Successful Strategies for QA-Based Security Testing

Rafal Los
Enterprise & Cloud Security Strategist
HP Software

ENTERPRISE SECURITY

Agenda

- Workshop-style for 90 minutes (or so)
- Participation is required (or I will call on you)
- If you brought your laptop, you can play along!

Application Security Testing Fundamentals

Unmasking the Hacker

the dark art of ‘hacking’

- So what is ‘hacking’?

- Defined as:

  “Unauthorized attempts to bypass the security mechanisms of an information system or network” (source: wiktionary.org)

The functional vs. security tester

is there any creative cross-over?

Functional Testers

- Validate what we know *should be true*
- Base assumptions off of requirements
- Formalized testing structures, methods
- Established procedures for testing
- Carefully defined data sets
- Log defects to bug trackers, defect system
- **Defects** are bad.

Security Testers

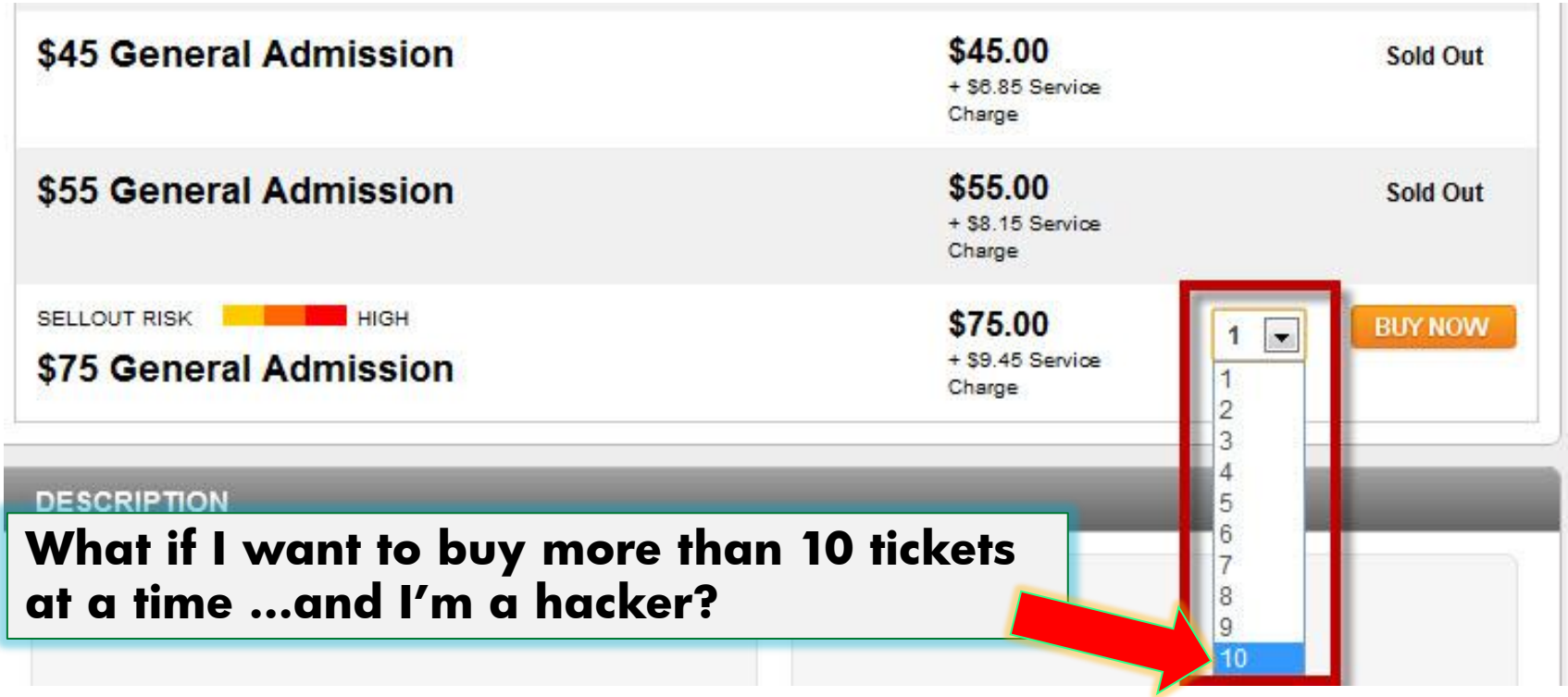
- Ignore what is known, look for *unknowns*
- Base assumption off experience, skills
- Often referred to as "anti-testing"
- Method varies by tester, tool, app type
- Carefully defined data sets
- Log vulnerabilities to testing framework
- **Vulnerabilities** are good.

Becoming a Hacker

how to think like a breaker

- **Terminology**
  - confirmed security defects are known as *vulnerabilities or vulns*
- **Mindset**
  - think “what can I do to make this application deviate from its programmed purpose?”
- **Method**
  - rely on critical thinking to circumvent inherent security controls (rely on amassed attack data)
- **Tools**
  - tool sets vary by budget, experience; rely on structured QA-positioned technologies to enable you
- **Goal**
  - discover ways to *abuse* application functionality, or to break process, manipulate the system

Demo – Manipulating Application Logic



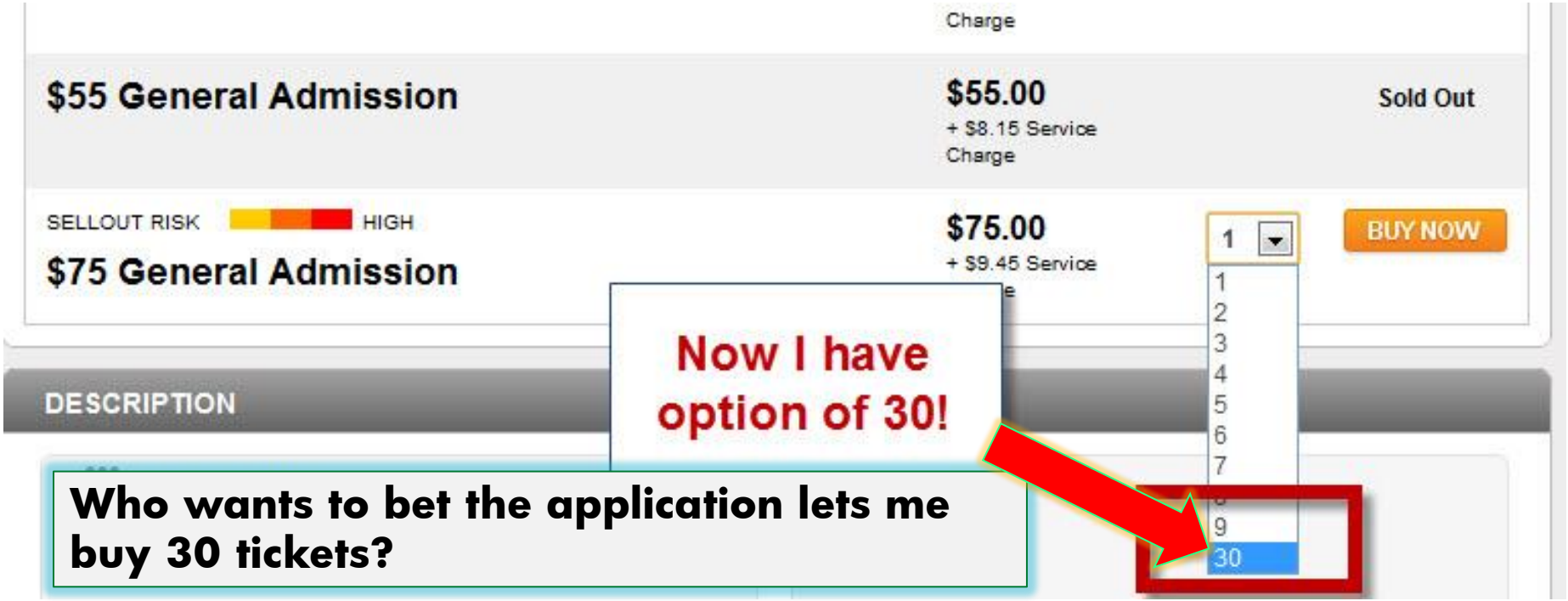
Demo – Manipulating Application Logic

```
▼<td class="QuantityCell">
  ▼<div id=
    "ctl00_mainContent_ucEventView_ucEventTicketItemsDisplay_lvTicketItems_ctrl4_pnQuantity"
    class="Quantity">
    ▼<select name=
      "ctl00$mainContent$ucEventView$ucEventTicketItemsDisplay$lvTicketItems$ctrl4$dropTicketC
      ount" id=
      "ctl00_mainContent_ucEventView_ucEventTicketItemsDisplay_lvTicketItems_ctrl4_dropTicketC
      ount">
        <option value="1">1</optio
        <option value="2">2</optio
        <option value="3">3</option>
        <option value="4">4</option>
        <option value="5">5</option>
        <option value="6">6</option>
        <option value="7">7</option>
        <option value="8">8</option>
        <option value="9">9</option>
        <option value="10">10</option>
```

Using Chrome's "Inspect Element" option we make a small change…

<option value="30">30</option>

Demo – Manipulating Application Logic



Functional vs. Security testing (again)

barely scratching the surface

Functional Testers

- Check options 1…10 for tickets
- Requirements say 1…10 tickets
- Formalized testing structures, methods
- QA Analyst would test 1…10 as defined
- Carefully defined data sets
- Log defects to bug trackers, defect system
- **Defects** are bad.

Security Testers

- Ignore given options, add your own
- Experience says try out of range
- "Lucky guess" app will take new input
- Not all security testers would catch this!
- Carefully defined data sets
- Log vulnerabilities to testing framework
- **Vulnerabilities** are good.

Application Security Testing 101

basics you should know

- Application vulnerabilities (security defects!) basics
- Lots of great resources to read about [web] application security
  - OWASP (Open Web Application Security Project) –maintains the “Top 10”
  - WASC (Web Application Security Consortium) – Threat classifications
  - CWE (Common Weakness Enumeration) – Classified application weaknesses into comprehensive taxonomy
- Lots of great resources on Offensive vs. Defensive application security
  - OWASP.org is a FREE great start (Open Web Application Security Project)
  - Mailing lists, books, conferences and webinars

OWASP Top 10

popular classification of defects

1. injection
2. cross-site scripting (XSS)
3. broken authentication or session management
4. insecure direct object reference
5. cross-site request forgery
6. security misconfiguration
7. insecure cryptographic storage
8. failure to restrict URL access
9. insufficient transport-layer protection
10. unvalidated redirects and forwards

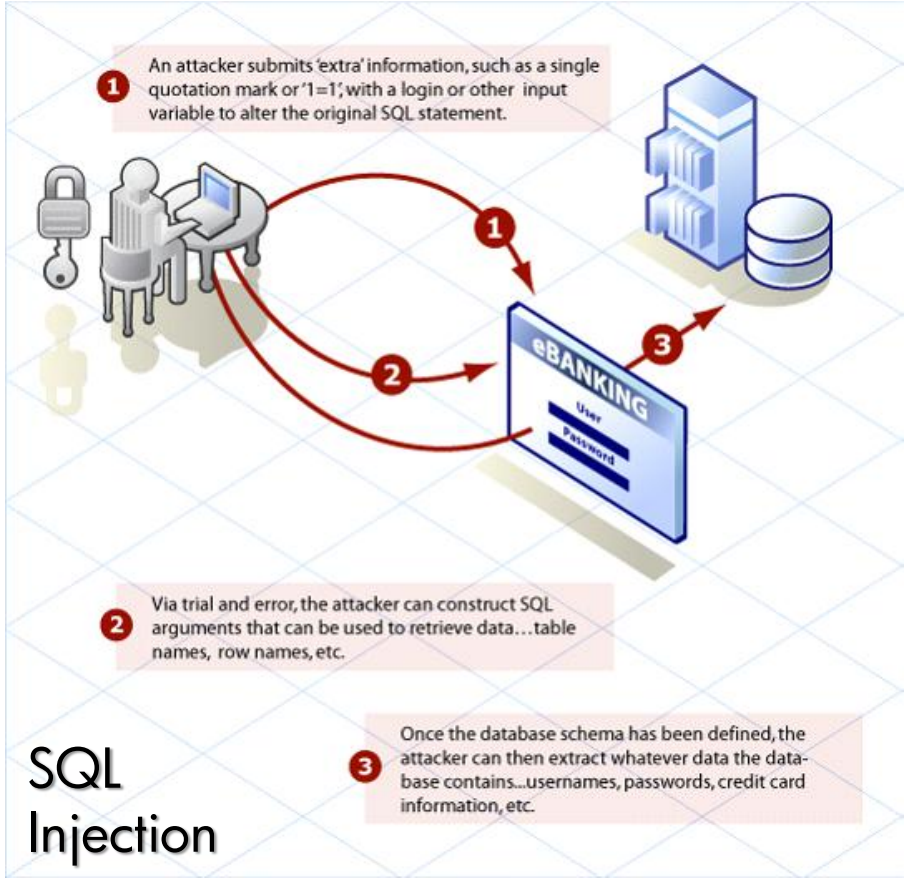
Many attacks you hear about on the news today are one of these.

Injection

injecting “into” application

- Injection attacks involve the ‘attacker’ pushing their own bits into the application, while the application fails to filter/sanitize that input.

- Results in usurping control of:
  - a process
  - a database
  - the application
  - the operating system

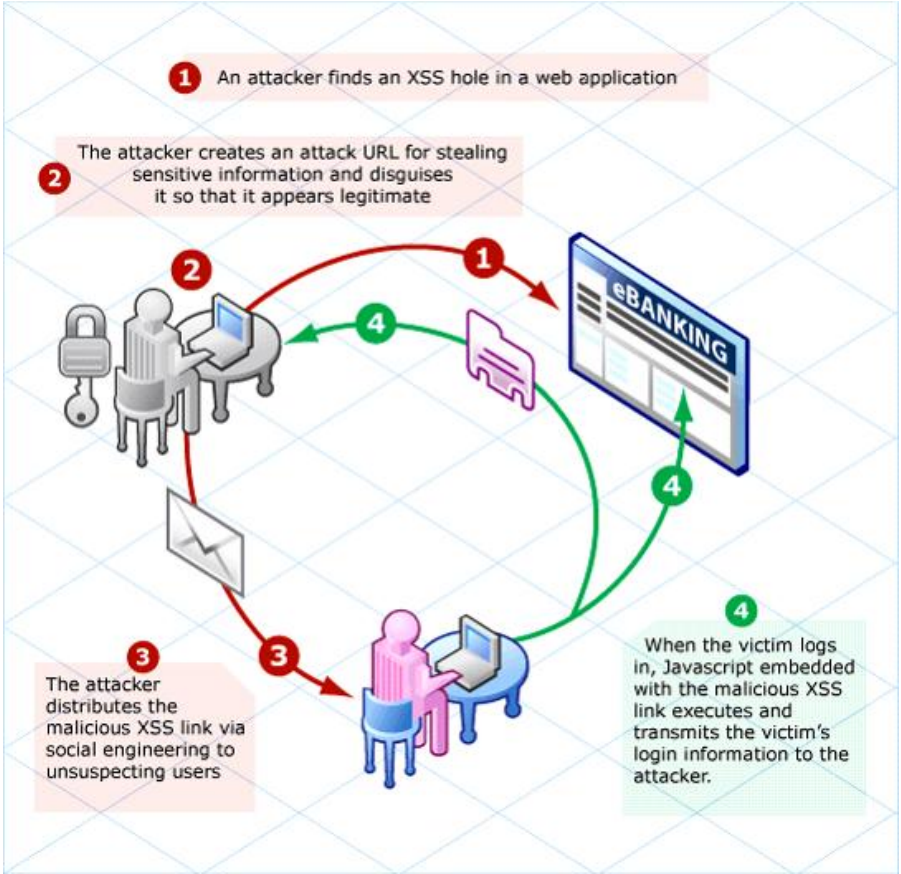


Cross-Site Scripting (XSS)

injecting (java)script into the app

- Cross-Site Scripting (XSS) usually involves injecting JavaScript into the application, to perform some action in the user's browser without their knowledge.

- Cross-Site Scripting happens in 2 forms:
  - Stored – attack permanent in the application
  - Reflected – user must click/interact to execute attack



Time for a quick demo

Software Security Testing: The Big Break-Up

Challenges of Security Testing

Application Security Testing

- Identifying all the *unintended functions* of the code
- Testing using data application *is not expecting*
- Trying to elicit *unintended responses* from the application
- Identifying *unplanned workflows* through the application

This is not a trivial task!

Breaking Security Testing Up

Time for application security to break up

- Prescriptive security mechanisms
  - Security mechanisms that can be described and identified
- Pattern-based *fuzzing*
  - Computer-generated iterative patterns
- Human based hacking and analysis
  - Manually manipulating the application, analyzing the results

Prescriptive Security Mechanisms

We should focus most of our attention and energy here.

Prescriptive → Well-Defined

Definitions → Requirements

- Application mechanisms we can define in requirements stage
- Assumption: If we can define it, we can test for its existence
- Key: Creating testable application security requirements

Defining Good Application Security

How can we define solid application security requirements?

- Keep it simple
- Be clear
- Be precise
- Use standard language
- Leave nothing to interpretation (binary *yes* or *no*)

Defining Solid Security Requirements

Simple exercise – let’s define a security requirement

Component:

Requirement(s):

Enabling Technologies

Good [security] requirements should not require *tools to verify* them.

Basic application security requirements are *prescriptive*

- What *should* the application do
- Must have test conditions for pass/fail
- Must have resultant states for pass/fail verification
- Doesn't need to go into details of why the mechanism exists, etc

Pattern-Based ‘Fuzzing’

Understanding anti-patterns

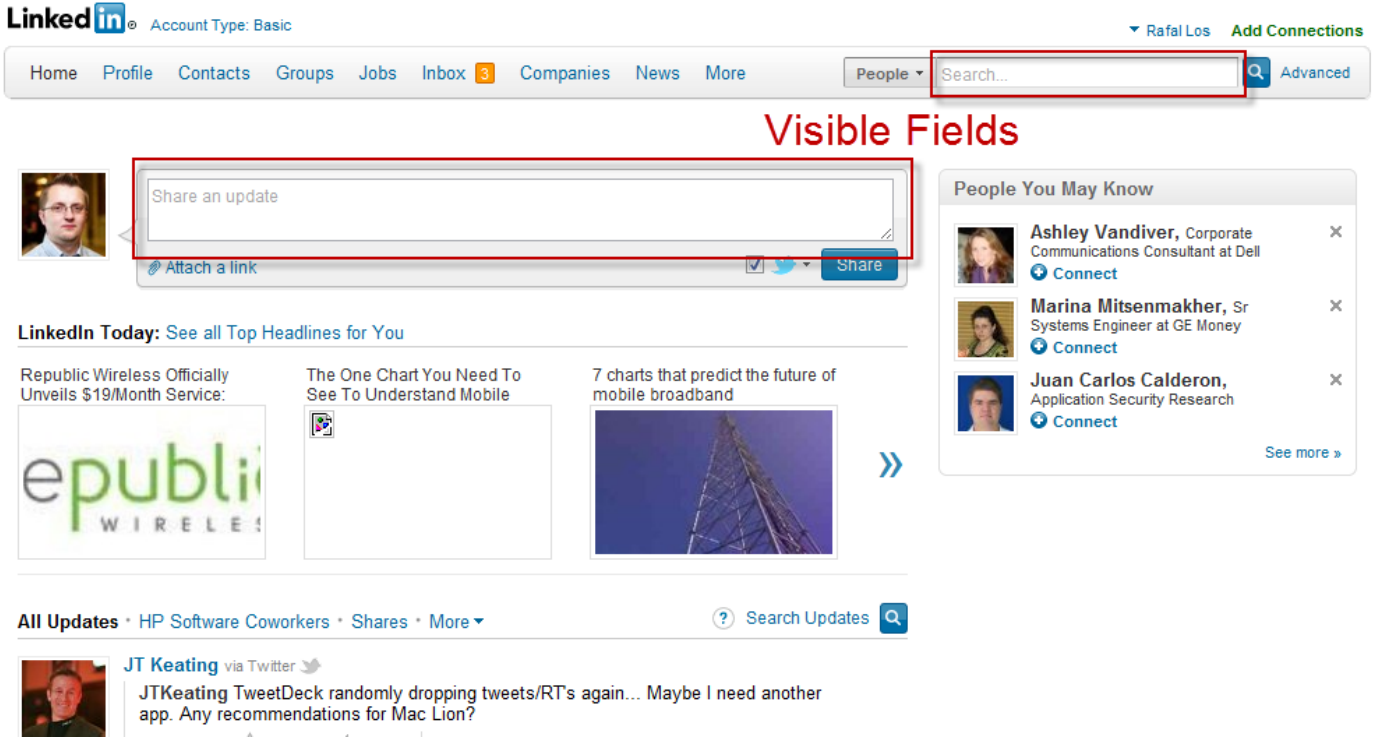
- Application abuse cases are generated from legitimate requirements
- Application fuzzing data derived from real test data
- Form-based (data-based) fuzzing is the simplest form
  - Iterate through various fields, data-types, permutations of possibilities
  - Generate types of data application is *not expecting*
- Logic-based fuzzing is difficult
  - Must be done to get it ‘right’

Data-Based ‘Fuzzing’

Fuzzing is technology assisted application security testing

- **Basic** – executed without advanced ‘security’ knowledge
- **Repetitive** – millions+ test cases are generated and executed
- **Automated** – enabling technology which can execute tests quickly
- **Comprehensive** – test every parameter in an application

Fuzzing Example

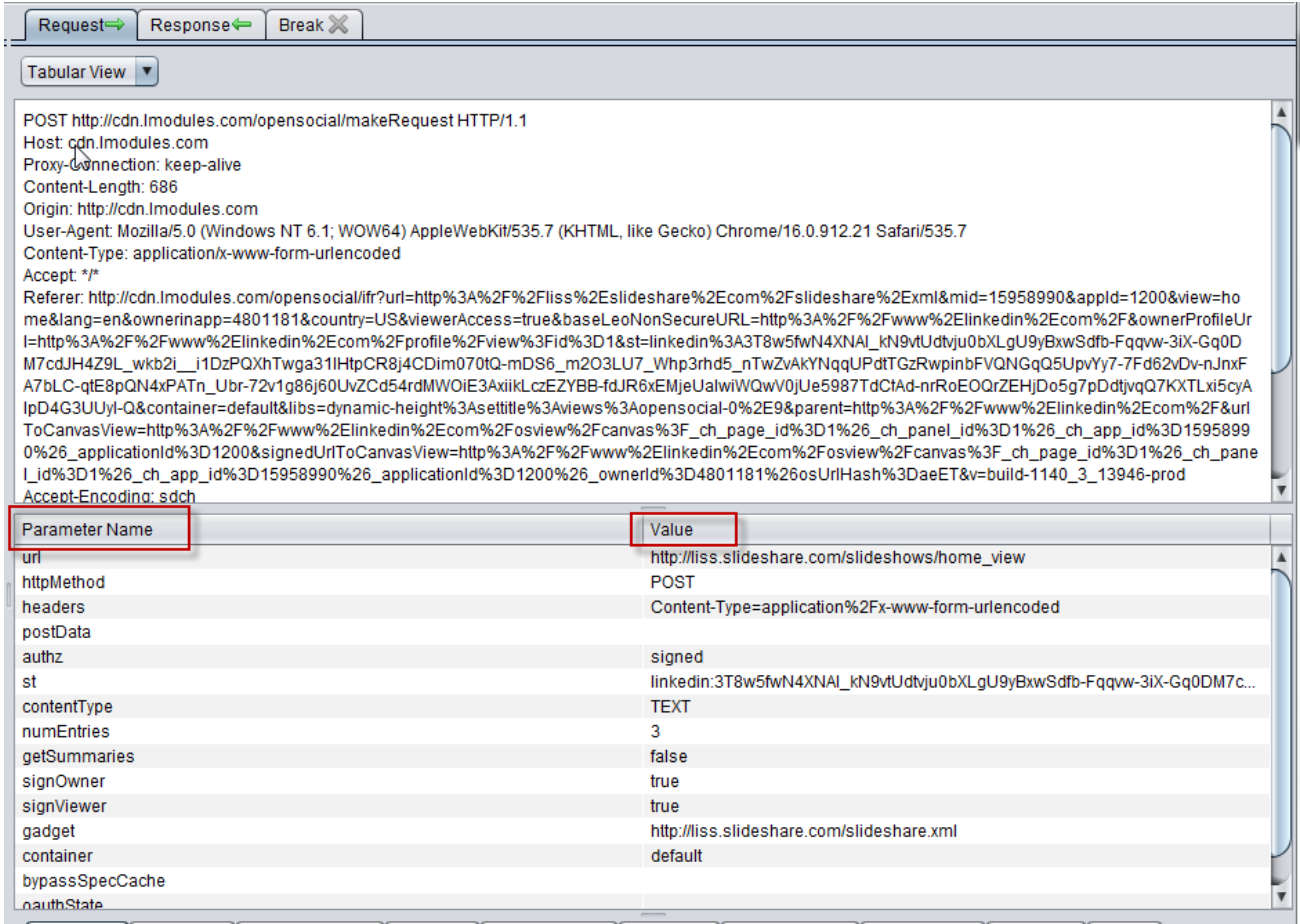


Most people see a site or application as a collection of "visible input fields" …

Fuzzing Example

Applications have many parameters which are not visible to the person browsing without some technology.

Automation will fuzz all the parameters it is coded to.



Fuzzing Demo Using ZAP Proxy

Human-Based Hacking and Analysis

Advanced ‘security testing’ can be left up to the ethical hackers

- Requires advanced skills from years of training/doing
- Requires advanced technology to iterate through millions of lines of code

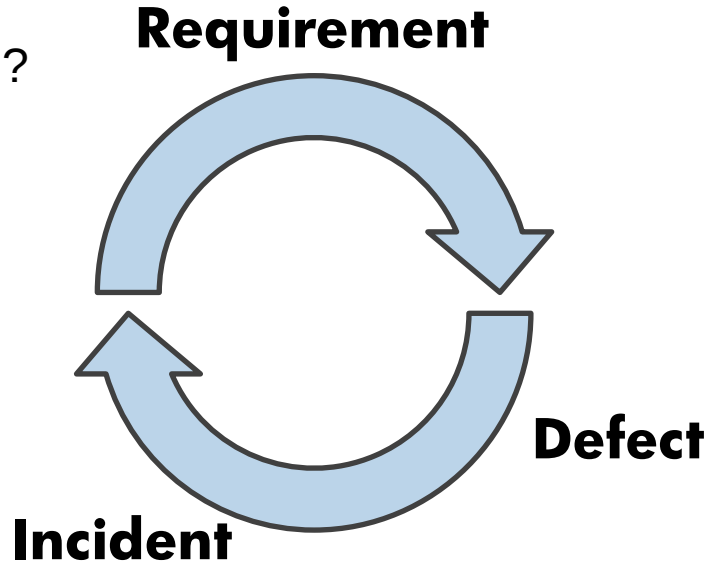
Moral of the story: Leave the hacking to the security team

Requirements – Defects Cycle

The Requirements – Defects Cycle

How are requirements – defects – incidents linked?

- Requirements are defined at the start of project
  - Pre-defined security mechanisms for the application
- Defects are misses against requirements
  - Feed into new requirement(s) potentially
- Incidents are defects discovered post-release
  - Feed into new requirement(s) potentially



Learning from Incidents

Einstein defined madness as performing the same task and expecting different results …

Do we keep re-using the same requirements and expecting better security?

- Incidents teach us 2 things:
  - Where our code failed
  - How we can test better in the future
  - *Depends on how well we have performed our forensic analysis!

Conclusions

Recapping …

- Application security ‘testing’ can be split into separate tasks
  - Things we can define/test
  - Things we need experts for
- Good requirements are verifiable- easily and simply
- Learning from failure is important for 2 reasons
  - Better testing
  - Better requirements

The most impoartant question:

Did you learn anything?

Follow me…

Twitter:

@Wh1t3Rabbit

Blog:

HP.com/go/white-rabbit

Podcast:

podcast.wh1t3rabbit.net

THANK YOU