Preventing WebKit code execution attacks with ASLR

Luke Walker

Manchester Metropolitan University
luke.walker3@stu.mmu.ac.uk

March 2, 2021

Abstract

This paper attempts to discuss and review material related to ASLR, alongside giving recommendations for future mitigations, especially within the XNU kernel and how this security feature prevents attacks from occurring on our devices.

I. Introduction

ALSR and other kernel attack mitigations have been slowly creeping their way into computer systems that we use daily without most of us even knowing about it. While these mitigations may never be used by the average person, they are essential for keeping our most private data hidden from attackers. This paper will solely be focusing on iOS/XNU Webkit vulnerabilities and the methodologies that Apple have implemented to prevent attackers exploiting our devices through the internet. This topic is extremely interesting to me due to the fact of sites such as `jailbreak.me` and the Pegasus malware chain actively bypassing the mitigations put in place, giving the user or attacker `task_for_pid(0)`/Kernel Read/Write. `task_for_pid(0)` is a function available within XNU that allows a process to get the task port of another process on the same host, therefore if an exploit can give tfp0, it's relatively easy to execute code as root.

ASLR stands for address space layout randomization, it makes sure that any code loaded into the RAM has a completely random address, including system libraries and program constructs, which makes it harder for exploits to use attacks such as memory corruption, it also attempts to reduce the possibility of an return to libc attack, which used to be a popular method for executing arbitrary code on the device. ASLR is computed via the following algorithm

$0x01000000 + ((1 + 0xRR) * 0x00200000)$

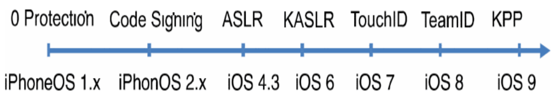
Where 0xRR is the random kernel slide, generated from a random byte from a SHA1 string generated on boot by the iOS boot loader, iBoot. ASLR is mandatory for applications and services running on the XNU kernel since iOS 4.3 and MacOS X Mountain Lion.

II. Literature Review

Due to iOS's closed operating system, the amount of research done in this sector is minimal, therefore this paper will attempt to collect all the knowledge around this topic in one place.

The first paper I'd like to discuss is by Charlie Miller [1, Mobile Attacks and Defense], the paper is quite old being published in August of 2011 however it goes into depth about apples iOS and Google's Android operating systems the paper starts off talking about how mobile phones were just mobile phones, and now we've effectively got a full computer in our pocket with additional hardware for communicating over radio waves. This means that these devices must be secure, the paper goes into detail about how iPhones won't run applications or load libraries unless it's been signed by Apple's private encryption key it also talks about sandboxes which effectively means of applications can't read other apps data, for example an application can't read stored SMS messages, there are certain permission systems within both operating systems which ask for consent before accessing certain information such as the address book. The paper then moves on to discuss how an attacker can attack iOS, specifically the paper talks about any malicious code that manages to bypass restrictions will be running as a less privileged user called mobile instead of root. This makes delivering malware such as keyloggers, spyware and other common forms that which usually see on our computers such as windows quite impossible. Both points about sandboxes and code not being able to run unless it's signed is extremely important along with ASLR for this mitigation. if an attacker can bypass ASLR there are still other mitigations such as the sandbox and code signing which would make it extremely difficult for malicious code to be delivered and executed through WebKit. The paper then starts to discuss known attacks against the iOS security model especially one we would be interested in, a website that could jailbreak a device just by visiting it. This website used two vulnerabilities to gain root access privileges through WebKit which would then drop a payload to jailbreak the device, ALSR didn't prevent these attacks from working Additionally once a device is jailbroken quite a lot of security features built into iOS are disabled, such as data execution prevention. This paper is good explaining the basic concepts behind iOS security however it doesn't go into detail about how we can attack webkit directly nor ASLR, however it does raise some good points about the sort of code an attacker might want to gain by exploiting webkit. For many people, malware and cyber security is a growing trend, and people would not like to get their devices infected with malware by visiting malicious sites. The iOS security model prevents this type of attack and makes it so vulnerabilities in the security stack would only be used to steal sensitive information such as what would be stored on politicians and executive staff's devices such as government or organisation secrets, This is because these exploits are quite hard to come by and sell for up to 10 million so it's mainly APT's that can afford these type of exploits which mitigates the average malicious attacker.

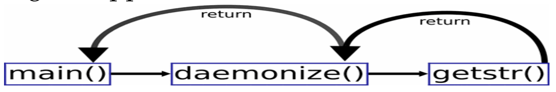
The second paper I'd like to look at is a review of iOS malware [2, Review of iOS Malware] written by Zhu Yixiang and Zhang Kang. With the discussion I've already had about what an attacker might want to gain, as seen in the first paper review, this seems like a good time to discuss and analyse papers regarding iOS malware. The paper goes into detail about the lack of papers regarding iOS malware, while there are prevalent papers regarding Android malware. The paper goes into discussion about the security stack that was slowly introduced to iOS devices, starting with ASLR with iOS 4 as has been mentioned in this paper already. Below is a figure displaying the security stack, taken from the current paper in discussion.



As we can see, iOS has greatly improved the security stack of it's operating system and due to the fact that iOS is designed to only run on Apple's hardware it's much easier to implement features such as hardware based encryption and kernel isolation, making attacking the kernel much harder on iOS. The paper also discusses how it's nearly impossible to downgrade the iOS firmware, so bugs which might be promising for jailbreakers and malicious actors are permanently patched. In relation to software deployment on iOS, Applications must be checked over by the App Store team, so any malicious apps are extremely difficult to publish on the App Store which narrows the gap for malicious actors to use. Apple do however have 3 types of signing certificates for Applications, a self signed certificate, which anyone with an Apple ID can create, these certificates expire after 7 days and have to be manually approved by the owner of the device before an App can be executed, there is also developer certificates, which last up to 3 months, which could be used by attackers, but they both require technical know how to use/install apps on the device, which is where we end up with the discussion of Kernel attack mitigations within the Safari browser and Webkit. With Webkit, Javascript is implemented with a just in time compiler, therefore we're able to make the device execute JavaScript in a isolated environment, this is great for attackers, as a misconfiguration in the sandbox may lead to the development of exploit chains. The paper then starts to discuss the impact of iOS jailbreak, and how easy it is for malicious attackers to upload packages to third party app stores with malware inside of them, two cases that are discussed are Ikee and AppBuyer, Ikee was a worm that attacked the SSH server installed, since most users don't change the default login for SSH, this worm was able to propagate, AppBuyer used a technique to steal Apple ID emails and passwords from the victims device, once the malicious app was installed. In review of this paper, the writers went into detail about common attacks, however there was little relevancy to the overall discussion of Webkit and ASLR, however it did go over important security stack points which are beneficial to us understanding iOS at a lower level.

The next paper I'd like to discuss is a paper on the Robustness of ASLR's randomisation written by Jonathan Ganz, Sean Peisert et al. [3, ASLR: How Robust Is the Randomness?]. This paper examines the security provided by different implementations of ASLR. As already discussed, the ASLR security mechanism increases the control flow integrity making more difficult for attacker to execute attacks such as buffer overflows. This paper looks at each operating system and each implementation of ASLR to find out which one has the strongest memory entropy randomization. This is important when discussing ASLR as it's good to get a picture of the entire operating system landscape and how they've each individually implemented ASLR

To test each implementation the writers of this paper wrote an application in C,this application acted as a server which accepted input over a network via a network socket and then copied the data to a buffer of limited length, setting up a buffer overflow. A figure describing the applications flow can be found below.



The research suggests that Debian 64 bit has the most entropy for the ASLR randomization algorithm, the paper reported 28 bits of total entropy within Debian and following up was 64 bit hardenedbsd with 25 bits of entropy, Now might be a good time to ask what type of entropy does XNU have? the paper doesn't have the answer to that, However the XNU source code is available on Github and Apple's Open Source website. We can calculate that the ASLR entropy for the XNU kernel is 8 bytes long, making up a 64 bit entropy for the ASLR implementation on iOS, another paper helped me calculate this, written by Tarjei Mandt [4] So what good is this? Well now we know that the ASLR implementation on XNU is a lot more random than other known Linux or Unix like operating systems, which will make it more difficult for attackers to exploit vulnerabilities such as buffer overflows in the JavaScript JIT compiler in Webkit, to attack this, realistically an attacker now must have a information leak in the JIT compiler, so they're able to calculate the kernel slide offset, rather than attempting to bruteforce the value, which is extremely unlikely to work due to the entropy of the kernel address base.

I finally want to circle back round and have a look at the papers presented during the Blackhat conference in Las Vegas, I found two relatively good papers discussing Webkit, one written by Liang Chen of the K33nTeam [5] and one written by Ivan Krstić, head of security engineering and architecture at Apple [6]. The paper by K33nTeam has a case study of CVE-2014-1303 which was found during the Pwn2Own 2014 conference and also goes into discussion about the security features of Webkit. Chen discusses how ASLR on iOS is weak, as all system libraries share a single base address, also known as $dyld_{s}hared_{c}ache$, this is where all the libraries the system might need to use are stored, Why is this bad? Well if an attacker is able to leak an address via a Webkit exploit, it's easy for them to calculate where the $dyld_{s}hared_{c}ache$'s memory space is, therefore to exploit a device an attacker can find vulnerabilities within a shared system library, then exploit it remotely via webkit. Chen discusses an out of bounds read and write that was found during Pwn2Own within the CSS engine included with webkit, using this bug, Chen and the K33nTeam are able to overwrite memory within the CSS memory space, which allows them to change the flow of the application, which eventually leads to Arbitary memory address read/write, from here, Chen discusses the ability to use Return oriented programming to inject shellcode into the device, since ASLR has already been bypassed here, it's relatively simple to gain tfp0 and potentially inject malicious code straight into the $dyld_{s}hared_{c}ache$. Now, what are Apple deciding to do to combat issues like this, luckily, as mentioned before Ivan Krstic, head of security engineering also gave a talk at Blackhat 2016, he discusses how Safari will now have a 32mb allocated memory address space that is sandboxed from the rest of the operating system for the Javascript compiler to operate inside of, he discusses how within the ARMv8 architecture, it's possible to create this memory space with Execute only permissions, therefore it's not possible to read or write data in and out of the sandbox, which means it's extra difficult to preform attacks like K33nTeam pulled off. This is good as it narrows the attack vector that an attacker can use, since they now have to subvert the control flow of Webkit via Return oriented programming to find a way to call an execute only JIT function. Ivan also talks about the secure enclave processor, which is a micro processor which contains all the decryption and encryption keys needed to decrypt data that could be pulled via an exploit, SEP or the secure enclave processor isn't writeable or readable by userland and will require a iBoot exploit to read the data from the chip which of only 2 known bugs exist (Checkm8 and SHA1tter). This mitigates attackers being able to steal sensitive information from the users device.

III. Summary of the Review

ASLR and kASLR in the current implementation on iOS is well documented for hackers to exploit, therefore more kernel level protections need to be put in place to prevent users devices from being exploited by malicious attackers, however due to the current security stack of iOS devices, it's extremely difficult, as discussed within Ivan's paper to fully exploit a device via the webkit engine on the latest releases of iOS. ASLR is quite an old anti exploit technology and therefore better ways of preventing exploitation have been created since 2001, which is when ASLR was first introduced as a concept. There is no one single type of kernel exploit mitigation that will stop attackers, as it usually comes down to a cat and mouse game, where attackers find new ways to attack a system and security engineers find a better way of preventing such attacks, which is discuessed within Ivan's paper above.

IV. Recommendations

Ensure that devices that are accessing the internet via Webkit are running the latest updates both in terms of operating system and Webkit version, if a user is using Webkit on a different operating system, such as Windows, ensure that Windows defender is activated and running correctly, in the case of iOS devices, educate users, especially ones that would be more likely to be the target of an attack, eg CEO, chairmen, executives with proper security etiquette, since as discussed above the battle between engineers and malicious attackers is always changing as new ways to circumvent systems is discovered. Companies must ensure that devices are kept up to date and a proper firewall with up to date rules is active on the corporate network which could potentially stop attacks via incepting requests to known exploit websites.

References

[1] C. Miller. Mobile attacks and defense. *IEEE Security Privacy*, 9(4):68–70, July 2011.

[2] Z. Yixiang and Z. Kang. Review of ios malware analysis. In *2017 IEEE Second International Conference on Data Science in Cyberspace (DSC)*, pages 511–515, 2017.

[3] J. Ganz and S. Peisert. Aslr: How robust is the randomness? In *2017 IEEE Cybersecurity Development (SecDev)*, pages 34–41, 2017.

[4] Tarjei Mandt. Revisiting ios kernel insecurity: Attacking the early random prng.

[5] Liang Chen. Webkit everywhere: Secure or not?

[6] Ivan Ivan Krstić. Behind the scenes with ios security.