Dynamic Java Applet Modification

October 20, 2013

Darkred

This page intentionally left blank

Abstract

Frequently Java provides an entry way for malicious code via so called Java "Drive-by" applets that attempt to silently install software onto a computer. Antivirus software attempts to block the majority of these by blacklisting domain names or blacklisting individual static signatures.

This paper presents an approach using dynamic compilation of Java applets to avoid static signature tags that antivirus software attempts to use to detect such drive-by attacks, thanks to reductions in costs of fast servers it is now possible to utilize obfuscation to create and serve a unique Java applet to each individual visitor thus rendering static detection impossible.

It could pose a threat to many antivirus systems that do not have a frequently updated domain blacklist. Currently there is no method of mitigation other than domain blacklists and Java applet whitelists.

Introduction

The Java "Drive-by" is well known as the most common attack vector in exploit packs circulating the internet. While Java applets are mainly used to deliver software to a customer with ease the same technology is frequently used for malicious intentions.

Malware authors have been searching out and exploiting such Java exploits for profit and personal gain causing an explosion in the number of effected computers. Oracle has been struggling to keep up with patching all the exploits found thus requiring antivirus vendors to rely on domain name blacklisting and finding static signatures as the applet is transferred to the client computer.

While static signatures work fine for detecting massively spread generic viruses they preform very poorly against specialized unique threats. This paper describes a method in which a malicious Java applet could be changed for each individual user thus severely hampering the effectiveness of static signature based detection.

Dynamic Java Applet Modification

Dynamic modification of malicious content to avoid static detections is not a new concept and has been implemented by many including myself[1] but when dynamic modification exploits of common plugins such as the Java platform get into the wild it can create a dangerous mix capable of exploiting many systems.

[1] Masse Crypter `https://github.com/vpnguy/MasseCrypter` utilized dynamic modification to create a prime number logic time loop

Experiments

Initially I created a simple test applet that automatically greeted a user with a small message box containing a fixed string. After creating a simple ghostwriting script to modify the class name and message box contents the final implementation of SandGrouse was created which allowed for dynamic modification, compilation and serving of a simple test applet.

Implementation

The included proof of concept is known as SandGrouse `http://github.com/vpnguy/sandgrouse` , It attempts to modify an included benign Java applet by changing the content of a message box and the Java class name. Sandgrouse accomplishes said modifications by utilizing a php script with basic string substitution functions.

SandGrouse uses a variety of techniques to assist in a malicious execution of a Java applet including usage of a HTML iframe to allow the necessary time for the php script to compile the payload applet. By utilizing an iframe to allow delayed applet activation it not only prevents automated virus scanners from flagging the site within their short observation window but it also prevents server load from causing suspiciously long load times that could cause potential traffic to become disinterested with the attack page.

SandGrouse also attempts to modify notification content and class names to prevent static tagging of the bytes by a HTTP interception system. By avoiding HTTP traffic based detection schemes SandGrouse is able to slip past most commercial antivirus and firewall products. SandGrouse can be detected via no means known to the author at the current time.

Implications

By allowing an attacker to create multiple unique exploits with the same payload SandGrouse and systems like it allow an attacker to extend the undetected lifespan of an exploit on the wild internet. With enough domains and HTML obfuscation SandGrouse could allow for an exploit to run rampant until Oracle issues a patch to fix such an underlying vulnerability which usually takes at least a couple weeks for the average Java exploit. If SandGrouse was combined with a XSS based redirection attack launched from social media it could yield extremely high exploitation rates and easily exploit thousands of computers.

Conclusion

In conclusion dynamic applet modification is a very serious threat that has no current means of mitigation. The only countermeasure against such attacks is retroactive blacklists on domains serving such malware or an automated scanner looking for *.class files laying around a suspicious server. Java exploits will continue to be a problem until Java is no longer a dominating force in the online world.