Abstract

Client programs are omnipresent in our digital age. Especially, web browsers are used by an enormous number of users for various tasks. These tasks include information gathering, social media activities or communication with each other. As the popularity of web browsers has grown, attackers shifted their attention towards this kind of client-side software to compromise systems. Because of their huge code base and tremendous complexity, exploitable vulnerabilities exist in a vast number in these programs.

In this thesis, various impacts of memory corruption vulnerabilities in client-side software are investigated from an offensive and a defensive perspective. The exploitation process of vulnerabilities in web browsers may obey certain steps, carried out subsequently. Usually, an adversary needs to find information about the address space of the vulnerable program. This important step is called information leak or memory disclosure. Once the attacker has gained enough knowledge about the address space of the program, she is able to hijack the control flow. This thesis considers information leaks and control-flow hijacking from attackers’ and defenders’ viewpoints.

We extend the technique of information leaks with a behavior in web browsers which was not known to that extend before (crash resistance). Therefore, the program is kept alive, although it should terminate due to critical memory errors, such as an illegal read access. This allows to evaluate defenses from an adversarial perspective, which promise to keep address space information of the program a secret from attackers. Information leaks are also approached from a defensive perspective. To detect this step of an exploit we introduce a concept for script engines, i.e., JavaScript. Therefore, two simultaneous processes of the same program are executed in parallel and their execution flow is synchronized. As we enforce a different address space layout in both of them, a memory disclosure manifests itself differently in both processes and can be unveiled.

This thesis also addresses control-flow hijacking. Code reuse is currently the most common method to perform arbitrary computations after gaining control of the execution. For an adversary it is important to assess the quantity of code she can reuse. Hence, we introduce a framework to help evaluating specific control-flow integrity defenses. Therefore, we attempt to analyze a given program in an architecture independent way to maximize the amount of reusable code which conforms to CFI policies.

Attackers are able to hijack the control flow in browsers with vtable hijacking. This thesis approaches this widely used, offensive technique from a defensive perspective. Virtual function tables (vtables) injected by attackers into the address space differ from real vtables. By using various heuristics and techniques we are the first to show that a vtable-hijacking mitigation for binary-only code is possible.