Security Analysis of FileZilla Server Using Threat Models

Michael Sanford, Daniel Woodraska, Dianxiang Xu
National Center for the Protection of the Financial Infrastructure
Dakota State University
Madison, SD 57042, USA
{michael.sanford, dcwoodraska, dianxiang.xu}@dsu.edu

Abstract— FTP is a widely used protocol for working with remote file systems. Various FTP implementations have had security problems reported as late as 2010. There lacks a systematic analysis of FTP security. In this paper, threat models are built to provide a systematic coverage of potential security attacks against an FTP server. Security tests are then generated from the threat models and applied to FileZilla Server, a popular FTP server implementation. When FileZilla Server is properly deployed, it holds fast against our security attacks. To further evaluate the effectiveness, the security tests are used to exercise a number of security mutants of FileZilla Server where various vulnerabilities are injected deliberately. The security tests have detected all but one of the injected vulnerabilities. This indicates that the threat model-based approach to security analysis of FileZilla Server is effective.

Keywords- Security testing, FTP, threat modeling, threat tree, mutation testing

I. INTRODUCTION

FTP (File Transfer Protocol) is a widely used method for working with remote file systems. While FTP is an old protocol, various implementations show security problems that were reported as late as 2010 [1-3]. Using FTP across an untrustworthy network (e.g., Internet) can still expose an organization or business to security risks, yet there is no systematic analysis of FTP security to help understand the risks and mitigate them.

This paper presents an approach to FTP security analysis using threat models. We build threat models against most of the FTP services using the STRIDE threat classification system [4]. STRIDE stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privileges [4]. It provides a systematic coverage of the potential security attacks against an FTP server. We generate security tests from STRIDE threat models and apply them to FileZilla Server, a popular open source FTP implementation. When FileZilla Server is properly deployed, all of the attacks against it fail. When the deployment uses the default settings, however, one of the security tests leads to a successful denial of service attack.

To further evaluate the effectiveness of our approach, we created a number of security mutants of the FileZilla server program by injecting various vulnerabilities into the code. Then we ran the security tests against each of the mutants. A mutant is said to be killed if one of the security tests is a successful attack. The security tests have killed all but two of the mutants. This demonstrates that the threat model-based approach to security analysis of FileZilla Server is effective.

II. THREAT MODELING OF FTP SERVER

Security threats are potential security attacks. Threat modeling usually consists of four steps: modeling system functions, specifying security threats, ranking threats for risk analysis, and mitigating threats [4]. Functional modeling helps identify the system assets (e.g., data) that need to be protected and the entry points that an attacker might exploit. As this paper focuses on the security analysis of an existing system, we are not concerned with threat ranking or mitigation.

Table 1. Partial list of FTP Services vs. STRIDE Threats

<table>
<thead>
<tr>
<th>Service</th>
<th>S</th>
<th>T</th>
<th>R</th>
<th>I</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Change directory</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Get current directory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change to passive mode</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Put file to server</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Get file from server</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create directory</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete a directory</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete a file</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute commands on server</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rename a file</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Append to a file</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-service specific</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The FTP standard has evolved over many years via RFCs (959, 1123, 3659, 4217) published by the Internet Engineering Task Force (IETF). FTP operations include login, change directory, list directory, get current directory, change to passive mode, put file to server, get file from server, create directory, delete directory, delete a file, execute commands on server, rename a file, append a file, etc. These operations are listed in the first column of Table 1. To provide a systematic coverage of the potential attacks against the FTP services, we identify the STRIDE threats with respect to each service. We have also considered the vulnerabilities reported by various
sources, including but not limited to SANS [1], US-CERT [3], Security Focus [2], Vulnerability Scanning [5], Offensive Security’s exploit database [6], the now extinct website milw0rm.com (went down during our research) [7], and FileZilla’s own bug report system [8].

Attack (threat) trees [9][10] are used to visualize the threat models. An attack tree shows the steps an attacker would go through to compromise a system. The root of an attack tree specifies the ultimate goal of an attack; child nodes represent the sub-goals of their parent node (or the steps to achieve the parent’s goal); and leaf nodes describe the primitive attack conditions or actions. An “AND” relation between two or more nodes means that all of the sub-goals or conditions in these nodes must be satisfied in order to achieve the parent’s goal. An “OR” relation means that the parent’s goal is reached when one of the sub-goals or conditions is satisfied [10].

Figure 1 shows denial of service attacks that prevent a legitimate user from logging in. The first attack attempts to fill up the login table that keeps track of the user connections. The second attack fills up the receive buffer of the server to the point to of crashing. The protocol requires a carriage return to be sent at the end of the command. Otherwise, it just keeps buffering the data coming in until it overflows.

III. SECURITY TESTING OF FILEZILLA SERVER

After the attack trees have been constructed, we conduct security testing as follows. For each attack tree, we first generate all attack paths (also called test sequences). Then they are converted into security tests. Consider the attack tree in Figure 2. A is achieved if B or C are satisfied. B can be satisfied by 1 and 2 and node C by 3 and 4. Therefore, the attack paths are <1, 2> and <3, 4>. In our approach, we generate the test sequences from an attack tree automatically.

Algorithm 1: concatSequences
Function: combinational concatenation of sequences
Input: {S[0], S[1], ..., S[n]}; Output: S[0]×S[1]×...×S[n]
Declare: S is a set of sequences
1. begin
2. if n=0
3. then return S[0];
4. S = Ø;
5. for i=0 to S[0].size-1
6. for j=0 to S[1].size-1
7. S ← S ∪ {<S[0][i], S[1][j]>}
8. endfor
9. endfor
10. if n=1
11. return S;
12. else /*n>1*/
13. return concatSequences(S, S[2], ..., S[n]);
14. end

Algorithm 2 describes how to generate test sequences from (the root of) a threat tree, where “∪” in line 12 refers to union of sets and “×” refers to combinational concatenation of sequences in Definition 1 and Algorithm 1.

Algorithm 2: genSequences
Function: Generate attack paths from a threat tree node.
Input: threat tree node; Output: attack paths
Declare: childNodes is the list of child nodes of a node; childNodes[i] is the i-th node in childNodes
S[i] is a set of sequences
1. begin
2. if n=0
3. then return S[0];
4. S = Ø;
5. for i=0 to S[0].size-1
6. for j=0 to S[1].size-1
7. S ← S ∪ {<S[0][i], S[1][j]>}
8. endfor
9. endfor
10. if n=1
11. return S;
12. else /*n>1*/
13. return concatSequences(S, S[2], ..., S[n]);
14. end
2. if node is leaf
3. then return \{<node>\};
4. childNodes ← all child nodes of node
5. if childNodes.size()=1
6. then return genSequences[childNodes[0]];
7. for i=0 to childNodes.size-1
8. \(S[i] \leftarrow \text{genSequences(childNodes[i])}\)
9. endfor
10. if the relationship between child nodes is “OR”
11. then
12. return \(S[0] \cup S[1] \cup \ldots \cup S[\text{childNodes.size}-1]\)
13. else // the relationship is “AND”
14. return \(S[0] \times S[1] \times \ldots \times S[\text{childNodes.size}-1]\)
15. end

Let us apply Algorithm 2 to the attack tree in Figure 2, genSequence(A) = genSequences(B) \cup genSequences(C) because the relation between B and C is “OR”. genSequences(B) = genSequences(1) \times genSequences(2) = \{<1>\} \times \{<2>\} = \{<1,2>\} because the relation between 1 and 2 is “AND”, genSequences(1) = \{<1>\}, and genSequences(2) = \{<2>\}. Similarly, genSequences(C) = genSequences(3) \times genSequences(4) = \{<3>\} \times \{<4>\} = \{<3,4>\}. Thus, genSequence(A) = \{<1,2>\} \cup \{<3,4>\} = \{<1,2>, <3,4>\}.

Figure 3. Informal attack tree with combined AND and OR

For simplicity, Algorithm 2 assumes that all the child nodes of a node, if any, have a single logical relation, i.e., either “AND” or “OR”, but not a combination of them. This does not lose generality because a mixed relation of “AND” and “OR” can be easily converted into a tree without mixed relation. For example, the nodes in the attack tree in Figure 3 have a combination of “AND” and “OR”. It can be converted to the attack tree in Figure 2. They have the same attack paths \{<1,2>, <3,4>\}.

Once the sequences were generated, they were converted into security tests by determining the system settings (i.e., user accounts and server setting), the actual parameters for the attack actions and conditions in the test sequence (e.g., user name and password for a login session), and oracle values (e.g., criteria about whether or not the attack is successful). If the primitive attack actions or conditions in the security test are programmable, we write code for the security test so that it can be executed automatically. It should be noted that security tests generated from the threat models are different from traditional functional tests in success and failure terminology. If a security test passes, the attack is successful (which would violate security requirements or policies). A failure can be due to a variety of issues, such as failure to connect to server, failure to get to a specific directory, unexpected response from the server, etc., or that the actual attack didn’t work.

From the 14 threat trees we created, we generated 52 security tests. 39 of them can be executed automatically, another 8 can be executed automatically with some tester intervention and 5 are entirely manual tests. Table 2 shows how many tests are related to each classification of STRIDE. Again, one test can belong to multiple STRIDE categories.

IV. EXPERIMENTS

Our experiments consist of two phases. First, we run the security tests generated from the attack trees against FileZilla Server (version 0.9.34). Second, we create security mutants of FileZilla Server by vulnerability injection and run the same security tests against each of the mutants. FileZilla Server version 0.9.34 has 88,596 lines of code (71,940 actual C++ code lines), 107 classes, 1,716 methods, 193 functions, with an average method complexity of 4.74. Method complexity is defined as the number of unique execution paths that can be made through a function or method. In the following, we present the results of testing FileZilla Server and its mutants and discuss the lessons learned from this work.

A. Security Testing of FileZilla Server

When FileZilla Server is properly deployed (i.e., settings are configured properly), all generated security attacks failed to accomplish their goal. This indicates that FileZilla Server is secure against these attacks. When the security tests were first executed against FileZilla Server with the default settings, two of the attacks were successful. One achieved denial of service through creating thousands of login connections with the same id. The attack overflowed the login table of FileZilla Server, which keeps track of the active users and the current activity of that id. A proper setting should limit the number of concurrent connections any given id may have. The default is zero which means unlimited. This default setting doesn’t give any hint to the administrator that there is an actual limit that might be exploited. The other successful test gained the version of the server that was running. While the version information is not always harmful, it can give the attacker clues as to what attacks might be possible by looking at the public bug tracker. In the attack trees, the disclosure of version information for production use (not development use) is considered as a security risk although the risk is not necessarily high.

B. Security Mutation Analysis of FileZilla Server

The fact that the properly deployed FileZilla Server has withstood off our security tests indicates the good security profile of FileZilla Server. But it does not reflect how effective the
security tests can be. In this paper, we use security mutation testing to evaluate the vulnerability detection capability of the security tests and the threat modeling process.

Mutation testing has been used for evaluating software for the past several decades [11]. The fundamental idea of mutation testing is to run test cases against mutants which are obtained by injecting faults deliberately into the correct or baseline version of a program. These faults would represent the same programming errors a programmer might make. Usually, the percentage of mutants killed by the test cases is an indicator of how effective the tests are [12]. So far, creation of mutants in the existing mutation testing research has focused on syntactic changes, such as replacing `&&` (and) with `||` (or). However, syntactic changes may not result in meaningful vulnerabilities in security-intensive software. In this paper, we create security mutants according to the causes and consequences of vulnerabilities. The causes of vulnerabilities include design-level defects (e.g., incorrect policy enforcement) and implementation-level programming errors (e.g., buffer overflow and unsafe function calls). Design-level vulnerabilities are a major source of security risks in software [13]. To create design-level mutants we took a high level look at the program by creating a software architecture document of the program. This document, along with a thorough code review gave us a broad view of how pieces fit together throughout the program and gave us the necessary insight to create some design level mutants. A mutant allowing a user to create directories without the necessary permissions would be classified as a design-level vulnerability in the FileZilla program. This is because no matter what the programmer does it is an unfixable problem using the current design. The consequences of vulnerabilities refer to various potential STRIDE attacks. We have created 30 mutants of FileZilla Server, 13 with design-level vulnerabilities and 17 with implementation vulnerabilities. The mutants have also included reported vulnerabilities of FileZilla Server.

Injection of security vulnerabilities into FileZilla Server is conducted independently of threat modeling and test generation. A different person from the team not involved with the threat modeling studied the FileZilla server and created the mutant injections. The security tests generated from our initial set of threat models killed 8 of the first 14 mutants (57.14%). The other mutants were not killed because the threat models did not cover these vulnerabilities. Then, we built more threat models and thus generated more tests to exercise the mutants. It is important to note that no implementation details were discussed between the person that implemented the mutations and the person that implemented the threat models. Only generic information such as “List command causes crash” is given. In the end, we were able to kill 28 of the 30 mutants for a 93.33% success rate. The two remaining mutants were a memory leak which is difficult to capture by threat models and an implementation dependent issue of an FTP command not covered by the threat model.

C. Discussion

Our experiments show that threat model-based testing is an effective approach to the security analysis of FileZilla Server. The security tests generated from the threat models have killed 93.33% of the deliberately-created mutants. One reason for this effectiveness is that the security tests generated from the threat models directly target the unintended behaviors or invalid inputs that are exploited by attacks. They meet the need of security testing to test the “presence of an intelligent adversary bent on breaking the system” [14].

From the experiments, we have learned several lessons. First, compared to traditional modeling that focuses on the intended behaviors of software, threat modeling requires a significantly different way of thinking. It would not be effective unless the threat models are built as if the builder were an intelligent adversary. Examining STRIDE threats against each system function can help build threat models in a systematic manner. This greatly reduces the chance of missing critical threats. Second, the vulnerability detection capability of the threat-based testing depends upon the threat models. A vulnerability may not be revealed if it is not captured by the threat models. Since threat models typically represent known or anticipated attacks, it can be hard to detect unknown vulnerabilities. Third, test automation can reduce the testing workload because the test execution can be easily repeated. For example, the security tests were executed against each of the 30 mutants. It would have been a daunting task if the security tests had not been automated. Although automation is desirable, not all tests can be automated. For example, user interactions may be needed to change server configuration (e.g., permissions), use administrative functions, and verify attack effects. Fourth, security mutation through injection of vulnerabilities is more difficult than traditional mutation. The former requires an in-depth understanding about the system functions and security requirements. The latter usually focuses on syntactic changes. Although the 30 distinct mutants have covered all FTP services and different vulnerabilities leading to various STRIDE attacks, they are far from complete.

V. RELATED WORK

Threat modeling has been a viable practice for secure software development for some time [15]. Threat models can be used to generate security tests for exercising an implementation. In the past, we have proposed security testing approaches based on threat models represented by UML sequence diagrams [16] and threat trees [17]. Our previous work on testing with threat trees focused on web applications. This paper applies threat modeling and testing to FTP services, which has not been found in the literature. This paper has formally described the algorithms for generating test sequences from attack trees and used the security mutation to evaluate the vulnerability detection capability of the threat model-based testing approach.

Blackburn et al. [18] have developed an approach for generating test vectors (i.e., test cases with test input values, expected output values and traceability information) from security properties. Jürjens [19] has developed an approach for testing security-critical systems based on UMLsec models. Test sequences for security properties are generated from UMLsec models to test the implementation for vulnerabilities.
Julliand et al. proposed an approach to generating security tests in addition to functional tests [20]. The above work does not involve threat models in security testing.

Recently, testing of access control policies has gained increasing attention. Martin et al. [21] have been investigating techniques for test generation from access control policy specifications written in XACML (OASIS eXtensible Access Control Markup Language). Masood et al. [22] have investigated various issues of model-based testing of access control policies [23]. Mallouli [24] et al. have used formal access control models for generating tests. None of this work has used threat models for security testing.

VI. CONCLUSIONS

We have presented the modeling of various security threats for FTP services. These threat models can be used for testing of different FTP server implementations. In this paper, the security tests generated from the threat models are executed against Filezilla Server, a popular FTP server implementation. Our experiments demonstrate that this is an effective approach to the security analysis of Filezilla Server. Although the security tests did not find risky vulnerabilities in the properly deployed Filezilla Server, they have detected the vast majority of the injected vulnerabilities in the security mutants. Our study indicates that modeling and testing with attack trees has some limitations. It is difficult to describe data flows among attack conditions and actions and represent repetitive actions. As a result, it requires some effort to transform the test sequences generated from attack trees to meaningful or executable security tests. This is a barrier to automated generation of executable code for those security tests where the attack actions are programmable. We are exploring how to generate executable security tests from formal threat models represented by Petri nets [13].

Currently, our study has created and used 30 security mutants of the Filezilla Server program. To better evaluate the vulnerability detection capability of the threat model-based testing approach, we plan to create mutants in a more systematic way. For example, we can try to inject vulnerabilities according to the various programming bugs that lead to security vulnerabilities.

ACKNOWLEDGMENT

This work was supported in part by NSF under grants CNS 0855106 and CNS 1004843.

REFERENCES