Executive Summary –Flasm

Chanakya Gaur, *cgaur1,* Sagar Wani, *swani1*, Prashanth Venkateswaran, *pvenkat6*

***Abstract*— Flasm is a disassembler for SWF including timelines and events. It also can be used to make optimizations on the disassembled code which are directly changed to the original SWF. This executive report is based on discovering format string vulnerabilities and exploiting them.**

***Index Terms*—Flasm, SWF, format string, vulnerability, buffer overflow, flash**

I. Introduction

Through this report, our aim is to explain the process followed to discover and execute format string vulnerabilities. According to OWASP, a format string exploit occurs when the input sting is evaluated as a command by the application. Using this exploit, the attacker could execute code, cause a segmentation fault or read the stack to make modify the behavior of the application. [1]

The threat assessment is divided systematically by first identifying the assets. Further, an architecture diagram is built which makes decomposing the application, the next step easier. Then, the threats are identified, documented and their risk values are calculated. The aim of this report is to reflect upon the process and result of the assessment.

II. Exploit Generation Process

Threat modeling process is used to analyze the security of a software that enables you to identify, quantify and address the associated security risks [1]. The inclusion of this process in SDLC will ensure that software is built with built-in security.

This process consists of six components which can identify and document all possible threats, they are:

1. Identifying Assets
2. Architectural Overview
3. Software Decomposition
4. Threat Identification
5. Threat Documentation
6. Threat Prioritization

A. *Identifying Assets*

An asset is something an attacker wants to access, control, or destroy [2]. To understand the assets of Knoqueror, we installed Konqueror on our systems and used it for a day. Then we inspected the source code and modelling it with the basic structure of a web browser, listed down the assets of Konqueror.

Konqueror has a variety of assets ranging from reputation and users as abstract assets to saved passwords and cookies data as physical assets. The full list of identified assets can be found in threat modelling report.

B. *Architectural Overview*

The architectural overview is the graphical depiction of all the functionalities a software performs. To generate an overview, we used the Microsoft Threat Modeling Tool.

Using the list of assets generated and the general architecture of the browser, we divided the functionalities into blocks. This helped us to understand how communication took place internally and how most of the features functioned. The overview also helped us to graphically represent all the components both internal and external which could prove to be a threat.

Using trust boundaries helped us to understand how the developers have tried to protect the assets and how maintaining these boundaries was necessary. It was necessary to ensure that separate modules of the software were separated by trust boundaries. Third party extensions and plugins which cannot be trusted were separated from the main component by trust boundaries. From the architectural overview, we have inferred that the browser engine, that connects to the other components of the browser is the center of the software. Additionally, the UI backend, persistent data and network interface need to be separated by trust boundaries and any data transfer should be protected.

The architectural diagram can be found in the threat modelling report.

C. *Software Decomposition*

In software decomposition, the software is broken down into functional components which are individually inspected and possible security factors are considered and security mechanisms are inspected.

Once we had broken down Konqueror into its functional components, we studied the source code and researched online to understand what are the common vulnerabilities of that components. We co-related that data along with the source code to list down what the possible security vulnerabilities could be and what aspects of the code should

Sagar Wani, Graduate student, Johns Hopkins University (e-mail:swani1@jhu.edu).
Chanakya Gaur, Graduate student, Johns Hopkins University (e-mail:cgaur1@jhu.edu).
Prashanth Venkateswaran, Graduate student, Johns Hopkins University (e-mail:pvenkat7@jhu.edu).

be checked for them. Reading the documentation of Konqueror helped us a lot to understand the basic flow of data within classes and how different tabs are run.

Looking at the common vulnerabilities online, we realized that the developers of Konqueror had ensure a decent level of security as most of the common vulnerabilities had been avoided. However, we found a list of vulnerabilities that could easily allow a user to launch a Denial of Service attack. For example, on inspection of the HTML parser, we realized that an overly long value in either a table or color tag kept in the webpage by an attacker could shut down the parser and in-turn the browser.

The complete software decomposition can be found in the threat modelling report.

D. *Identification of threats*

Attack, refers to a major science of threat modeling – the discipline of researching how attack patterns can possibly exploit software vulnerabilities and poorly designed countermeasures. [3]. In a cyber-attack, combatant seeks to reduce the effect of information by taking computer systems offline, or at least somehow blocking access. [4] Owing from the statements above, the need for identification of threats and being one step ahead of it is imperative. Up until this section, we have identified the assets that we trying to protect and also an architectural model of the Konqueror Web Browser.

To identify the threats to Konqueror Web browser, we have used the STRIDE [2] process. Based on this process we could find vulnerabilities and categorize the threats against our application within the STRIDE parameters which are Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege. An easy way to go about thinking the STRIDE model is to assess the application acronym STRIDE. An Information Disclosure STRIDE approach is demonstrated in Fig 1.

Property	Threat Definition	Examples	Mitigation
Information Disclosure	an application fails to properly protect sensitive information from parties that are not supposed to have access to such information in normal circumstances.	- attackers can send requests to the system they are attempting to attack in order to gather more information about it.	- Make sure that browser does not send information that revel internal methods or information.

Fig 1: Information Disclosure STRIDE Approach

We also used the Attack trees and Categorized threats lists to find a more streamlined approach to identify the potential threats to our system, an example Denial of Service attack tree is described in Fig 2.

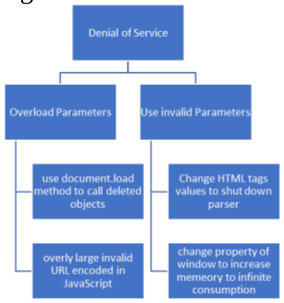
Denial of Service
Overload Parameters
Use invalid Parameters
use document.load method to call deleted objects
overly large invalid URL encoded in JavaScript
Change HTML tags values to shut down parser
change property of window to increase memeory to infinite consumption

Fig 2: Denial of Service Attack tree

We also used the attack patterns to put ourselves in the shoes of the attacker and devised steps that a potential avenue an attacker would take to compromise the Konqueror Web Browser.

Repudiation is claiming you were not responsible for what happened. People can repudiate knowingly or otherwise. Given the increasing knowledge often needed to understand the complex world, those honestly repudiating may really be exposing issues in your user experiences or service architectures. [2]

A complete list of all threats identified for Konqueror Web browser can be seen in Appendix D.

E. *Documentation of Threats*

Now that we have identified the threats, it is also beyond doubt that we also need to document the Threats. Documentation of threats is necessary for the mitigation of the threats found in step D.

Up until now, we have broken the stages of threat modeling in a structured approach to ensure that no stone is left unturned. This step is an aggregation of all of these functional blocks from Step A through D.

We used the categorized threat list system to get the countermeasures as well. After weighing the risks in terms of the difficulty, effect to the user and a variety of important checks, we have assigned a Risk level to all the threats recorded in step D.

F. *Prioritizing of Threats*

Apart from documentation of the threats, we also need to prioritize them. Prioritizing the threats have various advantages. Once a threat or a vulnerability is identified in a system, resources also needs to be assigned to fix these vulnerabilities. Obviously, it makes sense to prioritize the threats that are high risk rather than the low risk.

A high risk would be cookie stealing or hijacking since the effect of this threat is very high to the user since it most likely be of monetary loss for the victim. Whereas a Low Risk would be a URL address bar can be spoofed to hide some part of the URL. This vulnerability is a spoofing threat that we cause mild discomfort to the user’s browsing experience causing a reputation asset loss to Konqueror Web Browser.

For corporations, Risks prioritization is done as a factor of costs incurred as well. High risk threats are always given the most resources whereas the low risk threats are weighed on the basis of costs it the fix will take over time.

A full list of prioritized and documented threats will be found in Appendix E.

III. Conclusion

Based on this experience, we have learnt on a very structured approach on how to threat model an application such as the Konqueror Web browser. This process also sheds light on the various perspectives of how a software needs to be developed, tested and released. The perspectives mentioned above are the developer’s viewpoint, the attackers viewpoint and the user’s viewpoint. In the developer’s shoes, we have been able to identify that it is necessary to inculcate security in the software development lifestyle itself. The attackers perspective has taught us to think in the mindset as an external entity with a malicious intent and probing the application. This helps us find the flaw and attack surfaces in our application. Ultimately identifying, documenting-prioritizing and mitigating the threats.

IV. APPENDIX

A. Identifying assets – Konqueror Threat Modeling Report, Pg. 1

B. architectural overview – Konqueror threat modeling report, Pg. 2

C. software decomposition the application – Konqueror threat modeling, PG. 3,4

D. threat identification – Konqueror threat modeling, pg. 5,6,7,8,9,10,11

E. threat documentation – Konqueror threat modeling, pg. 12

V. References

VI. 1.The Internet, “Format string attack”, OWASP
2.Adam Shostack, “Threat Modeling: Designing for Security”, John Wiley & Sons
3. Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis By: Tony UcedaVelez; Marco M. Morana
4. Intorduction to Cyber-Warfare, **By:** Paulo Shakarian; Jana Shakarian; Andrew Ruef
5. The Internet, “Konqueror”, Github