Literature Review

Jordan Peters

March 1, 2020

I certify that all material in this dissertation which is not my own work has been identified.

Contents

1 Introduction

In 2010, a "worm" virus, one that spreads across and embeds itself into systems, infected approximately 100,000 systems [1] and controlled 984 nuclear centrifuges to repeatedly malfunction them [2] over the span of at minimum 1 year, and likely more. [1] It eluded detection by performing specific subroutines that would cause the equipment to only breakdown in such a way that it would cause no harm to people, and would make the scientists believe the equipment they were sold was just faulty and that they were unlucky. [2]

This virus was called Stuxnet, and its inception 10 years ago caused many researchers to become scared at the real-world implications that Stuxnet showed. [1,2] Stuxnet proved that cybernetic attacks on critical infrastructures such as nuclear reactors are possible, and aren't just the type of attack that exists within the realms of theory or movie plotlines. [1] Stuxnet was of such high complexity and danger that security researchers at Symantec said they hope to never see anything like this again. [1]

How do attacks like these ever get remotely close to the systems that they target? What are their routes of intrusion? Was it difficult for the attackers to attack this way, or is it easy if you have the knowhow? This literature review aims to answer the question of how cybersecurity penetration is often orchestrated, and what researchers are attempting to do to analyse these attack vectors and deny infection altogether before catastrophy can be inflicted.

2 Literature Review

2.1 What do researchers know?

Attack vectors, defined as the path used by an attacker to conduct an attack or to attempt to bypass system controls, [3] today are increasing as a result of the emerging nature of new technologies. As almost all of software developer's focus is put on developing software to solve a business problem, the security of this software becomes inherently neglected. The multi-contextual nature of software and its very large plethora of applications mean that the number of attack vectors attackers can choose to move through increase. These are commonly referred to in the same context as attack surfaces, known as the amount of ways an attacker can attack a given system. The greater the attack surface, the more vulnerable the system.

Stuxnet isn't fair to compare to the rest. It utilised 4 zero-day vulnerabilities, known as a vulnerability that was not previously known to the public, and used 2 stolen digital certificiates, a security mechanism used to secure communications between two machines over the Internet, [1] to complete its task of disrupting operations at Iran's illicit nuclear research facility. [2] This is not normal. Hackers don't reveal more than one zero-day unless they have to, and stolen digital certificates are almost never used in an attack due to the sheer difficulty of stealing the certificates physically from the companies to be capable of it. [2] A willing or unwilling contractor or insider for the Iran facility was likely used to carry out the attack, and another insider was likely used to steal the digital certificates from the certificate authority companies. [1] This was a marksman's job from beginning to end and does not compare to ordinary cyber attacks today. [1]

2.1.1 Software Vulnerabilities

This does provide us a glimpse into the common use of a **combination** of attack strategies to accomplish a goal. Here we see the use of zero-day vulnerabilities, through a type of attack vector known as software vulnerabilities, where bugged code in software is exploited. [2] For Stuxnet, vulnerabilities in antivirus software were exploited, along with system level vulnerabilities discussed in 2.1.2. During the virus' initial injection, it would scan the currently running antiviruses and system processes, read metadata like their version number, and decide which process to inject its malicious code into, optionally using one of two zero-day vulnerabilities for code injection that it contained. [1] The virus does this to elevate privileges so it can perform any action it desires on the target computer. [1] If no process was bypassable, the virus would become inert. [1]

Exploiting vulnerabilities like this is the most common way of breaching the security of a system. [3–5] The question then becomes, how do attackers find these zero-day vulnerabilities? While it's nice to know that this is one of the attack vectors attackers may take, not exploring the nature of these zero-day vulnerabilities would not describe software vulnerabilities to an acceptable extent and would keep it in a rather vague and mysterious light. Because of this, I believe its exploration is still within the scope of the review.

At first glance it appears that zero-day vulnerabilities exist because developing large, complex software is hard and error prone. While this is logically invariably true to some extent, it was found that this isn't the full truth. Software enjoy a honeymood period, [5] coined the "Honeymoon Effect" in a study in 2010, where it was found that at the weakest point in a software's life cycle, its initial release, security vulnerabilities were never usually found during this period. It was described that an attacker's increasing familiarity with the software results in the discovery of the first zero-day way after the wave of ordinary bugs in software, where subsequent zero-days take less time to discover. Counterintuitively, the paper showed that code reuse, a strong development philosophy, was the greatest contributor to the rate and number of vulnerability discovery, which would otherwise suggest that to be the reason why large software (along with their popularity with users) are a strong target.

Attackers find vulnerabilities in software by approaching features of an application with certain expectations that typically arise from their programming experience. [6] talks of Adobe Reader 5.0 as an example, where failing to validate user supplied information causes an error called a "buffer overflow" to occur. A buffer overflow is where a program tries to store more data in a buffer than intended. [7] The function that facilitates this overflow is only available in lower level languages such as C and C++, where lots of control is given to developers. Naturally, the ability to make these kinds of coding mistakes increases. Input validation is an example of a "kingdom" among other kingdoms such as API abuse, encapsulation and poorly used security features that are categories of mistakes that programmers make in software that open up abusable vulnerabilities. [6] An entire list of programming "sins" has been compiled alone to show how these common vulnerabilities easily manifest themselves within programs and what developers should look out for to stop them. [8] McGraw [9] talks of the same vulnerabilities: error handling, buffer overflows, etc. but also notes that Internet-enabled software and the increasing extensibility of software add this greater fuel to the fire of increasing vulnerability. [6,8] and [9] have shown that developers, even in perhaps thought to be safer languages like Java and C#, are creating common vulnerabilities in code every day, and [5] shows that code reuse only makes it worse as developers are under the falsehood that the code is safer and more reliable due to its maturity. To conclude the mystery of software vulnerability exploitation, [1, 3, 5, 6, 8, 9] demonstrate that attackers reverse engineer large, popular and complex software scanning for common security vulnerabilities in every major update of these software, discovering these lingering zero-day exploits, and selling them on the market [7] to the highest bidder or to stockpile for use themselves for use in attacks such as Stuxnet [1].

2.1.2 System Level Vulnerabilities

2.1.3 Spoofing Biometric Systems

The most prominent method of circumventing biometric systems, such as fingerprint scanners, iris scanners and facial recognition devices, is spoofing. [3] Spoofing is the presentation of an item, false data or a false biometric claiming to be legitimate, resulting in a system triggering a false positive and granting access. [3]

What some might consider to be reserved to James Bond movies is very much practical in real life. For example, the most basic form of spoofing appears to be lifting fingerprints and re-using them, which began research back in 1998. [3, 10] They found that 4 out of 6 devices tested were susceptible to these fake finger attacks. [10]

The next step up was the use of "gummy fingers" in 2002 [11], where finger sleeves were made from gelatine and fingerprints were imprinted on them. Their research showed a high acceptance rate from fingerprint readers that used optical (measuring the reflection of light [11]) or capacitative (the ability for something to store electrical charge) sensors, which were most fingerprint readers back then. [11]

Then two German hackers in August 2003 claimed to have developed a technique of using prints left behind on a scanner and converting them into a latex finger, reconstructing digital images of the prints left behind on scanners into three-dimensional replications of the fingerprint. [3, 12] The reliability of this information is called into question however. While discovered from [3] with 275 citations from this time of reading (1st March 2020), the source that article got it from [12] is from a website called Security Focus in 2003 with 7 citations. Further information or proof of the proceedings that [12] describes are incomplete, probably due to the age of the article and the limited nature of the Internet in 2003.

Other methods include playing back digital videos or images into iris sensors and facial recognition to gain access [3,13], where some demonstrations of attacking this way even included poking a hole in a photograph of an iris to trick the 'liveness' testing mechanism inside some iris sensors. Liveness is the primary security mechanism to counter spoofing. Put simply, liveness involves seeing whether whatever is being sensed is actually alive. [3] Considering spoofing isn't a type of payload, this is different from regular attack vectors, but it is still a way to attack a system. If any form of authentication relied on the use of biometrics, spoofing becomes an option to explore for attackers.

2.1.4 Payload Propagation

2.1.5 Removable Drives

2.1.6 Collusion and Coercion

The threat of being exposed for some sort of secret is a very effective motivator to make someone do something for you, such as giving up their biometric and access privileges [3] or inserting removable drives that carry a payload (see 2.1.4) into a confidential system for you (see 2.1.5). The reputation of a person can be damaged beyond repair with the right allegation, making this style of attack vector very powerful but difficult to pull off.

However, coercion doesn't have to be voluntary. Being tricked into opening infected files (2.1.4), being lured into visiting a malware propagating website (2.1.8) or having your login details stolen from an attacker's authentic-looking page (2.1.7) are all forms of involuntary coercion [2, 7], a promising attack vector. [3] and [7] both show that using people as a weapon can open up many vectors of attack within an organization, with both falling under the area of social engineering, described by Singer [2] as the manipulation of people to achieve a goal.

2.1.7 Phishing

2.1.8 Driveby

2.1.9 Botnet

When access to social media accounts has been established, attackers can impersonate someone's friend and use them as a kind of trojan horse to send spam to their friends. These links create a chain effect, adding more and more people to the botnet. [7]

2.2 What do they not know?

2.3 What has been researched and what has not been researched?

2.4 Is the research reliable and trustworthy?

2.5 Where are the gaps in the knowledge?

3 Conclusion

Summary and conclusions to take forward.

4 Bibliography

References

[1] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, p. 29, 2011. [Online]. Available: https://nsarchive2.gwu.edu//NSAEBB/NSAEBB424/docs/Cyber-044.pdf

[2] P. W. Singer and A. Friedman, *Cybersecurity: What everyone needs to know.* oup usa, 2014. [Online]. Available: https://books.google.co.uk/books?id=B88ZAgAAQBAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[3] C. Roberts, "Biometric attack vectors and defences," *Computers & Security*, vol. 26, no. 1, pp. 14–25, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740480600215X

[4] N. K. Ratha, J. H. Connell, and R. M. Bolle, "Enhancing security and privacy in biometrics-based authentication systems," *IBM systems Journal*, vol. 40, no. 3, pp. 614–634, 2001. [Online]. Available: https://www.researchgate.net/profile/Jonathan_Connell/publication/220353130_Enhancing_Security_and_Privacy_in_Biometrics-Based_Authentication_Systems/links/555a010508ae6fd2d8281b10/Enhancing-Security-and-Privacy-in-Biometrics-Based-Authentication-Systems.pdf

[5] S. Clark, S. Frei, M. Blaze, and J. Smith, "Familiarity breeds contempt: The honeymoon effect and the role of legacy code in zero-day vulnerabilities," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 251–260. [Online]. Available: https://doi-org.uoelibrary.idm.oclc.org/10.1145/1920261.1920299

[6] K. Tsipenyuk, B. Chess, and G. McGraw, "Seven pernicious kingdoms: A taxonomy of software security errors," *IEEE Security & Privacy*, vol. 3, no. 6, pp. 81–84, 2005.

[7] J. Jang-Jaccard and S. Nepal, "A survey of emerging threats in cybersecurity," *Journal of Computer and System Sciences*, vol. 80, no. 5, pp. 973–993, 2014.

[8] M. Howard, D. LeBlanc, and J. Viega, "19 deadly sins of software security," *Programming Flaws and How to Fix Them*, 2005.

[9] G. McGraw, "Software security," *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80–83, 2004.

[10] D. Willis and M. Lee, "Six biometric devices point the finger at security," *Computers & Security*, vol. 5, no. 17, pp. 410–411, 1998.

[11] T. Matsumoto, H. Matsumoto, K. Yamada, and S. Hoshino, "Impact of artificial" gummy" fingers on fingerprint systems," in *Optical Security and Counterfeit Deterrence Techniques IV*, vol. 4677. International Society for Optics and Photonics, 2002, pp. 275–289. [Online]. Available: http://cryptome.org/gummy.htm

[12] A. Harrison, "Hackers claim new fingerprint biometric attack," *Security Focus*, 2003. [Online]. Available: https://www.securityfocus.com/news/6717

[13] C. Body, T. Lisa, K. Jan, and Z. Peter-Michael, "Biometrie (translated from the original german by robert w. smith) c't magazine 2002; 114."