Mitigating the Unknown

Maxim Zavodchik, 2014-01-10

Mitigating “0-day” attacks, which are named like that because the programmer has zero days to fix the flaw, is apparently impossible. However in practice they can be significantly mitigated. We can buy some time by heavily reducing the “vulnerability window” (until the vulnerability is patched or a specific signature is deployed), thus shifting those attacks to be “N-day” attacks.

Once a widely used service has a “0-day” publicly disclosed, massive internet scans for vulnerable servers (known also as “campaigns”) are launched almost immediately. Those scans rely on bots either scanning the whole IP range or searching for potential targets using search engines (known as “Google Dorks”).

Being a “0-day” attack, there is no complete protection against it. However a good assumption will be that there will pass a certain time until the exploit will evolve enough to include variations and to deploy evasions or to be customized for a specific target. That’s where a good level of a proactive protection required. A typical “0-day” timeline might look like that:

A proactive mitigation strategy includes the following ingredients: positive security, proactive negative security, and attack symptoms mitigation. Taking the WAF as an example, positive security might consist of whitelisting only the needed meta-characters (while blocking all other), enforcing HTTP compliance, configuring mandatory request headers, and narrowing down HTTP methods, and file types. And much more can be whitelisted.

Whitelisting the entire application (building full positive security model) can be challenging sometimes. It is not less important to rely on proactive attack signatures which are not coupled with a specific CVE, but rather focusing on generic exploitation and evasion patterns and those which try to catch the actual post exploitation payload, regardless of the specific weakness which allowed delivering it in first place.

Due to the automation of the “campaign” process, a crucial mitigation factor during the “vulnerability window” might be also relying on detecting automation attack symptoms, such as deploying strong bot detection techniques, blocking TOR exit nodes and having a good IP reputation feed.

ShellShock Example

I want to use the latest high profile “ShellShock” vulnerability (CVE-2014-6271 and friends), and see how we take this theory into practice.

Let’s take some of the popular real attack vectors used in this recent attack and see how specifically BIG-IP ASM detected them using the proactive approach, before there was a designated “ShellShock” signature. The attack vectors are a mix taken from Exploit-DB, Metaslploit, shellshock.py, detectify.com portal, and requests recorded by honeypots.

Result of running the vectors against ASM

As we can see all of those vectors were blocked.

Blocked requests in the ASM event log

Let’s dig in and understand what prevented the exploitation. We can see that the exploit is sent via the “User-Agent” header. It is running “/bin/bash” to download the malware using “wget”, running it using Perl and finally removing the malware file itself.

Exploit in the wild

Signature triggered by an exploit in the wild

Without being aware to the actual weakness “() { :;};” which triggers the code execution, the exploit is caught for several reasons. First, we see that it is targeting the “/cgi-bin/bash” location, thus triggering 2 URI signatures looking for this senstive URL (200000034 and 200100316). Second, ASM caught the actual command that performs the server takeover (the “payload”) by a signature that looks for calling executables from “/bin” directory (200003058).

This is the exact example of proactive signatures which look for the actual “takeover” payload or the senstive location/resources that are being targeted, rather than only focusing on the exact weakness that opens the door for exploitation. As for postive security, customers who would fence themselves with non legitimate or very rare characters in headers such as “[“, ”]”, ”`”, ”{“, ”}” would have even prevented the 0-day attack itself (“{“ character in the case of ShellShock).

Several other signatures (2000021069 and 200021092) for automated user-agents, “wget” and “perl”, are also triggered as the payload is delivered throught the user-agent header (which is true for most of the "ShellShock" exploits).

Let’s observe another attack vector:

shellshock.py exploit

Signature triggered by shellshock.py

We see the same “/bin” execution signature (200003058), however we also detect a symptom of a suspicious behavior and a signature for automated python client is fired (200021101).

While looking on the exploit published on “Exploit-DB” and some other vectors in the wild we see that the only signature that bravely shields against their successful exploitation is the same “/bin” execution signature (200003058). Of course, if there was another command that does not have a corresponding signature (because it is not considered sensitive, most likely causes false positive or just missing) the attack vector could penetrate (like in the case of “() { :; }; ping x.x.x.x” vector).

But that’s where our previously mentioned assumption takes place. There is a crucial time, during the “vulnerability window”, just before the exploit expands to several variations or being evolved to other payloads as well. It is not by accident that we can rely on that single signature to buy us some time before the patch is applied.

Exploit from Exploit-DB

Signature triggered by exploit from “Exploit-DB”

Another proactive measure which is directly related to attack symptoms and can serve as a life belt during the “vulnerability period” is using ASM’s bot protection features which incorporates several state-of the art techniques to identify automated bots regardless of the payload they are trying to deliver.

Afterword

We are not stating that there is a complete protection against previously unknown attacks, however, there is definitely an already existing proactive set of tools that might significantly lower your chances for compromise in a critical exposed period until the full patch is deployed.