CSCI 599 : Software Engineering for Security

Research Paper Presentation

Ritu Pravakar

A fine-grained classification and security analysis of web-based virtual machine vulnerabilities

Yilmaz, F., Sridhar, M., Mohanty, A., Tendulkar, V., & Hamlen, K. W. (2021). A fine-grained classification and security analysis of web-based virtual machine vulnerabilities. Computers & Security, 105, 102246.

Presentation Outline

- Introduction
- Inscription
- Overview
- CVE and NVD databases classification
- Classification for vulnerabilities
- Less commonly exploited vulnerability classes and example vulnerabilities
- 2013-2020 ActionScript Vulnerability Statistics: Number, Type and Attack Vector
- Conclusion

Introduction

- Attackers primarily target web-based virtual machines due to their vulnerabilities in design and the web's ability to enable connectivity
- Vulnerability information that is inconsistent, erroneous, or ambiguous makes it more difficult to diagnose vulnerabilities in web-based virtual machines (VMs)
- Web-based virtual machines (VMs) are prone to vulnerabilities since they run web scripts that web browsers cannot quickly execute

Inscription

- The first completely automated approach for transforming Adobe Flash binary code to protect against the main categories of Flash vulnerabilities
- It doesn't alter Flash VMs that are susceptible
- It reduces a type of online threats that aim to harm outdated Flash virtual machines (VMs) and the applications that run on them
- ActionScript mitigation against significant Flash Player VM attacks that automatically converts and secures untrusted ActionScript binaries in-flight
- It doesn't need any web browser or virtual machine fixes or updates
- It adds additional security code to incoming Flash binaries, which self-checks against known virtual machine attacks
- Inscription-modified Flash programs are self-securing

Inscription

- It presumes that untrusted Flash binaries could be harmful in their entirety
- It needs defense against outside script code manipulation
- Assumption: Enemies may be aware of every Inscription implementation detail before planning an attack
- For each binary, it replaces and alters any potentially hazardous script actions to make sure that its security measures are impenetrable
- It can secure known dangerous operations with safe substitutes by identifying the version of the virtual machine (VM) that is executing and restricting its security guard implementation to operations

Overview

Higher up, Inscription functions automatically

- Disassembles and examines binary Flash programs before running them
- Equips them with additional binary operations
- Reassembles and packages the altered code into a fresh, Shockwave Flash (SWF) binary that has been security-hardened

There are two main types of guard code insertions in Inscription's guard code

- Inserting or changing bytecode instructions directly
- Using a package to replace classes that could be abused with wrapper classes

CVE and NVD databases classification

- "Memory Corruption" flaws account for 30% of ActionScript vulnerabilities
- It can also be classified as a buffer, integer, heap, or use-after-free (UAF) or double-free (DF) vulnerability
- A sizable portion of CVE entries are marked as "Unknown" because they fail to specify the kind of ActionScript vulnerability
- Potentially four "critical" and exploitable vulnerabilities affecting confidentiality, 42 vulnerabilities affecting integrity, and 46 vulnerabilities affecting availability were misclassified by the CVE and NVD databases
- This means that CVE vulnerabilities that exhibit any kind of memory corruption are likely to be given the ambiguous "Memory Corruption" title

Classification for vulnerabilities

- When researchers pick vulnerabilities, they typically use a coarse-grained classification, which results in information loss.
- A wide range of different vulnerability subclasses, including uninitialized pointer access, out-of-bounds reads, and pointer value returns outside of anticipated ranges, are collectively referred to as memory corruption

Use-After-Free

- Utilize-AfterFree UAF vulnerabilities cause memory that has been freed to have a dangling reference referencing it
- Even if the memory is eventually allocated to another object, referencing a released object allows unauthorized access to the memory
- The memory pointer is left in place when the object is released, creating a dangling pointer that an attacker could use to tamper with the contents in this specific memory segment

Classification for vulnerabilities

Double-free

- Double-free DF vulnerabilities result in the freed memory pointing to itself as both forward and backward memory and being placed twice at the head
- Overwriting pointers in freed memory can be used to craft nefarious actions like arbitrary code execution
- This vulnerability can be used to shift the program's control flow to an arbitrarily chosen code block through a carefully constructed attack

Out of Bounds Read

- Via out-of-bounds read vulnerabilities, data can be accessed beyond or before the intended buffer's commencement without authorization

Classification for vulnerabilities

Buffer overflow

- Because the return address for a function comes directly after a buffer in the stack that contains function arguments that could be user inputs, this vulnerability is mostly caused by overwriting the return address

Heap spraying

- Exploits employ the technique known as "heap spraying" to enable arbitrary code execution
- Heap spraying is used to maximize the likelihood that the malicious code that attackers wish to run on the target machine—injected shell code—will be appropriately transferred to the program flow. However, it is not a vulnerability in and of itself

Less commonly exploited vulnerability classes and example vulnerabilities

Integer overflow(or underflow)

- When an arithmetic operation tries to produce and allocate an integer, it might cause an integer overflow because it needs more main memory space than the operating system allows
- Either the value exceeds the maximum value or the value is less than the lowest value that can be displayed

Heap overflow

- A heap overflow, which is a type of buffer overflow, occurs when data is written to a section of memory that has been allocated to the heap without the data's bounds being checked

Less commonly exploited vulnerability classes and example vulnerabilities

Type confusion

- When a software allocates or initializes a resource—such as a pointer, object, or variable—using one type and then accesses that resource later using a different type that is incompatible with the original type, this is known as type confusion vulnerability

Security bypass

- Any security defensive strategy, like Address Space Layout Randomization (ASLR), which generates artificial diversity by randomly relocating some system components' memory locations to strengthen systems against buffer overflow attacks, may have security bypass flaws

2013-2020 ActionScript Vulnerability Statistics: Number, Type and Attack Vector

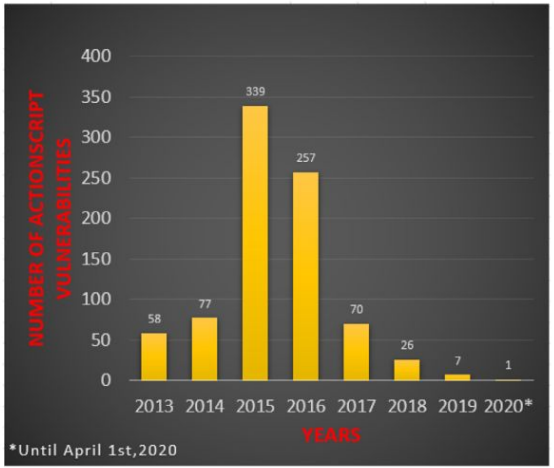

Fig. 15 – Number of ActionScript vulnerabilities per year in CVE and NVD databases between 2013 and 2020.

- The number of ActionScript vulnerabilities found in the CVE and NVD databases between 2013 and 20201 is shown in this graph.
- In 2015, the quantity of vulnerabilities found rose by approximately 340%.
- ActionScript vulnerabilities are becoming fewer in number each year, however attackers are still actively taking advantage of vulnerabilities from previous years because older, unpatched AVMs are still widely used on the internet, and since 2018, four new zero-day vulnerabilities have surfaced.

2013-2020 ActionScript Vulnerability Statistics: Number, Type and Attack Vector

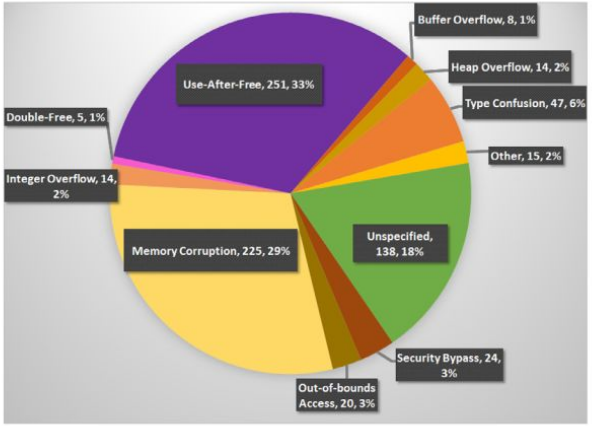

Fig. 16 – Types of ActionScript vulnerabilities as shown in the CVE and NVD collections between 2013 and 2020.

- Based on the raw vulnerability descriptions found in the CVE and NVD databases, this graph shows the number and types of ActionScript vulnerabilities found since 2013.

2013-2020 ActionScript Vulnerability Statistics: Number, Type and Attack Vector

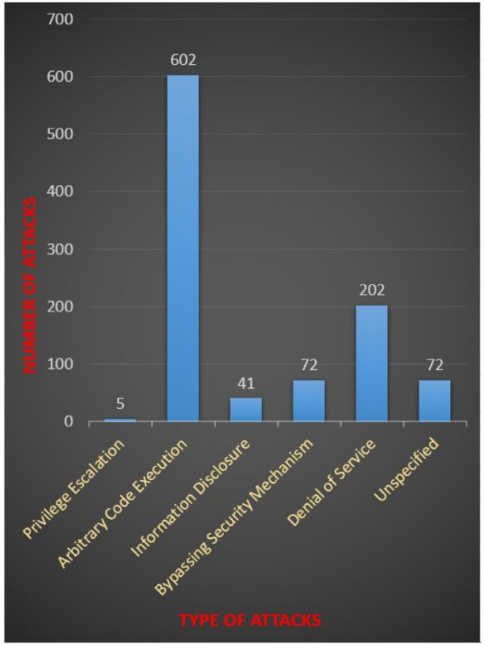

Fig. 17 – Types of exploits that ActionScript vulnerabilities can lead to.

- The number of attack types available in the CVE and NVD collections that can be carried out following the exploitation of ActionScript vulnerabilities is displayed in this graph
- The graph shows that one of the most dangerous kinds of attacks, arbitrary code execution, can result from almost 75% (601/761) of ActionScript vulnerabilities
- This approach allows an exploit to move the program flow to any random code segment—remote or on the victim machine—so that it can be executed on those computers

Conclusion

- With the help of a novel security tactic called Inscription, significant Flash vulnerability categories can be avoided without the need to directly patch susceptible virtual machines and browsers
- This is the first method of bytecode modification that makes the assumption that the underlying virtual machine may not be entirely reliable
- On the target computers they arrive to, the converted programs efficiently self-detect and self-mitigate Potential attacks of vulnerable VMs
- The method is successful in thwarting numerous significant types of remote code execution attacks that are common in malicious web scripts, according to experimental evaluation