Alternative Exploitation Vectors

A study of CVE-2010-3333[1]

By George Nicolaou

Presentation Objectives

- Share some Experience with Reverse Engineering Driven Exploitation
- Challenge what you know about
  - Software Exploitation
  - CVE-2010-3333 (Microsoft Office RTF Buffer Overflow Vulnerability)

Presentation Outline

- History of CVE-2010-3333
- What was published
- What was not published
- Overview of the RTF vulnerability
- Typical Exploitation Scenario (Live Demo)
- Alternate Vectors: Rewriting Exploit for Windows XP/Vista/7 (Live Demo)
- Conclusions

History of CVE-2010-3333

- Discovered by team509 working with VeriSign iDefence Labs[2].
- Disclosure Timeline
  - August 12, 2009 Initial Vendor Notification
  - August 12, 2009 Initial Vendor Reply
  - November 9, 2010 Coordinated Public Disclosure
- Vulnerabilities in RTF File format was a known fact

History of CVE-2010-3333

- Rumour has it that CVE-2010-3333 was discovered in public (Unverified)
- Vulnerability was patched by Microsoft prior to disclosure[3]
  - Microsoft Office 2003 (KB2289187)
  - Microsoft Office 2007 (KB2289158)
  - etc

What was published

- SecurityFocus clamed that all program versions from Microsoft Office are vulnerable[4]
  - In Microsoft Windows 98
  - Up to Microsoft Windows XP
- No reference of the vulnerability affecting Windows Vista or 7.

What was published

- An example document[6]
- A MetaSploit Module[5]
  - SEH Overflow Based
  - With Windows Vista/7 offsets only if ASLR and SafeSEH was disabled:

> "# In order to exploit this bug on Office 2007, a SafeSEH bypass method is needed."

What was not published

- The vulnerability can be turned into a typical Buffer Overflow without passing through Structured Exception Handling (SEH)
- The vulnerability is indeed exploitable on all Windows Operating Systems.

Overview of the RTF vulnerability

- Basic Shape Object Format[7]

The basic format for drawing objects in RTF is as follows:

```
{ \shp   ..... { \*\shpinst
                      { \spp  { \sn ..... }  { \sp ..... }  }
                  }
                  { \shprslt   ..... }
}
```

Overview of the RTF vulnerability

- Drawing Object Properties

The bulk of a drawing object is defined as a series of properties. The { \shp ..... control word is followed by { *\shpinst Following the { *\shpinst is a list of all the properties of a shape. Each of the properties is in the following format:

{ \sp { \sn PropertyName } { \sv PropertyValueInformation } }

The control word for the drawing object property is **\sp.** Each property has a pair of name (**\sn**) and value (**\sv**) control words placed in the shape property group. For example, the vertical flip property is represented as:
{**\sp**{**\sn** fFlipV}{**\sv** 1}}

Overview of the RTF vulnerability

- Drawing Object Properties

Property	Meaning	Type of value	Default
pFragments	Array	Fragments are optional, additional parts to the shape. They allow the shape to contain multiple paths and parts. This property lists the fragments of the shape.	NULL

Overview of the RTF vulnerability

- Drawing Object Properties

Arrays are formatted as a sequence of numbers separated by semicolons. The first number tells the size of each element in the array in bytes. The number of bytes per element may be 2, 4, or 8. When the size of the element is 8, each element is represented as a group of two numbers. The second number tells the number of elements in the array. For example, the points of a square polygon are written as:

{**sv** 8;4;{0,0};{100,0};{100,100};{0,100}}

\sv Destination for a drawing property value

Overview of the RTF vulnerability

- Drawing Object Properties

  …With this structure it is very simple to dispatch an RTF control word. Once the reader isolates the RTF control word and its (possibly associated) value, the reader then searches an array of SYM structures to find the RTF control word. If the control word is not found, the RTF reader ignores it, unless the previous control was *****, in which case the reader must scan past an entire group…

Overview of the RTF vulnerability

- Typical File format

```
{\rtf1
    {\shp
        {\sp
            {\sn pFragments
                {\sv x;y;[control_word1][control_word2][control_word3]
                    [array]
                }
            }
        }
    }
}
```

Overview of the RTF vulnerability

- Typical File format
  - x Must not be equal to 2, 4, 8
  - y can be any decimal number
  - [control_word1] cannot be zero
  - [control_word2] cannot be greater than [control_word1]
  - [control_word3] is the size of the array (which we control)
  - [array] is the actual array

Typical Exploitation Scenario (Live Demo)

Live Demo

Alternate Vectors: Rewriting Exploit for Windows XP/Vista/7 (Live Demo)

Live Demo

Conclusions

- Don’t believe everything your told about software vulnerabilities (obvious)
- Assembly and Reverse Engineering knowledge is a MUST in software exploitation

Questions?

Questions?

References

1. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3333
2. http://labs.idefense.com/intelligence/vulnerabilities/display.php?id=880 (Try Google Cache)
3. http://technet.microsoft.com/en-us/security/bulletin/MS10-087
4. http://www.securityfocus.com/bid/44652/info
5. http://downloads.securityfocus.com/vulnerabilities/exploits/44652.rb
6. http://www.exploit-db.com/sploits/cve-2011-3333_exploit.doc
7. Rich Text Format (RTF) Specification, Microsoft Technical Support, 2001 – Word 2002 RTF Specification