Aaron Marback, Hyunsook Do, Ke He,
Samuel Kondamarri, Dianxiang Xu

North Dakota State University

Motivation

- Security vulnerabilities can lead to:
  - Compromise of personal, financial, medical, or other records.
  - System wide or subsystem failures of business, mission, or safety critical systems.
  - Other undesired results
- Analyzing software and developing test cases to uncover these vulnerabilities can be time consuming and expensive.

- Software Security and Security Testing
- Secure Development Using Threat Modeling
- Test and Input Generation
- Case Study
- Results
- Conclusion

- **Software Security and Security Testing**
- Secure Development Using Threat Modeling
- Test and Input Generation
- Case Study
- Results
- Conclusion

Software Security

- Protect Data and Systems from Malicious Entities
- Security Testing
  - Test cases aimed at uncovering security vulnerabilities (Thompson and Whittaker2002)
  - Must consider an intelligent attacker (Potter *et al. 2004)*

Overview

- Software Security and Security Testing
- **Secure Development Using Threat Modeling**
- Test and Input Generation
- Case Study
- Results
- Conclusion

Secure Development Using Threat Modeling

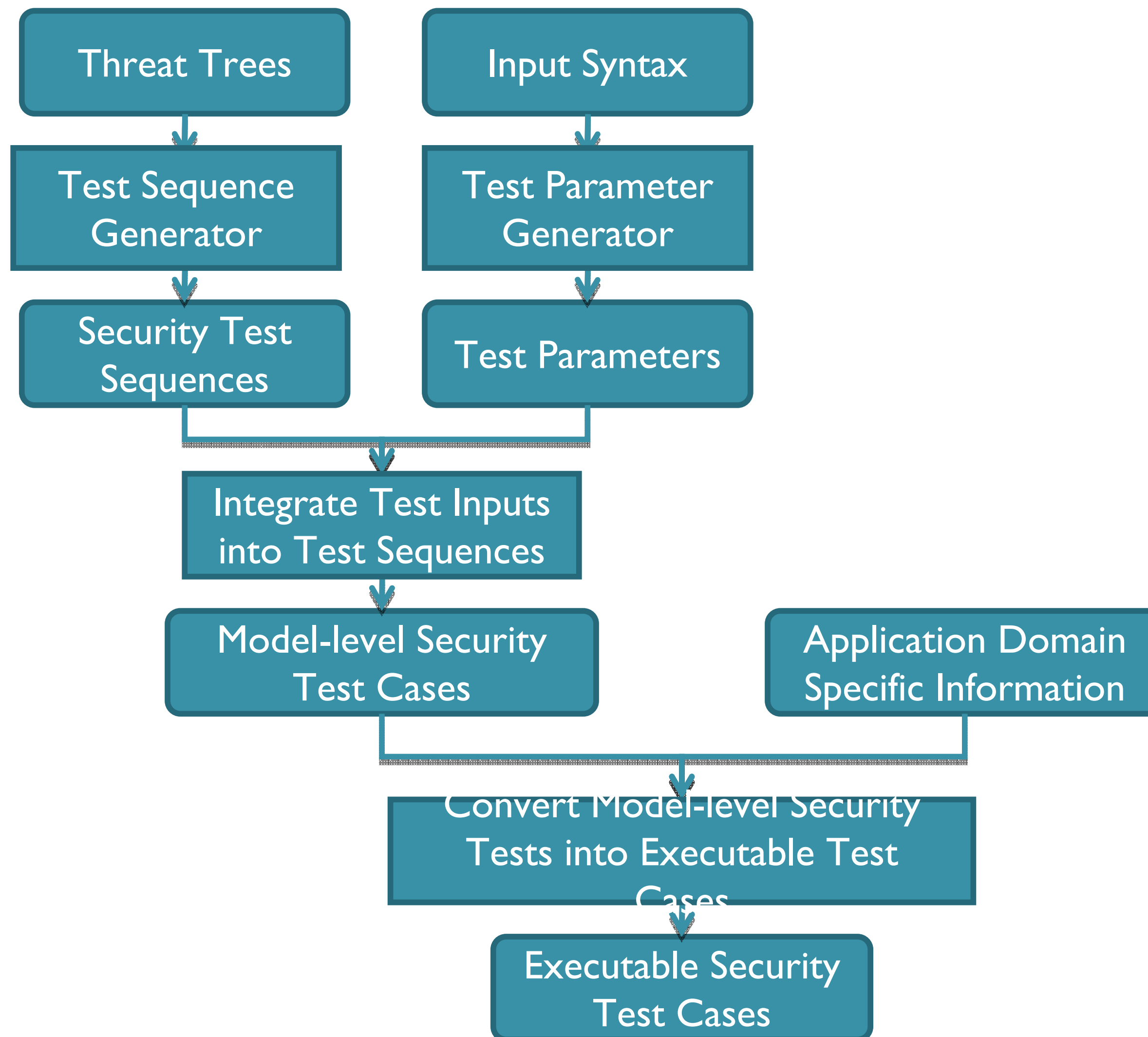
- Threat Modeling is a secure development practice the has emerged.
- 4 Basic Steps (*Howard and LeBlanc 2003*)
  1. Model an application into DFDs
  2. Identify components that may be threat targets and categorize them using STRIDE (**S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, **E**levation of Privileges). Create threat trees from the DFDs and STRIDE information.

- 4 Basic Steps cont.

3. Evaluate and calculate risk of the threats using DREAD (**D**amage potential, **R**eproducibility, **E**xploitability, **A**ffected users, and **D**iscoverability).
4. Decide how to respond to and mitigate the threats.

- Software security and security testing
- Secure development using threat modeling
- **Test and Input Generation**
- Case Study
- Results
- Conclusion

Generating Security Tests From Threat Trees



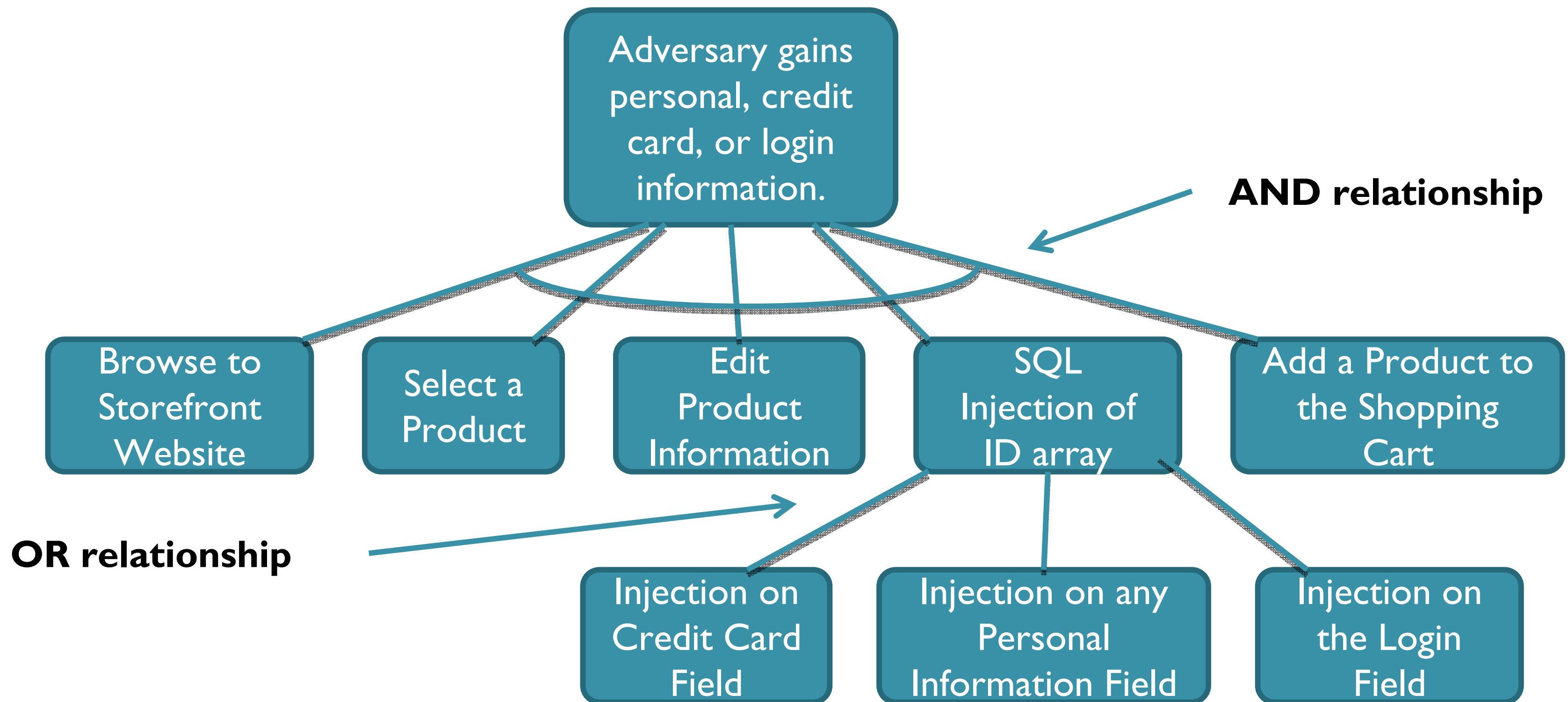
Threat Trees

- A variation of fault trees.
- Describe the sequence of events required to exploit a threat in an application (attack path).
- Child nodes have AND or OR relationships
- Child nodes describe the events required to achieve the parent goal.

Threat Trees

- Example
  - SQL Injection in osCommerce (http://www.oscommerce.com)
    - SQL Injection is a technique used to gain access and possibly modify data in a database by injecting SQL commands in a vulnerable component of an application.
  - Goal of this Attack: Adversary gains personal, credit card, or login information.

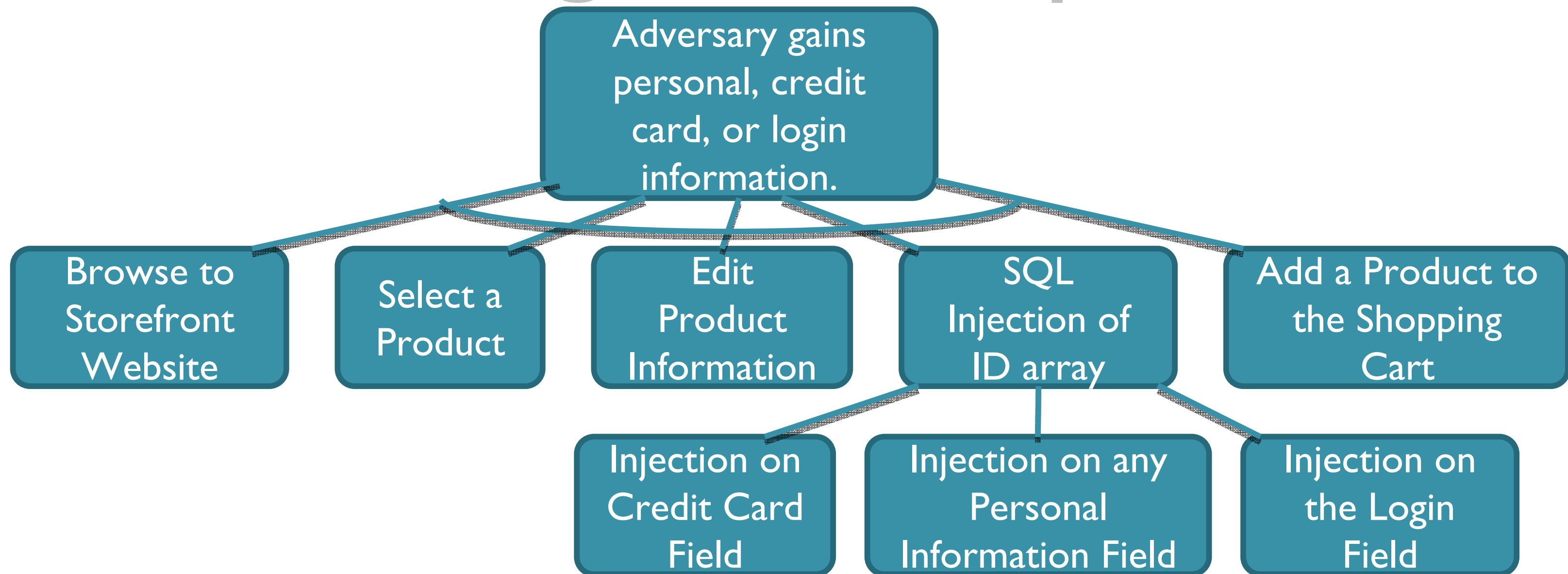
Threat Trees



Generating Test Sequences

- Threat trees (.tmd files) created by the Microsoft Threat Modeling tool are parsed and the relevant data placed in a tree like structures. *(Swiderski and Snyder, 2004)*.
- These trees are then traversed and sequences are created.

Generating Test Sequences



Browse to Storefront Website → Select a Product → Edit Product Information → SQL Injection of ID array → Injection on Credit Card Field → Add the Product to the Shopping Cart

Browse to Storefront Website → Select a Product → Edit Product Information → SQL Injection of ID array → Injection on any Personal Information Field → Add the Product to the Shopping Cart

Adversary gains personal, credit card, or login information.

Browse to Storefront Website → Select a Product → Edit Product Information → SQL Injection of ID array → Injection on the Login Field → Add the Product to the Shopping Cart

Syntax Based Input Generation

- Input information is gathered from the DFDs
- An input syntax is assigned to all applicable inputs.
- These input syntaxes can be reduced and enhanced by combining implementation knowledge with known security testing inputs strings.

Syntax Based Input Generation

- Example SQL Injection threat
  - Initial syntax: a^* (where *a* is any printable ASCII character)
    - Too large to be useful
  - Reduce by using common SQL injection strings
    - WHERE name LIKE '%
  - Reduce this syntax further using application domain specific information
    - SELECT * from customers WHERE cc_number LIKE '%

Reducing input

- Application domain specific information is used extensively to reduce input:
  - Table names, records, and fields
  - Names of different pages and scripts

Creating Executable Test Cases

- Test Sequences are gathered from the threat trees.
- Each parameter is assigned to an input file created from the input syntaxes.
- Each applicable event in a sequence is assigned to a test script that performs that action.

Creating Executable Test Cases

- The events from every sequence are combined and an executable test script is created from them.
- The inputs are combined into one large file containing all test inputs created during the input generation process.

Example

- Browse to Storefront Website
  - Combined with next event
- Select a Product

```
use WWW::Mechanize;
use HTTP::Cookies;
use WWW::Mechanize::FormFiller;
use URI::URL;
my $agent = WWW::Mechanize->new(autocheck => 1);
$agent->cookie_jar(HTTP::Cookies->new);
$myargs = 0;
$agent->get('http://example.com/product_info.php?
products_id=' ."$ARGV[$myargs]");
$myargs = $myargs + 1;
```

Example

- Edit Product Information
  - Combined with next event
- Injection on Credit Card Field

```
$agent->field("$ARGV[$myargs]", '99\' UNION SE-
LECT null,CONCAT('cc_number'), null,null FROM cus-
tomers');
$myargs = $myargs + 1;
```

- Add the Product to the Shopping Cart

```
$agent->click();
```

Example

- Combined

```
#usr/bin/perl -w
#p0:product_id p2:sql_string
use WWW::Mechanize;
use HTTP::Cookies;
use WWW::Mechanize::FormFiller;
use URI::URL;
my $agent = WWW::Mechanize->new(autocheck => 1);
$agent->cookie_jar(HTTP::Cookies->new);
$myargs = 0;
$agent->get('http://example.com/product_info.php?
products_id='."$ARGV[$myargs]");
$myargs = $myargs + 1;
$agent->field("$ARGV[$myargs]", '99\' UNION SE-
LECT null,CONCAT('cc_number'), null,null FROM cus-
tomers');
$myargs = $myargs + 1; $agent->click();
```

Overview

- Software Security and Security Testing
- Secure Development Using Threat Modeling
- Test and Input Generation
- **Case Study**
- Results
- Conclusion

Empirical Study

- Object of Analysis
  - osCommerce *(http://oscommerce.com)*
    - Web based storefront and shopping cart application
    - Written in PHP
    - Uses a database backend (MySQL in our environment)
    - Database has 46 tables
    - Over 10,000 lines of code

Experimental Environment

- Two Virtual machines running inside VirtualBox
- Server
  - Suse Linux version 9.1
    - Apache Web Server
    - MySQL DBMS
- Malicious Client
  - Xubuntu 8.04.1
    - Perl used to execute attacks

Experimental Setup

- Develop threat trees for 5 threats
  - 2 known to be properly handled (Threats 1 and 2)
  - 3 known vulnerabilities (Threats 3, 4, and 5)

Experimental Setup

- Properly handled threats
  - Client side data modification
  - User account information retrieval
- Known vulnerabilities
  - Cross-site scripting (XSS)
  - SQL injection
  - Multiple login attempts (brute force susceptibility)

Experimental Setup

- Input
  - Design and implementation information
    - Database table and field names
    - Variable names
  - Stored data
    - Database records
  - Common Security Strings
    - XSS strings
    - SQL injection strings

Overview

- Software Security and Security Testing
- Secure Development Using Threat Modeling
- Test and Input Generation
- Case Study
- **Results**
- Conclusion

Results

- Effectiveness

Threat	Sequences	Tot. Tests	Test that Exposed Vulnerabilities
1	18	1008	0
2	9	504	0
3	4	132	132
4	3	138	138
5	4	224	224
Totals	**38**	**2006**	**494**

Results

- Effectiveness
  - Properly handled threats
    - All tests passed (Did not report a vulnerability)
    - No false positives
  - Known vulnerabilities
    - All tests exposed vulnerabilities.

Results

- Efficiency
  - 7379 seconds (122 minutes) map all threats
  - 2006 tests were executed in 2032 seconds (33 minutes)

Results

- Effective and efficient test case development technique for discovering vulnerabilities.
- Can generate many sequences with a few events.
- Limitations:
  - Lack of variety in object program(s)
  - Is dependant on the amount of design information and quality of threat trees.

Overview

- Software Security and Security Testing
- Secure Development Using Threat Modeling
- Test and Input Generation
- Case Study
- Results
- **Conclusion**

Conclusions

1. Identify threat goals that attackers want to realize using threat trees.
2. Generated executable security tests automatically from threat trees considering actual inputs.
3. Ensured that threats tested are properly handled.

Future Work

- Examine using more threat trees.
- Examine more applications.
- Investigate ways to more efficiently gather input constraints.