Operation Clandestine Wolf – Adobe Flash Zero-Day in APT3 Phishing Campaign

fireeye.com/blog/threat-research/2015/06/operation-clandestine-wolf-adobe-flash-zero-day.html

Threat Research

Erica Eng, Dan Caselden

Jun 23, 2015

7 mins read

Zero Day Threats

In June, FireEye's FireEye as a Service team in Singapore uncovered a phishing campaign exploiting an Adobe Flash Player zero-day vulnerability (CVE-2015-3113). The attackers' emails included links to compromised web servers that served either benign content or a malicious Adobe Flash Player file that exploits CVE-2015-3113.

Adobe has already released a patch for CVE-2015-3113 with an out-of-band security bulletin (https://helpx.adobe.com/security/products/flash-player/apsb15-14.html). FireEye recommends that Adobe Flash Player users update to the latest version as soon as possible.

FireEye MVX detects this threat as a web infection, the IPS engine reports the attack as CVE-2015-3113, and the SHOTPUT backdoor is reported as Backdoor.APT.CookieCutter.

APT3

The China-based threat group FireEye tracks as APT3, aka UPS, is responsible for this exploit and the activity identified in our previous blog post, Operation Clandestine Fox. This group is one of the more sophisticated threat groups that FireEye Threat Intelligence tracks, and they have a history of introducing new browser-based zero-day exploits (e.g., Internet Explorer, Firefox, and Adobe Flash Player). After successfully exploiting a target host, this group will quickly dump credentials, move laterally to additional hosts, and install custom backdoors. APT3's command and control (CnC) infrastructure is difficult to track, as there is little overlap across campaigns.

Activity Overview

In the last several weeks, APT3 actors launched a large-scale phishing campaign against organizations in the following industries:

- Aerospace and Defense
- Construction and Engineering
- High Tech
- Telecommunications
- Transportation

Upon clicking the URLs provided in the phishing emails, targets were redirected to a compromised server hosting JavaScript profiling scripts. Once a target host was profiled, victims downloaded a malicious Adobe Flash Player SWF file and an FLV file, detailed below. This ultimately resulted in a custom backdoor known as SHOTPUT, detected by FireEye as Backdoor.APT.CookieCutter, being delivered to the victim's system.

The payload is obscured using xor encoding and appended to a valid GIF file.

Attack Vector

The phishing emails used by APT3 during this campaign were extremely generic in nature, almost appearing to be spam. An example email body:

Save between $200-450 by purchasing an Apple Certified Refurbished iMac through this link. Refurbished iMacs come with the same 1-year extendable warranty as new iMacs. Supplies are limited, but update frequently.

Don't hesitate . . .>Go to Sale

The string ">Go to Sale" was a link that used the following URL structure:

hxxp://<subdomain>.<legitdomain>.<TLD>/<directory>/<alphanumericID>.html

Exploit Details

The attack exploits an unpatched vulnerability in the way Adobe Flash Player parses Flash Video (FLV) files. The exploit uses common vector corruption techniques to bypass Address Space Layout Randomization (ASLR), and uses Return-Oriented Programming (ROP) to bypass Data Execution Prevention (DEP). A neat trick to their ROP technique makes it simpler to exploit and will evade some ROP detection techniques.

Shellcode is stored in the packed Adobe Flash Player exploit file alongside a key used for its decryption. The payload is xor encoded and hidden inside an image.

Exploit Packaging

The Adobe Flash Player exploit is packed with a simple RC4 packer. The RC4 key and ciphertext are BinaryData blobs that the packer uses to decrypt the layer 2 Adobe Flash Player file. Once decrypted, layer 2 is executed with loader.loadBytes.

Vector Corruption

Layer 2 uses a classic Adobe Flash Player Vector corruption technique to develop its heap corruption vulnerability to a full relative read/write available to ActionScript3. In this technique, the attacker sprays Adobe Flash Player Vectors to the heap, and triggers a write vulnerability to change the size of one of the vectors. The attacker can then perform subsequent reads and writes to memory outside the intended boundaries of the corrupted Vector object from AS3. For more details on this technique, see Flash in 2015.

Once the attacker has limited read/write access to memory, they choose to corrupt a second Vector to increase their access to a range of 0x3fffffff bytes. This second Vector is used for the remainder of the exploit.

Return-Oriented Programming

The attackers use a ROP chain to call kernel32!VirtualAlloc to mark their shellcode as executable before jumping to their shellcode.

Instead of writing their ROP chain to the heap along with their shellcode and payload, they used a different technique. Usually, exploit developers will corrupt a built-in Adobe Flash Player object such as a Sound object. Instead, the attackers chose to define their own class in AS3 with a function that takes a lot of arguments:

```
class CustomClass {
    public function victimFunction(arg1:uint, arg2:uint, …, arg80:uint):uint
}
```

Then, the attackers can simply overwrite the function pointer with a gadget that adds to the stack pointer and returns to pivot to ROP. They have no need to identify the absolute address of the ROP chain and preserve it in a register for a typical xchg reg32, esp pivot. Additionally, storing the ROP chain on the stack will evade ROP detection mechanisms designed around detecting when the stack pointer points outside of a thread's stack region.

```
this.customObj.victimFunction(
6f73b68b, // ret; (ROPsled)
…,
6f73b68a, //pop eax
1f140100,
6fd36da1, //call Kernel32!VirtualAlloc(0x1f140000, 0x10000, 0x1000, 0x40)
1f140000, // Address
00010000, // Size
00001000, // Type
00000040, // Protection = RWX
6f73b68b*9 // ret (ROPsled)
6fd36da7*2 // ret
6f73aff0 pop ecx
6fd36da7
6fd36da7 jmp [eax]
…
)
this.customObj.victimFunction pointer modified to:
00000000`6de533dc 5e          pop     rsi
00000000`6de533dd 83c448      add     esp,48h
00000000`6de533e0 c3          ret
```

Lastly, the ROP chain has a ROPsled following the call to VirtualAlloc. This could just be an artifact of development, or it could be designed to bypass detection mechanisms that test for valid return addresses up to a limited depth at calls to VirtualAlloc.

Full Exploit Flow

1. Create a new Video object

2. Fetch the payload

3. Attach the video to a new NetStream

4. Spray the heap with Adobe Flash Player Vectors

    a. Create a Vector containing 98688 Vectors containing 1022 uints

    b. Set the first two dwords in each Vector<uint> to 0x41414141, 0x42424242

5. Create holes for the controlled FLV object

    a. Free approximately every 3rd Vector in the spray

6. Spray custom class objects for future control transfer

    a. Define a new class CustomClass

        i. Define a function victimFunction with lots of arguments

    b. Create a Vector of 0x100 Vectors of 1007 references to an CustomClass instance

7. Fetch and play the FLV exploit

    a. The FLV file will allocate an attacker controlled object in one of the holes from step 5

    b. The attacker controlled object will overwrite the length field of an adjacent vector

8. Re-fill holes from step 5 with Vectors as in step 4

9. Find the corrupted vector

    a. Search through Vectors from step 4

    b. Check the length of each Vector to find one that is abnormally large

10. Corrupt a second Vector (Vector2)

    a. Using the corrupted Vector from step 9 to read/write relative memory addresses

        i. Search memory for an adjacent vector

        ii. Overwrite the length field with 0x3fffffff

        iii. Verify that a corrupted vector with length 0x3fffffff now exists in the spray

            1. If not, undo corruption and attempt to corrupt the next vector

11. Decrypt shellcode and store it and the payload on the heap

12. Overwrite the CustomClass.victimFunction function pointer

    a. Find the sprayed CustomClass object instance references from step 6

    b. The new function is a form of "pivot" that transfers control to the attacker

13. Build ROP chain on the stack and call it

    a. Find ROP gadgets in memory using Vector2

        i. Including a call to kernel32!VirtualAlloc

    b. Call the corrupted CustomClass.victimFunction from step 6.a.i

        i. Arguments to the function are the gadgets of the ROP chain

        ii. They are conveniently pushed onto the stack

        iii. Corrupted vtable from step 12 calls a pivot

            1. The "pivot" just adds to to the stack pointer and returns because the ROP chain is on

                the stack

14. ROP chain calls shellcode

    a. Call kernel32!VirtualAlloc

    b. jmp to shellcode

15. Shellcode calls payload

    a. Shellcode searches memory for the payload, which is stored inside an image

    b. Shellcode decodes the payload by xoring each byte (that is not 0 or 0x17) with 0x17

Conclusion

Once APT3 has access to a target network, they work quickly and they are extremely proficient at enumerating and moving laterally to maintain their access. Additionally, this group uses zero-day exploits, continually updated custom backdoors, and throwaway CnC infrastructure, making it difficult to track them across campaigns.

Acknowledgements

Thank you to the following contributors to this blog!

- Joseph Obed, Ben Withnell, Kevin Zuk, Genwei Jiang, and Corbin Souffrant of FireEye