EAI Endorsed Transactions on Security and Safety

Editorial ICST.ORG

Preface to special issue on miscellaneous emerging security problems

Kai Chen[1,*], Hs l Qf_m[2]

[1]Institute of Information Engineering, Chinese Academy of Sciences
[2]Zhejiang Gongshang University

Abstract

An introduction to the key topics and challenges in Botnet, buffer overflow vulnerability, privacy-preserving and others.

Keywords: security and safety, botnet, buffer overflow, privacy-preserving

Received on 01 June 2015; published on 05 October 2015

doi:10.4108/eai.5-10-2015.150475

1. Introduction

Security and safety become more and more important in cyber network. They impact not only the national critical departments (e.g., a state council) but also individuals' daily life. In the area of security and safety, botnet, buffer overflow vulnerability, privacy-preserving, and authentication are four hot topics impacting both those departments and individuals. With the billions users of smartphones, the threats become even severe.

In these areas, the threats from botnets are reaching an alarming level. A botnet is a collection of programs that communicate through Internet and perform specific tasks such as sending spam emails and performing DDoS attacks. The tasks are given by commands from an attacker via C&C (Command and Control) channels on Internet, which makes the programs as bot instances. The hosts containing the programs could be personal computers, workstations in enterprise networks or even sensitive servers in governments. In addition to those victims (e.g., websites in a DDoS attack performed by a botnet) directly attacked by bots, these hosts also become victims in an attack. Nowadays, with the rapid growth of smartphones, more and more botnets use smartphones as bots. Considering that there are billions of smartphones, botnets have much more targets than those in PC era.

A botnet is usually very good at concealing itself. The bots in hosts may not do anything harmful directly to the hosts. Sometimes, the only goal for these bots may be waiting for a command from a remote server to connect to some websites (e.g., for DDoS attacks). This connection can also be normally performed by an ordinary user, which is therefore difficult to detect. The bots could also reside in the host for a very long time waiting for a command. In this waiting period, nothing happens. In other words, no malicious behaviors can let users or anti-virus detectors aware the existence of the bots. Also, the waiting time can be too long to get noticed. Anti-virus software may also try to detect bots using their signatures. However, bots can also be highly customized (e.g., through obfuscation) to evade the detection. Moreover, before the awareness by those anti-virus software, enough time is given to the bots for their updates. Thus, to detect the malicious bots is generally very difficult from host sides.

Another direction for botnet detection is to detect botnet traffic. A basic idea is to use misuse detection. For example, some approaches generate signatures of known malicious traffic (e.g., using malicious hosts). But they cannot detect new C&C traffics. Some approaches classify traffic flows by protocols, length sequences of packages, and the encoding of URLs. However, bots' traffic can be disguised as normal ones especially when popular network protocols such as HTTP are used. Noises can also be injected to

[*]Corresponding author. Email: chenkai@iie.ac.cn

evade detection. Some approaches assume that C&C traffic connects to the same location within varying time windows. So they detect botnet through finding repeated combinations of traffic destinations. These approaches work when botnets like to use centralized architectures. Using these architectures, all the bots contact with one (or a few) C&C server(s) which distributes commands. This is easy for a botnet to implement, and is also easy to be detected. When the server is compromised, the botnet is destroyed.

However, botnets are becoming more resilient to detectors nowadays. They can structure their C&C channels in different ways. Recently, attackers start to build botnets on a pear-to-pear (P2P) architecture. Any node in the botnet could be a server to send commands, which makes them even more difficult to detect and take down. Anomaly detection can address the problem by observing deviations from normal traffic. However, this kind of detection usually needs a learning period. The malicious C&C traffics in this time period cannot be identified. Moreover, the "normal" traffic can be contaminated which makes future C&C traffics not easy to detect.

To solve this problem, in this special issue, Burghouwt, Spruit and Sips present a network-based anomaly detection approach. This approach estimates the trustworthiness of the traffic destinations. The flows from human input, prior traffic from a trusted destination, or a defined set of legitimate applications are considered as normal traffic. In this way, the approach can detect zero day malicious traffic destinations in real time. Even if the traffic is encrypted or uses normal popular protocols such as HTTP, the approach can still detect it without being noticed.

Vulnerabilities in software are highly related to botnet. To stealthily plant a bot into a system, lots of attackers will choose to exploit a vulnerability. In this process, nothing special happens when a bot is planted. Among different kinds of vulnerabilities, buffer overflow is one of the most dangerous. By overrunning the buffer's boundary and overwriting adjacent memory, attackers can leverage a carefully constructed input to overwrite important memory units (e.g., return address to a function). In this way, attackers can let a program run code in arbitrary memory addresses given by input data (constructed by attackers) to include arbitrary commands (e.g., downloading and executing the bot). When the bot spreads in the network, it can also exploit the vulnerabilities to infect more systems. In this process, those target systems look normal, which makes the detection quite difficult.

Buffer overflow vulnerabilities not only allow attackers to implant bots, but also give attackers opportunities to execute arbitrary commands. When the software with the vulnerabilities has root privileges, it becomes even more dangerous. All the data in the system may be exposed to attackers. With the rapid growth of smartphones, a vulnerability in system will impact billions of users. Even for a single popular app with such vulnerabilities, it may impact millions of users.

To deal with such attacks, researchers can either identify and fix all buffer overflow vulnerabilities in software or do a defense against such attacks. Identifying all buffer overflow vulnerabilities (especially in commercial software without source code) is extremely difficult. In most cases, the vulnerabilities can only be triggered by specific inputs. To find such inputs, a tester may use white-box testing to identify those vulnerabilities. In this process, he can check whether each path contains an instruction which overflows a buffer. However, for an instruction in different execution paths, the instruction may or may not trigger a vulnerability. To traverse all the execution paths in software is very time-consuming and almost impossible. He can also leverage black-box testing, which feeds programs with different inputs and observes the abnormal running states of programs (e.g., crashes). The black-box testing avoids checking the running states of the program, which seems more efficient than white-box testing. However, whether a vulnerability can be triggered highly depends on the inputs. It is impossible to enumerate all the inputs for the software. Current approaches combine white-box testing with black-box testing for efficient detection. But they still cannot find all vulnerabilities. Once there is one vulnerability missing, it could be exploited by attackers.

Another direction is to do a defense against the attacks. Some approaches dynamically check operations on buffers to see whether the operations exceed the boundaries of those buffers. But the checking process will impose high overhead against normal execution. A random value, or canary, is added to the memory unit right adjacent to important memory units in stack such as return address or saved frame pointer. Based on the design, once the buffer is overflowed to overwrite the important memory units, the canary will also be overflowed, which will warn the program (usually through exceptions) that an attack is happening.

Data Execution Prevention (DEP) marks important memory regions (e.g., stack) as unexecutable. Once a program tries to execute instruction inside these regions (usually caused by injecting code through buffer overflow), an exception (e.g., on hardware level) will be triggered. However, attacks based on Return-Oriented Programming link and execute a sequence of small code snippets (called gadgets) which are not in stack. In other words, attackers can let the return address point to an instruction in code area (e.g., the program itself or dynamic linked libraries). The instruction is chosen by attackers in advance, which is followed by a return instruction. After the instruction is executed, the return instruction will divert the program to another instruction according to the return address in stack which is given by attackers (still through injection). In this way, attackers can inject the program with a sequence of addresses to let the program run any code without the detection by DEP.

Address Space Layout Randomization (ASLR) is another approach to prevent shellcode from being successful. It randomizes the location of executables and libraries in memory or even in-memory structures. When attackers try to chain the gadgets that they find before attacks, the gadgets may not be in the original places in memory as those in attackers' computers. In this way, attacks can be prevented. However, ASLR must maintain page alignment (4KB on x86) within limited memory address space (typically less than 2GB on 32-bit x86), which also limits the spaces for target code. The correct address could be guessed by attackers depending on how often they can try. Combining two or more defenses (e.g., DEP and ASLR) will make the attacks even harder. But new attacks such as Just-in Time (JIT) Spraying can still bypass them.

To make a better defense, in this special issue, Krugel and Müller propose a compiler-level protection. This protection separates a stack to two stacks (i.e., control and data stacks). It protects sensitive data (e.g., return addresses and saved frame pointers) on a separated stack (i.e., control stack). In this way, when a buffer is overflowed, only the data in data stacks will be overwritten. The sensitive data will still keep the original values (in data stack) in this overflow. The authors implemented this idea on LLVM compiler infrastructure and made detailed evaluations on it. While protecting the sensitive data, this approach imposes little overhead on the running performance.

Privacy is always a major issue nowadays. It does not mean that the information related to individuals cannot be revealed to others, but it does mean that people has the right (ability) to decide who could collect, store, or use the information. However, many of existing web applications are against users' privacy, i.e., to complete their functionalities', they have to record user's identity information, action histories, or behavior patterns. One representative is web metering.

The web metering is a web application where interested enquirers can obtain the evidence of the number of visits done by users to the website. This evidence is the crucial factor to decide the price of advertisements on the websites. As we know, most websites nowadays earn their money not from the users who visit them but from the advertisements on the websites. The data used as the evidence is required to be non-repudiated, and can be linked to user's identity information or other actions.

The existing web metering schemes can be classified into the following three categories: user centric, web server centric and third party centric. In the first category, several cryptographic primitives, including digital signature, hash chain, and secret sharing, are applied. Although the applied cryptographic primitives make the evidence non-repudiated, the user's identity will be revealed to verify the validity of the evidence.

In the second category, the web server makes use of several techniques to obtain the non-repudiated evidence, including e-coupon, solving computational complexity problems, and audited hardware box. However, the resultant schemes cannot guarantee that the obtained data are always non-repudiated if the web server is not always honest, let alone the user's privacy.

In the third category, the data from the user side always go through the third party; hence, the user's privacy is not protected from the third party.

It seems that designing a privacy-preserving web metering scheme is a quite difficult task. Inspired by the hardware-based security systems, a new hardware-based web metering scheme is proposed in this special issue. There are three entities in the proposed web metering scheme: user, web server, and audit agency. To complete the web metering, the web server should obtain the non-repudiated data from a new key of the user. The certificate of the new key is generated by the audit agency after validating the hardware on the user side. To protect the user's privacy, the certificate is based on some anonymous authentication methods. More details of the new web metering scheme can be found in the third paper named "Towards Privacy-Preserving Web Metering Via User-Centric Hardware" by Alarifi and FernÍćndez. In the paper, they also analyze and compare the existing web metering schemes and show the gained privacy benefits of the proposed scheme. Furthermore, the proposed scheme supports different security countermeasures and users' privacy settings, such as security w/o privacy.

One of the basic techniques used in the paper by Alarifi and FernÍćndez is the anonymous authentication. Till now, there existing many cryptographic primitives providing anonymous authentication, including blind signature, group signature, ring signature, direct anonymous attestation, anonymous credential, and so on.

In a blind signature scheme, the signer will generate a signature without knowing the corresponding message, while the signature requester can obtain the pair (signature, message). With this pair, the requester can be authenticated by others in an anonymous way. In the real execution, the signer should make sure that the one knowing the corresponding signed message is an authenticated user.

Group signature can provide the limited anonymity on the authentication. In particular, every group member can sign messages on behalf of the group, and nobody outside of the group can reveal the identity of the signer, while the group manager can trace back to the signer. In other words, the signature is anonymous except the group manager.

Ring signature is quite similar to group signature, but without the group manager. Everyone can build up a group without agreements from others, and can also sign message on behalf of the built group. Unlike group signature, the resultant signature in the ring signature scheme is fully anonymous.

Direct anonymous attestation has been adopted by the Trusted Computing Group to protect the privacy of the Trusted Platform Module (TPM) platform. The key of direct anonymous attestation is to use a zero-knowledge proof to show the validity of the credential provided by the TPM platform without violating the TPM platform's privacy.

Anonymous credential is an anonymous system where the user can obtain credentials from an organization, and then at some later pint, s/he can be authenticated by others using the credentials without revealing her/his identity.

The careful reader may find that all the above cryptographic techniques for anonymous authentication can be used in the privacy-preserving web metering, or other applications where privacy and non-repudiation are both desired.

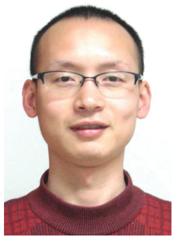
References

[1] Kopka H. and Daly P.W. (2003) *A Guide to LaTeX* (Addison-Wesley), 4th ed.

[2] Lamport L. (1994) *LaTeX: a Document Preparation System* (Addison-Wesley), 2nd ed.

[3] Mittelbach F. and Goossens M (2004) *The LaTeX Companion* (Addison-Wesley), 2nd ed.

KAI CHEN (chenkai@iie.ac.cn) obtained his Ph.D. degree from Graduate University of Chinese Academy of Sciences in 2010. He was a postdoctoral researcher at Pennsylvania State University from 2012 to 2014, and Indiana University from 2014 to 2015. Now, he is an associate professor in Institute of Information Engineering, Chinese Academy of Sciences. His research interests include system security, program analysis and network security.



JUN SHAO (chn.junshao@gmail.com) obtained his Ph.D. degree from Shanghai Jiao Tong University in 2008. Soon after, he was a postdoctoral researcher at Pennsylvania State University until 2010. Now, he is an associate professor in Zhejiang Gongshang University. His research interests include applied cryptography and network security.