IronWASP

A Web Application Security Testing Platform

Lavakumar Kuppan

@lavakumark

Disclaimer:

Views expressed in this talk are my own and do not necessarily reflect those of my employer

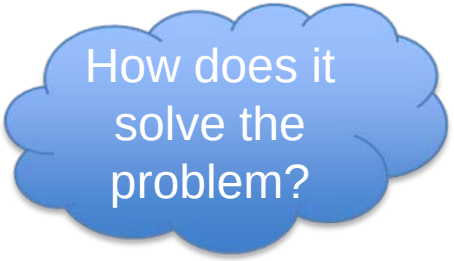
SECURITYBYTE 2011

CONFERENCE & WORKSHOPS

What to expect from the talk

Problem: Web Security Testing != Efficient

Solution: IronWASP



Bio

Penetration Tester

Day job at a large bank, 5+ yrs exp.

Web Security Researcher

Night-time hobby

Developer

Tools and scripts in C#, JS, PHP, Python, Perl…

Speaker

BlackHat, SecurityByte, OWASP Appsec Asia, ClubHack, NullCon…

Research

Attack and Defense Labs

Repository of all Research and Tools

http://www.andlabs.org

HTML5 Security, Browser-side Security

Topics of interest

#5 on Top 10 Web Hacks of 2010

CSRF-protection bypass using HPP and ClickJacking

Tools

IronWASP

Presenting in this talk

Ravan

JavaScript based Distributed Computing System

JS-RECON

HTML5 based JavaScript Network Recon Tool

Shell of the Future

XSS Reverse Web Shell

Imposter

Browser Phishing Framework

Problems with Web Security Testing

Automated Scanning Tools

- Can only check for few issues
- Poor site coverage
- Struggles with JS complexity
- Not designed for Security testers so:
  - Not transparent about working
  - Not honest about limitations
  - Not collaborative

Manual Testing

- Time & Effort intensive
- Not Scalable
- Not Repeatable
- Expensive
- Good testers are few in number

Solution - Empower the Manual Tester

The tester needs a tool that:

- Knows its limitations and is honest about it
- Transparent about its functioning
- Collaborates with the tester
- Can be altered and molded to suit the needs of the tester

IronWASP

What is IronWASP

- An environment for Web Application Security Testing
- Designed for optimum mix of Manual and Automated Testing
- Designed for Penetration Testers
- Let’s you write a custom Security Scanner in a very short time
- Open Source and Open Architecture
- GUI based & does not require installation

Key Components

- Built-in Crawler + Scan Manager + Proxy
- Python/Ruby based plug-ins
  - Active plug-ins for Scanning
  - Passive plug-ins for vulnerability detection
  - Format plug-ins for defining data format
  - Session plug-ins to customize the scans
- Integrated Python/Ruby Scripting Environment with IronWASP API
- JavaScript Static Analysis Engine

Plug-ins

- Written in Python/Ruby using the IronWASP API
- Easy to modify existing plug-ins
- Can easily add new custom plug-ins
- UI based API doc provided inside the tool
- Syntax highlighting Script Editor with error checking support built-in

Plug-ins

- IronPython Plug-ins

  Maintained by Me

  Location: https://github.com/Lavakumar/Iron-Plugins
- IronRuby Plug-ins

  Maintained by Manish Saindane (Project Contributor)

  Location: https://github.com/msaindane/Iron-Plugins

Passive Plug-ins

- Analyzes all traffic going through the tool
- Can also modify the traffic
- Identifies vulnerabilities passively

Eg:

- Passwords sent over clear-text
- Http-Only /Secure flag missing in cookies

Active Plug-ins

- Performs scans against the target to identify vulnerabilities
- Executed only when the user explicitly calls them
- Fine-grained scanning support

Eg:

- Cross-site Scripting
- SQL Injection

Format Plug-ins

- To deal with various data formats in Request body.
- Eg:
  - JSON
  - XML
- Allows scanning even for custom data formats

Format Plug-ins

- Consider this Login Request:

```
POST /login.php HTTP/1.1
Host: example.org
Content-Length: 21

username:lava|pass:s3cr3t
```

- Request body is in Custom Format

```
name:value|name:value
```

- Standard scanners don’t know how to scan this

Format Plug-ins

- Write a Format Plug-in to convert this format to XML and vice versa

```
username:lava|pass:s3cr3t
```

```
<xml>
    <username>lava</username>
    <pass>s3cr3t</pass>
</xml>
```

- IronWASP can now scan this format using existing Active Plug-ins

Format Plug-ins

- This technique can be used to handle any data format:
  - Java Serialized Objects
  - AMF
  - WCF
  - GWT
  - Multi-part POST

Session Plug-ins

- Every site has slight variations in:
  - Authentication
  - Session handling
  - CSRF protections
  - Logic-flow
- Automated Scanners don’t understand this
- Testers understand this
- Testers must feed this info in to the Scanner

Session Plug-ins

- In IronWASP the tester writes a Session Plug-in for this purpose
- This plug-in would determine:
  - How to login to the site
  - How to handle CSRF tokens
  - How to handle session
  - How to handle multi-step forms
- Customizes the Scans for the web site

Integrated Scripting Engine

- This is where the magic happens
- Python/Ruby scripts using IronWASP API
- Full access to all the functionality of the tool
- Can create precise Crawlers and Scanners
- Can analyze the HTTP logs for Access Control and other checks

IronWASP API

- Extremely simple and easy to use
- Some of the Available Classes:
  - Request
  - Response
  - IronSession
  - Crawler
  - Scanner
  - Tools
  - HTML

IronWASP API

- Create a Request, send to server and view response Body:

```
>>> r = Request(“http://example.org”)
>>> res = r.Send()
>>> print res.BodyString
```

- Get a Request from the Proxy log with ID 6, send to server and view response code:

```
>>> r = Request.FromProxyLog(6)
>>> res = r.Send()
>>> print res.Code
```

IronWASP API

- Repeat a Request 10 times:

```
>>> r = Request(“http://example.org/index.php”)
>>> for i in range(10):
…       r.Send()
```

- Fuzz a certain parameter with increasing integers:

```
>>> r = Request(“http://example.org/item.php?id=23”)
>>> for i in range(10):
…       r.Query.Set(“id”,i)
…       r.Send()
```

IronWASP API

- Creating a Scan Job:

```
>>> r = Request.FromProxyLog(12)
>>> s = Scanner(r)
>>> s.AddCheck("XSS")
>>> s.AddCheck("SQLi")
>>> s.ScanAll()
```

JavaScript Static Analysis

- IronWASP performs Taint Analysis for DOM based XSS
- Identifies Sources and Sinks and traces them through the code
- Custom Source and Sink objects can be configured
- Handles a few JavaScript quirks like **a.b** being presented as **a[“b”]** or

  var x = “b”; a[x]

Q& A