ModProfiler: Defending Web Applications from 0-day Attacks
Signatures out. Traffic profiling in.

Ivan Ristić and Ofer Shezaf
Breach Security
OWASP Israel 2008
“Broccoli is a plant of the Cabbage family, Brassicaceae (formerly Cruciferae). It is classified as the Italica Cultivar Group of the species Brassica oleracea. Broccoli possesses abundant fleshy flower heads....”

Wikipedia
About Us

Ivan Ristić and Ofer Shezaf, Breach Security

- **Web application firewall experts:**
  - Ivan created ModSecurity, the most popular WAF on earth, and wrote “Apache Security” for O’Reilly.
  - Ofer created WebDefend, the first and most advanced behavioral based WAF.

- **Web application security leaders:**
  - Officers of the Web Application Security Consortium (WASC).
  - Lead OWASP chapters in London and Israel.

- **Open source and community projects:**
  - Ivan leads the WASC Web Application Firewall Evaluation Criteria (WAFEC).
  - Ofer leads the WASC Web Hacking Incidents Database (WHID) project.
Breach is a leading WAF vendor.

Sole focus on web application security since 1999.

Managed by a group of experienced security professionals.

Best application security DNA in the industry. We write the books.

Home to ModSecurity, the open source WAF.
APPETIZER

THE PROBLEM

DOMAIN
Web Applications Are Dangerous

- Applications are **vulnerable**:
  - Unique, each one exposing its own vulnerabilities.
  - Change frequently, requiring constant tuning of application security.
  - Complex and feature rich with the advent of AJAX, Web Services and Web 2.0.

- Applications are **under threat**:
  - New business models drive “for profit” hacking.
  - Performed by professionals enabling complex attacks.

- Potential **impact** may be severe:
  - Web applications are used for sensitive information and important transactions.
  - Attack may target site customers.
What Are We Doing About It?
Web Application Security through the application lifecycle

Ensuring code is secure by training developers

- Programmers are not expected to be security experts. For example, they would not always understand CSRF.
- Security is always a secondary goal.
- Code developed externally due to outsourcing, M&A and packaged software.

- Very expensive to perform comprehensively: requires considerable expertise and time.
- Needs to be performed on each change in the application.

The cheapest solution.
- Last barrier for everything that sneaks through coding and testing.

Real time protection using Web Applications Firewalls (WAFs)

Inspecting applications for vulnerabilities: automated/manual/code review/pen testing

Can WAFs be effective?
WAF Protection Strategies

Negative security model: allow all, deny what's wrong

- Web specific IPS.
- Simple concept, generic to all applications and provides instant security.
- Based on rules instead of signatures: full parsing, complex logic, anti-evasion.
- Difficult to guard against every attack variant and evasion attempts.

Positive security model: deny all, allow what's right

- An independent input validation envelope for web applications.
- Provides the best protection.
- Rules must be written specifically for each page in the application.
- Rules needs to be maintained as the application changes.
- Easy to write for specific vulnerabilities (virtual patching)
Why is Positive Security Better?

- Classic example of an SQL injection attack
  - 1=1
  - Many IPS solutions include a signature to detect this attack.

- A WAF would easily overcome these evasions:
  - Encoding: 1%3D1
  - Including white space characters: 1 =%091
  - Adding SQL inline comments: 1 /* comment */ = 1

- But it is impossible to create a signature for every tautology:
  - 1+1=2, 2 > 1 and for some databases just 1.

- A positive security rule will provide the best security:

```
<Location "/login.php"
  SecRule ARGS:username "!\w+" "deny,log"
</LocationMatch>
```
Positive security models are generally difficult to build and maintain.
FIRST COURSE
MODSECURITY
What is ModSecurity?

- The most popular WAF in the world with (a lot) more than 10,000 installations.
- An open source production grade project, started in 2002.
- An Apache module which supports both embedded and reverse proxy deployments.
- Support and training by Breach Security.
ModSecurity Philosophy

- Tell it like it is
- Make the WAF technology available to everyone
- Document everything
- Everything is possible & nothing is done implicitly
- You need to know what you're doing or use pre-packaged rule sets
It's a simple event-based programming language.

- Five processing phases, one for each major processing step.
- Look at any part of the transaction.
- Transform data to counter evasion.
- Combine rules to form complex logic.

Common tasks are easy, complex tasks are possible.
Advanced Features

- Persist information across requests; You can create small databases of sorts.
- Support for anomaly-based rules.
- Support for sessions and application users.
- Log entire transactions or sessions, sanitize data before logging.
- Intercept file uploads.
- XML support (parse, validate, extract).
Example Rules

Very simple
- SecRule ARGS attack
- SecRule "ARGS||ARGS:p" attack

Different operator
- SecRule ARGS "@verifyByteRange 10,13,32-126"

Interesting
- SecRule REMOTE_ADDR "@rbl sc.surbl.org"
Real Life Example

Virtual patching example using the positive security approach:

```<Location /apps/script.php>
    SecRule &ARGS "!@eq 1"
    SecRule ARGS_NAMES "!^statid$"
    SecRule ARGS:statID "!\d{1,3}$"
</Location>```

Rules should include metadata, such as ID, revision, human-readable message, and so on.
Components

- **ModSecurity 2.5**
  - The core rules processing engine.

- **ModSecurity Core Rules**
  - An open source rule set providing a generic negative security application layer protection.

- **ModSecurity Community Console**
  - A free GUI tool for aggregating events from up to 3 ModSecurity sensors.

- **The community**
  - Glues everything together
HAVE A DRINK

POSITIVE SECURITY THROUGH LEARNING
Behavioral-Based Learning

Either each model separately or by anomaly scoring: aggregating multiple tests.

- Monitor inbound traffic and generate a profile.
- Generate a statistical model for normal values of the properties of the request.
- Validate request according to statistical model.

- Field length, character set, expected value or type, existence, order, cardinality and location.
- Properties not limited to fields: can include for example also properties of headers or uploaded files.
Sample Profile
Model Requirements

Portability
- Must work with any web-based application, irrespective of the underlying platform.

Partial model support
- In terms of coverage, but also in terms of quality.

Real-life usefulness
- Must correlate to the attack landscape.

Ease of use
- We want people to be able to write and maintain models by hand.
We have identified the following building blocks:

- Application
- Resource
- Resource behavior
- Parameter
- Parameter attribute
Real-Life Challenges

- **Data embedded in URLs**

- **Default pages (i.e. resource aliases)**
  - http://example.com the same as http://example.com/index.php

- **Internal request dispatching**

- **Parameters generated at run-time**
  - http://example.com/?a5647=89
Suddenly we realize the storage format for our model is useful for more than profiling through learning:

- Models could be distributed by application developers.
- Users can write and exchange application models.
- Community projects could maintain models as separate projects.
- The same model works well for virtual patching.
Collecting Data

Uses ModSecurity audit logs, as source of traffic

Contain complete HTTP transaction data.

Filter out invalid traffic

Ignore requests singled out by signatures.
Remove "noise" (non-application requests).

Extract properties

User defined mapping (dynamic URLs, custom separators).
Model Generation

- Blocking strategy set by user: warn-only, block, or mixed mode (block for well-learned resources, warn for all others).
- Recommended to use detection only mode initially to test rules and apply exceptions.

Exported as ModSecurity rules

Collect Sample

Generates tests for each model (length, char set, type) for each parameter

Matches ModSecurity rule capabilities.

Simple fixed size sample of requests used for elements and all models.
Real World Issues

Handling of partial learning
- Rules generated for URLs for which sample was too low can be set to alert even if other rules block.
- Rules generated to alert/block on URLs and parameters not seen during learning.

Handling of application changes
- A change may result in a flood of events.

Negative security should still be used
- Filter attacks for learning.
- Provide protection during learning period and for partially learned and unlearned resources.
- Protection for free form text fields.
DESSERT

CONCLUSION
Positive Model Benefits

What can positive security achieve:

- Prevent information leakage
- Reduce attack surface
  - Request methods
  - Content encodings
- Debug parameters
- Prevent injection in some cases
- Reduce the likelihood of injection in others
Future Development (Short-Term)

Make ModProfiler useful within the current scope:

- Test with a wide range of sites
  - Involve community
  - Refine and handle edge cases
- Create models for popular open source products
  - Some have pledged support
- Continuous learning
Future Development (Long-Term)

Extend scope of ModProfiler:

- Output modelling
- User profiling
- Session profiling
- Extend data coverage
  - JSON
  - XML
- Real-time operation
Questions?

Ivan Ristic, ivanr@breach.com
Ofer Shezaf, ofers@breach.com

Further information:
http://www.modsecurity.org/projects/modprofiler/