AKBuilder – the crowdsourced exploit kit

Document exploitation remains a favorite attack technique for distributing malicious content because it is easier to trick victims into opening document attachments than executables.

Exploited documents have the added benefit of not requiring victims to manually enable macros, as is often the case for VBA downloaders.

As of this writing in late 2016, the most popular of the Office exploit builders is a tool called AKBuilder.

This paper delves into the evolution, internal working and end effects of this toolkit.

Gabor Szappanos
Principal Malware Researcher
Sophos
Table of Contents

Introduction ......................................................................................................................... 3
Characteristics ...................................................................................................................... 4
  AK-1 ................................................................................................................................. 6
  AK-2 ................................................................................................................................. 12
Attribution ............................................................................................................................ 16
Conclusion ............................................................................................................................ 17
References ............................................................................................................................ 17
AKBuilder

Introduction

In an earlier research paper [1] we examined the use of Office exploits in distributing malware and concluded the majority of the cases could be attributed to three exploit builders that are very popular among cybercrime groups.

The first two -- Microsoft Word Intruder [2] and the Ancalog Builder [3] -- were already detailed in earlier SophosLabs research papers. This paper covers the third one, AKBuilder.

\[
\begin{array}{c|c}
\text{AKBuilder} & \text{CVE-2014-6352} \\
\text{Ancalog-1} & 36\% \\
\text{Ancalog-2} & 10\% \\
\text{AK-2} & 16\% \\
\text{MWI} & 14\% \\
\text{Other} & 4\% \\
\end{array}
\]

2015Q4 incident stats of the popular Office exploit builders

Unlike MWI or Ancalog, there is no catchy market name associated with this builder. The author(s) did not associate a unique brand name with it (unless you consider “FUD Office 2007/2010 Silent Exploit Builder” as a potential brand name). We decided to call it AKBuilder
because earlier versions of it avoided execution in the presence of Kaspersky Antivirus products.

AKBuilder is advertised in Youtube videos and sold on underground forums. The price of the kit is usually around 550 USD (payable in electronic currencies like Bitcoin and Perfect Money):

![Screenshot of an exploit document](image)

**Characteristics**

The exploit builder generates malicious Word documents, all of them in Rich Text Format.

There are two variations of the kit, which are differentiated by the Office vulnerabilities they are targeting. The earlier version (we denote it AK-1 throughout this paper) uses two exploits in the same document: CVE-2012-0158 and CVE-2014-1761. The newer version (denoted as AK-2) uses a single exploit: CVE-2015-1641.

Both versions of the builder are released as a Python script. Everything is hardcoded in the Python script and there is no configurable option apart from the file names.

The script takes three parameters. The first parameter is the name of the payload file, the second is the name of the decoy document, and the final is the name of the generated exploit document:

```
Usage: builder.py malware.exe decoy_document.doc Exploit_Sample.doc
```

---

*AKBuilder – the crowd-sourced exploit kit*
All of the known builders have the same rough structure. The hardcoded exploit block with first- and second-stage shellcodes are stored as a huge data block in the script:

```python
def Generate_Random_Bytes(x):
    rand_bytes = ""
    for i in range(0,x):
        byte = randint(0,255)
        rand_bytes += chr(byte)
    return rand_bytes
```

The encrypted payload and decoy files are appended after the template content. This is a very rigid structure, an update to make any modification. The beginnings of the generated files up to the embedded payload are identical.

This can only serve as the “release build” of the builder. The script contains the entire document as a single block of data. This block is often modified by the author in order to avoid the detection by antivirus engines. The modifications, though they could be done manually, are more likely done by an internal tool owned (and not released) by the author. This internal tool generates the highly obfuscated exploit document, which is then packaged in the Python script.

The only element of “randomness” in the generator comes at the end of the files, where (this may change with different releases, but the concept is the same) a few hundred random bytes and zeros are appended by the code snippet

```
Generate_Random_Bytes(0x200) + "\x00" * 0x100
```

Here, the random content generator is the following very minimalistic function:

The kit is used by various cybercrime groups, distributing dozens of different malware families. The most active (or least careful) of these criminals are Nigerian BEC groups, as indicated by the submissions of one of the versions of the kit to Virustotal:
AK-1

The first sample we think we saw generated by this kit was the document with the following indicators:

SHA1: 287545adf2a83af5bb3858f0f476d4fdf4e21304
Original name: Secret meeting.doc
First seen: 2015-01-29 17:37:45 UTC

From the first appearance of the kit we could observe a steady flow in the number of new generated samples as shown on the following chart:

![Number of new samples generated by AK-1](chart.png)

The use of AK-1 was most active between the middle of 2015 and 2016. The emergence of its successor AK-2 seemed to spell the end of the kit’s lifespan. By the summer of 2016, it seemed extinct.

But we recently started to observe a resurrection of AK-1 samples. It is probably too early to speculate, but we think it can be associated with the disappearance of the Ancalog builder: there was a significant market gap that needed the older Office exploits provided by AK-1, and when there is a need, there is a solution.
Characteristics

The document structure of the typical AK-1-generated samples is illustrated in the following schematics.

Structure of the AK-1 generated documents

AK-1 generates RTF documents that usually contain two exploits, CVE-2012-0158 and CVE-2014-1761. There are only a handful of exceptions where the second exploit is missing, like the sample with SHA1: 836109e392d14af113bf14aa70bcb3d660f80640 from the m_project report by Quickheal [6].

The malicious documents begin with the exploit blocks; the payload and decoy content are stored at the end of the documents as appended binary data. The samples either embed the executable payload directly or download and run the payload from an external website using a hardcoded URL.

Each of the exploit blocks tries to trigger a particular Word vulnerability in order to run a shellcode. Again each exploit includes its own shellcode but they are essentially the same: they find and run the second stage shellcode.

The CVE-2014-1761 block has two additional NOP-loaded leveltext structures (for details about this exploit refer to [7]). This block has no functionality at all, not needed in the exploitation process, but serves as a good identifier for AK-1 generated documents.

(-48831 is stored as 0x4141, these values fill the memory image of the leveltext structures with the character ‘A’)

(levelspace22873\*\plevelindent23130)
\pleveltext'ff\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831 ?\u-48831

(levelspace22873\*\plevelindent23130)
The shellcode starts by enumerating the open file handles to find the carrier RTF file that contains the second stage of shellcode. It identifies the start of the shellcode by a 20 bytes long signature. 12 of these bytes are string markers, the remaining 8 bytes are the start of the second stage shellcode. If the first two marker strings don’t match, it keeps looking for further files. If they do match, but further checks fail, the execution aborts to a hardcoded memory address. This is usually within the resource section of msores.dll which is not executable, causing Word to crash.

```
01 20 70 5C 72 74
2D 46
05 00 00 00
    cmp    DWORD PTR [eax], 'tr\c'
    jz     short loc_192
    add    eax, 8000h

no_match:
    inc    eax
    cmp    DWORD PTR [eax], 'j-/\'; marker 1
    jnz    short no_match
    cmp    DWORD PTR [eax+4], 't..0'; marker 2
    jnz    short no_match
    cmp    DWORD PTR [eax+8Bh], 8974274Ah; second stage bytes
    jz     short abort
    cmp    DWORD PTR [eax+8Eh], 0499B509Ah; second stage bytes
    jz     short abort
    cmp    DWORD PTR [eax+4], '0\c'; marker 3
    jz     short abort
    mov    esi, eax
    lea    edx, [esi+100h]
    mov    eax, edx, edi
    add    eax, 8
    mov    ecx, 5000h
    rep movsb
    jmp    eax

abort:
    ; CODE XREF: seg0000:00001081j
    ; seg0000:00001071j

00 10 02 48 01
FF E0
    mov    eax, 74A6218h
    jmp    eax
```

Searching for markers to locate the second stage

While the first stage code is the same in both variations, the difference in the second stage code differentiates between the downloaders and the droppers.

**Downloaders**

The downloader shellcode is in the appended binary data utilizing a couple of redirection layers intended to complicate analysis.

The shellcode performs a couple of AV evasion steps: firstly checking the Process Environment Block to see if the process is being debugged. If this is the case, the execution takes an alternate route, where a different payload is downloaded.

Assuming the code isn’t being debugged the shellcode proceeds to calculate a checksum over itself to ensure it has not been altered (e.g. breakpoints were not inserted in a debugger). If the shellcode has been modified, it proceeds to abort the execution. The same happens if anything other than the winword.exe process runs the shellcode (in case of debugging the parent process would be different). These countermeasures aim to avoid reverse engineering of the code.

The carrier document is recognized by a series of sanity checks. The shellcode searches for byte sequences from each of its main constituents:
• ‘a401FfE0’ from the CVE-2012-0158 exploit block
• ‘ecount25’ from the CVE-2014-1761 exploit block
• ‘6%m*<;+`, 0x2057e854 and ‘]/=j0].!’ from different parts of the second stage code

Another round of self-check

This self-check is redundant, as the carrier document was already located by the first stage shellcode and there would be no need to do it again.

Once the shellcode has done as much as it could to avoid reversing, it searches for Kaspersky Antivirus installation. First it checks to see if the **office_antivirus.dll** library (a component of KAV) is loaded and if it is the case, takes a different execution path, and the real payload is not downloaded. This alternative path is also taken if any of the following registry keys is detected:

```
SOFTWARE\KasperskyLab\protected\avp11\Data
SOFTWARE\KasperskyLab\protected\AVP12\Data
SOFTWARE\KasperskyLab\protected\AVP13\Data
SOFTWARE\KasperskyLab\protected\AVP14.0.0\Data
SOFTWARE\KasperskyLab\AVP15.0.0\Data
SOFTWARE\KasperskyLab\AVP15.0.1\Data
SOFTWARE\KasperskyLab\AVP15.0.2\Data
```

This AV detection code is only present in the early variants of the files generated by the builder; later version dropped this piece.

Both the normal and alternative code paths download and execute a program from a hardcoded URL. The only difference is in the URL itself. This feature was likely intended to provide different, even innocent content, when an antivirus solution is installed. Unfortunately, we have not been able to observe the alternative URLs as the analyzed samples all contained incomprehensible random bytes for the download URL.
**Droppers**

The first dropper samples appeared a few months after the downloaders, around the end of April 2015, and they essentially have the same code avoiding Kaspersky Antivirus.

After the shellcode finds itself in the memory (using the same markers as the downloader variation), it searches for the start marker (0xBABABABA) and end marker (0xBBBFFFFF) of the embedded payload then proceeds decrypting the data in between (using a 4 byte XOR key 0xCAFEBABE, preserving null values). On a successful decrypt, it saves the payload to disk and executes it.

Unlike the major competitors, the Microsoft Word Intruder and the Ancalog builder, decoys have always been supported by AK-1. This is an important feature in successful attacks, displaying a decoy content can help hide the malicious activities in the background, and it is less suspicious if this content is displayed after the victim opens what he thinks is a Word document.

The next thing the shellcode does is decrypts the decoy document content using the same algorithm as before but with different start/end markers (0xBBBFFFFF and 0xBCBCBCBC) and encryption key (0xBAADFD00D). When decrypting is finished, the decoy is displayed using Microsoft Word. The decoy document is not persistent; the content will be lost after closing Word.

All versions of the builder have this identical code for encrypting the payload executable:

```python
def makeExploit(doc_template, payload, decoy_doc):
    payload_start_tag = "\x8A\x8A\x8A\x8A\x8A\x8A\x8A"
    payload_end_decry_start_tag = "\x8B\x8B\x8B\x8B\x8B\x8B\x8B"
    decoy_doc_end_tag = "\x8C\x8C\x8C\x8C\x8C\x8C\x8C"
    Exploit_data = decoded_rtf_template + payload_start_tag + payload + payload_end_decry_start_tag +
    decoy_doc + decoy_doc_end_tag + Generate_Random_Bytes(0x200) + "\x00" + 0x100
    return Exploit_data

def MyMain():
    try:
        payload_encoded = encryptFile(sys.argv[1], 0xBEBAFECA)
        decoy_encoded = encryptFile(sys.argv[2], 0x8DFB00AD)
        exploitCode = makeExploit(decoded_rtf_template, payload_encoded, decoy_encoded)
        writeFile(sys.argv[3], exploitCode)
```

*The encryption in the builder implemented in Python*

And the reverse of this, the decryption process (note that for the simple XOR encryption the encryption and decryption algorithm is the same) is done by the shellcode in the following form:
The decryption in the shellcode implemented in assembly

In some of the dropper samples (e.g. the document with SHA-1: 1a90d16f1278123e50706b456ceff3a62bebe369f) the KAV-aware code is absent. As the generator does not support configurable options, it has to come from a different hardcoded template.

**Distributed malware**

We have identified about 760 malicious documents generated by AK-1, which were used to distribute over 50 different malware families. The most frequent are illustrated in this chart:
In its heyday the most popular Trojans (Zbot, Chisburg, Fareit, Neurevt) were favored but with the appearance of the AK-2 these variants have slowed down somewhat. It appears a few diehard groups are still using the older version of the kit, but they are mostly deploying the PredatorPain keylogger (which is the most frequent benefiter of the kit) and the NetWiredRC backdoor.

AK-2

We wrote about the characteristics of this kit back in a research paper we published on our security blog Naked Security [4] when we first saw document generated by it, in August 2015.

Like its predecessor we are not aware of other public reports related to this exploit kit, thus, we have no official name for it.

The source code of the builder is based on the AK-1 builder Python script, it shows the same characteristics.

Someone (could be the same author, the script itself did not contain contact info) used this script to create an EXE wrapper (Asprotect protected):

![Image of the executable interface]

The executable is nothing but a fancy GUI around the original, it contains the embedded builder script, drops it, and executes with the batch file:

```
cmd /c code.py malware.exe decoy_document.doc Exploit_Sample.doc
```

Interestingly, this wrapper contains another embedded document, which is a default decoy document. Likely if no decoy is specified, then this content gets added:
We have seen generated malware samples where the criminal didn’t bother to use their own decoy, rather relied on the hardcoded decoy content. In fact, the author of this builder version uploaded to Youtube a video [8] that features an earlier version of the builder, which has no option for decoy document setting, and displays the very same decoy.

The usage of the kit shows a steady activity over the time, with a strong peak in August 2016 as shown in the chart below.
Number of new samples generated by AK-2

Characteristics

There are many characteristics that connect this generator to AK-1: the shellcode is essentially identical including the encryption algorithm and encryption keys.

The marker bytes indicating the beginning of the second stage shellcode are changed, however, to 0xFEFEFEFE and 0xFFFFFFFF (illustrated in the following code fragment):

```
cmp  dword ptr [eax], '\t"
jnc short loc_2007a
add  eax, 1000h

next_handle:    ; CODE XREF: segBBB:00ZH9CGj
                ; segBBB:00Z89074j
83 CB 0h        add  eax, 9
81 3B FE FE FE    cmp  dword ptr [eax], EFEFEFEh ; marker 1 and 2
75 55          jnc short next_handle

skip_bytes:     ; CODE XREF: segBBB:00ZH9CGj
4B              inc  eax
80 3B FE        cmp  byte ptr [eax], 'M' ; marker
74 EA          jz  short skip_bytes
83 3B FF        cmp  dword ptr [eax], FFFFFFFFh ; marker 3
75 EA          jnc short next_handle
83 CB 0h        add  eax, 9
99 C0            mov  edi, eax
FF 77 10        push  dword ptr [edi+10h]
FF 77 1C        push  dword ptr [edi+1Ch]
FF 77 28        push  dword ptr [edi+28h]
FF 77 1F        push  dword ptr [edi+1Fh]
89 BF 00 00 00 00    mov  edi, [edi+0000h]
89 FE            mov  eax, edi
89 00 10 00 00    mov  ecx, 1000h
F3 48          rep  stosb
FF 00            jmp  eax
```

This new revision of the kit removed the Kaspersky-aware code and as a result the length of the shellcode is about half that of AK-1 droppers.

The overall document structure is illustrated on the following scheme:
The infected droppers are RTF documents that contain four embedded objects, each play some role in the exploitation.

Possible exploitation workflow:

- Object 1 makes sure that msvc71.dll is loaded
- Object 2 fills the memory with the first stage shellcode.
- Object 3 triggers the vulnerability
- Object 4 would play the same role as Object 3, with a different vulnerability; however it doesn’t seem to do anything

**Distributed malware**

Since this version of the kit is a newer development, it is used by a fewer number of groups. This likely explains why there are only 35 malware families currently being deployed using them. But these groups are very active, even though the life span is shorter than that of AK-1, the number of samples is more than double, about 1,385 malicious documents were identified as created by this version.

The most frequent of them are displayed in the following chart:
The most commonly distributed Trojans are Zbot, Neurevt/Betabot, Bladabindi/njRat and Fareit/Ponyloader. In addition to that several credential stealers, like Dyzap, KeyBase, HawkEye, PredetorPain was also favored choices.

**Attribution**

In case of AKBuilder, it is hard to tell how many individuals or groups are working on it. Because it is a simple Python script, it is very easy to steal the builder and start a new “development branch.” It is quite possible that the work was started by a single individual, and then others jumped in and stole the code, releasing their own versions.

It is clear though that the known builder versions come from the same origin and could be considered belonging to the same development branch even though there are multiple email accounts connected to it.

Some of the distributors (including the most persistent one) are seemingly from the Arabic regions. There is no proof that there is any connection between them, though.

But apart from them, there are a handful of other, seemingly unrelated developers/distributors who sell versions of this kit. We suspect that most of them work independently, purchasing one version of the kit, then modifying and distributing it on their own. Some of the distribute only this kit, others seem to be involved in selling a wide range of malicious software builders.

This is possible because the release version of the kit is written in Python which makes it easy to understand and modify.

We think that there are about half a dozen individuals who are involved in developing and distributing AKBuilder but we don’t know the exact connections between them.
Conclusion

Cyber criminals find Office documents useful for delivering malicious programs to their targets. They’ve been using this method steadily over the past two years, and there is no sign that they intend to give up.

The availability of black-market tools makes it possible for a wide range of criminals to generate the exploited documents. After the disappearance of the Ancalog builder, AKBuilder took over as the most popular choice of these tools.

The rigid hard-coded structure of AKBuilder means that for any change in the generated samples, a new version must be released. That information helps the defenders: even if the first few samples go undetected, a quick signature update can protect the users for days or even weeks.

The dependence of criminals on the commercial offerings has a disadvantage for them: The builder doesn’t use zero-day exploits or even exploits that could be considered as new.

AKBuilder shows a moderate progressivity: new exploits like CVE-2014-1761 and CVE-2015-1641 are supported relatively fast after their first availability. But the kit is far from using zero-day exploits, in both cases the first use of the exploit was months after the vulnerability was disclosed and the patch made available.

In the final analysis, it shouldn’t be difficult to protect against these kinds of activities:

Just applying recent patches for Microsoft Office should disarm the attack.

References


[3] [insert Ancalog paper link here]


[5] https://www.youtube.com/watch?v=fTEr8NOr30Y


[8] https://www.youtube.com/watch?v=vztuV-dYnbA&index=13&list=UU-W0ctXYxRhVXA29Msn6A