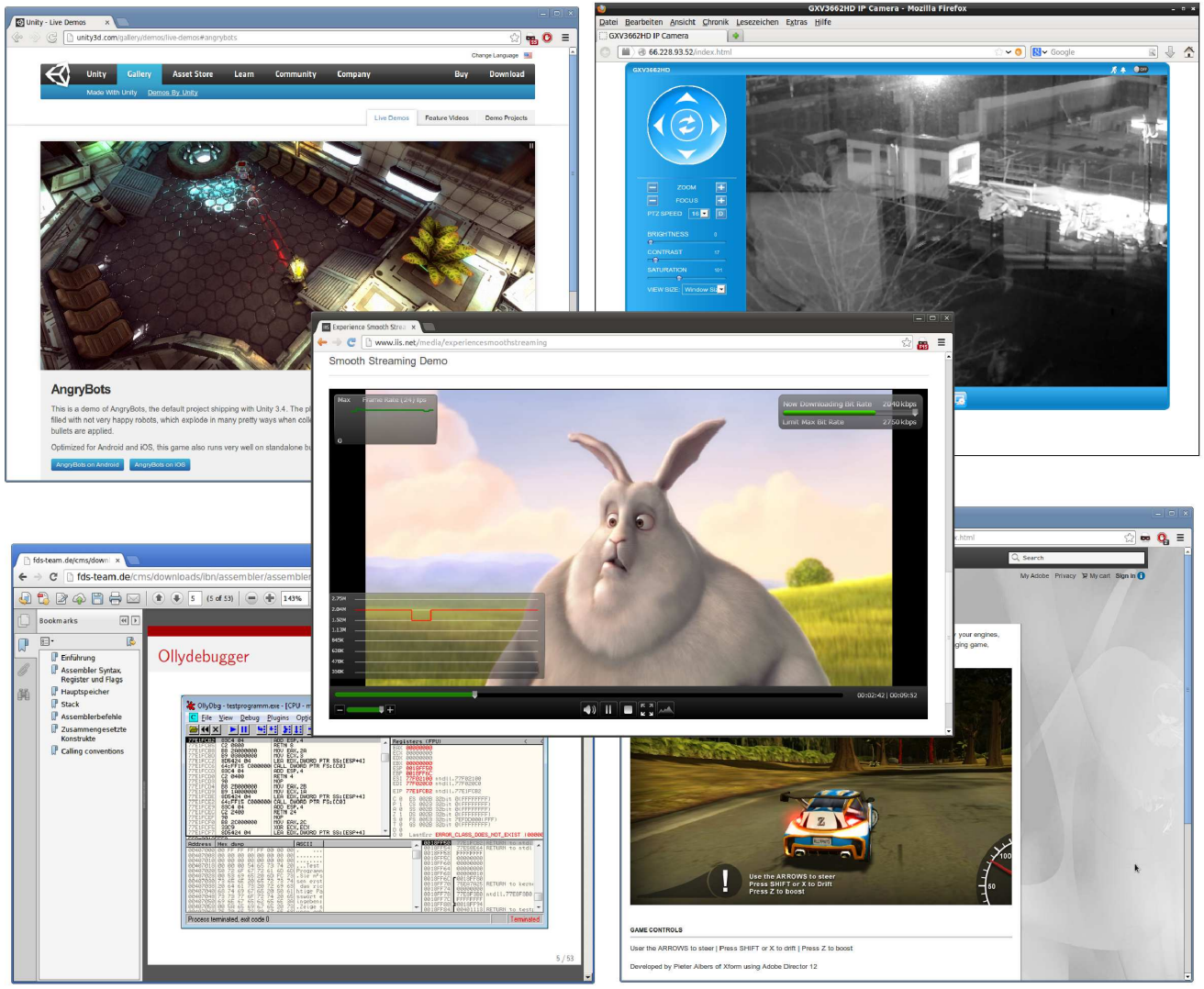
"Pipelight allows one to run Windows browser plugins in the context of Linux browsers"

To achieve this Pipelight ...

- connects the Windows DLL with the Linux process
- translates between platform dependent differences on the NPAPI
- uses a patched wine version
- and all that should be **invisible** and **transparent** for the user

Supported plugins

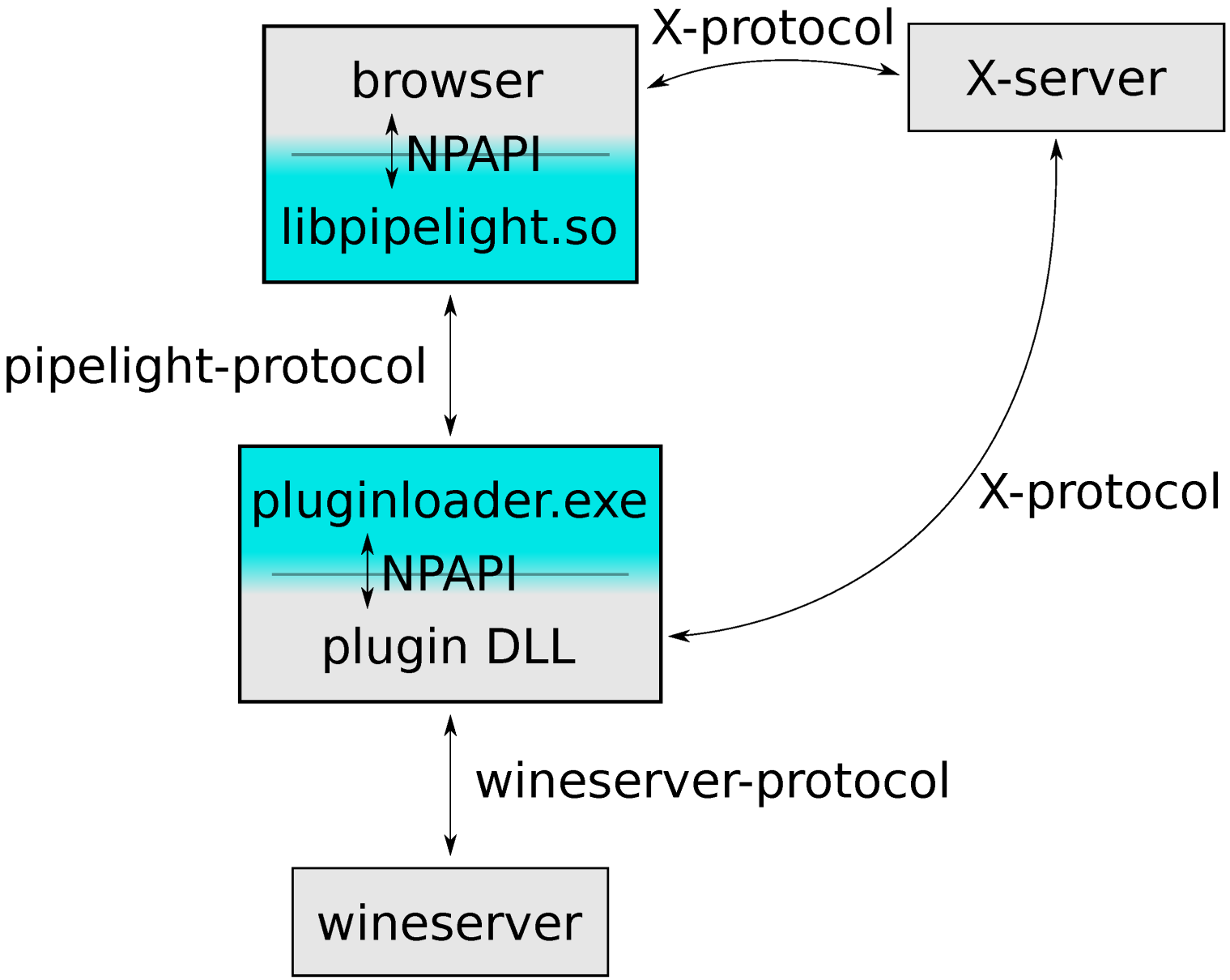
1. Silverlight
2. Flash
3. Widevine
4. Unity3D
5. Shockwave
6. Adobe Reader
7. ...



How does it work?

- It is very complicated (or even impossible) to load DLLs directly into a Linux process
  - → `fs` and `gs` register is used for different purposes, exception handling works completely different, ...
- We therefore splitted up our project in two parts:
  - **linux** shared object ⇒ NPAPI plugin loaded into the browser
  - **win32** `pluginloader.exe` ⇒ executed via Wine, loads the real DLL
- `pluginloader.exe` is started (by the Linux library) when required, afterwards communication via stdin & stdout

Pipelight


X-protocol
browser
NPAPI
libpipelight.so
X-server
pipelight-protocol
pluginloader.exe
NPAPI
plugin DLL
X-protocol
wineserver-protocol
wineserver

Design decisions 1/2

"Isn't this a bit complicated and slow?" No...

- **abstraction:** allows to load 32 bit plugins in 64 bit browser
- **performance:** audio and video is not transmitted via pipe
- **debugging:** very easy to track down errors
  - → `WINEDEBUG` captures only plugin and no browser API calls

"Why stdin & stdout?"

- **portability:** works the same way on all platforms, allows reusing code

Design decisions 2/2

"So Pipelight just acts as a wrapper?" No, Pipelight can also...

- **download:** necessary dependencies (e.g. DLLs missing / broken in Wine) and the plugin itself
- **install:** extract required files
- **configure:** the plugin (e.g. en/disable hardware acceleration)
- **update:** the plugins

→ Pipelight is comparable with CrossOver, PlayOnLinux, ..., but just for browser plugins

Browser API

&

Communication

Browser APIs

- Most Windows plugins support two plugin APIs:

ActiveX

- Internet Explorer only, COM based
- Documentation available (≈839 pages)
- ATL header files are only shipped with MSVC

NPAPI

- Crossbrowser and crossplatform API
- Easy: 4 header files, 19 plugin and 58 browser functions
- Almost all (Linux/Windows/MacOS) browser support this API

$\Rightarrow$ We use NPAPI on both the Windows and Linux side

NPAPI

Netscape Plugin Application Programming Interface (NPAPI)

- Introduced in 1996 and still gets extensions
  (but Chrome is going to drop the support soon ☹)
- The API mainly consists out of
  - Instances → e.g. running plugin instances
  - Objects → mainly used in conjunction with Javascript
  - Identifiers → IDs which correspond to strings or integers
  - Streams → HTTP(s) requests
  - Variants → objects or other basic data types
- To redirect NPAPI we have to send **all** types of objects through pipes

Transmitting data

- How to transmit the data of the function parameters?
  - Integer → just write the bytes into the pipe
  - Strings → calculate length and write it into the pipe
  - Structures → write memory block into the pipe

- What about more complicated objects?

  → Note: passing pointers doesn't make sense!

  → Serializing not possible, internal data is unknown.

⇒ Our solution: creating **fake objects** and use RPC-like function calls

Handle manager

Handle manager

- containing all the logic to transmit datatypes and create fake objects
- The handle manager offers convenience functions for writing and reading such datatypes:

```
void writeHandleObj(NPObject *obj);
void writeHandleIdentifier(NPIdentifier name);
void writeHandleInstance(NPP instance);
void writeHandleStream(NPStream *stream);
void writeHandleNotify(void *notifyData);
....
```

$\Rightarrow$ Simplifies the logic in all the wrapper functions

Example

```
bool NP_LOADDS NPN_HasProperty(NPP instance, NPObject *obj,
                               NPIdentifier propertyName){

        /* write arguments */
        writeHandleIdentifier(propertyName);
        writeHandleObj(obj);
        writeHandleInstance(instance);

        /* issue the command (async) */
        callFunction(FUNCTION_NPN_HAS_PROPERTY);

        /* wait for result, and return it */
        return (bool)readResultInt32();
}
```

Handle manager

Writing objects

- Check if this object pointer is known, if not:
  - Generate an unique ID
  - Add the mapping (ID $\Leftrightarrow$ pointer) into an associative array
- Write (type, ID) into the pipe

Reading objects

- Check if this ID is known, if not:
  - Generate a fake object (allocate memory, ...)
  - Add the mapping (ID $\Leftrightarrow$ pointer) into an associative array
  - Request additional information from the other side (if necessary)
- Return the pointer

Remote function calls 1/2

Calling remote function

caller:

- Write an arbitrary count of parameters on the stack (rev. order)
- Write the ID of the function onto the stack

callee:

- Remote side dispatches the function
- The called function has access to the values on the "stack"
- Afterwards it places one or more return values onto the remote stack

caller:

- The caller waits until all return values are received

Remote function calls 2/2

- We can now redirect NPAPI function calls through the pipe, do we need anything further? Yes ...
- The API is defined for all platforms, but it differs in some details, mainly:
  - Drawing
  - Input events
- The remaining challenge is to integrate seamless with Wine to overcome these differences

Drawing

&

Input events

Platform specific differences

Examples for platform specific differences of NPAPI plugins:

Drawing

- X11 windows ⇔ hWnd
- X11 drawables ⇔ hDC

Input events

- X11 events ⇔ window messages
- NPAPI timers ⇔ window message timers

NPAPI drawing modes

- The standard drawing mode of the NPAPI is the **windowed mode**:
  - browser provides a window handle of an container
  - plugin creates a child window in the container
- Besides that there is also **windowless mode**:
  - browser provides a drawable, plugin doesn't use any window

Windowed mode

Embedding a plugin into the browser

Windows:

- Create a new (invisible) plugin window with Wine
- Get X11 handle of the window

Linux:

- Use XEMBED extension to embed the plugin into the browser

Windows:

- Make the plugin window visible

XEMBED

- XEMBED needs interaction between the embedder and the client:
  - Synchronization of keyboard focus (very important!)
  - Synchronization of tab chain
  - Synchronization of keyboard short cuts
- Wine didn't didn't support any of the above requirements
  - No keyboard input possible in the plugin or browser
  - Not possible to maximize a browser window again
- We implemented keyboard focus synchronization into Wine and patched many deadlocks when using Direct3D in child windows

Example

```
/* Windows: Get X11 handle of the window */
XID x11window = (XID)GetPropA(hWnd,
                              "__wine_x11_whole_window");

...

/* Linux: Embed plugin into the container */
XReparentWindow(display, x11window, parent, 0, 0);

/* required for some very old toolkits */
sendXembedMessage(display, x11window,
                  XEMBED_EMBEDDED_NOTIFY, 0, parent, 0);

/* synchronize focus and show window ... */
```

Windowless mode

- Plugins wants to draw on a hDC, but we only have a X11 drawable
- Easiest solution: Copy data from hDC to X11 drawable
  - → slow and causes additional tearing
- Is there a way to convert a X11 into a hDC?
- **Trick:** Create a normal hDC and use a Wine internal `ExtEscape` command to replace the X11 drawable

Example

```
/* Windows: Create device context handle */
hDC = CreateDC("DISPLAY", NULL, NULL, NULL);

x11drv_escape_set_drawable args;
args.code       = X11DRV_SET_DRAWABLE;
args.drawable   = x11drawable; /* <---- */

/* fill out other arguments ... */

ExtEscape(hDC, X11DRV_ESCAPE, sizeof(args),
          (char *)&args, 0, NULL);

/* draw on hDC as usual ... */
```

Event handling in windowed mode

- Fundamental platform difference:
  - **Windows:** browser calls WndProc callback
  - **Linux:** plugin fetches events directly from the Xserver
- Additional problem: NPAPI is single threaded and we can not use an additional thread for the event processing

Event handling

Linux:

- Create an NPAPI timer (to issue calls in the main thread)

Windows:

- for each timer event call `PeekMessage()` / `DispatchMessage()`

Browser plugins and Wine

Browser plugins and Wine

Most browser plugins do not work out of the box ...

- Silverlight PlayReady DRM needs **Access Control Lists (ACL)**
- Unity3D needs **named pipes in message mode** for it's updater
- Flash checks for a current **driver date**, otherwise Stage3D is disabled
- Shockwave needs to be set to OpenGL mode
- The sandbox of Adobe Reader crashes because of the memory layout

Solutions used so far ...

- Wine patches *(preferred)*
- Change plugin configuration (e.g. disable GPU check in Flash)
- Use API hook if a patch could break other applications

Wine patches

- We created ≥ 50 patches (34 upstream) to fix such problems

Patches

- **Erich E. Hoover** - *ACL patches /*
  Address Change Notification / ...
- **Michael Müller** - *IDirect3DSwapChain9Ex /*
  VMR{7,9}MonitorConfig / ...
- **Sebastian Lackner** - *XEMBED /*
  various race conditions and bug fixes / ...
- **André Hentschel** - *Video Mixing Renderer 7 / ...*

full patch list: http://fds-team.de/cms/pipelight-compile-wine.html

Browser

&

Plugin compatibility

Browser compatibility and bugs

- **Chrom(ium), fixed in Firefox some time ago**:
  - Browser calls notification events for streams that are already destroyed
- **Midori and several others**:
  - no NPAPI timer support
- **Opera** does not support NPAPI timers, but sets the function pointer to a stub function instead of `NULL` ...

Plugin bugs 1/3

- Most browser plugins are crappy, so that Chromium has implemented many workarounds:

```
enum PluginQuirks {
    SETWINDOW_TWICE = 1,                          // Win32
    THROTTLE_WM_USER_PLUS_ONE = 2,                // Win32
    DONT_CALL_WND_PROC_RECURSIVELY = 4,           // Win32
    DONT_SET_NULL_WINDOW_HANDLE_ON_DESTROY = 8,   // Win32
    DONT_ALLOW_MULTIPLE_INSTANCES = 16,           // Win32
    DIE_AFTER_UNLOAD = 32,                        // Win32
    PATCH_SETCURSOR = 64,                         // Win32
    BLOCK_NONSTANDARD_GETURL_REQUESTS = 128,      // Win32
    WINDOWLESS_OFFSET_WINDOW_TO_DRAW = 256,       // Linux
    ...
```

Plugin bugs 2/3

```
    ...
    WINDOWLESS_INVALIDATE_AFTER_SET_WINDOW = 512, // Linux
    NO_WINDOWLESS = 1024,                         // Windows
    PATCH_REGENUMKEYEXW = 2048,                   // Windows
    ALWAYS_NOTIFY_SUCCESS = 4096,                 // Windows
    HANDLE_MOUSE_CAPTURE = 16384,                 // Windows
    WINDOWLESS_NO_RIGHT_CLICK = 32768,            // Linux
    IGNORE_FIRST_SETWINDOW_CALL = 65536,          // Windows
    EMULATE_IME = 131072,                         // Windows
};
```

⇒ Pipelight tries to workaround such problems, but this is not always possible

Plugin bugs 3/3

- Pipelight also fixes some **additional bugs**:
  - Silverlight expects that COM is initialized with `CoInitialize()`
  - Silverlight expects the floating point flags to be set appropriately *(how much time we wasted to find this bug...)*
  - Shockwave sometimes passes `NULL` as instance pointer *(is this a Wine bug?)*

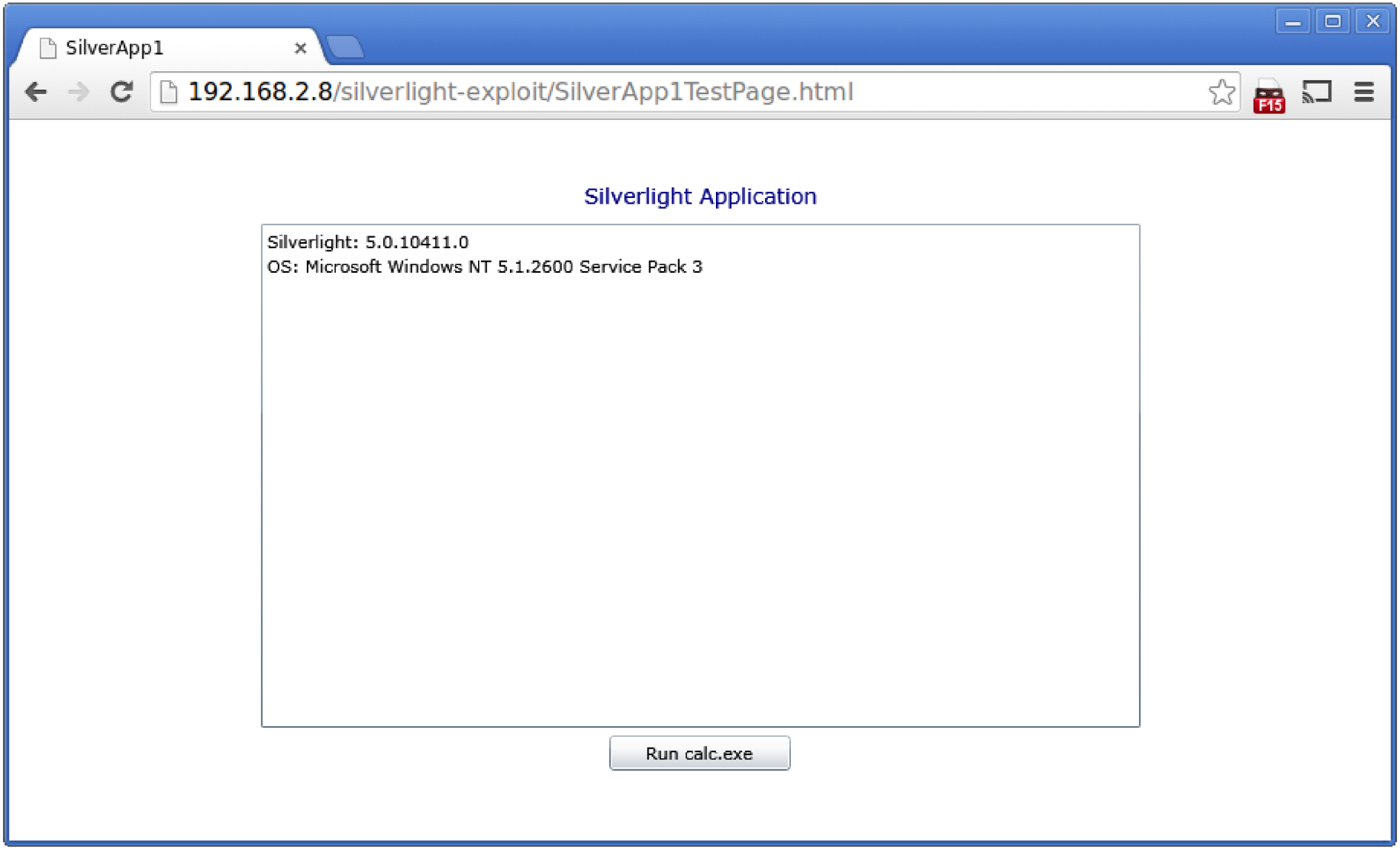
$\Rightarrow$ Pipelight detects such bugs and workarounds them (the plugins are even more compatible than on Windows!)

Security

Security

- With the increasing sandbox security of browsers the plugins get more and more interesting for attackers.
- The most attacked plugins are:
  - Flash
  - Java
  - but also Silverlight gains more interest
- So what about **plugin vulnerabilities** and Pipelight?
  - $\Rightarrow$ Lets take a closer look at a Silverlight exploit published recently

Silverlight Exploit - Screenshot



Silverlight Exploit - Explanation

How does it work?

- Silverlight tries to protect the user by allowing only a subset of the .NET API
- The exploit uses two security vulnerabilities to get around this restriction:
  - A way to read arbitrary memory and calculating the native pointer
  - Passing a native pointer as class constructor to a .NET command
- This allows the exploit to execute its native payload

$\Rightarrow$ What happens if we **try this with Pipelight**?

Silverlight Exploit - Pipelight

```
[PIPELIGHT:WIN:silverlight5.1] ../common/common.c:801:handleManager_ptrToId(): got non-existent pointer.
fixme:advapi:UnregisterTraceGuids 0: stub
[PIPELIGHT:LIN:silverlight5.1] ../common/common.c:183:receiveCommand(): unable to receive data.
```

- Pipelight hits an internal assertion and aborts. Why?

```
public class MyObject:System.Windows.Browser.HtmlObject{
    public void Init(IntPtr handle){
        // call agcore.dll DOM_ReferenceObject()
        Initialize(handle, (IntPtr)1, true, false);
    }
}
```

- The exploit references a self created HTML object
- Pipelight detects the invalid pointer and terminates the plugin

$\Rightarrow$ This was just luck, are there any further security layers?

Security layers

Protection offered by the separate components

- **Pipelight**
  - some mistakes are caught by assertions
- **Plugins**
  - often some kind of “sandbox”
  - can we trust the authors that there is no backdoor?
- **Wine**
  - Wine doesn’t provide any protection, it just translates API calls

$\Rightarrow$ Can we somehow increase the security?

Security - Ideas

- Some ideas on how to increase the security:
  - scanning for malicious code or viruses
  - removing `Z:` drive which points to "/"
  - check access restrictions on each API call
- It is easy to get around these checks:
  - a program could directly execute syscalls
  - or even generate opcodes at runtime

$\Rightarrow$ We worked on our own security system for plugins

Pipelight-Sandbox (beta) approach

- Pipelight-Sandbox runs plugins in a secure way using namespaces:
  - **PID namespace** - Other processes not visible
  - **Mount namespace** - Filesystem is readonly (except `WINEPREFIX`)
  - **IPC namespace** - Other sockets are not accessible
  - **Network namespace** - Restricted network access (i.e. blocked 192.168.*, 10.*, ...)
- Should protect against any kind of manipulation
- The sandbox is not only usable with Pipelight

Pipelight-Sandbox (beta) approach

- Pipelight-Sandbox can run any linux program and is highly configurable:
  - Allow X server access?
  - Allow Pulseaudio access?
  - Allow network access?
  - Define writeable directories
- works especially good with Wine as the number of writeable directories is small
- Some issues are still left:
  - allowing network access makes it possible to steal information
  - everything still **beta**, so use at your own risk!

Conclusion

- Now we've seen various aspects of Pipelight:
  - → how the communication of the Linux ⇔ Windows parts work
  - → transparent integration via XEMBED, ...
  - → methods to improve the compatibility with browsers/plugins
  - → how to enhance the security
- But we're not finished yet:
  - → what about alternative plugin interfaces like PPAPI and ActiveX?
  - → do we have similar options for Encrypted Media Extensions?
  - → similar concepts could also be useful for other programs
  - → *... and a lot more ideas!*

Questions?

Contact us

- Contact us:
  - **Mail:** michael@fds-team.de
    sebastian@fds-team.de
  - **IRC:** #pipelight on freenode

- Find out more about Pipelight:
  - https://launchpad.net/pipelight
  - http://fds-team.de

- Sourcecode:
  - https://bitbucket.org/mmueller2012/pipelight
  - https://bitbucket.org/mmueller2012/pipelight-sandbox
  - **Contributions are welcome!**