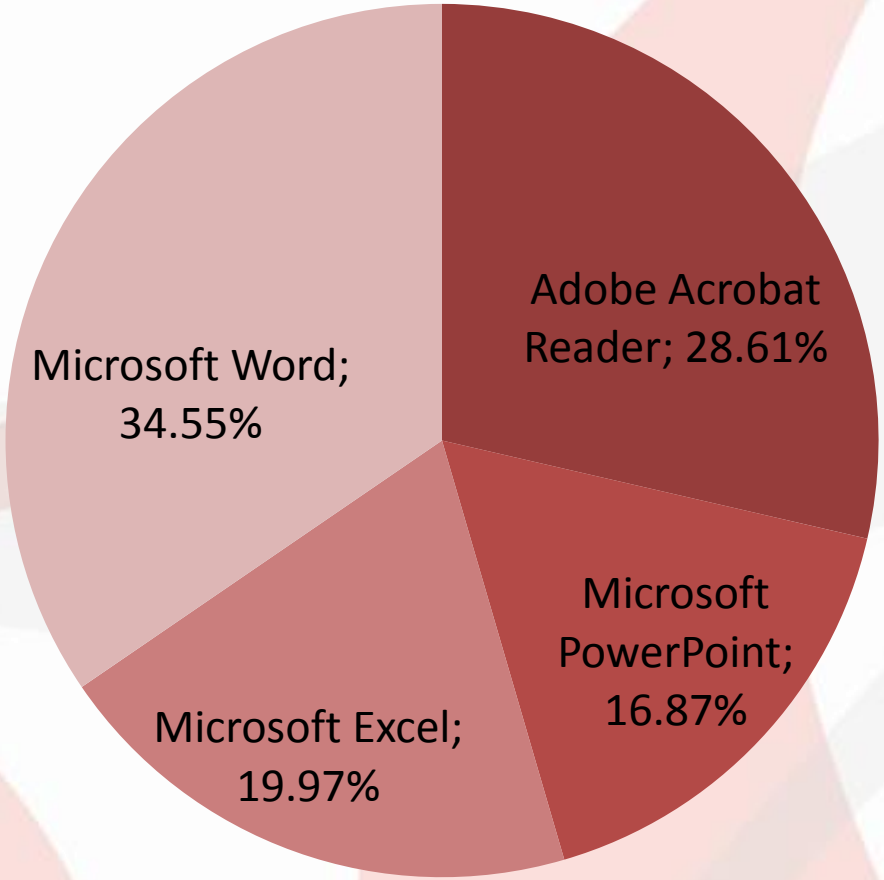
How to really obfuscate your PDF malware

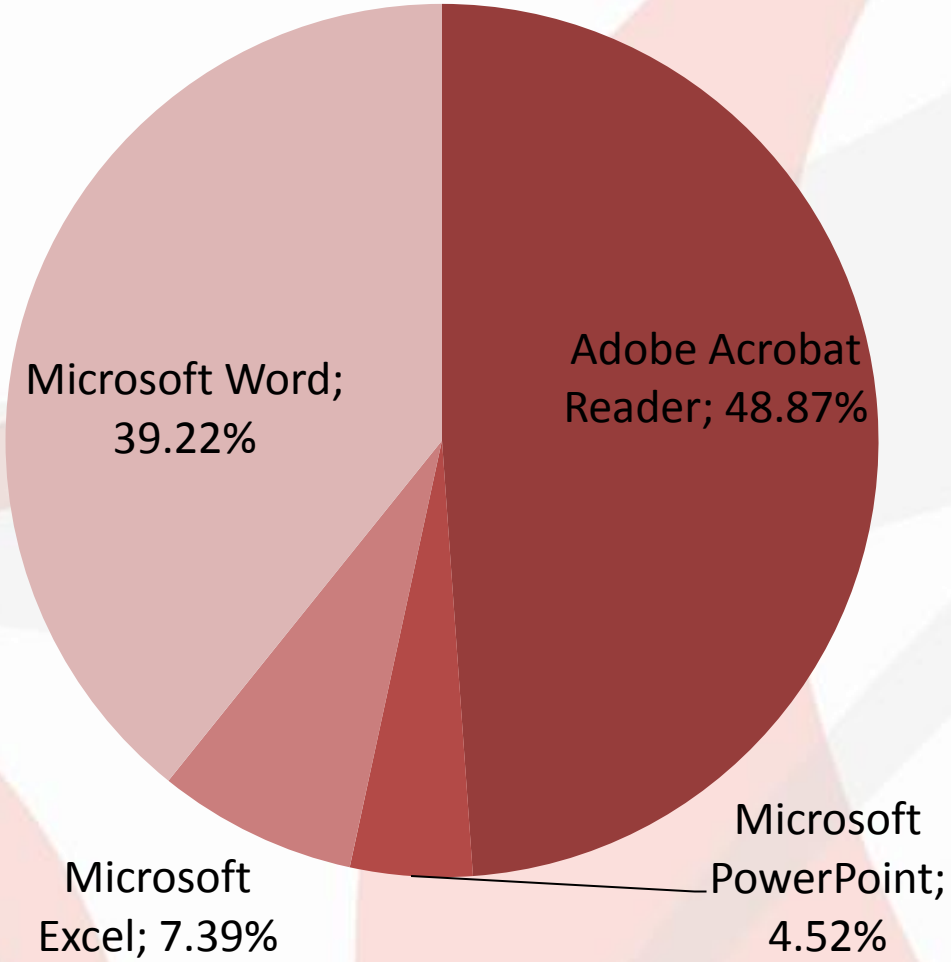
Sebastian Porst - ReCon 2010
Email: sebastian.porst@zynamics.com
Twitter: @LambdaCube

Targeted Attacks 2008

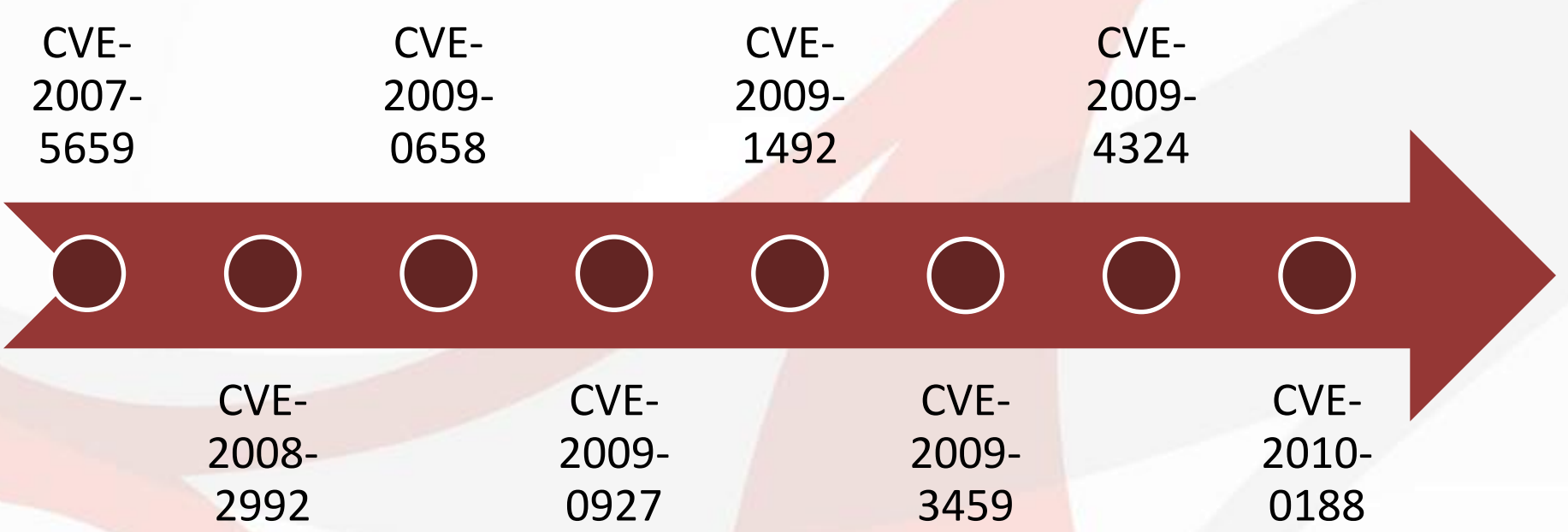


http://www.f-secure.com/weblog/archives/00001676.html

Targeted Attacks 2009



Exploited in the wild



Four common exploit paths

- Broken PDF Parser
- Vulnerable JavaScript Engine
- Vulnerable external libraries
- /Launch

PDF Malware Obfuscation

Different tricks for different purposes

- Make manual analysis more difficult
- Resist automated analysis
- Avoid detection by virus scanners

PDF Malware Obfuscation

Conflicting goals

How to achieve these goals

Being harmless

- Avoid JavaScript
- Do not use unusual encodings
- Do not try to break parser-based tools
- Ideally use an 0-day

Being evil

- Use heavy obfuscation
- Try to break tools

Let‘s be evil

Breaking tools

Rule #1: Do the unexpected

This is what tools expect

- ASCII Strings
- Boring encodings like #41 instead of A
- Wellformed or only moderately malformed PDF file structure

Malformed documents

- Adobe Reader tries to load malformed PDF files
- Very, very liberal interpretation of the PDF specification
- Parser-based analysis tools need to know about Adobe Reader file correction

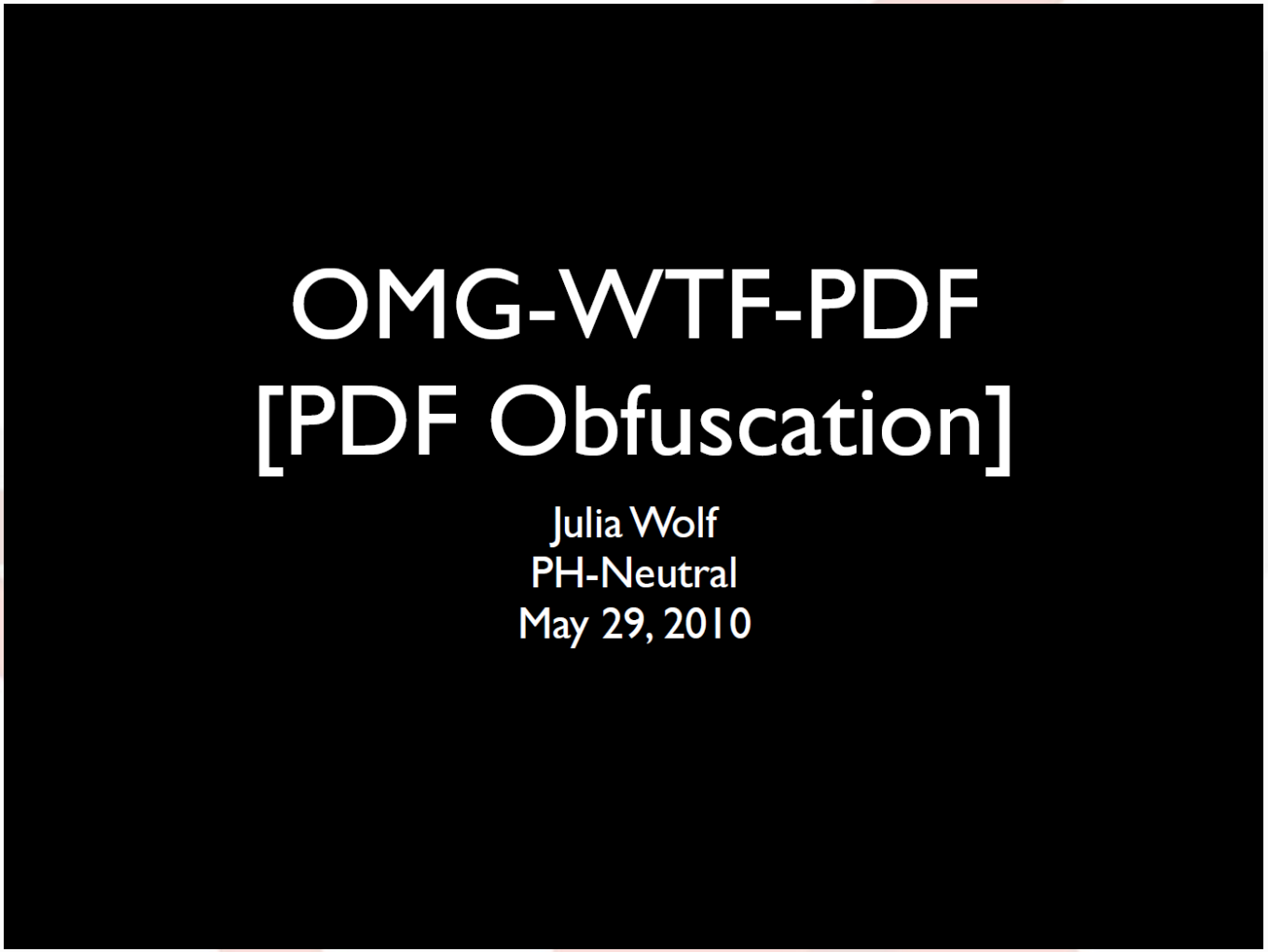
Malformed PDF file – Example I

```
7 0 obj
  <<
    /Type /Action
    /S    /JavaScript
    /JS   (app.alert('whatever');)
  >>
endobj
```

Malformed PDF file – Example II

```
5 0 obj
  << /Length 45 >>
  stream
    some data
  endstream
endobj
```

Further reading



Obfuscating JavaScript code

Goal of JavaScript obfuscation

Hide the shellcode

JavaScript obfuscation in the wild

- Screwed up formatting
- Name obfuscation
- Eval-chains
- Splitting JavaScript code
- Simple anti-emulation techniques
- callee-trick
- …

Screwed up formatting

- Basically just remove all newlines
- Completely useless: jsbeautifier.org

Name obfuscation

- Variables or function names are renamed to hide their meaning
- Most JavaScript obfuscators screw this up

Obfuscation example: Original code

```
function executePayload(payload, delay)
{
  if (delay > 1000)
  {
    // Whatever
  }
}

function heapSpray(code, repeat)
{
  for (i=0;i<repeat;i++)
  {
    code = code + code;
  }
}
```

Obfuscation without considering scope

```
function executePayload(hkof3ewhoife, fhpfewhpofe)
{
  if (fhpfewhpofe > 1000)
  {
    // Whatever
  }
}

function heapSpray(hoprwehjoprew, hoifwep43)
{
  for (jnpfw93=0;jnpfw93<hoifwep43;jnpfw93++)
  {
    hoprwehjoprew = hoprwehjoprew + hoprwehjoprew;
  }
}
```

Obfuscation with considering scope

```
function executePayload(grterttttrr, hnpfefwefee)
{
  if (hnpfefwefee > 1000)
  {
    // Whatever
  }
}

function heapSpray(grterttttrr, hnpfefwefee)
{
  for (hjnprew=0;hjnprew<hnpfefwefee;hjnprew++)
  {
    grterttttrr = grterttttrr + grterttttrr;
  }
}
```

Obfuscation: Going the whole way

```
function ____(____, _____)
{
  if (_____ > 1000)
  {
    // Whatever
  }
}

function _____(____, _____)
{
  for (_____=0; _____<____; _____++)
  {
    ____ = ____ + ____;
  }
}
```

Name obfuscation: Lessons learned

- Consider name scope
  - Deobfuscator needs to know scoping rules too
- Use underscores
  - Drives human analysts crazy
- Also cute: Use meaningful names that have nothing to do with the variable
  - Maybe shuffle real variable names

Eval chains

- JavaScript code can execute JavaScript code in strings through eval
- Often used to hide later code stages which are decrypted on the fly
- Common way to extract argument: replace eval with a printing function

Eval chains: Doing it better

- Make sure your later stages reference variables or functions from earlier stages
- Re-use individual eval statements multiple times to make sure eval calls can not just be replaced

JavaScript splitting

- JavaScript can be split over several PDF objects
- These scripts can be executed consecutively
- Context is preserved between scripts
- In the wild I‘ve seen splitting across 2-4 objects

JavaScript splitting: Doing it better

- One line of JavaScript per object
- Randomize the order of JavaScript objects
- Admittedly it takes only one script to sort and extract the scripts from the objects

Anti-emulation code

- Simple checks for Adobe Reader extensions
- Multistaged JavaScript code

Current malware loads code from

- Pages
- Annotations
- Info Dictionary

Example: Loading code from annotations

```
y = app.doc;

y.syncAnnotScan();

var p = y["getAnnots"]({nPage: 0});

var s = p[0].subject;

eval(s);
```

Problems with current approaches

Code is in the file

Easy to extract

Anti-emulation code: Improved

Key ideas behind anti-emulation code

Find idiosyncrasies in the Adobe JavaScript engine

Find extensions that are difficult to emulate

Exhibit A: Idiosyncrasy

```
cypher = [7, 17, 28, 93, 4, 10, 4, 30, 7, 77, 83, 72];
cypherLength = cypher.length;

hidden = "ThisIsNotTheKeyYouAreLookingFor";
hiddenLength = hidden.toString().length;

for(i=0,j=0;i<cypherLength;i++,j++)
{
  cypherChar = cypher[i];
  keyChar = hidden.toString().charCodeAt(j);
  cypher[i] = String.fromCharCode(cypherChar ^ keyChar);

  if (j == hiddenLength - 1)
    j = -1;
}

eval(cypher.join(""));
```

Exhibit A: Explained

JavaScript Standard

```
hidden = false;
hidden = "Key";
```

hidden has the value „Key“

Adobe Reader JavaScript

```
hidden = false;
hidden = "Key";
```

hidden has the value „true“

Exhibit A: Explained

The Adobe Reader JavaScript engine defines global variables that do not change their type on assignment.

(I suspect this happens because they are backed by C++ code)

Exhibit B: Difficult to emulate

- Goal: Find Adobe JavaScript API functions which are nearly impossible to emulate
- Then use effects of these functions in sneaky ways to change malware behavior
- The Adobe Reader JavaScript documentation is your friend

Exhibit B: Difficult to emulate

Functions to look for

- Rendering engine
- Forms extensions
- Multimedia extensions

Exhibit B: Difficult to emulate

```
crypt = "T^_]^[T IEYYD__ FuRRKBD ";
plain = Array();
key = getPageNthWordQuads(0, 0).toString().split(",")[1];

for (i=0,j=0;i<crypt.length;i++,j++)
{
   plain = plain + String.fromCharCode((crypt.charCodeAt(i) ^
key.charCodeAt(j)));

   if (j >= key.length)
       j = 0;
}

app.alert(plain);
)
```

Exhibit B: Difficult to emulate

Functions to avoid

Anything with security restrictions

Exhibit C: Multi-threaded JavaScript

- Multi-threaded applications are difficult to reverse engineer
- Problem: There are no threads in JavaScript
- Solution: setTimeOut
- Example: Cooperative multi-threading with message-passing between objects

Basic idea

- Multiple server objects
- String messages are passed between servers
- Messages contain new timeout value and code to evaluate

```
function Server(name)
{
   ...
}

s1 = new Server("S1");
s2 = new Server("S2");

s1.receive(ENCODED_MESSAGE);
```

```
function Server(name)
{
  this.name = name;

  this.receive = function(message)
  {
    recipient = parse_recipient(message)
    delayTime = parse_delay(message)
    eval_string = parse_eval_string(message)
    msg_string = parse_message_string(message)

    eval(eval_string);
    command = "recipient.receive('" + msg_string + "')";
    this.x = app.setTimeOut(command, delayTime);
}
};
```

How to improve this

- Use a global string object as the message queue and manipulate the object on the fly
- Usage of non-commutative operations so that execution order really matters
- Message broadcasting
- Add anti-emulation code to eval-ed code

callee-trick

- Not specific to Adobe Reader
- Frequently used by JavaScript code in other contexts
- Function accesses its own source and uses it as a key to decrypt code or data
- Add a single whitespace and decryption fails

callee-trick Example

```
function decrypt(cypher)
{
  var key = arguments.callee.toString();

  for (var i = 0; i < cypher.length; i++)
  {
    plain = key.charCodeAt(i) ^ cypher.charCodeAt(i);
  }

  ...
}
```

More ideas for the future

- Combine anti-debugging, callee-trick, and message passing
- Find more JavaScript engine idiosyncracies: Sputnik JavaScript test suite

Thanks

- Didier Stevens
- Julia Wolf
- Peter Silberman
- Bruce Dang