Adobe Flash Vulnerability CVE-2015-7663 and Mitigating Exploits
INTRODUCTION

Today Adobe released a patch for CVE-2015-7663 [1] that addresses a vulnerability we discovered in Flash Player.

The vulnerability exists due to the improper tracking of freed allocations associated with a “Renderer” object when handling multiple progress bar additions. This can be forced to overflow a Bitmap object corrupting adjacent memory. As we will discuss later, we originally exploited this bug in the lab using the common Vector length corruption target.

In this post I wanted to focus on mitigating the exploitation of Flash Player, and the challenges associated with it, instead of the traditional look at this particular vulnerability in detail.

But first, a little insight into why we see Flash in APT campaigns and exploit markets. From the attacker’s perspective, Flash is an amazing access capability.

1. It’s cross platform
2. It’s cross browser
3. It can be embedded in other documents and formats
4. It has a very rich programming language available
5. It’s easy to fuzz
6. There is so much code, vulnerabilities are sure to shake out

Because of this, attackers know that a good Flash exploit can give them reliable access to Windows, Linux, OS X, and Android systems through Chrome, Firefox, IE, Reader, Office, and more! For these reasons Flash exploitation is valuable, and will continue to be so. One capability could easily cover a large majority of all Desktop targets.
ADOBE VECTOR LENGTH CORRUPTION TECHNIQUE

The Vector corruption technique we used to exploit CVE-2015-7663 has been publicly known since at least 2013 \[2\]. It is a classic exploitation concept that provides a few “nice to haves” as an exploit writer.

1. Corruption gives the attacker read/write of virtual memory
2. You can allocate arbitrary sizes
3. It is resistant to corruption and application crash
4. There is no validation or protection of its contents

Due to the popularity of this technique over the past year, we have seen a rise \[3\] in Flash exploits \[4\] using it.

VECTOR ISOLATION

In response to the widespread use of this technique, Adobe has strengthened the security posture of Flash by adding two defenses to help reduce the effectiveness of zero day exploits \[5\].

Previously, Flash used a single heap for allocating all ActionScript objects. Doing this allowed an attacker to target a Vector objects length property when overflowing an adjacent buffer, by coercing the allocation algorithm to position different objects consecutively. Doing this after corrupting memory gave attackers read and write access to virtual memory, making it simple to bypass ASLR and execute code. Here is a graphic showing this memory layout.

However, starting with Flash Player version 18.0.0.209 \[6\], Adobe has made this more difficult.
Now, Flash Player allocates Vector objects in the default runtime heap, instead of the heap associated with ActionScript interpretation. This effectively removes the ability to coerce the allocator into creating adjacent blocks of memory for an attacker to use when corrupting the “length” property. The memory layout before now looks more like this.

This concept of moving specific allocations into separate heaps is called “Heap Isolation”.

The idea of isolating heap allocations for security purposes is not new. In fact, Microsoft[^7], Mozilla[^8], and others, have been pursuing the idea for years. This approach disrupts some of the steps in the exploitation process that are typical of almost all exploits leveraging memory corruption vulnerabilities.

1. Allocate memory linearly in a predictable order of specific sizes
2. Free a subset of those allocations creating predictable “holes”
3. Trigger a vulnerability that falls into the predictable locations and corrupts adjacent memory allocated in step 1.

Heap isolation effectively breaks Step 1 and 3 by placing certain objects in isolated sections of memory. For example, only Vector objects can be allocated in Heap A, and only ByteArray objects can be allocated in Heap B. Never together. Thus ensuring an attacker cannot allocate memory of another type adjacent to them making it impossible to control virtual memory enough to corrupt adjacent allocations of interest.
VECTOR PROPERTY GUARDING

Additionally, because moving an allocation is not sufficient in some cases, the Vector object now contains a precomputed value \(^9\), often called a “cookie”, that is checked for consistency before the length property is used. If an attacker corrupts this cookie the Flash application aborts and alerts the user. This breaks #4 in the “nice to haves” we discussed earlier.

Unfortunately, we know from experience that preventing a specific technique, such as Vector length corruption, will not stop attackers \(^{10}\). Instead, we see a new technique pop up with the same characteristics I have outlined above. There are likely many more “Vector like” objects available in ActionScript/Flash Player beyond ByteArrays and attackers have already found them.

ENDGAME HEAP ISOLATION

Beyond vulnerability discovery, Endgame’s Vulnerability Research and Prevention team is also focused on mitigating and detecting exploits. We provide protections to customers against whole classes of attacks, without the need for source. We feel particularly aligned to do this because of our extensive experience discovering and exploiting software vulnerabilities.

One of our first research efforts focused on generic enforcement of heap isolation. Instead of enforcing isolation on specific objects like Vectors, we apply it to every object that fits our criteria. This is particularly well suited for prevention of the previously described techniques, as well as vulnerability classes like Use-After-Frees (UAF).

An attacker exploiting a UAF must reallocate a different object into the freed memory location when an object has been released. This reallocation is what eventually gains the attacker code execution by controlling the function pointers in an object.
Forcing heap isolation ensures the attacker can only reallocate the original object, effectively preventing exploitation. The illustration below helps to visualize this effect.

![Heap Isolation Illustration]
This can be a powerful mitigation against specific bug classes and in our testing it has been proven to prevent a large portion of reported vulnerabilities. But we can do more.

ENDGAME CONTROL FLOW INTEGRITY

In addition to heap isolation we can also enforce control flow integrity (CFI) policies on an application. Whereas heap isolation can be very effective at preventing successful exploitation, a CFI based approach additionally allows us to detect active exploitation attempts since we are inspecting and validating when control flow — the path that an application executes — has changed. In the majority of exploits we have studied, there is a point when the attacker must “hijack” control of the process to begin executing a ROP chain — used to bypass DEP — or arbitrary code.

To accomplish this, Endgame has adapted and expanded on the idea of utilizing processor functionality to determine the exact moment when this happens. Inspired by a novel approach published by researchers at Fudan University\cite{11,12}, we leverage CPU branch misprediction\cite{13}, allowing us to introduce control flow integrity policies without expensive binary modifications to a target application such as hooking or dynamic instrumentation.

We have extended this technique to work on both Linux and Windows 64-bit operating systems and have used it to detect our exploitation of CVE-2015-7663 as well as others, including CVE-2014-0556\cite{14} and the exploit used in the APT campaign Russian Doll CVE-2015-3043\cite{15}.

The following output shows our system catching the exploitation of CVE-2014-0556 on a 64-bit Linux host.
The FROM_IP in the anomalous branch detection is the point when the exploit has control over execution.
The TO_IP is the beginning of the payload. In this case no ROP is used which would be missed by ROP only detection methods.

The following screenshots shows the full system preventing this exploit in real-time.

This work is exciting, as it has already shown its effectiveness at comprehensively detecting unknown exploits regardless of the specific technique used by observing abnormal program execution indicative of exploitation.
CONCLUSION

We know from experience that vulnerabilities and exploits will continue to make headlines. With the ubiquity of Flash and its high value, attackers will invent creative ways to exploit bugs. We have already seen how Adobe’s recent mitigations are a great step forward, but are not keeping pace with the attackers’ ability to exploit vulnerabilities. We understand it’s an iterative process that eventually poses a significant limitation to attackers, but there is still a long way to go.

Endgame is working hard to defend against advanced attacks on all software by developing cutting edge mitigations that work in tandem with strong vendor protections, affording the end user better defense in depth. Our unique experience allows us to test real exploits against real software, something we find necessary to providing adequate protections.

Look for future posts where we cover additional mitigations and share more vulnerabilities!
REFERENCES


[2] https://sites.google.com/site/zerodayresearch/smashing_the_heap_with_vector_Li.pdf


