Defying Logic

Theory, Design, and Implementation of Complex Systems for Testing Application Logic

Rafal Los, Prajakta Jagdale

HP Software & Solutions

Background

The testing of applications for security defects, or vulnerabilities, has evolved tremendously over the past decade and a half or so. What first started out as a completely manual and painstaking process has evolved over the years to utilize technology and automation to achieve greater coverage, more consistency, and better overall analysis capabilities with significantly less effort. One area where technology has failed to assist up until this point is with the identification of what are commonly referred to as *logic vulnerabilities* in applications. Uncovering flaws coded in the business processes and flow of an application using anything but manual processes has been an impossible task mostly because of the nature of the types of intelligence that would be required. Creating a system or a *testing framework* which can appropriately enable a previously entirely manual process is tricky- and while fully automating the finding of logic flaws may still be beyond our reach the research and ideas presented herein are steps forward to that direction.

Foundations

To fully understand the ideas and research presented here it is imperative to understand the foundational concepts upon which these thoughts are built. First, let us define clearly why we feel we need a new taxonomy. A next step is to define exactly what a logic flaw is. Finally we must define the taxonomy of logic flaws and their relevance to our work and advancements.

First, to understand our research it is important that we make clear our motivations for looking at logic flaws and creating a new taxonomy for classifying their different attacks. Acknowledging that there are already good taxonomies of application vulnerabilities (the OWASP Top 10, for example) in existence today, our taxonomy is different and necessary because it drives into previously unexplored territory. Identifying and classifying vulnerabilities at the heart of business processes which applications are built upon is not currently done in any existing classification or taxonomy –our goals is to eventually merge into one of the prevalent classification systems when appropriate.

Additionally, in order to fully appreciate why this research is being presented we must introduce you to our way of thinking. Black-box application testing technology has undergone several major evolutionary steps from being able to simply crawl → record → replay/manipulate to the modern way of assessing web-based applications which requires sequencing, client and server-side instrumentation and session handling and manipulation capabilities. These advancements, while leaps and bounds above what we started with many years ago, still confine testers to what black-box testing technologies do well, pattern-match. Our proposal is that there is a necessary evolution that is required for these enabling technologies to be able to address the existing gap of logic issues that are slipping by un-noticed by tool and analyst alike. Even when a site is free of many of the issues that make up lists like the OWASP Top 10, there may still be logic flaws present –which, for example, can have an even bigger impact than a Cross-Site Scripting (XSS) vulnerability.

*For the sake of this and further discussion we should identify a *vulnerability* as a subset of the more general class, *defect,* which is a synonym for the additional general term *flaw*.

So what is a *logic defect* in an application? A logic defect, as defined by our proposal, is defined as such:

> *"A defect that exposes the component business processes or component flows to manipulation from the attacker perspective to achieve unintended and undesirable consequences from the design perspective; without disrupting the general function or continuity of the application."*

This type of defect directly affects the business process of an application and can go completely undetected by even the most sophisticated network or application-level defensive systems. The danger of this type of defect is directly proportional to the impact of the business process being manipulated. In a simple example where an attacker can manipulate the price of an item by simply changing the parameter inside of an application can lead to large financial loss when a $100,000.00 item is purchased for a fraction of the cost rapidly, before the system has a chance to discover the issue. By attacking the way the application "*works"* the attacker can gain a substantial advantage over the application under attack. Manipulating a business process is significantly more difficult to detect (if even at all possible) using existing detection technologies like Intrusion Prevention Systems (IPS) or Application Firewalls (WAF) so this gives the attacker the element of time in which to operate and perpetrate the attack.

To clearly define the types of logic defects in applications, a taxonomy was created and defined. Two main categories of logic defects were identified as follows:

- Privilege Manipulation - Flaw based on broken/incomplete authentication/authorization mechanism
- Transaction Control Manipulation - Flaw based on broken business process continuity allowing for manipulation of intended business process/flow

Each of these two types of defects has unique properties, and can be tested using varying new methods being researched and presented from our research. The underlying message here is that it is possible to test for *logic defects* as security issues, once they've been understood, classified and properly investigated.

Research

The research we at the HP Web Application Security Research Group (WSRG), are putting forth deals with how to classify, and then approach these types of attacks against application *logic* using an automated testing approach. Our goal is to facilitate a framework for, initially, a repeatable and scriptable way of testing application logic for security flaws. The broader goal is to enable a wider adoption of application logic testing beyond the sparse manual testing that is done today. Over time, it is our hope that the framework will evolve into a community adopted and standardized, accepted methodology for testing application logic –which benefits novice as well as advanced testers alike.

Our approach is to create a programmatic framework for defining the business processes (application execution flows (See "Into the Rabbithole", OWASP AppSec USA 2010, Los)) and components which can be manipulated depending on the type of logic defect being tested for and the expected outcome – then having a way to identify when a test has been successful.

There are several necessary components, as identified in the above statement, which need to be accounted for in any testing framework of this nature. First we must know which type of application logic defect we are attempting to test the application for. This requires at least minimal knowledge of the application structure

and function from a user's perspective. This knowledge is used to structure the type of test that will be conducted, and the components accessible to the tester within the framework. Each type of defect has unique requirements and will be tested using unique methods, for example, attacks based on privilege manipulation logic defects require knowledge of the specifics of application roles, authorization, and session parameters, whereas a transaction control manipulation attack aims to disrupt and alter the flow of a transaction focusing less on data and more on the sequence of events and an outcome. Each of these are completely different and the resultant tests are proof of this.

Nearly everything useful a tester needs to understand and bring to the testing framework stems from properly identifying the appropriate type of logic defect being tested for. From the way a particular application framework handles authentication session tokens, to which parameters are used for managing user actions and application interaction –each of these is critical to getting a successful test created and executed within the framework.

Conclusions

The ideas presented herein and during the talk are an attempt to address a virtually untouched segment of security testing of applications –application logic testing. These vulnerabilities, we strongly feel, are just as dangerous as the garden variety Cross Site Scripting (XSS) or SQL Injection, but are more difficult to detect in the wild when being executed against your application. This combination of being unaddressed and difficult to identify make this area of ongoing research critical to the security community and the overall decrease in the threat posture of internet-connected applications.